A Comparative Study of Relational and Non-Relational Database Models in a Web- Based Application

Cornelia Győrödi
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

Roxana Sotoc
Department of Computer Science and Information Technology, University of Oradea
Oradea, Romania

***Abstract*—The purpose of this paper is to present a comparative study between relational and non-relational database models in a web-based application, by executing various operations on both relational and on non-relational databases thus highlighting the results obtained during performance comparison tests. The study was based on the implementation of a web-based application for population records. For the non-relational database, we used MongoDB and for the relational database, we used MSSQL 2014. We will also present the advantages of using a non-relational database compared to a relational database integrated in a web-based application, which needs to manipulate a big amount of data.**

Keywords—MongoDB; MSSQL; NoSQL; non-relational database

I. Introduction

As technology nowadays is tireless and evolves more and more every day, the amount of data is increasing and an application to handle a huge volume of data efficiently it is important to choose the right model of the database. Relational database model has a quite rigid schema that 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 [5]. It is difficult to change the schema of databases, especially when, it is a partitioned relational database that spreads across multiple servers. If our data capture and management requirements are constantly evolving, a rigid schema can quickly become an obstacle to change [6].

Generally a web application must support millions of users simultaneously and to handle a huge volume of data a relational database model is still widely in use today, even though it has serious limitations when to handle a huge volume of data.

Google, Amazon, Facebook and LinkedIn have been among the first companies that discovered those limitations of the relational database model as far as the demands of new applications. These limitations have led to the development of non-relational databases, also commonly known as NoSQL (Not Only SQL) [7].

Non-relational databases do not use the RDBMS principles (Relational Data Base Management System) and do not store data in tables, and have schema-less approach to data management. Non-relational databases do not require schema definition before inserting data nor changing the schema when data collection and management need evolve [6] [10]. Instead, they use identification keys and data can be found based on the keys assigned [8].

NoSQL could be categorized in 4 types [2]:

1) Key-Value databases – which are the simplest NoSQL data stores to use, from an API perspective. The most popular ones are Redis, Riak, etc.

2) Document databases – which store and retrieve documents as XML, JSON, BSON and so on. The most popular document database is MongoDB, which provides a rich query language.

3) Column family stores – these databases store data in column families as rows that have many columns associated with a row key. One of the most popular is Cassandra.

4) Graph Databases – allows you to store entities and relationships between these entities. There are many graph databases, but between this type of database we can mention OrientDB, FlockDB. etc. [2]

According to the article was written by Matt Asay, "NoSQL databases eat into the relational database market" [1] the NoSQL databases, especially MongoDB, occupy more and more space on the market, but with all these Oracle and SQL Server are still constant. In Fig 1. we can see the popularity of MongoDB and its evolution from 2014 to 2015.

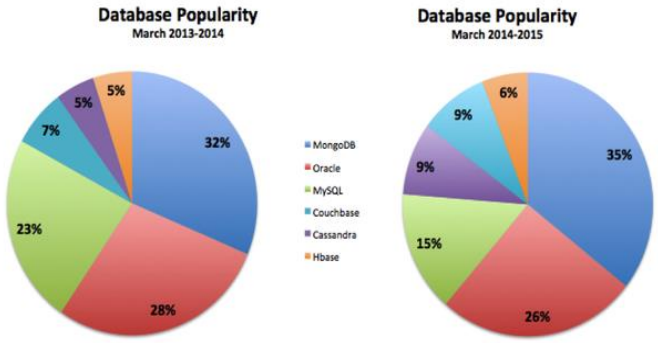

Fig. 1. Database Popularity [1]

In Fig. 1. we can see a growth of 3% for MongoDB from 2014 until 2015 and a decrease of 2% for Oracle and 8% for MySQL. This could mean that the companies are leaning toward NoSQL databases, so they can manipulate more data at a lower price [1].

A big advantage of non-relational databases is that they are more scalable and provide superior performance and their data model addresses several issues that the relational model is not designed to address like large volumes of structured, semi-structured and unstructured data, agile sprints, quick iteration, frequent code pushes, object-oriented programming, efficiency, monolithic architecture and so on [3] [9].

In this paper, we focus on one of the non-relational databases, namely MongoDB, and we make a comparison with one of the relational databases, namely MSSQL to highlight advantages and disadvantages of the two models. The study was based on the implementation of a web-based application for population records, which needs to manipulate a big amount of data.

II. Application Development Using MongoDB Vs. MSSQL

We created a comparative study between relational databases, namely MSSQL, and non-relational databases, namely MongoDB. The study has based on the implementation of a website for population records, which needs to manipulate a big amount of data. To highlight the advantages of using the non-relational database MongoDB compared to the relational database MSSQL, various operations were performed on the two databases. These operations are the four basic operations that can be performed on any database, namely: SELECT, INSERT, UPDATE and DELETE. These operations were made on 1, 100, 500, 1.000, 5.000, 10.000, 25.000 and 50.000 records. The application has been developed, in ASP.NET MVC 4 with C# programming language and we implemented the same methods for both MSSQL and MongoDB databases. For non-relational databases, we used the MongoDB C# Driver that is the officially supported C# driver for MongoDB. The version of the driver is 2.0 and the version of MongoDB is 3.0.

To be able to connect to MongoDB we used a MongoClient. In code, we used two namespaces that are specific to MongoDB: MongoDB.Driver and MongoDB.Bson. Finally we added the connection string in web.config file, like this:

```
<connectionStrings>
<add name = "MongoDB" connectionString = "
mongodb://localhost:27017/"/>
</connectionStrings>
```

We used this connection string through this code:

```
public static string connString =
System.Configuration.ConfigurationManager.Conn
ectionStrings["MongoDB"].ConnectionString;
```

After retrieving the correct connection string we used it in methods for calling the collections that we need like the one bellow:

```
public static IMongoCollection<BsonDocument>
ConnectToServer()
{
var client = new MongoClient(connString);
var db = client.GetDatabase("Dizertatie");
   IMongoCollection<BsonDocument> collection =
   db.GetCollection<BsonDocument>("People");
return collection;
}
```

In the method above, we call the database called “Dizertatie” and we get the collection called “People” and the results that will be returned will be a IMongoCollection of BsonDocument type which is a specific format for MongoDB.

Next, we created asynchronous methods to work with the data from MongoDB. For example, for deleting all the registered people from the collection we created a method like this:

```
public static async Task<DeleteResult>
DeleteAllConsumers()
{
       var collection = ConnectToServer();
       var filter = new BsonDocument("_id",
       new BsonDocument("$exists", true));

       var result = await
       collection.DeleteManyAsync(filter);
       return result;
}
```

The MongoDB method that we call for deleting the registered people, DeleteManyAsync, will delete multiple documents inside the collection that we are connected to. The number of the deleted documents depends on the filter that we need to create and provide when we want to call the method. In our case, we will delete all registers from this collection.

In the following section we will mainly focus on the performance results for both databases, MongoDB and MSSSQL, that we obtained after we executed operations of SELECT, INSERT, UPDATE and DELETE.

III. Comparative Study: MongoDB Vs MSSQL

Generally, depending on the scope of the application that we want to develop, when the project is in the planning stage, each company will establish the resources and the limits for the project. We also need to choose the database for the application that will be developed. Here, we should consider the amount of data that will be manipulated, the rapidity that the project needs and the budget.

Considering these factors, for an application that will need to store a large amount of data, if the application have to handle a huge volume of data, then we should think how we could achieve this in an efficient way or how many resources should be allocated for this scope. For example, in the application that we created, we took in consideration the fact that we need a huge space for storing the data and the queries that will be made every day like adding new data, deleting, updating and so on. All these queries are expensive and we should also, think about the rapidity at which they are processed. Another important fact that we need to consider is the number of users that will access the application, like the employees from all the country, plus the usual users that will need different information.

Considering all these facts, it is important to analyse the performance of the application in terms of insertion, update, deletion and selection operations.

The performance of the database that we have chosen can be a very important fact because of the storage space, all the hardware and other components that we need.

We began testing with the creation of databases both MongoDB and MSSQL after that we executed operations of INSERT, UPDATE, DELETE and SELECT on both databases. All these operations have been made on 1, 100, 500, 1.000, 5.000, 10.000, 25.000 and 50.000 records.

In MongoDB, for inserting a list of people into the database we write the following code:

```
public static async Task<string>
InsertPeople(List<BsonDocument> pplList)
{
   var collection = ConnectToServer();
   var result = await
collection.InsertManyAsync(pplList)
.ContinueWith(x => x.ToJson());
   return result;
}
```

And in MSSQL we write the following code:

```
using (SqlConnection sqlConn = new
SqlConnection(connStr))
{
 sqlConn.Open();
 using (SqlCommand sqlCmd = new SqlCommand())
{
sqlCmd.Connection = sqlConn;
sqlCmd.CommandType =
CommandType.StoredProcedure;
sqlCmd.CommandText =[dbo].[Set_people]";
sqlCmd.Parameters.AddWithValue("@peopleTbl",
data); sqlCmd.Parameters.Add("@isInsert",
SqlDbType.Bit).Value = isInsert;
int result = sqlCmd.ExecuteNonQuery();}
sqlConn.Close();}
}
```

In MongoDB the following method will insert one record:

```
public static async Task<string> AddUser(User
model) {
       var collection = ConnectToServer();
       var result = await
       collection.InsertOneAsync(model)
       .ContinueWith(x => x.ToJson());
       return result;
       }
```

After we executed these operations, we obtained the following results that are shown in Table 1.

The graphical representation of the results from Table 1 is shown in Figure 2. We notice that until 1.000 records, we obtained a maximum difference of 200 milliseconds, but we can see that the difference is more significant after 1.000 records and for 50.000 records we obtained a difference of 7 seconds and half.

TABLE I. THE RESULTS OF THE INSERT OPERATION

Insert	MongoDB - sec	SQL - sec
1 user	00:00:00:003	00:00:00:402
100 users	00:00:00:005	00:00:00:096
500 users	00:00:00:018	00:00:00:183
1.000 users	00:00:00:033	00:00:00:387
5.000 users	00:00:00:162	00:00:00:736
10.000 users	00:00:00:521	00:00:01:085
25.000 users	00:00:00:816	00:00:03:378
50.000 users	00:00:01:835	00:00:08:306

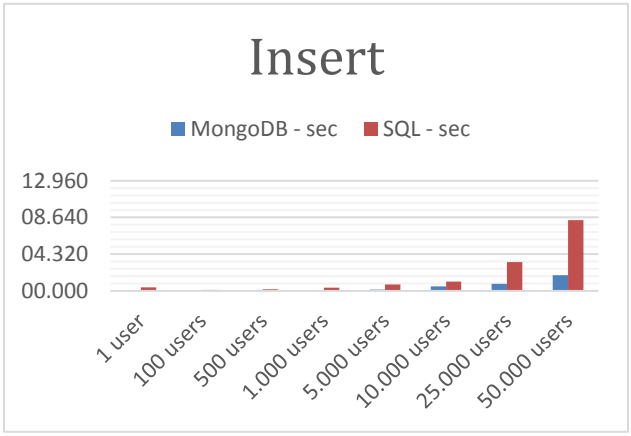

Fig. 2. MSSQL vs MongoDB insert

In MongoDB to select records we write the following code:

```
List<People> pplList = new List<People>();
var collection = ConnectToServer();
var filter = new BsonDocument("_id", new
BsonDocument("$exists", true));
using (var cursor = await
collection.FindAsync(filter)) {
while (await cursor.MoveNextAsync()){
var batch = cursor.Current;
foreach (var document in batch) {
People ppl = new People();
ppl.ID =
document["_id"].AsObjectId.ToString();
ppl.CNP = document["Cnp"].AsString;
ppl.Nume = document["Nume"].AsString;
ppl.Prenume = document["Prenume"].AsString;
ppl.Varsta =
document["Varsta"].AsInt32.ToString();
ppl.Sex = document["Sex"].AsString;
ppl.Adresa = document["Adresa"].AsString;
ppl.Ocupatie =
document["Ocupatie"].AsString;
ppl.AdresaOcupatie =
document["AdresaOcupatie"].AsString;
ppl.Telefon = document["Telefon"].AsString;
ppl.StareCivila =
document["StareCivila"].AsString;
ppl.OrasLocalitate =
```

```
document["OrasLocalitate"].AsString;
ppl.Judet = document["Judet"].AsString;
pplList.Add(ppl); } }}
```

In MSSQL to select records we write the following code:

```
DataTable dt = new DataTable();
try{
using (SqlConnection sqlConn = new
SqlConnection(connStr)) {
sqlConn.Open();
using (SqlCommand sqlCmd = new
SqlCommand()){
sqlCmd.Connection = sqlConn;
sqlCmd.CommandType =
CommandType.StoredProcedure;
sqlCmd.CommandText = "[dbo].[Get_People]";
SqlDataAdapter da = new
SqlDataAdapter(sqlCmd);
da.Fill(dt); }
sqlConn.Close();}}
catch (Exception) { }
return dt;
```

After we executed these operations for select we obtained the following results that are shown in Table 2:

TABLE II. THE RESULTS OF THE SELECT OPERATION

Select	MongoDB - sec	SQL - sec
1 user	00:00:00:003	00:00:00:083
100 users	00:00:00:004	00:00:00:002
500 users	00:00:00:017	00:00:00:005
1.000 users	00:00:00:031	00:00:00:006
5.000 users	00:00:00:206	00:00:00:028
10.000 users	00:00:00:291	00:00:00:052
25.000 users	00:00:00:830	00:00:00:190
50.000 users	00:00:01:616	00:00:00:327

The graphical representation of the results from Table 2 are shown in Figure 3.

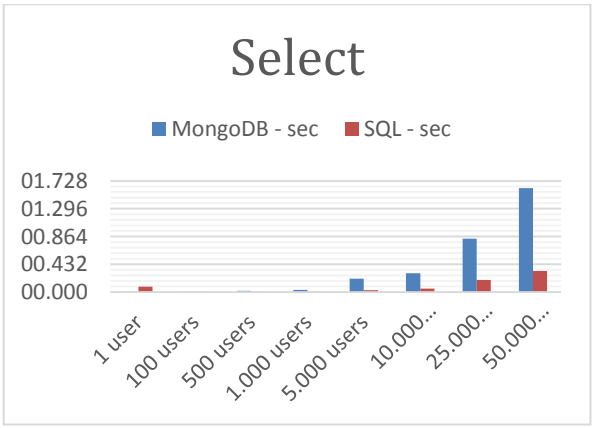

Fig. 3. MSSQL vs MongoDB select

From figure Fig.3 we notice that the select operation is more fast and efficient in MSSQL than in MongoDB. We also see that at 1.000 records, it is a difference 26 milliseconds and this grows up to 1.3 seconds when we select 50.000 records.

To update the records in MongoDB we write the following code:

```
   var collection = ConnectToServer();
   var filter = new BsonDocument("_id", new
BsonDocument("$exists", true));
   var update = Builders<BsonDocument>.Update
        .Set("Nume", "Updated")
        .Set("Adresa", "Updated");
   var result = await
collection.UpdateManyAsync(filter, update);
```

The methods that we created for MSSQL to insert a new record or a list of records will be used for UPDATE operation too, but the parameter *@isInsert* will be equal with false in that case. The results of the update operation are shown in Table 3.

TABLE III. THE RESULTS OF THE UPDATE OPERATION

Update	MongoDB - sec	SQL - sec
1 user	00:00:00:005	00:00:00:039
100 users	00:00:00:007	00:00:00:048
500 users	00:00:00:059	00:00:00:059
1.000 users	00:00:00:042	00:00:00:159
5.000 users	00:00:00:245	00:00:02:219
10.000 users	00:00:00:463	00:00:04:634
25.000 users	00:00:01:294	00:00:19:946
50.000 users	00:00:02:224	00:00:31:205

The graphical representation of the results from Table 3 is shown in Figure 4.

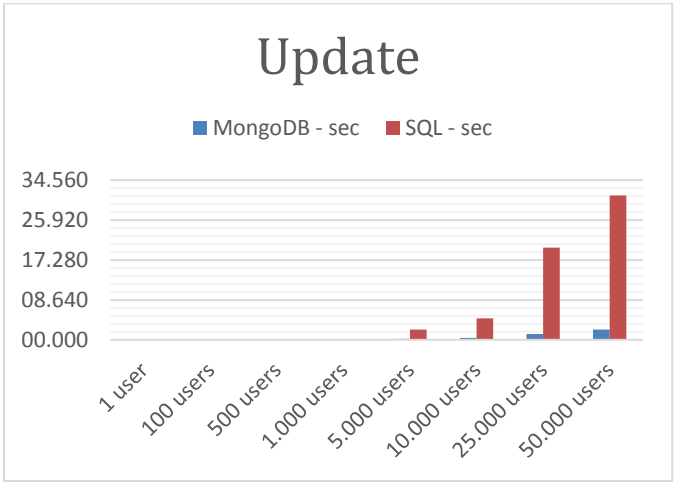

Fig. 4. MSSQL vs MongoDB update

For the update operations we can see a bigger difference from 5.000 records, which is approximatively 2 seconds and until 50.000 records it grows up to 29 seconds and gives MongoDB an advantage. We notice that MongoDB spends less time than MSSQL, for performing update operation as shown in figure Fig. 4.

To delete the records in MongoDB we write the following code:

```
var collection = ConnectToServer();
```

```
var filter = new BsonDocument("_id", new
BsonDocument("$exists", true));
var result = await
collection.DeleteManyAsync(filter);
return result;
```

And in MSSQL to delete records we write the following code:

```
using(SqlConnection sqlConn = new
SqlConnection(connStr)) {
sqlConn.Open();
using(SqlCommand sqlCmd = new SqlCommand()){
sqlCmd.Connection = sqlConn;
sqlCmd.CommandType =
CommandType.StoredProcedure;
sqlCmd.CommandText =
"[dbo].[Del_All_Consumers]";
int result = sqlCmd.ExecuteNonQuery();
success =
!string.IsNullOrEmpty(result.ToString()) ?
true : false; }
sqlConn.Close(); }
```

The results of the delete operation are shown in Table 4.

TABLE IV. THE RESULTS OF THE DELETE OPERATION

Delete	MongoDB - sec	SQL - sec
1 user	00:00:00:004	00:00:00:081
100 users	00:00:00:003	00:00:00:019
500 users	00:00:00:007	00:00:00:063
1.000 users	00:00:00:017	00:00:00:082
5.000 users	00:00:00:053	00:00:00:143
10.000 users	00:00:00:106	00:00:00:200
25.000 users	00:00:00:317	00:00:00:350
50.000 users	00:00:01:508	00:00:01:787

The graphical representation of the results from Table 4 is shown in Figure 5.

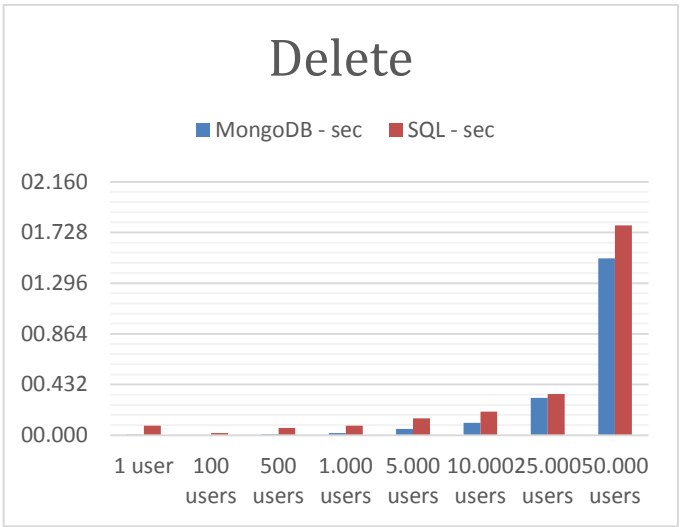

Fig. 5. MSSQL vs MongoDB delete

From figure Fig. 5 we notice that MongoDB provided lower execution times than MySQL in delete operations, especially when the number of records increases, which is essential when an application should provide support to thousands of users simultaneously.

IV. CONCLUSIONS

In this paper, we showed the results of different operations that had been applied for MongoDB and MSSQL databases. MongoDB provided lower execution times than MSSQL in INSERT, UPDATE and DELETE operations, which is essential when an application should provide support to thousands of users simultaneously. The only time when the MSSQL obtained an advantage was with the SELECT operations, the other ones gave advantages to MongoDB.

We can also notice that the difference between the results of each database was not noticeable until around 1.000 records. Thus, we can say that relational databases, namely MSSQL is suitable for small and medium applications. Relational databases are widely used in most of the applications and they have good performance when they handle a limited amount of data.

We need to be carefull when we want to choose a model of the database for the application that we will want to create. We should take into consideration main factors as the amount of data, the flexibility of schema, the budget, the amount of transactions that would be made and how frequent they are called. These days, companies, depending on the application that they want to develop, have the possibility to choose the most suitable database from a wide range of databases. Generally, for smaller and medium applications, a relational database would be chosen and for big applications, that use and manipulate large quantities of data, a non-relational database will be chosen. Of course, these are not the only criteria for choosing a database, but it depends on each company and the purpose of the application that would need to be developed.

Considering that, in our days, the request for storing a big volume of data at a low price is bigger every day, we tend to choose a non-relational database. Also, MSSQL being commercial at a pretty big price and MongoDB being an open source solution, it is a big disadvantage for MSSQL.

In the end, for choosing the correct database that can satisfy all the needs that an application demands in order to have been developed, all the things presented above should be taken into consideration before the start of development of the project. Depending on what each application needs, we can choose the most suitable database, a non-relational or a relational database.

REFERENCES

[1] Matt Asay, “NoSQL databases eat into the relational database market”, Published March 4, 2015, Available: http://www.techrepublic.com/article/nosql-databases-eat-into-the-relational-database-market/ , accessed july 2015.

[2] Pramod Sadalage,”NoSQL Databases: An Overview”, Published Octomber 1, 2014, Available: http://www.thoughtworks.com/insights/blog/nosql-databases-overview, accessed july 2015

[3] S. Hoberman, “Data Modeling for MongoDB**”**, Publisher by Technics Publications, LLC 2 Lindsley Road Basking Ridge, NJ 07920, USA, ISBN 978-1-935504-70-2, 2014.

[4] Michael Kennedy, ”MongoDB vs. SQL Server 2008 – Performance Shutdown”, Published April 29, 2010, Available: http://blog.michaelckennedy.net/2010/04/29/mongodb-vs-sql-server-2008-performance-showdown/, accessed july 2015

[5] 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.

[6] C. Bazar, C. Iosif, "The transition from RDBMS to NoSQL. A comparative analysis of three popular non-relational solutions: Cassandra, MongoDB and Couchbase", Database Systems Journal, vol. V no 2/2014, pp.49-59.

[7] 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.

[8] Cornelia Győrödi, Robert Győrödi, George Pecherle, Andrada Olah, "A comparative study: MongoDB vs. MySQL", IEEE - 13th International Conference on Engineering of Modern Electric Systems (EMES), 2015, Oradea, Romania, 11-12 June 2015, ISBN 978-1-4799-7649-2, pag. 1-6.

[9] MongoDB, NoSQL Database Explained, Available: https://www.mongodb.com/nosql-explained , accessed july 2015.

[10] R. P Padhy, M. R. Patra, S. C. Satapathy, "RDBMS to NoSQL: Reviewing Some Next-Generation Non-Relational Database's", International Journal of Advance Engineering Sciences and Technologies, Vol. 11, Issue No. 1, 015-030, 2011.