International Journal of Computer Science and Applications,
©Technomathematics Research Foundation
Vol.15, No. 1, pp. 65 – 82, 2018

A FRAMEWORK FOR MAPPING THE MYSQL DATABASES TO MONGODB – ALGORITHM, IMPLEMENTATION AND EXPERIMENTS

LIANA STANESCU

Computer Science and Information Technology, University of Craiova, Bvd. Decebal 106, Craiova, 200440, Romania
stanescu@software.ucv.ro

MARIUS BREZOVAN

Computer Science and Information Technology, University of Craiova, Bvd. Decebal 106, Craiova, 200440, Romania
mbrezovan@software.ucv.ro

COSMIN STOICA SPAHIU

Computer Science and Information Technology, University of Craiova, Bvd. Decebal 106, Craiova, 200440, Romania
stoica.cosmin@software.ucv.ro

DUMITRU DAN BURDESCU

Computer Science and Information Technology, University of Craiova, Bvd. Decebal 106, Craiova, 200440, Romania
dburdescu@yahoo.com

The paper presents an automated solution for migration of a SQL database to a NoSQL database, namely MongoDB. Relational databases began to be used less and less in the last years, especially by companies working with very large volume of information. Relational databases are not seen anymore as the best solution available for storing the information. Instead, the NoSQL solutions started to evolve and they are currently seen as a possible replacement for relational servers, due to better performances in data processing and retrieval. The problem which tries to be solved in this paper is how can be automatically transfered the existing information from a relational database to a MongoDB database, with no information loss. The framework implements an original algorithm that uses the metadata stored in the relational system tables. It takes into consideration the concepts from Entity-Relationship (ER) model: entity type represented by a relation in the Relational Model (RM), 1:1 and 1:M relationship type represented with Foreign Keys (FK) in the RM and N:M relationship type represented in RM with a join table that contains the Primary Keys (PK) from the original tables, each representing a FK and two 1:M relationships between the original tables and the join table. The paper also made a comparative study between the performances obtained in uptading and retrieving data in the old and new format.

Keywords: Relational databases; MySQL; document databases; MongoDb.

1. Introduction

NoSQL databases have evolved intensly in the las years due to their flexible structure, less constrained than relational ones and offering faster access to information. MongoDB is an example of a document-oriented NoSQL database which stores data in the form of JSON-like objects. It has emerged in the last years as one of the leading databases servers due to the possibility to define dynamic schema, it has a high scalability, optimal query performance, and an active user community which helped him evolve [Krisciunas (2014)], [Celko (2014)], [Sadalage and Fowler (2013)].

The problem which tries to be solved in this paper is how can be automatically transfered the existing information from a relational database to a MongoDB database, with no information loss. MongoDB is a cross-platform, document oriented database that provides high performance, high availability and easy scalability. The main concepts in MongoDB are collection and document. Database is a physical container for collections. Each database receives its own set of files on the file system. A single MongoDB server typically manages multiple databases [Kvalheim (2015)], [Trivedi (2014)], [Membrey *at al.* (2014)].

The collection is a group of MongoDB documents. It has as correspondent a RDBMS table. A collection exists only within a single database.

A MongoDB document is a set of key-value pairs. The documents do not have to necessarily respect a schema. Typically, all documents in a collection are of similar or have a related purpose. Dynamic schema means that documents in the same collection do not need to have the same set of fields or structure, and common fields in a collection's documents may hold different types of data.

The paper presents the technical details of the software implementation. The application takes as input the relational database and offers as output the non-relational database.

This new tool can be very useful for companies where the need of an isolated stand-alone application evolved to enterprise applications, and where relational databases are not fitting anymore.

Although no schema or database documentation is needed for small applications, data modeling become critical for more complex structures. This is because it is not enough anymore just to store information, but businesses needs to have the posibilty to take decisions based on data understanding (data mining).

The decision to make a switch in the way data is stored is hard to take, and involve a series of risks. One of them is the possible loss of information during data migration and conversion. This risk is the current work tries to mitigate by proposing an automated tool.

The migration starts with a set of standard steps that needs to be performed. They will be performed for each table in part [Stanescu *et al*. (2016)][Stanescu *et al*.(2017)].

The implementation was done in C# programming language using C# WinForm - .NET Framework, MySQL SGBD and MongoBD NoSQL output database. The system generates both the MongoDB conversion and data mapping to the new format. This type

of application is used for companies who want to migrate to another database provider, or to change the way information is stored.

The users have some important additional features offered by the system in order to decide which format is better for a particular usage. These features give the possibility to generate performance reports by comparing the two databases (the original one – SQL and the converted database – No SQL DB).

When a decision was taken to swith to a new database, the new server needs to be tested and validated. Specific features are need in this case for data analysis. A failure in this domain is a high risk both for owner company and their customers. This is why data handling- performance is a very sensitive aspect: most of the applications need to be able to execute queries in order to retrieve information and needs to store the results of the analysis. This is done using different databases servers. It can be concluded, without exagerating that the databases are one of the key parts of a company's work.

The paper is organized as follows: Section 2 presents related work for NoSQL databases and database migration, Section 3 presents the system's architecture and the algorithm used, Section 4 presents a comparative study between performances of a SQL and NoSQL database and Section 5 presents the conclusions of the paper.

2. Related Work

In order to reach the objectives of the business, large volume of information needs to be analised and manipulated, this generating more and more information each day. The companies need to hadle nowadays huge volume of information. Companies like Amazon, eBay, Target, Sears are ones of the most affected by this problem and most of the have their own proprietary solutions implemented.

The solution provided using Cloud Computing offers a flexible and cost-efficient platform that can be used with success in this domain. A new concept is introduced for this, in adition to the well-known ones (Infrastructure as a Servise (IaaS), Software as a Service (SaaS) and Platform as service): Database as Service (DaaS).

The new cloud approach has a huge advantage for the databases users because ofers enhanced fiability and availability for a much bigger group of users. It is obtained a better horizontal and vertical scalability and the possibility to work with huge quantity of information distributed on many servers.

This NoSQL approach is highly scalable, it offers a high disponibility, error tolerance, a heterogeneus environment, and it ensures data consistency, integrity, security and portability. The data infrastructure is fully flexible to handle all types of requests. They put accent more on OLAP (Online Analytical Process) for data extraction, analysis and taking decisions. These are all part of the business intelligence.

In [Trivedi (2014)] there are proposed some guidelines to convert a relational database to MongoDb. The author focuses on designing basic relationships in MongoDB and map simple SQL queries to MongoDB.

The current paper presents an automated translation tool that can handle complex queries including aggregation.

The architecture, implementation and results obtained are presented in the next chapter.

3. System Architecture

The tool was developed in C# and the design was split on functionalities, as presented in Fig.1. Each of the functionalities was implemented in separate file and structured hierarchically. This was needed in order to ensure trasability and an easy extension in the future for new features.

The structure includes 3 folders. They contain several files, each of them implementing different functionalities:

- UserInterfaces: contains the interfaces files (GetDatabases.cs, GetDatabaseDetails.cs, Test.cs, MigrationOptionForm.cs)
- Migration: contains the migration algorithm (DatabaseDetailsInformation.cs, MigrateEmbededType.cs, MigrateRelationalType.cs)
- DatabaseModels: contains the files that stores the relational database schema information (Column.cs, Table.cs, TableConstraints.cs)

Three libraries which were used to communicate with MongoDB, are: MongoDB.Bson, MongoDB.Driver and MongoDB.Driver.Core.

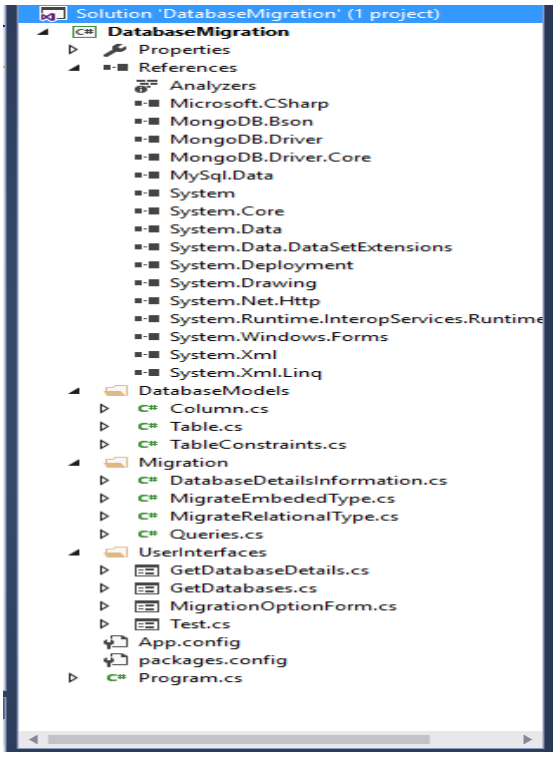

Fig. 1. Project structure

3.1 Migration algorithm description

In order to implement the automated relational databases migration to MongoDB server, there are needed initially the metadata information from relational server. This is extracted from INFORMATION_SCHEMA table [DuBois (2013)], [DuBois (2014)].

The obtained information will be mapped to generic and dinamic objects needed during migration process. The conceptual process is described in Fig.2.

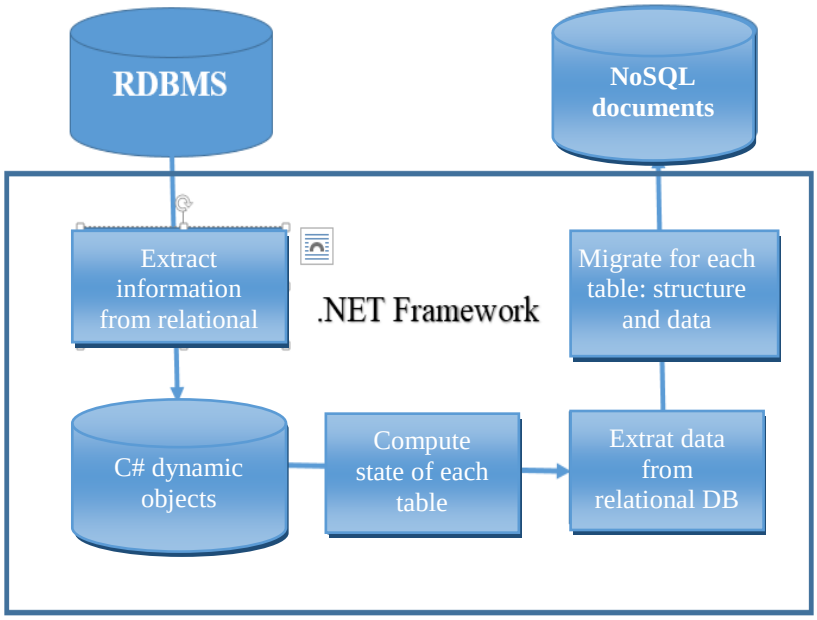

Fig. 2. Data migration flow

The following steps can be deducted from the diagram:

1. Extract information from relational database schema.
   The application connects to the relational database and executes system queries in order to read the tables' metadata which needs to be mapped to C# objects (Table, TableColumns, TableConstraints):
   a) Information about tables names:
      "select table_name from information_schema.tables where table_schema=databaseName"
   b) Information about attributes in tables:
      "select COLUMN_NAME, COLUMN_TYPE, COLUMN_KEY from information_schema.COLUMNS where TABLE_SCHEMA = database_name and table_name = table_name"
   c) Information about constraints created on tables
      "select COLUMN_NAME, CONSTRAINT_NAME, REFERENCED_TABLE_NAME, REFERENCED_COLUMN_NAME

from information_schema.COLUMNS where TABLE_SCHEMA = database_name and table_name = table_name"

2. Mapping relational schema information to C# models
   The information obtained above will be mapped to data types defined in DatabaseModels:
   a) Table data are mapped to data type “Table”
   b) Columns data from each table are mapped to data type “Column”
   c) Constraints data for each table are mapped to data type “TableConstraints”

3. Analyse the relationships between tables [Stanescu *et al*. (2016)], [Stanescu *et al*. (2017)]
   a) If no foreign key exists for a table and it is not used in other tables, then it will be represented by a new MongoDB collection
   b) If no foreign keys are defined, but the table is used in other table, then it will be represented by a new MongoDB collection
   c) If a foreign key exists for a table and it is used in other tables, then it will be represented by a new MongoDB collection. For this type of tables the relationship method is used. The same concept for foreign keys will be used as in relational databases.
   d) If a foreign key is defined, but is not used in another table, the algorithm uses a one-way encapsulation model. The table is included in the coletion that represent the “1” part of the relation.
   e) If two foreign keys are defined, but is not used in another table, the algorithm use a two-way encapsulation model.
   f) If three or more foreign keys are defined (it is the result of a m:n relation) the algorithm uses another encapsulation model: a conectiong model with foreign keys which refer all original tables involved in relation and already represented as MongoDb colections. The solution is working even if the table will be used or not in other tables

4. Data extraction from each table in relational model.
   For each table in relational model it will be executed an SQL query during migration, in order to extract all data. The data will be inserted in the MongoDB collections, already created. The BsonDocument, BsonArray and BsonElement classes will be used for this. The queries are generated combining the fixed part “select * from” with table name information already store in the C# objects. For each table, it will be created a BSON document where data will be stored. The algorithm will iterate for each non-foreign key column and will generate a new BSON element containing column name and value. This element will be added in the document created before.

The result of the data migration schema should look as presented in Fig.3.

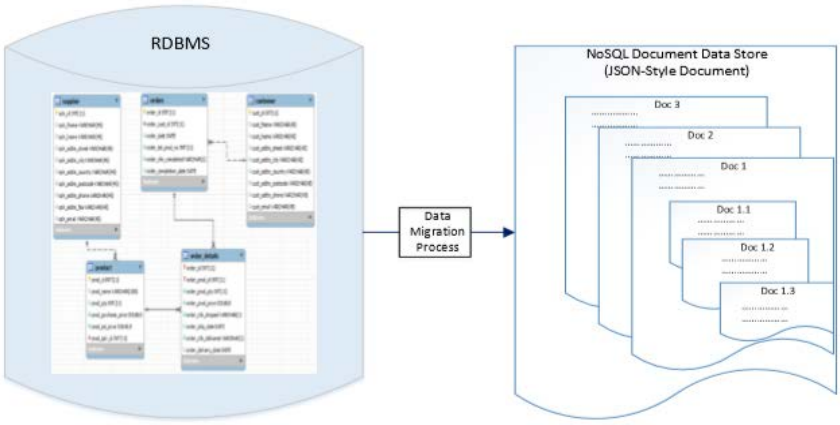

Fig. 3. RDBMS vs. NoSQL Document

3.2 Algorithm implementation in C#

The C# implementation follows the steps presented in 3.1.
There are used 3 classes:

Class Queries

There are 4 methods implemented in this class, all of them related to “Select” operations from tables.

a) GetTablesQuery returs the Select command which extracts informations about tables available in database:

```
publicstaticstring GetTablesQuery() {
        return"select table_name from information_schema.tables where
        table_schema = '" + QueryParameters.DatabaseNameParameter + "';";
}
```

b) GetColumnsInTableQuery returns Select command that extract metadata about available attributes in a table:

```
publicstaticstring GetColumnsInTableQuery() {
        return"select COLUMN_NAME, DATA_TYPE, COLUMN_TYPE,
        COLUMN_KEY from information_schema.COLUMNS where
        TABLE_SCHEMA = '" +
        QueryParameters.DatabaseNameParameter + "' and TABLE_NAME =
        '" + QueryParameters.TableNameParameter + "';";
}
```

c) GetTableConstraintsQuery returns the Select command that extract metadata about defined constraints in a atable:

```
publicstaticstring GetTableConstraintsQuery()  {
```

```
return"select COLUMN_NAME, CONSTRAINT_NAME,
REFERENCED_TABLE_NAME,
REFERENCED_COLUMN_NAME from " +
"INFORMATION_SCHEMA.KEY_COLUMN_USAGE where
TABLE_SCHEMA = '" + QueryParameters.DatabaseNameParameter +
"' and TABLE_NAME = '" + QueryParameters.TableNameParameter + "';";
}
```

d) GetAllTableDataQuery returns the Select command that extract all records from a specified table:

```
publicstaticstring GetAllTableDataQuery(){
 return"select * from " + QueryParameters.TableNameParameter;
  }
```

The methods presented above are related to the first step of the algorithm. The information extracted is stored in the system in order to be used during the next steps. The following data types are needed:

- Column – stores the columns metadata for each table, as a set of 3 values: ColumnName, ColumnType, ColumnKey (stores information about primary and foreign keys)
- TableContraints – stores information about the constraints in each table, as a set of 4 values: ColumnName, ConstraintName, ReferencedTable, ReferencedColumn
- Table – the most complex data type. It includes sub-types for storing information related to table's metadata, list of attributes, list of constraints, list of child tables, and methods to find table's relations:
  - TableName

```
publicstring TableName{ get; set; }
```

  - PrimaryKeyColumnName

```
publicstring PrimaryKeyColumnName{
 get { return TableConstraints.Where(y => y.ConstraintName ==
"PRIMARY").Select(x => x.ColumnName).FirstOrDefault();
        }
}
```

- The list of attributes and constrains is represented as „lazy" lists:

```
publicList<Column> Columns
      {
get { return _tableColumns.Value; }
set { _tableColumns = newLazy<List<Column>>(() =>value); }
      }
publicList<TableConstraints> TableConstraints
      {
get { return _tableConstraints.Value; }
```

```
set { _tableConstraints = newLazy<List<TableConstraints>>(()
=>value); }
    }
```

Because this type is very complex and it can store high volume of information, it was used the "Lazy loading" paradigm for implementation. This design allows the compiler to delay the initialization of an object until it is used. This will provide an increase of system performances when working with huge volume of data.

Class GetDatabaseDetails

This class contains the methods executed in the first phase of the migration algorithm: obtaining all needed information from relational database and mapping to C# models.

The evolution of the execution can be monitored by the user in a visual interface. The migration algorithm is executed on a separate thread using class BackgroundWorker. This class offeres 3 types of events which can be customized as neeed:

- ProgressChanged – the event which updates the visual interface with user. It can display the following information:
  lblItem.Text = "Load tables information";
  lblItem.Text = "Load columns information";
  lblItem.Text = "Load constraints information";
  lblItem.Text = "Create mongo collections";
- RunWorkerCompleted – the event is triggered when all operations executed by "DoWork" are completed. It will also update the interface information with message "Migration completed";
- DoWork – the event which handles all time-consumming operations. It will execute methods for LoadTables, LoadColumnsForEachTable.

a) LoadTables method – extract tables data and maps it to the Table data type. The information is obtained from class Queries, GetTablesQuery():

```
var tablesQuery = Queries.SchemaQueries.GetTablesQuery();
```

b) LoadColumnsForEachTable method – extract metadata from attributes for each table and maps them in data type Column. For obtaining the data it is executed the query from class Queries, GetColumnsInTableQuery:

```
var columnsInTableQuery =
    Queries.SchemaQueries.GetColumnsInTableQuery();
```

c) MapColumnsForTable – this method is executed for each table. The prformed steps are: open database conection, execute the query and extract the information, map the obtained information to C# objects, close the database conection.

d) LoadConstraintsForEachTable method – extracts the metadata from constraints in all tables and maps them to data type TableConstraints. The following data is obtained from class Queries, GetTableConstraintsQuery().

```
var tableConstraints = Queries.SchemaQueries.GetTableConstraintsQuery()
```

For each table in list will be executed "MapTableConstraints" to obtain all needed information. The steps performed are similar as above: open database conection, execute

the query and extract the information, map the obtained information to C# objects, close the database conection.

After this step, there were completed the first 2 steps of the algorithm. The data was extracted and mapped to C# objects.

The next steps will start the migration and will be executed by methods in class MigrateEmbededType.

Class MigrateEmbededType

This class implements the most complex part of the algorithm. It is computed here the state of each table, create the mongoDB objects and add all information from tables in these objects. It contains the following methods:

e) Migrate method: use MigrateEmbededType to provide needed information to other classes. For each table it will be executed sequentially the methods CalculateTableChilds, InsertTableData, and InsertTableDataForChilds.

The operation on MongoDB server will be executed using object MongoDatabase of data type IMongoDatabase. This is obtained by calling method GetDatabase from MongoDB .NET: it is called the object IMongoClient _client. These 2 objects store the connection with server. The database is not needed to be explicitely created because it is automatically done when method GetDatabase is executed: _client.GetDatabase(Queries.DatabaseName);

b) CalculateTableChilds method – a recursive method that considers each table as a parent. It will be created a parent-child hierarchy between tables and data. The list of children will be obtained by executing a LINQ expression (similar as an SQL expression but it can be executed in C# to sort or filter data). If a parent has one or several children then the method performs a recursive depth processing in order to create the full hierarchy.

c) InsertTableData method – considers all tables not involved in M:M relations and creates a collection with the same name as the original table.

d) GetChildrenEmbededData - After the collection is created there will be obtained the data from the original table. The result will be a BsonDocument registered in the newly created colection.

e) ExecuteCommand_GetTableData_Embeded – returns an object of type MySqlDataReader containing all records from a table.

Data can be extracted using two methods: extract all data from tables using the query returned by class Queries using method “GetAllTableDataQuery”. The method extracts data from table where the primary key has a specific value. This value is received as parameter. If it has a value different than null, it will be extracted data filtered by primary key. It the value is null, all data in table will be extracted.

f) GetChildrenEmbededData method – is one of the most recursive methods in the system. It has several responsibilities: extract data from tables, migrate tables to MongoDBDocument, data migration, migrate the children’s data for each table. The

type of the method is BsonDocument. After the recursivity cycle is completed will return a MongoDB document which will also integrate other documents – if needed.

g) ExecuteCommand_GetTableData_Embeded method – extract the information from a BsonDocument document. Initially there are mapped and saved the recorded data in attributes which are not foreign keys. Each pair key-value will be composed by attribute/column name and the stored value. Bothe information is stored in the object BsonElement.

After all attributes which are not foreign keys were mapped, it is processed the list with attributes which are foreign keys. The method GetChildrenEmbededData will be used for this task, because the foreign keys represent 1:M relations and the execution will be recursive. In order to avoid infinite loops, an exit criterion must be defined. This criterion is the moment when the table doesn't contain any additional foreign key. After this step the migration is considered complete.

4. Experiments and Results

It will be analised in this chapter the performance differences between a MySQL database and the result obtained after migration to MongoDB. It was chosen for this a test database from a Movie providing company.

The relational database contains the following tables (figure 4):

a) Playingmovies – represent the schedule of the movies to be played: days, movies, rooms, prices
b) Bookings – the bookings made for the movies stored in "Playingmovies". It is considered a 1-M relation between Playingmovies and Bookings tables.
c) Movies – stores the information about movies (name, date, genre)
d) CinemaHall – stores the information for each available room (e.g. no of seats)

The migration to MongoDB will start by processing table "Playingmovies" because it has a 1-M relations with the rest of the tables. The data from "Bookings", "Movies" and "CinemaHall" will be already integrated in each record of "PlayingMovies" collection, in MongoDB.

The same query for obtaining this information is more complex in MySQL because there are needed several jonctions between all 4 tables.

"Playingmovies" table will be represented as a collection of JSON documents, and the data is stored also as a JSON document. Each object or record from table "Playingmovies" is composed from subsets of objects. These objects represent another individual JSON document, integrated in "Playingmovies". These documentes are records from tables "Movies", "CinemaHall" and "Bookings". The structure of the tables in MongoDB is presented in the figure 5.

Fig. 4. Relational Database – example

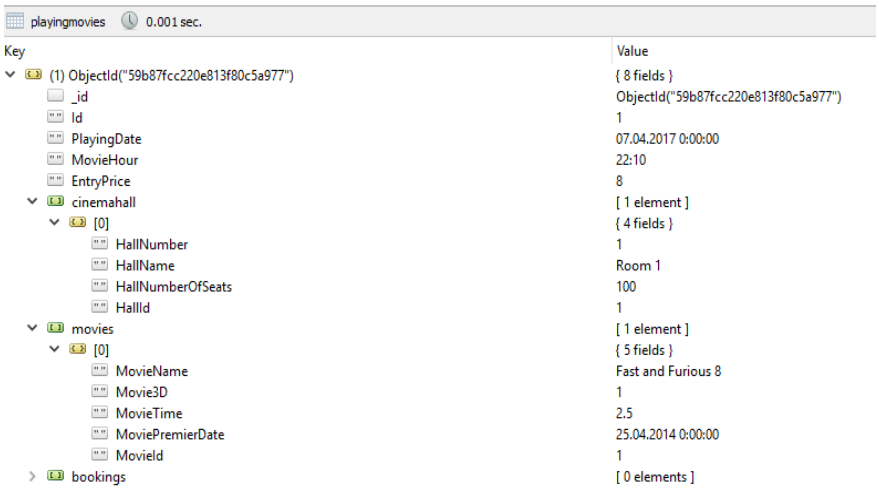

Fig. 5. MongoDB structure after migration

In order to test the performance and differences between the 2 databases, the tool provides a visual interface where it can be executed SELECT, INSERT, UPDATE, and DELETE operations.

4.1 INSERT, UPDATE, DELETE operations

The performance analysis compares the time needed both in MySQL and MongoDB to execute INSERT, UPDATE and DELETE operations. It is recorded the time needed to execute the operations on a database containing variable numbers of records:

- Initial there are compared the results in a small database containing 100 records.
- Secondly it is compared the time in a database containing up to 50000 records.

The differences can be seen in tables 1 and 2, and in figures 6 and 7.

For a bigger number of records, can be oserved an important difference up to tens of seconds. This is best visible in the insert operation.

Table 1 – Insert operation in MySQL vs MongoDB (100 records)

Număr înregistrări	INSERT		Update		DELETE	
	MySQL	MongoDB	MySQL	MongoDB	MySQL	MongoDB
10	626	82	361	68	648	94
20	1219	121	705	105	1490	121
30	1780	147	1162	134	2135	143
40	2250	174	1532	165	2793	167
50	2718	204	1851	195	3581	196
60	3279	228	2207	227	4159	222
70	3914	250	2541	260	4768	246
80	4373	283	2882	289	5248	270
90	4816	313	3213	318	5795	294
100	5383	343	3549	346	6258	324

Table 2 - Insert operation in MySQL vs MongoDB (50000 records)

No of records	MySQL (miliseconds)	MongoDB (miliseconds)
500	485	122
1000	968	232
2000	1920	510
5000	4915	1173
10000	11866	1976
20000	22502	4749
50000	59325	10984

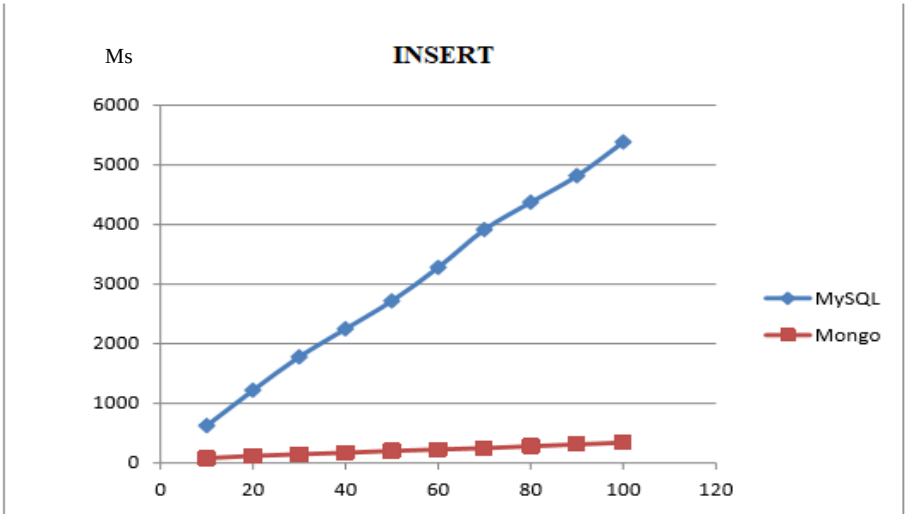

Fig. 6. No of records vs. time for INSERT operation

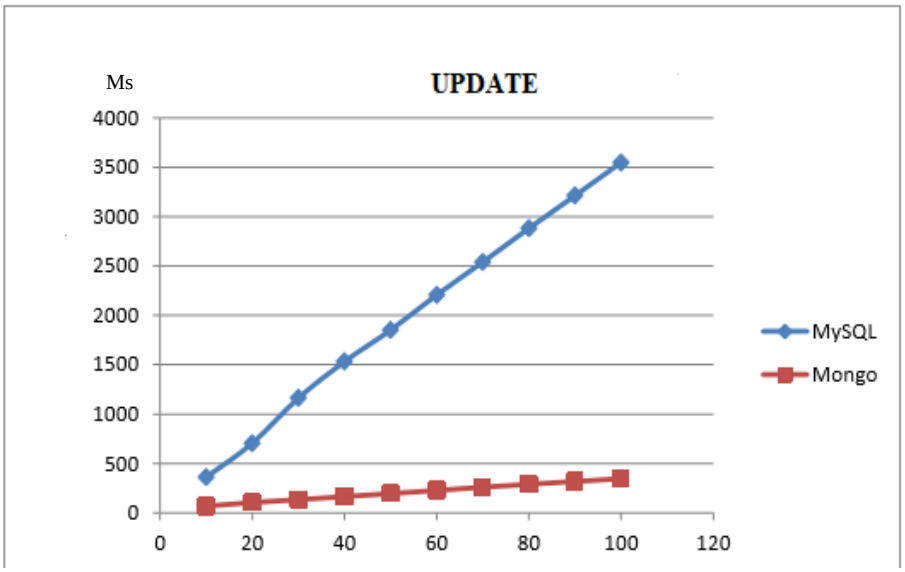

Fig. 7. No of records vs. time for UPDATE operation

4.2 SELECT operations

The next tests measured the time needed to retrieve the information, both in MySQL and MongoDB.

The SELECT command is executed for SQL database and "find()" method is used for MongoDB to retrieve information in collections. The queries in MongoDB have the scope of a single collection. Queries can return all documents in a collection or only the documents that match a specified filter or criteria. You can specify the filter or criteria in a document and pass as a parameter to the find() method.

The operations are made on a variable set of data, the number of records being up to 10000. It is recorded the time needed to execute each of the queries, and it is kept the average obtained after 10 executions (both for MySQL and MongoDB).

Table 3 - SELECT operation in MySQL vs MongoDB (10000 records)

No of records	MySQL (miliseconds)	MongoDB (miliseconds)
1000	3	28
2000	3	28
3000	3	30
4000	3	25
5000	3	27
6000	3	26
7000	3	28
8000	3	26
9000	3	28
10000	3	28

The results of the tests presented in table 3 show a better performance of MySQL database compared to MongoDB. The performances are constant and they do not semnificatively vary with the number of records.

4.3 SELECT operation using ordering option

The next set of tests is similar as previous one, but the retrieved information must be obtaind ordered. The same set of data is used as for previous tests, and the results are presented in Table 4 and Figure 8.

Tabel 4 - SELECT operation in MySQL vs MongoDB using ordering (10000 records)

No of records	MySQL (miliseconds)	MongoDB (miliseconds)
1000	3	89
2000	4	91
3000	3	94
4000	3	86
5000	84	87
6000	107	91
7000	121	91
8000	127	90
9000	140	89
10000	293	93

In this case can be oserved a better performance of the MongoDB database compared to the MySQL one.

The time needed to extract the information is almost linear for MongoDB database, depending on the number of records. However, the MySQL has a better performace for lower number of records but it is worsening when the number of records is increased.

Fig. 8. No of records vs. time for SELECT and ORDER BY operation

4.4 Data aggegation

Data aggregation is the process where informations are extracted and presented in a summarized form. This type of information is needed in statistical analysis. This set of tests includes a comparative analysis of the performances for data aggregation in MySQL (SELECT operation using Group By) and MongoDB (method Aggregate).

The results can be seen in table 5 an figure 9.

Table 5 - Aggregate operation in MySQL vs MongoDB (10000 records)

No of records	MySQL (miliseconds)	MongoDB (miliseconds)
1000	7	2
2000	11	2
3000	13	2
4000	16	2
5000	37	2
6000	45	2
7000	52	2
8000	55	2
9000	62	2
10000	70	3

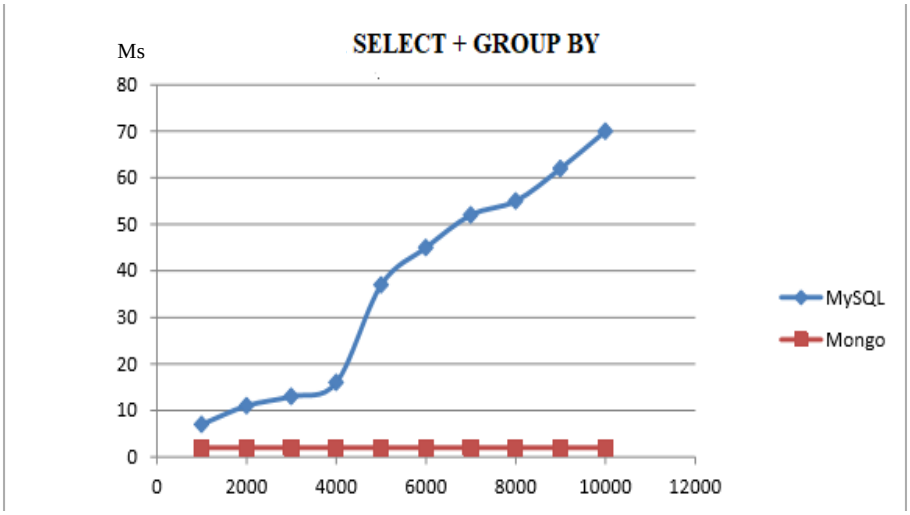

Fig. 9. No of records vs. time for SELECT and GROUP BY operation

Table 5 and Figure 9 show a much better performance of the MongoDB database, compared to MySQL one. According to the above graphic, the time is almost constant for the NoSQL database and does not vary with the number of records. The MySQL performance is dependent of the number of records and it is significantly dropping while the numbers of records is increasing.

5. Conclusion and future work

The scope of this research was to implement and analyse an automated system for migrating relational databases to a NonSQL database, namely MongoDB. The performed tests had the scope to analyse the differences in performaces of the two servers, for different working conditions (inserting, updating, deleting and retrieving data).

As it could be observed in the results presented in chapter 4, MongoDB has an advantage in performances over relational databases. Although the obtained results were good for the relational server up to approx. 10000 records (diference is at miliseconds level), they significally dropped above this number of records. The performances are also much better for MongoDB when the complexity of the queries is increased.

Several improuvements can be added in the future for the system. The first and most important is the support for other relationl servers (e.g. Microsoft SQL Server). A second improuvement that can be done is related to migration. More options for customizing the results would be needed (e.g. add the possibility to change tables name, or attributes names).

References

Celko, J. (2014). Joe Celko's Complete Guide to NoSQL: What Every SQL Professional Needs to Know about Non-Relational Databases, 1st Edition. Elsevier, USA

DuBois, P. (2013). MySQL (5th Edition) (Developer's Library) 5th Edition. Addison Wesley, New Jersey

DuBois, P. (2014). MySQL Cookbook: Solutions for Database Developers and Administrators. 3rd Edition. O'Reilly Media, USA

Krisciunas, A. (2014). Benefits of NoSQL: https://www.devbridge.com/articles/benefits-of-nosql/

Kvalheim, C. (2015). The Little Mongo DB Schema Design Book. The Blue Print Series

Membrey, P., Hows. D., Plugge, E. (2014). MongoDB Basics. Apress

Sadalage, P.J.; Fowler, M. (2013). NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence, 1st Edition, Addison Wesley, USA

Stanescu, L.; Brezovan, M.; Burdescu, D.D. (2016). Automatic Mapping of MySQL Databases to NoSQL MongoDB. FedCSIS 2016, pp.837-840

Stanescu, L.; Brezovan, M.; Burdescu, D.D. (2017). An Algorithm for Mapping the Relational Databases to Mongodb – A Case Study, International Journal of Computer Science and Applications, Technomathematics Research Foundation Vol. 14, No. 1, pp. 65 – 79

Trivedi, A. (2014). Mapping Relational Databases and SQL to MongoDB: http://code.tutsplus.com/articles/mapping-relational-databases-and-sql-to-mongodb--net-35650