DOI: 10.15439/2016F45
ACSIS, Vol. 8. ISSN 2300-5963

Automatic Mapping of MySQL Databases to NoSQL MongoDB

Liana Stanescu
University of Craiova Faculty of Automation, Computers and Electronics Bvd. Decebal 106 Craiova, Romania
Email: stanescu@software.ucv.ro

Marius Brezovan
University of Craiova Faculty of Automation, Computers and Electronics Bvd. Decebal 106 Craiova, Romania
Email: mbrezovan@software.ucv.ro

Dumitru Dan Burdescu
University of Craiova Faculty of Automation, Computers and Electronics Bvd. Decebal 106 Craiova, Romania
Email: dburdescu@yahoo.com

Abstract—**The paper presents a framework that implements our original algorithm of automatic mapping a MySQL relational database to a MongoDB NoSQL database. The algorithm uses the metadata stored in the MySQL 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 initial results of our algorithm that was tested on small size databases (10-15 tables with many relationships and 100 records/ table) are presented in this paper.**

I. Introduction

RELATIONAL Database Management Systems (RDBMS) have became the first choice for the storage of information in databases mostly used for financial records, manufacturing information, staff and salary data, and so on starting with 1980. RDBMSs are based on the relational model defined by a schema. This model uses two concepts: table and relationship. A relational table represents a well defined collection of rows and columns and the relationship is established between the rows of the tables. Relational data can be queried and manipulated using SQL query language [8].

In practice, there are situations in which storing data in the form of a table is inconvenient, or there are other kinds of relationships between records, or there is the necessity to quickly access the data. In order to solve such problems a new type of NoSQL has been created. A NoSQL or Not Only SQL database provides a mechanism for storage and retrieval of data that is different from the typical relational model.

Another issue that NoSQL solves is the mismatch between relational databases and object-oriented programming. It is known that SQL queries are not well suited for the object oriented data structures that are used in most applications now [8].

Another closely related issue is storing or retrieving an object along with all relevant data. Some operations require multiple and complex queries. In this case, data mapping and query generation complexity rise too much and becomes difficult to be maintained on the application side [8].

Some of these problems have found their answer both in Object-relational mapping (ORM) frameworks, even though it still requires a lot of development effort and also in Object-Oriented Database Management Systems (OODBMS). The down side in this last alternative is the fact that it did not gain much popularity in replacing relational databases. However, most object oriented databases may be considered NoSQL solutions as well [8].

Another problem that relational databases cannot handle is related to an exponentially increasing amount of data. The direct consequence is the so-called big data problem. This problem arises when standard SQL query operations do not have acceptable performances, especially when transactions are involved [8].

As a result, the subject of developing an automatic mapping instrument has been brought up. This instrument will be able to represent the existing relational databases, already populated with a large number of records, as NoSQL databases. A significant amount of time and human effort is spared this way, which would have otherwise been needed to create and populate the NoSQL database from scratch.

This paper proposes a framework which implements an algorithm for automatic mapping of MySQL relational databases to MongoDB.

The paper is organized in the following way: Section II presents the main concepts used by MongoDB, one of the most used NoSQL database, Section III presents general principles of mapping relational databases to MongoDB considering the main concepts from ER model and RM. Section IV presents in detail our proposed algorithm for automatic mapping of a MySQL database to MongoDB and also an example of application of this algorithm. Conclusions and future work are shown in Section V.

II. MongoDB

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 [2], [3], [4], [5].

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.

Table 1 shows the relationship of RDBMS terminology with MongoDB [3], [4], [5].

TABLE I.
THE RELATIONSHIP OF RDBMS TERMINOLOGY WITH MONGODB

RDBMS	MongoDB
Database	Database
Table	Collection
Tuple/Row	Document
Column	Field
Table Join	Embedded Documents
Primary Key	Primary Key (Default key _id provided by MongoDB itself)

Any relational database has a certain design schema that shows the tables and the relationships between them. In MongoDB there is no concept of relationship.

The advantages of MongoDB over RDBMS are [2], [3], [4], [5]: schema-less (MongoDB is a document database in which one collection holds different documents whose number of fields, content and size can be different from one document to another), structure of a single object is clear, no complex joins, deep query-ability (MongoDB supports dynamic queries on documents using a document-based query language that is almost as powerful as SQL), tuning, ease to scale-out, conversion / mapping of application objects to database objects is not needed, uses internal memory for storing the working set, enabling faster access to data.

III. THE GENERAL PRINCIPLES OF MAPPING RELATIONAL DATABASES TO MONGODB

In MongoDB the relational database remains a database. A relational table is mapping to a MongoDB collection. The tuples or rows become documents inside MongoDB collections [3], [4], [5].

The 1:1 relationship describes a relationship between two entities. For example a Student has a single Address relationship. A Student lives at a single Address and an Address only contains a single Student. The 1:1 relationship can be modeled in two ways using MongoDB. The first is to embed the relationship as a document and the second is as a link to a document in a separate collection [4], [5].

In the one to one relationship embedding is the preferred way to model the relationship as it’s more efficient to retrieve the document.

The 1:M relationship describes a relationship where one side can have more than one relationship while the reverse relationship can only be single sided.

The 1:M relationship can be modeled in several different ways using MongoDB. The first model is embedding, the second is linking and the third is a bucketing strategy that is useful for cases like time series [4], [5].

A N:M relationship in the ER model is an example of a relationship between two entity types where they both might have many relationships between entities. An example might be a Book that was written by many Authors. At the same time an Author might have written many Books.

N:M relationships are modeled in the relational database by using a join table that contains the primary keys from the original ones, each representing a foreign key, and two 1:M relationships.

In MongoDB we can represent this situation in many ways. The first way is called Two Way Embedding [4], [5].

In Two Way Embedding we will include the Book foreign keys under the book field in the author document. Mirroring the Author document, for each Book we include the Author foreign keys under the Author field in the book document.

Another way of modeling N:M relationships is called One Way Embedding [4], [5].

The One Way Embedding strategy chooses to optimize the read performance of a N:M relationship by embedding the references in one side of the relationship. An example might be a N:M relationship between books and categories. The case is that several books belong to a few categories but a couple categories can have many books. Let’s look at an example with the categories represented into a separate document.

An example of Category documents:

```
{id=1,
Cname=”Multimedia”}
{id=2,
Cname=”Databases”}
```

An example of a Book document with foreign keys for Categories:

```
{id:1,
title”Multimedia Databases”,
categories:[1, 2],
authors:[1]}
id:2,
title”Multimedia”,
categories:[1],
authors:[1,2]}
```

IV. FRAMEWORK DESCRIPTION

The framework implements an algorithm of automatic mapping of MySQL relational databases to MongoDB.

The algorithm uses the MySQL INFORMATION_SCHEMA that provides access to database metadata. Metadata is data about the data, such as the name of a database or table, the data type of a column, or access privileges. INFORMATION_SCHEMA is the information database, the place that stores information about all the other databases that the MySQL server maintains. Inside INFORMATION_SCHEMA there are several read-only tables. They are actually views, not base tables [6], [7]. For examples we use a database db1 that contains 5 tables: Employee, Department, Project, Child and Works_on. The relationships are 1:M between Department and Employee, Employee and Child, Department and Project (figure 1). In the ER model there is a N:M relationship, implemented in the relational model by the join table Works_on and two 1:M relationship between Employee and Works_on and Project and Works_on.

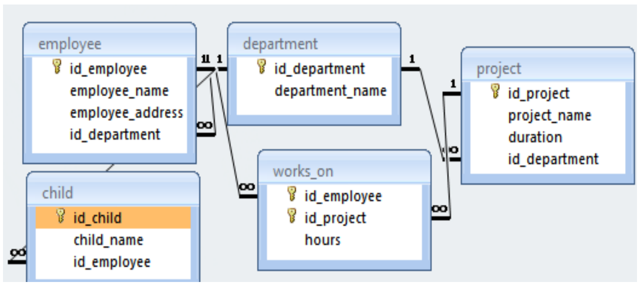


Fig. 1 A relational database

The steps of the algorithm implemented in our framework are presented next.

1. Creating the MongoDB database

The user must specify the MySQL database that will be represented in MongoDB. The database is created with the following MongoDB command: use DATABASE_NAME [5].

```
>use db1
switched to db db1
```

2. Creating tables in the new MongoDB database

The algorithm verifies for each table in what relationships is involved, if it has foreign keys and/or is referred by other tables.

2.1 If the table is not referred by other tables, it will be represented by a new MongoDB collection.

2.2 If the table has not foreign keys, but is referred by another table, it will be represented by a new MongoDB collection.

2.3 If the table has one foreign key and is referred by another table, it will be represented by a new MongoDB collection. In our framework, for this type of tables we use linking method, using the same concept of foreign key.

2.4 If the table has one foreign key but is not referred by another table, the proposed algorithm uses one way embedding model. So, the table is embedded in the collection that represents the table from the part 1 of the relationship.

2.5 If the table has two foreign keys and is not referred by another table, it will be represented using the two way embedding model, described in section IV.

2.6 If the table has 3 or more foreign keys, so it is the result of a N:M ternary, quaternary relationships, the algorithm uses the linking model, with foreign keys that refer all the tables initially implied in that relationship and already represented as MongoDB collections. The solution is good even the table is referred or not by other tables.

In order to find the name of the tables stored in MySQL database the next Select command is used:

```
Select table_name From information_schema.tables
Where table_schema='db1' Order By table_name;
```

The INFORMATION_SCHEMA.TABLES provides information about tables in databases [6], [7].

The table TABLES_CONSTRAINTS from INFORMATION_SCHEMA database describes which tables have constraints [6], [7]. It must be executed a Select command on each table in database, as in the next example:

```
Select constraint_type
From information_schema.tables_constraints
Where table_schema='db1'
And table_name='department';
```

The CONSTRAINT_TYPE value can be unique, primary key or foreign key. We are interested by primary key and foreign key constraints [6], [7].

The table REFERENTIAL_CONSTRAINTS from the same MySQL system database provides information about foreign keys. The attributes constraint_schema and constraint_name identify the foreign key. The attributes: unique_constraint_schema, unique_constraint_name and referenced_table_name identify the referenced key [6], [7].

With the data from these system tables: tables, tables_constraints and referential_constraints the framework can establish what step of the algorithm (2.1-2.6) must be applied.

Relational tables become collections in MongoDB. The collections are created using createCollection() method.

Basic syntax of **createCollection()** command is as follows [5]:

```
Db.createCollection(name, options)
```

Where name represents the collection name and options specify options about memory size and indexing.

For the relational database from figure the mapping according to the presented algorithm is presented next. The table Department has no foreign keys but is referred by other two tables, so it becomes a MongoDB collection (step 2.2).

The table Employee has one foreign key and is referred by the table Works_on, so it becomes a collection (step 2.3).

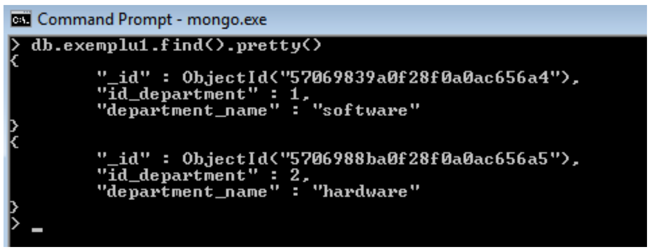
The table Project has one foreign key and is referred by Works_on, so it becomes also a collection (step 2.3).

The table Works_on has two foreign keys and is not referred by other tables, so it will be implemented using two way embedding model (step 2.5). The projects will be assigned to each employee and also, to each project will be assigned the employees that work on that project.

The table Child has foreign key but is not referred by another table, so it will be represented using one way embedding model (step 2.4). So it will be embedded in Employee collection.

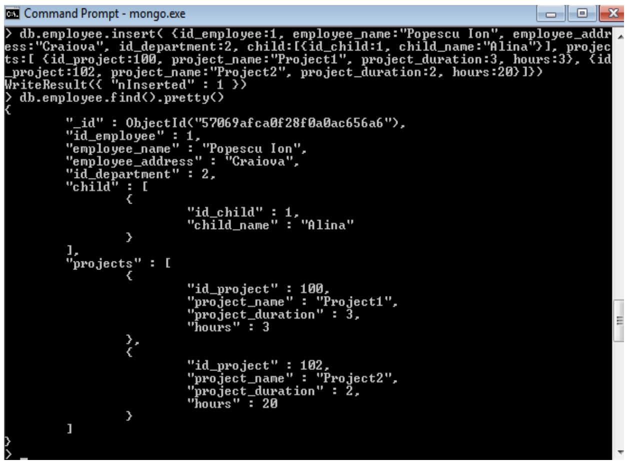
The five relational tables will be represented by three MongoDB collections. Next, there are some samples of the MongoDB collections generated by the presented algorithm.

Department collection is presented in figure 2.



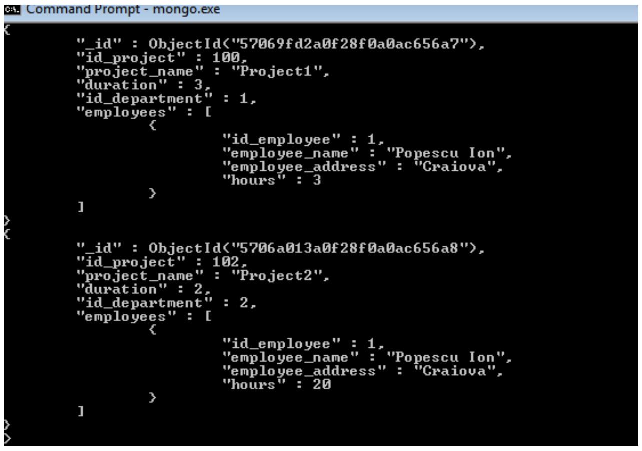
```
Command Prompt - mongo.exe
> db.exemplu1.find().pretty()
{
        "_id" : ObjectId("57069839a0f28f0a0ac656a4"),
        "id_department" : 1,
        "department_name" : "software"
}
{
        "_id" : ObjectId("5706988ba0f28f0a0ac656a5"),
        "id_department" : 2,
        "department_name" : "hardware"
}
> _
```

Fig. 2 MongoDB collection that represents the Department table



```
Command Prompt - mongo.exe
> db.employee.insert( {id_employee:1, employee_name:"Popescu Ion", employee_addr
ess:"Craiova", id_department:2, child:[{id_child:1, child_name:"Alina"}], projec
ts:[ {id_project:100, project_name:"Project1", project_duration:3, hours:3}, {id
_project:102, project_name:"Project2", project_duration:2, hours:20}]})
WriteResult({ "nInserted" : 1 })
> db.employee.find().pretty()
{
        "_id" : ObjectId("57069afca0f28f0a0ac656a6"),
        "id_employee" : 1,
        "employee_name" : "Popescu Ion",
        "employee_address" : "Craiova",
        "id_department" : 2,
        "child" : [
                {
                        "id_child" : 1,
                        "child_name" : "Alina"
                }
        ],
        "projects" : [
                {
                        "id_project" : 100,
                        "project_name" : "Project1",
                        "project_duration" : 3,
                        "hours" : 3
                },
                {
                        "id_project" : 102,
                        "project_name" : "Project2",
                        "project_duration" : 2,
                        "hours" : 20
                }
        ]
}
> _
```

Fig. 3 MongoDB Employee collection



```
Command Prompt - mongo.exe
{
        "_id" : ObjectId("57069fd2a0f28f0a0ac656a7"),
        "id_project" : 100,
        "project_name" : "Project1",
        "duration" : 3,
        "id_department" : 1,
        "employees" : [
                {
                        "id_employee" : 1,
                        "employee_name" : "Popescu Ion",
                        "employee_address" : "Craiova",
                        "hours" : 3
                }
        ]
}
{
        "_id" : ObjectId("5706a013a0f28f0a0ac656a8"),
        "id_project" : 102,
        "project_name" : "Project2",
        "duration" : 2,
        "id_department" : 2,
        "employees" : [
                {
                        "id_employee" : 1,
                        "employee_name" : "Popescu Ion",
                        "employee_address" : "Craiova",
                        "hours" : 20
                }
        ]
}
>
```

Fig. 4 MongoDB Project collection

Employee Collection that contains the document Child and Projects is presented in figure 3.

Project collection that embeds the documents employees that work on these projects is shown in figure 4.

V. Conclusion and future work

The paper presents a framework that implements our original algorithm of automatic mapping a MySQL relational database to a MongoDB NoSQL database. The algorithm uses the metadata stored in the MySQL 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 algorithm was presented in detail, on steps. Also, the paper contains an example of automatic mapping of a MySQL database to MongoDB using our algorithm.

The paper also presents the initial results of our algorithm that was tested on small size databases (10-15 tables with many relationships and 100 records/table), the results being encouraging.

The future work will include the next steps:

- Experiments on complex databases (many tables and a large number of records/table)
- Taking into consideration the number of records in the tables and the operations on the database (insert, update, delete query) in order to implement the more appropriate model of mapping to MongoDB
- Modeling tree structures with parent references
- Extending the framework to execute mapping to MongoDB of other relational databases (Oracle, MS SQL Server and so on)

References

[1] http://www.datastax.com/nosql-databases
[2] https://www.thoughtworks.com/insights/blog/nosql-databases-overview
[3] https://leanpub.com/mongodbschemadesign/read
[4] http://code.tutsplus.com/articles/mapping-relational-databases-and-sql-to-mongodb--net-35650
[5] https://docs.mongodb.org/manual/core/data-modeling-introduction/
[6] https://dev.mysql.com/doc/refman/5.0/en/information-schema.html
[7] https://dev.mysql.com/doc/refman/5.5/en/introduction.html
[8] https://www.devbridge.com/articles/benefits-of-nosql/