Automated Algorithm for Data Migration from Relational to NoSQL Databases

Alza A. Mahmood
Computer Engineering Dept.
University of Technology, Baghdad, IRAQ
ajm_alubaidy@yahoo.com

Received: *19-Sep.-2017* ***Revised:*** *16-Jan.-2018* ***Accepted:*** *18-Jan.-2018*
http://doi.org/10.29194/NJES21010060

Abstract

One of the barriers that the developer community face once turning to the newly, highly distributable, schema agnostic and non-relational database, called NoSQL, which is how to migrate their legacy relational database (which is already filled with a large amount of data) into this new class of database management systems. This paper presents a new approach for converting the already filled relational database of any database management system to any type of NoSQL databases in the most optimized data structure form without bothering of specifying the schema of tables and relations between them. In addition, a simplified software as a prototype based on this algorithm is built to show the results of the output for testing the validity of the algorithm.

Keywords: NoSQL, SQL, Data Migration, Relational Databases, MongoDB

1 Introduction

Most of the business world is now relying heavily on technologies –big data, cloud storage, and real-time database systems. The most widely used and conventional type of database systems is The Relational Database Management Systems (RDBMSs). Relational databases are based on the relational model of data where data is organized into tables of columns and rows, with a unique key identifying each row [1]. Connections between these tables can be established to cross-reference information between them, called Relationships. RDBMS use SQL for managing stored data in the relational databases.

Digital economy based businesses longed for a real time data management with unmatched levels of scalability, speed, and data variability. Relational databases are unable to meet such new requirements for these reasons:

- They cannot process unpredictable and unstructured data, as a predefined schema that is required in advance.
- Horizontal scalability (running them on multiple servers that work together) [2] is very difficult to implement with RDBMS that achieve ACID (Atomicity, Consistency, Isolation, Durability)
- They do not support object oriented data structure, so complex queries are required to manipulate a stored object with its data.

NoSQL (originally referring to "non SQL" or "non-relational" also known as “Not Only SQL” [3]) is a new database technology which is proposed as a consequence of the growth of the Internet to overcome the drawbacks of the RDBMS. Many enterprises, especially popular Internet companies such as Twitter, LinkedIn, NetFlix, Amazon, and Google, are now adopting it for their crucial business applications as it can afford the massive amount of data. The work with NoSQL started in the early of 2009 [4].

NoSQL is a cloud-based, dynamic approach to process unstructured data efficiently. It is a mechanism for storing and retrieving data using more flexible data structure than the tabular relations of RDBMS [5]. A basic classification of NoSQL databases, by data model, is explained in the following:

- Columns: each column represents a tuple (a key-value pair) and consist of three elements, a unique name, a value and a timestamp [6]. Cassandra, Accumulo and Druid are good examples of this model.
- Key-value: it is the simplest form of NoSQL where a value of any data type can be stored along with a unique key referring to that value. Examples: Dynamo, Aerospike and Oracle NoSQL
- Document: most widely used model of NoSQL. Documents are nearly equivalent to the concept of an object in OOP languages. Different types of documents can be stored in a single store. Popular commercial NoSQL database engines use this model, such as MongoDB, Couchbase, Apache CouchDB, and Clusterpoint [4].

Most of the data are transmitted between systems in standard formats, such as XML and JSON. NoSQL gives the ability to store data in these formats as it is, means lessen the effort to extract, transform, and load (ETL) the transmitted data. NoSQL, unlike its rival relational databases, is considered a schema-less storage (or schema-

free) which allows adding different new types of data whenever needed without specifying its schema. Making it a good choice for rapid application development (RAD), as it does not need to be prepared ahead of time [6].

The most well-known feature of NoSQL that surpass the traditional RDBMS, is that it can be easily scaled horizontally as NoSQL handle partitioning of a database across several inexpensive servers that works together as a single data center. In contrast to the RDBMS where more expensive hardware for the same single server are required to be scaled up (vertical scaling) [2].

One of the challenges that is faced while moving to NoSQL is how to map existing data of the relational database in its tabular form into the NoSQL structure form. Another problem is how to deal with all relationships between tables and how to make sure related data entities reference to each other, as NoSQL does not support relationships. It is almost impossible to do that manually without writing a program just for that purpose, especially when dealing with huge datasets. Beside it is time consuming, writing a program for migrating can result in poor data structure if it is not built carefully. This is the aim of the proposed algorithm in this paper. It is a short way for migrating any filled relational database to a NoSQL database of any desired DBMS in an optimized way without bothering about relationships and data structures.

The paper is organized as follows: related work is presented in Section 2 of this paper. Section 3 presents the main concept of the proposed algorithm with a detailed representation in steps. Section 4 presents an implementation of the algorithm as a software to demonstrate it and show the results. Finally, conclusions are presented in Section 5.

2 Related Work

As NoSQL considered a revolutionary, newborn mechanism in information technology field, many academic works have reviewed it. Vatika Sharma and Meenu Dave gave an overview of NoSQL about how it has weakened the domination of SQL systems and described its background, characteristics, and fundamentals [7]. Aparna Babu and Subu Surendran presented an overview of several approaches for performing relational to NoSQL migration and compared NoSQL databases based on their structure, performance, scalability, consistency, and transactional features [3]. Neelima Kuderu and Dr. Vijaya Kumari presented RDBMS-to-NoSQL migration framework in two components: schema mapping and SQL to NoSQL query mapping [8]. Steve A. T et al. showed the limitations of the relational model in large systems architecture and proposed an approach for migration from a relational database to column oriented NoSQL database [9]. Mohamed Hanine et al. presented a methodology for data migration from RDBMS to NoSQL database and implemented a software prototype using MySQL as an RDBMS and MongoDB as a NoSQL database to illustrate that methodology [1]. Liana Stanescu et al. presented a framework that implements their original algorithm of automatic mapping a MySQL relational database to a MongoDB NoSQL database [10].

3 Proposed Algorithm

The main input of the algorithm is a relational database of any DBMS, with other inputs for simple preferences. Developers prefer the document data model as any structure of data can be easily stored and merged, without forsaking data access and indexing functionality. This is why the output of the algorithm is a document-oriented NoSQL database.

The main concept of the algorithm is to iterate through rows of each table in the input relational database and map data of each row into a *document*, as shown in Fig. 1, where documents mapped from rows of the same table will be stored together as a *collection* in the output NoSQL database.

Relationships between tables in RDBMS are achieved using primary and foreign key columns. NoSQL has no such a straightforward mapping of these relationships; it uses instead the concept of embedded and linking documents. Let us term the table with the primary key column as *parent* while the one with the foreign key as *child*.

The algorithm works in three different modes, each mode determines the data structure of the resulted NoSQL database, that are explained in the following:

ID	Type	Cost	Date
1	Wages	1220	1996-06-04

(a)

```
{
  "_id" : 1,
  "Type" : "Wages",
  "Cost" : 1220,
  "Date" : 1996-06-03
}
```

(b)

Figure 1: (a) A row in a table of an SQL database
(b) NoSQL document in JSON format

- Embedded: In embedded mode, collections are created only for parent tables that have no

further reference into other tables. Each parent row will be mapped into a parent document, while all its child rows (those refer to that parent row) will be mapped to an array of documents and included as a field in that parent document. The same process recurs for nested related tables (See Fig. 5). This mode ensures a fast query of child documents from their parent. It is not recommended for large, complex hierarchical data, as there is a limit for the size of a document in most NoSQL databases.

- Linking: Linking mode means creating a collection for each table, whether it has a reference to another table or not. As the name states, each mapped document from a row in a child table refers to its parent document using its corresponding foreign key field value, making it possible to seek child documents from their parent (See Fig. 6). The linking mode is used if data of related documents is expected to grow larger in size to avoid exceeding the document maximum size.
- Auto: considered as the most optimized way to structure data. The maximum data size of a document of the output NoSQL database is required as an input value to the algorithm, so each array of child documents will be mapped in embedded mode as its size in addition to the size of its parent document does not exceed that maximum value, otherwise linking mode will be applied.

Finally, an input of Boolean type is required to work as a flag that indicates whether or_not to neglect *null* values in rows of the input database during migrating.

SQLDataBase is the input that holds the relational database, *StructureMode* is for the working mode, and *MaxDocSize* is the maximum allowed size in bytes of a document in the output chosen NoSQL database type.

The *main* procedure of the proposed algorithm can be summarized as in the following:

Main Procedure

Input: *SQLDatabase, StructureMode*

Step 1: Create an empty *NoSQLDatabase*

Step 2: If *StructureMode* is *Linking* then create a *collection* for each table in *SQLDatabase* using *CreateCollection* procedure and insert it to the *NoSQLDatabase*

Step 3: Else if *StructureMode* is *Linking*, then for *Parent* tables only of *SQLDatabase* which has no parent relations (no foreign keys) create a *collection* for each one of them using *CreateCollection* procedure and insert it to the *NoSQLDatabase*

Step 4: End

Output: *NoSQLDatabase*

The *CreateCollection* procedure receives an SQL data table as a parameter and return a collection of documents as an output, it states as follow:

Create Collection Procedure

Input: *Table, StructureMode, MaxDocSize, and NeglectNull*

Step 1: Create a *collection* as a list of NoSQLDocuments using NoSQL commands with the same name as *Table* name

Step 2: For each row in *Table* **do step 3 to 7**

Step 3: Create a *document* As NoSQLDocument with ID field equal to the value of Primary Key column of the row

Step 4: For each other column in *Table*, insert a field into the *document* with column name as name and value equal to value of the row in that column or an empty value if its value is null and *NeglectNull* is set to False. Otherwise, neglect that column if *NeglectNull* is set to True and its value is null.

Step 5: If *StructureMode* is *Linking* **Then go to step 7**

Step 6: For each child table of *Table* do the following:

(i) Query all *childRows* (those refer to that row using a foreign key column) according to that child table.

(ii) If there are no *childRows* for that row then skip **steps iii, iv, and v.** If *NeglectNull* is set to False then insert a field into the *document* with child table name as name and an empty value before skipping.

(iii) Create a *collection* for that child table filled with *childRows* only using *CreateCollection* procedure

(iv) If *StructureMode* is *Auto* then check the size of *childRows* plus the size of the document, if the total size exceeds *the MaxDocSize* then Insert the *collection* created in **step iii** into the *NoSQLDatabase*

(v) Else if the total size does not exceed the *MaxDocSize* or *StructureMode* is *Embedded* then remove the foreign key field of each document in the *collection* created in **step iii** and insert a field into the *document* with child table name as name and that *collection* as value

Step 7: Insert the *document* into the *collection*

Step 8: End

Output: *collection*

4 Proposed Software

For the purpose of testing and achieving output results as what the novel algorithm propose, an implementation of it as a software working on PC platforms should be used. The software is written in Visual Basic. NET (VB. NET) which work in .NET framework. The input database that has been used for testing is running under Microsoft SQL Server ®, an acclaimed/famous relational DBMS that is a favorite of developers because of its reliability, efficiency, and ease of use. MongoDB ® is used as the output NoSQL database. MongoDB is known for its easy quick setup. Other features of MongoDB are high-performance, high availability, automatic scaling, and ability to support fast iterations.

The interface is so simple and user-friendly, as showing in Fig. 2. Firstly, the user needs to connect to the source relational database that is required to be converted, by specifying the database provider, which is known better as the type of the DBMS, along with other parameters of the connection string. The proposed software comes with a wide list of common types of databases providers for the source and the output databases. Secondly, the input parameters, which explained in Section III of this paper, should be filled carefully. Notice that if the user chose a structure mode other than *Auto*, then there is no need to determine the maximum document size. Finally, the user should provide the connection string of the output NoSQL server, where the output NoSQL database will be hosted.

The schema of the demo source database is very simple. However, it utilizes all the features of the algorithm (Fig. 3). The database has been filled with sample data as shown in Fig.4.

After filling the fields of the software form and begin migrating, the output (Fig. 5 and 6) can be explored using a MongoDB management tool (such as *Robo 3T*). Notice that attributes with null values of a row such as 'Address' of the employee 'Davolio Nancy' are suppressed in the output NoSQL document when setting *Neglect Null* to true (Fig. 5) while when setting it to false, they remained with an empty value (Fig. 6).

If *Auto* mode is chosen, and *Maximum Document Size* is set to 0.0027 MB (which equals to 2700 bytes) then orders of the employee 'Davolio Nancy' will be migrated into orders collection with foreign key field refers to that employee (*Linking* mode) as their size exceed that maximum value. While when using 0.004 MB as maximum size, all orders will be submitted to the *Embedded* mode technique as they are in the safe side of the document size limit.

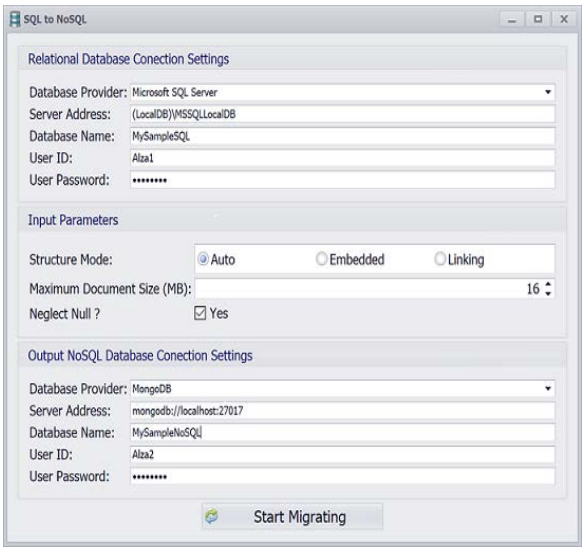

Figure 2: The GUI interface of the proposed software.

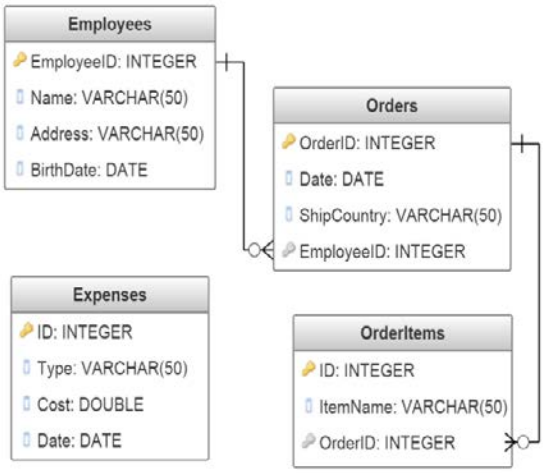

Figure 3: Schema design of the source demo database.

Employees

EmployeeID	Name	Address	BirthDate
1	Davolio Nancy	*NULL*	1948-12-08
2	Fuller Andrew	908 W. Capital Way	1952-02-19
3	Leverling Janet	722 Moss Bay Blvd.	*NULL*

Orders

OrderID	Date	ShipCountry	EmployeeID
1	1996-07-04	France	1
2	1996-07-05	Germany	1

OrderItems

ID	ItemName	OrderID
1	Chai	1
2	Chang	1

Expenses

ID	Type	Cost	Date
1	Wages	1220	1996-06-04
2	Electricity Bill	NULL	1996-07-12

Figure 4: Sample data of the source demo database.

```
Employees:
{
"_id" : 1,
"Name" : "Davolio Nancy",
"BirthDate":ISODate("1948-12-
07T21:00:00.000Z")
},
{
"_id" : 2,
"Name" : "Fuller Andrew",
"Address" : "908 W. Capital Way",
"BirthDate":ISODate("1952-02-
18T21:00:00.000Z")
},
{
"_id" : 3,
"Name" : "Leverling Janet",
"Address" : "722 Moss Bay Blvd."
}
Orders:
{
"_id" : 1,
"Date":ISODate("1996-07-
03T20:00:00.000Z"),
"EmployeeID" : 1,
"ShipCountry" : "France"
},
{
"_id" : 2,
"Date":ISODate("1996-07-
04T20:00:00.000Z"),
"EmployeeID" : 1,
"ShipCountry" : "Germany"
}
OrderItems:
{
"_id" : 1,
"ItemName" : "Chai",
"OrderID" : 1
},
{
"_id" : 2,
"ItemName" : "Chang",
"OrderID" : 1
}
Expenses:
{
"_id" : 1,
"Type" : "Wages",
"Cost" : 1220.0,
"Date":ISODate("1996-06-
03T20:00:00.000Z")
},
{
"_id" : 2,
"Type" : "Electricity Bill",
"Date":ISODate("1996-07-
11T20:00:00.000Z")
}
```

Figure 5: NoSQL output result in JSON format when using *Linking* mode and setting *Neglect Null* to true.

```
Employees:
{
"_id" : 1,
"Name" : "Davolio Nancy”,
"Address" : "",
"BirthDate":ISODate("1948-12-
07T21:00:00.000Z"),
"Orders" : [
    {
     "_id" : 1,
     "Date":ISODate("1996-07-
        03T20:00:00.000Z"),
     "ShipCountry" : "France",
     "OrderItems" : [
         {
                  "_id" : 1,
                  "ItemName" :
                "Chai"
                  },
                  {
                  "_id" : 2,
                  "ItemName" :
                "Chang"
                  }
                  ]
    },
    {
     "_id" : 2,
     "Date":ISODate("1996-07-
        04T20:00:00.000Z"),
     "ShipCountry" : "Germany",
     "OrderItems" : ""
    }
    ]
}
Expenses:
{
 "_id" : 1,
 "Type" : "Wages",
 "Cost" : 1220.0,
 "Date" : ISODate("1996-06-
 03T20:00:00.000Z")
},
{
 "_id" : 2,
 "Type" : "Electricity Bill",
 "Cost" : "",
 "Date" : ISODate("1996-07-
 11T20:00:00.000Z")
}
```

Figure 6: NoSQL output result in JSON format when using *Embedded* mode and setting *Neglect Null* to false.

5 Discussion

After discussing obstacles and problems of relational database systems and how today world is in need of real time, highly scalable technologies, NoSQL has been demonstrated and proposed as an alternative solution to handle the

shortcomings of the conventional relational DBMS. The paper also discussed a classification of the NoSQL systems.

The main issue that the paper aimed is how the developer community encounter quandaries when deciding to switch to this new technology from their relational-based systems. Next, the paper presented a novel, easy-to-implement, fully automated algorithm used for migrating data from any type of relational database to a new NoSQL database of any type with no effort. The algorithm taken into consideration the relations between tables of the source relational database and allows the user to choose between three different modes: *Embedded*, *Linking*, and *Auto* to decide how to map these relations to the output database. The chosen mode will define the structure of the output NoSQL database in an efficient way.

An application software has been developed using VB.NET as an implementation of this algorithm. To show the results of the algorithm, a simple MS SQL Server database filled with sample data and used as an input to that application. After the migration process finished, a MongoDB NoSQL database has been created automatically with data taken from the source database as the algorithm proposed.

References

[1] Mohamed H., Abdesadik B., Omar B., Data Migration Methodology from Relational to NoSQL Databases, International Journal of Computer, Electrical, Automation, Control and Information Engineering, Vol. 9, No.12, pp. 2556-2560, 2015.

[2] Hesham E., Mostafa A. , Advanced Computer Architecture and Parallel Processing, John Wiley & Sons, 2005.

[3] Aparna B., Subu S., Relational to NoSQL Database Migration, National Conference on Advanced Computing, Communication and Electrical Systems, Vol. 6, No.5, pp. 58-62, 2017.

[4] Ganesh D., Penthuru R., Cloud Infrastructures for Big Data Analytics, IGI Global, 2014.

[5] Rick C., Scalable SQL and NoSQL data stores, Newsletter-ACM SIGMOD Record, Vol. 39, No. 4, pp. 12-27, 2010.

[6] Gaurav V., Getting started with NoSQL, Packt, 2013.

[7] Vatika S., Meenu D., SQL and NoSQL Databases, International Journal of Advanced Research in Computer Science and Software Engineering, Vol. 2, No.8, pp. 20-27, 2012.

[8] Neelima K., Vijaya K., Relational Database to NoSQL Conversion by Schema Migration and Mapping, International Journal of Computer Engineering In Research Trends, Vol. 3, No.9, pp. 506-513, 2016.

[9] Steve A. T., Luís G. M., Patrick A. B., From Relational Database to Column-Oriented NoSQL Database: Migration P nNrocess, International Journal of Engineering Research & Technology, Vol. 4, No.5, pp. 399- 403, 2015.

[10] Liana S., Marius B., Dumitru D. B., Automatic Mapping of MySQL Databases to NoSQL MongoDB, Proceedings of the Federated Conference on Computer Science and Information Systems, Vol. 8, pp. 837–840, 2016.

خوارزمية الية لترحيل البيانات من قاعدة بيانات علائقية الى غير علائقية

الزا عبدالجبار محمود
قسم هندسة الحاسوب
الجامعة التكنولوجية

الخلاصة

احدى العوائق التي يواجها مجتمع المطورين عند التحويل الى نظام قاعدة البيانات الغير علائقية الحديثة التي تسمى بـNoSQL وتتصف بسهولة التوزيع ولا تحتاج الى مخطط قاعدة بيانات، هي كيفية ترحيل قاعدة البيانات العلائقية الخاصة بهم، التي تحتوي على كم هائل من البيانات، الى هذا الصنف الجديد من أنظمة إدارة قواعد البيانات. هذا البحث يقدم خوارزمية مبتكرة لتحويل قاعدة بيانات علائقية من أي نوع من أنظمة إدارة قواعد البيانات الى أي نوع من قواعد بيانات الـNoSQL بأفضل صيغة هياكل بيانات ممكنة ومن دون عناء تحديد مخطط للجداول والعلاقات بينها. بالإضافة لذلك، تم تطوير برنامج مبسط كنموذج تجريبي بالاعتماد على هذه الخوارزمية وذلك لعرض نتائجها من اجل اختبار صحتها.