ETL Based Framework for NoSQL Warehousing

Rania Yangui[1], Ahlem Nabli[2] and Faïez Gargouri[1]

1 Sfax University, Institute of Computer Science and Multimedia, BP 1030 - Tunisia
yangui.rania@gmail.com, faiez.gargouri@isimsf.rnu.tn
2 Sfax University, Faculty of Sciences, BP 1171 Tunisia
ahlem.nabli@fsegs.rnu.tn

Abstract. Over the last few years, NoSQL systems are gaining strong popularity and a number of decision makers are using it to implement their warehouses.

Building the ETL process is one of the important tasks of creating NoSQL warehouse. Traditional ETL tools require the structure of the target system to be known at advance. As NoSQL databases are schema-free, this increases the need for extending the existing ETL tool in order to be able to designing schema while integrating data. In spite of the importance of ETL processes in the NoSQL warehousing, little researches have been done in this area due to its complexity.

In this paper, we propose an ETL-based platform for transforming a multidimensional conceptual model into document-oriented one. We model the transformation rules using the Business Process Modeling Notation (BPMN). The resulting warehouse was evaluated in term of "Write Request Latency" and "Read Request Latency" using TPC-DS benchmark.

Keywords: Extract Transform and Load, Business Process Modeling Notation, NoSQL, Data Warehouse, Transformation rules.

1 Introduction

The explosion of social network has lead to the generation of massive volumes of user-generated data that has given birth to a novel area of research, namely data warehousing from social network. By integrating external data from social network (SN) with a company's data warehouse (DW), decision-makers can better anticipate changes in customer behavior, strengthen supply chains, improve the effectiveness of marketing campaigns, and enhance business continuity. Nevertheless, current warehousing methodologies with relational databases cannot be successfully applied for storing and analyzing such data. As result, many new technologies have emerged such as NoSQL warehousing in order to increase the performance and the availability of services. NoSQL, also known as "Not Only SQL" is a term often used to describe a class of non-relational and schema-less databases that scale horizontally to very large data sets.

As argued by many researchers, building the ETL process is the biggest tasks of building a warehouse. In fact, it is complex, time consuming, and uses most of data warehouse projects implementation efforts, costs, and resources [1]. What makes this process even more challenging is the schema-less feature of NoSQL databases. The schema-less nature of the document-oriented database means that the designer can store documents in any shape but the notion of schema itself does not disappear from the model. Therefore, the mapping to NoSQL data storage means a move from explicitly defined data structures to implicit ones. These databases assign the responsibility to maintain the schema to the developer. Consequently, the creation of a schema occurs while inserting data at ETL level.

In the literature, there were similar researches for loading data to NoSQL databases. However, those have considered only the extraction and loading tasks. The transformation operations that reflect the implicit target schema are not addressed.

This paper provides a general implementation strategy for NoSQL data warehousing. In this context, we explain the need of a NoSQL data warehouse over traditional warehouse and we propose rules for implementing a DW under document-oriented system. These rules are implemented using java routines integrated with Talend "Talend Open Source for Big Data". In our proposal, we suggest modeling the ETL process using BPMN that allows covering the deficit of communication between the design and implementation of such process. The obtained NoSQL data warehouse was evaluated in terms of "Write Request Latency" and "Read Request Latency" using TPC-DS benchmark.

This paper is organized as follows. Section 2 represents a literature survey. Section 3 introduces the generic approach. Section 4 presents a rule-based approach for generating NoSQL Data Warehouse. Section 5 addresses the ETL process. Section 6 represents the evaluation tasks. Section 7 concludes the paper and draws future research directions.

2 Literature Survey

In this section, we review two main themes of literature related to the contributions made in this paper. Firstly, we present works that treat the implementation of data warehouses under NoSQL databases. Secondly, we discuss previous research in modeling ETL process using BPMN.

2.1 NoSQL Data Warehousing

NoSQL systems have shown their advantages over relational systems in terms of flexibility and the ability to handle massive data. The authors in [2] explained the need of a NoSQL data warehouse over traditional warehouse. They recognized that traditional Data warehouse designed over MOLAP or ROLAP are not always easy to leverage the opportunities of storing big data. In fact, growth in data volumes, number of data sources and type of data are a major area of concern for the organization. Consequently, they proposed to use NoSQL Database for implementing a data warehouse.

In the literature, many researchers have proposed approaches for the migration from relational databases to NoSQL ones. However, few works have focused on the transformation of the multidimensional conceptual model into NoSQL logical one. For example, the author in [3] showed how to build and store multidimensional data on top of big data systems and how to extend typical OLAP operators and parallel OLAP query processing on distributed key-value data store systems. However, the main goal of this paper is to propose a benchmark.

Moreover, in [4] and [5], the authors proposed three approaches which allow big data warehouses to be implemented under the column oriented NoSQL model but without giving the formalization for the modeling process. Each approach differs in terms of structure and the attribute types used when mapping the conceptual model into logical model is performed.

Furthermore, the author in [6] and [7] tried to define a logical model for NoSQL data stores (oriented columns and oriented documents). The authors proposed a set of rules to map star schemas into two NoSQL models: column-oriented (using HBase) and document oriented (using "MongoDB").

Recently, the authors in [8] presented a set of rules for the transformation of multidimensional data models into Hive tables, making available data at different levels of detail. These several levels are suited for answering different queries, depending on the analytical needs.

Similarly, in [9], the authors discussed the possibilities to create data warehouse solutions by using NoSQL database management systems. The main challenge is to find a good balance between characteristics of classical data warehouses using relational database management systems and opportunities offered by NoSQL database management systems. The paper described processes of creation and production of data warehouse using a NoSQL data mart and outlined requirements for technology necessary for such processes. The research is based on practical experience when implementing NoSQL data marts with MongoDB and Clusterpoint DB.

As NoSQL data stores are becoming increasingly popular in application development, the above presented works have shown that it is possible to implement DW under NoSQL storage. These systems are attractive for developers due to their ability to handle large volumes of data, as well as data with a high degree of structural variety. However, it appears that the majority of researchers use HBase for implementing a NoSQL DW. This is justified by the resemblance between the logic model HBase and that of relational databases, particularly in terms of concepts of tables and rows. Besides, there is an interesting concept that has never been used in the proposed transformation rules, namely embedded documents and super columns.

In the other hand, the majority of authors neglected the ETL process, which is an essential stage in the construction of data warehouse. They did not explicitly define the different function of the ETL process. However, as NoSQL databases claim to be schema-less, it need careful migration due to the implicit schema in any code that accesses the data. As a result, it requires more programming to obtain needed DW.

2.2 BPMN based ETL process modeling

ETL is often a complex combination of process and technology that consumes a significant portion of the data warehouse development efforts and requires the skills of business analysts, database designers, and application developers. A solid, well-designed, and documented ETL system is necessary for the success of a data warehouse project.

As the ETL process is critical in building and maintaining the DW systems, some researchers tried to represent it using BPMN language. In this context, the authors in [10] presented a method to guide BPMN specifications in the definition of conceptual models of ETL systems.

Similarly, the authors in [11] provided an independent platform for the conceptual modeling of an ETL process based on BPMN. Using the same BPMN objects presented by [12], the authors proposed a correspondence between the ETL process and the needs of decision-makers to easily identify which data are necessary and how include them in the DW.

Thereafter, in [13], the authors defined specific conceptual model that takes into account othe capture of evolutionary data, the change of dimensions, the treatment of the substitutions keys and the data quality.

Later, the authors in [14] and [15] presented a two-levels approach for the construction of web warehouse by means of BPMN. The first level is focused on helping the user to configure the web data sources and the desired data quality characteristics. The second level uses the defined configuration to generate the warehouse.

As first initiative, the authors in [16] identified and implemented a framework for NoSQL Databases. In the Methodology, it is realized that it would take a lot more time and effort to create an enterprise level application for the NoSQL ETL Framework.

Notice that, the majority of works proposed ETL frameworks that work well with relational systems, only the work of [16] takes into consideration the ETL process for NoSQL warehousing. However, this work does not detail and model the proposed process. Moreover, many software vendors, including “IBM”, “Informatica”, “Talend”, and “Pentaho”, provide ETL software tools allowing migration to NoSQL systems. However, the transformations are simple and several NoSQL concepts such as “embedded document” or “super column” are not explored.

Based on the above discussion, there is a strong need for a significant ETL-based platform that allows the implementation of DW under NoSQL system. Representing the ETL process by means of BPMN notations is also recommended.

3 ETL-based framework for NoSQL Warehousing

In the absence of a clear approach that allows the implementation of data warehouses under NoSQL system, we propose in this section a two-level approach for data warehouse building under document-oriented system (Fig.1).

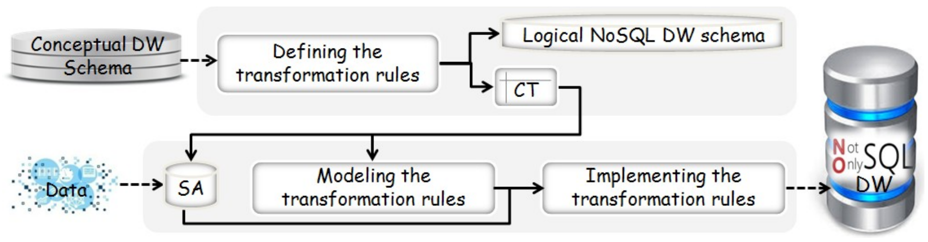

Conceptual DW Schema
Defining the transformation rules
Logical NoSQL DW schema
CT
Data
SA
Modeling the transformation rules
Implementing the transformation rules
Not Only SQL DW

Fig.1. Approach overview.

The first level relates to the definition of the transformation rules for logical NoSQL schema definition. In fact, OLAP servers that base on relational data storage solutions cannot process big volumes of data. In this level, we need to save traceability of the transformation rules on a table called Correspondence Table (CT).

Data are extracted from data sources and loaded into the staging area. The second level consists on using the resulting CT for modeling and implementing the proposed rules. As NoSQL databases are schema-less, to create a NoSQL DW, we are required to write a program that relies on some form of implicit schema. The implicit schema is a set of assumptions about the data's structure in the code that manipulate the data [17]. That mean, the creation of a schema occurs when inserting data at ETL level. In this level, we implement the defined transformation rules that reflect the implicit schema using java routines.

4 NoSQL warehouse logical schema definition

Given that a well-designed DW requires a well planned logical design, all updates and versions of a DW lead to a revision of the logical design. Generally, the mapping from the conceptual to the logical model is made according to three approaches: ROLAP (Relational-OLAP), MOLAP (Multidimensional-OLAP) and HOLAP (Hybrid-OLAP) [18]. All these models are inadequate when dealing with large amount of data which need scalable and flexible systems. As an alternative, NoSQL systems begin to grow.

4.1 The choice of the adequate NoSQL system

In the literature, four types of NoSQL data stores exist [Sharma, 2012]: Key-value Stores, Document Stores, Columnar Stores and Graph Stores. In previous work, we implemented two data warehouses using Cassandra as a columns-oriented system and “MongoDB” as documents-oriented system. These warehouses were evaluated in terms of "Write Request Latency" and "Read request Latency" using “TPC-DS” benchmark. We noted that the DW built under Cassandra is faster than the DW built under “MongoDB” (1000000 rows in 203.23s for Cassandra while 1000000 rows in 437.36s for “MongoDB”). This is argued by the fact that Cassandra uses less memory space and it is known for effective data compression (due to column redundancy).

Subsequently, we tested the behavior of the systems against a set of queries. The results showed that the documents-oriented NoSQL data warehouse is more efficient in terms of interrogation. This is justified by the fact that data in the column-oriented systems are not available in the same place. Subsequently, we choose to implement our DW under document-oriented systems and specially MongoDB.

4.2 Transformation Rules: Hierarchical Transformation to Document-oriented Model

Recall that a data warehouse schema consists of fact with measures, as well as a set of dimensions with attributes, we map the dimensions according to its attributes and the facts according to its measures.

The hierarchical transformation uses different collections for storing facts and dimensions, and uses the simple documents for representing measure and the composed attributes for representing dimension attributes while explaining hierarchies.

Rule 1 represents the transformation of a fact and its measures to the document-oriented model.

Rule 1: *Fact/Measures Transformation.*

Each fact F $\in$ MS^{Fact} is transformed to a document DC (DC^{N}, DC^{Att}) where:

- The name of the document is the name of the fact / $DC^{N} \leftarrow F^{N}$;
- Each measure M $\in$ F is transformed to a simple attribute SA $\in$ DC^{Att} / $SA^{N} \leftarrow M^{N}$;
- Each identifier of a related dimensions is transformed to a simple attribute SA $\in$ DC^{Att} / $SA^{N} \leftarrow D^{id}$.

Moreover, Rule 2 mentions the hierarchical transformation of a dimension and its attributes (Strong and Weak) to the document-oriented model.

Rule 2: *Dimension/Parameters Transformation*

Each dimension $D(D^{N}; D^{Att}; D^{Hier})$ $\in$ MS^{Dim} is transformed to a document $DC(DC^{N}; DC^{Att})$ where:

- The name of dimension D is equivalent to the name of the document / $DC^{N} \leftarrow D^{N}$;
- Each hierarchy $H(H^{N}, H^{P}, H^{A})$ is transformed to a composed attribute CA $\in$ DCatt where:
  - The name of the composed attribute is the hierarchy name / $CA^{N} \leftarrow H^{N}$
  - The values of composed attributes are the simple attributes that represent the weak and the strong attributes / $CA^{Val} \leftarrow H^{P} \cup H^{A}$.

Based on the above defined rules (Rule 1 and Rule 2), we deduce the hierarchical transformation of a multidimensional schema (MS) as mentioned by Rule 3.

Rule 3: *Multidimensional Schema Transformation.*

Each multidimensional schema MS (MS^N, MS^{Fact}, MS^{Dim}, Func) is transformed to a documents collection DCC (DCC^N; DCC^{Val}) where:

- The name of the collection is the name of the MS / $DCC^N \leftarrow MS^N$;
- Each fact F $\in MS^{Fact}$ is transformed to a document DC (DC^N, DC^{Att}) where the name of the document is the name of the fact / $DC^N \leftarrow$ FN (Rule 1);
- Each dimension D(D^N; D^{Att}; D^{Hier}) $\in MS^{Dim}$ is transformed to a document DC(DC^N; DC^{Att}) (Rule 2).

Fig.2 shows an example of transforming an excerpt of TCP-DS multidimensional schema.

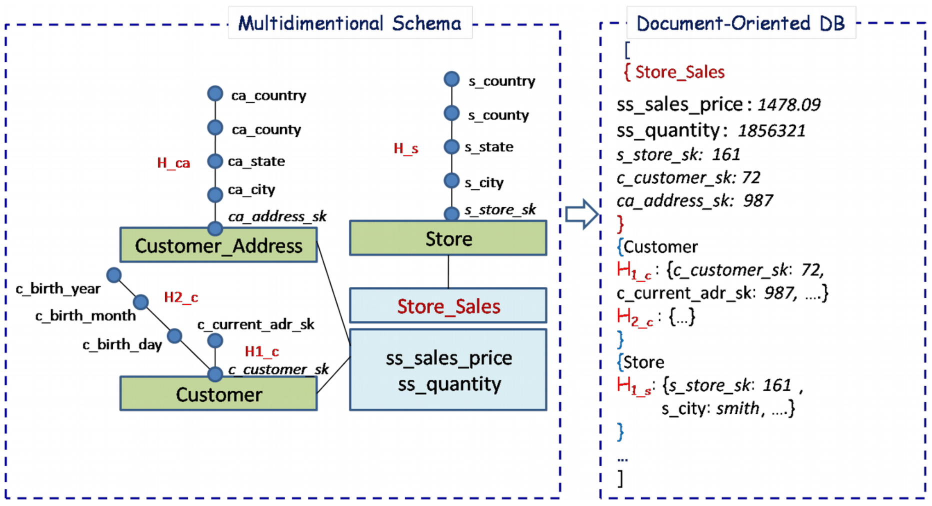


Fig.2. MS Transformation Example (TCP-DS example).

In this level, the correspondence table is generated to keep trace of different transformations. Table 1 presents an excerpt of the generated correspondence table (CT).

Table 1. Correspondence Table excerpt.

Object Source	Type	Operation	Target	Type Data
Store_Sales	Fact	Fact Transf	Store_Sales	Document
ss_sales_price	Measure	Fact Transf	ss_sales_price	Simple Attribute
Customer	Dimension	Dimension Tranf	Customer	Document
H1_c	Hierarchy	Dimension Transf	H1_c	Composed Attribute

c_customer_sk	Week Attribute	Dimension Transf	c_customer_sk	Simple Attribute
c_current_adr_sk	Strong Attribute	Dimension Transf	c_current_adr_sk	Simple Attribute

As output of this level, we have the correspondence table with full documentation of all transformation operations. This table will be used for modeling and implementing the transformation rules within ETL. In fact, to feed the NoSQL DW, data must be identified and extracted from the source. Consequently, the data must be transformed and verified before being loaded into "MongoDB". Moreover, loading data require the structure of the target to be known at advances. As NoSQL DBs are schema-less, this increases the need for extending the existing ETL tool in order to be able to designing data warehouse while integrating data. ETL tool should be adapted with the constant changes, to produce and to modify executable code quickly. This process will be addressed in the next section.

5 Extract, Transform and Load (ETL) Process

Business Intelligence (BI) solutions are very important as they require the implementation and the design of complex ETL process. In fact, ETL plays an important role in data warehousing architecture since these ETL processes move the data from transactional or sources systems to data staging areas and from staging areas into the data warehouse. ETL is often a complex combination of process and technology that consumes a significant portion of the data warehouse development efforts and requires the skills of business analysts, database designers, and application developers. New applications, such as, NoSQL data warehousing, require agile and flexible tools. In fact, the free-schema feature of these DBs imposes new requirements on the ETL process implementation and maintenance.

To facilitate and minimize the complexity of ETL process, we propose to model it using BPMN (Business Process Modeling Notation) language. BPMN notation seems to be a good choice since it can cover a deficit of communication that often occurs between the design and implementation of business process.

5.1 BPMN Modeling of ETL Process

In the last years, several modeling methodologies for ETL scenarios have been proposed, covering both conceptual and logical level. The chosen model is based on the Business Process Modeling Notation (BPMN), a standard for specifying business processes. BPMN provides a conceptual and implementation-independent specification of such processes, which hides technical details and allows users and designers to focus on essential characteristics of such processes. An advantage of BPMN is that it helps in graphically portraying also complex processes. Another benefit is the empowerment of business process workflow with execution details, thus providing a mapping between the graphics of the notation and the underlying constructs of execution languages.

Several works have dealt with ETL process modeling using BPMN such as [12] and [13]. In this section, we are based on the above works to model ETL process using BPMN (Fig.3).

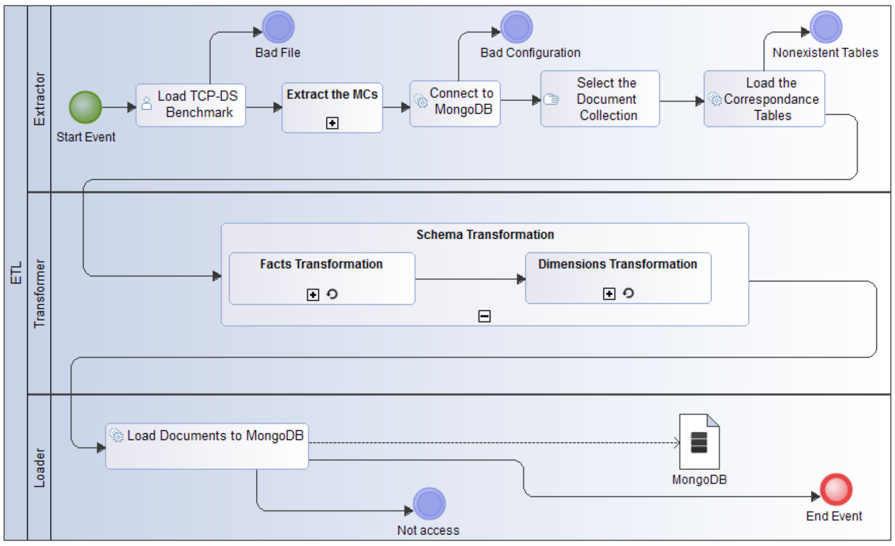


Fig.3. BPMN modeling of ETL process.

The BPMN model presented in Fig.3 is composed of one pool which encloses three lanes.

The first lane (Extractor) models the ETL extraction step. This step is responsible for extracting data from the source systems (TCP-DS Benchmark) and the configuration of the target system ("MongoDB"). The Extraction component must solve the problem of format heterogeneity, since data can be found in a variety of formats. Each data source has its distinct set of characteristics that need to be managed in order to effectively extract data for the ETL process. During the loading of the sources, there may occur "Throw" events, which will be fired whenever they are some inconsistent files.

The Second lane (Transformer) models the ETL transformation step. This step tends to make some conforming on the incoming data to obtain the needed system. This process includes schema transformation and data integration. It defines the granularity of the target schema. All transformation rules are implemented at this step using java routines. The Transformer lane has one sub-process which is composed of two others, representing the fact transformation and the dimension transformation.

Iteratively, the first sub-process handle all the records contained in the fact table and, one by one, invokes other two sub-processes that is in charge to select the measures and the identifiers of related dimensions (Fig.4). Once guaranteed the integrity of the transformation, the process updates the documents.

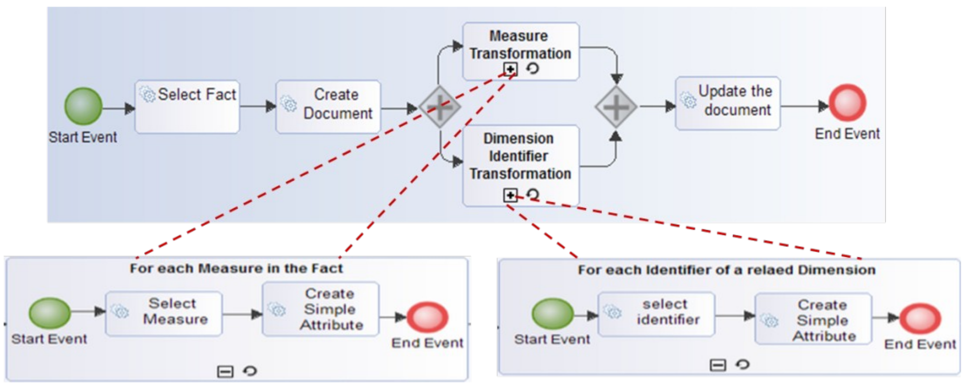


Fig.4. BPMN Representation of Fact Transformation.

The second sub-process contained in the Transformer lane is Dimension Transformer (Fig.5). Iteratively, this sub-process handle all the records contained in the dimension table and, one by one, invokes another sub-processes that is in charge to create a composed attribute for each hierarchy. This sub-process contains, in turn, two others sub process that describes the week and the strong attributes transformations.

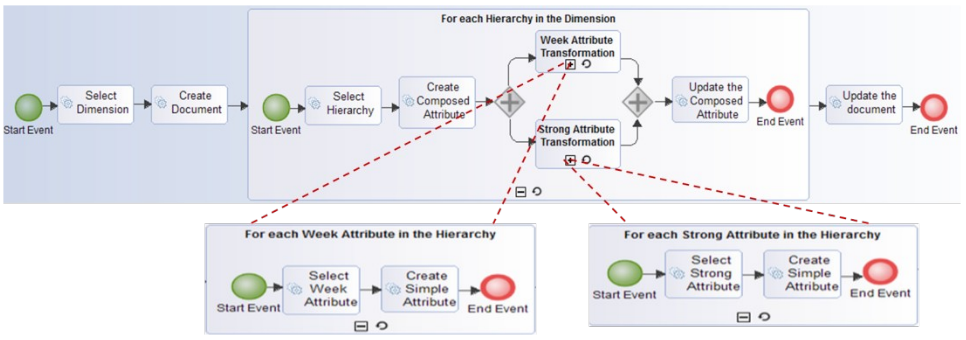


Fig.5. BPMN Representation of Dimension Transformation.

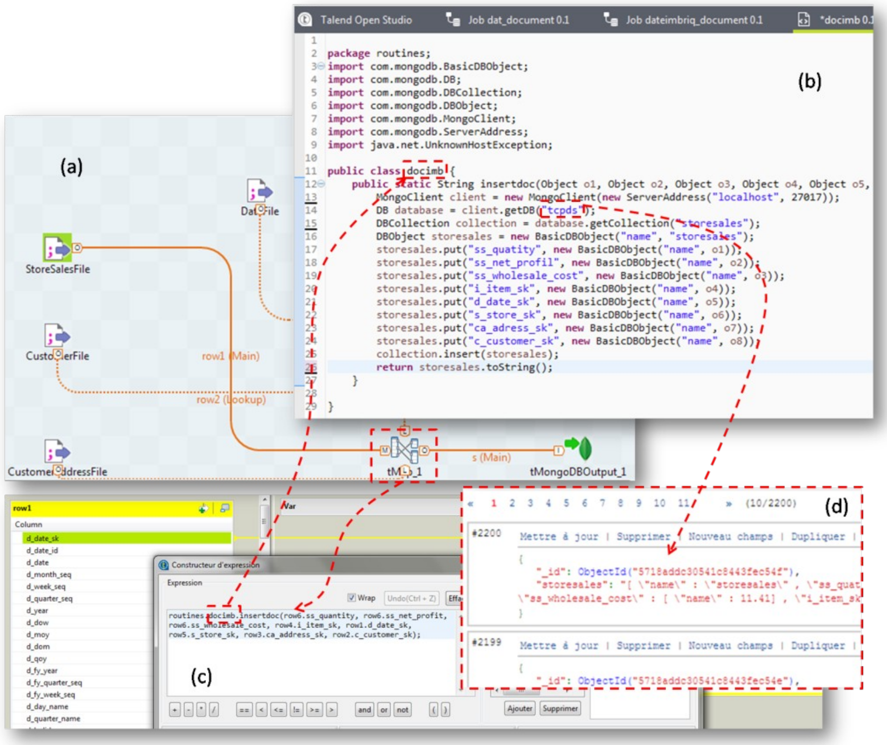
The third lane (Loader) models the ETL loader step. The Loader mechanism loads data into the target of an ETL process such as documents in “MongoDB”. Loading data to the target NoSQL DB is the final ETL step. In this step, extracted and transformed data is written into the document-oriented structures actually accessed by the end users and application systems.

It can be seen that, most tasks in the presented ETL process are automatic of type service task, and only a few remain of type user or manual task. This is due to the focus of the process in implementing the transformation rules in order to automatically generate the NoSQL DW.

5.2 Implementing the Transformation Rules

Data migration to NoSQL database that is a schema-free becomes a primary issue to many companies with multiple types of applications in e-commerce, business intelligence and politics. In fact, schema-less databases need careful migration and more programming efforts. The schema-less nature of the document-oriented database means that the designer can store documents in any shape but the notion of schema itself does not disappear from the model. Therefore, to create a NoSQL DW, we are required to write a program that relies on some form of implicit schema.

In this work, schema migration to document-oriented model is done according to the proposed transformation rules. These rules are implemented using java routines integrated with data integration tool "Talend for Big Data" (Fig.6).



```
package routines;
import com.mongodb.BasicDBObject;
import com.mongodb.DB;
import com.mongodb.DBCollection;
import com.mongodb.DBObject;
import com.mongodb.MongoClient;
import com.mongodb.ServerAddress;
import java.net.UnknownHostException;

public class docimb {
    public static String insertdoc(Object o1, Object o2, Object o3, Object o4, Object o5,
        MongoClient client = new MongoClient(new ServerAddress("localhost", 27017));
        DB database = client.getDB("tcpds");
        DBCollection collection = database.getCollection("storesales");
        DBObject storesales = new BasicDBObject("name", "storesales");
        storesales.put("ss_quatity", new BasicDBObject("name", o1));
        storesales.put("ss_net_profil", new BasicDBObject("name", o2));
        storesales.put("ss_wholesale_cost", new BasicDBObject("name", o3));
        storesales.put("i_item_sk", new BasicDBObject("name", o4));
        storesales.put("d_date_sk", new BasicDBObject("name", o5));
        storesales.put("s_store_sk", new BasicDBObject("name", o6));
        storesales.put("ca_adress_sk", new BasicDBObject("name", o7));
        storesales.put("c_customer_sk", new BasicDBObject("name", o8));
        collection.insert(storesales);
        return storesales.toString();
    }

}
```

```
routines.docimb.insertdoc(row6.ss_quantity, row6.ss_net_profit,
row6.ss_wholesale_cost, row4.i_item_sk, row1.d_date_sk,
row5.s_store_sk, row3.ca_address_sk, row2.c_customer_sk);
```

```
«  1  2  3  4  5  6  7  8  9  10  11      »  (10/2200)

#2200   Mettre à jour | Supprimer | Nouveau champs | Dupliquer |
        {
            "_id": ObjectId("5718addc30541c8443fec54f"),
            "storesales": "[ \"name\" : \"storesales\" , \"ss_quat
        \"ss_wholesale_cost\" : [ \"name\" : 11.41] , \"i_item_sk
        }

#2199   Mettre à jour | Supprimer | Nouveau champs | Dupliquer |
        {
            "_id": ObjectId("5718addc30541c8443fec54e"),
```

Fig.6. Example of routine implemented with Talend.

Fig.6 shows four main interfaces: (a) the created Job, (b) an excerpt of the implemented java routine, (c) the expression editor, and (d) an excerpt of the resulting "MongoDB" database.

A Job is a graphical design, of one or more components connected together such as tFileInputDelimited (CustomerFile, StoreSalesFile, etc.), tMap and FileOutputDelimited (“MongoDB”). Otherwise, a routine is a complex Java code, generally used to optimize data processing and improve Job capacities. In this work, the routine is used for implementing the transformation rules. The implemented routine is then called and edited in the expression editor. This editor provides visual interface to write any function or transformation in a handy dedicated view.

6 Evaluation

For implementing the document-oriented DW, our choice is oriented towards MongoDB. Like any other NoSQL database, MongoDB can quickly handle large volumes of data and offers the ability to create flexible schema. Our choice is justified by the fact that MongoDB provides a rich document oriented structure with dynamic queries.

In our context, we use TPC-DS benchmark to load the NoSQL DW. As TPC-DS is a logical model, we generated the conceptual schema, sand then we used our proposed rules to obtain the corresponding NoSQL models.

In order to evaluate the implemented NoSQL DW, we chose to use two metrics: “Write Request Latency” and “Read request Latency”. The first metric has the purpose to test the speed of the system during the data loading stage. As for the second metric, it evaluates the system’s ability to respond quickly to user requests. Regarding requests, we chose to use two. The first category is simple. As for the second query, it is more and consists on using some operators.

Table 2. Request description.

Request	SQL Langage	MongoDB Langage
Q1	Select * From date where d_year BETWEEN ‘1990’ and ‘2000’;	> db.date.find({"d_year" : {"$gte" : 1990, "$lte" : 2000} })
Q2	SELECT SUM(ss_quantity) FROM store_sales WHERE d_moy IN(‘2415022’, ‘2415023’) GROUP BY d_date_sk;	> db.store_sales.group({ > "key": {"d_date_sk": true}, > "initial": {" sumquantity ": 0}, > "reduce": function(obj, prev) {prev.sumquantity = prev.sumquantity + obj.ss_quantity ; }, > "cond": { "d_date_sk": { "$in": [2415022, 2415023]} } > });

Table.3 describes the system behavior against requests.

Table 3. Obtained results.

Number Records	Request	Returned Values Number	WRL (s)	RRL (s)
400000	Q1	363	0.20	0.19
	Q2	2415022: 326 2415023: 129		0.21
800000	Q1	1245	0.38	0.24
	Q2	2415022: 793		0.32

1000000	Q1	2415023: 513 3652	0.43	0.35
	Q2	2415022 875 2415023:614		0.46

As can be seen from the above results, NoSQL data warehouse can respond to increasing queries without performance degradation (0.35 for Q1 and 0.46 for Q2). A NoSQL data warehouse can rapidly adapt to growth in data volume and query intensity without degrading performance (0.21 for 400000 records while 0.46 for 1000000 records using the same query Q2). Moreover, the NoSQL data warehouse can quickly adapt to changes in data structure and content without requiring any schema redesign, additional data migration, or new data storage structures.

Additionally, the schema-less nature of the NoSQL system eliminates the need of transformation of schema between various data sources and the data warehouse. This transformation was just a waste of time in traditional data warehouse.

7 Conclusion

Because of the wide development of the social media, a huge amount of data is now continuously available to decision support.
Since the relational systems are lack of scaling and inefficient of handling big data it's vital to extract transform and loading the data from traditional data warehouses to NoSQL ones. Some approaches have been introduced to handle this problem however the ETL process, which is an essential stage in the construction of data warehouse, was neglected. ETL processes are very important problem in the current research of data warehousing. The schema-free feature of NoSQL systems increases the need for extending the existing ETL tool in order to be able to designing schema while integrating data.

In this paper, we proposed an ETL-based framework for NoSQL warehousing. In this context, we proposed rules for transforming a multidimensional conceptual schema into document-oriented system. As NoSQL databases are schema-less, the creation of a schema occurs while inserting data at ETL level. We used the BPMN for modeling the ETL process then we implemented the defined rules as routines that reflect the implicit schema. Experiments are carried out using the benchmark TPC-DS. The obtained results showed that a NoSQL data warehouse can rapidly adapt to growth in data volume and query intensity without degrading performance.

In the future work to this paper, we aim implementing OLAP operators (D-Roll up, D-Slice, D-Dice) with the phases of the invisible join technique for a document-oriented data warehouse.

References

1. Favre, C., Bentayeb, F., Boussaid, O., Darmont, J., Gavin, G., Harbi, N., Kabachi, N., Loudcher, S.: Les entrepots de donnees pour les nuls .ou pas. 2eme Atelier d'aide à la Decision a tous les Etages (2013).
2. Chandwani, G.: NOSQL DATA-WAREHOUSE. International Journal of Innovative Research in Computer and Communication Engineering, pp. 96-104 (2016).
3. Zhao, H., & Ye, X.: A practice of tpc-ds multidimensional implementation on nosql database systems. *Proceedings of the 5th International Conference on Performance Characterization and Benchmarking*, pp. 93–108 (2014).
4. Dehdouh, K., Boussaid, O., & Bentayeb, F.: Columnar NoSQL Star Schema Benchmark. Proceedings of the 6th International Conference on Model and Data Engineering. pp. 281-288 (2014).
5. Dehdouh, K., Boussaid, O., Bentayeb, F.: Using the column oriented NoSQL model for implementing big data warehouses. Proceedings of the 21st International Conference on Parallel and Distributed Processing Techniques and Applications, pp. 469-475 (2015).
6. Chevalier, M., El Malki, M., Kopliku, A., Teste, O., & Tournier, R.: Implementing Multidimensional Data Warehouses into NoSQL. Proceedings of the 17th International Conference on Enterprise Information Systems, pp. 172-183(2015).
7. Chevalier, M., El Malki, M., Kopliku, A., Teste, O., & Tournier, R.: Document-oriented Models for Data Warehouses. Proceedings of the 18th International Conference on Enterprise Information Systems, pp. 142-149 (2016).
8. Santos, M.Y., Martinho, B. & Costa, C.: Modelling and implementing big data warehouses for decision support. *Journal of Management Analytics*, pp. 1-19, (2017).
9. Bicevska,Z., Oditis, I.: Towards NoSQL-based Data Warehouse Solutions. Procedia Computer Science, Volume 104, pp. 104-111 (2017).
10. Wilkinson, K., Simitsis, A., Castellanos, M., & Dayal, U.: Leveraging Business Process Models for ETL Design. *Lecture Notes in Computer Science*, pp 15-30 (2010).
11. El Akkaoui, Z., Mazon, J., Vaisman, A., & Zimanyi, E.: Defining ETL worfklows using BPMN and BPEL. Data Warehousing and OLAP, pp 41-48 (2009).
12. El Akkaoui, Z., Mazon, J., Vaisman, A., & Zimanyi, E.: BPMN-Based Conceptual Modeling of ETL Processes. Proceedings of the 14th International Conference on Data Warehousing and Knowledge Discovery, pp. 1-14 (2012).
13. Oliveira, B., & Belo, O.: BPMN Patterns for ETL Conceptual Modelling and Validation. *International Syposium on Methodologies for Intelligent Systems*, pp. 445-454 (2012).
14. Delgado, A., Marotta, A., González, L.: Towards the construction of quality-aware Web Warehouses with BPMN 2.0 Business Processes. *RCIS*. Pp. 1-6 (2014).
15. Marotta, A., Delgado, A.: Data Quality Management in Web Warehouses using BPM. *ICIQ*, pp. 18-27 (2016).
16. Sahiet, D., Asanka, P. D.: ETL FRAMEWORK DESIGN FOR NOSQL DATABASES IN DATAWARE HOUSING. *International journal of research in computer applications and robotics, Vol.3,* pp. 67-75 (2015).
17. Sadalage, P. J., & Fowler, M.: NoSQL distilled: a brief guide to the emerging world of polyglot persistence. *Pearson Education* (2012).
18. Chaudhuri, S., Dayal, U., & Ganti, V.: Database technology for decision support systems. IEEE Computer Society, pp.48-55 (2002).