Automatic Transformation of Data Warehouse Schema To NoSQL Data Base: Comparative Study

Rania Yangui\textsuperscript{a}, Ahlem Nabli\textsuperscript{b}, Faiez Gargouri\textsuperscript{a}

\textsuperscript{a}MIRACL Laboratory, Institute of Computer Science and Multimedia Sfax, BP 1030, Tunisia
\textsuperscript{b}MIRACL Laboratory, Faculty of Computer Science and Information Technology, Al-Baha University, KSA

Abstract

Driven by the ever-growing of data from social network (SN), data warehouse (DW) approaches must be adapted. Generally the star, snowflake or constellation models are used as logical ones. All these models are inadequate when dealing with social data which need scalable and flexible systems. As an alternative, NoSQL systems begin to grow.

In the absence of a clear approach which allows the implementation of data warehouse under NoSQL model, we propose in this paper, new rules for transforming a multidimensional conceptual model into two NoSQL ones: column-oriented and document-oriented models. For each model, we distinguish two types of transformation: simple and hierarchical. To validate our transformation rules, we implemented four data warehouses using Cassandra as a column-oriented NoSQL system and MongoDB as document-oriented NoSQL system. These systems were implemented using java routines in Talend Data Integration tool and evaluated in terms of "Write Request Latency" and "Read request Latency" using TPC-DS benchmark.

© 2016 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

Keywords: NoSQL Data Warehouse, Social Networks, Column-oriented model, Document-oriented model;

* Corresponding author. Tel.: +216 20004271.
E-mail address: yangui.ania@gmail.com

1877-0509 © 2016 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).
Peer-review under responsibility of KES International
doi:10.1016/j.procs.2016.08.138
1. Introduction

Because of the huge amount of data available on the Web, integrating external data from social network with a company's internal data in a data warehouse is a promising approach. This latter broaches a paramount importance and draws the attention of many researches. However, as argued by many works\textsuperscript{12}, current warehousing methodolgies with relational databases cannot be successfully applied to handle the growing complexity and volume of data generated from SN. The rules that were designed for relational data cannot be enforced on SNs generated data. In fact, many new technologies have emerged such as NoSQL databases in order to increase the performance and the availability of services\textsuperscript{5}. NoSQL is a term often used to describe a class of non-relational databases that scale horizontally to very large data sets.

Given that a well-designed DW requires a carefully planned logical design, all updates and versions of a DW lead to a revision of the logical design\textsuperscript{6}. 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)\textsuperscript{9}. All these approaches are inadequate when dealing with SN generated data. To solve a part of this issue, some works have followed the NoSQL model.

However, the choice of NoSQL database type with the best transformation is a key problem for the implementation of a NoSQL DW. Decision support systems must not only have large storage capacity, but also must be able to respond properly and promptly to the queries of decision makers. Thus the ability to respond to relatively complex queries and response time are main criteria in the choice of the NoSQL model.

In order to benefit from the NoSQL model advantages, and in the absence of a clear approach allowing the implementation of the NoSQL DW, the objective of this paper is to determine the appropriate NoSQL model that will be used when creating DW from SN. For that, this paper deals with a set of transformation rules from conceptual model to NoSQL ones. These rules are implemented using java routines in Talend Data Integration tool and 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 state of art. Section 3 gives a formal representation of the multidimensional schema. Section 4 presents a formal representation of the NoSQL Columnar and Document-oriented Databases. Section 5 details the transformation rules. Section 6 represents the evaluation tasks. Section 7 concludes the paper and draws future research directions.

2. Related Work on NoSQL data warehousing

NoSQL systems have shown their advantages over relational systems in terms of flexibility and handling massive data. In the literature, a number of researchers have recognized the deficiencies of the traditional OLAP data model and have proposed approaches for the migration from relational databases to NoSQL ones (indirect approaches). However, few works have focused on the transformation of the multidimensional conceptual model to NoSQL logical one (direct approaches). The first type of transformation (indirect) is provided in two stages. The first stage\textsuperscript{11, 12} is to use a set of transition rules from multidimensional conceptual level to relational logical one. The second stage\textsuperscript{13, 14, 15} finds the correspondences between the obtained relational model and the NoSQL target system. In\textsuperscript{13} the authors propose rules allowing the storage of a time dimension in HBase table. In\textsuperscript{14}, the authors propose a set of transformation rules for translating a relational model to column-oriented model via HBase. As an attempt to migrate to a document-oriented NoSQL model, the authors have focused in\textsuperscript{15} on the system performance study when using MongoDB as NoSQL system. However, this work has focused only on testing the performance without specifying the rules of transforming the logical model to document-oriented one.

Indirect Migration approaches have shown that it is possible to migrate from a relational database to NoSQL one. However, this first generation of solutions have certain limitations. First, these approaches are limited to converting a logical representation to another logical one and does not consider the conceptual model of data warehouses. Additionally, at the beginning, all data should be stored in a relational database. This latter showed these limits during the storage of large volumes of data.

For the direct approaches, we can cite few recent works that are aimed at developing data warehouses in NoSQL systems. In\textsuperscript{16}, the authors have developed a new benchmark for the columnar NoSQL DW, but without giving the formalization for the modelling process. This work is considered as the first work which proposes implemented star DW under column oriented NoSQL DBMS directly from dimensional model. This work is extended by proposing
three approaches which allow big data warehouses to be implemented under the column oriented NoSQL model. Each approach differs in terms of structure and the attribute types used when mapping the conceptual model into logical model is performed. Other recent works, have tried to define logical models for NoSQL data stores (oriented columns and oriented documents). The authors have proposed a set of rules to map star schema into two NoSQL models: column-oriented (using HBase) and document-oriented (using MongoDB).

Table 1 highlights a summary of the literature review based on seven criteria.

<table>
<thead>
<tr>
<th>Work</th>
<th>NoSQL DB</th>
<th>Source</th>
<th>Source Model</th>
<th>Transformation Type</th>
<th>Formalization</th>
<th>Evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Han et al., 2012)</td>
<td>HBase</td>
<td>DW</td>
<td>Logical</td>
<td>Indirect, Simple</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>(Li et al., 2010)</td>
<td>HBase</td>
<td>DB</td>
<td>Logical</td>
<td>Indirect</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>(Chevalier et al., 2015)</td>
<td>HBase/MongoDB</td>
<td>DW</td>
<td>Conceptual</td>
<td>Direct, Simple</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>(Dehdouh et al., 2015)</td>
<td>HBase</td>
<td>DW</td>
<td>Conceptual</td>
<td>Direct, Simple</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

All the mentioned works present several interesting mining. They have shown that it is possible to convert a multidimensional conceptual model to a NoSQL storage. However, we note that the concepts of the conceptual multidimensional model were never considered in the context of indirect transformations. Also, direct approaches have focused only on the rules for transforming the concepts of fact and dimension in a NoSQL structure. Furthermore, 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. Nevertheless, during the transformation of a DW to a NO SQL DB, we confront several possible transformations that we need to evaluate. Moreover, we emphasize there’s two interesting concepts that have never been used in the proposed transformation, namely embedded documents and super columns. In our work, we have used these concepts for detailing hierarchies. To achieve that, we start with formally describing the source (multidimensional model) and the target (columnar and documents-oriented models).

3. Formal Representation of the Multi-Dimensional Schema

Conceptual modeling is the necessary foundation phase. In this phase, DW are modeled in a multidimensional way. The basic concepts of multidimensional modeling are: facts, measures, dimensions and hierarchies. According to , we formalize it as follows:

**Fact and measures.** A fact represents the analyzed subject. It is composed of measures reflecting the information to be analyzed. The measures of a fact are generally numeric and continuously valued to summarize a large number of records. Formally, a fact is a pair \((F^N, F^M)\) where:

- \(F^N\) is the name of the fact;
- \(F^M\) is a set of attributes forming the Fact measures: \(F^M = \{M_1, \ldots, M_n\}\).

Each measure \(M\) is defined by: \(M = (M^N, M^{F_{\text{Fonc}}})\) where:

- \(M^N\) is the name of the measure;
- \(M^{F_{\text{Fonc}}}\) is an aggregate function (sum, average, etc.).

**Dimension and Hierarchy.** The dimensions represent the axes of the multidimensional analysis. They are usually textual and discreet. They are used to restrict the scope of queries to limit the size of the responses. Formally, a dimension is a triplet \((D^N, D^{\text{Att}}, D^{\text{Hier}})\) where:

- \(D^N\) is the name of the dimension;
- \(D^{\text{Att}} = \{A_1, \ldots, A_m\}\) is the set of strong and weak attributes of a dimension;
- \(D^{\text{Hier}} = \{H_1, \ldots, H_k\}\) is a set of hierarchies.

The attributes of the dimensions are organized into one or more hierarchies. A hierarchy is composed of several levels, representing different degrees of information accuracy. A hierarchy of a dimension \(H\) is a path defined by \((H^N, H^P, p^{F_{\text{Att}}})\) where:
• $H_N$ is the name of the hierarchy;
• $H_P = \langle P_1, \ldots, P_n \rangle$ is an ordered list of the strong attributes used in the hierarchy;
• $p^W$ is a function that associates each strong attribute its weak attributes.

**Multidimensional Schema.** A DW is characterized by its multidimensional schema (MS) composed of a fact schema with a single or a constellation of facts and dimensions. Formally, a multidimensional schema MS is defined by $(MS^N, MS^{Fact}, MS^{Dim}, Func)$ where:

- $MS^N$ is the name of the MS;
- $MS^{Fact} = \{F_1, \ldots, F_l\}$ is a set of facts;
- $MS^{Dim} = \{D_1, \ldots, D_q\}$ is a set of dimensions;
- Func a function that associates to a fact $F_i$ the list of its dimensions with $\forall i \in [1, \ldots, l]$, Func $(F_i) = \{D_{y_1}, \ldots, D_{y_x}\}$ such that $\forall j \in [v, \ldots, x]$, $\exists D_j \in MS^{Dim}$ and $\forall D_f \in MS^{Dim}$, $\exists F_j \in MS^{Dim}$ such that $D_f \in Func (F_i)$.

4. Formal Representation of Column-oriented and Document-oriented Databases

NoSQL databases are becoming increasingly popular and have many interesting strengths such as scalability and flexibility. In order to define the rules that cover the mapping process from MS to the target NoSQL logical ones, we begin by formalizing the different concepts of column-oriented and document-oriented DBs.

4.1. Column-oriented Database

A column-oriented database can be seen as a complex data structure with five main components: KeySpace, Columns, Super-Columns, Column-Families, and Super-Column-Families.

**Column.** A column is the smallest unit in the column-oriented databases. A column is identified by a name and a value defined by the user as well as a timestamp that indicates the last date at which the data was changed. Formally, a column is a triple $(C_N, C_{Val}, C_{Ts})$ where:

- $C_N$ is the name of the column;
- $C_{Val}$ is the value of the column;
- $C_{Ts}$ is the timestamp that indicates the last date on which the data has changed.

**Super-Column.** A super-column is a structure that has a name and that has as value an infinite number of related columns. Formally, a super-column SC is a pair $(SC_N, SC_{Val})$ where:

- $SC_N$ is the name of the super-column;
- $SC_{Val} = \{C_1, \ldots, C_n\}$ is a set of columns that constitutes the super-column / $\forall i \in [1, \ldots, n], C_i \in C$.

**Column-Family.** A column-family is the main component in the column-oriented database that can be assimilated to a table in relational database. Linked columns (those accessed simultaneously) must be grouped in the same column family. The column-families can be created dynamically and their number is not limited. Formally a column-family CF is defined by $(CF_N, CF_{Val})$ where:

- $CF_N$ is the name of the column-family;
- $CF_{Val} = \{L_1, \ldots, L_n\}$ is a set of lines that constitute the column-family such as each line $L_i$ is defined by $(L_{i_{Key}}, L_{i_{Val}})$ where:
  - $L_{i_{Key}}$ is a key identifying the line $L_i$;
  - $L_{i_{Val}} = \{C_1, \ldots, C_n\}$ is a set of columns that constitute the line / $\forall j \in [1, \ldots, n], C_j \in C$.

**Super-Column-Family.** A super-column-family is a NoSQL object that contains Lines composed of super-columns. It can be seen as a map of tables. The flexibility of this model is that we can represent relationships and hierarchies in a simple and flexible manner. Formally a super-column-family SCF is defined by $(SCF_N, SCF_{Val})$ where:

- $SCF_N$ is the name of the super-column-family;
The above formal descriptions will be used to facilitate the definition and the automation of the transformation rules from conceptual model to NoSQL ones.

5. Transformation Rules

The aim of this section is to propose transformation rules to NoSQL DWs. Recall that a DW 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 fact according to its measures. As, several alternatives are possible, we will detail some of these alternatives in order to choose the best. We define two types of transformations namely: simple and hierarchical transformations. The first one proposes storing the fact and dimensions into column-families/collections, and uses only the simple columns/documents for representing measure and dimension attributes. The second transformation uses also
different column-families/collections for storing facts and dimensions, and uses the simple columns/documents for representing measure and the super-columns/composed attributes for representing dimension attributes while explaining hierarchies. For each type of transformation, we define a set of rules ensuring the mapping from MS concepts to column-oriented and document-oriented models.

5.1. Simple Transformation

The simple transformation is the mapping to NoSQL model while highlighting the concepts of the MS but without detailing the hierarchies. Otherwise, this transformation distinguishes between measures of fact and attributes of dimension. The dimensions and the facts are stored separately on different column-families/collections. To ensure the links between these two entities (dimension-fact), the dimension identifier is duplicated in column-family/Collections representing the fact. In the following, we present the ST-C and ST-D, rules for Simple Transformation to Column-oriented and Document-oriented DWs.

**Rule1: Simple Transformation to Column-oriented model (ST-C).** Each multidimensional schema MS ($MS^N$, $MS^{Fact}$, $MS^{Dim}$, $Func$) is transformed to a keyspace KS ($KS^N$, $KS^{Val}$) where:

- The name of the keyspace is the name of the MS / $KS^N \leftarrow MS^N$;
- Each fact $F \in MS^{Fact}$ is transformed to a column-family $CF(CF^N, CF^{Val})$ where :
  - The name of the column-family is the name of the fact / $CF^N \leftarrow FN$;
  - Each measure $M \in F$ is transformed to a column $C \in CF^{Val} / C^N \leftarrow M$;
  - Each identifier of the related dimensions is transformed to a column $C \in CF^{Val} / C^N \leftarrow D^id$;
- Each dimension ($DN$, $D^{Att}$, $D^{Hier}$) / $D \in MS^{Dim}$ is transformed to column-family $CF(CF^N, CF^{Val})$ where:
  - The name of the column-family / $CF^N$ is equivalent to the name of the dimension $CF^N \leftarrow D^N$;
  - Each attribute $A \in D^{Att}$ is transformed to a column $C \in CF^{Val}$ where the name of the column is the attribute name / $C^N \leftarrow A$.

The overall simple transformation process from DW conceptual schema to column-oriented model is presented in (Fig 1.A).

**Rule2: Simple Transformation to Document-oriented model (ST-D).** 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 / $DC^N \leftarrow FN$;
  - Each measure $M \in F$ is transformed to a simple attribute $SA \in DC^{Att} / SA^N \leftarrow M$;
  - Each identifier of the related dimensions is transformed to a simple attribute $SA \in DC^{Att} / SA^N \leftarrow D^id$;
- Each dimension ($DN$, $D^{Att}$, $D^{Hier}$) / $D \in MS^{Dim}$ is transformed to a document DC($DC^N$, $DC^{Att}$) where:
  - The name of document is equivalent to the name of the dimension / $DC^N \leftarrow D^N$ ;
  - Each attribute $A \in D^{Att}$ is transformed to a simple attribute $SA \in DC^{Att}$ where the name of the simple attribute is the attribute name / $SA^N \leftarrow A$.

The overall simple transformation process from DW conceptual schema to document-oriented model is presented in (Fig 1.B).

---

**Fig. 1. ST examples.**
5.2. Hierarchical transformation

We are looking in this section to transform the multidimensional model of DW to a NoSQL model while explaining hierarchies. It consists to explore the concepts of super-column and super-column-family, respectively embedded document, to burst the dimensions of a multidimensional diagram.

**Rule 3: Hierarchical Transformation to Column-oriented model (HT-C)** Each multidimensional schema MS \((MS_N, MS_{Fact}, MS_{Dim}, Func)\) is transformed to a keyspace \(KS(KS_N, KS_{Val})\) where:

- The name of the keyspace is the name of the MS /\(KS_N \leftarrow MS_N\);
- Each fact \(F \in MS_{Fact}\) is transformed to a column-family \(CF(CF_N, CF_{Val})\) where:
  - The name of the column-family is the name of the Fact /\(CF_N \leftarrow FN\);
  - Each measure \(M \in F\) is transformed to a column \(C \in CF_{Val} /C_N \leftarrow M\);
  - Each identifier of a related dimension is transformed to a column \(C \in CF_{Val} /C_N \leftarrow D_id\);
- Each dimension \((DN, DA^t, DH^r) /D \in MS_{Dim}\) is transformed to super-column-family \(SCF(SCF_N, SCF_{Val})\) where:
  - The name of the super-column-family /\(SCF_N\) is equivalent to the name of the dimension \(DN \leftarrow DN\);
  - Each hierarchy \(H(H_N, HP, P_{Att})\) is transformed to a super-column \(SC(SC_N, SC_{Val})\) in the line \(M_i \in SC_{Val}\) where:
    * The name of the super-column is the hierarchy name /\(SC_N \leftarrow H_N\)
    * Each attribute (strong and week) \(P_j\) becomes column in \(SC_{Val}\)

The hierarchical transformation of the multidimensional schema to the column-oriented model is illustrated by (Fig 2.A).

**Rule 4: Hierarchical Transformation to Document-oriented model (HT-D)**. 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 /\(DC_N \leftarrow FN\);
  - Each measure \(M \in F\) is transformed to a simple attribute \(SA \in DC_{Att} /SA_N \leftarrow M\);
  - Each identifier of a related dimensions is transformed to a simple attribute \(SA \in DC_{Att} /SA_N \leftarrow D_id\);
- Each dimension \((DN, DA^t, DH^r) /D \in MS_{Dim}\) is transformed to a document \(DC(DC_N, DC_{Att})\) where:
  - The name of the document \(D\) is equivalent to the name of the document /\(DC_N \leftarrow DN\)
  - Each hierarchy \(H(H_N, HP, P_{Att})\) is transformed to a composed attribute \(CA \in DC_{Att}\) 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.

The hierarchical transformation of the MS to the document-oriented model is illustrated by (Fig 2).

These proposed transformation rules allow transforming a multidimensional model of a DW to two NoSQL models: column-oriented and document-oriented. Using these rules, we can implement and analyze several decision systems. We begin by validating these transformations by the implementation of the decision systems concerning the simple
and hierarchical transformations under Cassandra and MongoDB systems. We then compare their performance on loading and query phases using the decision benchmark TPC-DS.

6. EXPERIMENTS

Extensive experiments were conducted to validate our proposals including the implementation of the transformation rules to map multidimensional model to NoSQL column-oriented and document-oriented ones.

6.1. NoSQL DW Implementation Environment

For implementing the column-oriented DW, our choice is oriented towards Cassandra. Like any other NoSQL database, Cassandra can quickly handle large volumes of data and offers the ability to create flexible schema. Our choice is justified by the fact that Cassandra is the only NoSQL database which includes the concepts of super-column and super-column-family. Otherwise, we used MongoDB to implement the document-oriented DW because it is the fastest-growing database. It provides a rich document oriented structure with dynamic queries. Also, MongoDB allows dividing data into collections.

In our context, we use TPC-DS benchmark to load the different NoSQL DWs. As TPC-DS is a logical model, we generated the conceptual schema, then we used our proposed rules to obtain the corresponding NoSQL models. An expert of the TCP-DS generated MS is depicted by Fig 3.

To load data in the NoSQL DWs, we chose to use the data integration tool "Talend for Big Data". This tool allows extracting data from large and heterogeneous data sources and integrates them into NoSQL database. In the context of our work, data integration is done according to our transformation rules. These rules are implemented using ETL routines in the same tool (Fig 4).
6.2. Discussion

In order to evaluate the implemented NoSQL DWs, 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 a set of six queries that we classified into two categories. The first category is simple and consists on increasing the number of dimensions and attributes to test the performance of the decision system with the presence of joins in the user’s queries. As for the second category, it is more complex (integrated with TPC-DS benchmark) and consists on using some operators. Table 2 describes the used queries.

Table 2. Requests description.

<table>
<thead>
<tr>
<th>Request</th>
<th>R. Category</th>
<th>Dimension(s)</th>
<th>Attribute(s)</th>
<th>Condition(s)</th>
<th>Group by</th>
<th>Order by</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>Simple</td>
<td>Date</td>
<td>d_year</td>
<td>d_year=&quot;1990&quot;</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Q2</td>
<td>Simple</td>
<td>Date, Item</td>
<td>d_year, i_category</td>
<td>d_year=&quot;2001&quot;</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Q3</td>
<td>Simple</td>
<td>Date, Item, Customer Address</td>
<td>d_year, i_category, ca_state</td>
<td>d_year=&quot;1999&quot;</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Q4</td>
<td>Simple</td>
<td>Item</td>
<td>i_category</td>
<td>i_category = &quot;Children&quot;</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Q5</td>
<td>Simple</td>
<td>Item</td>
<td>i_category, i_class</td>
<td>i_category = &quot;Men&quot;, i_class = &quot;pants&quot;</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Q6</td>
<td>Simple</td>
<td>Item</td>
<td>i_category, i_class, i_brand</td>
<td>i_category = &quot;Women&quot;, i_class = &quot;fragrances&quot;, i_brand = &quot;exportimporto&quot;</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Q7</td>
<td>Complex</td>
<td>Date, Store Sales, Item</td>
<td>d_year, i_brand, sum(ss_quantity)</td>
<td>d_moy=&quot;12&quot;</td>
<td>d_year, i_brand</td>
<td>d_year</td>
</tr>
<tr>
<td>Q8</td>
<td>Complex</td>
<td>Date, Store Sales, Item</td>
<td>i_brand, sum(ss_quantity)</td>
<td>d_moy=&quot;11&quot;, d_year=&quot;1999&quot;</td>
<td>i_brand</td>
<td>i_brand</td>
</tr>
<tr>
<td>Q9</td>
<td>Complex</td>
<td>Date, Store Sales, Item</td>
<td>d_year, i_category, sum(ss_quantity)</td>
<td>d=&quot;12&quot;, d_year=&quot;1998&quot;</td>
<td>d_year, i_category</td>
<td>i_category</td>
</tr>
<tr>
<td>Q10</td>
<td>Complex</td>
<td>Item, Store Sales, Store</td>
<td>d_year=&quot;1998&quot;</td>
<td>s_store_sk, s_country</td>
<td>s_store_sk</td>
<td>s_state</td>
</tr>
<tr>
<td>Q11</td>
<td>Complex</td>
<td>Date, Store Sales, Item</td>
<td>d_year=&quot;12&quot;, d_year=&quot;2000&quot;</td>
<td>d_year, i_brand</td>
<td>d_year</td>
<td>i_brand</td>
</tr>
<tr>
<td>Q12</td>
<td>Complex</td>
<td>Date, Store Sales, Item</td>
<td>d_year=&quot;12&quot;, d_year=&quot;2001&quot;</td>
<td>i_brand</td>
<td>i_brand</td>
<td></td>
</tr>
</tbody>
</table>

We started the evaluation of the DWs by comparing their data loading times Write Request Latency. 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). Thereafter we tested the behavior of the systems(ST-C, ST-D, HT-C and HT-D) against the queries sets (Read Request Latency)(Table 3).

Table 3. Read Request Latency (ms).

<table>
<thead>
<tr>
<th>Request</th>
<th>ST-C</th>
<th>ST-D</th>
<th>HT-C</th>
<th>HT-D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>0.2</td>
<td>0.05</td>
<td>0.182</td>
<td>0.02</td>
</tr>
<tr>
<td>Q2</td>
<td>0.975</td>
<td>0.173</td>
<td>0.51</td>
<td>0.16</td>
</tr>
<tr>
<td>Q3</td>
<td>1.41</td>
<td>0.80</td>
<td>1.62</td>
<td>1.13</td>
</tr>
<tr>
<td>Q4</td>
<td>0.18</td>
<td>0.02</td>
<td>0.12</td>
<td>0.012</td>
</tr>
<tr>
<td>Q5</td>
<td>1.62</td>
<td>1.14</td>
<td>1.83</td>
<td>1.125</td>
</tr>
<tr>
<td>Q6</td>
<td>2.3</td>
<td>2.125</td>
<td>2.33</td>
<td>2.108</td>
</tr>
<tr>
<td>Q7</td>
<td>3.192</td>
<td>2.24</td>
<td>2.981</td>
<td>2.03</td>
</tr>
<tr>
<td>Q8</td>
<td>2.96</td>
<td>2.01</td>
<td>2.36</td>
<td>1.94</td>
</tr>
<tr>
<td>Q9</td>
<td>4.02</td>
<td>3.12</td>
<td>3.23</td>
<td>3.09</td>
</tr>
<tr>
<td>Q10</td>
<td>2.14</td>
<td>1.82</td>
<td>1.98</td>
<td>1.66</td>
</tr>
<tr>
<td>Q11</td>
<td>3.23</td>
<td>2.67</td>
<td>2.87</td>
<td>2.53</td>
</tr>
<tr>
<td>Q12</td>
<td>3.124</td>
<td>3.03</td>
<td>2.74</td>
<td>2.59</td>
</tr>
</tbody>
</table>
Initial results show that the document-oriented NoSQL data warehouse is more efficient in terms of interrogation when dealing with two categories of queries. The best transformation is the hierarchical one. We observed also that the loading times is different for each system and gives advantage to column-oriented one. This is argued by the fact that Cassandra uses less memory space and it is known for effective data compression (due to column redundancy). A major difference between the different NoSQL systems concerns interrogation. For queries that demand multiple attributes, the column-oriented approaches might take longer because data will not be available in one place. This remains an advantage to MongoDB compared to Cassandra. Studying differences with respect to interrogation using complex queries (drill-down, roll-up, etc.) is listed for future work.

7. Conclusion and future works

In this paper, we have proposed transformation rules that ensure the successful translation from conceptual DW schema to two logical NoSQL models (column-oriented and document-oriented). We proposed two possible transformations namely: simple and hierarchical transformations. The first one stores the fact and dimensions into one column-family/collection. The second transformation uses different column-families/collections for storing fact and dimensions while explaining hierarchies. Experiments are carried out using the benchmark TPC-DS. Preliminary results show that MongoDB with hierarchical transformation is more suitable when dealing with OLAP queries. The used queries are qualified as simple, so in terms of perspectives we want to study the NoSQL systems with respect to more complex querying (roll-up, drill-down). We need to study different types of queries and identifying queries that benefit mostly from NoSQL models. We can also consider other NoSQL models (graph-oriented NoSQL model).

References