Document-Oriented Data Warehouses: Models and Extended Cuboids

Extended Cuboids in Oriented Document

Max Chavalier[1], Mohammed El Malki[1,2], Arlind Kopliku[1], Olivier Teste[1], Ronan Tournier[1]

[1] - University of Toulouse, IRIT (UMR 5505)
Toulouse, France
http://www.irit.fr
{First name.Last name}@irit.fr

[2] - Capgemini
109, avenue du General Eisenhower
BP 53655, F-31036 Toulouse, France
http://www.capgemini.com

***Abstract*— Within the Big Data trend, there is an increasing interest in Not-only-SQL systems (NoSQL). These systems are promising candidates for implementing data warehouses particularly due to the data structuration/storage possibilities they offer. In this paper, we investigate data warehouse instantiation using a document-oriented system (a special class of NoSQL systems). On the one hand, we analyze several issues including modeling, querying, loading data and OLAP cuboids. We compare document-oriented models (with and without normalization) to analogous relational database models. On the other hand, we suggest improvements in order to benefit from document-oriented features. We focus particularly on extended versions of OLAP cuboids that exploit nesting and arrays. They are shown to work better on workloads with drill-down queries. Research in this direction is new. As existing work focuses on feasibility issues, document-specific implementation features, modeling and cross-model comparison.**

Keywords*— *NoSQL, document-oriented system, big data warehouse, OLAP cuboid.

I. INTRODUCTION

The data volume managed by computer systems is in a continuous growth. Data management is becoming critical: huge and diverse amounts of data are to be stored and analyzed [16]. To ease data analysis, and decision making, it is common to centralize them in data warehouses [4],[19]. These latter are suitable for on-line analysis called OLAP (On-Line Analytical Processing): they support efficiently interactive exploration of data on different analysis dimensions and at different levels of detail [4],[9]. The most successful data warehousing implementations are primarily based on Relational Database Management Systems (RDBMS), called R-OLAP approaches.

In the recent year, important changes have affected the database domain mainly dictated by the growth of the Web. Major Web companies such as Google, Facebook, Twitter or Amazon had to face unprecedented amounts of data which is not necessarily in a relational format, well-structured and stable. Moreover, it is not convenient or possible to store all the data on one machine. Big Data is mainly concerned by data volume as well as data variety (heterogeneity) and velocity (real time processing). Relational databases cannot deal with all the big data issues [26],[29],[31] and a new class of data-store systems has met commercial success, namely NoSQL (Not Only SQL) systems [3].

In contrast to Relational Database Management Systems (RDBMS), NoSQL systems are famous for horizontal scaling, elasticity, availability and schema flexibility. There are 4 major classes of NoSQL systems: document-oriented, column-family, graph-oriented and key-value systems. Investigating new opportunities in NoSQL for data warehousing becomes interesting and some research work have already considered document-oriented [1]and column-oriented [5],[34] systems for this purpose. However, related work is relatively new [5],[6],[8],[10],[20],[34] and focuses rather on feasibility issues and technology specific implementations.

Instantiating a multidimensional data warehouse [4],[9] with this new technology is not an easy process. Data needs to be extracted and transformed in a model more suitable for document-oriented systems. OLAP queries must be rewritten in a technology specific language, their execution optimized and OLAP cuboids pre-computed for speeding them up. Most data warehouse instantiations with NoSQL systems are direct mappings of R-OLAP instantiations. However, we need to distinguish NoSQL logical models from relational data models (see Fig 1). We need an explicit document-oriented model formalization but as what works well with relational databases cannot be guaranteed to work well on document-oriented systems, we also need to investigate for NoSQL specific advantages for data warehousing. This paper investigates further the potential of document-oriented systems for multidimensional data warehousing.

Similarly to other NoSQL systems, document-oriented systems are known for schema flexibility, scalability and elasticity: handling heterogeneous data models; providing

richer structures (nesting, arrays, etc.) and offering options for data processing (e.g. map-reduce or aggregation pipelines).

In this context, we bring forward our existing work on document-oriented implementation of data warehouses [1]. We study two data models for this purpose and provide two OLAP cuboid extensions that can be implemented using documents. The paper contributions can be summarized as follows:

i) We instantiate multidimensional data warehouses in document-oriented systems using two different document models that are equivalent to normalized and denormalized data storage. It is known that some NoSQL systems work better with flat data (denormalized) in contrast to relational databases. We show the direct mapping from the multidimensional model to these models. Advantages of each model are shown by comparing different instantiations on different data warehouse features including: loading and querying data, as well as computing OLAP cuboids.

ii) Then, we propose and study **extended versions of OLAP cuboids** that use nesting and arrays. These offer fast drill-down capabilities and allow answering more queries. These types of cuboids have already been studied for data warehouses [17],[33], but they are not compatible with the relational data model. However in document-oriented systems, these cuboids can be stored using the document data model, i.e. no need for non-document features or specific extensions.

iii) We also compare our document-oriented models to classical relational models. This comparison is technology specific. Our goal is to illustrate current performance differences.

The paper is organized as follows. The next section details related work. In section III, we define our multidimensional data model. In section IV we define the extended OLAP cuboids. In section V, we detail and discuss experimental results. The last section concludes and lists future works.

II. Related Work

In the past recent years, an increasing interest has focused on NoSQL systems [12],[14],[26],[35]. They represent database systems that are alternatives to relational databases, offering interesting new features including new query languages (not only SQL), new data storage techniques and new data processing techniques. These different NoSQL solutions have been compared with each other in different settings [32]. They have also been compared to relational databases [14],[22]. In [29], the authors compare a document-oriented system (MongoDB) with a relational system (SQLServer) on OLTP[1] queries. In [14], the authors compare query execution using map-reduce on NoSQL systems with query execution on distributed RDBMS and identify the advantages on both architectures. Consequently, we find work on how to map data from relational databases to NoSQL systems and vice-versa[6],[8]. Recently, a new class of approaches is being studied called NewSQL systems [29], aiming to preserve relational database advantages while answering big data requirements: scalability, elasticity and flexibility.

Recently, NoSQL systems have been tested on OLTP and OLAP features and they are being considered for implementing data warehouses [5],[6],[8],[14],[34]. In [34], the authors implement a data warehouse on a column-oriented store (HBase [35]). They show how to instantiate efficiently OLAP cuboids with MapReduce-like functions. In [14], the authors compare a column-oriented system (Hive on Hadoop) with a distributed version of a relational system (SQLServer PDW) on OLAP queries, where the relational system is shown to perform better in most cases. In [5],[6],[8] , we have already studied column-oriented and document-oriented models for implementing multidimensional data warehouses. However, the focus of the study was limited to mapping a conceptual multidimensional model to logical NoSQL models.

Existing benchmarks for data warehousing are designed to be compatible with relational systems: they generate uniform and csv-like data, and the queries are in SQL. Complex but more complete benchmarks (e.g. TPC-H or TPC-DS) need substantial efforts to be adapted for evaluating NoSQL systems. Until now, the only data warehouse benchmark that has been adapted for NoSQL systems is the Star Schema Benchmark [7],[13],[23].

Document-oriented systems offer interesting data structures such as nested sub-documents and arrays. These features also exist in object-oriented and XML-like systems. However, none of the above has met the success of RDBMS for implementing data warehouses and in particular for implementing OLAP cuboids as we do in this paper. In [18], different document logical models are compared to each other, using denormalized and normalized data as well as models that use nesting. However, this study is in a "non-OLAP" setting (i.e. OLTP).

III. From a Multidimensional Data Model to Documents

A. Multidimensional data model for data warehouses

We use the reference conceptual model in data warehouses is the multidimensional data model.

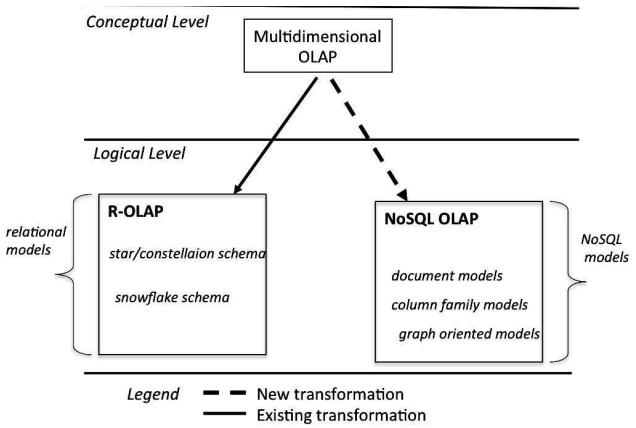

Fig 1 Translation of a conceptual model into logical models

[1] OLTP (On-Line Transactional Processing) manages data transactions rather than data analyses (i.e. OLAP).

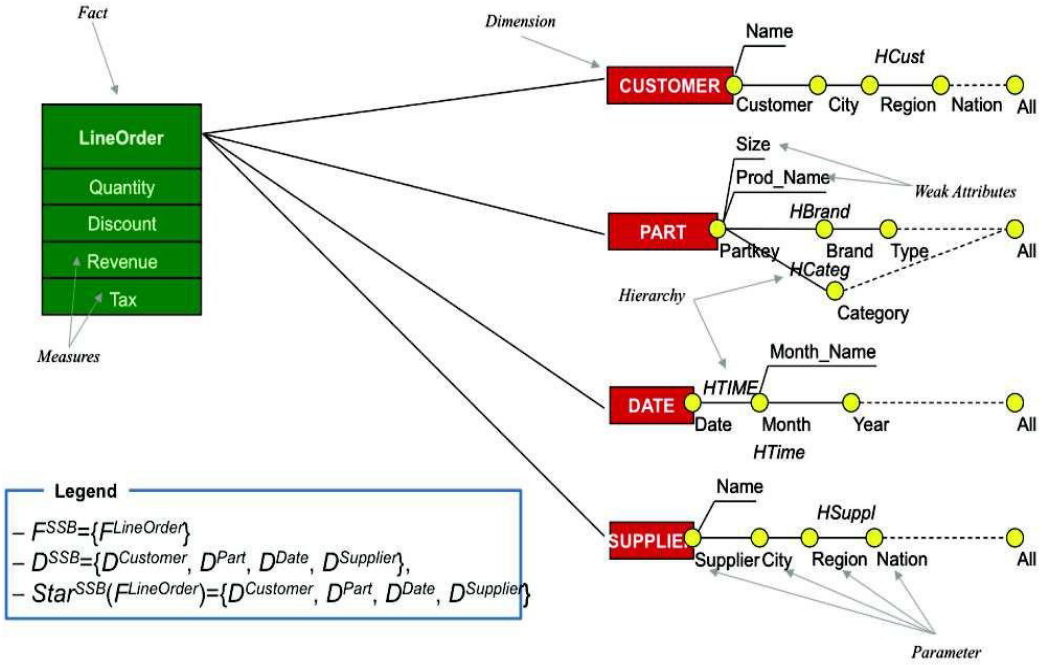

Fig 2 Graphical notations of the multidimensional conceptual model.

It is important to map this model into logical models specific to document-oriented systems that we call "document models". This has already been done for relational databases where we map a conceptual model (see figureFig 2 for an example) into a logical star model or snowflake model.

We will first introduce our conceptual multidimensional model. [15],[25], defined here after.

A **multidimensional schema**, denoted E, is defined by (F^E, D^E, $Star^E$) where:

- $F^E = \{F_1, \ldots, F_n\}$ is a finite set of facts;
- $D^E = \{D_1, \ldots, D_m\}$ is a finite set of dimensions;
- $Star^E$: $F^E \rightarrow 2^{D^E}$ is a function that associates each fact to set of dimensions along which it can be analyzed.

A **dimension**, denoted $D \in D^E$ (noted abusively D_i), is defined by (N^D, A^D, H^D,) where:

- N^D is the name of the dimension,
- $A^D = \{a_1^D, \ldots a_u^D\} \cup \{id^D, all^D\}$ is a set of attributes,
- $H^D = \{H_1^D, \ldots, H_v^D\}$ is a set hierarchies.

A **hierarchy**, denoted $H_i \in H^D$, is defined by (N^{Hi}, $Param^{Hi}$, $Weak^{Hi}$) where:

- N^{Hi} is the name of the hierarchy,
- $Param^{Hi} = < id^D, pH_1^i, \ldots, pH_{vi}^i, all^D >$ is an ordered set of attributes which are called parameters of the relevant graduation scale, $\forall k \in [1..v_i]$, $p_k^{Hi} \in A^D$.
- $Weak^{Hi} : Param^{Hi} \rightarrow 2^{A^D - Param^{Hi}}$ is a function possibly associating with each parameter one or more weak attributes.

A ***fact***, noted $F \in F^E$, is defined by (N^F, M^F) where:

- N^F is the name of the fact,
- $M^F = \{f_1(m_1^F), \ldots, f_v(m_v^F)\}$ is a set of measures. Typically, we apply aggregation functions on measures.

A combination of dimensions represents the analysis axes, while the measures and their aggregations represent the analysis values.

An example of a multidimensional conceptual schema is displayed in Fig 2 using notation from [15],[25] Here, the data model is almost the same as the one used in the SSB benchmark [6],[13],[23]. There are one fact *LineOrder* and 4 dimensions: *Customer, Part, Date* and *Supplier*. There are attribute hierarchies such as *{id, date, month, year, all}*.

B. Generic document model

Here, we provide key definitions and notation we will use to formalize documents.

A **document** is defined as a set of key-values. Keys define the structure of the document; they act as meta-data. Each value can be an atomic value (number, string, date…) or a document. Such documents within documents are called sub-documents.

The **document structure** (or document schema) corresponds to a generic document without atomic values i.e. only keys. A document belongs to a collection *C* and has a unique identifier. We refer to such a document as *C*(*id*). We use the colon symbol **":"** to separate a key from its value, **"[ ]"** to denote arrays, **"{ }"** to denote documents and a comma **","** to separate key-value pairs from each other.

Example. The document below belongs to the "*Persons*" collection, it has *30001* as identifier and it contains keys such as "*name*", "*addresses*", "*phone*". The address values correspond to an array and the phone value corresponds to a

sub-document. The document schema is: `{name, addresses: [{city, country}], phone: {prefix, number}}`.

```
Persons(30001):
{ name: "John Smith",
  addresses: [{city: "London", country: "UK"}, {city: "Paris", country: "France"}],
  phone: {prefix: "0033", number: "61234567"} }
```

C. Logical document models for data warehousing

It is at the logical level that we have to make choices on technology specific modeling. We consider here two logical document models by analogy with the relational models used for data warehousing. The models, denoted **DFL** and **DSH**, are respectively equivalent to complete data denormalization (flat) and star-like normalization in ROLAP [20]. Each model is defined below. As an illustration, we use a conceptual model example with one fact named *"LineOrder"* and measures M^F = {*"l_quantity", "l_shipmode", "l_price"*} and one dimension *"Customer"* composed of the attributes A^D = {*"c_name","c_city","c_nation_name"*}.

Model DFL (Document FLat) corresponds to a simple (denormalized) flat model. Every fact F is stored in a collection C^F with all attributes of its associated dimensions. It corresponds to denormalized data (in RDBMS). Documents are flat (no data nesting) where all attributes are at the same level. The schema S^F of the collection C^F is:

$$S_F = \{id_F, m_1, m_2, \dots m_{|M^F|}, a_1^{D_1}, a_2^{D_1}, \dots a_{|A^{D_1}|}^{D_1}, a_1^{D_2}, a_2^{D_2}, \dots a_{|A^{D_2}|}^{D_2}, \dots\}$$

In our example, this corresponds to the schema {id, l_quantity, l_shipmode, l_price, c_name, c_city, c_nation_name} with a possible instance:

{id:1,l_quantity: 4,l_shipmode: "mail", l_price:400.0, c_id:4} $\in C^{\text{LineOrder}}$

{id:4,c_name: "John", c_city: "Rome", c_nation_name: "Italy"} $\in C^{\text{Customer}}$

Model DSH (shattered): It corresponds to a simple data model where fact records are stored separately from dimension records to avoid redundancy (equivalent to normalization). The fact F is stored in a collection C^F and each associated dimension D is stored in a collection C^D**.** The fact documents contain foreign keys towards the dimension collections. The schema S^F of C^F and the schema S^D of a dimension collection C^D are as follows:

$$S_F = \{id_F, m_1, m_2, \dots m_{|M^F|}, id_{D_1}, id_{D_2}, \dots\} \quad S_D = \{id_D, a_1^{\mathrm{D}}, a_2^{\mathrm{D}}, \dots a_{|A^D|}^{\mathrm{D}}\}$$

In our example, this corresponds to two collections, one for the fact with the schema {id, l_quantity, l_shipmode, l_price} and another collection for the dimension with schema {c_name, c_city, c_nation_name}. Below, we provide two possible instances:

{id:1,l_quantity: 4,l_shipmode: "mail",l_price:400.0,c_id:4} $\in C^{LineOrder}$

{id:4,c_name: "John", c_city: "Rome", c_nation_name: "Italy"} $\in C^{Customer}$

IV. Extended OLAP cuboids

A. OLAP cuboids

An **OLAP cuboid** corresponds to a materialized view on aggregated data; it contains a subset of aggregated measures on a subset of analysis dimensions. A cuboid is defined on a set of attributes A from some dimensions of interest D, and T a set of aggregation functions on M^F, all the measures from some fact F of interest. In a simplified manner, we will note a cuboid with $c(A,T)$ defined as a pre-computed view where we group data on the dimension attributes from A and we compute aggregation functions (e.g. *min, max, sum*) on the measures of interest M^F. An example of an OLAP cuboid with our notation is $c(day^{Date}, c\_id^{Customer}, sum(revenue^{Sales}), count(revenue^{Sales}))$. In the latter, we group data on dimensions *Date* and *Customer* respectively on attributes *day* and *c_id*. The aggregation functions (*sum* and *count*) are applied on the measure *revenue* from the fact *LineOrder*.

Partial ordering: Cuboids can be placed in a partially ordered set based on the analysis detail level. The partial order operator $\prec$ is defined on attribute sets (analysis axis). We have $\{a\} \prec \{b\}$ when a and b are attributes in the same dimension hierarchy and a is lower in a hierarchy than b e.g. $\{month\} \prec \{day\}$. $\{a\} \prec \{b\}$ means that we go to finer detail when we analyze data on more attributes.

Cube, lattice: If an OLAP cuboid is generated for each dimension combination, the resulting set of cuboids is called an *OLAP cube* and the partially ordered set is called an *OLAP cube lattice*.

B. Extended OLAP cuboids

In addition to the traditional OLAP cuboids, we study other types of cuboids that are not possible with relational databases.

A **nested cuboid (N-cuboid)** is an extension of a classic OLAP cuboid where we nest at each record of the cuboid arrays of aggregated measures at a finer level of granularity. Let A and A' be sets of dimension attributes (at most one attribute per dimension) such that $A \prec A'$. A nested cuboid $c(A,T,[A',T'])$ is an extension of the classic cuboid $c(A,T)$ where data is grouped first on attributes from A and then on attributes from A'. In each document, we have a distinct tuple from A, aggregation results from T followed by an array of dimensions at lower-granularity (combination of A and A') and aggregation functions from T'. For instance, $c(country, sum(revenue),[city, sum(revenue)])$ is a nested cuboid that can have records such as:

```
{ country: "FRA", sum_revenue: 45,0$,
  by_city: [
   {city: "Paris",    sum_revenue: 12,0$},
   {city: "Toulouse", sum_revenue: 13,0$},
   {city: "Lyon",     sum_revenue: 20,0$}] }
```

We observe that the above cuboid groups data on the attribute country and then it nests cities ($\{country\} \prec \{city\}$). The nested cuboid has the following advantages:

- It can be used to group multiple cuboids in one, saving memory.
- It allows drilling-down directly using the cuboid data without having to use the detailed data.

A **detail cuboid (D-cuboid)** is an extension of a classical cuboid where we nest arrays of detailed data. Let *M* be a set of measures. *A* detail cuboid *c(A,T,[M])* contains data grouped on attributes *A*, similar to the cube *c(A,T)*, with the addition of arrays of measure values from *M*; e.g. *c(country, , sum(revenue),[id, revenue])* is a detail cuboid that can have records such as:

```
{ country: "FRA",  sum_revenue: 45,0$,
  detail: [{id: 15,    revenue: 2,0$},
           {id: 18,    revenue: 3,0$}, …,
           {id: 10048, revenue: 3,5$} ] }
```

The above cuboid groups data using "*day*" and "*customer*" dimension attributes and it stores not only the aggregated function result (*sum*) but also the detailed data of *revenue* and product *id*-s. This extended model comes at the cost of memory usage, but it has the following advantages compared with the traditional approach:

- We can drill down on data easily i.e. view fact details.
- We can run faster random aggregation functions (not known before hand) on data and not only the traditional: max, min, count, average, sum.
- We can compute set operations: intersection, union, frequent items, etc. e.g. the common purchases on a given group of customers.
- The above notation is coherent with all types of cuboids we described (classic, nested and detailed). We can also imagine and denote nested cuboids with detailed data.

Note that the extended cuboids are not possible with relational databases. Even if nesting arrays is possible in XML or object-oriented databases, the latter has not been studied thoroughly for an OLAP usage.

V. Experiments

A. Experiments general settings

The experimental setup is briefly introduced here and then detailed in the next paragraphs. We generate data according to the SSB (Star Schema Benchmark) data model [7],[13],[23] . Data is loaded in MongoDB v3.0, a popular document-oriented system and PostgresSQL v8.4, a popular RDBMS. On both systems we consider a flat model and a star-like normalized model. On each dataset, we issue sets of OLAP queries and we compute OLAP cuboids on different combinations of dimensions. We also test extended versions of OLAP cuboids. Experiments in MongoDB are done in both a single-node and a distributed 3-nodes cluster setting. Experiments in PostgreSQL are done in a singlenode setting. The rest of the experimental setup is detailed below.

Data. We generate data using an extended version of the Start Schema Benchmark SSB [1],[13],[23] because it is the only data warehousing benchmark that has been adapted to NoSQL systems [1],[22]. The SSB benchmark models a simple product retail reality. It contains one fact "*LineOrder*" and 4 dimensions "*Customer*", "*Supplier*", "*Part*" and "*Date*". The extended version is part of our previous work [22]. It allows generating raw data directly as JSon which is the preferable data format for loading data in MongoDB. We use improve scaling factor issues that have been reported. In our experiments we use different scale factors (*sf*) such as *sf=1, sf=10* and *sf=25*. In the extended version, the scale factor *sf=1* corresponds to 10^7 records for the *LineOrder* fact, for *sf=10* we have $10x10^7$ records and so on.

Settings/hardware/software. The experiments have been done in two different settings: single-node architecture and a cluster of 3 physical nodes. Each node is a Unix machine (CentOs) with a 4 core-i5 CPU, 8GB RAM, 2TB disks, 1Gb/s network. The cluster is composed of 3 nodes, each being a worker node and one node acting also as a dispatcher. Each node has a MongoDB v.3.0 running. In MongoDB terminology, this setup corresponds to 3 shards (one per machine) and one machine also acts both as configuration server and client.

Models. We will refer to 4 data models depending on the database system used and the logical data model underneath. We will use the abbreviations **DFL** and **DSH** for respectively flat and star-like normalized document models. In analogy, we will consider flat and star-like normalization data models in relational models that will be named **RFL** and **RSH**.

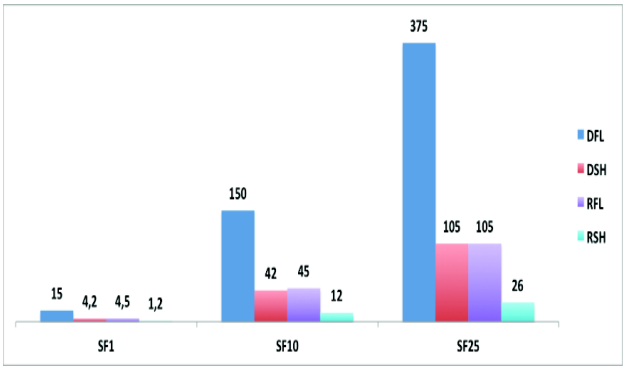

Fig 3 Memory usage (GB) by model on different scale factors.

B. From data warehouse loading to OLAP cuboids

Loading. In Fig 3, we show the storage space required by each approach (document models DFL and DSH and relational models RSH and DSH) on 3 scale factors (*sf=1, sf=10, sf=25*). Instantiation on PostgreSQL requires less space than in MongoDB (from 3 to 5 times). This is easily explained: document-oriented systems repeat field names on every document and specifically in MongoDB data types are also stored explicitly. To store data with flat models we need about 4 times more space, due to data redundancy. For instance, at scale factor *sf=10* (10^8 line order records) we need 150GB and

Table 1 Query execution times per model for sf=10, in seconds.

	Mongo, singlenode		PostgreSQL, singlenode		Mongo, cluster	
	DFL	**DSH**	**RFL**	**RSH**	**DFL**	**DSH**
QS1.1	1317s	1403s	*720s*	*143s*	*625s*	*400s*
QS1.2	1448s	1587s	*577s*	*60s*	*585s*	*392s*
QS1.3	1001s	1315s	*364s*	*60s*	*2091s*	*388s*
QS1 avg	*1255s*	*1433s*	*553s*	*88s*	*1100s*	*393s*
QS2.1	1284s	1384s	*399s*	*67s*	*364s*	*922s*
QS2.2	1002s	1202s	*397s*	*70s*	*373s*	*884s*
QS2.3	1411s	1611s	*443s*	*60s*	*361s*	*885s*
QS2 avg	*1232s*	*1399s*	*413s*	*66s*	*361s*	*897s*
QS3.1	1438s	1645s	*738s*	*61s*	*363s*	*942s*
QS3.2	1442s	1701s	*740s*	*61s*	*404s*	*955s*
QS3.3	1127s	1821s	*657s*	*61s*	*374s*	*963s*
QS3 avg	*1335s*	*1722s*	*711s*	*61s*	*380s*	*953s*
avg	***1274s***	***1518s***	***559s***	***71s***	***610s***	***747s***

42GB for respectively DFL and DSH and 45GB and 12GB for RFL and RSH.

In Fig 4, we show loading times by model at different scale factors (single node). Denormalized data is loaded about 23% to 26% faster with PostgresSQL (model RFL) than with MongoDB (model DFL). Instead, star-like normalized data is loaded about 35-40% faster on MongoDB (model DSH) than on PostgresSQL (model RSH). The latter observation is explained by the fact that in PostgresSQL, the foreign key constraint slows down the data loading process significantly. If loading performance is analyzed using MB stored per second, we observe that MongoDB stores about 18MB/s to 21MB/s while PostgresSQL stores about 3MB/s to 8MB/s. This explains why the loading time for denormalized data models is comparable across systems although MongoDB requires about 4 times more storage space.

In we show loading times in two settings: single node and cluster, both with a scale factor *sf=1*. Loading data is observed to be slower in a distributed setting. For instance, data (*sf=1*) is loaded into the DFL model in 588s on a single cluster, while it needs 1912s on a distributed setting; about 3 times more. This is mainly due to penalization related to network data transfer: MongoDB balances data volumes and it tries to distribute equally data across all shards implying more network communication.

OLAP queries. Each model and server configuration is tested using 3 sets of OLAP queries (QS1, QS2, QS3). To do so, we use the SSB benchmark query generator that generates 3 query variants per set. The query complexity increases from QS1 to QS3: QS1 queries filter on one dimension and aggregate all data; QS2 queries filter data on 2 dimensions and group data on one dimension; and QS3 queries filter data on 3 dimensions and group data on 2 dimensions.

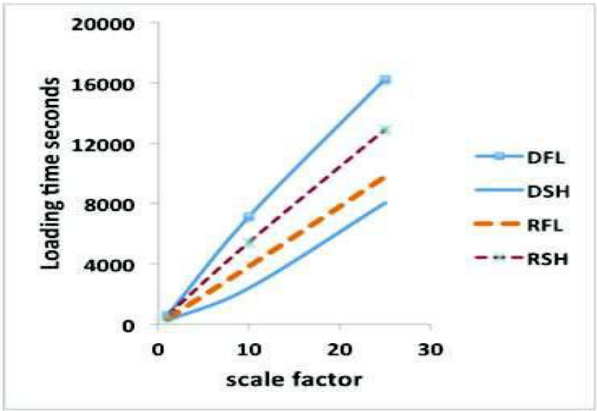

Fig 4 Loading times by model and scale factor

In Table 1, we show query execution times on all query variants with a scale factor *sf=10,* for all models. In MongoDB we consider two settings (single node and cluster).

Table 2 Execution times on the computation of 3 dimensional OLAP cuboids

Cuboid	DFL, singlenode	RSH, singlenode	DFL cluster
c_city, s_city, p_brand	7466s	9897s	6480s
c_city, s_city, d_date	3540s	6742s	2701s
c_city, p_brand, d_date	4624s	9302s	*3358*s
s_city, p_brand, d_date	4133s	8509s	3301s
avg	**4941s**	**8612s**	**3960s**

Query variant results are averaged (using an arithmetic mean) for each query set.

In MongoDB, the execution is faster on the denormalized data model (DFL) than on the star-like data model DSH. This is explained by the poor support of joins by MongoDB. The resuts on DSH are better than the ones that can be obtained on a simple translation of the queries in MongoDB querying language. We improved query execution by filtering data before having to perform join-like operations. This kind of optimizations will be manual until document-oriented systems will provide better support for joins. Instead, with PostgresSQL we observe that queries from these 3 sets run significantly faster on the star-like normalized model RSH than on the denormalized model RFL, from about 6 to 12 times. This is not surprising if we consider that RDBMS are optimized on joins and the fact that on given circumstances we need to load in memory less data with DSH.

We observe that these queries run faster on PostgresSQL than on MongoDB. They run about 15 to 22 times faster on PostgresSQL with data model RSH than on MongoDB with data model RFL. On these query sets, PostgreSQL is shown significantly superior to MongoDB. However, we observed these queries are particularly selective. Due to compact storage in relational systems, all data to be processed can fit in main memory after filtering. This is an advantage to RDBMS systems.

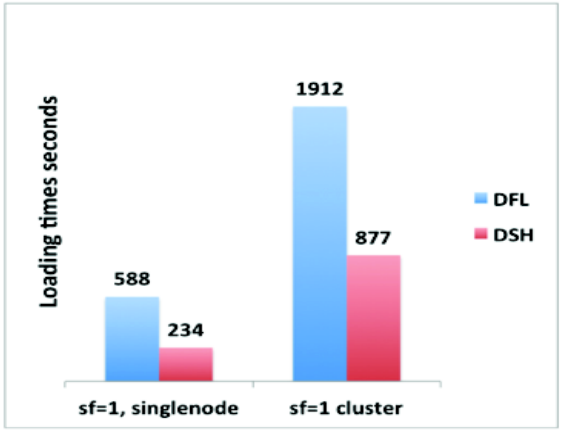

Fig 5 Loading times on single node versus cluster mode with sf=1.

With MongoDB, we observe that query execution times are generally better in a distributed setting. For many queries, execution times improve 2 to 3 times depending on the cases. In a distributed setting, query execution is penalized by network data transfer, but it is improved by parallel computation.

OLAP cuboid queries: In addition, we considered OLAP queries that correspond to the computation of OLAP cuboids. These queries are computationally more expensive than the queries considered previously. More precisely, we consider here the generation of OLAP cuboids on combinations of 3 dimensions. Computation times are shown in . The cuboids are denoted on the dimension attributes they group data on. For instance, *c_city, s_city, p_brand* stands for a cuboid that groups data on customer city, supplier city and part brand.

Results are shown only on best performing datasets on PostgresSQL and MongoDB, namely on data model DFL and RSH. This is done because we observed that computation times became significantly important (unrealistic) on data models RFL and DSH.

We observe that the situation is reversed on this query set: queries run faster (2 times) on MongoDB with data model DFL (singlenode); they run slower on PostgresSQL with data model RSH. We also observe a further improvement on the distributed setting. On these queries we have to keep in memory much more data than for queries in QS1, QS2 and QS3. Indeed, on the query sets QS1, QS2 and QS3, the amount of data to be processed is reduced by filters (equivalent of SQL where instructions). Then data is grouped on fewer dimensions (0 to 2). The result is fewer data to be kept in memory and fewer output records. Instead for computing 3 dimensional cuboids, we have to process all data and the output has more records. Data will not fit in main memory neither for MongoDB nor for PostgreSQL, but MongoDB seems suffering less than PostgreSQL.

We can conclude that MongoDB scales better when the amount of data to be processed increases significantly. It can also take advantage of distribution. Instead, PostgresSQL performs very well when all data fits in main memory.

OLAP cuboids and querying: Direct querying on raw data is rare; instead it is common to compute OLAP cuboids that will somehow cache results or intermediary results**.** We considered executing the queries from sets QS1,QS2, QS3 directly on OLAP cuboids. For comparative reasons, we consider 4 data warehouse systems:

- **S0**: Data generated in MongoDB with model DFL without any OLAP cuboids;
- **S1**: Data generated in MongoDB with model DFL and OLAP cuboids;

- **R0**: Data generated in PostgreSQL with model RSH without any OLAP cuboids
- **R1**: Data generated in PostgreSQL with model RSH and OLAP cuboids.

Table 3 Query execution times with or without OLAP cuboids, sf=10

	QS1	QS2	QS3	avg
S0	1255s	1232s	1335s	1274s
S1	0.1s	0.1s	0. 2s	0.1s
R0	88s	66s	61s	4.2s
R1	0.1s	0.3s	0.5s	0.3s

Average query times per query set are reported in Table 3, Results show that when we use OLAP cuboids to support query execution, query execution is significantly faster; more than 10000 times on MongoDB and about 600-800 times on PostgreSQL. The performance differences are no longer huge between PostgreSQL and MongoDB; they are more than comparable. This is because in the OLAP cuboids contain the results of the queries and we just need to find the right record. In cases, MongoDB is able to retrieve aggregated data from the respective OLAP cuboids faster than PostgreSQL.

Concluding remarks: This set of experiments illustrates multidimensional data warehouse instantiation from loading to OLAP cuboid computation. It allows initial comparison among modeling and technology choices. Some findings are interesting, others intuitive. We observed that querying a document-oriented system is slower than on a RDBMS when queries are selective, but it performs better when a lot of data is to be processed. Document-oriented systems scale well with the increase of data volume; its performance is further improved through distribution. We also observe that querying time is significantly reduced when we use OLAP cuboids.

C. Extended OLAP cuboids

In this section, we focus on extended cuboids defined earlier. We study their computation and their utility on two different workloads. We have computed nested and detail cuboids on different combination of dimensions. We use a document-oriented system (MongoDB v3.0) with raw data following the data model DFL. During the experiments, we had some issues. MongoDB has a limit on document size (16MB). The records of nested or detail cuboids did not always fit within the document limit. In this case, we split the document into parts.

Utility of extended cuboids: To assess the utility of nested cuboids and detail cuboids, we designed two experiments with query loads that include drill-down like queries and detail-level queries. We consider 3 types of data warehouse systems:

- *S1*: we have raw data and classical cuboids on all dimension combinations
- *S2*: we have raw data and nested cuboids on all dimension combinations
- *S3*: we have raw data and detail cuboids on all dimension combinations

In all systems, we cache the result of the last executed query. This is useful if we want to drill-down in the result.
We consider two different experimental settings:
Setting 1: We consider a workload of OLAP queries followed by drill-down queries. This is a typical sequence of queries in OLAP; e.g. we start analyzing data by country and then we want to drill down to specific cities. More precisely, we consider a workload of 40 queries. They include 8 normal OLAP queries and for each of them we have produced 4 random drill-down queries. More precisely, we have:

- QS_1 ={Q_1,Q_2, Q_3,Q_4,Q_5,Q_6,Q_7,Q_8, Q_9, Q_{10},…Q_{40}} is a workload of 40 queries
- QT = {Q_1, Q_6, Q_{11}, Q_{16}, $Q_{19,}$ …Q_{36}} are 8 random OLAP queries. For every query Q_i in QT, we have produced 4 random queries {Q_{i+1}, Q_{i+2}, Q_{i+3}, Q_{i+4}} that ask for aggregate functions at a lower level of granularity i.e. a drill down operation.

Setting 2: We consider a workload of OLAP queries with the intrusion of some detail-level queries. This setting can also happen; e.g. we start analyzing data on some attributes and we want to explain the observed data showing raw data. More precisely, we consider a workload of multiple OLAP queries where every 4 queries we issue one query that asks for detailed data. We have:

- QS ={Q_1,Q_2, Q_3,Q_4,Q_5,Q_6,Q_7,Q_8, Q_9, Q_{10},…Q_{40}} a workload of 40 queries
- Queries {Q_1,Q_2, Q_3,Q_4,Q_6,Q_7,Q_8, Q_9,Q_{11},Q_{12}… } are random OLAP queries whereas QD = {Q_5, $Q_{10,}$ Q_{15}, $Q_{20,}$ Q_{25}, …} are detail-level queries and the other are normal OLAP queries. They retrieve raw data that concern the preceding OLAP query in the workflow.
- **Results**: In Fig 6 and Fig 7, we show the total query time on the evolution of the query load (query count). On the Fig 6, we compare the system with classic cuboids (*S1*) to the system with nested cuboids (*S2*) using the workload from setting 1. We observe that *S2* performs better than *S1*. In particular, we observe that the major performance difference is met with drill-down queries. In fact, the system *S1* has to query on another OLAP cuboid, while the system *S3* can re-use the last query result that has data of lower-level detail.

Table 4 Compute time and memory use of OLAP cuboids by types and number of dimensions

No. of dimensions, Cuboid type	OLAP cuboid example	Computation time	Memory use
3D, classic	*c(supplier,date,part,T)*	428s	4065MB
2D, classic	*c(supplier,date,T)*	197s	86MB
1D, classic	*c(date,T)*	1s	0.3MB
average		207s	1198MB
3D, nested	*c(supplier,date,part, T,[customer,T])*	476s	4950MB
2D, nested	*c(supplier,date,T[part,T])*	227s	1897MB
1D, nested	*c(date,T,[supplier,T])*	10s	512MB
average		189s	2373MB
3D, detail	*c(supplier,date,part, T,[M])*	521s	5917MB
2D, detail	*c(supplier,date,T, [M])*	251s	2600MB
1D, detail	*c(date,T,[M])*	59s	2110MB
average		273s	3407MB

In Fig 7, we compare the system (*S1*) with classic cuboids to the system with detail cuboids (*S3*) using the workload from setting 2. We observe that *S3* performs better than *S1*. In particular, we observe that the main performance difference is met with the detail-level queries (one every 5 queries). This is because when the query asks for detail-level data, the system *S1* has to query on raw data while the system *S3* has already the detail data available from the last query result.

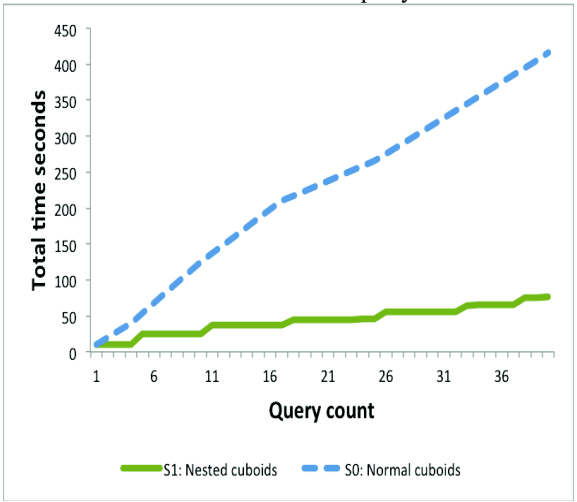

Fig 6 Cuboids compared on different query loads.

Memory use and computation times: In Fig 4, we compare classic cuboids, detail cuboids and nested cuboids computed on computation time and memory usage. On a given type of cuboid we generate OLAP cuboids for every combination of 3 dimensions, 2 dimensions and 1 dimension. We aggregate data on 3 measures. We average the computation time and memory use.

Results show that the extended cuboids demand for more space and the computation time is higher. The difference increases on 2-dimensional cuboids and 1-dimensional cuboids. **Concluding remarks:** We analyze the utility of detail cuboids and nested cuboids. They are shown to answer efficiently workloads with drill-down and detail-level queries.

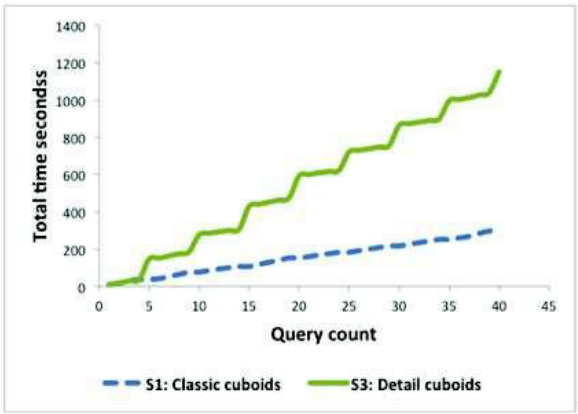

Fig 7 Cuboids compared on different query loads

VI. CONCLUSION

In this paper, we have studied the instantiation of multidimensional data warehouses with document-oriented systems. For this purpose, we formalized and analyzed two logical models. Our study highlights weaknesses and strengths across the models; it studies extended OLAP cuboids and we compare performance with a RDBMS.

We show how to instantiate a multidimensional data warehouse with two data models: denormalized model (DFL) and star-like normalized model (DSH). We study data loading

at different scale factors (10^7 fact records – 2.5 x 10^8 fact records). The star-like normalized model (DSH) requires less space and data loads faster. However, this latter model suffers on data joins that are not well supported by document-oriented systems. On the other side, the denormalized data model shows better querying performance and queries are easier to write.

The document-oriented systems are compared with RDBMS on two equivalent data models. Results show that RDBMS is faster on querying raw data. But performance slows down quickly when data does not fit on main memory. Instead, the analyzed document-oriented system is shown more robust i.e. it does not have significant performance drop-off with scale increase. As well, it is shown to benefit from distribution. This is a clear advantage with respect to RDBMS which do not scale well horizontally; they have a lower maximum database size than NoSQL systems. Querying times are shown to reduce significantly when we query directly on OLAP cuboids. This makes both RDBMS and document-oriented systems comparable on OLAP querying performance.

In the second phase of our experiments, we study two extended versions of OLAP cuboids (that are not possible in traditional RDBMS): nested cuboid and detail cuboid. They come at the cost of additional memory usage, but they are shown to perform well on specific workloads. More precisely, nested cuboids support well workloads with drill-down queries, while detail cuboid support well detail-level queries. These cuboids are not stored naturally in relational databases i.e. they need to be transformed and split in separate tables. In document-oriented systems, we can store naturally the extended cuboids records within documents due to arrays and nesting. The only limitation is the maximal document size. However, this is document specific and it can be solved by appropriate array spitting.

We can conclude that document-oriented systems are a promising playground for data warehouses. There is more to be investigated. As future work, we recommend focusing on the novel queries that can be answered by document-oriented systems. Map-reduce like optimizations are to be investigated as well for making possible faster and more advanced analysis on data.

Acknowledgements

This work is supported by the ANRT funding under CIFRE-Capgemini partnership.

References

[1] D.J. Abadi, D.S. Myers, D.J. DeWitt and S.R. Madden. Materialization Strategies in a Column-Oriented DBMS. *IEEE 23rd Int. Conf. on Data Engineering* (*ICDE*), IEEE, pp. 466-475, 2007.

[2] A. Bosworth, J. Gray, A. Layman, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. ACM, 52(8):36{44, Aug. 2009.

[3] R. Cattell. Scalable SQL and NoSQL data stores. SIGMOD Record 39(4), ACM, pp. 12-27. 2011. 39, 4 (May 2011), 12-27. DOI=http://dx.doi.org/10.1145/1978915.1978919

[4] S. Chaudhuri and U. Dayal. An overview of data warehousing and OLAP technology. SIGMOD Record 26(1), ACM, pp. 65-74, 1997.

[5] M. Chevalier, M. El Malki, A. Kopliku, O. Teste, Ronan Tournier. Not Only SQL Implementation of multidimensional database. International Conference on Big Data Analytics and Knowledge Discovery (DaWaK 2015a), p. 379-390, 2015.

[6] M. Chevalier, M. El Malki, A. Kopliku, O. Teste, R. Tournier. Implementation of multidimensional databases in column-oriented NoSQL systems. East-European Conference on Advances in Databases and Information Systems (ADBIS 2015b), p. 79-91, 2015.

[7] M. Chevalier, M. El Malki, A. Kopliku, O. Teste, R. Tournier. Benchmark for OLAP on NoSQL Technologies. IEEE International Conference on Research Challenges in Information Science (RCIS 2015c), p. 480-485, 2015.

[8] M. Chevalier, M. El Malki, A. Kopliku, O. Teste, R. Tournier. *How Can We Implement a Multidimensional Data Warehouse Using NoSQL?* Dans : *Enterprise Information Systems*. Slimane Hammoudi, Leszek Maciaszek, Ernest Teniente, Olivier Camp, José Cordeiro (Eds.), Springer, p. 108-130, Vol. 241, Lecture Notes in Business Information Processing (LNBIP, 2015d).

[9] G. Colliat. OLAP, relational, and multidimensional database systems. SIGMOD Record 25(3), ACM, pp. 64.69, 1996.

[10] A. Cuzzocrea, I.-Y. Song, and K. C. Davis. Analytics over large-scale multidimensional data: The big data revolution! In Proceedings of the ACM 14th International Workshop on Data Warehousing and OLAP, DOLAP '11, pages 101{104, New York, NY, USA, 2011. ACM.

[11] A. Cuzzocrea, L. Bellatreche and I.Y. Song. Data warehousing and OLAP over big data: current challenges and future research directions. 16th international workshop on Data warehousing and OLAP (DOLAP), ACM, pp. 67-70, 2013.

[12] E. Dede, M. Govindaraju, D. Gunter, R.S. Canon and L. Ramakrishnan. Performance evaluation of a mongodb and hadoop platform for scientific data analysis. 4th ACM Workshop on Scientific Cloud Computing (Cloud), ACM, pp.13-20, 2013.

[13] K. Dehdouh, O. Boussaid and F. Bentayeb. Columnar NoSQL star schema benchmark. Model and Data Engineering, LNCS 8748, Springer, pp. 281-288, 2014.

[14] A. Floratou, N. Teletia, D. Dewitt, J. Patel and D. Zhang. Can the elephants handle the NoSQL onslaught? Int. Conf. on Very Large Data Bases (VLDB), pVLDB 5(12), VLDB Endowment, pp. 1712–1723, 2012.

[15] M. Golfarelli, D. Maio and S. Rizzi. The dimensional fact model: A conceptual model for data warehouses. Int. Journal of Cooperative Information Systems 7(2-3), World Scientific, pp. 215-247, 1998.

[16] Adam Jacobs. 2009. The pathologies of big data. *Commun. ACM* 52, 8 (August 2009), 36-44. DOI=http://dx.doi.org/10.1145/1536616.1536632

[17] F. M. Jiang, J. Pei, A. W. Fu. Ix-cubes: iceberg cubes for data warehousing and olap on xml data. In Conf. on Information and Knowledge Management (CIKM), ACM, pp. 905-908, 2007.

[18] A. Kanade, A. Gopal and S. Kanade. A study of normalization and embedding in MongoDB. IEEE Int. Advance Computing Conf. (IACC), IEEE, pp. 416-421, 2014.

[19] R. Kimball and M. Ross. The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling. John Wiley & Sons, 3rd ed., 2013.

[20] Michael J. Mior. 2014. Automated schema design for NoSQL databases. In *Proceedings of the 2014 SIGMOD PhD symposium* (SIGMOD'14 PhD Symposium). ACM, New York, NY, USA, 41-45. DOI=http://dx.doi.org/10.1145/2602622.2602624

[21] Konstantinos Morfonios, Stratis Konakas, Yannis Ioannidis, and Nikolaos Kotsis. 2007. ROLAP implementations of the data cube. *ACM Comput. Surv.* 39, 4, Article 12 (November 2007).

[22] Ameya Nayak, AnilPoriya, DikshayPoojary Type of NOSQL Databases and its Comparison with Relational Databases International Journal of Applied Information Systems (IJAIS) ISSN 2249-0868 Foundation of Computer Science FCS, New York, USA Volume 5 No.4, March 2013 www.ijais.org

[23] P. ONeil, E. ONeil, X. Chen and S. Revilak. The Star Schema Benchmark and augmented fact table indexing. Performance Evaluation and Benchmarking, LNCS 5895, Springer, pp. 237-252, 2009.

[24] A. Pavlo, E. Paulson, A. Rasin, D.J. Abadi, D.J. DeWitt, S. Madden and Michael Stonebraker. A comparison of approaches to large-scale data analysis. Int. Conf. on Management of data (SIGMOD), ACM, pp. 165-178, 2009.

[25] F. Ravat, O. Teste, R. Tournier and G. Zuruh. Algebraic and graphic languages for OLAP manipulations. Int. J. of Data Warehousing and Mining (IJDWM), 4(1), IDEA, pp. 17-46, 2008.

[26] . Schindler. I/O characteristics of NoSQL databases. Int. Conf. on Very Large Data Bases (VLDB), pVLDB 5(12), VLDB Endowment, pp. 2020-2021, 2012.

[27] J. Shute, R. Vingralek, B. Samwel, B. Handy, C. Whipkey, E. Rollins, M. Oancea, K. Littlefield, D. Menestrina, S. Ellner, J. Cieslewicz, I. Rae, T. Stancescu, H. Apte. F1: A distributed SQL database that scales. *Int*. Conf. on *Very Large Data Bases* (*VLDB*), pVLDB 6(11), VLDB Endowment, pp. 1068-1079. 2013.

[28] A. Simitsis, P. Vassiliadis, and T. Sellis. Optimizing etl processes in data warehouses. In Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on, pages 564{575, April 2005.

[29] M. Stonebraker, S. Madden, D.J. Abadi, S. Harizopoulos, N. Hachem and P. Helland. The end of an architectural era: (it's time for a complete rewrite). *33rd Int*. *conf*. *on Very large Data Bases* (*VLDB*), ACM, pp. 1150-1160, 2007.

[30] M. Stonebraker. New opportunities for New SQL. Comm. of the ACM, 55(11), ACM, pp. 10-11, 2012.

[31] D. Tahara, T. Diamond, and D. J. Abadi. Sinew: a SQL system for multi-structured data. *Int*. *Conf*. *on Management of data* (SIGMOD). ACM, pp. 815-826, 2014.

[32] T. Vajk, P. Feher, K. Fekete and H. Charaf. Denormalizing data into schema-free databases. 4th Int. Conf. on Cognitive Infocommunications (CogInfoCom), IEEE, pp. 747-752, 2013.

[33] P. Zhao, X. Li, D. Xin and J. Han. Graph cube: on warehousing and OLAP multidimensional networks. In Int. Conf. on Management of Data (SIGMOD), ACM, pp. 853-864, 2011.

[34] H. Zhao and X. Ye. A practice of TPC-DS multidimensional implementation on NoSQL database systems. Performance Characterization and Benchmarking, LNCS 8391, Springer, pp. 93-108, 2014.

[35] https://s3-eu-west-1.amazonaws.com/benstopford/nosql-comp.pdf

[36] https://hbase.apache.org/