Multidimensional Database Design via Schema Transformation: Turning TPC-H into the TPC-H*d Multidimensional Benchmark

Alfredo Cuzzocrea
ICAR-CNR and University of Calabria
Italy
cuzzocrea@si.deis.unical.it

Rim Moussa
LaTICE Lab., University of Tunis
Tunisia
rim.moussa@esti.rnu.tn

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: [Miscellaneous]; D.2.8 [**Software Engineering**]: Metrics—*Complexity measures*

General Terms

Experimentation, Performance, Design

Keywords

Multidimensional Databases, DataWarehousing, Schema Evolution, Logical OLAP Design.

ABSTRACT

Compared to relational databases, multidimensional database systems enhance data presentation and navigation through intuitive spreadsheet like views and increase performance through aggregated data. In this paper, we present a framework for automating multidimensional database schema design. We successfully used the framework to revolve the well known TPC-H benchmark to become a multidimensional benchmark -TPC-H*d benchmark, and translated into MDX language (MultiDimensional eXpressions) the TPC-H workload. In order to assess the effectiveness and the efficiency of our proposal, we benchmark the open source Mondrian ROLAP server and its OLAP4j driver with TPC-H*d benchmark.

1. INTRODUCTION

Decision Support Systems (DSS) are designed to empower the user with the ability to make effective decisions regarding both the current and future activities of an organization. One of the most powerful and prominent technologies for knowledge discovery in DSS environments are *Business Intelligence (BI) Suites* and particularly *On-line Analytical Processing (OLAP) technologies* [1, 8]. OLAP relies heavily upon a data model known as the *multidimensional databases* (MDB) [26]. Compared to relational databases, MDB increase performance by storing aggregated data and enhance data presentation. Indeed, MDB systems, offer the following three advantages [1],

- *Presentation*: MDB enhance data presentation and navigation by intuitive spread-sheet like views that are difficult to generate using SQL technologies,
- *Ease of maintenance*: Multidimensional databases are very easy to maintain, because data is stored in the same way as it is viewed, that is according to its fundamental attributes, no additional computational overhead is required for queries' processing.
- *Performance*: MDB systems increase performance. Indeed, through OLAP operations (e.g. slice, dice, drill down, roll up, and pivot), MDB systems allow intuitively the analyst to navigate through the database and screen very fast for a particular subset of the data.

The BI market continues growing and information analysts embrace OLAP concepts and technologies. According to research from market watchers, such as Pringle & Company and Gartner, the market for Business Intelligence platforms will remain one of the fastest growing software markets in most regions [13, 18]. Despite the BI booming market, there are hurdles around dealing with the volume and variety of data, and there are also equally big challenges related to the conceptual design of multidimensional databases. Also, regarding benchmarks, DSS technologies should be evaluated with appropriate OLAP benchmarks. Practically, most BI project managers focus on the following milestones for implementing a DSS (refer to [10] for details):

- Architecture sketch milestone: It consists in capturing the technologies to use, designing the data warehouse data model and business logic for extractions and transformations,
- System usage milestone: It consists in delivering a BI solution which meets end-users business requirements,
- GUI ergonomics milestone: It consists in implementing user-friendly interfaces of OLAP clients.

The MDB design milestone is very often neglected. Consequently, OLAP cubes are defined in a haphazard way, without worrying about the performance of running queries against data cubes and the costs of the maintenance the cubes. BI developers questions are, How to define cubes? will there be a single cube or multiple cubes? Which optimizations are the most suitable for running the workload? This is very complex if we consider a broad range of DSS

The 19th International Conference on Management of Data (COMAD), 19th-21st Dec, 2013 at Ahmedabad, India.

workloads with conflicting recommendations. Indeed, TPC-H benchmark [25] enumerates 22 business queries, while its successor TPC-DS [24] enumerates a hundred business queries.

The outline of this paper is as follows: Section 2 presents related work and highlights our contribution. Section 3 proposes a framework automating multidimensional database design. Section 4 presents the multidimensional TPC-H*d benchmark. The latter is obtained by application of the proposed framework to the well known TPC-H benchmark. Section 5 presents a performance evaluation of *OLAP4j driver* embedding in its core the open source *ROLAP server Mondrian* using TPC-H*d benchmark. Finally, we conclude the paper and open new work perspectives.

2. RELATED WORK

Hereafter, we present related work for both the MDBs' design based on requirements and DSS benchmarks.

2.1 Multidimensional Database and OLAP Data Cube Design

In the related literature there are a number of papers that have pointed out the necessity of OLAP cube design. Next, we overview the most relevant to our work, following the chronology of their publication,

Niemi et al. present a technique that automates cube design given the data warehouse, functional dependency information, and sample OLAP queries [14]. The user can accept the proposed OLAP cube or improve it by giving more queries.

Cheung et al. [9] demonstrate that data cube schema design problem is NP-hard. Their work aims to find OLAP cubes maximizing query performance and minimizing maintenance cost by cube merging. They propose approximate Greedy algorithms for optimal finding of a data cube schema of an OLAP system with limited memory. They evaluated the efficiency of their algorithms via an empirical study using TPC-D benchmark.

Romero et al. [19] present a method into 11 steps to validate user multidimensional requirements expressed in terms of SQL queries. In [20], Romero et al. overviewed and compared multidimensional design methodologies.

Nair et al. [12], propose a framework which analyses the user requirements and formalizes the business related needs in the form of a graph.

Malinowski et al. [11] propose a temporal extension of the multidimensional model inspired by temporal databases. The proposed model provides temporal support for levels, attributes, hierarchies, and measures.

Overviewed requirement-based methologies for MDB design introduced by Niemi et al. [14], Romero et al. [19], Nair et al. [12], Malinowski et al. [11], Thanisch et al. [22] were not generalized and applied to any of the existing DSS benchmark and no empirical study was conducted.

Open source and commercial OLAP technologies (such Pentaho BI suite, MS Analysis Services, ...), provide ETL tools (Integration services) for different data sources integration, visual tools for the design of a multidimensional databases, and different OLAP engines (such as ROLAP, HOLAP or MOLAP). Nevertheless, they do not implement an advisor for supporting analysts in the design of the multidimensional database schema.

2.2 DSS Benchmarks

To our knowledge, there are few decision-support benchmarks out of the TPC benchmarks. The only open-source benchmarks for decision support systems are APB-1 [15], DWEB [6] and TPC Benchmarks [25, 24]. Hereafter, we briefly describe existing DSS benchmarks.

2.2.1 Non-TPC Benchmarks

APB-1 has been released in 1998 by the OLAP council -a now inactive organization. APB-1 warehouse dimensional schema is structured around five fixed size dimensions and its workload is composed of 10 queries. Along Thomsen et al. [7], APB-1 is quite simple and is proved limited to evaluate the specificities of various activities.

Data Warehouse Engineering Benchmark (DWEB) proposed by Darmont et al. [6], helps in generating various ad-hoc synthetic data warehouses. DWEB is fully parameterized to fulfill data warehouse design needs and may be considered as a benchmark generator.

2.2.2 TPC Benchmarks

The Transaction Processing Performance Council (TPC) has issued several decision-support benchmarks, including TPC-H benchmark [25]. The latter is the most prominent benchmark for evaluating decision support systems. The TPC-H benchmark exploits a classical product-order-supplier model. It consists of a suite of business oriented adhoc queries and concurrent data modifications. The workload is composed of (i) twenty-two parameterized decision-support SQL queries with a high degree of complexity and (ii) two refresh functions, namely RF-1 *new sales* (new inserts of orders and related lineitems) and RF-2 *old sales* (deletes of orders and related lineitems). Scale factors used for TPC-H database test must be chosen from the set of fixed scale factors defined as follows: 1, 10, ... 100,000; resulting raw data volumes are respectively 1GB, 10GB,..., 100TB. Fig. 1 illustrates the relational database schema and shows the number of tuples of each table in function of the scale factor (SF).

2.2.3 Variants of TPC-H Benchmark

Hereafter, we present and discuss two variants of TPC-H, namely La Brie et al. [27] who conducted MDX performance evaluation and the *Star Schema Benchmark* by O'Neil et al. [16].

La Brie et al. [27] work aims to examine the differences between the optimization techniques that database designers need to consider when developing relational versus multidimensional data warehouses. La Brie et al. demonstrate, by performance measurement, the capabilities of MS Analysis Services performances, compared to MS SQL Server with indexes usage. They conclude that database designers must shift from the index paradigm for relational databases to the aggregate paradigm for dimensional databases. La Brie et al. propose the design of one OLAP cube for the 4th query of TPC-H benchmark (Q4), and report good performance results showing outperformance of MDX compared to SQL on MS Analysis Services.

O'Neil et al. [16] propose the *Star Schema Benchmark* (SSB), which is a variation of TPC-H benchmark leading to a truly star-schema database design. The main changes to the TPC-H database schema are,

- De-normalization through combination of TPC-H re-

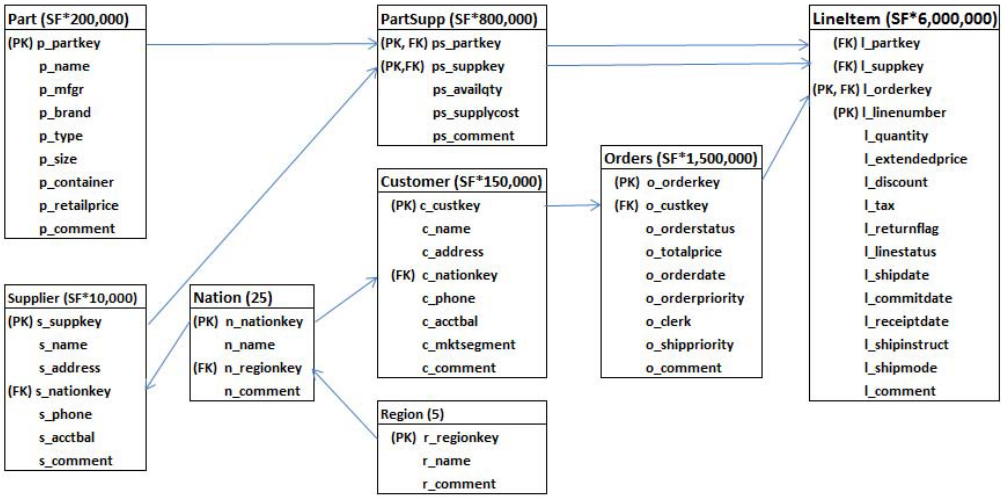


Figure 1: Relational Database Schema of TPC-H Benchmark.

lations. De-normalization renders many joins unnecessary and consequently improves performances. Thus,
  - LINEITEM and ORDERS tables are combined into SALES fact table,
  - CUSTOMER, NATION and REGION are combined into CUSTOMER table,
  - SUPPLIER, NATION and REGION are combined into SUPPLIER table.
- Drop of PARTSUPP table, arguing it should belong to a different data mart.
- Deletion of useless attributes which are either never invoked in the workload (such as *l_comment*) or considered uninteresting for a decision workload (such as *o_comment* invoked in Q13 and *l_shipinstr* invoked in Q19). These changes reduce space requirements and improve performances.
- Creation of the DATE dimension table, as is standard for a warehouse and time series analysis.

Obviously, changes of the relational schema of the data warehouse imply changes to the workload. Some business queries are dropped since they involve no anymore existing tables or attributes. We think that the *star schema benchmark* focuses on the transformation of TPC-H database schema into a star schema warehouse, dropping all snowflake dimensions through de-normalization. Added to that, they reduced the TPC-H benchmark workload by half. Indeed, SSB workload is composed of 12 SQL queries.

2.3 Contribution

It is commonly claimed that for complex queries, OLAP cubes allow fast data retrieval. The most important mechanism in OLAP which allow to achieve such performance is the use of aggregations as well as *approximation methods* (e.g., [2, 4]). Aggregations are built from the fact table by changing the granularity on specific dimensions and aggregating up data along these dimensions [21]. In this paper, we propose a framework for MDB design. In order to prove the effectiveness and the reliability of our proposal, we tested the framework over TPC-H benchmark. The latter is the most prominent DSS benchmark. Our framework, along the experience of turning the TPC-H benchmark into a multidimensional benchmark, resolves design issues which were not raised in related work [14, 9, 19, 12, 11, 22]. We succeed in (i) revolving TPC-H benchmark into a multidimensional benchmark TPC-H*d [23], and (ii) assessing the capacities of the ROLAP server Mondrian[17] with TPC-H*d benchmark. The new benchmark TPC-H*d can be used to compare a vast number of OLAP servers' implementations. Indeed, the proposed cubes could also be built using a MOLAP server.

3. MULTIDIMENSIONAL DESIGN

The term *On-line Analytical Processing* (OLAP) is introduced in 1993 by E. Codd. This model constitutes a decision support system framework which affords the ability to calculate, consolidate, view, and analyze data according to multiple dimensions. OLAP relies heavily upon a data model known as the *multidimensional databases* (MDB) [26]. Compared to relational databases, MDB increase performance by storing aggregated data and enhance data presentation. In this Section, we first recall principles of multidimensional design. Then, we present our framework.

3.1 Principles of Multidimensional Design

Next, we first recall the structure of an *OLAP cube*. Then, we briefly introduce *OLAP operations* and *MultiDimensional eXpressions language*.

3.1.1 OLAP Cube

An *MDB schema* contains a logical model consisting of *OLAP cubes*. An *OLAP Cube* is characterized by a *fact table* (facts), a set of *dimensions* and a set of *measures*. Next, we briefly define these concepts.

- *Facts*: a *fact table* consists of facts of a business process. For example, for a company which sells products to customers. Every sale is a fact that happens, and the fact table is used to record these facts.
- *Measures*: Each *measure* quantifies items such as costs, revenues or units of service, that are counted, summarized or aggregated. For this purpose, *measures* use

appropriate aggregate functions such as: *sum*, *average*, *count*, *count-distinct*, and so on.

- *Dimensions*: Dimensions are variables by which measures are summarized. Each *dimension* is composed of *levels*. The *levels* of a dimension are organized as a hierarchy, i.e. a set of parent-child relationships, typically where a parent member summarizes its children. For instance, a *time dimension* could include the following levels: *Year*, *Quarter*, *Month*, *Week*, and *Day*. Each *level* may contain *properties*. For example, considering the geographic dimension of sales being *Region, Country, City, Store*. The *Country* level might be described by properties such as *area*, *currency*, *population* and *time zone* of the country.

3.1.2 OLAP Operations

Along Codd et al., [1] an *OLAP query enables a BI analyst to easily and selectively extract and view data from different points of view* (refer to E. Codd seminal paper [1] for OLAP explanation). OLAP tools enable users to analyze multidimensional data interactively from multiple perspectives. OLAP consists of five basic analytical operations, namely *roll-up*, *drill-down*, *slice*, *dice* and *pivot*.

- *Roll-up* operation involves the aggregation of data that can be accumulated and computed in one or more dimensions. For instance, a *roll-up* shows *average sales' revenue per country* instead of *average sales' revenue* using the *hiearchy of the customer geographic dimension*: *country* > *city* > *store*.
- *Drill-down* operation allows users to navigate through the details. For instance it shows *average sales' revenue* using the *hiearchy of the customer geographic dimension*: *country* > *city* > *store* instead of *average sales' revenue per country*.
- *Slice* operation allows picking a rectangular subset of a cube by choosing a single value for one of its dimensions, creating a new cube with one fewer dimension. For instance, if the cube calculates *average sales' revenue per country per year-quarter per category of product*, a slice shows *average sales' revenue per year-quarter per product category for a given country*.
- *Dice* operation produces a subcube by allowing the analyst to pick specific values of multiple dimensions. For example, if the cube calculates *average sales' revenue per country per year-quarter per product category*, a dice shows *average sales' revenue per month per category of product for a given country and a given year*.
- *Pivot* operation allows an analyst to rotate the cube in space. For instance, a dimension switches from horizontal to vertical axis, to see another perspective on the data.

Microsoft proposed the query language *MultiDimensional eXpressions* (MDX). The latter provides functionality for creating and querying OLAP cubes. MDX is now a non-proprietary standard and is the most widely supported query language for querying multidimensional database systems. It is supported by many OLAP technologies namely, Microsoft Analysis Services, Hyperion Essbase, Mondrian OLAP server, Palo and IBM Infosphere Warehouse Cubing Services et cetera.

3.2 Proposed Framework

We propose automating MDB conceptual design. First, an *initial schema* is formed. The initial schema consists of all the cubes required to efficiently answer the user queries. Additional steps aim at tuning the workload and consist in creation and refresh of derived data (i.e., aggregate tables, derived attributes, OLAP indexes) or creation of virtual cubes.

We devise the *initial schema* as follows: each input business query -presented in its SQL statement template, is analyzed in order to infer relevant multidimensional knowledge. Thus, *measures*, *facts*, *dimensions* are identified. Our framework consists of three main steps for initial MDB schema design. Next, we detail the three steps:
Step 1 -Measures Identification: Measures use aggregate functions such *count, count-distinct, maximum, minimum, sum, average, median, variance*. *Measures* appear in the SELECT clause.
Step 2 -Fact Table Identification: The *fact table* is the table containing all attributes invoked in measures. Consequently, if a measure involves attributes from different tables, the *fact table* is the combination of these tables. Notice that tables are combined using referential constraints. Moreover, a *fact table* could be filtered along a set of non-parameterized predicates which appear in the WHERE clause.
Step 3 -Dimensions' Composition: First, we extract (i) all attributes from the SELECT clause not invoked in measures, (ii) all attributes in the WHERE clause not used for joining tables and not used for pair-columns comparisons and (iii) all attributes in the GROUP-BY clause. *Second*, attributes are split into groups along their source tables. Within the same group, we distinguish *levels* from *properties* using functional dependencies. Indeed, all properties are in functional dependency with a unique level attribute. *Third*, we consolidate dimensions' hierarchies using hierarchical relationships (i.e. parent-child relationships). Hence, the groups of identified levels are considered to belong to the same dimension's hierarchy such that the attribute from the parent table precedes the attribute from the child table in the dimension hierarchy.

Having multiple and small cubes results in faster query performance than one big cube. Nevertheless, it induces additional storage cost and CPU computing if the workload is run against OLAP cubes having same fact table and shared dimensions. A virtual cube represents a subset of a physical cube. *Virtual Cubes* are recommended for minimal maintenance cost of OLAP cubes. They allow finding out shared and relevant materialized pre-computed multidimensional cubes. The pairwise-comparisons of N OLAP cubes results into $\frac{N\times(N-1)}{2}$ comparisons. In order to automate OLAP cubes comparisons, we implemented *AutoMDB* [23]. *AutoMDB* parses an XML description of TPC-H*d OLAP cubes. Then, builds matrices, which show the similarities and the differences for each pair of OLAP cubes. Similarities and differences are based on comparing fact tables, and counting (i) the number of shared dimensions, (ii) the number of different dimensions, (iii) the number of possibly coalescable dimensions, (iv) the number of shared measures, (v) the number of different measures, and (vi) the number of possibly derivated measures.

4. TPC-H*$_D$: THE MULTIDIMENSIONAL TPC-H BENCHMARK

In this Section, we first present the relational schema

of TPC-H*d benchmark (§4.1), then we present the multidimensional schema consisting of TPC-H*d OLAP cubes (§4.2). Optimizations based on derived data are investigated in §4.3 and examples of virtual cubes are presented in §4.4. The proposed benchmark is available for download [23].

4.1 TPC-H*d Relational Schema

Bill Inmon defined a data warehouse as *a collection of subject-oriented, integrated, non-volatile, and time variant data to support management decisions. Time dimension* is very important for time series analysis. Consequently, it is necessary to store computed values such *year, semester, quarter, month, week, day, week-end, holiday, ...* instead of ordinal date formats. The database schema of TPC-H*d benchmark is illustrated in Fig.2. The few changes to the original TPC-H database schema committed are listed below,

- Create TIME table, for capturing the time dimension. The required levels are defined with respect to TPC-H workload, and are *year*, *quarter*, *month* and *full date*.
- Add *c_countrycode* attribute to CUSTOMER table. This attribute is required by Q22 -the 22^{nd} business query of TPC-H benchmark workload. The attribute values are extracted from *c_phone* attribute,
- Alter LINEITEM and ORDERS tables. First, we create four attributes, respectively one replacing *o_orderdate* and three replacing *l_commitdate*, *l_shipdate*, *l_receiptdate*, in respectively ORDERS and LINEITEM tables, Then, we set the attributes' values to corresponding time keys from TIME dimension table. Finally, referencial constraints on these attributes are enabled.

4.2 TPC-H*d Multidimensional Schema

Each business query is mapped to a minimal number of OLAP cubes. We design each OLAP cube with the relevant fact table, dimensions and measures. This leads to the definition of multiple and small cubes. Hereafter, we detail the process leading to the definition of each cube.

4.2.1 Measure Definition

Measures use aggregate functions such as *count, count-distinct, maximum, minimum, sum, average, median, variance*. We distinguish three types of measures,

- A *Simple Measure* is defined over a single attribute,
- A *Measure Expression* is defined over multiple attributes,
- A *Calculated Member* combines multiple measures.

```
SELECT  l_returnflag, l_linestatus,
        sum(l_quantity) as sum_qty,
        sum(l_extendedprice) as sum_base_price,
        sum(l_extendedprice*(1-l_discount)) as sum_disc_price,
        sum(l_extendedprice*(1-l_discount)*(1+l_tax)) as sum_charge,
        avg(l_quantity) as avg_qty,
        avg(l_extendedprice) as avg_price,
        avg(l_discount) as avg_disc,
        count(*) as count_order
   FROM lineitem
   WHERE l_shipdate <= date '1998-12-01' - interval '[DELTA]' day
   GROUP BY l_returnflag, l_linestatus
   ORDER BY l_returnflag, l_linestatus;
```

Figure 3: SQL Statement of Business Query Q1.

Example 1. Business query Q1 -*Pricing Summary Report Query* provides a summary pricing report for all lineitems shipped as of a given date, and grouped by *line return flag* and *line status*. The SQL statement of Q1 is illustrated in Fig.3 and measure expressions are framed in blue. The extracted measures are listed below,

- Simple measure sum_qty defined as $sum(l\_quantity)$,
- Simple measure sum_base_price defined as $sum(l\_extendedprice)$,
- Measure expression sum_disc_price defined as $sum(l\_extendedprice * (1 - l\_discount))$,
- Measure expression sum_charge defined as $sum(l\_extendedprice * (1 - l\_discount) * (1 + l\_tax))$,
- Simple measure $count\_order$ defined as $count(*)$;
- Calculated Member avg_qty defined as the quotient $\frac{sum\_qty}{count\_order}$,
- Calculated Member avg_price defined as the quotient $\frac{sum\_base\_price}{count\_order}$,
- Simple measure sum_disc defined as $sum(l\_discount)$. Notice that this measure is not required by the business query, and might be declared not visible to the business analyst. *sum* and *count* measures are mandatory for performing OLAP operations (i.e., drill-down, roll-up, slice and dice) and *average* measures.
- Calculated Member avg_disc defined as the quotient $\frac{sum\_disc}{count\_order}$,

4.2.2 Fact Table Definition

The *fact table* is the table containing all attributes invoked in measures. For instance, the fact table of OLAP cube C1, which SQL statement is illustrated in Fig.3, is LINEITEM table. However, the study of TPC-H workload revealed two exception cases. The first case relates to the definition of a fact table from multiple tables and is explained in *Example 2*. The second case relates to the definition of a fact table taking into account the query's predicates and is explained in *Example 3*.

Example 2. Business query Q9 -*Product Type Profit Measure Query*, determines how much profit is made on a given line of parts, broken out by supplier nation and year. The measure expression sum_profit, extracted from Q9 SQL statement (illustrated in Fig.4), is defined as:
$sum(l\_extprice \times (1 - l\_disc) - ps\_suppcost \times l\_qty)$.
Notice that sum_profit involves attributes from two different tables, namely (i) attributes *l_extprice*, *l_disc* and *l_qty* which belong to LINEITEM table (highlighted in yellow); and (ii) *ps_suppcost* which belongs to PARTSUPP table (highlighted in blue). Thus, the fact table of Cube C9 should combine both LINEITEM and PARTSUPP tables. It also selects only required attributes, those invoked in measures and those required for performing joins with dimension tables. The algebraic expression of the fact table for OLAP cube C9 is the following:

$$\prod_{\substack{\{l\_partkey, l\_ext.price,\\ l\_disc, l\_qty, l\_orderkey,\\ l\_suppkey, ps\_suppcost\}}} lineitem \underset{\substack{\text{l\_suppkey=ps\_suppkey}\\ \text{and l\_partkey=ps\_partkey}}}{\bowtie} partsupp$$

Example 3. Business query Q16 -*Parts/Supplier Relationship Query* counts the number of suppliers who can supply parts that satisfy a particular customer's requirements. In Fig.5, the measure *supplier_cnt* -defined as,
count(distinct ps_suppkey) and table PARTSUPP (source of *ps_suppkey* attribute) are highlighted in blue. Notice that, the business query retreives parts not from a supplier who has had complaints registered at the Better Business Bu-

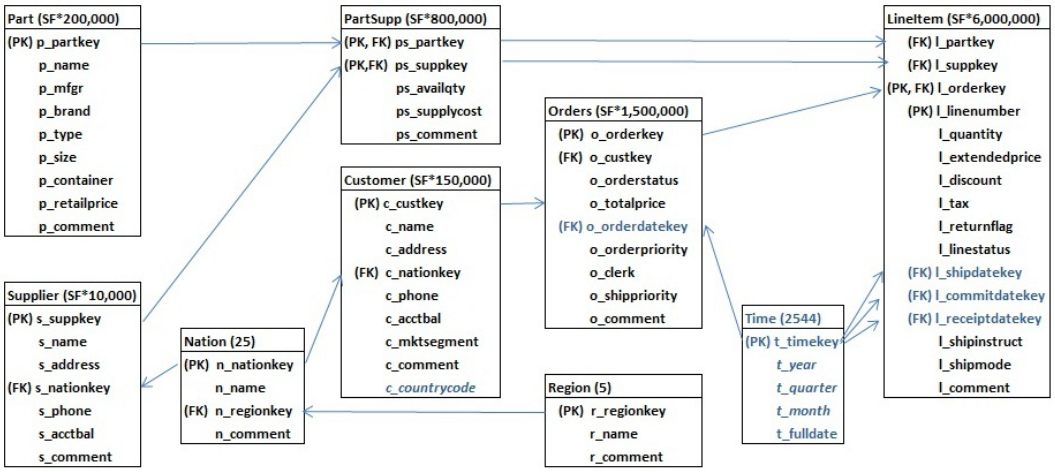


Figure 2: Relational Database Schema of TPC-H*d Benchmark.

```
SELECT nation, o_year, sum(amt) as sum_profit
FROM (SELECT  n_name as nation,
             extract(year from o_orderdate) as o_year,
             l_extprice * (1 - l_disc)-ps_suppcost * l_qty as amt
     FROM part, supplier, lineitem, partsupp, orders, nation
     WHERE s_suppkey = l_suppkey
       AND ps_suppkey = l_suppkey
       AND ps_partkey = l_partkey
       AND p_partkey = l_partkey
       AND o_orderkey = l_orderkey
       AND s_nationkey = n_nationkey
       AND p_name like '%[COLOR]%') as profit
GROUP BY nation, o_year
ORDER BY nation, o_year desc;
```

Figure 4: SQL Statement of Business Query Q9.

reau. The predicate shown in a blue box in Fig.5 is used to filter facts from PARTSUPP table and allows selection of suppliers who has not had complaints. Thus, the algebraic expression of the fact table for OLAP cube C16 is the following:

$$\underset{\substack{\text{ps\_suppkey not in}\\ \text{(select s\_suppkey from supplier where}\\ \text{s\_comment like '\%customer\%complaints\%')}}}{\sigma} partsupp$$

```
SELECT p_brand, p_type, p_size, count(distinct ps_suppkey) as supp_cnt
FROM partsupp, part
WHERE p_partkey = ps_partkey
AND p_brand <> ['Brand']
AND p_type not like ['Type%']
AND p_size in ([S1], [S2], [S3], [S4], [S5], [S6], [S7], [S8])
AND ps_suppkey not in (select s_suppkey
                       from supplier
                       where s_comment like '%Customer%Complaints%')
GROUP BY p_brand, p_type, p_size
ORDER BY supplier_cnt desc, p_brand, p_type, p_size;
```

Figure 5: SQL Statement of Business Query Q16.

4.2.3 *Dimension Definition*

Next, we apply our proposed framework's rules on business query Q10, and describe an exception processing case related to a dimension defined from a view.

Example 4. The OLAP cube C10 is defined as a transform of Q10 -*Returned Item Reporting Query*, into an OLAP cube. The SQL statement of Q10 is illustrated in Fig.6. Q10 identifies customers who might be having problems with the parts that are shipped to them, and who have returned parts. The query considers only parts that were ordered in a specified quarter of a year. The OLAP cube computes the measure all lost revenues per customer (customer details and customer nation name:n_name) and per order date. The measure expression *Revenue* is defined as follows $sum(l\_extentedprice \times (1 - l\_discount))$. It is calculated over LINEITEM facts satisfying l_returnflag = 'R'. Notice that, for OLAP cube C10, *Line Return Flag* can be considered as a dimension, and in this case the fact table is LINEITEM. A more refined solution consists in filtering LINEITEM table along the predicate $l\_returnflag$ = *'R'*. Then, the algebraic expression of the fact table corresponding to OLAP cube C10 is the following,

$$\underset{\text{l\_returnflag = 'R'}}{\sigma} lineitem$$

Notice that OLAP cube C10 aggregates *lost revenue* by customer: its nation (n_name) and its details ($c\_custkey$, c_name, $c\_acctbal$, $c\_address$, c_phone and $c\_comment$), as well as by order date. Next, we detail the process leading to the definition of both *Order Date dimension* and *Customer dimension*. First of all, in Fig.6, non selected attributes are striped in red, and the rest of attributes are highlighted in different colors along their source table.

- The *Order Date dimension* hierarchy derives from TIME table. The latter is a snowflake dimension table reached through ORDERS table (see Fig.2: LINEITEM >> ORDERS >> TIME). The *Order Date dimension* hierarchy is composed of two levels which are *order year* (typical values are 1992, 1993, .., 1998) and *order quarter* (typical values are Q1, Q2, Q3 and Q4). Predicates used to derive required time dimension hierarchy levels are highlighted in blue in Fig.6.

- The *Customer dimension* is composed as follows: the attribute *c_custkey* is a level within *customer dimension hierarchy*, and has properties: *c_name*, *c_acctbal*, *c_address*, *c_phone* and *c_comment*. These attributes are in functional dependency with *c_custkey*. Indeed, $\{c\_custkey\} \longrightarrow \{c\_name, c\_acctbal, c\_address, c\_phone, c\_comment\}$. Since, it exists a hiearchical relationship between CUSTOMER table and NATION table, *Customer Nation* is a snowflake dimension and *n_name* is a level within *customer dimension hierarchy*. Also, within *customer dimension hierarchy Customer nation level -n_name*, is prior to *Customer key level -c_custkey*, because NATION is a parent table of CUSTOMER table. In Fig.6, *Customer nation level* derives from text highlighted in green, while *Customer key level* and its properties namely: *c_name*, *c_acctbal*, *c_address*, *c_phone* and *c_comment* derive from text highlighted in yellow.

```
SELECT c_custkey,c_name,c_acctbal, n_name, c_address, c_phone,
       c_comment, SUM(l_extendedprice*(1-l_discount)) as rev
FROM customer, orders, lineitem, nation
WHERE c_custkey = o_custkey
  AND l_orderkey = o_orderkey
  AND o_orderdate >= date '[DATE]'
  AND o_orderdate < date '[DATE]' + '3' month
  AND l_returnflag = 'R'
  AND c_nationkey = n_nationkey
GROUP BY c_custkey,c_name,c_acctbal,c_phone,n_name,c_address,c_comment
ORDER BY revenue desc;
```

Figure 6: SQL Statement of Business Query Q10.

Example 5. Multidimensional data retrieval mode is not intended for comparing columns to each other, or to *exists/ not exists* usage. Hence, such predicates are included in views. The views are used as data sources for facts (as shown in *Example 3*), as well as for dimensions (as shown in this example). Business query Q12 (illustrated in Fig.7), counts the number of urgent and high priorities orders (i.e. *high_line_count* measure), and the number of not urgent and not high priorities orders (i.e. *low_line_count* measure) per *line ship mode* for a given *line receipt year*. Thus, the fact table is ORDERS and dimensions are *line_ship_mode* and *line_receipt_year*. Notice that Q12 counts distinct orders which related lines verify (*l_commitdate* < *l_receiptdate* AND *l_shipdate* <*l_commitdate*). Consequently, both dimensions are defined from a subset of LINEITEM table as follows:

$$\underset{\substack{\text{l\_commitdate} < \text{l\_receiptdate} \\ \text{and l\_shipdate} < \text{l\_commitdate}}}{\sigma} lineitem$$

4.3 Optimizations based on Derived Data

Design, implementation and optimizations should be developed based upon business needs. Thus, an understanding of the business workload is necessary for performance tuning. Derived data, namely (i) aggregate tables (a.k.a. materialized views), (ii) calculated attributes, (iii) Indexes and (iv) data synopsis are well known techniques for performance tuning. In this paper, we do not discuss indexes and data synopsis usage. We propose the classification of OLAP business queries, along two variables, namely *dimensionality* (whether the cube size is scale factor dependent or not?)

```
SELECT l_shipmode,
    SUM(case when o_orderpriority ='1-URGENT' or ='2-HIGH' then 1
             else 0 end) as high_line_count,
     SUM(case when o_orderpriority != '1-URGENT' and != '2-HIGH then 1
             else 0 end) as low_line_count
FROM orders, lineitem
WHERE o_orderkey = l_orderkey
  AND l_shipmode in ('[SHIPMODE1]', '[SHIPMODE2]')
  AND l_commitdate < l_receiptdate
  AND l_shipdate < l_commitdate
  AND l_receiptdate >= date '[DATE]'
  AND l_receiptdate < date '[DATE]' + '1' year
GROUP BY l_shipmode
ORDER BY l_shipmode;
```

Figure 7: SQL Statement of Business Query Q12.

and *sparsity* (whether the cube is very sparse or not?). Each variable has two categorical values, and this results into four types of OLAP business queries. The workload taxonomy will motivate the choice of appropriate derived data type among aggregate tables and calculated attributes.

4.3.1 OLAP Cube Characteristics

Next, we present two OLAP cube characteristics, namely *dimensionality* and *sparsity*,

Dimensionality: The OLAP cube size is calculated using both (i) cardinalities of dimensions and (iii) number of measures. Dimensionality is either data warehouse scale factor dependent or not, i.e., when the data warehouse becomes larger, does dimensionality of OLAP cubes becomes larger or is the same? Below, we give two examples of queries,

Example 6. Business query Q4 –*The Order Priority Checking Query*, counts the number of orders placed in a given quarter of a given year in which at least one lineitem was received by the customer later than its committed date. The query lists the count of such orders for each order priority. The OLAP cube has two dimensions, namely (i) the *order date dimension*, which hierarchy is bi-level and composed of: *order_year level* and *order_quarter level* and (ii) *order priority dimension*, which hierarchy is mono-level. C4 size is TPC-H scale factor independent and always equal to 135 ($\sharp$ *order priorities*:5 $\times$ $\sharp$ order years:7 $\times$ $\sharp$ quarters/year:4).

Example 7. Business query Q15 –*The Top Supplier Query*, finds suppliers who contributed the most to the overall revenue for parts shipped for a given quarter of a given year. In case of a tie, the query lists all suppliers whose contribution was equal to the maximum. The cube processing includes the calculus of total revenue for each supplier, $\sum$*l_extendedprice* $\times$ (1-*l_discount*) per *shipping_year* and *shipping_quarter*, as well as the maximum revenue recorded per year/quarter period of time. C15 size is SF$\times$28,000 (*line_ship_year*:7 $\times$ $\sharp$ *quarters/year*:4 $\times$ $\sharp$ *suppliers*: SF$\times$10,000). Notice that the size of C15 is TPC-H scale factor dependent.

Sparsity: In OLAP cube, cross products of dimensional members form the intersections for measure data. But in reality, most of the intersections will not have data. This leads to compute the sparsity (or inversely the density) of the multidimensional OLAP cube. We propose two categorical values, namely *very sparse* and *acceptable sparsity*.

Example 8. Business query Q18 –*The Large Volume Customer Query*, finds a list of customers who have ever placed large quantity orders (i.e., $\sum$ *l_quantity* $\geq$ 300). The size of C18 is the $\sharp$ *orders*: SF $\times$ 1,500,000. Nevertheless, 3.8ppm

(parts per million) of orders are big orders. Thus, we consider OLAP cube C18 as very sparse.

Example 9. Business query Q2 –*The Minimum Cost Supplier Query* finds, in a given region, for a part of a certain type and size, suppliers who can supply it at minimum cost, with a preference to suppliers having highest account balances. The cross product of dimensions *part_type*, *part_size* and *supplier_region* does not present empty for more than 98% of combinations.

4.3.2 Recommendations

Hereafter, we briefly recall definitions of *aggregate tables* and *derived attributes*, and motivate their usage for each type of business query of TPC-H workload.

Aggregate Tables:(a.k.a, materialized view), an aggregate table summarizes large number of detail rows into information that has a coarser granularity. As the data is precomputed, an aggregate table allows faster cube processing. We recommend aggregate tables for business queries having a dimensionality, which is scale factor independent (i.e. fixed number of rows as Q4), and also for business queries having very sparse cubes (i.e. return few rows and most dimensions' combinations are empty as Q18).

Derived Attributes: (a.k.a. calculated fields), Derived attributes are calculated from other attributes. We recommend derived attributes for OLAP cubes which dimensionality is scale factor dependent, as Q10. Indeed, for this type of business queries, *derived attributes* are much less space consuming than *aggregate tables*. Q10 identifies customers who might be having problems with the parts that are shipped to them, and have returned them, for so, it calculates the *lost revenue* for each customer for a given quarter of a year. In order to improve the response time of Q10, we propose the following alternatives, (1) Either add 28 derived attributes *c_sumLostRev /year/quarter* to CUSTOMER relation, or (2) add one attribute *o_sumLostRev* to ORDERS relation. Notice that, the second alternative is better than the first with respect to both storage overhead and cost of refresh of stale derived attributes. Indeed, following inserts or deletes of orders (respectively TPC-H refresh functions RF1 and RF2), the 28 derived attributes are stale, while refreshes do not render stale the attribute *o_sumLostRev*. As a consequence of derived attributes, the schema of OLAP cube C10 is the following: henceforth, the measure is $sum(o\_sumLostRev)$, the fact table is ORDERS and dimensions are *customer dimension* and *order date dimension* as defined in *Example 4*. The gain in performance results from not performing the join of LINEITEM and ORDERS tables.

We conducted a numerical study over TPC-H benchmark workload. We were interested in computing the maximal number of rows returned and the OLAP cube size. We concluded that TPC-H business queries fall into three categories (see Table 1), for which different sound recommendations are proposed. Notice that, *derived data*, i.e., aggregate tables and derived attributes, could be stale following the execution of warehouse refresh functions. Refreshes are implemented at the business logic using stored procedures.

4.4 Virtual Cubes

Virtual Cubes are recommended for minimal maintenance cost of OLAP cubes. They allow finding out shared and relevant materialized pre-computed multidimensional cubes. We implemented *AutoMDB* for recommending merge of OLAP cubes based on maximum shared properties and minimum different properties [23]. For instance, *AutoMDB* detects that, (i) OLAP cubes C5 and C7 have the same fact table LINEITEM. Moreover, (ii) both cubes calculate the same measure $sum(l\_extendedprice \times (1 - l\_discount))$, and (iii) two dimensions of OLAP cube C7 could be collapsed within dimensions of OLAP cube C5. Hereafter, we describe both dimensions sets of OLAP cubes C5 and C7,

OLAP cube C5 dimensions are the following,

- $D_{C5,1}$ customer geography: *customer_region* > *customer_nation*,
- $D_{C5,2}$ supplier geography: *supplier_region* > *supplier_nation*,
- $D_{C5,3}$ order date: *order_year*.

OLAP cube C7 dimensions are:

- $D_{C7,1}$ customer geography: *customer_nation*,
- $D_{C7,2}$ supplier geography: *supplier_nation*,
- $D_{C7,3}$ item ship date: *ship_year*.

AutoMDB recommends building a physical cube which englobes C5's dimensions namely: $D_{C5,1}$, $D_{C5,2}$, $D_{C5,3}$, C7's dimension $D_{C7,3}$ and C5's measure. Consequently, C5 and C7 are defined as virtual cubes.

5. PERFORMANCE ANALYSIS

Next, we first describe system implementation. Then, we present performance results.

5.1 System Implementation

We revolved TPC-H benchmark into a multidimensional benchmark, and we translated the SQL workload into MDX workload. For each business query, we propose an MDX statement for the query and an MDX statement for the cube. For test, we used MySQL as a relational DBMS, and Mondrian ROLAP server [17]. Mondrian is an open source ROLAP server of Pentaho BI suite. It executes queries written in the MDX language, by reading data from a relational database (RDBMS), and presents the results in a multidimensional format (a.k.a. pivot table) via JPivot. For instance, business query Q10 of TPC-H benchmark -*Returned Item Reporting Query* -which SQL statement template is illustrated in Fig.6, identifies customers who might be having problems with the parts that are shipped to them, and who have returned parts. The query considers only parts that were ordered in a specified quarter of a year. The OLAP cube computes all lost revenues per customer dimension and per order date dimension. The user interacts with the pivot table (via an OLAP dice operation) shown in Fig. 8, in order to retrieve lost revenues for French customers during first quarter of 1992. Fig. 8 shows excerpts of MDX statements corresponding respectively to Cube C10 and Query Q10.

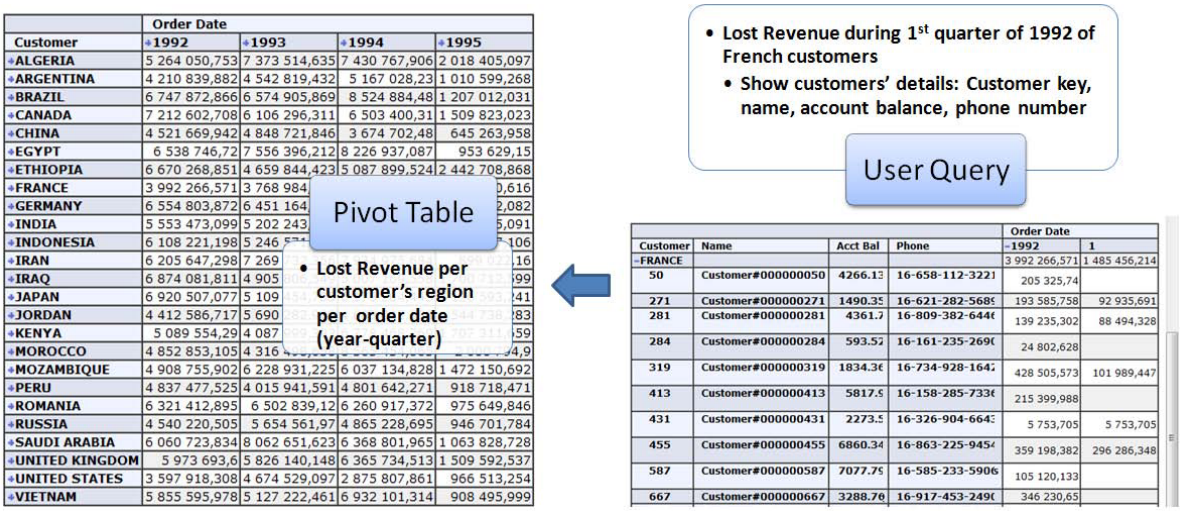
Fig. 9 shows the system architecture under test. We make use of the following software: (i) Schema Workbench to generate the multidimensional database schema in XML, (ii) Mondrian as ROLAP Server, (iii) Apache Tomcat as JSP Container, (iv) JPivot as OLAP Client, and MySQL5 as DBMS back-end.

5.2 Performance Results

The hardware system configuration used for performance measurements are Adonis/ Edel nodes located at Grenoble site of GRID5000. Each node has 24 GB of memory, its CPUs are Intel Xeon E5520, 2.27 GHz, with 2 CPUs per node and 4 cores per CPU, and run Lenny Debian Operating

Type	Dimensionality	Sparsity	TPC-H Business Queries	Recommendation
A	SF dependent	very sparse	Q15, Q18	Aggregate Tables
B	SF dependent	dense enough	Q2, Q9, Q10, Q11, Q20, Q21	Derived Attributes
C	SF independent	very sparse	—	Aggregate Tables
D	SF independent	dense enough	Q1, Q3, Q4, Q5, Q6, Q7, Q8, Q12, Q13, Q14, Q16, Q17, Q19, Q22	Aggregate Tables

Table 1: TPC-H Workload Taxonomy.




```
SELECT [Order Date].members ON COLUMNS,
    [Customer].Members ON ROWS
FROM [Cube10]
```

```
SELECT {[Order Date].[1992].[1]} ON COLUMNS,
 {[Customer].[FRANCE]} ON ROWS
FROM [Cube10]
```

Figure 8: Screenshots of Pivot Tables of C10 and Q10 and corresponding MDX statements.

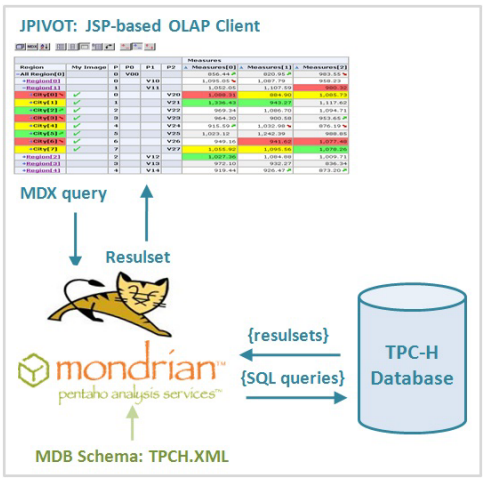


Figure 9: System under test.

System.

For experiments, the client sends a stream of MDX queries in a random order to the database tier, and measures performance of MDX queries for two different workloads. The first workload stream is a *Query workload*. It is composed of TPC-H queries translated into MDX ((i.e., Qi, Qj, ...)), while the second is a *Cube-then-Query workload*. It is composed of TPC-H*d cubes' MDX statements followed by Queries' MDX statements (i.e., Ci-Qi, Cj-Qj, ...). Second workload type should allow query result retrieval from built cubes and consequently, it is expected to lead to better performance results. Table 2 and Table 3 show respectively detailed performance results for SF=1 and SF=10. Response times are measured over 3 runs, and the variance is negligible.

Experiments show that,

- Cube building is memory consuming, run exceptions are, either related to memory leaks (OutOfMemoryError) or Mondrian limits (Size of CrossJoin result exceeded limit 2,147,483,647).
- SQL outperforms MDX, except for queries retreiving responses from built cubes (*cube-then-query workload* type), for which, data is aggregated and measures are pre-computed and are in-memory.
- For some queries, cube building is not improving performances such Q2. These MDX queries include new members calculus (i.e., measures or named sets), and perform filtering on levels' properties (readers could check MDX statements available on-line [23]).
- For SF=1, most cubes allow fast data retrieval after their deployment. Elapsed times fo running following business queries are better compared to SQL after cube building as Q1, Q4-Q8, Q14, Q16, Q17, Q19 and Q21. For Q3, Q9-Q11, Q13, Q15, Q18 and Q22, their respective response times were improved compared to query workload execution, but are still not competing with SQL. Overall, for SF=1, improvements vary

	SQL Workload (sec)	MDX Workload (sec)			Enhanced MDX Workload (sec)		
		Query Workload	Cube-then-Query Workload		Query Workload	Cube-then-Query Workload	
			Cube	Query		Cube	Query
Q1	7.36	54.90	85.25	0.16	0.56	0.68	0.19
Q2	0.26	36.07	26.48	34.88	255.18	n/a^{*1}	-
Q3	1.77	79.94	976.17	37.20	59.75	934.84	31.17
Q4	1.72	14.27	114.53	0.33	0.07	0.06	0.05
Q5	47.42	8.42	157.68	0.38	0.09	0.66	0.04
Q6	1.03	15.78	18.22	0.27	0.39	0.62	0.31
Q7	1.81	5.24	11.88	0.04	0.08	0.88	0.05
Q8	1.65	5.04	45.16	0.61	0.31	3.27	0.23
Q9	3.16	305.83	1195.62	56.98	431.80	1,007.25	54.16
Q10	1.86	109.14	288.51	2.95	35.38	120.03	2.29
Q11	1.01	55.18	55.89	31.33	54.53	56.58	31.04
Q12	1.50	16.52	40.23	8.82	0.06	0.10	0.05
Q13	4.05	203.15	882.80	16.79	0.12	0.35	0.03
Q14	1.20	1.77	24.89	0.04	0.05	0.06	0.05
Q15	2.87	369.05	18,829.68	369.17	0.01	-	-
Q16	0.63	14.28	35.69	0.91	1.71	5.05	0.60
Q17	1.30	3.42	34.07	0.18	0.06	0.13	0.05
Q18	3.52	3,515.31	3,340.55	2,167.40	0.01	-	-
Q19	1.98	48.23	50.45	1.43	2.27	4.55	0.22
Q20	2.13	127.36	n/a^{*1}	-	47.48	n/a^{*1}	-
Q21	4.75	9.62	99.42	0.05	0.29	3.44	0.08
Q22	0.43	5.63	32.46	2.63	4.33	20.78	1.12

• n/a^{*1}: java.lang.OutOfMemoryError: GC overhead limit exceeded.

Table 2: Performance results for SF = 1.

from 81.37% to 99.71%, for Q1, Q4-Q8, Q12, Q13, Q14, Q16-Q19 and Q21.

- For SF=10, most cubes allow fast data retrieval after their deployment. Elapsed times fo running following business queries are better compared to SQL after cube building as Q1, Q4-Q8, Q12, Q14, Q16, Q17 and Q21. Also, the system under test was unable to build cubes related to business queries: Q3, Q9, Q10, Q13, Q18, and Q20. For Q11, Q15, Q19 and Q22, their respective response times were improved compared to query workload execution, but are still not competing with SQL. Overall, for SF=10, improvements vary from 42.78% to 100%, for Q1, Q4, Q5, Q6, Q7, Q8, Q12, Q13, Q14, Q16, Q17, Q19 and Q21.
- Business queries, for which aggregate tables were built, namely (Q1, Q3-Q8, Q12, Q13, Q14, Q15, Q16, Q17, Q18), were improved. Aggregate tables allow very fast building of cubes, in some cases very close to query response times, and this allows fast navigation within the muldimensional structure.
- Since, adding derived attributes changes the cube schema and saves join operations processing, the results illustrated in Table 4 show good improvements, except for Q2. For the latter, the response time of the SQL statement rewritten using the *ps_isminimum* derived attributes reduces the response time to less than 1sec. Notice that C2 is a very dense cube with a very high dimensionality.
- Ideally, the system under test maintains same performance results for any SF. Experiments show that the ratio of SF=10 performance results to SF=1 performance results even for SQL is higher than 10 for most business queries of TPC-H workload. Indeed the average *SQL SF=10 - SQL SF=1* ratio is equal to 135 . With MDX, the average *MDX SF=10 - MDX SF=1* ratio is 70 for *query workload* with no optimizations, and 36.36 for *cube workload* with no optimizations. Since, the ROLAP server translates MDX statements into SQL statements, obtained results show that the OLAP engine is bad performing. With derived data, and especially with aggregate tables having same size for both SF=1 and SF=10, as for business queries: Q1, Q4, Q5, Q6, Q7, Q8, Q12, Q13, Q14,Q15, Q16, Q17 and Q19 (see Table 5), the average ratio becomes almost 1.8 for query workload, and it is 1.4 for cube workload. For business query Q10, the derived attribute *o_lostrevenue* decreases the ratio from 65.06 to 9.3 for query workload. Ditto, for business query Q21, the derived attribute *s_nbrwaitingorders* decreases the ratio from 60.06 to 7.86 for query workload. Some derived attributes allow the system to become sub-linear, while with aggregate tables, the system under test is ideal.

The cost of derived data is threefold (i) storage overhead, (ii) computing overhead and (iii) refresh overhead. Aggregate tables were proposed for business queries which related OLAP cubes have a dimensionality independent of scale factor or are very sparse cubes. Elapsed times for aggregate tables creation vary along the business query complexity.

For aggregate tables, which can be refreshed incrementally, we load *RF-1 stream new sales* and *RF-2 stream old sales*, into temporary tables, create delta-aggregate tables. Then, run PL/SQL code in order to refresh the aggregate tables. *RF-1 new sales* is composed of 4,500 × SF new orders and their related new item lines (17,981 × SF lines), while *RF-2 old sales* is composed of 4,500 × SF orders and their related lines for delete. Complete refresh is selected either when it is not possible to refresh incrementally or when it is better performing than incremental refresh. Performance results are reported in Table 5 and Table 6.

Storage requirements for derived attributes are linear to the scale factor. The creation and refresh of derived at-

	SQL Workload (sec)	MDX Workload (sec)			Enhanced MDX Workload (sec)		
		Query Workload	Cube-then-Query Workload		Query Workload	Cube-then-Query Workload	
			Cube	Query		Cube	Query
Q1	211.94	2,147.33	2,778.49	0.29	1.10	1.39	0.21
Q2	459.18	1,598.54	346.92	1,565.51	n/a^{*1}	n/a^{*1}	-
Q3	56.75	n/a^{*1}	n/a^{*1}	-	n/a^{*2}	n/a^{*1}	-
Q4	11.22	1,657.60	7,956.45	5.33	0.11	0.12	0.05
Q5	19.10	54.53	3,200.64	0.46	0.08	1.01	0.48
Q6	38.63	282.11	371.80	0.53	0.49	0.85	0.31
Q7	133.92	260.23	617.20	0.06	0.13	1.39	0.06
Q8	37.18	50.63	2,071.00	4.61	0.46	2.83	0.23
Q9	645.86	n/a^{*1}	n/a^{*1}	-	n/a^{*1}	n/a^{*1}	-
Q10	191.69	7,100.24	n/a^{*2}	-	329.01	n/a^{*1}	-
Q11	4.00	2,558.21	3,020.27	1,604.10	2,738.46	2,723.85	1,585.81
Q12	144.36	456.81	735.67	123.43	0.06	0.07	0.05
Q13	38.68	n/a^{*2}	n/a^{*2}	-	0.08	0.38	0.06
Q14	122.11	391.06	946.16	0.06	0.10	0.11	0.05
Q15	90.97	13,005.27	32,064.90	12,413.74	0.07	-	-
Q16	47.92	414.82	461.90	4.62	1.36	4.7	0.68
Q17	4.22	1,131.37	5,711.14	2.03	0.06	0.25	0.05
Q18	905.16	n/a^{*2}	n/a^{*1}	-	0.30	-	-
Q19	1.56	598.9	727.72	37.57	4.61	4.30	0.16
Q20	1.55	14,662.53	n/a^{*3}	-	2,802.65	n/a^{*4}	-
Q21	511.54	578.09	855.46	0.15	2.28	27.38	0.78
Q22	2.40	68.74	402.16	39.33	69.93	282.18	30.68

- n/a^{*1}: java.lang.OutOfMemoryError: GC overhead limit exceeded,
- n/a^{*2}: java.lang.OutOfMemoryError: Java heap space,
- n/a^{*3}: Mondrian Error:Size of CrossJoin result (200,052,100,026) exceeded limit (2,147,483,647),
- n/a^{*4}: Mondrian Error:Size of CrossJoin result (200,050,000,000) exceeded limit (2,147,483,647).

Table 3: Performance results for SF = 10.

	Impacts for SF=1		Impacts for SF=10	
	Query (%)	Cube (%)	Query (%)	Cube (%)
Q2 -*ps_isminimum*	-514.00	*n/a*	*n/a*	*n/a*
Q9 -*l_profit*	26.68	48.34	*n/a*	*n/a*
Q10 -*o_sumlostrevenue*	72.75	68.76	95.36	*n/a*
Q11 -*n_stockvalue*	13.23	10.94	-7.50	9.80
Q17 -*p_sumqty, p_countlines*	52.57	34.11	99.99	99.99
Q20 -*ps_excess_YYYY*	66.97	*n/a*	80.88	*n/a*
Q21 -*s_nbrwaitingorders*	96.98	96.54	99.60	96.80

Table 4: Impacts of Derived Attributes on Performance Results.

tributes vary along complexity to compute them. The space overhead caused by derived attributes is calculated with respect to the original database volume, respectively 1.3GB for SF=1 and 11.95GB for SF=10.

6. CONCLUSIONS AND FUTURE WORK

Starting from limitations of actual relational database management systems, in this paper we have provided a framework for automating multidimensional database schema design, which embeds several points of research innovation. In order to prove the effectiveness and the reliability of our proposal, we tested it over the well-known TPC-H benchmark as to make it a kind of multidimensional benchmark, TPC-H**d*, along with the transformation of the classical TPC-H workload into suitable MDX queries. The experimental evaluation has been conducted by benchmarking the popular ROLAP server Mondrian and its driver OLAP4j via TPC-H**d*.

Future work is mainly oriented towards two different directions. First, we aim at dealing with the related-context represented by *streaming multidimensional data* (e.g., [3]). Second, we pursue the idea of adpating our framework to the novel and exciting research context represented by so-called *Big Data* (e.g., [5]).

7. REFERENCES

[1] E. F. Codd, S. B. Codd, and C. T. Salley. Providing OLAP (on-line analytical processing) to user-analysts: An IT mandate. *Codd and Date*, 32:3–5, 1993.

[2] A. Cuzzocrea. Providing probabilistically-bounded approximate answers to non-holistic aggregate range queries in olap. In *DOLAP*, pages 97–106, 2005.

[3] A. Cuzzocrea. Retrieving accurate estimates to olap queries over uncertain and imprecise multidimensional data streams. In *SSDBM*, pages 575–576, 2011.

[4] A. Cuzzocrea and P. Serafino. *LCS*-hist: taming massive high-dimensional data cube compression. In *EDBT*, pages 768–779, 2009.

[5] A. Cuzzocrea, I.-Y. Song, and K. C. Davis. Analytics over large-scale multidimensional data: the big data revolution! In *DOLAP*, pages 101–104, 2011.

[6] J. Darmont, O. Boussaid, and F. Bentayeb. DWEB: A

	SF = 1					SF = 10				
	Nbr of Rows	Data Volume	Build (sec)	RF1 run (sec)	RF2 run (sec)	Nbr of Rows	Data Volume	Build (sec)	RF1 run (sec)	RF2 run (sec)
agg_C1	129	16.62KB	10.54	0.11	1.67	129	16.62KB	343.91	0.06	0.34
agg_C3	222,268	12.00MB	16.65	0.11	2.07	2,210,908	103.32MB	173.45	0.17	41.60
agg_C4	135	5.22KB	5.41	0.08	1.28	135	5.22KB	138.45	7.49	0.60
agg_C5	4,375	586.33KB	19.73	0.07	4.21	4,375	586.33KB	822.29	4.94	4.18
agg_C6	1,680	84.67KB	4.19	0.59	0.83	1,680	84.67 KB	148.29	0.38	0.90
agg_C7	4,375	372.70KB	19.20	0.25	4.36	4,375	372.70KB	720.26	2.69	2.19
agg_C8	131,250	12.77MB	7.31	0.32	69.61	131,250	12.77MB	2894.38	1,623.03	1,676.79
agg_C12	49	3.15KB	3.22	0.06	1.75	49	3.15KB	186.68	0.10	1.30
agg_C13	656	24.10KB	1,631.72	1,940.06	1,516.74	721	26.33KB	9,819.46	12,334.67	9,124.85
agg_C14	84	6.33KB	9.51	0.13	1.69	84	6.33KB	367.88	0.18	0.80
agg_C15	28	3.84KB	25.90	26.73	26.53	28	3.84KB	10,904.00	16,380.43	10,075.21
agg_C16	123,039	7.00MB	4.29	-	-	187,495	10.03MB	63.05	-	-
agg_C17	1,000	45.92KB	7.78	6.49	6.42	1,000	45.92KB	3,180.26	169.22	220.56
agg_C18	57	4.20KB	3.65	0.07	0.01	624	37.56KB	905.16	23.27	0.49
agg_C19	121,165	11.00MB	3.09	0.45	2.71	854,209	80.65MB	88.57	19.72	45.78
agg_C22	25	1.73KB	0.41	0.47	0.94	25	1.73KB	6.25	4.62	3.08

Table 5: Aggregate tables and related refresh costs.

	SF = 1				SF = 10			
	Over-head (%)	Creation Time (sec)	RF1 run (sec)	RF2 run (sec)	Over-head (%)	Creation Time (sec)	RF1 run (sec)	RF2 run (sec)
ps_isminimum	0.07	52.86	-	-	0.07	862.40	-	-
l_profit	1.20	455.04	44.74	-	4.20	4377.51	400.60	-
o_sumlostrevenue	0.08	90.59	2.30	-	3.05	1027.98	10.85	-
n_stockval	0.00	0.63	-	-	0.00	20.22	-	-
p_sumqty, p_countlines	0.15	25.97	1.36	2.89	0.14	1139,94	1.29	10.32
ps_excess_YYYY	0.60	731.10	759.62	542.9	0.50	18,195.48	20,357.87	17,778.1
s_nbrwaitingorders	0.00	29.47	0.08	5.08	0.00	299.15	38.28	10.04

Table 6: Derived Attributes and related refresh costs.

data warehouse engineering benchmark. In *Proc. of DaWaK*, pages 85–94, 2005.

[7] T. Erik. Comparing different approaches to OLAP calculations as revealed in benchmarks. In *Intelligence Enterprises Database Programming & Design*, 1998.

[8] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Journal of Data Mining and Knowledge Discovery*, 1(1):29–53, 1997.

[9] E. Hung, D. W.-L. Cheung, and B. Kao. Optimization in data cube system design. *Journal of Intelligent Information Systems*, 23(1):17–45, 2004.

[10] D. Leffingwell. *Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise*. Addison-Wesley Professional, 1st edition, 2011.

[11] E. Malinowski and E. Zimányi. A conceptual model for temporal data warehouses and its transformation to the ER and the object-relational models. *Journal of Data Knowledge Engineering*, 64(1):101–133, 2008.

[12] R. Nair, C. Wilson, and B. Srinivasan. A conceptual query-driven design framework for data warehouse. *Intl. Journal of Computer and Information Science and Engineering*, 1(1), 2007.

[13] L. Nicole. Business intelligence software market continues to grow. `www.gartner.com/it/page.jsp?id=1553215`, 2012.

[14] T. Niemi, J. Nummenmaa, and P. Thanisch. Constructing OLAP cubes based on queries. In *Proc. of DOLAP*, pages 9–15, 2001.

[15] OLAP Council. APB-1 benchmark. `www.olapcouncil.org`.

[16] P. O'Neil, E. O'Neil, X. Chen, and S. Revilak. The star schema benchmark and augmented fact table indexing. In *Proc. of TPC-TC*, pages 237–252, 2009.

[17] Pentaho. BI suite: Mondrian OLAP Server. `http://mondrian.pentaho.org/`.

[18] Pringle & Company. Business intelligence software & services market 2011-2016. `http://www.pringleandcompany.com/#/bi-market-2011-2016/4572252894`, 2012.

[19] O. Romero and A. Abelló. Multidimensional design by examples. In *Proc. DaWaK*, pages 85–94, 2006.

[20] O. Romero and A. Abelló. A survey of multidimensional modeling methodologies. *Intl. Journal of Data Warehousing and Mining (IJDWM)*, 5(2):1–23, 2009.

[21] C. Surajit and D. Umeshwar. An overview of data warehousing and OLAP technology. In *SIGMOD Rec.*, volume 26, pages 65–74. ACM, 1997.

[22] P. Thanisch, T. Niemi, M. Niinimäki, and J. Nummenmaa. Using the entity-attribute-value model for OLAP cube construction. In *Proc. of BIR*, pages 59–72, 2011.

[23] TPC-H*d. Multidimensional TPC-H benchmark. `https://sites.google.com/site/rimmoussa/auto_multidimensional_dbs`.

[24] TPC-DS benchmark. `http://www.tpc.org/tpcds`.

[25] TPC-H benchmark. `http://www.tpc.org/tpch`.

[26] P. Vassiliadis. Modeling multidimensional databases, cubes and cube operations. In *Proc. of SSDBM*, pages 53–62, 1998.

[27] L. Ye and R. C. Labrie. A paradigm shift in database optimization: From indices to aggregates. In *Proc. of AMCIS*, pages 29–33, 2002.