Data Variety, Come As You Are in Multi-model Data Warehouses

Data Variety, Come As You Are in Multi-model Data Warehouses

Sandro Bimonte[a], Enrico Gallinucci[b], Patrick Marcel[c], Stefano Rizzi[b,*]

[a]*INRAE - TSCF, University of Clermont Auvergne, Aubiere, France*
[b]*DISI, University of Bologna, Italy*
[c]*LIFAT Laboratory, University of Tours, France*

Abstract

Multi-model DBMSs (MMDBMSs) have been recently introduced to store and seamlessly query heterogeneous data (structured, semi-structured, graph-based, etc.) in their native form, aimed at effectively preserving their variety. Unfortunately, when it comes to analyzing these data, traditional data warehouses (DWs) and OLAP systems fall short because they rely on relational DBMSs for storage and querying, thus constraining data variety into the rigidity of a structured, fixed schema. In this paper, we investigate the performances of an MMDBMS when used to store multidimensional data for OLAP analyses. A multi-model DW would store each of its elements according to its native model; among the benefits we envision for this solution, that of bridging the architectural gap between data lakes and DWs, that of reducing the cost for ETL, and that of ensuring better flexibility, extensibility, and evolvability thanks to the combined use of structured and schemaless data. To support our investigation we define a multidimensional schema for the UniBench benchmark dataset and an ad-hoc OLAP workload for it. Then we propose and compare three logical solutions implemented on the PostgreSQL multi-model DBMS: one that extends a star schema with JSON, XML, graph-based, and key-value data; one based on a classical (fully relational) star schema; and one where all data are kept in their native form (no relational data are introduced). As expected, the full-relational implementation generally performs better than the multi-model one, but this is balanced by the benefits of MMDBMSs in dealing with variety. Finally, we give our perspective view of the research on this topic.

Keywords: OLAP, Multi-Model Databases, Data Variety, Data Warehouse

*Corresponding author
Email addresses: sandro.bimonte@inrae.fr (Sandro Bimonte), enrico.gallinucci@unibo.it (Enrico Gallinucci), patrick.marcel@univ-tours.fr (Patrick Marcel), stefano.rizzi@unibo.it (Stefano Rizzi)

1. Introduction

Big Data are notoriously characterized by the 5 V's: volume, velocity, variety, veracity, and value. To handle velocity and volume, some distributed file system-based storage (such as Hadoop) has been created on the one hand, new Database Management Systems (DBMSs) supporting NoSQL databases have been introduced on the other. Specifically, NoSQL databases are distinguished into four main categories [1]: key-value, extensible record, graph-based, and document-based. Although NoSQL DBMSs have successfully proved to support the volume and velocity features, variety is still a challenge to some extent [2]. Indeed, several practical applications (e.g., in the field of agroecology and health) ask for collecting and analyzing data of different types: structured (e.g., relational tables), semi-structured (e.g., XML and JSON), and unstructured (text, images, etc.). Using the right DBMS for the right data model is essential to grant good storage and analysis performance.

Traditionally, each DBMS has been conceived for handling a specific type of data; for example, relational DBMSs for structured data, document-based DBMSs for semi-structured data, etc. Therefore, when an application requires different types of data, two solutions are actually possible: (i) integrate all data into a single DBMS, or (ii) use two or more DBMSs together. The former solution presents serious drawbacks: first of all, some types of data cannot be stored and analyzed (e.g., the pure relational model does not support the storage of XML, arrays, etc. [3]). Besides, even when data can be converted and stored in the target DBMS, querying performances may be unsatisfactory. The latter approach (known as *polyglot persistence* [4]) presents important challenges as well, namely, technically managing more DBMSs, a steep learning curve for developers, inadequate performance optimization, complex logic in applications, data inconsistency, etc. [5].

Multi-model databases (MMDBMSs) have recently been proposed to overcome these issues. An MMDBMS is a DBMS that natively supports different data models under a single query language to grant performance, scalability, and fault tolerance [2]. Remarkably, using a single platform for multi-model data promises to deliver several benefits to users besides that of providing a unified query interface; namely, it will reduce maintenance and data integration issues, speed up development, and eliminate migration problems [5, 2]. Examples of MMDBMSs are PostgreSQL, ArangoDB, Cosmos DB, and CouchBase. PostgreSQL (`www.postgresql.org/`) is a relational DBMS that natively supports the row-oriented, column-oriented, key-value, and document-oriented data models, offering XML, hstore, JSON/JSONB data types for storage. ArangoDB (`www.arangodb.com/`) supports the graph-based, key-value, and document-oriented data models.

Handling variety while granting at the same time volume and velocity is even more complex in Data Warehouses (DWs) and OLAP systems. Indeed, warehoused data come as a result of the integration of huge volumes of heterogeneous data, and OLAP requires very good performances for data-intensive analytical queries [6]. Traditional DW architectures rely on a single, relational DBMS for

storage and querying[1]. To offer better support to volume while maintaining velocity, some recent works propose the usage of NoSQL DBMSs; for example, [7] relies on a document-based DBMS, and [8] on a column-based DBMS. However, all NoSQL proposals for DWs are based on a single data model, and all data must be transformed to fit that model. Indeed, although these approaches offer interesting results in terms of volume and velocity, they have been mainly conceived and tested for structured data, without taking variety into account.

To facilitate OLAP querying, DWs are normally based on the multidimensional model, which introduces the concepts of facts, dimensions, and measures to analyze data, so source data must be forcibly transformed to fit a multidimensional logical schema following a so-called *schema-on-write* approach. Since this is not always painless because of the schemaless nature of some source data, some recent papers (e.g., [9, 10]) propose to directly rewrite OLAP queries over schemaless data sources (specifically, over document stores) that are not organized according to the multidimensional model, following a *schema-on-read* approach (i.e., the multidimensional schema is not devised at design time and forced in a DW, but decided by every single user at querying time). However, even these approaches rely on a single-model DBMS.

An interesting direction towards a solution for effectively handling the 3 V's in DW and OLAP systems is represented by MMDBMSs. A *multi-model data warehouse* (MMDW) can store data according to the multidimensional model and, at the same time, let each of its elements be natively represented through the most appropriate model. Among the benefits we envision for MMDWs, that of bridging the architectural gap between data lakes and DWs, that of reducing the cost for ETL, and that of ensuring better flexibility, extensibility, and evolvability thanks to the use of schemaless models.

In this paper, we conduct an investigation of the effectiveness and efficiency of MMDWs to store multidimensional data. Since no benchmark dataset for DWs supports variety, for our experiments we give a multidimensional form to the data provided by UniBench [11], a benchmark for MMDBMSs that well represents variety, and define an OLAP workload on it. For the implementation we use PostgreSQL[2], which gives native multi-model support for all data models present in UniBench —except graph-based, for which we use the AgensGraph extension (`bitnine.net/agensgraph/`). In this scenario, we describe and compare three different solutions. The first one relies on a logical schema that extends the star schema [6] by introducing semi-structured (JSON, XML, graph-based, and key-value) data in the multidimensional elements. This solution goes in the direction of coupling the pros of schema-on-write approaches (mainly, better performances and simpler query formulation with no need for query rewriting) with those of schema-on-read approaches (higher flexibility in

[1]More precisely, this is true for so-called *ROLAP* architectures. In *MOLAP* architectures, data are stored in multidimensional arrays. Finally, in *HOLAP* architectures, a MOLAP and a ROLAP systems are coupled.

[2]Other MMDBMSs, such as ArangoDB, could not be used since they do not offer support for all the models involved.

ad-hoc querying, simpler ETL, and lower effort for evolution). The other two solutions consist, respectively, of a full-relational implementation based on a classical star schema, and on a non-relational implementation where no multidimensional elements are introduced at the logical level (essentially, a data lake-like approach). This paper extends our previous contribution [12] in several ways:

1. by also including the graph-based data of UniBench,
2. by adopting better optimized schemata,
3. by comparing three different solutions rather than two,
4. by quantitatively evaluating the efficiency and effectiveness of MMDWs from five points of view: querying, storage, ETL, flexibility & extensibility, and evolvability, using ad hoc metrics defined in the literature.

The paper outline is as follows. After discussing the related literature in Section 2, in Section 3 we present the UniBench case study. Sections 4 and 5 introduce our logical schema for MMDWs and the related OLAP workload, respectively. Section 6 describes the two alternative logical schemata to be used for comparisons. Section 7 discusses the results of the experiments we made, while Section 8 presents our vision of future MMDW research and draws the conclusions.

2. Related work

Some recent work concerns warehousing and OLAP using NoSQL DBMSs of different kinds. In [13], three different logical models are proposed, using 1 or N document collections to store data in document-based DBMSs and highlighting the utility of nested document and array types [14]. The same authors also investigate how to handle complex hierarchies and summarizability issues with document-based DWs [15]. The introduction of spatial data in document-based DWs has been discussed in [16], which proposes a new spatial multidimensional model to avoid redundancy of spatial data and improve performances. A logical model for column-based DWs has been proposed by [8] and [17] to address volume scalability. In [18], transformation rules for DW implementation in graph-based DBMSs have been proposed for better handling social network data. To the best of our knowledge, only [19] presents a benchmark for comparing NoSQL DW proposals; specifically, this benchmark is applied to MongoDB and Hbase. Some works also study the usage of XML DBMSs for warehousing XML data [20]. Although XML DWs represent a first effort towards native storage of semi-structured data, their querying performances do not scale well with size, and compression techniques must be adopted [21].

Among all these proposals, it is hard to champion one logical and physical implementation for NoSQL and XML DWs, since no approach clearly outperforms the other on the 3 V's. Moreover, these single-model proposals do not

address other issues related to warehousing big data, such as reducing the cost of ETL, evolution, and improving flexibility.

Recently, some approaches to execute OLAP queries directly against NoSQL data sources were proposed. In [9], a schema-on-read approach to automatically extract facts and hierarchies from document data stores and trigger OLAP queries is proposed. A similar approach is presented in [10]; there, schema variety is explicitly taken into account by choosing not to design a single crisp schema where source fields are either included or absent, but rather to enable an OLAP experience on some sort of "soft" schema where each source field is present to some extent. In the same direction, [22] proposes a MapReduce-based algorithm to compute OLAP cubes on column stores, while [23] aims at delivering the OLAP experience over a graph-based database.

The approaches mentioned above rely on a single-model database. Conversely, [24] proposes a pay-as-you-go approach which enables OLAP queries against a polystore supporting relational, document, and column data models by hiding heterogeneity behind a dataspace layer. Data integration is carried out on-the-fly using a set of mappings. Even this approach can be classified as schema-on-read; the focus is on query rewriting against heterogeneous databases and not on the performances of the approach.

A survey of the existing multi-model DBMSs and their features is presented in [2]. Multi-model DBMSs support different models using specific storage strategies. For example, PostgreSQL stores data using relational tables, text, or binary format, while ArangoDB uses a document storage technique. In order to enable queries on different data models, these DBMSs provide new query languages, namely, extended-SQL and AQL for PostgreSQL and ArangoDB, respectively. Importantly, depending on the storage strategy, each DBMS implements a particular set of physical structures (indexes and partitions). Research on MMDBMSs is currently moving towards query optimization, evolution, and design as described in [2].

3. Case study: UniBench

UniBench is a benchmark for multi-model databases proposed in [11]. It includes a retail dataset composed of relational, XML, JSON, key-value, and graph data as shown in Figure 1, which makes it a good representative for variety. The UniBench dataset is sketched in Figure 3 and briefly commented below:

- Customer data are stored in graph-based form: :InfoCust is a class, [:knows] is a relationship between couples of nodes of class :InfoCust modeling the friendship relationships between customers.
- Product data are stored in XML form within the InfoPrdt document. Each product has a single vendor.

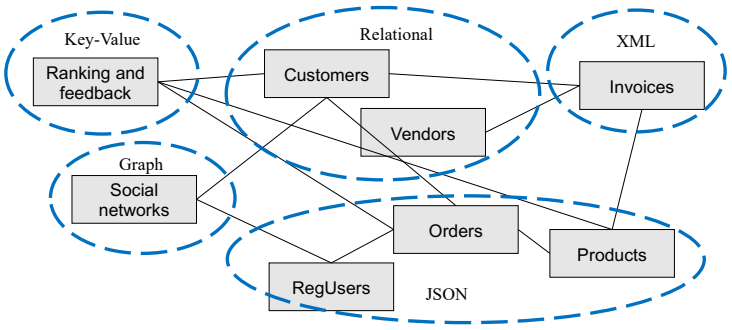


Figure 1: Overview of the UniBench data

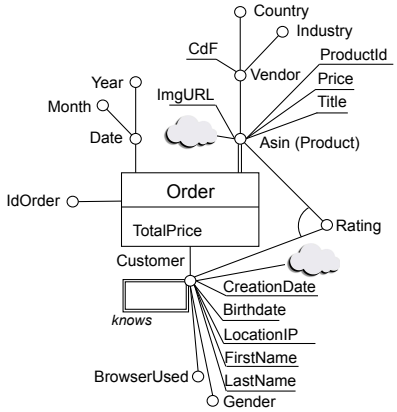


Figure 2: Multidimensional schema for UniBench (the DFM notation [25] is used)

- Order data are stored in JSON form within the InfoOrder collection. An order is made on a date for a total price by a customer; it has several lines, each referring to a product.
- Feedbacks are stored in key-value form within the Feedback collection. A feedback is given by a customer on a product, and is quantified by a rating.

UniBench is not a multidimensional database. Since our goal is to handle variety with specific reference to DWs, we had to derive a multidimensional schema from UniBench. To this end we adopted a classical data-driven approach based on functional dependencies; since these dependencies are not explicitly represented in schemaless sources, we had to infer them from the data. The resulting schema represents the Order fact; as shown in Figure 2 using the DFM notation, Order has one measure, TotalPrice, and four dimensions:

- An IdOrder (degenerate) dimension.
- A Time dimension with levels Day, Month, and Year (note that Month here is not month-in-year, so it belongs to a separate branch of the hierarchy).

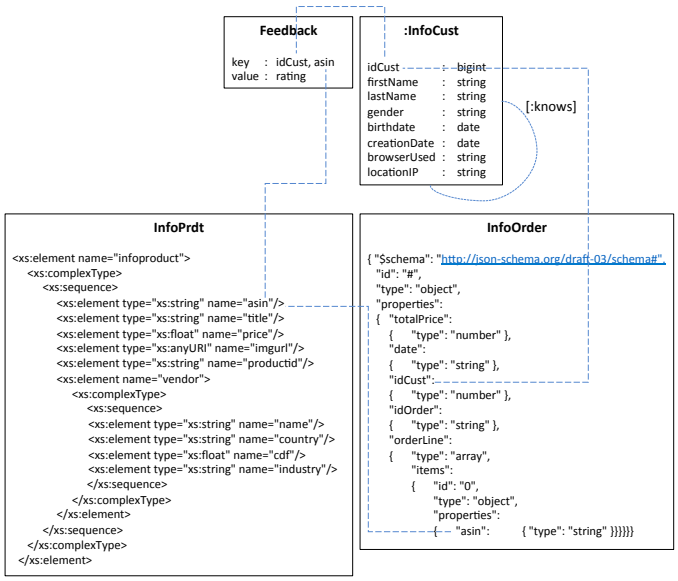


Figure 3: The UniBench dataset; dashed lines represent implicit inter-attribute relationships, dotted lines graph arcs

- An Asin (Product) dimension with some descriptive attributes (e.g., Title) and a Vendor hierarchy[3]. The cloud symbol in the schema denotes that a product can have some additional attributes not specified at design time; this feature will be used later in the paper to discuss extensibility issues. Since an order is associated with many products, a many-to-many relationship is set between the fact and the product dimension (*non-strict hierarchy*, represented in the DFM with a double arc).

- A Costumer dimension with levels Gender and UsedBrowser, plus some descriptive attributes, e.g., LastName. To model the graph of inter-customer acquaintances, a many-to-many recursive association (knows) is set on Customer.

Attribute Rating is cross-dimensional, i.e., its value is jointly determined by Product and Customer (a customer can rate several products).

UniBench comes with a workload consisting of 10 read-only queries and 2 read-write transactions (ignored in what follows). Each query is described from two perspectives: a business perspective, illustrating common business cases (e.g., mining common purchase patterns in a community and analyzing the communitys influence on the individuals purchase behaviors), and a technical perspective, pointing out common technical challenges for the multi-model query processing. These challenges relate to different components of the

[3] Asin stands for Amazon Standard Identification Number.

MMDBMS (e.g., query optimizer and storage system) and address typical multi-model processing problems (especially, choosing the right join type and order and performing complex aggregations). Complexity-wise, the workload spans from conjunctive to analysis queries.

4. A multi-model star schema for UniBench

In this section, we present a Multi-Model, MultiDimensional (in short, M^3D) logical schema for the Order fact introduced above. Essentially, we use a classical star schema with fact and dimension tables, extended with semi-structured data in JSON, XML, key-value, and graph-based form. Starting from a star schema has several clear advantages: (i) the star schema is supported by all OLAP servers and already in use in a huge number of enterprise DWs; (ii) the best practices for designing a star schema from a conceptual schema are well understood and commonly adopted by practitioners; (iii) fact-dimension relationships are ruled by foreign keys so their consistency can be natively checked by the DBMS; (iv) performance optimization of star schema has been long studied and practiced at both the logical (e.g., via view materialization) and the physical (e.g., via indexing) level.

Clearly, several possible alternatives arise for modeling the Order fact with an extended star schema. Defining a set of best practices for designing an M^3D schema that achieves the best trade-off between the advantages listed in Section 1 is out of the scope of this paper; so, we opted for designing the schema based on a simple guideline: *preserve as much as possible the source data variety*, i.e., minimize the transformations to be applied to UniBench source data. We also had to keep in mind that PostgreSQL's support to non-relational models is only given in terms of column data types (e.g., a JSON column) in relational tables —except for graph data, which are modeled by AgensGraph and do not need to be hosted within a relational table. In the light of this, the approach we followed to create the M^3D schema starting from the multidimensional schema in Figure 2 and from the UniBench source data can be sketched as shown below. Note that the approach adopted is workload-agnostic, as the schema it creates is not specifically optimized for any set of user queries.

1. **Locate the fact**. Search in the source data for a piece of data f representing the fact. The Order fact is represented by the InfoOrder JSON collection.

2. **Find dimensions**. Check in f for references d_i to dimensions appearing in the multidimensional schema. Each document in InfoOrder refers to one customer (through field idCust), one order (idOrder), one date (date), and to an array of products (asin). All of these are shown as dimensions in the multidimensional schema.

3. **Find measures**. Check in f for references to measures appearing in the multidimensional schema. Each document in InfoOrder is characterized by a (numerical) price (totalPrice), which is shown as a measure in the multidimensional schema.

4. **Create fact table**. Initialize the fact table FT by adding a reference to f (in our example, Fact_Order is initialized with attribute InfoOrder).

5. **Create dimensions**. For each dimension d_i found:

   a. Search in the source data for a piece of data p_i referring field d_i.
      - If not found (e.g., idOrder), add d_i to FT as a foreign key. In case the multidimensional schema requires attributes that can be derived from d_i, create a relational dimension table DT_i to store them (e.g., for field date, create table Dim_Date with attributes Date, Month, and Year) and add its primary key to FT as a foreign key instead of d_i.
      - If found:
        - If p_i is graph-based, add d_i to FT (e.g., for idCust).
        - Otherwise, create a relational dimension table DT_i with key d_i and a reference to p_i (e.g., for asin); then add d_i to FT as a foreign key.

An exception to this basic flow is given by the product dimension, where the multidimensional schema shows a non-strict hierarchy (one order refers to several products). In this case the key Asin of the Dim_Product dimension table is not included in the Fact_Order fact table, as the many-to-many relationship between orders and products is already implicitly established by the InfoOrder collection.

Figure 4 shows the resulting M^{3}D schema, which can be described as follows:

- The fact table, Fact_Order, has one tuple for each order (identified by an order identifier, a customer identifier, and a date identifier); it references the order date via a foreign key, and it relates to customers by including IdCust. Each tuple includes a JSON document that stores the totalPrice measure and an array of orderlines, each specifying a product.

- The temporal dimension table, Dim_Date, enables useful aggregations by storing, besides dates, months, and years.

- The product dimension table, Dim_Product, for each product stores an XML document with the product name (title), price, image, identifier, and with the details of its vendor. Each product also has a Feedback attribute that stores all its ratings in key-value form, with the customer code as a key.

- As shown in Figure 2, each order refers to several products. To model this non-strict hierarchy, rather than opting for the classical relational solution (a many-to-many bridge table [25]), we established a connection between the InfoOrder document stored in the fact table and the Dim_Product dimension table via the asin attribute.

- Customers' data are stored in graph-based form to represent the customers each customer knows. The connection with the fact table is made through idCust.

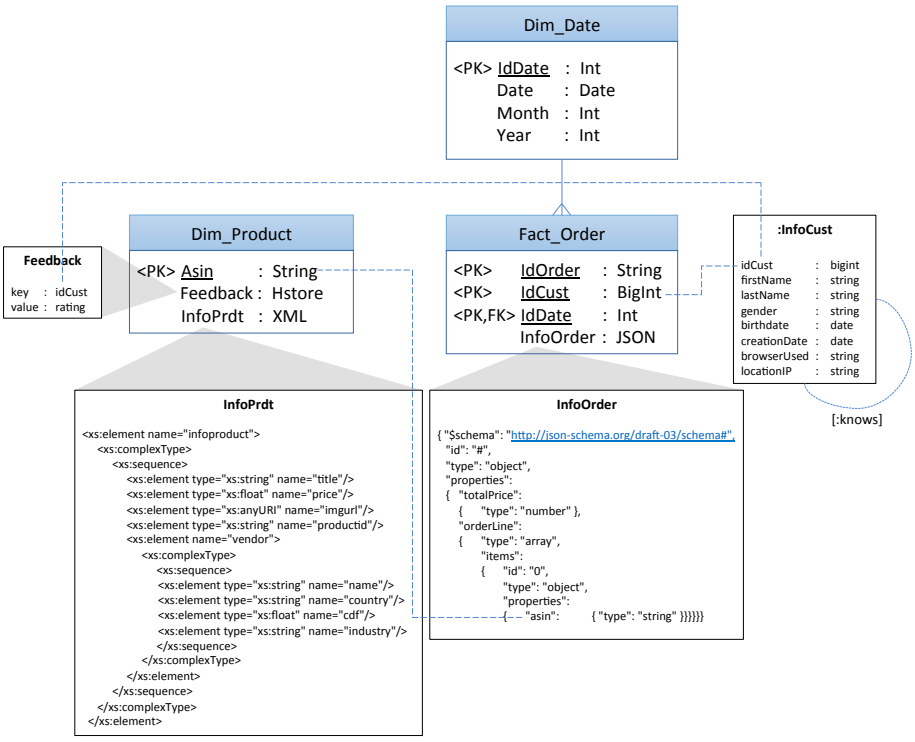


Figure 4: Multi-model star schema in PostgreSQL (solid lines represent foreign key relationships, dashed lines implicit inter-attribute relationships, dotted lines arcs in a graph)

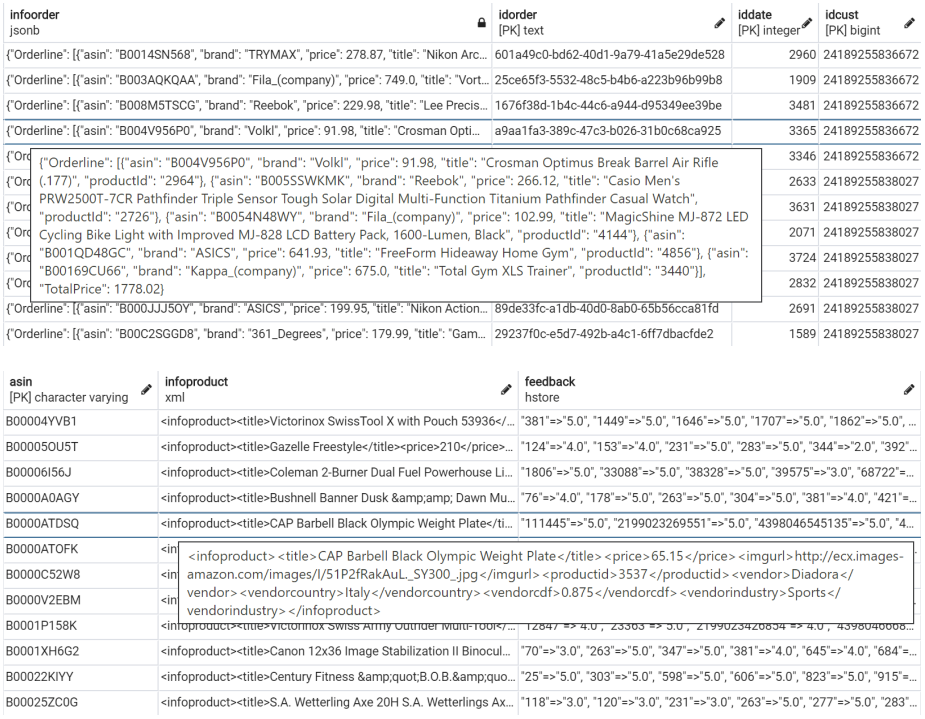
An example of instances of the fact table and the product dimension table are shown in Figure 5.

The cloud symbols in Figure 2 denote that the product and customer dimensions can include some additional attributes not specified at design time (hence, not included in the JSON/XML schema). For instance, some InfoPrdt documents will have an EU attribute storing the category of product according to the EU classification (see Figure 6), while some InfoCust documents will have a Boolean gold attribute specifying whether a customer is a top one or not.

5. An OLAP workload for UniBench

The workload we introduce to test our M^{3}D schema comprises two sets of OLAP queries, which we will call WL1 and WL2, respectively. As expected of an interactive OLAP workload, the queries feature several joins and vary in the combinations of group-by sets and selection predicates to produce results of low cardinality (e.g., coarse group-by set with low selectivity).

WL1 includes 12 simple OLAP queries meant to put the M^{3}D schema to the test on its most peculiar features, such as inter-model joins and graph naviga-



infoorder jsonb	idorder [PK] text	iddate [PK] integer	idcust [PK] bigint
{"Orderline": [{"asin": "B0014SN568", "brand": "TRYMAX", "price": 278.87, "title": "Nikon Arc...	601a49c0-bd62-40d1-9a79-41a5e29de528	2960	24189255836672
{"Orderline": [{"asin": "B003AQKQAA", "brand": "Fila_(company)", "price": 749.0, "title": "Vort...	25ce65f3-5532-48c5-b4b6-a223b96b99b8	1909	24189255836672
{"Orderline": [{"asin": "B008M5TSCG", "brand": "Reebok", "price": 229.98, "title": "Lee Precis...	1676f38d-1b4c-44c6-a944-d95349ee39be	3481	24189255836672
{"Orderline": [{"asin": "B004V956P0", "brand": "Volkl", "price": 91.98, "title": "Crosman Opti...	a9aa1fa3-389c-47c3-b026-31b0c68ca925	3365	24189255836672
{"Orderline": [{"asin": "B004V956P0", "brand": "Volkl", "price": 91.98, "title": "Crosman Optimus Break Barrel Air Rifle (.177)", "productId": "2964"}, {"asin": "B005SSWKMK", "brand": "Reebok", "price": 266.12, "title": "Casio Men's PRW2500T-7CR Pathfinder Triple Sensor Tough Solar Digital Multi-Function Titanium Pathfinder Casual Watch", "productId": "2726"}, {"asin": "B0054N48WY", "brand": "Fila_(company)", "price": 102.99, "title": "MagicShine MJ-872 LED Cycling Bike Light with Improved MJ-828 LCD Battery Pack, 1600-Lumen, Black", "productId": "4144"}, {"asin": "B001QD48GC", "brand": "ASICS", "price": 641.93, "title": "FreeForm Hideaway Home Gym", "productId": "4856"}, {"asin": "B00169CU66", "brand": "Kappa_(company)", "price": 675.0, "title": "Total Gym XLS Trainer", "productId": "3440"}], "TotalPrice": 1778.02}		3346	24189255836672
{"Orc		2633	24189255838027
{"Orc		3631	24189255838027
{"Orc		2071	24189255838027
{"Orc		3724	24189255838027
{"Orc		2832	24189255838027
{"Orderline": [{"asin": "B000JJJ5OY", "brand": "ASICS", "price": 199.95, "title": "Nikon Action...	89de33fc-a1db-40d0-8ab0-65b56cca81fd	2691	24189255838027
{"Orderline": [{"asin": "B00C2SGGD8", "brand": "361_Degrees", "price": 179.99, "title": "Gam...	29237f0c-e5d7-492b-a4c1-6ff7dbacfde2	1589	24189255838027

asin [PK] character varying	infoproduct xml	feedback hstore
B00004YVB1	<infoproduct><title>Victorinox SwissTool X with Pouch 53936</...	"381"=>"5.0", "1449"=>"5.0", "1646"=>"5.0", "1707"=>"5.0", "1862"=>"5.0", ...
B00005OU5T	<infoproduct><title>Gazelle Freestyle</title><price>210</price>...	"124"=>"4.0", "153"=>"4.0", "231"=>"5.0", "283"=>"5.0", "344"=>"2.0", "392"...
B00006I56J	<infoproduct><title>Coleman 2-Burner Dual Fuel Powerhouse Li...	"1806"=>"5.0", "33088"=>"5.0", "38328"=>"5.0", "39575"=>"3.0", "68722"=...
B0000A0AGY	<infoproduct><title>Bushnell Banner Dusk & Dawn Mu...	"76"=>"4.0", "178"=>"5.0", "263"=>"5.0", "304"=>"5.0", "381"=>"4.0", "421"=...
B0000ATDSQ	<infoproduct><title>CAP Barbell Black Olympic Weight Plate</ti...	"111445"=>"5.0", "2199023269551"=>"5.0", "4398046545135"=>"5.0", "4...
B0000ATOFK	<in	
B0000C52W8	<in	
B0000V2EBM	<in	
B0001P158K	<infoproduct><title>Victorinox Swiss Army Outrider Multi-Tool</...	"12847"=>"4.0", "23363"=>"5.0", "2199023426854"=>"4.0", "4398046668...
B0001XH6G2	<infoproduct><title>Canon 12x36 Image Stabilization II Binocul...	"70"=>"3.0", "263"=>"5.0", "347"=>"5.0", "381"=>"4.0", "645"=>"4.0", "684"=...
B00022KIYY	<infoproduct><title>Century Fitness "B.O.B.&quo...	"25"=>"5.0", "303"=>"5.0", "598"=>"5.0", "606"=>"5.0", "823"=>"5.0", "915"=...
B00025ZC0G	<infoproduct><title>S.A. Wetterling Axe 20H S.A. Wetterlings Ax...	"118"=>"3.0", "120"=>"3.0", "231"=>"3.0", "263"=>"5.0", "277"=>"5.0", "283"...

<infoproduct><title>CAP Barbell Black Olympic Weight Plate</title><price>65.15</price><imgurl>http://ecx.images-amazon.com/images/I/51P2fRakAuL._SY300_.jpg</imgurl><productid>3537</productid><vendor>Diadora</vendor><vendorcountry>Italy</vendorcountry><vendorcdf>0.875</vendorcdf><vendorindustry>Sports</vendorindustry></infoproduct>

Figure 5: Sample instances of Fact_Order (top) and Dim_Product (bottom)

```
<title>5 LED BicyleRear Tail</title>
<price>8.26</price>
<imgurl>http://ecx.images-amazon.com/SY300.jpg</imgurl>
<productid>428a784b-8f24-42ba-8e98-fca8f0cdd7b2</productid>
<EU>Electronics</EU>
```

Figure 6: An InfoPrdt document including an extra-schema attribute, EU

tion. The queries, listed in Table 1, can be grouped depending on the model being tested:

- Query Q1-01 involves only the relational model: it accesses the fact table but it does not require any JSON data to be accessed.
- Queries Q1-02 and Q1-03 focus on JSON data. The former accesses the whole fact table and requires to unnest every InfoOrder array; the latter applies a selection predicate on asin, which is nested within InfoOrder.
- Queries Q1-04 and Q1-05 focus on XML data. The former accesses the whole InfoPrdt collection, while the latter applies a selection predicate on an XML attribute.
- Queries Q1-06 to Q1-08 focus on the key-value model. Q1-06 requires to

Table 1: The WL1 workload on the Order fact (G stands for graph-based, KV for key-value, R for relational)

Query	Description	Models joined	Models in GB	Models in sel.	Perc. orders	Result card.
Q1-01	Number of orders by month	R	R	-	100%	12
Q1-02	Number of orders by product	JSON, R	JSON	-	100%	1814
Q1-03	Number of orders for a given product	JSON, R	JSON	JSON	0.1%	1
Q1-04	Number of orders by vendor	R, JSON, XML	XML	-	100%	59
Q1-05	Number of orders by vendor for a given vendor country	R, JSON, XML	XML	XML	49%	6
Q1-06	Number of orders by customer rating	JSON, KV, R	KV	-	100%	5
Q1-07	Number of orders by customer rating for a given vendor country	JSON, KV, R, XML	KV	XML	39%	5
Q1-08	Number of orders by customer rating for a given vendor country and customer	JSON, KV, R, XML	KV	G, XML	0.01%	5
Q1-09	Number of orders by browser for the customers known by female customers (1 hop)	G, R	G	G	92%	5
Q1-10	Number of orders by browser for the customers known by the one with a given IP (2 hops)	G, R	G	G	5%	5
Q1-11	Number of orders by EU category for a given vendor country	JSON, R, XML	XML	-	35%	10
Q1-12	Number of orders by EU category and gold status for a given vendor country	G, JSON, R, XML	G, XML	-	17%	44

access the whole Feedback data, while Q1-07 and Q1-08 apply selection predicates with increasing selectivity. Since rating is a cross-dimensional attribute, Q1-07 applies a filter on one dimension, while Q1-08 filters on both dimensions.

- Queries Q1-09 and Q1-10 focus on the graph-based model. In this case, we test different navigation patterns: Q1-09 implies a *wide* navigation of the graph (i.e., several graph nodes are involved, and only one hop is made), while Q1-10 implies a *deep* navigation of the graph (i.e., few graph nodes are involved, and two hops are made).

- Queries Q1-11 and Q1-12 focus on the schemaless property of non-relational models. In particular, these queries apply selection predicates with increasing selectivity on missing attributes.

WL2 includes 10 more complex queries, meant to test M^3D in a realistic scenario. For these queries, listed in Table 2, we were inspired by the workload of the classical SSB benchmark [26], itself loosely based on the TPC-H benchmark. The SSB workload is meant to functionally cover the different types of star schema queries while varying fact table selectivity. SSB queries are organized in 4 flights, where each flight is a list of 3 to 4 queries. Query flight 1 has restrictions on only 1 dimension, flight 2 has restrictions on 2 dimensions, flight 3 on 3, and flight 4 represents a what-if sequence of the OLAP type. We adopt the same approach, by adapting a selection from the UniBench workload essentially respecting the technical perspective of UniBench (cf. Section 3),

Table 2: The WL2 workload on the Order fact

Query	Description	Models joined	Models in GB	Models in sel.	Perc. orders	Result card.
Q2-01	Total price by vendor and rating for the customers known by a given customer, for a given month	G, JSON, KV, R, XML	KV, XML	G, R	0.0005%	11
Q2-02	Number of orders by industry and rating for the customers known by a given customer, for a given month	G, KV, R, XML	KV, XML	G, R	0.0002%	3
Q2-03	Total price by customer for a given product and period	G, JSON, R	G	R, XML	0.02%	424
Q2-04	Number of orders by customer for a given product and period, for bad ratings	G, KV, R	G	KV, R, XML	0.001%	31
Q2-05	Total price for 2 given customers and their friends (3-hops)	G, JSON, R	—	G	77%	1
Q2-06	Total price by rating for 2 given customers and their friends (3-hop), for a given product and for high ratings	G, JSON, KV, R	KV	G, KV, XML	76%	2
Q2-07	Total price by customer and product for 2 given customers plus the customers in the shortest path	G, JSON, R	G	G	0.002%	3
Q2-08	Total price by product, rating, and connected customers for a given year, vendor, genre, having total price greater than some value	G, JSON, KV, R, XML	G, KV, XML	G, XML, R	0.0001%	2
Q2-09	Total price by industry for a given country, order by total price	JSON, R, XML	XML	XML	37%	1
Q2-10	Total price by country for the top 3 customers, order by country	G, JSON, R, XML	XML	G	0.003%	24

while at the same time coupling some queries to simulate short OLAP sessions, namely:

- from Q2-01 to Q2-02, roll-up and projection, with high selectivity;
- from Q2-03 to Q2-04, slice-and-dice and projection, with medium selectivity;
- from Q2-05 to Q2-06, drill-down and slice-and-dice, with low selectivity.

For instance, queries Q2-05 and Q2-06 of WL2 adapt query Q5 of the Unibench workload [11]: *Given a customer and a product category, find persons who are this customer's friends within 3-hops friendship in the* knows *graph and they have bought products in the given category. Finally, return feedback with 5-rating review of those bought products.* The resulting queries in WL2 are aggregated cube queries over different combinations of joined models (from 3 to 5), varying in number (from 1 to 3) and complexity (from atomic to selecting nodes in a shortest path) of selections.

In both Tables 1 and 2, each query is characterized by the models joined together (i.e., those used by the pieces of data that must be accessed to answer the query), those in the group-by clause (i.e., those used by the pieces of data

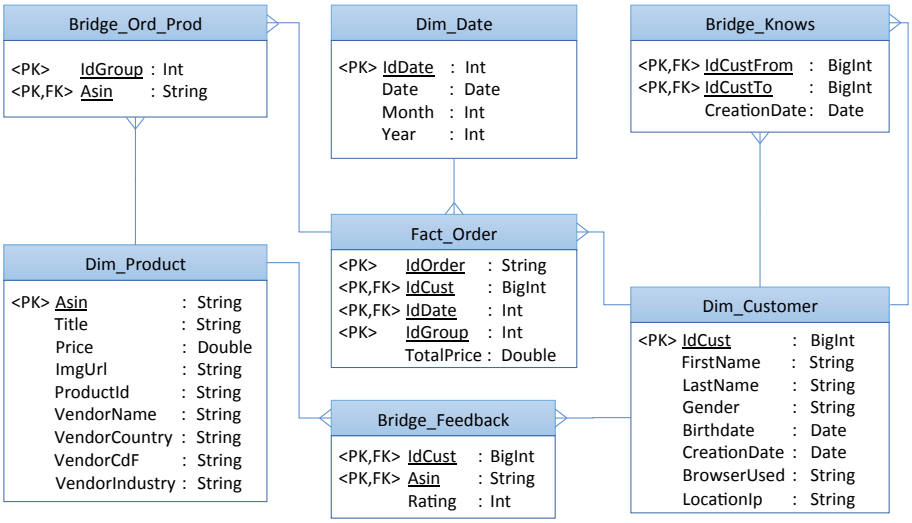


Figure 7: Full-relational star schema in PostgreSQL

storing the levels in the group-by set), and those in the selection clause (i.e., those used by the pieces of data storing the levels mentioned in the selection predicate); the percentage of orders accessed to answer the query (before aggregation) and the cardinality of the result (after aggregation) are shown as well. Note that the cardinality of the result is expressed with reference to the UniBench dataset generated with the highest scaling factor available (30), which includes 2,368,510 orders. Finally, in the query descriptions, "by" introduces a group-by and "for" a selection.

6. Two alternative schemata for UniBench

In this section, we describe two alternative implementations of the Order fact: a full-relational one (from now on, FR) and a non-relational one (NR), which in Section 7 we will compare to the M^{3}D schema.

6.1. Full-relational schema

The FR schema is meant to conform to the classical design guidelines for star schemata implemented in relational DBMSs. For this schema, we used three bridge tables as shown in Figure 7. The first one, Bridge_Ord_Prod, stores the many-to-many relationship between an order and its products. The second one, Bridge_Feedback, is necessary to store the Rating cross-dimensional attribute. The third one, Bridge_Knows, stores the graph of inter-customer relationships. All the product and customer data are stored in the corresponding dimension tables.

As explained in Section 4, attributes EU and gold were not known at design time. One possibility to deal with this would be to adopt the EAV model, where

separate tables are used to store entity-attribute-value triples, thus encouraging flexibility and extensibility. The EAV model is often used in the healthcare domain to store and manage highly-sparse patient data in a compact way [27]. However, here we chose not to adopt the EAV model for three reasons: (i) it would lead to a significant deviation from a classical star schema; (ii) it would add significant burden to the formulation of OLAP queries; (iii) it is known to cause performance issues in presence of large volumes of data —which indeed is normally the case in DWs. Thus, EU and gold could not be included in the FR schema. Clearly, unless some (costly) evolution of the schema is carried out, these attributes cannot be loaded and they cannot be used for querying.

6.2. Non-relational schema

The NR schema is meant to reproduce an architecture where raw data are stored in a data lake. A *data lake* [28] ingests heterogeneously-structured raw data from various sources and stores them in their native format, enabling their processing according to changing requirements [29]. Differently from DWs, data lakes support the storage of any kind of data with low-cost design, provide increasing analysis capabilities, and offer an improvement in data ingestion; however, analysis tasks are more complex and time-consuming since data are directly queried in an OLAP fashion without putting them in multidimensional form. As previously mentioned, this approach is commonly called schema-on-read to distinguish it from schema-on-write approaches, in which raw data are put into multidimensional form and stored in a DW —as done with the FR schema [9].

Since the idea here is to keep all source data in their native form, the NR schema is the one already shown in Figure 3. We recall that PostgreSQL supports non-relational models only in terms of column data types in relational tables —except for graph data, which are modeled by AgensGraph. Thus:

- Customer data are stored in graph-based form using AgensGraph.
- Product data are stored in XML form within a relational table named table_InfoPrdt, consisting of an InfoPrdt column of XML data type.
- Order data are stored in JSON form within a relational table named table_InfoOrder, consisting of an InfoOrder column of JSON data type.
- Feedbacks are stored in key-value form within a relational table named table_Feedback, consisting of a Feedback column of hstore data type.

7. Experimental evaluation

In this section, we evaluate the efficiency and effectiveness of the M^3D schema from different points of view, by comparing it with the two alternative solutions described in Section 6: a classical relational one (FR) and a non-relational one (NR).

We have implemented all three solutions in PostgreSQL 10.4, which gives support to JSON, XML, and key-value storage; for the M^{3}D and NR schemata we used AgensGraph 2.2, i.e., an open-source extension of PostgreSQL that includes support to graph storage. AgensGraph relies on relational structures to store nodes and edges: several tables are created, one for each class; dynamic node properties are supported by modeling each node as a JSON object, and B-tree indexes are automatically computed to support efficient querying; ultimately, Cypher queries are mapped to SQL queries on such structures [30]. Although being different from a pure graph implementation, we remark that — at the time of writing — there is no other multi-model system that supports all the data models considered.

For all three implementations, B+trees have been used to index (i) primary and foreign keys in relational tables, and (ii) identifiers and attributes referencing them in JSON/XML/graph-based data. Also, a GIN index has been used in M^{3}D and NR to index the asin attribute within InfoOrder, since many attribute values may exist within the same order. A GIN index is an inverted index appropriate for data that contain multiple components, such as arrays; it contains a separate entry for each component value, and it can efficiently handle queries that search for specific component values. Data used to feed dimensions and facts have been extracted from the UniBench benchmark [11] with the highest scaling factor available (i.e., 30). Specifically, we have 2,225 dates (|Dim_Date|), 165,586 customers (|Dim_Customer|), 9,691 products (|Dim_Product|), and 2,368,510 orders (|Fact_Order|).

All tests have been run on a Core i7 with 8 CPUs @3.6GHz server with 32 GB RAM running Ubuntu. PostgreSQL's memory parameters have been set as follows:

- *shared_buffers* (i.e., the number of shared memory buffers used by the server) is set to its default, 128MB; this avoids the entire database being stored in memory;
- *effective_cache_size* (i.e., the estimate that the query planner makes of how much memory is available for disk caching by the operating system and within the database itself) is set to 4GB;
- *work_mem* (i.e., the amount of memory actually used by PostgreSQL for each user query) is set to 80MB; this setting enables 100 concurrent connections to the MMDW.

The three solutions and the workload queries are all publicly available at `https://github.com/big-unibo/m3d`.

7.1. Querying

All the OLAP queries proposed in Section 5 have been successfully formulated and executed over the M^{3}D schema, which confirms the feasibility of using PostgreSQL as a platform for storing and querying MMDWs. In particular, PostgreSQL extends standard SQL with new operators and functions to

```
select sum((o.InfoOrder->>'totalPrice')::float) as totalPrice, f.vendorName, f.rating
from  Fact_Order as o,
      Dim_Date as d,
      ( MATCH (:InfoCust {idCust: 132991})-[:KNOWS]->(k:InfoCust) RETURN k.IdCust ) as k,
      jsonb_array_elements(o.InfoOrder->'orderLine') as op,
      ( select skeys(p.Feedback) as IdCust, Asin, svals(p.Feedback) as rating,
          (xpath('//InfoPrdt/vendor/name/text()',p.InfoPrdt))[1]::varchar as vendorName
        from Dim_Product p) as f
where   op->>'asin' = f.Asin
and     o.IdDate = d.IdDate
and     o.IdCust = k.IdCust::varchar::bigint
and     f.IdCust::bigint = o.IdCust
and     d.Month = 10
group by f.vendorName, f.rating;
```

(a)

```
select sum(TotalPrice) as totalPrice, p.VendorName, f.Rating
from  Fact_Order as o, Dim_Date as d, Dim_Product as p, Bridge_Knows as k,
      Bridge_Feedback as f, Bridge_Ord_Prod as op
where   op.Asin = p.Asin
and     op.IdGroup = o.IdGroup
and     o.IdDate = d.IdDate
and     o.IdCust = f.IdCust
and     f.Asin = p.Asin
and     k.IdCustTo = o.IdCust
and     d.Month=10
and     k.IdCustFrom = 132991
group by p.VendorName, f.Rating;
```

(b)

```
select  sum((o.InfoOrder->>'totalPrice')::float) as totalPrice,
        ( xpath('//InfoPrdt/vendor/name/text()',p.InfoPrdt))[1]::varchar as vendorName,
        f.value
from  table_InfoOrder as o,
      table_Feedback as f,
      table_InfoPrdt as p,
      ( MATCH (:InfoCust {idCust: 132991})-[:KNOWS]->(k:InfoCust) RETURN k.IdCust ) as k,
      jsonb_array_elements(o.InfoOrder->'orderLine') as op
where op->>'asin' = (xpath('//InfoPrdt/asin/text()',p.InfoPrdt))[1]::varchar
and     concat(op->>'asin',',',o.InfoOrder->>'idCust') = f.key
and     o.InfoOrder->>'idCust' = k.IdCust::varchar
and     to_char(to_date((o.InfoOrder->>'date')::text,'YYYY-MM-DD'),'MM')::int = 10
group by (xpath('//InfoPrdt/vendor/name/text()',p. InfoPrdt))[1]::varchar, f.value;
```

(c)

Figure 8: SQL formulation of query Q2-01 in PostgreSQL over the M^{3}D (a), FR (b), and NR (c) schemata

query the contents of hstore columns (i.e., the data type that supports key-value data) and JSON/XML data (e.g., the @> operator and the xpath() function to apply selection predicates to JSON and XML data, respectively). Additionally, AgensGraph supports cross-queries between the PostgreSQL instance and the graph extension by including Cypher queries [31] as inner queries within an SQL query. Conversely, Q1-11 and Q1-12 could not be executed on the FR schema because they use attributes (gold and EU) which were not known at design time so they are not part of that schema.

Figure 8 shows the SQL formulation of query Q2-01 over the M^{3}D, FR, and NR schemata. A qualitative comparison between Figures 8.a and 8.b suggests that the formulation over the M^{3}D schema is more complex; however, we wish to emphasize that there is no real difficulty in formulating queries on an MMDW in comparison to a traditional star schema, except that some knowledge of the

```
select sum((o.InfoOrder->>'totalPrice')::float) as totalPrice, o.IdCust
from Fact_Order as o,
     ( MATCH p=allShortestPaths((a:InfoCust {idCust: 132991})-[:KNOWS*]->(b:InfoCust {idCust: 140644}))
       UNWIND nodes(p) AS k RETURN k.idCust ) as c
where o.IdCust = c.idCust::varchar::bigint
group by o.IdCust;
```

(a)

```
with recursive t(IdCustFrom, IdCustTo, dist) as ( select IdCustFrom, IdCustTo, 1
                                                  from Bridge_Knows
                                                  union
                                                  select t.IdCustFrom, g.IdCustTo, dist+1
                                                  from Bridge_Knows g, t
                                                  where t.IdCustTo = g.IdCustFrom ),
     minpath(IdCustFrom, IdCustTo, dist) as ( select IdCustFrom, IdCustTo, min(dist) as dist
                                              from t
                                              group by IdCustFrom, IdCustTo )
select sum(TotalPrice) as totalPrice, o.IdCust
from Fact_Order o
where o.IdCust in ( select distinct m1.IdCustTo
                    from minpath m1, minpath m2, minpath m3
                    where m1.IdCustFrom = 132991 and m1.IdCustTo = m2.IdCustFrom
                    and   m2.IdCustTo = 140644 and m3.IdCustFrom= 132991
                    and   m3.IdCustTo= 140644 and m3.dist=m1.dist+m2.dist )
group by o.IdCust;
```

(b)

```
select sum((o.InfoOrder->>'totalPrice')::float) as totalPrice, o.InfoOrder->>'idcust'
from table_InfoOrder o,
     ( MATCH p=allShortestPaths((a:InfoCust {idCust: 132991})-[:KNOWS*]->(b:InfoCuts {idCust: 140644}))
       UNWIND nodes(p) AS k RETURN k.idCust ) as c
where o.InfoOrder->>'idCust' = c.idCust::varchar
group by o.InfoOrder->>'idCust';
```

(c)

Figure 9: SQL formulation of query Q2-07 in PostgreSQL over the M^{3}D (a), FR (b), and NR (c) schemata

DBMS-specific operators to manipulate key-value, JSON, and XML types is required.

Figure 9 shows another comparative example of query formulation, the one related to Q2-07. This query requires to find the shortest path between two customers on the knows graph; thus, differently from Q2-01, its formulation is much simpler in M^{3}D and NR than in FR because the complexity of computing shortest paths is hidden inside a predefined function in the Cypher language.

In an attempt to precisely and systematically quantify the complexity of the three different formulations of WL1 and WL2, we computed for each formulation various indicators proposed by Jain & al. [32] to understand the cognitive load on the user during SQL query authoring (see Table 3). The first indicator (Length) measures the character length (sum, mean, and standard deviation) of the query as a string, a proxy for the effort it takes to craft the query. The second indicator (Operator#) measures the number of physical operators in the query execution plan (unique and in total), to estimate the number of steps of computation. Finally, the last indicator (Top-10 operators) shows the most commonly used physical operators, allowing to assess the workload complexity by providing the minimum requirement of SQL features for the workload to run.

Regarding query length, results are as expected. Queries over FR are the

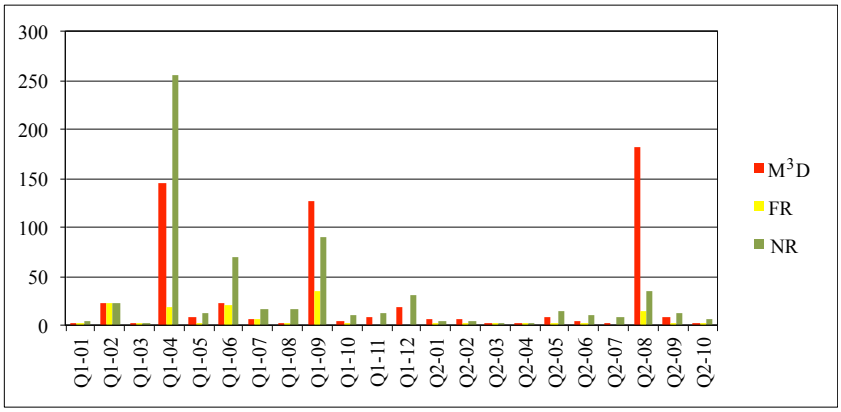


Figure 10: Query performance (in seconds)

shortest ones, while queries over NR are the longest ones and queries over M^{3}D are in between, closer to NR. As explained above, this illustrates that more SQL constructs are needed for unstructured schemata. We note that for longer, more complex OLAP queries (those in WL2) the difference between FR and the two others tends to be reduced, the difference between M^{3}D and NR being quite steady. This indicates that expressing complex queries over M^{3}D may only be moderately difficult (and in some case, e.g., query Q2-07, even simpler) compared to FR. Checking the query plans and the physical operators allows for a finer characterization. Unsurprisingly, the simplest model (FR) needs less distinct operations. Overall, roughly the same set of most frequent operators is used for each schema, which is expected because (i) the workload consists of the same queries, phrased differently, and (ii) PostgreSQL often translates non-relational data to relational form for processing them. However, notable observations can be seen:

- Sort is more often used on M^{3}D, and even more on NR, than it is on FR.
- Joins are more frequent on M^{3}D than on FR and NR. Without limiting to top-10 operators, joins are less frequent on FR (61 in total) than on NR (67) and M^{3}D (68).
- Some forms of scan (parallel sequential scan, function scan) are more frequent on M^{3}D and NR than they are in FR. Without limiting to top-10 operators, scans are less frequent on M^{3}D (122 in total) than on FR (138) and NR (146).
- No particular difference is seen for aggregate operators.

This tends to indicate that the higher formulation effort imposed by M^{3}D and NR is closely related to the effort the system has to do to process the queries.

Table 3: Complexity of queries for workloads WL1 and WL2

WL	Indicators	NR	M^{3}D	FR
WL1	Length (sum)	4505	4308	2573
	Length (avg)	375.41	359.0	285.88
	Length (stdev)	135.77	127.43	86.7
WL2	Length (sum)	6320	6145	5378
	Length (avg)	632.0	614.5	597.55
	Length (stdev)	183.05	179.35	258.55
WL1 and WL2	Operator#	324	324	277
	Unique operator#	31	31	27
	Top-10 operators	Sort/43	Nested Loop/37	Nested Loop/32
		Seq Scan/33	Sort/31	Seq Scan/30
		Nested Loop/33	Seq Scan/28	Sort/27
		Hash/24	Hash/26	Hash/27
		Hash Join/24	Hash Join/26	Hash Join/27
		Par. Seq Scan/20	Function Scan/19	Index Scan/25
		Function Scan/19	Index Scan/19	Bmap Idx Scan/17
		Gather/15	Par. Seq Scan/17	GroupAggr./15
		GroupAggr./13	GroupAggr./16	Bmap Heap Scan/14
		Bmap Idx Scan/13	Gather/14	Par. Seq Scan/11

Finally, Table 4 and Figure 10 show the query execution times in seconds for the three implementations, obtained by running a PL/pgSQL[4] procedure that runs each query of the workload in random order; the reported execution times are the average times of ten workload runs[5]. Not surprisingly, the full-relational implementation outperforms the multi-model implementation over most queries. This can partly be explained by recalling that PostgreSQL was originally born as a relational DBMS, so semi-structured and complex data querying is not fully optimized yet. Additionally, the fact table in M^{3}D is quite larger than the one in the FR schema, which results in slower star joins (even using the JSONB type instead of JSON, the improvement is very small).

In WL1, the goal is to put the M^{3}D schema to the test on its most peculiar features. We can summarize our findings as follows:

- Queries on M^{3}D are generally faster than those on NR, showing that the simple ETL necessary to produce the M^{3}D schema enables a significant improvement of querying performance.
- The complexity of JSON documents comes in handy when the query is restricted to attributes within the document itself. For instance, Q1-03 does not require M^{3}D and NR to join the product dimension (whereas FR needs to access Bridge_Ord_Prod), which translates in better execution times than in FR. On the other hand, the same feature becomes an inconvenience when a join is required and the orderline array needs to be

[4]PL/pgSQL is the procedural language in PostgreSQL.

[5]Please note that PostgreSQL does not provide any functionality to clear the cache; we amend by running the queries multiple times in random order.

Table 4: Query performance (in seconds)

Query	M^{3}D	FR	NR
Q1-01	0.5	2.0	3.5
Q1-02	22.5	22.3	23.2
Q1-03	0.3	0.6	0.3
Q1-04	144.9	18.3	255.8
Q1-05	8.5	2.7	13.1
Q1-06	22.0	19.7	70.4
Q1-07	5.3	6.4	15.9
Q1-08	1.3	0.8	15.7
Q1-09	126.0	34.1	90.7
Q1-10	3.7	0.4	10.3
Q1-11	8.5	-	13.0
Q1-12	18.8	-	29.7
Q2-01	6.1	0.7	5.0
Q2-02	5.6	0.5	4.7
Q2-03	1.5	1.2	1.0
Q2-04	0.2	0.2	1.0
Q2-05	8.9	2.3	13.6
Q2-06	3.9	1.9	9.8
Q2-07	0.8	OOM	8.1
Q2-08	182.1	14.8	34.9
Q2-09	8.6	1.9	12.7
Q2-10	0.9	1.1	6.4

unnested.

- PostgreSQL lacks specific optimization structures adapted to XML data; thus, scanning XML data is quite expensive, as Q1-04 and Q1-05 clearly show.
- The performance of the key-value storage is directly proportional to the selectivity on the Feedback data; indeed, queries from Q1-06 to Q1-08 show that the higher the selectivity, the faster the query. The same trend is not visible in NR due to PostgreSQL's inability to properly use the index when filtering on the key.
- Similarly, the difference between Q1-09 and Q1-10 shows that the graph implementation is more suitable for long and narrow graph navigations (i.e., those requiring a higher number of hops on few nodes) rather than short and wide navigations (i.e., those requiring a single hop on many nodes). The difference between M^{3}D and NR is due to the latter needing to join graph data with JSON data, which adds an extra layer of complexity.

WL2 is a collection of realistic OLAP queries adapted from a selection from the UniBench workload. Due to the higher selectivity of these queries, execution times are generally lower than those from WL1 —even though this is balanced by a higher complexity in terms of expressiveness. The main remarks on this workload concern queries Q2-07 and Q2-08. Q2-07 requires to compute the shortest path between two given customers; whereas this operation is straightforward in the graph used by M^{3}D and NR, it requires the definition of a recursive table on FR. Aside from the query formulation complexity (shown in Section 7.1), this results in an out-of-memory error (OOM) that prevents

Table 5: Storage size (in MB)

Table	**M^{3}D**	**FR**	**NR**
Order	2312	1524	2959.4
Product	242	2.5	200
Customer	69	18	69
Date	0.2	0.2	—
Feedback	—	1515.5	1146.9
Knows	4116.5	915	4116.5
Total	7602.1	3975.2	8491.8

its execution on FR. Conversely, the performance of M^{3}D in answering Q2-08 is significantly worse than FR and NR due to a bad execution plan chosen by PostgreSQL's optimizer. Indeed, the mixture of relational and non-relational attributes is not always easily handled by the optimizer and may lead to non-optimal execution plans; we have found the same issue in other queries (e.g., Q2-01 and Q2-02), although with a far less significant impact. Overall, excluding Q2-07 and Q2-08, M^{3}D obtains an average execution time of 4.9 seconds per query, i.e., worse than FR (1.2 seconds) but better than NR (6.8 seconds).

7.2. Storage

Table 5 shows the storage size of every implementation. Unsurprisingly, the relational implementation FR is overall more sober than the multi-model one M^{3}D (about one half), while NR takes about 10% more space than M^{3}D. More specifically:

- Indeed, the relational model stores data in a much cheaper way than JSON, XML, and graphs, which also have to store tag names. This is made clear for instance by comparing the space taken by the product attributes within the Dim_Product dimension table in FR (2.5 MB) and within the InfoPrdt XML documents in NR (200 MB).

- The Order row refers to orders and their relationships with products; thus, for FR, it also includes the Bridge_Ord_Prod bridge table (which takes only 835 MB out of 1524). The significant overhead for M^{3}D and NR derives from storing the measures and the ordered products in JSON form within the InfoOrder collection.

- In NR, dates are directly stored with orders.

- In the Customer row, the values for M^{3}D and NR only include the space for storing the graph nodes. The arcs are considered in row Knows, which clearly shows the overhead for graph-based storage over the Bridge_Knows bridge table.

- As to feedbacks, for M^{3}D the space taken by the hstore attribute is taken into account in the Product row. The value for FR is the size of the

Bridge_Feedback bridge table, while the one for NR is the size of the Feedback key-value store. Note that the hstore attribute in M^{3}D takes less space because it does not include the asin string in the key field.

7.3. ETL

Empirical evidence shows that the design and maintenance of ETL procedures make up for up to 60% of the resources spent in a DW project [33]. Note that in this section we will consider static ETL (which is performed when a DW is loaded for the first time), not incremental ETL (periodically performed to extract, transform, and load the data inserted/updated in the sources since the last run of the ETL). Besides, ETL procedures are written in terms of SQL statements to enable a better characterization of complexity and be independent of the specific features of ETL tools.

The full-relational implementation required all the UniBench data to be translated into relational form according to the star schema in Figure 7. While the queries to feed data to dimension and fact tables of the M^{3}D schema mostly correspond to simple INSERT queries, the corresponding queries for FR are more complex. For instance, feeding the Bridge_Ord_Prod table requires a SQL query that (i) joins InfoPrdt from NR with both Fact_Order and Dim_date from FR to obtain the IdGroup associated to each order, and (ii) uses the extended operators for JSON manipulation to unnest the orderline array within InfoPrdt and to obtain the Asin of each product. This means that transformations may require a significant time and can be error-prone, so they may be unsuitable in specific settings such as those of real-time DWs.

To characterize the complexity of ETL formulation, we have used the same indicators (plus the number of queries and execution time) employed for the workload complexity (see Section 7.1) over the SQL statements used during ETL. The results are shown in Table 6. As expected, transforming source data to lead them in the FR schema requires queries that are more numerous, longer, with more operators, and also more diverse. Consequently, the execution times for ETL queries in FR are also significantly higher than those in M^{3}D. Note that no ETL is needed to feed the NR schema since all data are kept in their native form.

We close this section with some remarks about incremental ETL. Practical experience shows that incremental ETL is always more complex in terms of queries: in the simplest case, source data are timestamped, so only the data modified since the last run of the ETL must be extracted; otherwise, either application-based or log-based extraction must be implemented. In the worst-case incremental extraction is impossible or inconvenient, so all data must be extracted at each ETL run, then they are compared with the current content of the DW to determine which is the "delta" to be loaded. As to execution time, the performance of incremental ETL is normally better than the one of static ETL, because only part of the data is extracted and loaded; however, when incremental extraction is not implemented as mentioned above, performances get significantly worst because not only all data are to be extracted, but they even have to be compared to compute the delta. Considering that the adoption

Table 6: Complexity of ETL

Indicators	**M^{3}D**	**FR**
Query#	12,0	24,0
Length (sum)	1610	4510
Length (avg)	134.16	187.82
Length (stdev)	138.19	177.02
Operator#	13	25
Unique operator#	7	12
Top-10 operators	Seq Scan/5 Sort/2 Subquery Scan/2 Hash Left Join/1 Hash/1 GroupAggregate/1 HashAggregate/1	Seq Scan/11 Sort/2 Hash Join/2 Hash/2 Nested Loop/1 WindowAgg/1 Function Scan/1 Index Scan/1 Merge Join/1 Subquery Scan/1
Time (minutes)	65	224

of one technique or the other only depends on the characteristics of the source data and applications, not on those of the target schema, we can reasonably expect that the M^{3}D vs. FR arguments raised for static ETL will also hold in the incremental case.

7.4. Flexibility & extensibility

We start this section by observing that, differently from the FR one, the M^{3}D and NR schemata preserve the data variety existing in the data sources. This is particularly relevant for instance in self-service business intelligence scenarios, where data scientist will write ad-hoc queries to satisfy situational analysis needs [34]. Besides, mixing different models in an MMDW enables the achievement of higher flexibility in the modeling solutions taken, for instance when dealing with many-to-many relationships. An example is the Knows relationships between customers, that can be modeled by the arcs in a graph or through a bridge table. The experimental evaluation in Section 7.1 has shown the pros and cons of these solutions, as different queries perform differently on the two implementations. Ultimately, an MMDW allows choosing the implementation that better suits the workload in a specific scenario.

Within a schemaless setting, a further issue arising is that of the occasional presence, in some documents, of attributes not considered at design time (in our working example, EU and gold). Clearly, these attributes can be queried in the M^{3}D and NR schemata —while they cannot in the FR schema. Remarkably, this allows adopting querying approaches capable of coping with variable schemata and structural forms within a collection of documents (i.e., missing or additional fields, different names or types for a field, and different structures for instances). An example is *approximate OLAP* [10], where the user can include a concept in a query even if it is present in a subset of documents only; to make her aware

of the impact of variety, the query results are associated with some indicators describing their quality and reliability. The main indicators introduced to this end are *level support* and *query density*. The first one measures, for level l, the percentage of documents for which l is not missing (i.e., it has a value). The second one is defined for an aggregate query q as the percentage of groups returned by q that are complete in all their group-by levels. This is different from evaluating the mere support of levels, because the presence of more or less null values may vary depending on the selection predicates in the query. For instance, in our dataset, the support of both EU and gold is about 50%; the densities for queries Q1-11 and Q1-12, which use these attributes, are 69% and 35%, respectively. Concerning Q1-11, this indicates that the percentage of missing EU values for the chosen country is significantly lower than the overall support.

In summary, the advantages of MMDWs in terms of flexibility and extensibility can be referred to (i) querying, by preserving the variety of data sources and smoothly coping with the presence of variable source schemata, and (ii) modeling, by supporting alternative solutions so as to adapt the schema to the workload.

7.5. Evolvability

Software maintenance makes up for at least 50% of all the resources spent in a project [35]. In particular, an evolution in the schema of a DW is typically triggered either by a change in the schema of the data sources (which, for NoSQL sources, can be very frequent [36]) or by some new analysis requirement expressed by users. In turn, each schema evolution affects not only the database itself but also the surrounding applications (queries, views, reports, and ETL activities).

The multi-model implementation is partially schemaless, so it transparently supports evolution to some extent. The situation with the full-relational implementation is quite different. Indeed, even adding a couple of simple levels (as EU and gold in our case study) requires, at the very least, changing the relational schema of one or more tables, editing the ETL procedures, and migrating the data from the old schema to the new one. A more complex evolution, e.g., one involving a new many-to-many relationship, would have even more impact because it would require creating new tables. In case end-users ask for a full versioning of the schemata, the effort would be greater still. An M^3D schema represents a good trade-off here because most evolutions can be handled seamlessly with no impact on tables and ETL; clearly, a more invasive evolution (such as adding a new dimension or measure) would still require a change to the relational part of the schema and to the ETL.

To give a quantitative assessment of how the three solutions compared will react to evolution issues, we use the graph-theoretic metric proposed in [33], namely, *out-degree*, which has been empirically validated and shown to be the most reliable one. This metric operates on a graph-based representation (*evolution graph*) of schemata and queries. The evolution graph of a relational schema R is a directed graph that includes (i) a *relation node* R; (ii) one *attribute node*

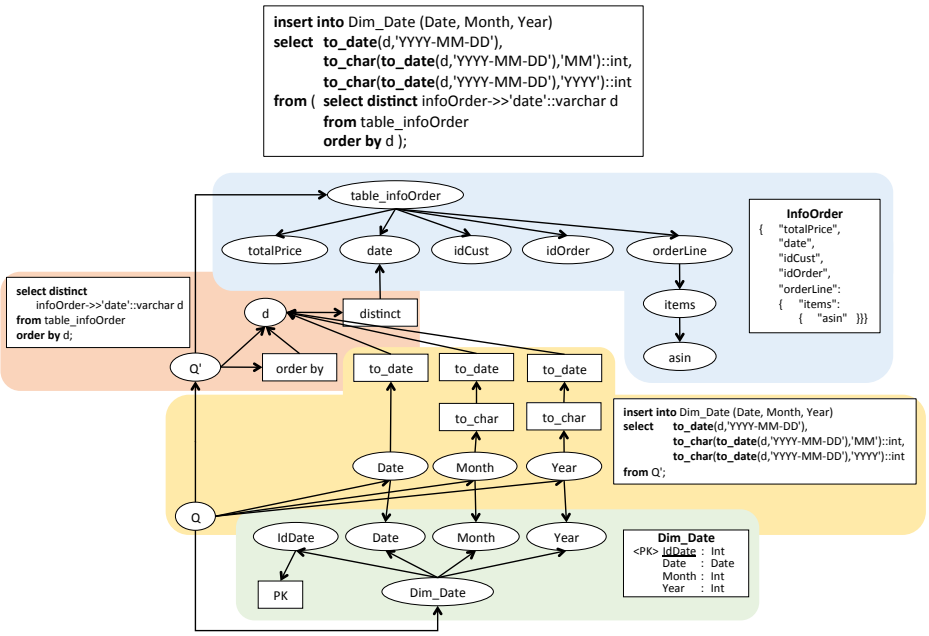


Figure 11: SQL formulation of an ETL query in PostgreSQL and corresponding evolution graph; the parts of the graph modeling each part of the query and the two schemata involved are highlighted in different colors

for each attribute of R; (iii) one *PK* node for the primary key of R; and (iv) a set of *schema relationships* directed from the relation node towards the other nodes. Similarly, the evolution graph of a query Q is a directed graph that includes (i) a *query node* Q; (ii) a set of attribute nodes corresponding to the schema of the query; (iii) a set of schema relationships directed from the query node towards the attribute nodes; and (iv) a *from relationship* from the query node to each relation schema used by Q. WHERE and HAVING clauses are modeled via a tree of logical operands to represent the selection formulae. Aggregate queries add a *GB node* connected to the attributes in the GROUP BY clause, plus one node per aggregate function. An example is proposed in Figure 11, which shows the ETL query that loads Dim_Date in the FR and M^{3}D schemata and the corresponding evolution graph. The out-degree of a node is the number of its outgoing arcs; the lower the total out-degree, the better the maintainability of the solution (the total out-degree of the evolution graph in Figure 11 is 34). Remarkably, this metric was specifically conceived and empirically validated for DWs, aimed at predicting the vulnerability of a DW to future evolutions so as to facilitate the comparison of alternative design solutions from the evolution point of view.

The results are summarized in Figure 12, which shows the evolution effort, estimated using the out-degree metric, for our three solutions, distinguishing between the effort for schema, the one for ETL, and the one for end-user queries

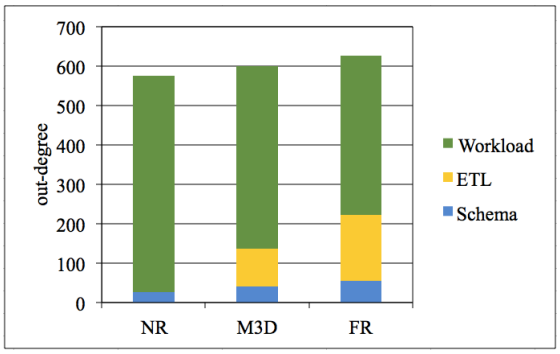


Figure 12: Evolution effort for schema, ETL, and end-user queries

(with reference to the WL2 workload, which is the one simulating end-user interaction). Unsurprisingly, the total evolution effort for the NR schema (which requires no ETL) is the lowest one, since the higher effort for evolving workload queries is largely compensated by the lower complexity of the schema and by the absence of ETL. The effort for the M^3D schema is lower than the one for the FR schema for two reasons: (i) in terms of schema, FR is less maintainable due to the larger number of primary and foreign keys defined; (ii) in terms of ETL, a relevant contribution to the complexity of FR is due to the presence of the three bridge tables (which are not used in M^3D).

8. Conclusion and research perspective

Handling big data variety, volume, and velocity is an important challenge for decision-making information systems. On the one hand, data lakes have been proposed to ensure flexible storage of raw data, but at the price of making analyses more complex. On the other hand, classical DW architectures provide an efficient framework for analyzing transformed and integrated data, but they fall short in natively handling data variety. Motivated by the emerging trend of MMDBMSs, in this work we have investigated the feasibility of a multi-model approach to DW based on an extension of the well-known star schema with schemaless data as dimensions and facts. The experiments we conducted in this work show that all queries of our multi-model OLAP workload can run over the proposed multi-model star schema in acceptable time compared to a full-relational implementation, and that the transformation and evolution overhead compared to a data lake implementation is reasonable.

Though devising complete guidelines and best practices for multi-model design is out of the scope of this paper, based on the results of our experimental evaluation we can observe that:

- The relational model is still more efficient, so it can be used, during logical design, for the data sources that can be smoothly transformed into relational form (i.e., those whose transformation does not entail a loss of

information content and can be accommodated within the time frame of ETL).

- Conversely, the data sources that hardly fit into the fixed structure of a relational schema, e.g., because their schema is not completely known in advance, should be left in their native form. This will cut the ETL effort on the one hand, will encourage flexibility and extensibility on the other.
- As to evolvability, schemaless data (i.e., graphs, JSON/XML documents, and key-value data) turned out to be better maintainable than relational tables. In fact, the evolution effort measured through the out-degree metric is substantially the same with the different schemaless models.
- The adoption of non-relational models is more suited for workloads that exploit some of the specific peculiarities of the different models, which ultimately may provide better performance than the relational implementation. For instance:
  - a key-value store is more suited for queries with high selectivity, and it is a performing solution in presence of cross-dimensional attributes;
  - a graph is a performing solution in presence of recursive hierarchies, and it achieves better performances when the query involves a long traversal of a limited number of nodes;
  - the best use case for nested models like JSON is when the query involves attributes contained within each document (i.e., it relieves from the burden of joining different tables); this is true, in particular, when a hierarchy includes a multiple arc.
- The verbosity of JSON and XML entails bigger tables which require more time to be scanned; thus, it should be compensated by properly indexing the attributes most used in the queries.

These observations are summarized in Table 7, which gives suggestions about the model(s) to be used for storing hierarchies depending on the specific multidimensional construct they include on the one hand, on the design goal (improve querying performance, resuce storage space, cut the ETL effort, encourage flexibility and extensibility, encourage evolvability) to be pursued on the other. In particular, consistently with what said above, when the design goal is to cut the ETL effort, preserving the native models is the best choice. When it comes to flexibility, extensibility, and evolvability, all schemaless models (i.e., G, JSON, KV, and XML) are substantially equivalent.

Overall, the advantages of MMDWs over relational DWs can be summarized as follows:

1. An MMDW will natively and efficiently support OLAP querying over large volumes of multi-model and multidimensional data, thus ensuring support to both volume, velocity, and variety.

Table 7: Best model(s) for different design goals and multidimensional constructs

Design goal	Standard hierarchy	Cross-dim. attribute	Non-strict hierarchy	Recursive hier.
Querying	R	R, KV	R, JSON	G
Storage	R	KV	R	R
ETL	native mod.	native mod.	native mod.	native mod.
Flex. & extens.	schemaless mod.	schemaless mod.	schemaless mod.	schemaless mod.
Evolvability	schemaless mod.	schemaless mod.	schemaless mod.	schemaless mod.

2. Storing data in their native model means reducing the data transformations required; hence, the effort for writing (time-consuming and error-prone) ETL procedures will be reduced in MMDWs, and the freshness of data in the DWs will be increased.

3. MMDWs will bridge the architectural gap between data lakes and DWs. We believe MMDWs will offer an effective architectural trade-off by enabling both OLAP multidimensional analyses and ad-hoc analytics on the same repository.

4. Schema evolution is a crucial issue in traditional DW architectures, since modifying relational schemata to accommodate new user requirements is a complex and expensive task. MMDWs can store schemaless data, so they will ensure more effective support to schema evolution [37].

5. Again thanks to their support of schemaless data, higher flexibility and extensibility will be granted, which will enhance analysis capabilities thus generating added value for users [38].

6. More specifically, key-value stores on the one hand, and the array constructs supported by document-based databases on the other, provide an alternative solution to model many-to-many relationships appearing in some multidimensional schemata.

Our experiments are encouraging enough to set a mid-term perspective of the research on MMDWs. The open research issues we envision can be summarized as follows:

- *Multidimensional design from multi-model databases.* The existing data-driven approaches to multidimensional design are based on detecting functional dependencies in single-model data sources, namely, relational, XML, linked-open data, JSON [39]. Using a multi-model data source for design requires integrating different techniques into a synergic methodology.

- *Conceptual models.* Existing conceptual models for DWs are mostly aimed at designing multidimensional schemata with a fixed structure. To take full advantage of the flexibility ensured by MMDWs, new models capable of coping with schemaless data (as naively done with the cloud symbol in Figure 2) are needed.

- *Best practices for logical design.* In presence of variety, several alternatives emerge for the logical representation of dimensions and facts [40]. Indeed, some combinations of models may be better than others when coupled with star schemata. A specific set of guidelines for the logical design of MMDWs is thus needed to find the best trade-off between performances, fidelity to source schemata, extensibility, and evolvability; this should also include the issues related to view materialization.

- *OLAP benchmark.* Effectively benchmarking MMDBMSs [2] and non relational DBMSs [19] is still a challenge. Providing a benchmark for MMDWs is a further challenge since it requires defining a dataset representative of DW volume and multi-model variety, as well as a full range of representative OLAP queries over this dataset.

- *Indexing.* PostgreSQL offers different types of indexes over multi-model databases, e.g., B-trees, hash, GiST (for geo data), GIN (for document and hstore data), etc. Ad hoc indexing strategies will have to be devised, in presence of variety, to cope with the specific features of multidimensional data and OLAP queries.

- *OLAP tools.* Last but not least, more sophisticated OLAP tools are required to let users benefit from the additional flexibility introduced by MMDWs while ensuring good performances. Specifically, there is a need for devising techniques to automatically generate efficient SQL queries over MMDWs from the (MDX-like or graphical) language used by the front-end.

Acknowledgement

This work was partially supported by the French National Research Agency as part of the "Investissements d'Avenir" through the IDEX-ISITE initiative CAP 20-25 (ANR-16-IDEX-0001), and the project VGI4bio (ANR-17-CE04-0012).

References

[1] P. Atzeni, F. Bugiotti, L. Rossi, Uniform access to NoSQL systems, Inf. Syst. 43 (2014) 117–133.

[2] J. Lu, I. Holubová, Multi-model databases: A new journey to handle the variety of data, ACM Comput. Surv. 52 (3) (2019) 55:1–55:38.

[3] T. Shimura, M. Yoshikawa, S. Uemura, Storage and retrieval of XML documents using object-relational databases, in: Proc. DEXA, Florence, Italy, 1999, pp. 206–217.

[4] V. Gadepally, P. Chen, J. Duggan, A. J. Elmore, B. Haynes, J. Kepner, S. Madden, T. Mattson, M. Stonebraker, The BigDAWG polystore system and architecture, in: Proc. HPEC, Waltham, MA, USA, 2016, pp. 1–6.

[5] T. Tsunakawa, Road to a multi-model database – making PostgreSQL the most popular and versatile database, presented at PGConf.ASIA, Tokyo, Japan (2017).
URL `https://www.pgconf.asia/EN/2017/day-1/#B2`

[6] R. Kimball, M. Ross, The data warehouse toolkit: the complete guide to dimensional modeling, 2nd Edition, Wiley, 2002.

[7] M. Chevalier, M. E. Malki, A. Kopliku, O. Teste, R. Tournier, Implementation of multidimensional databases with document-oriented NoSQL, in: Proc. DaWaK, Valencia, Spain, 2015, pp. 379–390.

[8] M. Boussahoua, O. Boussaid, F. Bentayeb, Logical schema for data warehouse on column-oriented NoSQL databases, in: Proc. DEXA, Lyon, France, 2017, pp. 247–256.

[9] M. L. Chouder, S. Rizzi, R. Chalal, EXODuS: Exploratory OLAP over document stores, Inf. Syst. 79 (2019) 44–57.

[10] E. Gallinucci, M. Golfarelli, S. Rizzi, Approximate OLAP of document-oriented databases: A variety-aware approach, Inf. Syst. 85 (2019) 114–130.

[11] C. Zhang, J. Lu, P. Xu, Y. Chen, UniBench: A benchmark for multi-model database management systems, in: Proc. TPCTC, Rio de Janeiro, Brazil, 2018, pp. 7–23.

[12] S. Bimonte, Y. Hifdi, M. Maliari, P. Marcel, S. Rizzi, To each his own: Accommodating data variety by a multimodel star schema, in: Proc. DOLAP@EDBT/ICDT, Copenhagen, Denmark, 2020, pp. 66–73.

[13] M. Chevalier, M. E. Malki, A. Kopliku, O. Teste, R. Tournier, Document-oriented models for data warehouses - NoSQL document-oriented for data warehouses, in: Proc. ICEIS, Rome, Italy, 2016, pp. 142–149.

[14] M. Chevalier, M. E. Malki, A. Kopliku, O. Teste, R. Tournier, Document-oriented data warehouses: Models and extended cuboids, extended cuboids in oriented document, in: Proc. RCIS, Grenoble, France, 2016, pp. 1–11.

[15] M. Chevalier, M. E. Malki, A. Kopliku, O. Teste, R. Tournier, Document-oriented data warehouses: Complex hierarchies and summarizability, in: Proc. UNet, Casablanca, Morocco, 2016, pp. 671–683.

[16] I. Ferrahi, S. Bimonte, M. Kang, K. Boukhalfa, Design and implementation of falling star - a non-redundant spatio-multidimensional logical model for document stores, in: Proc. ICEIS, Porto, Portugal, 2017, pp. 343–350.

[17] M. Chevalier, M. E. Malki, A. Kopliku, O. Teste, R. Tournier, Implementation of multidimensional databases in column-oriented NoSQL systems, in: Proc. ADBIS, Poitiers, France, 2015, pp. 79–91.

[18] A. Sellami, A. Nabli, F. Gargouri, Transformation of data warehouse schema to NoSQL graph data base, in: Proc. ISDA, Vellore, India, 2018, pp. 410–420.

[19] M. E. Malki, A. Kopliku, E. Sabir, O. Teste, Benchmarking big data OLAP NoSQL databases, in: Proc. UNet, Hammamet, Tunisia, 2018, pp. 82–94.

[20] Z. Ouaret, R. Chalal, O. Boussaid, An overview of XML warehouse design approaches and techniques, IJICoT 2 (2/3) (2013) 140–170.

[21] D. Boukraâ, M. A. Bouchoukh, O. Boussaïd, Efficient compression and storage of XML OLAP cubes, IJDWM 11 (3) (2015) 1–25.

[22] K. Dehdouh, Building OLAP cubes from columnar NoSQL data warehouses, in: Proc. MEDI, Almería, Spain, 2016, pp. 166–179.

[23] A. Castelltort, A. Laurent, NoSQL graph-based OLAP analysis, in: Proc. KDIR, Rome, Italy, 2014, pp. 217–224.

[24] H. B. Hamadou, E. Gallinucci, M. Golfarelli, Answering GPSJ queries in a polystore: a dataspace-based approach, in: Proc. ER, Salvador de Bahia, Brazil, 2019, pp. 189–203.

[25] M. Golfarelli, S. Rizzi, Data Warehouse Design: Modern Principles and Methodologies, McGraw-Hill, Inc., New York, NY, USA, 2009.

[26] P. E. O'Neil, E. J. O'Neil, X. Chen, S. Revilak, The star schema benchmark and augmented fact table indexing, in: Proc. TPCTC, Lyon, France, 2009, pp. 237–252.

[27] D. Löper, M. Klettke, I. Bruder, A. Heuer, Enabling flexible integration of healthcare information using the entity-attribute-value storage model, Health Inf. Sci. Syst. 1 (1) (2013) 9.

[28] J. Couto, O. T. Borges, D. D. Ruiz, S. Marczak, R. Prikladnicki, A mapping study about data lakes: An improved definition and possible architectures, in: Proc. SEKE, Lisbon, Portugal, 2019, pp. 453–578.

[29] F. Ravat, Y. Zhao, Data lakes: Trends and perspectives, in: Proc. DEXA, Linz, Austria, 2019, pp. 304–313.

[30] B. G. Inc., Architecture of agensgraph, `https://bitnine.net/blog-agens-solution/architecture-of-agensgraph/`, [Online; accessed 20-Nov-2020] (2017).

[31] N. Francis, A. Green, P. Guagliardo, L. Libkin, T. Lindaaker, V. Marsault, S. Plantikow, M. Rydberg, P. Selmer, A. Taylor, Cypher: An evolving query language for property graphs, in: G. Das, C. M. Jermaine, P. A. Bernstein (Eds.), Proc. SIGMOD, Houston, TX, USA, 2018, pp. 1433–1445.

[32] S. Jain, D. Moritz, D. Halperin, B. Howe, E. Lazowska, SQLShare: Results from a multi-year SQL-as-a-Service experiment, in: Proc. SIGMOD, San Francisco, CA, USA, 2016, pp. 281–293.

[33] G. Papastefanatos, P. Vassiliadis, A. Simitsis, Y. Vassiliou, Metrics for the prediction of evolution impact in ETL ecosystems: A case study, J. Data Semantics 1 (2) (2012) 75–97.

[34] A. Abelló, J. Darmont, L. Etcheverry, M. Golfarelli, J. Mazón, F. Naumann, T. B. Pedersen, S. Rizzi, J. Trujillo, P. Vassiliadis, G. Vossen, Fusion cubes: Towards self-service business intelligence, IJDWM 9 (2) (2013) 66–88.

[35] G. Papastefanatos, P. Vassiliadis, A. Simitsis, Y. Vassiliou, Design metrics for data warehouse evolution, in: Proc. ER, Barcelona, Spain, 2008, pp. 440–454.

[36] S. Scherzinger, S. Sidortschuck, An empirical study on the design and evolution of NoSQL database schemas, in: Proc. ER, Vienna, Austria, 2020, pp. 441–455.

[37] S. Scherzinger, M. Klettke, U. Störl, Managing schema evolution in NoSQL data stores, in: Proc. DBPL, Riva del Garda, Italy, 2013, pp. 1–10.

[38] N. Berkani, L. Bellatreche, S. Khouri, C. Ordonez, Value-driven approach for designing extended data warehouses, in: Proc. DOLAP@EDBT/ICDT, Lisbon, Portugal, 2019, pp. 1–5.

[39] O. Romero, A. Abelló, A survey of multidimensional modeling methodologies, IJDWM 5 (2) (2009) 1–23.

[40] I. Ferrahi, S. Bimonte, K. Boukhalfa, A model & DBMS independent benchmark for data warehouses, in: Proc. EDA, Lyon, France, 2017, pp. 101–110.