GAMM: Graph-Based Agile Multidimensional Model*

†

Redha Benhissen
ERIC Laboratory
University of Lyon 2
Bron, France
redha.benhissen@univ-lyon2.fr

Fadila Bentayeb
ERIC Laboratory
University of Lyon 2
Bron, France
fadila.bentayeb@univ-lyon2.fr

Omar Boussaid
ERIC Laboratory
University of Lyon 2
Bron, France
omar.boussaid@univ-lyon2.fr

ABSTRACT

The advent of big data has continuously offered new opportunities for analysis due to the growth and diversification of data sources. However, traditional data warehouses based on multidimensional models have limitations when it comes to changing and evolving. In recent years, innovative work in the field of data warehouse evolution has been performed; it has mainly focused on the management of big data and the chronological evolution of models in terms of their structures and data. However, an agile decision model is still required. We propose in this paper an agile approach for schema evolution in data warehouses that allows designers to integrate new data sources and take into account new user needs in order to enrich the analysis possibilities of data warehouses in a flexible way. Our approach is based on a multi-version evolutionary schema model. The data instances corresponding to the different versions of the schema are stored in a graph data warehouse. A meta-model will allow the management of the warehouse schema versions. We also propose evolution functions on the schema level. We validate our approach with a software prototype and a case study that illustrates queries on schema versions, cross-queries and the runtime of our approach.

1 INTRODUCTION

Data warehouses (DWs) are the core of modern decision-making systems, enabling decision-makers to consider their environments for better decision-making. Since the appearance of DWs [17, 18], the warehousing approach has become an important research field in which many problems still need to be solved, particularly problems related to the evolutionary aspect of DWs. Indeed, DWs based on classical multidimensional models centralise data from different sources to meet the analysis needs of users. One of the key tasks that must be carried out in a successful data warehousing process is the definition of the warehouse model according to the data sources and analysis needs. With the advent of big data, there has been a proliferation of data sources, which is leading to new analysis needs. Big data offers due to the growth and diversification of data sources. A DW's schema is designed to meet predefined analysis needs; if these needs change, it can be costly to change the schema. The classical multidimensional model, based on the star schema and its variants, has limited possibilities when it comes to change, and its evolution is complex. These limitations are related to the fixed star model: it is created for analysis needs that are known in advance. We are interested in the evolution of data warehouses in this context. Even though this problem is not new, as many works

concerning it exist in the literature, it still raises new research questions. In this article, we propose an agile multidimensional model for big data called the graph-based agile multidimensional model (GAMM), which is based on an extension of the classical multidimensional model, to support chronological evolution on the conceptual level and the evolution of the graph structure on the logical and physical levels. GAMM allows the evolution of a schema in a data warehouse by creating a new version of the schema at each evolution using evolution functions. Each version corresponds to a data instance extracted from an agile graph data warehouse. A meta-model has been proposed to manage the different schema versions. The rest of the article is organised as follows. It starts by presenting a running motivating example (SECTION 2). Then, related work and a comparative description of evolution in data warehouses are presented (SECTION 3). In SECTION 4, we present GAMM, its architecture, the meta-model for schema version management, the agile graph-based warehouse for data storage and the formalisation of the model. We then describe in SECTION 5 the schema evolution process of our model, which is performed by the implemented evolution functions. In SECTION 6, we give an example of an instance and queries. In SECTION 7, we describe the use of our software prototype, which has a 3-tier architecture, and we present a use case based on the Star Schema Benchmark (SSB) dataset [1] [2] for functional validation and to study the runtime of our model. Finally, a conclusion and future research topics are presented.

2 RUNNING MOTIVATING EXAMPLE

We define agility in a multidimensional model as its organisational capacity to build data warehouses in a scalable way while prioritising the goals of business teams. It is about responding to new business objectives and the integration of new data sources in a flexible and incremental way while maintaining all previous builds. This flexibility delivers more business value to decision-makers and allows them to better manage their environment. We use the dimensional fact model (DFM) formalism [13] to represent the conceptual schemas of our example (FIGURE 1).
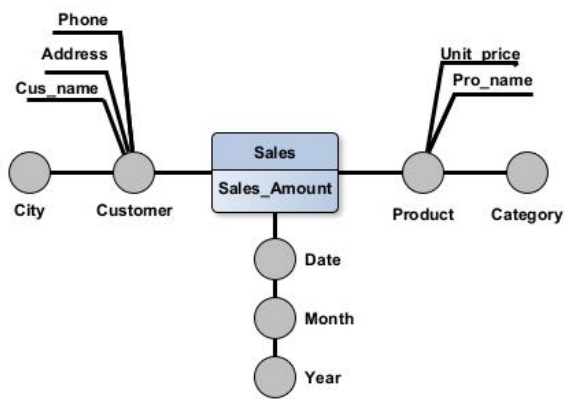
Consider the initial schema at t_0 in FIGURE 1a, which is composed of a fact SALES and three dimensions, the PRODUCT, CUSTOMER and DATE dimensions. For the sake of consistency, we will use this schema configuration as a running example during the presentation of all parts of our proposal. The dimension PRODUCT is described by the attributes *Product_Name* and *Unit_Price* and the hierarchical level CATEGORY. This dimension therefore has a single hierarchy on the analysis axis (PRODUCT, CATEGORY). The dimension CUSTOMER is described by the attributes *Customer_Name*, *Address* and *Phone*, and the hierarchical level CITY.

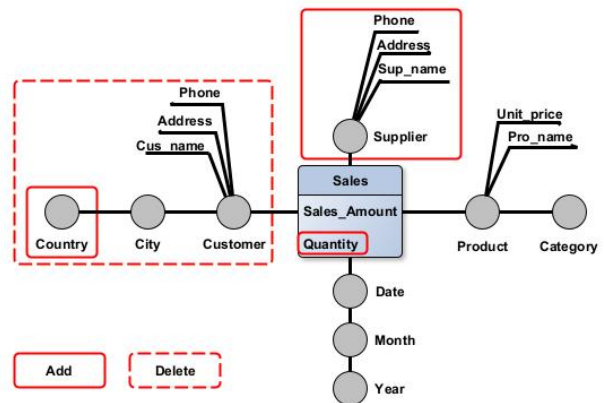*Produces the permission block, and copyright information
†The full version of the author's guide is available as `acmart.pdf` document

[1]https://jorgebarbablog.wordpress.com/2016/03/21/how-to-load-the-ssb-schema-into-an-oracle-database/
[2]https://github.com/Kyligence/ssb-kylin

(a) Schema version at t_0 (b) Schema version at t_3

Figure 1: Multidimensional model schemas for retail sales using DFM formalism

The dimension DATE has two hierarchical levels, i.e. MONTH and YEAR. The fact SALES is indicated by the measure SALES_AMOUNT.

Let us assume that this schema undergoes the following changes: (i) a new level COUNTRY is added to the dimension CUSTOMER dimension at t_1; (ii) a new dimension SUPPLIER with the attributes *Sup_Name*, *Address* and *Phone* is added at t_2; and (iii) the dimension CUSTOMER is deleted and a measure QUANTITY is added at t_3. The conceptual schema after these changes is depicted in FIGURE 1B. (The dimension CUSTOMER and its hierarchies have been left on the diagram with a broken line box for explanatory purposes.)

If, at the conceptual level, schema changes are quite easy to represent, the concretisation of these changes is quite complex on the logical and physical levels. Indeed, multidimensional modelling based on the entity–relationship (ER) model presents a resistance to schema changes that we describe in detail in the following section.

3 RELATED WORK

At the end of the 1990s, decision-making systems based on multidimensional modelling were progressively implemented to provide tools and methods that allow decision-makers to better visualise their environments and to offer them adequate support for strategic decision-making [17]. The concept of multidimensional modelling, particularly the evolutionary aspect, has become a research field in its own right [15]. Indeed, the evolution of multidimensional models has been the subject of several studies proposing different approaches. These works can be grouped into two categories: (i) those focused on data evolution [2, 10, 12, 14, 20, 22], and (ii) those focused on schema evolution without change-history support [7–9, 11, 16, 19, 25] and with change-history support [1, 3, 21, 23, 24, 26].

In our literature study (TABLE 1), we compared the various existing research works based on the evolutionary aspect of the schemas and data, the history of these evolutions, data integrity and the cross-queries.

The work in the literature on evolving multidimensional models is based on the ER model, which limits the evolution of these

Table 1: Comparative review of previous work

Study	Temporal evolution of the data	Schema evolution	History	Data integrity	Cross-queries
[10]	√				
[16]		√		√	
[9]		√		√	
[20]	√				
[7]		√			
[21, 26]		√	√		√
[6]		√	√		
[8]		√		√	
[23]		√	√		√
[19]		√		√	
[11]		√		√	
[24]		√	√		√
[14]	√				
[25]		√		√	
[1, 4]		√	√		√
[12]	√				
[22]	√			√	
[2, 3]	√	√	√		√

multidimensional models, particularly in terms of the schema. Indeed, these solutions proposed either implicit or explicit versioning. Implicit versioning was accomplished by providing temporal extensions to the schema, such as the use of the *bitmap* to share data between several reference versions, which is constraining in terms of queries, especially in a big data context. Meanwhile, explicit versioning was performed through the duplication of data and entities (elements of the schema), which is also constraining due to the quantity of data generated and the complex structure that must be managed. Moreover, the extension of relational entities causes a loss in the integrity of the data due to the creation of NULL values.

The evolution of multidimensional models in various previous research works is both logically and physically constrained. Indeed, the modelling of facts connected to several dimensions considerably limits the possibilities of evolution because of a design based on a multidimensional schema. Moreover, the relational

structure used in almost all of the previous studies is not adapted to this evolution because of the tabular structure of the entities and because each schema is established for a specific need, which means that updating these schemas is complex. Additionally, the creation of multiple versions of the facts with different levels of granularity considerably complicates the cross-queries. The approach that we propose allows the evolution of the schemas with a history of all previous versions. The approach adopted for the formalisation of our model will facilitate the incremental integration of data.

4 GRAPH-BASED AGILE MULTIDIMENSIONAL MODEL

4.1 Architecture of GAMM

GAMM is a flexible approach to schema and data evolution in data warehouses. The model allows designers to integrate new data sources and accommodate new user requirements to enrich the analytical capabilities of the data warehouse. It is an approach based on a multi-version scalable schema model and a data warehouse stored in a unique global graph database (Figure 2).
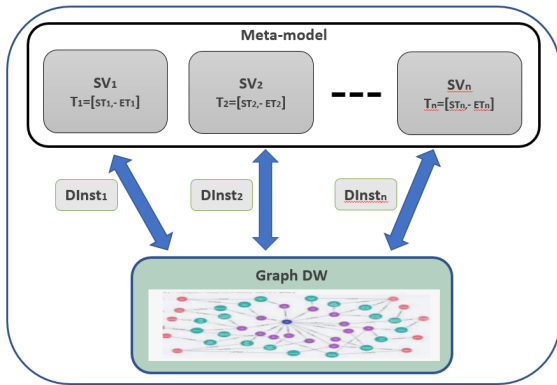

Figure 2: Architecture of GAMM

Each schema version (SV) is valid for a period of time (T) characterised by a Starting_Time (ST) and an Ending_Time (ET), and it corresponds to a data instance (DInst) extracted from the graph-based data warehouse (GDW). A meta-model is implemented for the management of schema versions whose validity periods are in sequential order. We will develop in the following subsections the meta-model of GAMM for schema versioning (Subsection 4.2), the graph-based data warehouse for data storage, the rules for moving from a classical data warehouse to a GDW (Subsection 4.3) and the formalisation of GAMM (Subsection 4.4).

4.2 Meta-model of GAMM

We have built a meta-model that enables the identification of all the schema structures that exist in the model in order to manage the many schema versions arising from evolution in GAMM. The foundation of this meta-model is a single class called *Version* that contains the version identification, an *ST* and an *ET* to specify the duration of each version's validity and each version's status. We present a diagram of the meta-model in Figure 3.

The organisation of each schema is determined by the *Version* class, which serves as the hub around which all other classes revolve. The diagram effectively establishes the facts pertaining
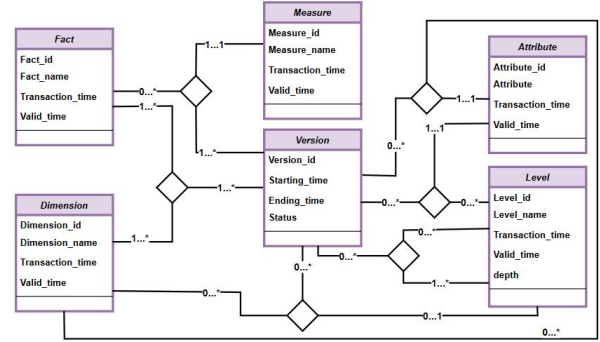

Figure 3: Meta-model of GAMM

to the versions, as well as the measurements and dimensions related to the versions. Additionally, it establishes the attributes connected to the dimensions/levels, the order in which these levels are determined and the hierarchical levels connected to the dimensions at any given time.

A new version is created with the appropriate schema each time the schema is changed by the addition/deletion of a new entity or link between existing entities (fact/dimension, dimension/level, level/level, etc.). The ST of the new version is set to the date of creation, the ET has the value Null until the end of the schema's validity is determined by the creation of another version and the status takes the value Last to indicate the active version. In a similar way, the ET from the previous version has been upgraded and has an Old status. The active schema and the additional data sources will be taken into account as the data integration process is modified. Each schema, then, corresponds to a specific piece of data that has been integrated into a GDW and is identifiable by its chronological label TT in explicit queries or by the relationships of entities in implicit queries, especially for cross-queries. The VT parameter will be used in an analysis context. The data integration process will be readapted according to the active schema and the new data sources.

4.3 Graph data warehouse

The conception of the GDW is characterised by the separation of business concepts from their descriptors (attributes), allowing each entity to have an independent evolution. The graph formalisation and implementation in a graph-oriented database were adopted to overcome the constraints generated by the use of an ER model. Indeed, the graph formalisation offers more flexibility for the model, particularly in terms of evolution [5]. In addition to the representative quality of the interconnected data, the use of graphs was preferred due to the absence of integrity constraints, particularly for the management of keys during the evolution of the schemas; the absence of a pre-established schema; and the possibility of representing each value of a tuple (or all of the tuple according to the need) using a node of the graph. This graph implementation is unlike relational tables, which are order schemas composed of horizontal lines and vertical columns in which the addition/removal of a column affects the whole structure. A graph-oriented NoSQL database (GDB) was chosen to represent the GDW because of its concordance with the proposed graph formalisation. Indeed, graph-oriented databases present data in the form of a graph (vertex/edge) using physical pointers between nodes, thus avoiding joins in queries, which is

advantageous, particularly in a big data context. GDBs are also characterised by the absence of a data type and the possibility of integrating information into the relationships between data. Additionally, the concept of graphs has been adopted mainly to allow us to carry out an advanced analysis on the model; this makes it possible to perform an online analysis to produce explicative and predictive models. The business concepts, as well as the descriptors, will be represented by nodes, and the relations between these concepts will be represented by edges (Figure 4). In this context, we have established the following rules for moving from a classical multidimensional model to the graph model:

(1) Each tuple of a fact is represented by its own node.
(2) Each (business/descriptor) value of a dimension, level or attribute is represented by its own node.
(3) All relationships are represented by edges.
(4) The fact nodes contain the measures.
(5) The facts nodes are directly related to the dimension nodes.
(6) The level nodes are related to a dimension / other level nodes according to their depth.
(7) The attributes nodes are directly related to the dimension/level nodes.
(8) The levels constitute hierarchies according to axes of analysis organised from the finest to the coarsest level of aggregation.
(9) All entities have a chronological tag (Valid_Time (VT) and Transaction_Time (TT)) in accordance with the principles of bi-temporal databases[14].

By applying these rules of passage to the schema of our running example, we obtain the schema represented in Figure 4. We consider the three dimensions Product, Customer and Date, the fact Sales, the measure *Sales_Amount* and the hierarchical levels Category for Product and City for Customer. Dimensions and levels are described by nodes representing the descriptors.
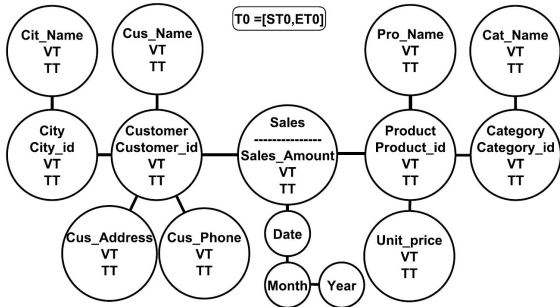

Figure 4: Logical schema of GAMM(t_0)

This model separates the business concepts from the descriptors (attributes) in order to allow an independent evolution of each entity. Indeed, facts, dimensions and hierarchy levels are represented by nodes corresponding to the basic multidimensional concepts to which other nodes representing descriptors are linked. A temporal label consisting of a VT and a TT has been assigned to all entities to allow the identification of the different instances and to indicate which version they belong to. Thus, the data of each schema version can be consulted using the TTs that belong to the time interval of validity of the schema concerned. This representation allows us to provide flexibility in the evolution of the model both at the schema level and at the data level while preserving the history of these evolutions.

Indeed, the data instances in the GDW evolve incrementally and without redundancy. All data instances of previous schemas will be available for consultation. Thus, the deletion of any entity from the schema is done only at the meta-model level, without deleting the data corresponding to this entity. In our article, we are interested in the evolution of the schema, and a formalisation of this model is proposed in the following subsection.

4.4 Formalisation of GAMM

On the conceptual level, the GAMM approach represents an extension of classical multidimensional modelling based on the *fact, measure, dimension, level and hierarchy* concepts. Indeed, due to the evolution of the schema over time and for historical purposes, the temporal concept has been introduced according to the following definitions.

Definition 1: GAMM is represented as follows:

$$GAMM(t) = \{F, D, FAssoc[F, D](t)\}.$$

$GAMM(t)$ represents the schema version at a time t.
$t \in T = [ST, ET]$ represents the period of validity of the schema version, where ST is the *Starting_Time* of the version and ET is the *Ending_Time* of the version.
$F = \{f_i(t)\}, i \in [1, *]$, represents the set of facts at the moment t.
$f_i(t)$ represents the fact f_i at a time t.
$D = \{d_j(t)\}, j \in [1, *]$, represents the set of dimensions according to which $f_i(t)$ is analysable at a time t.
$d_j(t)$ represents the dimension d_j at a time t.
$FAssoc[F, D](t) : f_i(t) \implies \{d_j(t), ST, ET\}$, where $j \in [1, *]$, represents the association function of the set of dimensions $\{d_j(t)\}$ and the fact $f_i(t)$ at a time t.
Example:

$$GAMM(t_0) = \{\{Sales\}, \{Customer, Product\}, \{Sales \implies Customer,$$
$$Product\}\}$$

and $ST_0 = < t_0 < ET_0$.

Definition 2: A fact represents a subject analysed by GAMM. It is defined as follows:

$$F(t) = \{F\_Id, TT, VT, [M], MAssoc[F, M](t)\}.$$

F_Id represents the fact identifier.
$TT = Transaction\_Time$ represents the period during which a fact is stored in the model.
$VT = Valid\_Time$ represents the period during which a fact is true in relation to reality.
$[M] = \{m_k(t)\}, k \in [1, *]$, represents the set of measures associated with the fact at a time t.
$m_k(t)$ represents a measure m_k at an instant t.
$MAssoc[F, M](t) : f_i(t) \implies \{m_k(t), ST, ET\}$ represents an association function of the set of measures $\{m_k(t)\}$ and the fact $f_i(t)$ at a time t, where ST is the *Starting_Time* and ET is the *Ending_Time*.
Example:

$$F(t_0) = \{Sales\_Id, Transaction\_Time_o,$$
$$Valid\_Time_o, M_o, MAssoc[F, M](t_o)\}.$$

$M_o : \{Sales\_Amount\}.$
$MAssoc[F, M](t_o) : \{Sales \implies Sales\_Amount\}.$

Definition 3: A measure is an indicator allowing the analysis of the business subject represented by the fact; it is numerical and aggregable and is defined as follows:

$$M(t) = \{M\_Id, Measure, TT, VT\}.$$

M_Id represents the measure identifier.
$Measure$ represents the indicator.
$TT = Transaction\_Time$ represents the period during which a measure is stored in the model.
$VT = Valid\_Time$ represents the period during which a measure is true in relation to reality.
Example:

$$M(t_o) = \{Sales\_Amount\_Id, Sales\_Amount, Transaction\_Time_o,$$

$$Valid\_Time_o\}.$$

Definition 4: A dimension is an axis of analysis according to which the business subject is analysed. It determines the level of detail of the measures and is defined as follows:

$$D(t) = \{D\_Id, TT, VT, [A], [L], DAssoc_a[D, A](t),$$

$$DAssoc_l[D, L](t)\}.$$

D_Id represents the dimension identifier.
$TT = Transaction\_Time$ represents the period for which a dimension is stored in the model.
$VT = Valid\_Time$ represents the period during which a dimension is true in relation to reality.
$[A] = \{a_i(t)\}, i \in [1, *]$, represents the set of attributes associated with the dimension at a time t.
$a_j(t)$ represents the attribute a_j at a time t.
$[L] = \{l_k(t)\}, k \in [0, *]$, represents the set of levels associated with the dimension at a time t.
$DAssoc_a[D, A](t) : d_j(t) \implies \{a_i(t), ST, ET\}$ represents the association function of the set of attributes $\{a_i(t)\}$ and the dimension $d_j(t)$ at a time t, where ST is the $Starting\_Time$ and ET is the $Ending\_Time$.
$DAssoc_l[D, L](t) : (d_j(t) \implies \{l_k(t), ST, ET\}$ represents the association function of the set of levels $\{l_k(t)\}$ and the dimension $d_j(t)$ at a time t, where ST is the $Starting\_Time$, ET is the $Ending\_Time$ and $j \in [0, *]$.
Example:

$$D(t_0) = \{Product\_Id, Transaction\_Time_o, Valid\_Time_o,$$

$$A_0, L_0, DAssoc_a[D, A](t_0), DAssoc_l[D, L](t_0)\}.$$

$A_0 : \{Product\_Name, Unit\_Price\}.$
$L_0 : \{Category\}.$
$DAssoc[D, A](t_o) : \{Product \implies Product\_Name, Unit\_Price\}.$
$DAssoc_l[D, L](t_0) : \{Product \implies Category\}.$

Definition 5: A hierarchy is a projection of analysis by level along the axis defined by the dimension. It is organised from the finest to the coarsest granularity level, thus offering analysis possibilities for ascending groupings through *roll-up* and descending groupings through *drill-down*. The hierarchy is defined as follows:

$$H(t) = \{L, R_h[L_j, L_k](t)\}.$$

$L = \{l_i(t)\}, i \in [1, *]$, represents the set of aggregation levels constituting a hierarchy H(t) at a time t.
$R_h[l_j, l_k](t)$ represents the aggregation function between the different levels $\{l_i(t)\}$ constituting a hierarchy $H(t)$ at a time t, with $j \in [1, *], k \in [1, *]$ and $j \neq k$.

Assume that $D_i(t) \in D$ with $i \in [1, *], \forall H_j(t) \in H$ with $j \in [1, *]$, and $L_1^{h_j}(t)$ is directly related to $D_i(t)$ so that

$$D_i(t) \prec L_1^{h_j}(t) \prec L_2^{h_j}(t) \prec ... \prec L_k^{h_j}(t) \prec All.$$

$D_i(t)$ represents the finest level of aggregation of the analysis axis.
$L_k^{h_j}(t)$ represents the coarsest level of aggregation in the hierarchy h_j.
All represents the global aggregation level of the dimension.
Example: Analysis projection by COUNTRY and CITY for the CUSTOMER dimension can be performed as follows:

$$D(t) = (Customer) \prec L_1^{h_j}(t) = (City) \prec L_2^{h_j}(t) = (Country) \prec All.$$

Definition 6: A level represents the degree of detail of an analysis perspective according to a given hierarchy. It is defined as follows:

$$L(t) = \{L\_Id, TT, VT, A, LAssoc[L, A](t)\}.$$

L_Id represents the level identifier.
$TT = Transaction\_Time$ represents the period during which a level is stored in the model.
$VT = Valid\_Time$ represents the period during which a level is true compared to reality.
$A = \{a_i(t)\}, i \in [1, *]$, represents the set of attributes associated with the level at a time t.
$LAssoc[L, A](t) : L_j(t) \implies \{a_i(t)\}$ represents the association function of the set of attributes $\{a_i(t)\}$ and the level $L_j(t)$ at a time t.
Example:

$$L(t_0) = \{Category\_Id, Transaction\_Time_o,$$

$$Valid\_Time_o, A_0, LAssoc[L, A_0](t_0)\}.$$

$A_0 : \{Category\_Name\}.$
$LAssoc[L, A_0](t_0) : \{Category \implies Category\_Name\}.$

Definition 7: An attribute is an element of description of a dimension or a hierarchical level to which it is associated. It is defined as follows:

$$A(t) = \{A\_Id, Attribute, TT, VT\}.$$

A_Id represents the attribute identifier.
$Attribute$ represents the value of the description attribute.
$TT = Transaction\_Time$ represents the period during which an attribute is stored in the model.
$VT = Valid\_Time$ represents the period during which an attribute is true with respect to reality.
Example:

$$A(t) = \{Product\_name\_Id, Product\_name, Transaction\_Time_o,$$

$$Valid\_Time_o\}.$$

5 SCHEMA EVOLUTION IN GAMM

The warehouse schema is likely to evolve to meet new analysis needs and to be able to handle the new data sources required to meet those needs. Indeed, GAMM is a multi-version evolutionary model in which each version represents the state of the schema at time *t*. The evolutions are performed according to the following function:

$$GAMM(t_{n+1}) = GAMM(t_n) + Evolution\_Operation(t_n).$$

where GAMM(t_{n+1}) represents the schema after the evolution, GAMM(t_n) represents the schema before the evolution and

Evolution_Operation(t_{n+1}) represents the evolution to be applied to a given version according to the type of this evolution. The evolution operators are defined in TABLE 2.

Table 2: Schema evolution operators

Evolution	Evolution operator	Description
Add an attribute	Add_attribute(a1,d1), Add_attribute(a1,l1)	Add the attribute a1 to the dimension d1 or the level l1
Add a measure	Add_measure(m1,f1)	Add the measure m1 to the fact f1
Add a dimension	Add_dimension(d1,f1,«a»)	Add the dimension d1 to the fact f1 with attributes «a»
Add a level	Add_level(l1,d1), Add_level(l1,l2)	Add the level l1 to the dimension d1 or to the level l2
Add a fact	Add_fact(f1,«d»,«m»)	Add the fact f1 to the dimensions «d1» with measures «m»
Delete an entity	Delete_entity(e)	Delete an entity e (fact, measure, dimension, level or attribute)
Create a relationship	Add_link(e1,e2)	Create a link between entity e1 and entity e2
Delete a relationship	Delete_link(e1,e2)	Delete a link between entity e1 and entity e2

For the function *Delete_entity*, several cases are considered according to the type of the entity to be deleted and the objective of the user. The operations performed by the model according to each case are not described in detail in this article due to the lack of space. Additionally, the modification of an entity will be described when the chronological evolution in terms of data is discussed. In what follows, we reuse our current example to propose the previously described schema modifications that correspond to three new analysis needs and illustrate for each of these needs the evolution of the schema, as well as the appropriate evolution functions for each operation.

5.0.1 Example 1: Add a new level COUNTRY to the dimension CUSTOMER. The addition of a level COUNTRY to the level CITY of the dimension CUSTOMER, which is represented by the schema in Figure 5, creates a new hierarchy on the axis of analysis (Customer, City, Country), thus responding to a new need for a more detailed geographic analysis using the level COUNTRY.
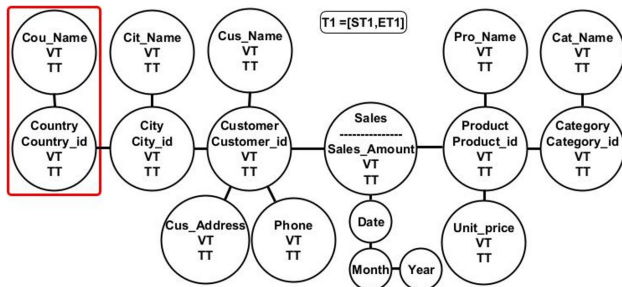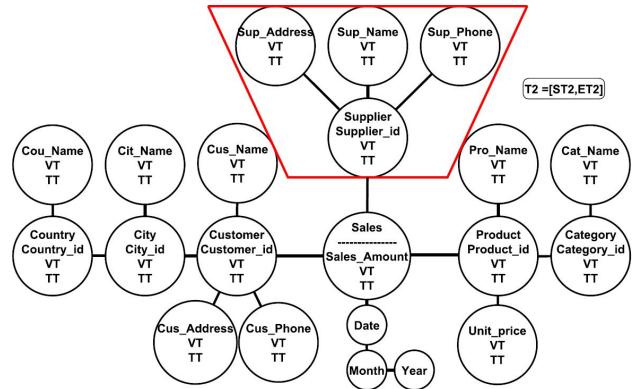

Figure 5: Logical schema of GAMM(t_1)

The addition of this new hierarchy does not require any changes to the factual level of *Sales*, nor does it require changes to the measures associated with *Sales* since the added level does not represent a base level and therefore does not correspond to the finest level of granularity of the *Customer* dimension. However, the analysis queries will need to be updated, especially for the generation of the OLAP cubes that are built. The evolution function corresponding to this operation is represented as follows:

$$GAMM(t_1) = GAMM(t_0) + Add\_level(\text{COUNTRY}, \text{CITY}).$$

5.0.2 Example 2: Add a new dimension SUPPLIER. The addition of a new dimension SUPPLIER with the attributes *Sup_Name*, *Address* and *Phone*, which is represented by the schema in Figure 6, offers the model a new axis of analysis, thus responding to a new analysis need using the added dimension.
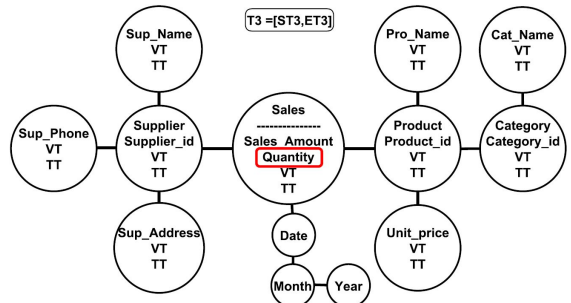

Figure 6: Logical schema of GAMM(t_2)

This new axis of analysis, created by the addition of the dimension SUPPLIER, increases the level of granularity of the measure present in the fact Sales and modifies the schema of the model by creating the new version. Consequently, new instances of these measures, as well as of the corresponding fact, will be created to respond to this new level of detail. The old instances will be kept for historical purposes. The evolution function corresponding to this operation is represented as follows:

$$GAMM(t_2) = GAMM(t_1) + Add\_dimension(\text{SUPPLIER}, \text{SALES},$$
$$[Sup\_Name, Sup\_Address, Sup\_Phone]).$$

5.0.3 Example 3: Delete a dimension CUSTOMER with hierarchical levels and add a measure QUANTITY. The addition of this new measure, which is represented by Figure 7, meets a new analysis need, namely the ability to determine the quantity of sales.

Figure 7: Logical schema of GAMM(t_3)

This measure is represented only by the dimensions PRODUCT and SUPPLIER, which implies the removal of the dimension CUSTOMER from the schema. This operation decreases the level of granularity of the measures associated with the fact SALES. Therefore, new instances of these measures will be created to meet this new level of detail. The evolution function corresponding to this operation is represented as follows:

$$GAMM(t_3) = GAMM(t_2) + Delete\_entity(\text{CUSTOMER})$$

$$+ Add\_measure(\text{QUANTITY}, \text{SALES}).$$

The evolutions in GAMM are characterised by the creation of a new schema version for each modification of the schema. $GAMM(t_0)$ represents the initial model; $GAMM(t_1)$ represents the first evolution, which corresponds to the addition of a hierarchical level; $GAMM(t_2)$ represents the second evolution, which corresponds to the addition of a dimension; and $GAMM(t_3)$ represents the last evolution, which corresponds to the addition of a measure and the deletion of a dimension.

6 QUERIES IN GAMM

In most of the research works related to the evolution of classical multidimensional models based on the ER model [3], the recognised approach is the creation of a new fact table for each version, particularly in the case of a change in the level of granularity of the measures. Here, we present an example of a fact instance in a relational model corresponding to the three schema versions in our running example. Tables 3, 4 and 5 show extracts of versions of the SALES fact table, which displays the fact instances according to the different schema evolutions.

Table 3: Fact v1

Sale_id	Cus_id	Pro_id	Orderdate	Sales_Amt
S1001	Cus001	Pro001	15062020	1800

Table 4: Fact v2

Sale_id	Cus_id	Pro_id	Sup_id	Orderdate	Sales_Amt
S2001	Cus001	Pro001	Sup001	12072020	1200
S2002	Cus001	Pro001	Sup002	25072020	1500

Table 5: Fact v3

Sale_id	Pro_id	Sup_id	Orderdate	Sales_Amt
S3001	Pro001	Sup001	08082020	2000
S3002	Pro001	Sup002	22082020	2500

In this approach, the queries on each version are performed, respectively, on the fact table corresponding to this version. Additionally, cross-queries are done through the union of several queries on different versions of the fact table and an alignment of the granularity level of the results, which becomes very complex, especially when there are many versions.

We represent the same instances with our approach based on graphs in FIGURE 8. Indeed, the TT_1 nodes represent the Sales_amount between the customer Cus001 and the product PRO001 in the version valid for $T_1 = [01062020, 30062020]$. The TT_2 nodes represent the Sales_amount between the customer Cus001, the product PRO001 and the suppliers SUP001 and SUP002 in the version valid for $T_2 = [01072020, 31072020]$, and the TT_3 nodes represent the Sales_amount between the product PRO001 and the suppliers SUP001 and SUP002 in the version valid for $T_3 = [01082020, 31082020]$.
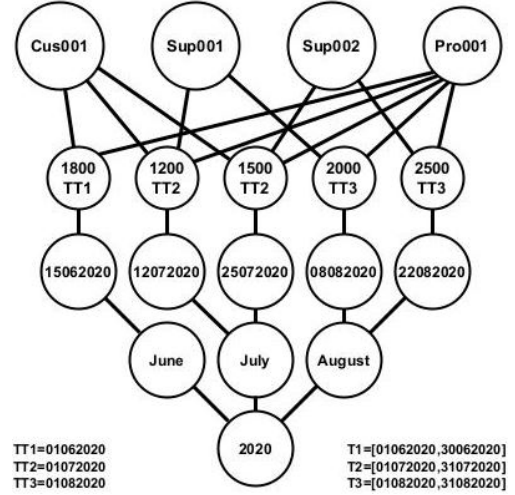
Figure 8: Example of a data instance from a GDW

GAMM offers great flexibility in the elaboration of explicit and implicit queries, which we show in the following examples. The queries are written in the Cypher request language (CRL), which is specific to the Neo4j graph database.

6.0.1 Explicit requests. Explicit requests allow one to extract data from a specific version or set of versions based on the time parameter TT using the clause *where $ST_n <= TT < ET_n$; $T_n = [ST_n, ET_n]$* is the period of validity of version V_n.

Query 1:

MATCH (M:MONTH) <-[:DATE_MONTH] - (:DATE) <- [:SALE_DATE] - (S:SALES) - [SALE_CUST] -> (C:CUSTOMER) WHERE ST_2 <= S.TT < ET_2 RETURN C.CUST_ID, M.MONTH, SUM(S.SALES_AMT) ORDER BY M.MONTH

Table 6: Results of query 1

Cust_id	Month	Sales_amt
Cus001	July	2700

Query 1 shows the sales amounts of customer Cus001 for each month. According to the TT parameter, the result is obtained only from version 2, although the same client has instances in version 1. Note that in the CRL, the clause :
RETURN $value_1, .., value_n$, AGGREGATE_FUNCTION(attribute) allows one to group the aggregation by $value_1 ... value_n$.

6.0.2 Implicit requests. Implicit requests allow one to browse all instances related to the entities specified in the query without version restriction. This feature, which is due to the ability of graph databases to traverse instances through the relationships between entities, offers a major advantage: cross-queries can be formulated using the same simple queries regardless of the number of versions.

Query 2:

MATCH (M:MONTH) <-[:DATE_MONTH] - (:DATE) <- [:SALE_DATE] - (S:SALES) - [SALE_CUST] -> (C:CUSTOMER) RETURN C.CUST_ID, M.MONTH, SUM(S.SALES_AMT) ORDER BY M.MONTH

Table 7: Results of query 2

Cust_id	Month	Sales_amt
Cus001	June	1800
Cus001	July	2700

Query 2 is essentially query 1 without the TT parameter. Therefore, the results are the sales amounts of customer Cus001 for each month according to all versions.

Query 3:

MATCH (M:MONTH) <-[:DATE_MONTH] - (:DATE) <- [:SALE_DATE] - (S:SALES) RETURN M.MONTH, SUM(S.SALES_AMT) ORDER BY M.MONTH

Table 8: Results of query 3

Month	Sales_amt
June	1800
July	2700
August	4500

Query 3 shows the sales amounts per month according to all versions. The query is quite basic, but one would have to consult three fact table versions in the relational approach to get the same result.

7 USE CASES

To illustrate our approach, we have developed a software prototype that manages the meta-model, the graph-based data warehouse and the evolution operators. This prototype has been developed as a 3-tier architecture: the first tier (*front end*) is based on WEB technology and uses the GoJS library to create and manipulate diagrams and interactive graphs on the web. It allows one to view the logical schema of the different versions. The second tier (*back end*), developed in JavaEE, represents the core of the processing, and the third tier is responsible for metadata and data management. A metadata component is implemented using a relational database and a data component is implemented using the NoSQL Neo4j graph database.

For the validation of our approach, we have established two case studies based on the Star Schema Benchmark (SSB): the first study was performed for the functional validation of our approach, and the second study was used to perform runtime tests for a graph DW.

As part of the functional validation study, we chose to perform the instantiation process on Neo4j for several chronological versions using the SSB data; the corresponding schema is represented in Figure 9. Then, direct and cross-queries were performed on the schema versions.
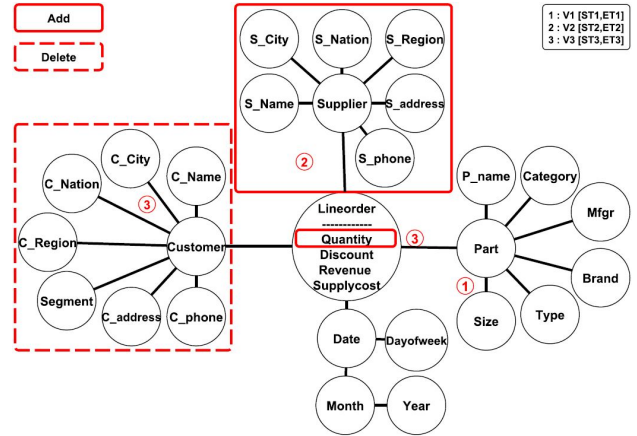

Figure 9: Logical schema for graph SSB versions

As the SSB data spans from 1992 to 1998, the versioning was set as follows:

(1) A first schema for 1992 and 1993 data was composed of a fact Sales analyzed according to the Part and Customer dimensions. The fact Sales is indicated by the measures Sales_Amount, Discount and Supplycost. The dimension Part is described by the attributes *P_Name* and *Size* and the hierarchical levels Category, Mfgr, Brand and Type. The dimension Customer is described by the attributes *C_Name*, *C_Address* and *C_Phone*, and the hierarchical levels C_City, C_Nation, C_Region and Segment. Note that for further study, we ourselves created these levels, which were attributes in the original SSB dataset. This version is valid during the period T_1 =[01/01/2019,31/12/2019].

(2) A second schema version for data from 1994 to 1996 was characterised by the addition of a new *Supplier* dimension described by the attributes *S_Name*, *S_Address* and *S_Phone*, and the hierarchical levels S_City, S_Nation and S_Region. This version is valid during the period T_2 =[01/01/2020,31/12/2020].

(3) A third version of the schema for the 1997 and 1998 data was characterised by the removal of the Customer dimension and the addition of a measure Quantity. This version is valid during the period T_3 = [01/01/2021,31/12/2021].

We were able to apply the 13 queries proposed for the SSB, which we wrote in CRL, either explicitly to query a particular

version or implicitly to query all versions according to the existing schema. Some queries were readjusted to match the time intervals established during the versioning. We provide below, as an example, the first query of the SSB in CRL. Note that the query has been adjusted by changing the attribute D_YEAR to 1997 instead of 1994 in the SSB to match the versioning we established, because the measure Quantity was only created from the third schema version and using data from the years 1997 and 1998.

OPTIONAL MATCH (D:DATED_YEAR:1997) <- [R:ORDER_-
DATE] - (L:LINEORDER) WHERE L.TRANSACTION_TIME >=
DATE(YEAR:2021,MONTH:01) AND 1<= L.LO_DISCOUNT <=3
AND L.LO_QUANTITY < 25 RETURN SUM(L.LO_REVENUE)

The dataset, the commands for creating the graph versions of the DW and all the queries are available at GAMM GITHUB. However, we illustrate the potential of our approach, especially for cross-queries, using the following three queries:

Query 1:

MATCH (D:DATE) <- [:ORDER_DATE] - (L:LINEORDER) -
[R:ORDER_CUSTOMER] -> (C:CUSTOMER) RETURN D.D_-
YEAR, COUNT(DISTINCT(C.CUSTKEY)), SUM(L.LO_REVENUE)
ORDER BY D.D_YEAR

Table 9: Results of query 1

YEAR	C CUSTOMERS	SUM(REVENUE)
1992	79974	13233029288673
1993	79978	13253674263486
1994	79979	13234066608062
1995	79972	13207130575971
1996	79980	13270332561589

Query 2:

MATCH (D:DATE) <- [:ORDER_DATE] - (L:LINEORDER) -
[R:ORDER_SUPPLIER] -> (S:SUPPLIER) RETURN D.D_YEAR,
COUNT(DISTINCT(S.SUPPKEY)),   SUM(L.LO_REVENUE)
ORDER BY D.D_YEAR

Table 10: Results of query 2

| YEAR	C SUPPLIERS	SUM(REVENUE)
1994	8000	13234066608062
1995	8000	13207130575971
1996	8000	13270332561589
1997	8000	13216215992478
1998	8000	7763622653194

Query 3:

MATCH (D:DATE) <- [:ORDER_DATE] - (L:LINEORDER) -
[R:ORDER_PART] -> (P : PART) RETURN D.D_YEAR, COUNT(
DISTINCT(P.PARTKEY)), SUM(L.LO_REVENUE) ORDER BY
D.D_YEAR

Table 11: Results of query 3

YEAR	C PART	SUM(REVENUE)
1992	399956	13233029288673
1993	399956	13253674263486
1994	399953	13234066608062
1995	399959	13207130575971
1996	399965	13270332561589
1997	399965	13216215992478
1998	398036	7763622653194

From the results obtained, we can clearly observe that the versions queried change from one query to another implicitly, even if the version to be queried has not been specified. Indeed, query 1, which displays the sales_amount per year and the number of customers, showed the results between the years 1992 and 1996; this corresponds to versions 1 and 2 of the schema, which included the dimension customer. Additionally, query 2, which displays the sales_amount per year and the number of suppliers, showed the results between the years 1994 and 1998; this corresponds to versions 2 and 3 of the schema, which included the dimension supplier. In the same way, query 3, which displays the sales_amount per year and the number of pieces, showed results for all of the years, as the dimension part is common to all three versions. The alignment of the granularity level was done with the aggregation function SUM.

For the runtime performance study and due to the lack of a baseline for scalable data warehouses, we generated the same SSB schema using our graph approach, spread over the entire validity period of the data, and compared the execution times of the 13 queries on both approaches. The dataset, the commands for creating the full graph DW and all the queries are available at GSSB GITHUB. The experimental process was performed on Windows 10 Professional with an Intel(R) Core(TM) i7-10700 CPU @ 2.90 GHz and 16.0 GB of RAM. Neo4j 4.4.5 was used for the graph approach and Oracle 11g was used for the relational approach. The execution times displayed in Figure 10 represent the average ten executions for each query using the two approaches (the same results were obtained on an Ubuntu platform with the same configuration).
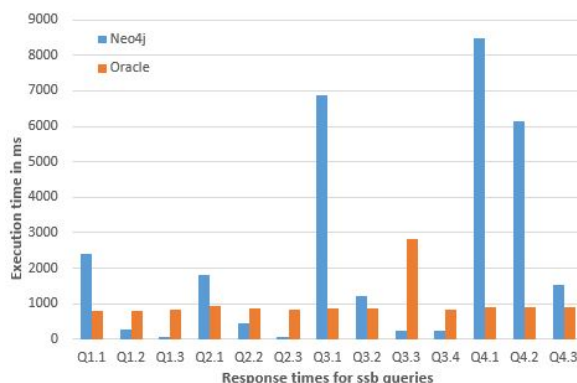
Figure 10: Response times for SSB queries using the graph and relational approaches

We found through the results obtained that the graph approach performed better for the queries Q1.2, Q1.3, Q2.2, Q2.3, Q3.3 and Q3.4 (from 2x to 11x). The two approaches were quite close, with the relational approach being slightly faster, for the queries Q1.1, Q2.1, Q3.2 and Q4.3 (1.5x to 3x), and the relational approach performed better for the queries Q3.1, Q4.1 and Q4.2 (6x to 8x). These differences in the execution times of the response graph approach depend on the filter factors (FFs), the number of dimensions and the number of edges to be covered. Indeed, Neo4j can be very efficient when the FF is low, just as it can become less efficient as the FF and the number of dimensions increase, which implies that many relations need to be browsed to operate an aggregation function; on the other hand, Oracle is quite homogeneous in terms of the execution time of the queries. However, in the context of evolutionary approaches in which the schema is constantly evolving, a performance study must be carried out on the entire storage process, taking into account not only the execution time of the versions but also the evolution procedure, the ETL process after evolution, the cross-requests and other evaluation factors to be determined according to the case being considered.

8 CONCLUSION

To overcome the schema evolution of DWs, we have proposed in this paper a flexible multidimensional model called GAMM, which is based on an extension of the classical multidimensional model, to support chronological evolution at the conceptual level, the evolution of the graph structure on the logical level and the evolution of a NoSQL graph database on the physical level. Thanks to the proposed functions, GAMM allows the evolution of a schema in a data warehouse by creating a new version of the schema at each stage. Each version corresponds to a data instance extracted from a graph data warehouse. A meta-model has been proposed to manage the different schema versions. An example of an instance and direct and cross-queries have been provided. We have validated GAMM with a software prototype with a 3-tier architecture, and we utilised use cases for the functional validation of the approach and to test its runtime performance. In the short term, we will study the temporal evolution of data instances and What-If analysis. We will also carry out a performance study of the entire integration process and further research OLAP queries and graph cubes. In addition, we plan to carry out a study on advanced analyses in graph models in order to utilise online analysis to produce explanatory and predictive models.

REFERENCES

[1] Waqas Ahmed and Esteban Zimányi. 2015. Querying multiversion data warehouses. In *New Trends in Databases and Information Systems – AD-BIS 2015 Short Papers and Workshops, BigDap, DCSA, GID, MEBIS, OAIS, SW4CH, WISARD, Poitiers, France, September 8–11, 2015. Proceedings (Communications in Computer and Information Science)*, Tadeusz Morzy, Patrick Valduriez, and Ladjel Bellatreche (Eds.), Vol. 539. Springer, 346–357. https://doi.org/10.1007/978-3-319-23201-0_36

[2] Waqas Ahmed, Esteban Zimányi, Alejandro A. Vaisman, and Robert Wrembel. 2020. A temporal multidimensional model and OLAP operators. *Int. J. Data Warehous. Min.* 16, 4 (2020), 112–143. https://doi.org/10.4018/IJDWM.2020100107

[3] Waqas Ahmed, Esteban Zimányi, Alejandro A. Vaisman, and Robert Wrembel. 2021. Schema evolution in multiversion data warehouses. *Int. J. Data Warehous. Min.* 17, 4 (2021), 1–28. https://doi.org/10.4018/IJDWM.2021100101

[4] Waqas Ahmed, Esteban Zimányi, and Robert Wrembel. 2014. A logical model for multiversion data warehouses. In *16th International Conference on Data Warehousing and Knowledge Discovery (DaWaK 2014), Germany (Lecture Notes in Computer Science)*, Ladjel Bellatreche and Mukesh K. Mohania (Eds.). Springer. https://doi.org/10.1007/978-3-319-10160-6_3

[5] Jacky Akoka, Isabelle Comyn-Wattiau, Cédric du Mouza, and Nicolas Prat. 2021. Mapping multidimensional schemas to property graph models. In *Advances in Conceptual Modeling – ER 2021 Workshops CoMoNoS, EmpER, CMLS, St. John's, NL, Canada, October 18–21, 2021, Proceedings (Lecture Notes in Computer Science)*, Iris Reinhartz-Berger and Shazia W. Sadiq (Eds.), Vol. 13012. Springer, 3–14. https://doi.org/10.1007/978-3-030-88358-4_1

[6] Bartosz Bebel, Johann Eder, Christian Koncilia, Tadeusz Morzy, and Robert Wrembel. 2004. Creation and management of versions in multiversion data warehouse. In *Proceedings of the 2004 Symposium on Applied Computing (SAC), Cyprus*, Hisham Haddad, Andrea Omicini, Roger L. Wainwright, and Lorie M. Liebrock (Eds.). ACM. https://doi.org/10.1145/967900.968049

[7] Zohra Bellahsene. 2002. Schema evolution in data warehouses. *Knowledge and Information Systems* 4, 3 (2002).

[8] Edgard Benitez-Guerrero, Christine Collet, and Michel Adiba. 2004. The WHES approach to data warehouse evolution. *e-Gnosis Num.002* (2004).

[9] Markus Blaschka, Carsten Sapia, and Gabriele Höfling. 1999. On schema evolution in multidimensional databases. In *International Conference on Data Warehousing and Knowledge Discovery*. Springer, 153–164.

[10] Rasa Bliujute, Simonas Saltenis, Giedrius Slivinskas, and Christian S Jensen. 1998. Systematic change management in dimensional data warehousing. In *Proceedings of the Third International Baltic Workshop on Data Bases and Information Systems, Riga, Latvia*. Citeseer.

[11] Cécile Favre, Fadila Bentayeb, and Omar Boussaid. 2007. Evolution of data warehouses' optimization: A workload perspective. In *International Conference on Data Warehousing and Knowledge Discovery*. Springer.

[12] Georgia Garani, George K. Adam, and Dimitrios Ventzas. 2016. Temporal data warehouse logical modelling. *Int. J. Data Min. Model. Manag.* 8, 2 (2016), 144–159. https://doi.org/10.1504/IJDMMM.2016.077156

[13] Matteo Golfarelli, Dario Maio, and Stefano Rizzi. 1998. Conceptual design of data warehouses from E/R schema. In *Thirty-First Annual Hawaii International Conference on System Sciences, Kohala Coast, Hawaii, USA, January 6–9, 1998*. IEEE Computer Society, 334–343. https://doi.org/10.1109/HICSS.1998.649228

[14] Matteo Golfarelli and Stefano Rizzi. 2011. Temporal data warehousing: Approaches and techniques. In *Integrations of Data Warehousing, Data Mining and Database Technologies – Innovative Approaches*, David Taniar and Li Chen (Eds.). Information Science Reference, 1–18. https://doi.org/10.4018/978-1-60960-537-7.ch001

[15] Matteo Golfarelli and Stefano Rizzi. 2018. From star schemas to big data: 20+ years of data warehouse research. In *A Comprehensive Guide Through the Italian Database Research Over the Last 25 Years*, Sergio Flesca, Sergio Greco, Elio Masciari, and Domenico Saccà (Eds.). Springer International Publishing. https://doi.org/10.1007/978-3-319-61893-7_6

[16] Carlos A Hurtado, Alberto O Mendelzon, and Alejandro A Vaisman. 1999. Maintaining data cubes under dimension updates. In *Proceedings of the 15th International Conference on Data Engineering (Cat. No. 99CB36337)*. IEEE, 346–355.

[17] William H. Inmon. 1992. *Building the Data Warehouse*. John Wiley & Sons, Inc., USA.

[18] Ralph Kimball. 1996. *The Data Warehouse Toolkit: Practical Techniques for Building Dimensional Data Warehouses*. John Wiley & Sons, Inc., USA.

[19] Jose-Norberto Mazón, Jesús Pardillo, and Juan Trujillo. 2006. Applying transformations to model driven data warehouses. In *Data Warehousing and Knowledge Discovery*, A. Min Tjoa and Juan Trujillo (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 13–22.

[20] Alberto Mendelzon and Alejandro Vaisman. 2000. Temporal queries in OLAP. *Proceedings of the 26th International Conference on Very Large Data Bases (VLDB'00)*, 242–253.

[21] Tadeusz Morzy and Robert Wrembel. 2003. Modeling a multiversion data warehouse: A formal approach. In *Proceedings of the 5th International Conference on Enterprise Information Systems (ICEIS 2003), Angers, France, April 22–26, 2003*. 120–127.

[22] Thanapol Phungtua-Eng and Suphamit Chittayasothorn. 2019. Slowly changing dimension handling in data warehouses using temporal database features. In *11th Asian Conference on Intelligent Information and Database Systems (ACIIDS 2019), Indonesia*, Ngoc Thanh Nguyen, Ford Lumban Gaol, Tzung-Pei Hong, and Bogdan Trawinski (Eds.). Springer. https://doi.org/10.1007/978-3-030-14799-0_58

[23] Franck Ravat, Olivier Teste, and Gilles Zurfluh. 2006. A multiversion-based multidimensional model. In *Data Warehousing and Knowledge Discovery, 8th International Conference, DaWaK 2006, Krakow, Poland, September 4–8, 2006, Proceedings (Lecture Notes in Computer Science)*, A Min Tjoa and Juan Trujillo (Eds.), Vol. 4081. Springer, 65–74. https://doi.org/10.1007/11823728_7

[24] Stefano Rizzi and Matteo Golfarelli. 2007. X-Time: Schema versioning and cross-version querying in data warehouses. In *Proceedings of the 23rd International Conference on Data Engineering (ICDE 2007), Istanbul, Turkey, April 15–20, 2007*, Rada Chirkova, Asuman Dogac, M. Tamer Özsu, and Timos K. Sellis (Eds.). IEEE Computer Society, 1471–1472. https://doi.org/10.1109/ICDE.2007.369038

[25] Kanika Talwar and Anjana Gosain. 2012. Hierarchy classification for data warehouse: A survey. *Procedia Technology* 6 (2012), 460–468.

[26] Robert Wrembel and Tadeusz Morzy. 2006. Managing and querying versions of multiversion data warehouse. In *Advances in Database Technology (EDBT 2006)*, Vol. 3896. Springer. https://doi.org/10.1007/11687238_73