International Journal of Data Warehousing and Mining
Volume 16 • Issue 4 • October-December 2020

A Temporal Multidimensional Model and OLAP Operators

Waqas Ahmed, CODE WIT, Université Libre de Bruxelles, Belgium

Esteban Zimányi, CODE WIT, Université Libre de Bruxelles, Brussels, Belgium
https://orcid.org/0000-0003-1843-5099

Alejandro Ariel Vaisman, Department of Information Engineering, Instituto Tecnológico de Buenos Aires, Argentina
https://orcid.org/0000-0002-3945-4187

Robert Wrembel, Faculty of Computing and Telecommunications, Poznan University of Technology, Poznan, Poland
https://orcid.org/0000-0001-6037-5718

ABSTRACT

Usually, data in data warehouses (DWs) are stored using the notion of the multidimensional (MD) model. Often, DWs change in content and structure due to several reasons, like, for instance, changes in a business scenario or technology. For accurate decision-making, a DW model must allow storing and analyzing time-varying data. This paper addresses the problem of keeping track of the history of the data in a DW. For this, first, a formalization of the traditional MD model is proposed and then extended as a generalized temporal MD model. The model comes equipped with a collection of typical online analytical processing (OLAP) operations with temporal semantics, which is formalized for the four classic operations, namely roll-up, dice, project, and drill-across. Finally, the mapping from the generalized temporal model into a relational schema is presented together with an implementation of the temporal OLAP operations in standard SQL.

KEYWORDS

Data Warehouse, OLAP, OLAP Operations, Temporal Data Warehouse, Temporal Databases, Temporal Multidimensional Model

INTRODUCTION

Typically, the Business Intelligence (BI) process is supported by a centralized repository called a Data Warehouse (DW), which integrates data coming from different sources (Vaisman & Zimányi, 2014). A DW is typically represented using the multidimensional (MD) model, where data are represented as a collection of facts, measures, dimensions, levels, and hierarchies. These notions are informally introduced next in a streamlined fashion, through a running example which represents sales in a fictitious retail company. The DW schema is depicted in Figure 1, using the MultiDim (Vaisman & Zimányi, 2014) notation.

DOI: 10.4018/IJDWM.2020100107

Copyright © 2020, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

Multidimensional Data Model

In a DW, data are perceived in an *n*-dimensional space. In this space, a *fact* is a subject of interest, e.g., a business event. Each observation in a fact is called a *fact member*. A fact is quantified by one or more *measures*, which are typically numerical quantities. *Dimensions* provide context to facts. For instance, sales events can be perceived in a three-dimensional Sales fact, each one representing information about when and where a specific product was sold at a certain price. In this case, the Product, the Time of the sale, and the City where the product was sold represent the dimensions, and the Amount sold is a measure. *Levels* provide dimension values, which in turn are described by *attributes*. For example, a level Day provides all possible values for dimension Date. Instances of a level are called *level members* or *members*. A DW may contain multiple levels, and various facts may share these levels. A level is connected to a fact if it provides values for any dimension in that fact. Such a level determines the *granularity* of a dimension, which determines the level of detail at which measures are recorded. For example, in the fact Sales in Figure 1, values for the Date dimension come from level Day, meaning that the granularity of the dimension is at the Day level. A level may provide values for more than one dimension in a fact, and such dimensions are called *role-playing dimensions*.

The relationship between levels is called an *aggregation relationship*, which associates the members of a parent level and a child level, where the latter is the level defined at a finer granularity in the relationship. An aggregation relationship can be named. Further, the *cardinality* of a relationship between dimension levels indicates how level members relate to each other. Analogously, the cardinality of a relationship between a fact and a level indicates how fact members are associated with level members. As usual in conceptual modeling, cardinalities can be one-to-one (1-1), many-to-one (m-1), and many-to-many (m-m). Furthermore, the cardinality can be optional (denoted 0) for any of the two participating entities of a relationship, meaning that the participation of members of an entity is not mandatory in the relationship. For example, the cardinality between level Product and fact Sales is *many-to-one*, meaning that a sales transaction contains one product, and a product member may appear in multiple sales transactions. Note that aggregation relationships can be recursive, meaning that members of a level are the parent members of the same level. For example, the relationship Supervisor between members of level Employee is a recursive relationship, since both supervisors and their subordinates are members of level Employee.

A *hierarchy* is a collection of logically related aggregation relationships and allows aggregating measure values to a coarser level of detail from values at a finer level of detail. In a *strict hierarchy*, all aggregation relationships are many-to-one, whereas, in a *non-strict hierarchy*, at least one aggregation relationship has a many-to-many cardinality.

The notion of *summarizability* refers to the correctness of the aggregation of measure values from a lower level to a higher level of granularity. Ensuring summarizability is an essential condition of any MD model. Summarizability conditions where defined by (Lenz & Shoshani, 1997), which state that members in a dimension level *L* must define a partition of the members in the level immediately below *L*. It follows that non-strict hierarchies are non-summarizable since they contain many-to-many relationships.

Temporal Multidimensional Data Model

The time dimension in a traditional DW is used to associate the notion of time with measure values. Having a time instant associated with each fact member allows for aggregating and analyzing the measure values at various points in time, which is very helpful in the decision-making process. However, a traditional DW does not have any mechanism to track the changes over its dimension members, caused by routine business processes or the correction of existing data. The most frequent use-case of temporal data comes from a sales business, where prices of products may change every day. Less frequent changes apply to the assignment of products to categories, i.e., changes in the structure of a dimension instance.

Capturing content changes is essential for reconstructing the state of the business world at specific points in time, keeping an audit trail of changes, and simulating hypothetical ("what-if") business scenarios. A user may decide to add the notion of time to the (1) existence of dimension members, (2) properties of dimension members, and (3) relationships between dimension members.

Slowly changing dimensions (SCDs) (Kimball & Ross, 2013) were presented as a solution for the problem of maintaining the history of changes in the level attributes of relational DWs. However, these are ad-hoc solutions with their shortcomings (see Section Related Work) and often require schema modifications. Moreover, SCDs do not provide aggregation semantics for time-varying data, thus, obtaining correct aggregation results remains the user's responsibility.

To keep the full history of the DW model presented in the previous section, it can be extended with orthogonal time dimensions, along the lines of classic temporal databases (Tansel, et al., 1993). For example, the former can be achieved through the addition of temporal features, namely temporal attributes, temporal levels, and temporal aggregation relationships, that allow keeping the history of dimension members in a MD model.

A *temporal attribute* keeps track of the changes in its value and the time when these changes occurred. A *temporal level* is a level for which the application requires storing a time frame associated with its members. A *temporal aggregation relationship* keeps a record of the history of parent-child relationships in dimension hierarchies. A comprehensive literature review on the subject is given in the related work section.

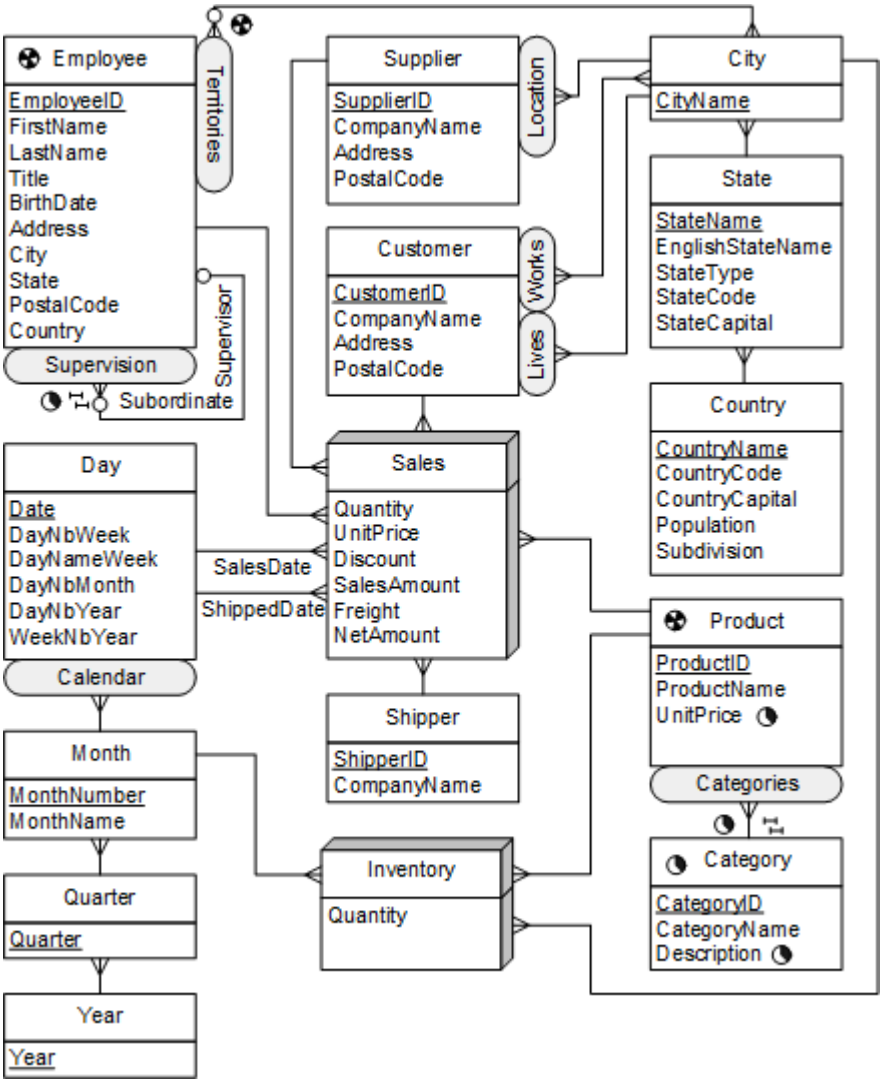
Temporal aspects can be associated with real-world objects and events, using the so-called temporality types (Snodgrass & Ahn, 1985). *Valid time* (VT), typically supplied by the user, refers to the time during which a record is considered valid in the real world, and it can be used to represent the state of an object at different points in time. *Transaction time* (TT), usually maintained by a system, represents the time during which a record is stored in the system. It represents the state of the database at different points in time and is mainly used for roll-back or auditing purposes. Valid and transaction times are orthogonal and can be used either individually or combined to define *bitemporal time* (BT). Another temporal aspect, called *lifespan* (LS), represents the time during which an object exists. The lifespan of an object is the valid time of a fact telling that the object exists. Finally, a user may also include any other temporal aspects to the data referred to as *user-defined* temporality types.

Figure 1 shows the schema of the temporal MD data warehouse of the retail company using the temporal MultiDim notation (Malinowski & Zimányi, 2008). The temporal Multidim model allows various temporal data types and represents them graphically with pictograms. An instant (⏲) represents a single point in time, an interval (◔) denotes a set of successive instants between two instants, an instant set (✇) is used for representing the same event occurring at different time instants, and an interval set (☢) can be used to describe the same state in different intervals of time. The symbols (☢) next to levels Employee and Product, and (◔) next to level Category, indicate that those are temporal levels and thus, the lifespan of their members must be maintained. The symbol (◔) next to attributes UnitPrice and Description in levels Product and Category, respectively, indicates that those are temporal attributes, and thus, the history of changes in their values must be maintained. The same symbol in the relationship between Product and Category, and the recursive relationship between Subordinate and Supervisor, and the symbol (☢) next to the relationship between Employee and City, tell that the history of the assignment of child members to their parent member should be maintained. The symbol (⊆) next to a relationship means that the validity period of the association must be contained within the lifespan of the child and parent members. For example, when assigning a product to a category, the category must exist throughout the existence of the product.

Research Problems

It follows from the discussion above that a MD model should include the following features:

Figure 1. An example data warehouse to analyze the sales and inventory of a retail company



1. **Expressiveness:** A MD model must allow representing measures at various granularities, role-playing dimensions, reusability of level instance across multiple hierarchies, multiple aggregation pathways for the same dimension, non-strict hierarchies, correct aggregation of measures while using a non-strict hierarchy, and optional cardinality between a child-parent or fact-level relationship;
2. **Temporal Features:** A MD model should be able to represent time-varying data, allowing temporal levels, temporal attributes, temporal child-parent relationships, and temporal hierarchies. It must also include data consistency constraints such as synchronization constraints between two temporal levels and constraints between a fact and a temporal level;
3. **Modular Data Model:** Normally, in real-world applications, it is not necessary to keep the history of all the elements in a MD model. Therefore, a modular data model must allow both nontemporal and temporal features to be included independently of each other (Hümmer, Lehner, Bauer, & Schlesinger, 2002);

4. **OLAP Operators:** A MD model must include a set of operations that could be combined to analyze both nontemporal and temporal data. The purpose of these operations is to serve as a high-level analysis language and transparently exploit the model constructs such as hierarchies to produce the correct results;
5. **Easy Implementation:** The data structure, data consistency constraints, and the OLAP operator constituting a MD model should be implementable using the existing technologies such as relational databases.

Contributions and Paper Organization

This paper proposes a semantically rich MD model that can represent both, nontemporal and temporal data, and which can be implemented in standard SQL. The model also includes a set of OLAP operations, each of which takes as an input a fact, and adds, as a result, another fact into the model. Thus, the operations can be used in a combined fashion to answer analytical queries. Furthermore, data constraints are provided to preserve data consistency.

The paper begins with formalizing the notions of a MD data model and provides the semantics of the OLAP operators over the model. Then, a generalized MD model is presented, supporting both nontemporal and temporal features independently. Further, the semantics of temporal OLAP operators are also given. After this, the paper shows how the data structures and the data consistency constraints of the generalized temporal model can be mapped into a relational schema. Also, with the help of example queries, it is shown how the OLAP operations of the model can be implemented in the standard SQL. A study of the related work follows, and finally, the conclusions and future research directions are presented.

A FORMAL MULTIDIMENSIONAL MODEL

This section formally defines the data structure of the multidimensional DW model.

In what follows, the domain of a type b is denoted *dom(b)* and it is extended with a special value $\perp$. Furthermore, *Card* is a set of cardinality constraints where each element follows the usual semantics as in conceptual modelling. Before defining a MD model, it is necessary to formalize the concepts of level, aggregation relationship, hierarchy, and fact.

Definition 1 (Level Schema). The schema of a level L is defined by $L\left(A_k : O_k, A+1 : O_1, \ldots, A_n : O_n\right)$, where L is the level name, A_k is the name of the key attribute, and it has a base type O_k, and each $A_i, i = 1, \ldots, n,$ is an attribute name, and it has a base type O_i. Also, all attribute names are unique in L.

Example 1. The schema of level Category $\in L$ is Category(CategoryID:Integer, CategoryName:

String, Description: String), and CategoryID is the name of the key attribute

Definition 2 (Level Instance). A level instance $L \in \mathcal{L}$ is a set of level members defined by $L \subset k \times a_1 \times \ldots \times a_n \mid k \in dom(O_k) \wedge a_i \in dom(O_i), i = 1,\ldots,n \cup \overline{l}$ where $\overline{l}$ is the default member of *L*. Also, L satisfies the key constraint, that is, $\forall\ l_1, l_2 \in L,\ l_1 \neq l_2 \Rightarrow l_1.k \neq l_2.k$. The set of the key values of a level instance is denoted by $L_{\mathrm{s}} = \{l.k \mid l \in L\}$.

Definition 3 (Aggregation Relationship Schema). An aggregation relationship R is defined by $R\left(L_c, L_p, \mathrm{card}\right)$, where R is the aggregation relationship name, L_c, L_p are the level names in

$\mathcal{L}$, and *card* ∈ *Card* specifies the cardinality of the relationship between the child level L_c and the parent level L_p.

Example 3. The schema of all aggregation relationships between levels Customer, City, State, and Country is Customer_City_Lives(Customer, City, m-1), Customer_City_Works(Customer, City, m-m), City_State(City, State, m-1), and State_Country(State, Country, m-1).

Depending on the cardinality, a child member may be associated with one or multiple parent members. In case of an optional cardinality on the parent side, the orphan child members are associated with the default member of the parent level. Also, the default member of a child level is related to the default member of the parent level. In this way, each aggregation relationship is always complete.

Definition 4 (Aggregation Relationship Instance). An aggregation relationship instance $R \in \mathcal{R}$ of schema $R\left(L_c, L_p, \text{card}\right)$, is a relation defined by $R = \hat{R} \cup \left(\overline{l}_c.k, \overline{l}_p.k\right)$, where $\hat{R} \subset L_{c\kappa} \times L_{p\kappa}$, and $\overline{l}_c \in L_c$ and $\overline{l}_p \in L_p$ are the default members of levels L_c and L_p, respectively. If $l_c \in L_{c\kappa}, l_p \in l_p \in L_{p\kappa}$ and $\left(l_c, l_p\right) \notin R$, then $\left(l_c, \overline{l}_p.k\right) \in R$, where $\overline{l}_p$ is the default member of level L_p. Further, R satisfies *card*.

Example 4. An aggregation relationship instance for Customer_City_Works is $\text{Customer_City_Works} = \left(\text{c}_1, \text{Brussels}\right), \left(\text{c}_2, \text{Antwerp}\right), \ldots, \left(\text{c}_\text{n}, \text{Brussels}\right)$.

Definition 5 (Roll-Up Hierarchy Schema). The schema of a roll-up hierarchy is defined by $H\left(R_1^{'}, \ldots, R_n^{'}\right)$, where each $\text{R}_\text{i}^{'}, i = 1, \ldots, n,$ is an aggregation relationship name, and $R_i^{'}.L_p = R_{i+1}^{'}.L_c, i = 1, \ldots, n-1$. This constraint ensures that the parent level of a relationship $\text{R}_\text{i}^{'}$ is the same as the child level of the next relationship $\text{R}_{\text{i+1}}^{'}$, except for the last level of the hierarchy. The *base level* of hierarchy H is denoted $\text{L}_\text{b} = \text{R}_1^{'} \cdot \text{L}_\text{c}$. Also, $\text{L}_\text{t} = \text{R}_\text{n}^{'} \cdot \text{L}_\text{p}$ is called the *top level* of H. There is also a function $levles\left(H\right)$, which returns the set of all level names that form the parent-child relationships in H.

Example 5. The $\text{structure}\,\text{Works}\left(\text{Customer_City_Works, City_State, State_Country}\right)$ is the schema of the Works hierarchy in Figure 1. Also, $L_b = \text{Customer}$ is the base level of the hierarchy, works, $L_t = \text{Country}$ is the top level, and $\text{levels}\left(\text{Works}\right) = \left\{\text{Customer, City, State, Country}\right\}$ is the set of levels that form the hierarchy.

Depending on the cardinality of relationships in a hierarchy, a base level member can be transitively associated with one or multiple top-level members. The number of parents a base level member is connected to is called its distribution factor.

Definition 6 (Roll-Up Hierarchy Instance). The instance of hierarchy H, denoted H, is given by $H = R_1^{'} \cup \ldots R_n^{'}; H*$ denotes the transitive closure of H meaning that if relations $\left(l, l'\right), \left(l', l''\right) \in H$ then relation $\left(l, l''\right) \in H*$. Further, $df_{l_c} = \left\{l_t \middle| l_c \in H.L_b \wedge l_t \in H.L_t \wedge \left(l_c.k, l_p.k\right) \in H*\right\}$ is the distribution factor of a base level member l_c of hierarchy H.

Example 6. An instance of the Works hierarchy is:

$$\text{Works} = \text{Customer_City_Works} \cup \text{City_State} \cup \text{State_Country}.$$

Definition 7 (Fact Schema). A fact F is defined by $F\left(K : O_k, \mathcal{D}, \mathcal{M}\right)$, where F is the fact name, K is the name of the key attribute, and it has a base type O_k, the tuple $\mathcal{D}\left(D_1, \ldots, D_m\right)$ defines the dimensions of F, and each $D_i \in \mathcal{D}, i = 1, \ldots, m,$ is a pair $D_i\left(L_i, card_i\right)$ such that D_i is a dimension name, L_i is a level name in $\mathcal{L}$, and $card_i \in \left\{01-01, 01-1, 1-01, 1-1, 0m-01, 0m-1, m-01, m-1\right\}$ specifies the cardinality of the relationship between fact F and level L_i. The tuple $\mathcal{M}\left(M_1 : O_1, \ldots, M_n : O_n\right)$ defines the measures of F, and each $M_i \in \mathcal{M}, i = 1, \ldots, n,$ is a measure name that has a base type O_i. The key name K, a dimension name D, and a measure name M are unique in F.

Example 7. The schema of fact Sales is Sales$\left(K, \mathcal{D}, \mathcal{M}\right)$, where $K =$ SalesID:Integer. The dimensions in the fact are $\mathcal{D}$ (Customer, Employee, SalesDate, ShippedDate, Product, Supplier, Shipper), and the measures are $\mathcal{M}$ (Quantity:Integer, Discount:Decimal, SalesAmount:Integer, NetAmount:Decimal, Freight:Decimal). The schemas of the dimensions in Sales are Customer(Customer, m-1),Employee(Employee, m-1), SalesDate(Day, m-1), ShippedDate(Day, m-1), Product(Product, m-1), Supplier(Supplier, m-1), and Shipper(Shipper, m-1).

Definition 8 (Fact Instance). A fact instance $F \in \mathcal{F}$ is a set of fact members defined by $F = \left\{ f \middle| \hat{F}\left(f\right) \wedge \text{if } f.L_i = \perp \text{ then } f.L_i = \overline{l}_i.k \right\}$, where:

$$\hat{F} \subset \left\{ \begin{array}{l} k \times l_1 \times \ldots \times l_m \times m_1 \times \ldots \times m_n \middle| k \in dom\left(O_k\right), \\ l_i \in L_{i^\circ}, m_j \in dom\left(O_j\right) \wedge i = 1, \ldots, m \wedge j = 1, \ldots, n \end{array} \right\}$$

and $\overline{l}_i$ is the default member of the level L_i. The tuple $e\left(f.k, f.l_1, \ldots, f.l_m\right), f \in F$, is an *m-dimensional* cell which is a placeholder for the values of *n* measures. Each $e.l_i, i = 1, \ldots, m$, is called a coordinate value for the dimension D_i of the cell. The fact instance F satisfies the key constraint and the cardinality constraints similarly as defined above for the level instances and the aggregation relationship instances.

Example 8. A fact instance Sales is shown in Figure 3. (for clarity, in table format). For brevity, only dimensions Product and Date and measure Quantity is shown.

Definition 9 (Multidimensional Schema). A multidimensional schema is a structure $\mathcal{S}\left(\mathcal{L}, \mathcal{R}, \mathcal{H}, \mathcal{F}\right)$, where S is the schema name, $\mathcal{L} = \left\{L_1, \ldots, L_n\right\}$ is a set of levels, and all levels in $\mathcal{L}$ have a unique name and have the same domain O_k for the key attribute A_k, $\mathcal{R} = \left\{R_1, \ldots, R_n\right\}$ is a set of aggregation relationships between levels in $\mathcal{L}$ and all aggregation relationship names in $\mathcal{R}$ are unique, $\mathcal{H} = \left\{H_1, \ldots, H_n\right\}$ is a set of hierarchies, and all hierarchy names in $\mathcal{H}$ are unique, and $\mathcal{F} = \left\{F_1, \ldots, F_n\right\}$ is a set of facts, and all fact names in $\mathcal{F}$ are unique.

Definition 10 (Multidimensional Instance). Given a schema $\mathcal{S}\left(\mathcal{L},\mathcal{R},\mathcal{H},\mathcal{F}\right)$ as in Definition 9, a multidimensional instance $\mathcal{I}$of $\mathcal{S}$ is defined as $\mathcal{I}\left(\mathcal{L},\mathcal{R},\mathcal{H},\mathcal{F}\right)$, where $\mathcal{L}=\left\{L_1,\ldots,L_n\right\}$ is a set of level instances, $\mathcal{R}=\left\{R_1,\ldots,R_n\right\}$ is a set of aggregation relationship instances, $\mathcal{H}=\left\{H_1,\ldots,H_n\right\}$ is a set of hierarchy instances, and $\mathcal{F}=\left\{F_1,\ldots,F_n\right\}$ is a set of fact instances.

OLAP Operators

This section introduces a formal syntax and semantics for the four classic OLAP operations, namely *roll-up*, *dice*, *project*, and *drill-across*, associated with the model explained above. Each of the operations below takes as an input a multidimensional data structure MD_s, adds a new fact F' to the fact schema $MD_s.\mathcal{S}$ and a fact instance F' to the fact instance of $MD_s.\mathcal{I}$, and returns the result as a new MD data warehouse MD_o. In this way, the output of one operator can be the input of another operator. Thus, these operators, combined, define a closed OLAP algebra. For notation, in what follows, $F.S$ is used to refer to the component $\mathcal{S}$ of the element F, e.g., the dimension set $\mathcal{D}$ in the fact schema F. Using this notation, the schema and instance of the resultant MD_o are defined as $MD_o.\mathcal{S}=MD_s.\mathcal{S}\cup F'$, and $MD_o.\mathcal{I}=MD_s.\mathcal{I}\cup F'$. The schema and the instance of F' depends on the individual operator. Next, each operator is formally defined.

Roll-Up

The Roll-up operation summarizes a measure from a lower level to a higher level of a hierarchy, using an aggregate function. The aggregation operation is based on the relational calculus with aggregation, defined by Klug (Klug, 1982). The functions in such calculus take as input a set of tuples (instead of a set of values) and the position of an attribute from the tuples to be aggregated. For example, the function $sum_2\left(\text{SalesKey, Quantity}\right)$ returns the sum of the second attribute of input tuples, i.e., Quantity. For clarity, the model introduced here uses the traditional SQL aggregate functions *sum*, *average*, *min*, *max*, *count*, and *count-distinct*, although it does not prevent other ones from being defined.

Definition 11 (Roll-Up). The Roll-up operator is defined as $\left(MD_s,F,D_j,H,\mathcal{M}'\right)\rightarrow MD_o$, where MD_s is a MD structure; F is a fact name in MD_s; D_j is a dimension in $F.\mathcal{D}$; H is a roll-up hierarchy such that $H.L_b=D_j.L_j$; $\mathcal{M}'=\left\{\left(M_1,\alpha_1\right),\ldots,\left(M_{n'},\alpha_{n'}\right)\right\}$ where $M_i, i=1,\ldots,n'$, is a measure name in F, and $\alpha_i\in\left\{sum,average,\min,\max,count,count-distinct\right\}$ is an aggregate function. Also, if $F.\mathcal{M}\supset\mathcal{M}'$, then the measures in $F\cdot\mathcal{M}/\mathcal{M}'$ are removed. Finally, MD_o is the resulting MD structure.

Given the input fact F with the schema $F\left(K:O_k,\mathcal{D},\mathcal{M}\right)$ where $D_j\in\mathcal{D}=\left(L_j,card_j\right)$, and its instance F containing members of the form $\left(d_1,\ldots,d_j,\ldots,d_m,m_1,\ldots,m_n\right)$ where $d_j\in L_{j\kappa}$, the schema and instance of F', the fact that is added to the MD database is defined as follows:

$$F'\left(K:O_k,\mathcal{D}',\mathcal{M}\right)$$

where $\mathcal{D}' = \left(D_1,\ldots,D_j^{'},\ldots,D_m\right)$ and $D_j^{'} = \left(H.L_t,1\text{-}m\right)$:

$$F' = \{\left(k',l_1.k,\ldots,l_t.k,\ldots,l_m.k,m_1^{'},\ldots,m_{n'}^{'}\right)\Big|k' \in dom\left(O_k\right) \wedge L_i\left(l_i\right)$$
$$for\, i = 1\ldots m,\ i \neq j \wedge H.L_t\left(l_t\right) \wedge m_x^{'} = \alpha_x\left(\{\left(f.k,f.m_x\right)F\left(f\right) \wedge f.l_i,= l_i.k\}\right)$$

for $i = 1,\ldots,n, i \neq j \wedge \left(f.l_j,l_t.k\right) \in H^*\})\ /df_{l_j}$ for $x = 1,\ldots,n'\}$.

The schema of F' is similar to the schema of F, except the level L_j in dimension D_j is replaced with the top-level $H.L_t$ of the hierarchy H. In the instance F', for each tuple, k' is a new key of the fact, and $l_t.k$ is a key of a member from the level $H.L_t$; df_{l_j} is the distribution factor of l_j as defined in Definition 6, and $m_x^{'}$ is the aggregated value of the measure m_x, computed using the aggregate function α_x. Intuitively, in F' starting from $L_j \in H$, the values of the dimension D_j that belong to the same parent are grouped at level $H.L_t$, and for each group, the input measures $m_1,\ldots,m_{n'}$ are aggregated using their respective aggregate functions.

The instance F' is generated as follows. For each new key k', the new dimension members for each tuple are fixed. These members are the same as in the input fact tuple, except for the key $l_j.k$ of the initial level of the roll-up dimension D_j, which is replaced with its parent's key, that is, $l_t.k$ from $H.L_t$. Next, in the inner expression of the definition of F', a subset of F containing tuples corresponding to the dimension members fixed in the outer query is obtained. Here the existence of $\left(l_j.k,l_t.k\right) \in H\,*$ ensures that only tuples in which children of l_t are present are selected from F. After that, the aggregate function α_x is applied over each measure m_x. Finally, in case a child member is connected to more than one parent, the aggregated values of measures should be uniformly distributed among all parents, and that is why the aggregated measure value is divided by df_{l_j} to obtain the distributed share of aggregated measure value for l_t. Note that if a measure is aggregated using the *min* or *max* function, then $df_{l_j} = 1$. Since the *average* is a non-distributive function, i.e., it cannot be computed from a sub-aggregate, dividing the total average value among all parents would be incorrect. For this reason, the average is calculated from the distributive share of *sum* and *count*.

As an example, consider the MD database SalesDW, and the query: *Total quantity sold for each product category.* This query requires a Roll-up operation over dimension Product up to level Category along hierarchy Categories = $\left\{\text{Product_Category}\right\}$. The syntax of such operation is as follows:

- **Roll-Up:** (SalesDW, Sales, Product, Categories, $\left\{\left(\text{Quantity:sum}\right)\right\}$) $\rightarrow$ $\text{Sales}^{'}$.

The operation adds a new fact $\text{Sales}^{'}$ to SalesDW. In $\text{Sales}^{'}$, products are grouped into their respective categories. For each category, the Quantity of all products belonging to this category where all other dimension values are the same is aggregated using the function *sum*. For the product to category assignment shown in Figure 2, and the initial fact Sales in Figure 3, the result is shown in Figure 4. Note that in $\text{Sales}^{'}$, keys from level Product are replaced with keys from level Category. Also note that in the Categories hierarchy, the cardinality of the aggregation relationship Product $\rightarrow$ Category is 1-m. However, in the Works hierarchy in Figure 1, the cardinality of the aggregation relationship Customer $\rightarrow$ City is m-m, meaning that a customer may be assigned to more than one

Figure 2. Instance of hierarchy categories

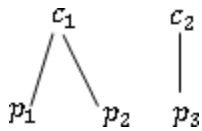


Figure 3. Fact sales before roll-up

SalesKey	Product	Date	Quantity
1	p_1	t_7	5
2	p_1	t_3	10
3	p_1	t_1	15
4	p_2	t_1	5
5	p_3	t_1	8

Figure 4. Fact sales after roll-up

SalesKey	Category	Date	Quantity
1	c_1	t_7	5
2	c_1	t_3	10
3	c_1	t_1	20
4	c_2	t_1	8

city. Therefore, to obtain the total sales made to a customer c_1 who is working in both Mechelen and Antwerp, the total quantity sold to the customer c_1 must be distributed uniformly between the two cities. This is done using the distribution factor df in the above definition.

Dice

The Dice operation selects the fact members from an input fact that either satisfies a Boolean condition on measure values or level attribute values. There are two kinds of conditions, defined next.

Definition 12 (Atomic Conditions on Measure Values). Given a fact F, and $M\theta M'$ and $M\theta c$, atomic conditions on F, where M and M' are measure names in F; $\theta \in \{=,\neq,\langle,\leq,\rangle,\geq\}$ is a binary operator; $c \in dom(b)$ is a value constant where b is a base type. Semantically, for a fact member $f \in F$, $M\theta M'$ is true iff $f \cdot M\theta f \cdot M'$ holds, and $M\theta c$ is true iff $f \cdot M\theta c$ holds.

Conditions can be defined on dimension levels associated with a fact, as follows.

Definition 13 (Atomic Condition on Level Attribute Values). Given a fact F, and $D_i.H.L.A\theta D_j.H'.L'.A'$ or $D_i.H.L.A\theta c$, atomic conditions on attribute values for F, where D_i and D_j are dimensions in $F.\mathcal{D}$; H and H' are roll-up hierarchies such that $H.L_b = D_i.L_i$ and $H'.L_b = D_j.L_j$; A and A' are attributes of levels $L \in levels(H)$ and $L' \in levels(H')$ respectively; and $c \in dom(b)$, where b is a base type and a value constant.

For an instance F of fact F containing fact members of the form $\left(l_1,\ldots,l_i,\ldots,l_j,\ldots l_m,m_1,\ldots,m_n\right)$ where $l_i \in D_i.L_{i\kappa}$ and $l_j \in D_j.L_{j\kappa}$, the semantics of $D_i.H.L.A\theta D_j.H'.L.A'$ and $D_i.H.L.A\theta c$ is defined as follows:

- $D_i.H.L.A\theta D_j.H'.L.A'$ is true for a fact member $f \in F$ iff $l \in L$:

$\wedge l' \in L' \wedge l_i \in L_i \wedge l_j \in L_j \wedge \left(f.l_i = l_i.k\right) \wedge \left(l_i.k,l.k\right) \in H^* \wedge \left(f.l_j = l_j.k\right) \wedge \left(l_j.k,l'.k\right) \in H'^* \wedge l.A,\ l'.A'$ holds

- $D_i.H.L.A\theta\ c$ is true for a fact member $f \in F$ iff:

$\exists lL \wedge l_i \in L_i \wedge \left(f.l_i = l_i.k\right) \wedge \left(l_i.k,l.k\right) \in H^* \wedge l.A\theta c$ holds

Intuitively, $D_i.H.L.A\theta D_j.H'.L'.A'$ is true for all fact members from F for which l_i and l_j are transitively connected to level members l and l' via the hierarchies H and H', respectively, and $l.A\theta l'.A'$ holds. For example, the query: *For each employee, compute the sales made to customers living in the city the employee works in*, the condition would be Employee.Territories.City.CityName = Customer.Lives.City.CityName, and it would evaluate to true only for such fact members that coincide at the City level when they roll-up to them.

Similarly, $D_i.H.L.A\theta c$ is true for all fact members in F for which an $l_i \in F$ is transitively connected to the level members $l \in L$ by following hierarchy H and $l.A\theta c$ holds. For example, the query *Compute the sales of the products belonging to the category Food*, is expressed by the condition Product.Categories.Category.CategoryName = "Food", and it evaluates to 'True' only for the fact members where SalesKey is 1, 2, 3, and 4 (see Figures 3 and 5).

Figure 5. Instance of level category

CategoryID	CategoryName	Description
c_1	*Food*	*Food items*
c_2	*Drink*	*Beverages*
...	...	...

Definition 14 (Dice). The Dice operator is defined as $\left(MD_s,F,\varphi\right) \rightarrow MD_o$, where MD_s is a source MD structure; F is a fact in MD_s and has an instance F; φ is a propositional formula as in Definitions 12, and 13, with logical operators $\wedge,\vee,$ and $\neg$; and MD_o is an output MD structure. The schema of F' is the same as the one of F, and its instance F' is:

$$F' = \left\{f \middle| F\left(f\right) \wedge \varphi\left(f\right)\right\}$$

The new fact, F' keeps only the fact members form F for which φ holds. To obtain *the sales where the quantity sold was more than five units and products were assigned to the category Food*, the dice operation is written as follows, and its result is the subset of the input fact Sales, shown in Figure 6.

Figure 6. Result of dice operation on fact sales

SalesKey	Category	Date	Quantity
2	c_1	t_3	10
3	c_1	t_1	15

Dice: (SalesDW, Sales, φ = Quantity > 5 $\wedge$ Product.Categories.Category.CategoryName = "Food") $\rightarrow$ Sales$^{'}$.

Project

The project operation removes a given dimension from a fact. Formally, the project operator is defined as follows.

Definition 15 (Project). The Project operation is defined as $\left(MD_s, F, D_j, \mathcal{M}'\right) \rightarrow MD_o$, where MD_s is a MD structure; F is a fact in MD_s; D_j is a dimension name in F.$\mathcal{D}$; and $\mathcal{M}' = \left\{\left(M_1, \alpha_1\right), \ldots, \left(M_{n'}, \alpha_{n'}\right)\right\}$ where M_i is a measure name in F, and $\alpha_i \in \left\{sum, average, \min, \max, count, count - distinct\right\}$ is an aggregation function. Also, if $F.\mathcal{M} \supset \mathcal{M}'$, then the measures in $F \cdot \mathcal{M} / \mathcal{M}'$ are removed. Finally, MD_o is a resultant MD structure.

Given the input fact F with schema $F\left(K : O_k, \mathcal{D}, \mathcal{M}\right)$ where $\mathcal{D} = \left(D_1, \ldots, D_m\right)$ and $D_j \in \mathcal{D} = \left(L_j, card_j\right)$, and its instance F containing members of the form $\left(l_1, \ldots, l_j, \ldots, l_m, m_1, \ldots, m_n\right)$ where $l_j \in L_{j\kappa}$, the schema and instance of F' is defined as follows:

- $F'\left(K : O_k, \mathcal{D}', \mathcal{M}\right)$, where $\mathcal{D}' = \left(D_1, \ldots, D_{j-1}, D_{j+1} \ldots, D_m\right)$;
- $F' = \{\left(k', l_1.k, \ldots, l_{j-1}.k, l_{j+1}.k, \ldots, l_m.k, m_1^{'}, \ldots, m_{n'}^{'}\right) \middle| k' \in dom\left(O_k\right) \wedge L_i\left(l_i\right)$ for:

$$i{=}1 \ldots m \wedge i \neq j \wedge m_x' = \alpha_x(\ \{\left(f.k, f.m_x\right) \middle| F\left(f\right) \wedge \left(f.l_i = l_i.k\right)$$

for $i{=}1 \ldots m,\ i \neq j\ \})$ for $x = 1, \ldots, n'\}$.

In Definition 15, the schema of the output fact F' is similar to the input fact F except for the removed dimension D_j. Intuitively, in F', the dimension coordinate l_j is first removed from F. Then, the tuples with the same dimension coordinate for the remaining dimensions are grouped. After that, for each group, the input measures $m_1, \ldots, m_{n'}$ are aggregated using their corresponding aggregate functions. In F', k' is a new key for each tuple, $l_i.k$ is a dimension coordinate of dimension D_i, and $m_x^{'}$ is an aggregated value of measure m_x, which is computed using the aggregate function α_x. Note that the dimension coordinate l_j corresponding to dimension D_j does not exist in an $f' \in F'$. For example, the result of removing the dimension Date from the fact Sales in Figure 3 by applying the Project operator is given in Figure 7:

Figure 7. Result of projecting dimension *Date* from fact *Sales*

SalesKey	ProductID	Quantity
1	p_1	30
2	p_2	5
3	p_3	8

Project: (SalesDW, Sales, Date, $\left\{\left(\text{Quantity} : sum\right)\right\}$) $\rightarrow \text{Sales}'$.

Drill-Across

The Drill-Across operation links data from two facts that have at least one dimension in common. It is formally defined as follows.

Definition 16 (Drill-Across). The Drill-Across operation is defined as $\left(MD_s, F_a, F_b, \Psi\right) \rightarrow MD_o$, where MD_s isaMDstructure; F_a and F_b arefactsin MD_s;and $\Psi = \left(D_1^a = D_1^b\right) \wedge \ldots \wedge \left(D_{n'}^a = D_{n'}^b\right)$ is a propositional formula such that $\left(D_i^a = D_i^b\right), i = 1, \ldots, n', D_i^a$ is a dimension name in F_a and D_i^b is a dimension name in F_b.

Given the input fact schemas F_a and F_b, and their associated instances F_a and F_b, respectively, the schema of F' and its instance F' are defined as follows:

- $F' = \left(K' : O_k, \mathcal{D}', \mathcal{M}'\right)$ where, $\mathcal{D}' = F_a.\mathcal{D} \cup F_b.\mathcal{D}$, and $\mathcal{M}' = F_a.\mathcal{M} \cup F_b.\mathcal{M}$

$$F' = \left\{\begin{array}{l}\left(k' \cup \left(f_a.l_1, \ldots, f_a.l_m, f_a.m_1, \ldots f_a.m_n\right) \cup \left(f_b.l_1, \ldots, f_b.l_m, f_b.m_1, \ldots f_b.m_n\right)\right) \\ \left|F_a\left(f_a\right) \wedge F_b\left(f_b\right) \wedge \left(f_a.D_i^a = f_b.D_i^b\right), \text{for } i = 1, \ldots, n'\right.\end{array}\right\}$$

The schema of F' is the union of the schemas of F_a and F_b. In the instance F', k' is the new key and the rest of the elements of a tuple in F' are the union of the elements of the tuples from F_a and F_b, respectively. Intuitively, the drill-across operation merges the fact members from facts F_a and F_b that have the same value for all the dimension coordinates provided in $\mathcal{D}_c$. This operation is like the inner equijoin of the relational algebra.

For example, the fact Shipment showed in Figure 8, stores the shipment details for each sale in the fact Sales in Figure 3. To *compute the sales quantity and freight paid together*, a drill-across operation can be performed. Notice that dimension Date is renamed as OrderDate in the Shipment fact, whereas the Date dimension represents the same date in the Sales fact. Since drill-across performs a union of schema elements, the information about the order date will be included twice in the resultant fact, once from Shipment and once from Sales. To avoid this, the redundant dimension can be renamed. For this, a Rename operation is defined (details are omitted for the sake of space). The result of the drill-across operator below is shown in Figure 9:

- Rename: (SalesDW, Shipment, OrderDate = Date) $\rightarrow \text{Sales}'$;

Figure 8. Fact shipment storing the freight for each sales transaction

ShipmentKey	ProductID	OrderDate	Freight
11	p_1	t_7	10
22	p_1	t_3	15
33	p_1	t_1	20
44	p_2	t_1	10
55	p_3	t_1	25

Figure 9. Result of drill-across operation on fact sales and shipment

Key	ProductID	Date	Quantity	Freight
111	p_1	t_7	5	10
222	p_1	t_3	10	15
333	p_1	t_1	15	20
444	p_2	t_1	5	10
555	p_3	t_1	8	25

- Drill-across: (SalesDW, Sales', Shipment, $\mathcal{D}_c = \{$ (ProductID = ProductID), (Date = Date) $\}$ $\rightarrow \text{Sales}''$.

A FORMAL TEMPORAL MULTIDIMENSIONAL MODEL

This section extends the formal MD model introduced in the previous section, with temporal features. After defining the data structure for the temporal MD model, the temporal equivalent of roll-up and dice operators are presented. In what follows, the *time domain* $\mathcal{T}$ is a discrete and linearly ordered set of elements called *instants*. Instants are isomorphic (i.e., structurally similar) to the natural numbers. A special instant in $\mathcal{T}$, called now, denotes the current time instant. A *time interval* ι is a convex set of instants over $\mathcal{T}$, and it is represented by two bounds t_s and t_e, denoting its starting and ending instants, respectively. This paper uses a closed-open representation of an interval defined as $\left[t_s, t_e\right) = \{t \in \mathcal{T} \left| t_s \leq t < t_e\right.\}$. Given $\Omega = \{$*meets, contains, equals, starts, precedes, overlaps, covers, disjoint, finishes, succeeds*$\}$, a set of synchronization constraints, and the set of intervals I and $I' \in \mathcal{T}$, $I \omega I'$, the semantics of $\omega \in \Omega$ between I and I' is the one defined by Allen's interval algebra (Allen, 1983) generalized for a non-convex set of intervals (Ladkin, 1986).

The value of a temporal attribute can be represented as a function that maps each instant of the time domain to a value of the domain of its base type. In particular, the value of a nontemporal attribute can be represented as a constant function that maps all instants of the time domain to a single value in the domain of its base type. However, this representation must be generalized when the attribute is bitemporal. In this case, the value of a bitemporal attribute can be a function that maps each instant in the time domain of one temporality, say transaction time, to a function that maps each instant in the time domain of the other temporality to a value of the domain of its base type.

Definition 17 (Temporal Domain of an Attribute). The temporal domain of an attribute A with base type b and temporality $T \subseteq \left\{VT, TT\right\}$, denoted $tdom\left(A\right)$, is defined as follows:

$$\left\{v \left| v\left(t\right) = c, \forall t \in \mathcal{T}, c \in dom\left(b\right)\right.\right\}, \qquad if\, T = \varnothing$$

$$\left\{v \left| v : \mathcal{T} \rightarrow dom\left(b\right)\right.\right\}, \qquad if \left|T\right| = 1$$

$$\left\{v \middle| v : \mathcal{T} \to \mathcal{T} \to dom\left(b\right)\right\}, \qquad if \left|T\right| = 2$$

Example 9. Consider a product's price represented as a bitemporal attribute, whose valid time value was 15€ during $\left[t_1, t_5\right)$; assume this was stored in the system from t_2 onwards (transaction time). At t_{10}, it was realized that during $\left[t_1, t_5\right)$, the valid price has been 10 . The functions $v'\left(t\right) = 15$ and $v''\left(t\right) = 10$, $t_1 \geq t < t_5$ represent the valid time value of the attribute. Further, the function $v\left(t\right) = v', t_2 \geq t \leq t_{10}$ and $v\left(t\right) = v'', t \geq t_{10}$ gives the valid time values known to the system before and after t_{10}.

A level may have both temporal and nontemporal attributes. Also, a member of a temporal level has a lifespan, that is the time during which it exists. The lifespan of each level member is a subset of the time domain. The lifespan of a nontemporal level member covers the complete time domain. Furthermore, a member's lifespan can be a set of consecutive instants, making it one interval, or may have gaps between the two instants, making it a set of intervals. Also, for an attribute with VT temporality, the valid time value must be defined throughout a member's lifespan.

Definition 18 (Level Schema). The schema of a level is defined as $L\left(A_k : O_k, A_1 : O_1\left(T_1\right), \ldots, A_n : O_n\left(T_n\right), T_L\right)$, where L is the level name; A_k is the name of the key attribute and it has the base type O_k; each $A_i, i = 1, \ldots, n$, is an attribute name and it has the base type O_i, and $T_i \subseteq \left\{VT, TT\right\}$ are the temporality types associated with A_i. Also, all attribute names are unique in L. Furthermore, if A_i is nontemporal, then $T_i = \varnothing$. Finally, $T_L \in \left\{LS_i, LS_s\right\}$ is the temporality of L and if it is a nontemporal level, then $L.T_L = \varnothing$.

Example 10. The schema of level Product is:

$$Product\left(ProductID : Integer, ProductName : String\left(\varnothing\right), UnitPrice : Decimal\left(\left\{VT\right\}\right), T_L = LS_i\right)$$

Definition 19 (Level Instance). An instance L of a level $L \in \mathcal{L}$ is a set of level members defined as:

$$L \subset \left\{k \times a_1 \times \ldots \times a_n \times ls \middle| k \in dom\left(A_k\right) \wedge a_i \in tdom\left(A_i\right), i = 1, \ldots, n, \wedge ls \subseteq \mathcal{T}\right\} \cup \left\{\bar{l}\right\}$$

where $\left\{\bar{l}\right\}$ is the default member of L. Also, if $L.T_L = \varnothing$, then $l.ls = \left(-\infty, +\infty\right), \forall l \in L$, and if $L.T_L = LS_i$, then $l.ls$ is a convex set. Furthermore, for each attribute $l.A$ such that $VT \in l.A.T, dom\left(l.A\right) = l.ls$. Also, L satisfies the key constraint as in Definition 2. Finally, $\text{lifespan}\left(k\right), k \in L_\kappa$, gives the lifespan and $\delta\left(k, A.temp, t\right)$ gives the value of attribute A of for temporality type $temp$ at instant t of the level member with the key k.

Example 11. Consider product p_1, introduced at instant t_1 with unit price 10 , belonging to category c_1. At t_2, the unit price changed to 15 , at t_3 it was assigned to category c_2, and at t_4, it was discontinued. Finally, the product was reintroduced at t_6 with the unit price 18 , and it is available for selling until today. The evolution of p_1 and the instance of level Product after evolution are shown in Figure 10 and Figure 11, respectively.

Figure 10. Timeline of product evolution

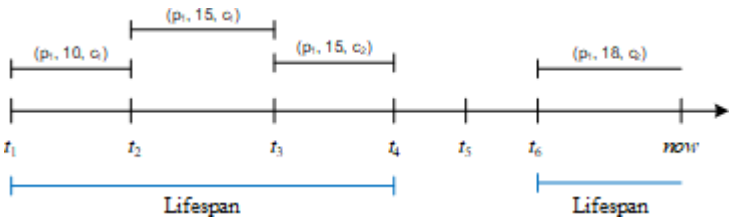


Figure 11. Instance of level product with nontemporal attribute ProductName and temporal attribute UnitPrice

ProductID	ProductName	UnitPrice	Is
p_1	Milk	$\{([t_1,t_2),10),([t_2,t_3),15),([t_6,now),18)\}$	$\{[t_1,t_4),[t_6,now)\}$
p_2	Cola	$\{([t_1,now),12)\}$	$\{[t_1,now)\}$
p_3	Juice	$\{[t_1,now),8\}$	$\{[t_1,now)\}$

A temporal aggregation relationship between a child and a parent level is a function that maps each instant of the time domain to a relation between the members of the child and parent levels. For a nontemporal aggregation relationship, this function maps all instants in the time domain to a fixed relation between the members of the child and parent levels. The instance of a bitemporal aggregation relationship is a function that maps each instant of the time domain of one temporality, say transaction time, to a function that maps each instant of the time domain of the other temporality to a relation between the members of the child and parent levels. In addition to a synchronization constraint, the aggregation relationship instance must ensure that a child member is associated with a parent member throughout its lifespan. Furthermore, both a child and its parent must exist at each instant in the validity interval of a relationship.

Definition 20 (Aggregation Relationship Schema). An aggregation relationship R is defined by $R\left(L_c, L_p, card, T, \omega\right)$, where R is the aggregation relationship name; $L_c, L_p \in \mathcal{L}$ are the child and parent levels, respectively, $card \in Card$ specifies the cardinality between the child level L_c and the parent level L_p at each instant of the time domain, $T \subseteq \left\{VT, TT\right\}$ specifies the temporality of R and if R is nontemporal, then $T = \varnothing$, each $\omega \in \Omega$ is a synchronization constraint, and if L_c or L_p is nontemporal, then $\omega = \varnothing$.

Example 12. The schema of the temporal aggregation relationships between levels Product and Category is Categories (Product, Category, m-1, $\{\text{VT}\}$, overlaps).

Definition 21 (Aggregation Relationship Instance). Given the set $\Pi = R_i, \ldots, R_n\}$ such that $\Pi \subset \mathcal{P}\left(L_{c^\circ} \times L_{p^\circ}\right)$ and each $R_i \in \Pi$ is a relation between members of levels L_c and L_p as in Definition 4, an aggregation relationship instance of schema $R\left(L_c, L_p, card, T, \omega\right)$, denoted ρ_R, is as follows:

$$\rho_R\left(t\right) = R, \forall t \in \mathcal{T}, R \in \Pi, \qquad if\, T = \varnothing$$

$$\rho_R : \mathcal{T} \to \Pi, \qquad if\ \left|T\right| = 1$$

$$\rho_R : \mathcal{T} \to \mathcal{T} \to \Pi, \qquad if \left|T\right| = 2$$

Furthermore, $\forall\left(l,l'\right)\in\rho_R\left(t\right), t\in l.ls\ and\ t\in l'.ls.lifespan\left(l\right)\omega lifespan\left(l'\right)$ is satisfied. Also, the value of function ρ_R at instant t is called the state of the relationship at t.

Example 13. For product p_1, the aggregation relationship instance $\rho_{Categories}\in\mathcal{R}$ is:

$$\left\{\left(\left[t_1,t_3\right),\left(p_1,c_1\right)\right),\left(\left[t_3,t_4\right),\left(p_1,c_2\right)\right),\left(\left[t_6,now\right),\left(p_1,c_1\right)\right)\right\}$$

The instance of the aggregation relationship Categories at t_2 is:

$$\rho_{Categories}\left(t_2\right)=\left\{\left(p_1,c_1\right)\ldots,\left(p_{default},c_{default}\right)\right\}$$

A temporal roll-up hierarchy has at least one temporal aggregation relationship. A nontemporal hierarchy consists of nontemporal aggregation relationships only. The state of a temporal hierarchy at an instant can be obtained from the state of its constituent aggregation relationships at that instant. A nontemporal hierarchy has the same state for all instants in the time domain.

Definition 22 (Roll-Up Hierarchy Schema). The schema of a roll-up hierarchy H is the same as in Definition 5. However, each $H.R_i^{'}, i=1,\ldots,n$, is an aggregation relationship name as in Definition 20.

Definition 23 (Roll-Up Hierarchy Instance). The instance of a roll-up hierarchy H is given by $H=\rho_{R_1}\cup\ldots\cup\rho_{R_n}$, where each $\rho_{R_i}, i=1,\ldots,n$, is an instance of the aggregation relationship $H.R_i$. Furthermore, the instance of H at an instant t is $H_t=\rho_{R_1}\left(t\right)\cup\ldots\cup\rho_{R_n}\left(t\right)$, and H_t^*, called the state of H at t, is the transitive closure of H_t as in Definition 6.

Example 14. An instance of the Categories hierarchy at an instant t_1 is $\text{Categories}_{t_1}=\left\{\left(p_1,c_1\right),\left(p_2,c_1\right),\ldots,\left(p_3,c_1\right),\ldots,\left(p_n,c_n\right)\right\}$. The current instance of the same hierarchy is $\text{Categories}_{\text{now}}=\left\{\left(p_1,c_2\right),\left(p_2,c_2\right),\ldots,\left(p_3,c_1\right),\ldots,\left(p_n,c_n\right)\right\}$. Notice that the product p_1 is assigned to the category c_2 now.

If a member of a temporal level is related to a fact member, then an optional fact synchronization constraint ensures that this level member exists at the time of the fact occurrence. Usually, for each fact member, the time of a fact occurrence is given by an attribute of another related level member.

Definition 24 (Fact Schema). A fact F is defined by $F\left(K:O_k,\mathcal{D},\mathcal{M},\mathcal{C}\right)$, where F is the fact name, and $K,\mathcal{D}$, and $\mathcal{M}$ are the same as in Definition 7. The set $\mathcal{C}=\left\{C_1,\ldots,C_n\right\}$ defines fact synchronization constraints and each $C_i\in\mathcal{C}, i=1,\ldots,n$, is a pair $C_i\left(D,D'.A\right)$ such that D and D' are dimensions in $\mathcal{D}$, and $D'.A$ is an attribute in $D'.L'$. The key name K, a dimension name D, a measure name M, and a fact synchronization constraint name C are unique in F.

Example. The schema of fact Sales is the same as in Example 7. It also includes a set of fact synchronization constraints $\mathcal{C}=\left\{\text{Product_SalesDate}\left(\text{Product,SalesDate.Date}\right)\right\}$. The constraint Product_SalesDate ensures that each product exists on the date of the sale.

Definition 25 (Fact Instance). A fact instance $F\in\mathcal{F}$ is a set of fact members as in Definition 8.

Additionally, F satisfies all fact synchronization constraints in $\mathcal{C}$ and a $C\left(D, D'.A\right) \in \mathcal{C}$ is satisfied iff:

$$\forall f \in F, \exists l \in D.L \wedge \exists l' \in D'.L' \wedge f.l = l.k \wedge f.l' = l'.k \wedge l'.A \in l.ls$$

Definition 26 (Temporal Multidimensional Schema). A temporal multidimensional schema is defined by $\mathcal{S}\left(\mathcal{L}, \mathcal{R}, \mathcal{H}, \mathcal{F}\right)$, where $\mathcal{S}$ is the schema name; $\mathcal{L} = \left\{L_1, \ldots, L_n\right\}$ is a set of temporal and nontemporal levels, and all level names are unique in $\mathcal{L}$ and have the same base type O_k for the key attribute A_k; $\mathcal{R} = \left\{R_1, \ldots, R_m\right\}$ is a set of temporal and nontemporal aggregation relationships, and all relationships name in $\mathcal{R}$ are unique; $\mathcal{H} = \left\{H_1, \ldots, H_n\right\}$ is a set of temporal and nontemporal hierarchies, and all hierarchy names in $\mathcal{H}$ are unique; and $\mathcal{F} = \left\{F_1, \ldots, F_n\right\}$ is a set of facts, and all fact names in $\mathcal{F}$ are unique.

Definition 27 (Temporal Multidimensional Instance). Given a schema $\mathcal{S}\left(\mathcal{L}, \mathcal{R}, \mathcal{H}, \mathcal{F}\right)$ as in Definition 26, a temporal MD instance $\mathcal{I}$ of $\mathcal{S}$ is defined as $\mathcal{I}\left(\mathcal{L}, \mathcal{R}, \mathcal{H}, \mathcal{F}\right)$, where $\mathcal{L} = \left\{L_1, \ldots, L_n\right\}$ is a set of temporal and nontemporal level instances, $\mathcal{R} = \left\{R_1, \ldots, R_m\right\}$ is a set of temporal and nontemporal aggregation relationship instances, $\mathcal{H} = \left\{H_1, \ldots, H_n\right\}$ is a set of temporal and nontemporal hierarchy instances, and $\mathcal{F} = \left\{F_1, \ldots, F_n\right\}$ is a set of fact instances.

The temporal MD model can be used regardless the fact that operational data sources are temporal or nontemporal. If a user does not wish to use any temporal feature, then the model behaves like a traditional MD model. In case where data sources are temporal, it is straightforward to load data and maintain the history in the DW. If data sources are nontemporal and it is intended to keep the history of changes in the DW, then the ETL processes must be designed to take into account the current state of a member or a relationship, and record the new state accordingly.

Temporal OLAP Operations

This section introduces a formal syntax and semantics for the temporal equivalents of the roll-up and dice operators, which are associated with the formal model previously explained.

Temporal Roll-Up

The temporal roll-up is like the traditional one except that it must consider the state of the roll-up hierarchy at a specific instant. Depending on the instant considered for obtaining the state of the hierarchy, the operation is called temporally consistent or time-slice roll-up. In the temporally consistent roll-up, for each fact member, the state of the roll-up hierarchy is obtained at a specific instant, typically an instant when a fact occurred in the business world and the instance value is provided by the attribute of another dimension. Formally, a temporally consistent roll-up is defined as follows.

Definition 28 (Temporally Consistent Roll-Up). The Temporally consistent roll-up is defined as $\left(MD_s, F, D_j, H, D_{\ddot{A}}.A, \mathcal{M}'\right) \rightarrow MD_o$, where MD_s is a source temporal MD data structure, F is a fact name in MD_s, D_j is a dimension in $F.\mathcal{D}$, H is a hierarchy such that $H.L_b = D_j.L_j$, $D_{\ddot{A}}.A$ is an attribute name in level $D.L_{\ddot{A}}$ where D_τ is a dimension in $F.\mathcal{D}$, $\mathcal{M}' = \left\{\left(M_1, \alpha_1\right), \ldots, \left(M_{n'}, \alpha_{n'}\right)\right\}$ where $M_i, i = 1, \ldots, n'$, is a measure name in F, and $\alpha_i \in \left\{sum, average, \min, \max, count, count\text{ - }distinct\right\}$ is an aggregation function. Also, if

$F.\mathcal{M} \supset \mathcal{M}'$, then the measures in $F\,\mathcal{M}/\mathcal{M}'$ are removed. Finally, MD_o is an output temporal MD database.

Given the input fact F with the schema $F\left(K : O_k, \mathcal{D}, \mathcal{M}\right)$ where $D_j \in \mathcal{D} = \left(L_j, card_j\right)$, and $D_\tau \in \mathcal{D} = \left(L_\tau, card_\tau\right)$, and its instance F containing members of the form $\left(d_1, \ldots, d_j, \ldots, d_{\ddot{A}} \ldots d_m, m_1, \ldots, m_n\right)$ where $d_j \in L_{j\kappa}$ and $d_\tau \in L_{\tau\kappa}$, the schema and instance of F' are given as follows:

- $F' = \left(K' : O_k, \mathcal{D}', \mathcal{M}'\right)$ where $\mathcal{D}' = \left(D_1, \ldots, D_j^{'}, \ldots, D_m\right)$ and $D_j^{'} = \left(H.L_t, 1\text{-}m\right)$:

$$\begin{aligned}
&[\![F']\!] = \Big\{\left(k', l_1 \cdot k, \ldots, l_t \cdot k, \ldots, l_\tau \cdot k, \ldots, l_m \cdot k, m'_1, \ldots, m'_{n'}\right) \Big| k' \in dom\left(O_k\right) \wedge \\
&[\![L_i]\!]\left(l_i\right) for\ \ i = 1 \ldots m, i \neq j, i \neq \tau, \wedge [\![L_\tau]\!]\left(l_\tau\right) \wedge H \cdot [\![L_\tau]\!]\left(l_\tau\right) \wedge m'_x = \\
&\alpha_x\left(f \cdot k, f \cdot m_x\right) \Big| [\![F]\!]\left(f\right) \wedge f \cdot l_i = l_i \cdot k\ for\ \ i = 1 \ldots m, i \neq j \wedge \left(f \cdot l_j, l_t \cdot k\right) \in [\![H]\!]^*_{l_\tau \cdot A} / \\
&df_{l_j}\ \ for\ \ x = 1, \ldots, n' \Big\}
\end{aligned}$$

In the above definition of F', for each fact member $f \in F$, the dimension coordinate $f.D_\tau$ specifies the time instant at which the state of the hierarchy H should be obtained. For each $f \in F$, first, the state of the hierarchy $H_{l_{\tau}.A}$ at the instant $l_{\ddot{A}}.A$ is obtained, and then similar to the non-temporal roll-up, all fact members are aggregated using the hierarchy instance $H_{l_{\ddot{A}}.A}$.

The Time-slice roll-up aggregates measures, for all fact members, using the same state of hierarchy obtained at a fixed instant t. The formal definition is omitted for the sake of space. The examples next illustrate both operations.

Example 16. As an example of the temporally consistent roll-up, the query: *Compute the total quantity sold, aggregated by category, considering the product-category assignment valid at the time of sales*, is expressed as follows:

- Temporally consistent roll-up: (SalesDW, Sales, Product, Categories,Date.Date, { Quantity: *sum* }) $\rightarrow$ Sales'.

For Example 11, the result of the temporally consistent roll-up over the Sales fact in Figure 3 is shown in Figure 13. In the resulting fact, the sales of each product are aggregated up to the category to which it belonged at the time in Date.

As an example of the time-slice roll-up, consider the query: *the total quantity sold, aggregated by category, using the product-category assignment valid now* is expressed as:

- Time-slice roll-up: (SalesDW, Sales, Product, Categories, now, { Quantity: *sum* }) $\rightarrow$ Sales'.

Considering Example 11, the result of the above operation over the fact Sales in Figure 3, is shown in Figure 14. As depicted in Figure 12, the sales of product p_1 are aggregated to category c_2. To ensure summarizability in a temporal roll up, the temporal granularity of t or $f.D_{\ddot{A}}$ must be less

Figure 12. The instance of categories valid now

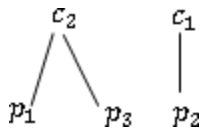


Figure 13. Fact sales after temporally consistent roll-up to level category using categories hierarchy at SalesDate

SalesKey	Category	Date	Quantity
1	c_1	t_1	20
2	c_2	t_3	10
3	c_2	t_7	5
4	c_2	t_1	8

Figure 14. Fact sales after time-slice roll-up to level category using categories hierarchy valid now

SalesKey	Category	Date	Quantity
1	c_1	t_1	5
2	c_2	t_1	23
3	c_2	t_3	10
4	c_2	t_7	5

than or equal to the temporal granularity of all temporal aggregation relationships of the roll-up hierarchy H.

Temporal Dice

Like in the nontemporal case, the Temporal Dice operator reduces a given fact to a sub-fact based on a Boolean condition on measure values and attribute values of the connected levels. Also, as in the case of the roll-up, there are two possibilities. For example, consider the query: *Sales of products that are assigned to category* c_2 *and have a price higher than* 10. For the condition over the category, either of the two interpretations explained for the temporal roll-up case, could be applied. Further, since the price of a product may also change over time, the time-slice and temporally consistent description apply to the price measure as well. To be able to apply a selection on a temporal attribute value, the following conditions are used.

Definition 29 (Time-Slice Atomic Condition on Temporal Attribute Values). Given a fact F and a time instant $t, D_i.H.L.A.temp\theta D_j.H'.L'.A'.temp$ or $D_i.H.L.A.temp\theta c$ is a temporal atomic condition on fact F, where D_i and D_j are dimensions in $F.\mathcal{D}$, H and H' are non-temporal or temporal hierarchies such that $H.L_b = D_i.L_i$ and $H'.L_b = D_j.L_j$, A and A' are attributes of levels $L \in levels(H)$ and $L' \in levels(H')$, respectively, and $c \in dom(B)$, such that B is a base type and a value constant.

Given the instance F of F containing fact members of the form $(l_1,\ldots,l_i,\ldots,l_j,\ldots l_m,m_1,\ldots,m_n)$ where $l_i \in D_i.L_{i\kappa}$ and $l_j \in D_j.L_{j\kappa}$, the semantics of $D_i.H.L.A.temp\theta D_j.H'.L'.A'.temp$ and $D_i.H.L.A.temp\theta c$ are as follows:

- The condition $D_i.H.L.A.temp\theta D_j.H'.L'.A'.temp$ is true for fact member $f \in F$ iff:

$$l \in L \wedge l' \in L' \wedge l_i \in L_i \wedge l_j \in L_j \wedge \left(f.l_i = l_i.k\right) \wedge \left(l_i.k, l.k\right) \in H_t^* \wedge \left(f.l_j = l_j.k\right) \wedge \left(l_j.k, l'.k\right) \in H_t'^* \wedge \delta\left(l.k, A.temp, t\right)\theta\delta\left(l'.k, A'.temp, t\right) \text{ holds}$$

- The condition $D_i.H.L.A.temp\theta c$ is true for fact member $f \in F$ iff:

$$l \in L \wedge l_i \in L_i \wedge \left(f.l_i = l_i.k\right) \wedge \left(l_i.k, l.k\right) \in H_t^* \wedge \delta\left(l.k, A.temp, t\right)\theta c \text{ holds}$$

The first condition is true for all fact members from F for which l_i and l_j are transitively connected at instant t to level members l and l' via hierarchies H and H', respectively and θ holds between the values of attributes $l.A$ and $l'.A'$ at instant t. Similarly, the second condition is true for all fact members for which l_i is transitively connected to level member l at instant t via hierarchy H, and θ hols between the value of temporal attribute $l.A$ at instant t and c.

For example, Product.Categories.Price.UnitPrice.VT $>$ 10 on fact Sales at instant *now* is true for all fact members that have a product with the current price higher than 10 . The temporally consistent equivalent of the above condition is the same, except that for all $f \in F$, the condition is evaluated at an instant provided by dimension coordinate $f.D_\tau$. For example, given fact Sales, temporally consistent condition Product.Categories.Price.UnitPrice.VT >10 on fact Sales with $D_{\text{Ä}} =$ Date is true for fact members having a product with a price higher than 10 on the date of the sale.

Definition 30 (Temporal Dice). The Temporal Dice is defined as $\left(MD_s, F, \varphi_\tau\right) \rightarrow MD_o$, where MD_s is an input MD database, F is a fact in MD_s and has instance F, φ_τ is a propositional formula and it consists of atoms as in Definitions 12, 13 and 29, and logical operators $\wedge$, $\vee$, and $\neg$, and MD_o is an output temporal MD structure. The schema of F' is the same as F and its instance F' is defined as follows:

$$F' = \left\{f \middle| F\left(f\right) \wedge \varphi_\tau\left(f\right)\right\}$$

To obtain the sales of the product having a price more than 10€ and assigned to category c_2 at the time of sales, the following operation can be used.

Temporal Dice: (SalesDW, Sales, φ_τ = Product.Categories.Price.UnitPrice.VT $>10 \wedge$ Product. Categories.Category.Categoryname = 'Food') $\rightarrow$ Sales'

Given the product instance from Figure 11, the result of the above operation over fact Sales from Figure 3 is shown in Figure 15.

Figure 15. Result of temporal dice operation on fact sales

SalesKey	Product	Date	Quantity
1	p_1	t_3	10
2	p_1	t_7	5

IMPLEMENTING A TEMPORAL DATA WAREHOUSE IN RELATIONAL DATABASES

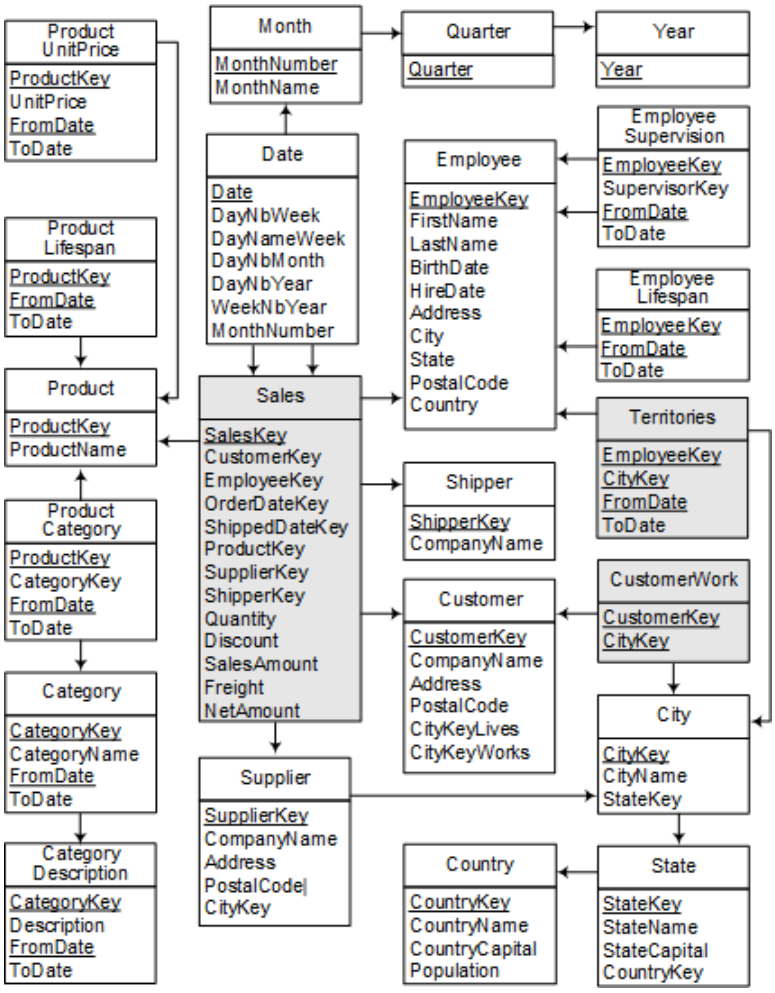
The SQL-2011 standard includes some temporal features (cf. Related Work). Therefore, vendors have started to provide limited support for these features in their RDBMSs (Al-Kateb, et al., 2013). However, most DW are implemented over non-temporal RDBMS. In addition, experience shows that temporal extensions to databases are seldom used, and failed to gain public acceptance, which supports the idea of using standard SQL to implement temporal OLAP operations (Wang & Zaniolo, 2000; Wang, Zaniolo, & Zhou, 2005). Therefore, it seems reasonable to implement temporal relational DW over traditional RDBMS. Along these lines, this section shows a translation from the formal temporal MD model to a relational schema. Over this model, it is shown through a comprehensive example, how the formal temporal OLAP operations introduced in the previous section, can be converted to temporal relational operations and implemented in standard SQL, without the in-built support for temporal operations (like the ones provided in SQL-2011).

Relational Schema Mapping

Figure 16 shows the relational representation of the schema shown in Figure 1. The translation is explained next:

- **Level Mapping:** A level $L \in \mathcal{L}$, provided it is not related to a fact with a one-to-one relationship, is mapped to a table that contains all nontemporal attributes of the level. A surrogate key is added as the primary key of the table. Each *temporal attribute* of the level is mapped as a separate table. This table contains the primary key of the level, the value attribute, and two attributes FromDate and ToDate to associate the interval for storing the evolution of the value. The key of the table is composed of the primary key of the level and FromDate. For example, level Product is mapped to the table with the same name. The temporal attribute UnitPrice of level Product is mapped to table ProductUnitPrice. The mapping of *lifespan* depends on whether it is an interval (LS_i) or a set of intervals (LS_s). In the former case, two attributes FromDate and ToDate are added to the table corresponding to the level. In the latter case, an additional table is created to store the level's lifespan, and it contains the primary key of the level and two attributes FromDate and ToDate. For example, the lifespan of level Product is stored in table ProductLifespan because it is an interval set. On the other hand, the lifespan of categories is captured by the attributes FromDate and ToDate of table Category because it is an interval;
- **Fact Mapping:** A fact $F \in \mathcal{F}$ is mapped to a table that includes as attributes all measures of the fact. Further, a surrogate key is added to the table. In addition, if the fact has an associated synchronization constraint, a trigger may be added to ensure that the constraint is satisfied for all fact members. Note that additional attributes are added to tables corresponding to a level and a fact when mapping relationships. As an example, applying the above rules to the fact Sales results in a table that contains the surrogate keys of all levels, as well as the corresponding referential integrity constraints and triggers to ensure fact synchronization constraints;

Figure 16. Relational representation of the example data warehouse



- **Relationship Mapping:** A relationship between either a fact F and a level L, or between parent-child levels L_p and L_c can be mapped in three different ways, depending on its cardinality:
  - If the relationship is one-to-one, the table corresponding to the fact or to the child level is extended with all the attributes of the level or the parent level, respectively;
  - If the relationship is one-to-many, the table corresponding to the fact or to the child level is extended with the surrogate key of the table corresponding to the base level or the parent level, respectively, that is, there is a foreign key in the fact or child table pointing to the other table;
  - If the relationship is many-to-many, a new bridge table is created that contains as attributes the surrogate keys of the tables corresponding to the fact and the dimension level, or the parent and child levels, respectively. The key of the table is the combination of both surrogate keys.

Further, if the relationship has an associated synchronization constraint, a trigger may be added to ensure that the constraint is satisfied by all instances of the relationship. Since time-invariant surrogates identify level members, this mapping does not depend on whether the levels are temporal or not. For example, the mapping of the parent-child relationship between levels Product and Category

gives the table ProductCategory. The instance of hierarchies can be obtained by joining the constituent relationship instances; therefore, we do not need to map it explicitly. However, the data dictionary may include the hierarchy schema that can be exploited by an application to generate the SQL to obtain the hierarchy instances automatically.

- **Consistency Constraints:** Many constraints must enforce the semantics of time and data integrity when implementing a temporal data warehouse over a relational database. Whenever possible, these constraints are applied as explicit integrity constraints; otherwise, triggers are used. Consider the tables for level Product in Figure 16, i.e., tables Product, ProductLifespan, ProductUnitPrice, and ProductCategory. Several constraints must be defined on these tables to ensure data integrity. Some of these constraints are:
  - The interval defined by FromDate and ToDate must be valid;
  - The lifespan of a member should be a unique tuple; and
  - All lifespan intervals of a member should be disjoint.

The first two constraints can be implemented via two check constraints, and the third one is enforced via a trigger. Similarly, triggers are also required to ensure additional constraints such as a temporal attribute's value must be defined for each instant in a member's lifespan; and for temporal parent-child relationships, the lifespan of a relationship instance must be covered by the intersection of the lifespans of the participating members. A trigger can also be used to ensure that the lifespan of a child member is equal to the union of the lifespans of its links to all parent members. Further, temporal parent-child relationships must satisfy a constraint stating that the lifespan of a relationship instance must be covered by the intersection of the lifespans of the participating members. A trigger ensures this constraint for the relationship between products and categories for the lifespan of products.

Temporal Relational Operations

Querying time-varying data in a relational schema requires extending the traditional relational operations. A detailed explanation of these operations can be found in (Snodgrass R. T., 1999) (Ahmed, Zimányi, & Wrembel, 2015) (Zimányi, 2006). For completeness, a short description of these extensions of the temporal operators is given next.

The *temporal selection* is like the traditional one, except that the selection predicate may include a condition on the time interval during which a temporal attribute or a temporal relationship is valid. A particular case of temporal selection is a *time-slice* that returns the tuples of a relation valid at a given instant. The *temporal projection* returns a set of computed attributes with an associated time interval from a relation and often requires coalescing the value-equivalent tuples. The *temporal coalescing* merges together the tuples that have the same non-temporal attribute values but have either adjacent or overlapping associated intervals. The result is a single tuple representing the interval during which the value is valid, obtained from the temporal coalescing. The *temporal join* merges the tuples of joining relations if the join condition is met, and the joined records are temporally coincident during the given period. The *temporal union* is equivalent to the traditional relational one and returns the set union of two relations. The *temporal difference* operation is typically used to filter out information from a superset of potential answers.

Querying a Temporal Data Warehouse in SQL

This section shows example queries addressing the example temporal DW. Some of these queries use the temporal relational operations from the previous section.

Query 1. Compute the average sales by category as of January 1st, 1997.

This is a time-slice roll-up query, which requires aggregating the sales using the state of the Categories valid on January 1st, 1997. The query reads in SQL:

```
SELECT        C.CategoryName, AVG(S.SalesAmount) as AvgSalesAmount
FROM          Sales S, Product P, ProductCategory PC, Category C
WHERE         S.ProductKey = P.ProductKey
AND           P.ProductKey = PC.ProductKey
AND           PC.CategoryKey = C.CategoryKey
AND           PC.FromDate <= '1997-01-01' AND '1997-01-01' < PC.ToDate
GROUP BY      C.CategoryName
```

The SQL query performs a join of the tables Sales, Product, ProductCategory, and Category to extract the product to category assignment valid at the requested date. Then, the sales of each category are aggregated using the average function to obtain the average sales amount per category.

Query 2. Calculate the total sales amount per customer, year, and product category.

This query performs the traditional roll-up using the Calendar hierarchy, and a temporally consistent roll-up using the Categories hierarchy. The temporally consistent roll-up considers the state of Categories hierarchies valid on the SalesDate. The SQL query is expressed as:

```
SELECT          C.CompanyName, D.Year, A.CategoryName,
                FORMAT(SUM(SalesAmount), '$###,##0.00') AS SalesAmount
FROM            Sales S, Customer C, Date D, Product P,
                ProductCategory PC, Category A
WHERE           S.CustomerKey = C.CustomerKey AND S.OrderDateKey = D.DateKey
AND             S.ProductKey = P.ProductKey AND P.ProductKey = PC.ProductKey
AND             PC.CategoryKey = A.CategoryKey
AND             PC.FromDate <= D.Date AND D.Date < PC.ToDate
GROUP BY        C.CompanyName, D.Year, A.CategoryName
```

The query considers that the assignment of products to categories varies in time. These assignments are stored in table ProductCategory. The last line of the WHERE clause ensures that products are rolled up to the category they were associated with, at the *order date*.

Query 3. List the personal sales amount made by an employee compared with the total sales amount made by herself and her subordinates during 1997.

This query is somehow more involved and requires a detailed explanation. First, a dice on the Sales fact is performed, to keep only the sales made in 1997. Then, a temporally consistent recursive roll-up is applied on the diced fact, to obtain the sales of each supervisor. Also, the sales of each employee in 1997, regardless of the supervisor, are computed by projecting all other dimensions except Employee from the diced fact. Finally, the sales of supervisors and employees can be combined in a single fact using a drill-across operation. Note that the roll-up operator in this paper does not include the semantics of a recursive roll-up. However, such a roll-up can be achieved by combining a series of roll-up operations, each for a level in the recursive hierarchy. The SQL query is:

```
WITH Supervision AS (
     SELECT        EmployeeKey, SupervisorKey, FromDate, ToDate
     FROM          EmployeeSupervision
     WHERE         SupervisorKey IS NOT NULL
     UNION ALL
     SELECT        E.EmployeeKey, S.SupervisorKey,
                   dbo.MaxDate(S.FromDate, E.FromDate) AS FromDate,
                   dbo.MinDate(S.ToDate, E.ToDate) AS ToDate
     FROM          Supervision S, EmployeeSupervision E
     WHERE         S.EmployeeKey = E.SupervisorKey AND
                   dbo.MaxDate(S.FromDate, E.FromDate) <
                   dbo.MinDate(S.ToDate, E.ToDate)),
SalesByEmp1997 AS (
     SELECT        EmployeeKey, SUM(S.SalesAmount) AS SalesAmount
     FROM          Sales S, Date D
     WHERE         S.OrderDateKey = D.DateKey AND D.Year = 1997
     GROUP BY      EmployeeKey),
SalesBySubord1997 AS (
     SELECT        SupervisorKey AS EmployeeKey,
                   SUM(S.SalesAmount) AS SalesAmount
     FROM          Sales S LEFT OUTER JOIN Supervision U ON
                   S.EmployeeKey = U.EmployeeKey JOIN Date D ON
                   S.OrderDateKey = D.DateKey
     WHERE         D.Year = 1997 AND
                   ((U.FromDate <= D.Date AND D.Date < U.ToDate) OR
                   U.FromDate IS NULL)
     GROUP BY      SupervisorKey)
SELECT FirstName + ' ' + LastName AS EmployeeName,
       FORMAT(S1.SalesAmount, '$###,##0.00') AS PersonalSales,
       FORMAT(ISNULL(S1.SalesAmount + S2.SalesAmount, S1.SalesAmount),
       '$###,##0.00') AS PersSubordSales
FROM Employee E JOIN SalesByEmp1997 S1 ON
     E.EmployeeKey = S1.EmployeeKey LEFT OUTER JOIN
     SalesBySubord1997 S2 ON S1.EmployeeKey = S2.EmployeeKey
ORDER BY EmployeeName
```

The first temporary table Supervision computes the temporal transitive closure of the supervision relationship using a with a recursive query. The second temporary table SalesByEmp1997 computes the total sales by employee during 1997 regardless the supervisors. The third temporary table SalesByEmpSup1997 calculates the total sales by employee and supervisor during 1997. This requires a temporal join between the EmployeeSupervision table and the recursive view Supervision. The main query is composed of the union of two queries. The first one computes personal sales and the total sales amount for supervisors. For this, the inner query in the FROM clause computes the total amount made by the subordinates of an employee, and the additional join with the table SalesByEmp1997 is used to obtain the total sales of a supervisor in order to add the two amounts. Finally, the second query in the union retrieves, from the SalesByEmpSup1997, the data of the employees who had no supervisor during 1997.

Query 4. For each employee, compute the total sales amount of the products she sold, with unit price higher than 30€ at the time of the sale.

Since the unit price of products is a temporal attribute, a temporal dice over the Sales fact is used to keep only the sales of the product that have a price higher than 30€ at the time of the sale. Then, all other dimensions except employees are projected from the diced fact. The query can be written in SQL as follows:

```
SELECT      FirstName + ' ' + LastName AS EmployeeName,
            FORMAT(SUM(SalesAmount), '$###,##0.00') AS SalesAmount
FROM        Sales S, Date D, Employee E, Product P, ProductUnitPrice U
WHERE       S.OrderDateKey = D.DateKey AND S.EmployeeKey = E.EmployeeKey AND
            S.ProductKey = P.ProductKey AND P.ProductKey = U.ProductKey AND
            U.UnitPrice > 30 AND U.FromDate <= D.Date AND D.Date < U.ToDate
GROUP BY    FirstName + ' ' + LastName
ORDER BY    FirstName + ' ' + LastName
```

The temporal attribute UnitPrice of level Product is stored in table ProductUnitPrice. The query first performs a temporal selection over the price and then groups the results by an employee to compute the total sales amount.

Query 5. Compute the total sales amount by supervisor.

In this query, a temporal dice if performed first, keeping only the sales of employees who have a supervisor at the time of the sale. Then, a temporally consistent roll-up aggregates the sales of an employee to her supervisor at the time of the sales. The SQL query is given below:

```
WITH Supervisors AS (
         SELECT     SupervisorKey AS EmployeeKey, FromDate, ToDate
         FROM       EmployeeSupervision
         WHERE      SupervisorKey IS NOT NULL),
SupervisorsCoalesced AS (
         -- Coalescing the table Supervisors above ...),
SELECT   FirstName + ' ' + LastName AS EmployeeName,
         FORMAT(SUM(SalesAmount), '$###,##0.00') AS Total,
         FromDate, ToDate
         FROM Sales S, Date D, Employee E, SupervisorsCoalesced U
WHERE    S.OrderDateKey = D.DateKey AND S.EmployeeKey = E.EmployeeKey AND
         E.EmployeeKey = U.EmployeeKey AND
         U.FromDate <= D.Date AND D.Date < U.ToDate
GROUP BY FirstName + ' ' + LastName, FromDate, ToDate
ORDER BY FirstName + ' ' + LastName, FromDate
```

Since the assignment of employees to supervisors varies with time, the temporary table Supervisors computes the periods during which a supervisor was supervising at least one employee. Then, this table is coalesced in table SupervisorsCoalesced. Therefore, these two tables are the result of a temporal projection. Finally, the primary query computes the total sales by an employees during the periods they were supervisors.

Query 6. Compute the total sales amount for products assigned to the categories Beverages or Dairy Products.

First, a temporal dice obtains the sales fact of products in either the Beverages or Diary Products categories, at the time of sales. After that, the total sales can be obtained by keeping only the Product dimension in the diced fact. This query can be written in SQL as follows:

```
WITH BeverageOrDairy AS (
         SELECT      ProductKey, P.FromDate, P.ToDate
         FROM        ProductCategory P, Category C
         WHERE       P.CategoryKey = C.CategoryKey AND
                     (C.CategoryName = 'Beverages' OR
                     C.CategoryName = 'Dairy Products')),
BeverageOrDairyCoalesced AS (
         -- Coalescing the table BeverageOrDairy above ...),
SELECT   ProductName, FORMAT(SUM(SalesAmount), '$###,##0.00') AS Total,
         FromDate, ToDate
FROM     Sales S, Date D, Product P, BeverageOrDairyCoalesced B
WHERE    S.OrderDateKey = D.DateKey AND S.ProductKey = P.ProductKey AND
         P.ProductKey = B.ProductKey AND
         B.FromDate <= D.Date AND D.Date < B.ToDate
GROUP BY ProductName, FromDate, ToDate
ORDER BY ProductName, FromDate
```

Since the assignment of products to categories varies across time, the temporary table BeverageOrDairy computes the periods during which a product was assigned to one of the categories requested. Then, this table is coalesced and stored in table BeverageOrDairyCoalesced. Finally, the main query computes the total sales of the products during the periods they were assigned to one of the categories requested.

RELATED WORK

Although there are many logical and conceptual models in the data warehousing literature, like in (Malinowski & Zimányi, 2006), and formalizations of the semantics of OLAP operations, like in (Kuijpers & Vaisman, 2017), these approaches do not deal with the problem of maintaining the history of content changes in a DW. This section reviews some work on temporal DW. For the sake of space, the discussion only addresses the problem at the DW instance level.

Logical Modelling of Temporal Dimensions

Probably the most well-known approach to tackle changes in dimension attribute values are Kimball's so-called *slowly changing dimensions* (SCDs) (Kimball & Ross, 2013). Kimball initially proposed three types of SCDs that were later extended to seven. Based on the tuple-timestamping approach, well known in traditional temporal database theory (Tansel, et al., 1993), SCDs of type 2 preserve the history of dimension attributes by adding a new row in the dimension table instance, for every change that occurs in the attribute value. If changes are frequent, then the dimension table size may grow very fast, leading to storage and performance concerns, hence calling for other solutions. SCDs of Type 3 keep the last two values of each temporal attribute. Attempting to overcome some drawbacks of these three types, Kimball proposed four additional types, not covered here, since this is beyond the scope of this paper.

Bliujute et al. (Bliujute, Saltenis, Slivinskas, & Jensen, 1998) introduce the *temporal star schema* to deal with the problem of slowly changing dimensions. In this approach, the time dimension is omitted from the traditional star schema. Instead, timestamps are defined for each row in every table of the schema. A single timestamp in a table means that data are event-oriented and represent the

time when an event has occurred. Similarly, a pair of timestamps indicates that the data in the table are state-oriented and denote the period during which a state has persisted. This approach allows performing the date and time calculations directly on the raw facts omitting a join with the time dimension. However, excluding the time dimension makes it a challenge to aggregate data over time and to define information on specific instants (like holidays, weekdays, etc.).

The *temporal starnest schema* (Garani, Adam, & Ventzas, 2016) is based on the nested relational model (Paredaens, De Bra, Gyssens, & Van Gucht, 1989). In the model, starting from the most generic level to the most specific one, the levels of a dimension hierarchy are recursively nested. Each parent level's tuple nests the child member sub-tuples as a value of one of its attributes. In this way, just like in the traditional star schema, each dimension consists of a single table, but the parent-child relations are hard-wired within the data itself. The identifier of the most specific level is used to link a dimension to the fact table. The main drawback of this model is its complex structure, which makes its implementation far from trivial.

Golfarelli and Rizzi (Golfarelli & Rizzi, 2007) address the problem of keeping track of the history of measure values, in particular for flow and stock measurements (Lenz & Shoshani, 1997). The former is taken over an interval and accumulated at the end of that interval, while the latter is taken at a fixed point in time. They propose valid time and bitemporal solutions. In the former, valid time is associated with facts, and in case an update of a measure value occurs, its pre-existing value is replaced with the new one. Thus, a fact table contains the most up-to-date values only. The bitemporal solution manages the measure updates as either a delta or a consolidated solution. The delta solution stores updated measure values as the difference between the new and the pre-existing value. The consolidated solution stores the consolidated value for each updated measure value and a pair of timestamps representing the transaction-time.

Mahlknecht et. al. (Mahlknecht, Dignös, & Kozmina, 2019) propose to model temporal fact data by means of three distinct logical schemas. In these schemas, time periods are represented as: (1) sets of all time points in a given period (called *instant model*), (2) pairs of start and end time points of a period (*period model*), and (3) atomic units that are explicitly stored in a new period dimension (*period* model*) that includes data about days within a period.

Querying Temporal Data Warehouses

Many different temporal query languages for OLAP have been proposed in the research DW community. As an example, TOLAP, standing from Temporal Online Analytical Processing (Mendelzon & Vaisman, 2000), is a Datalog-like query language that supports querying data warehouses with time-varying dimensions and facts. The query language is based on an underlying temporal multidimensional data model in which dimension members are timestamped to maintain their historical context. However, the language syntax is still far from the well-known OLAP operators, i.e., roll-up, drill-down, slice, dice, drill-across. There is a call, then, for practical, SQL-based temporal query languages supporting a relational representation of temporal DWs.

The widespread use of relational databases and the appearance of many temporal query languages in the early 1990s raised the need for a standardized temporal relational query language. As a result, TSQL2 (Snodgrass R. T., 1995) was proposed as an upward compatible temporal extension to the SQL-92. In terms of the underlying data model, TSQL2 introduced a new *period* datetime data type of specifiable range and precision. TSQL supports valid time, transaction time, and user-defined time. Being upward compatible with SQL92, defining a snapshot table is also possible. Additionally, the model allows two types of tables, namely, state and event tables. In a *state* table, each tuple is timestamped with a time element that is the union of periods. On the other hand, in an *event* table, each tuple is timestamped with an instant set. Valid time and transaction time can be combined orthogonally to model bitemporal tables.

The SQL-2011 standard includes temporal features (Kulkarni & Michels, 2012), namely, a period data type, temporal primary keys, temporal referential integrity constraints, temporal predicates,

valid-time (called application-time), transaction-time (called system-time) and bitemporal (called system-application-time) tables, as well as temporal insertions, updates, and deletions. However, temporal operators like temporal coalescing (Zhou, Wang, & Zaniolo, 2006), temporal joins, and temporal aggregation are not included in the standard.

Another challenge in this domain is the efficiency of executing SQL operations on temporal data. The problem is due to complex algorithms for computing joins, e.g., (Bouros & Mamoulis, 2018; Pilourdault, Leroy, & Amer-Yahia, 2016) and aggregates, e.g., (Mahlknecht, Dignös, & Gamper, 2015).

As mentioned above, (Wang, Zaniolo, & Zhou, 2005) show how to support historical and temporal queries without extending the standards with temporal features. The authors argue that lessons learned from temporal database research showed that they had not achieved significant commercial success, which claims for exploring solutions that remain within the SQL standard. The authors also use historical OLAP queries to illustrate their approach.

CONCLUSION AND FUTURE WORK

This paper presented a generalized multidimensional model, capable of representing traditional and time-varying data independently. The proposed model allows multiple facts at different granularities, role-playing dimensions, analyzing the data using multiple and non-strict hierarchies, and reusability of level instances across various hierarchies, that means, the most. The model includes data consistency constraints for both static and time-varying data. Finally, the model comes equipped with a set of OLAP operations, whose semantics is formally defined. Finally, the translation of the model to a relational representation, together with an SQL-based implementation of the OLAP operations, is presented.

In future work, the semantics of the roll-up operation can be extended to allow recursive roll-up and windows-based aggregation. Like dimension members, fact members may also change over time. The multidimensional model can be extended to enable and analyze the time-varying measures as well, not addressed at this stage. Furthermore, the data warehouses evolve in their structure as well; therefore, an utterly generalized model should also be able to capture the schema changes.

REFERENCES

Ahmed, W., Zimányi, E., & Wrembel, R. (2015). *Temporal DataWarehouses: Logical Models and Querying. In Actes des 11es journées francophones sur les Entrepôts de Données et l'Analyse en Ligne*. EDA.

Al-Kateb, M., Ghazal, A., Crolotte, A., Bhashyam, R., Chimanchode, J., & Pakala, S. P. (2013). Temporal query processing in Teradata. *Proc. of Conference on Extending Database Technology Conferences EDBT*, 573-578.

Allen, J. F. (1983). Maintaining Knowledge about Temporal Intervals. *Communications of the ACM*, *26*(11), 832–843. doi:10.1145/182.358434

Bliujute, R., Saltenis, S., Slivinskas, G., & Jensen, C. S. (1998). Systematic Change Management in Dimensional Data Warehousing. Tech. rep., Time Center.

Bouros, P., & Mamoulis, N. (2018). Interval Count Semi-Joins. *Proc. of the International Conference on Extending Database, EDBT*, 425-428.

Garani, G., Adam, G. K., & Ventzas, D. (2016). Temporal data warehouse logical modelling. *International Journal of Data Mining. Modelling and Management*, *8*, 144–159.

Golfarelli, M., & Rizzi, S. (2007). Managing late measurements in data warehouses. *International Journal of Data Warehousing and Mining*, *3*(4), 51–67. doi:10.4018/jdwm.2007100103

Hümmer, W., Lehner, W., Bauer, A., & Schlesinger, L. (2002, 9). A Decathlon in Multidimensional Modeling: Open Issues and Some Solutions. In *Proc. of the International Conference on Data Warehousing and Knowledge Discovery, DaWaK* (pp. 275-285). Springer. doi:10.1007/3-540-46145-0_27

Kimball, R., & Ross, M. (2013). *The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling* (3rd ed.). John Wiley & Sons.

Klug, A. (1982). Equivalence of relational algebra and relational calculus query languages having aggregate functions. *Journal of the Association for Computing Machinery*, *29*(3), 699–717. doi:10.1145/322326.322332

Kuijpers, B., & Vaisman, A. (2017). An algebra for OLAP. *Intelligent Data Analysis*, *21*(5), 1267–1300. doi:10.3233/IDA-163161

Kulkarni, K., & Michels, J.-E. (2012). Temporal features in SQL: 2011. *SIGMOD Record*, *41*(3), 34–43. doi:10.1145/2380776.2380786

Ladkin, P. B. (1986). Time Representation: A Taxonomy of Internal Relations. *Proc. of the National Conference on Artificial Intelligence*, 360-366.

Lenz, H. J., & Shoshani, A. (1997). Summarizability in OLAP and statistical data bases. In *Proc. of theInternational Conference on Scientific and Statistical Database Management, SSDBM* (pp. 132-143). Olympia: IEEE.

Mahlknecht, G., Dignös, A., & Gamper, J. (2015). Efficient Computation of Parsimonious Temporal Aggregation. *Advances in Databases and Information Systems, ADBIS*, 320-333.

Mahlknecht, G., Dignös, A., & Kozmina, N. (2019). Modeling and querying facts with period timestamps in data warehouses. *Applied Mathematics and Computer Science*, *29*(1), 31–49. doi:10.2478/amcs-2019-0003

Malinowski, E., & Zimányi, E. (2006). Hierarchies in a multidimensional model: From conceptual modeling to logical representation. *Data & Knowledge Engineering*, *59*(2), 348–377. doi:10.1016/j.datak.2005.08.003

Malinowski, E., & Zimányi, E. (2008). A conceptual model for temporal data warehouses and its transformation to the ER and the object-relational models. *Data & Knowledge Engineering*, *64*(1), 101–133. doi:10.1016/j.datak.2007.06.020

Mendelzon, A., & Vaisman, A. A. (2000). Temporal Queries in OLAP. *Proceedings of the International Conference on Very Large Data Bases, VLDB*, 242-253.

Paredaens, J., De Bra, P., Gyssens, M., & Van Gucht, D. (1989). The Nested Relational Database Model. In *The Structure of the Relational Database Model* (pp. 177–201). Springer. doi:10.1007/978-3-642-69956-6_7

Pilourdault, J., Leroy, V., & Amer-Yahia, S. (2016). Distributed Evaluation of Top-k Temporal Joins. *Proc. of the International Conference on Management of Data, SIGMOD*, 1027-1039.

Snodgrass, R., & Ahn, I. (1985). A taxonomy of time databases. *SIGMOD Record*, *14*(4), 236–246. doi:10.1145/971699.318921

Snodgrass, R. T. (Ed.). (1995). *The TSQL2 Temporal Query Language*. Kluwer Academic Publishers. doi:10.1007/978-1-4615-2289-8

Snodgrass, R. T. (1999). *Developing Time-Oriented Database Applications in SQL*. Morgan Kaufmann.

Tansel, A. U., Clifford, J., Gadia, S., Jajodia, S., Segev, A., & Snodgrass, R. T. (1993). *Temporal Databases: Theory, Design, and Implementation*. Benjamin-Cummings.

Vaisman, A., & Zimányi, E. (2014). *Data Warehouse Systems: Design and Implementation*. Springer. doi:10.1007/978-3-642-54655-6

Wang, F., Zaniolo, C., & Zhou, X. (2005). Temporal XML? SQL Strikes Back! In *International Symposium on Temporal Representation and Reasoning, TIME* (pp. 47-55). IEEE Computer Society. doi:10.1109/TIME.2005.36

Wang, H., & Zaniolo, C. (2000). Using SQL to Build New Aggregates and Extenders for Object- Relational. *Proc. of International Conference on Very Large Data Bases, VLDB*, 166-175.

Zhou, X., Wang, F., & Zaniolo, C. (2006). *Efficient Temporal Coalescing Query Support in Relational Database Systems. In Proc. of Database and Expert Systems Applications*. DEXA.

Zimányi, E. (2006). Temporal aggregates and temporal universal quantification in standard SQL. *SIGMOD Record*, *35*(2), 16–21. doi:10.1145/1147376.1147379

Esteban Zimányi is a full professor and the director of the Department of Computer & Decision Engineering (CoDE) at Université libre de Bruxelles (ULB), Belgium. He started his studies at the Universidad Autónoma de Centro América, Costa Rica. He received a B.Sc. (1988) and a doctorate (1992) in computer science from the Faculty of Sciences at the ULB. During 1997, he was a visiting researcher at the Database Laboratory of the Ecole Polytechnique Fédérale de Lausanne, Switzerland. He co-authored and co-edited 16 books, and published multiple articles in international journals and conferences. He is coordinator of the BDMA and IT4BI-DC Erasmus Mundus programmes. He was Editor-in-Chief of the Journal on Data Semantics published by Springer until 2020. His current research interests include data warehouses, moving object databases, geographic information systems, and semantic web.

Alejandro Vaisman received a BA degree in Civil Engineering, a BA in Computer Science, and a PhD in Computer Science from UBA, under the supervision of Prof. Alberto Mendelzon, from the University of Toronto, Canada. He has been a post-doctoral researcher at the University of Toronto. He has been an Associate Professor at UBA between 1994 and 2013, and is currently full professor at the Instituto Tecnológico of Buenos Aires (ITBA). He has been Vice-Head of the Computer Science Department at UBA, and chair of the Masters Pro- gram in Data Mining (between 2005 and 2010). He was a visiting researcher at the University of Toronto, Universidad Politecnica de Madrid, University of Hasselt, and Universidad de Chile. His research interests are in the field of databases, particularly in Business Intelligence, OLAP and Data Warehousing, the Semantic Web and Geographic Information Systems. He has authored and co-authored several scientific papers presented at major database conferences and journals.

Robert Wrembel is an associate professor in the Faculty of Computing, at Poznan University of Technology, in Poland. In 2008 he received the post-doctoral degree in computer science (habilitation), specializing in database systems and data warehouses. He has been a deputy dean of the Faculty of Computing, in terms 2008-2012 and 2012-2016. Involved in 8 research and 8 industrial projects in the area of data processing technologies; consultant at software house Rodan Systems (2002-2003); lecturer at Oracle Poland (1998-2005). Currently he leads at his University the Erasmus Mundus Joint Doctorate Programme "Information Technologies for Business Intelligence - Doctoral College" (2013-2020). Robert Wrembel visited numerous research and education centers, including: Universitat Politècnica de Catalunya - BarcelonaTech (Catalunya), Université Lyon 2 (France), Universidad de Costa Rica (Costa Rica), Klagenfurt University (Austria), Loyola University (USA), INRIA Paris-Rocquencourt (France), and Université Paris Dauphine (France). In 2012 he graduated from a 2-months innovation and entrepreneurial programme at Stanford University. In 2013 he has done an internship in a BI company Targit (USA). In 2010 he received the IBM Faculty Award for highly competitive research, in 2011 he was awarded the Medal of the Committee of National Education (from the Minister of National Education), in 2016 - the Silver Medal for Long-lasting Service (from the President of the Republic of Poland), in 2019 - IBM Shared University Research Award, and in 2019 - International Federation for Information Processing (IFIP) Service Award. He is a member of ACM, country representative in IFIP Technical Committee TC 2 - Software: Theory and Practice, and a chair of IFIP Working Group 2.6 (Database). Since Apr 2018 he is a regular editorial board member of the Data & Knowledge Engineering journal.