Temporal Queries in OLAP

Alberto O. Mendelzon
mendel@db.toronto.edu
University of Toronto

Alejandro A. Vaisman
av2n@dc.uba.ar
Universidad de Buenos Aires

Abstract

Commercial OLAP systems usually consider OLAP dimensions as static entities. In practice, dimension updates are often necessary in order to adapt the multidimensional database to changing requirements. We have already defined a taxonomy for these dimension updates in previous works, and a minimal set of operators to perform them. In this paper, we show the need to keep track of the history of the data warehouse. In order to address this problem, we propose a new (temporal) multidimensional model, along with a query language supporting it. We formally define the model, introduce the language by means of examples, and define its syntax and semantics. Finally, we discuss implementation issues, and how a translation into SQL:99, TSQL2 or other SQL-based languages can proceed.

1 Introduction

OLAP (On Line Analytical Processing) has received a lot of attention from the database community in the last few years. As a consequence, several models for OLAP applications have been proposed [Kim96, CT98, Leh98]. In these models, data is organized into *dimensions* and *fact tables* [Kim96]. Dimensions are usually organized as hierarchies, providing a way of defining different levels of data aggregation, a central issue in data analysis. It is a frequent assumption in these works that data in fact tables reflect the dynamic aspect of the data warehouse, while dimension data represent static information. This assumption is often violated in practice [HMV99a, HMV99b]. For example, a dimension like *Store* (in a retail warehouse application) may change as new stores open and close, or a store is reassigned from one region to another; in addition, the structure of the Store hierarchy itself may change as a level of grouping, such as *Region*, is eliminated, or a new one introduced. Since the schema of the fact table or tables is composed of attributes from the dimensions, such changes may trigger schema evolution in the fact tables. We argue that in an envolving scenario like this, OLAP systems need temporal features to keep track of the different states of a data warehouse throughout its lifespan.

There are many real life situations in which these requirements arise. Consider for example an NBA (professional basketball) warehouse where the fact table *Points* has just two dimensions, *Player* and *Time*, and a measure, *Points Scored*. The *Player* dimension hierarchy is structured by grouping players into teams called *Franchises*. Suppose a user wants to know the total number of points scored by the players of the Portland Blazers. This query could be interpreted in two different ways: the user could be asking for the sum of total points ever scored by all players who are currently on the Blazers, or for the sum of the points scored by these same players since they joined the Blazers. For instance, the points scored by Damon Stoudamire (who is currently on the Blazers) while playing for the Toronto Raptors in the 1998-99 season should only be added under the first interpretation. A query language of a standard commercial OLAP system will not be able to distinguish one interpretation from the other. The reason is that state-of-the-art OLAP systems just record the last value of dimensional attributes and give no access to their historic values. In the language we will introduce in this paper, a query for the first interpretation will be expressed as:

$$\texttt{Q(x,SUM(p))} \longleftarrow \texttt{Points(x,p,t)}, \; \texttt{x} \xrightarrow{Now} \texttt{franchise:'Blazers'}.$$

This means: for each player `x`, add up all the points scored by `x` where `x` currently "rolls up" to the Blazer franchise. The query for the second interpretation will

Proceedings of the 26th VLDB Conference, Cairo, Egypt, 2000.

read:

$$\texttt{Q(x,SUM(p))} \longleftarrow \texttt{Points(x,p,t),}$$
$$\texttt{x} \xrightarrow{t} \texttt{franchise:'Blazers'.}$$

Descriptive attributes make queries like *"total number of points scored by Stoudamire while playing for the Toronto Raptors"* easy to express.

$$\texttt{Q(SUM(p))} \longleftarrow \texttt{Points(x,p,t),}$$
$$\texttt{x} \xrightarrow{t} \texttt{franchise:'Raptors',}$$
$$\texttt{x.name} \overset{t}{=} \texttt{'Stoudamire'.}$$

The queries above operate at a high level of abstraction, without requiring low-level knowledge about the database design that underlies the dimensional model.

1.1 Motivating example

We will use the same retail data warehouse throughout the paper, adding dimensions or fact tables when it becomes necessary. For the remainder of this section we will consider dimensions as snapshot relations representing data as of the current time. This is a standard practice in commercial OLAP. Let us start with the following dimensions: $Time, Product, Customer, Salesperson$. Moreover, as dimensions are organized in hierarchies, let us also assume the hierarchy $\{itemId \rightarrow itemType, itemId \rightarrow brand\}$; and the following rollup functions from *itemId* to *itemType* : $\{(i_1,t_1),(i_2,t_1),(i_3,t_2),(i_4,t_2)\}$ (we will not be using *brand* at this time). The following fact table represents sales facts.

timeId	spId	customerId	itemId	salesAmount
d_1	s_1	c_1	i_1	100
d_2	s_2	c_2	i_1	100
d_3	s_1	c_3	i_3	100
d_4	s_2	c_4	i_4	100

A query asking for the *total sales per salesperson and product type* would return the following table:

spId	itemType	salesAmount
s_1	t_1	100
s_2	t_1	100
s_1	t_2	100
s_2	t_2	100

Suppose now that at an instant immediately after $d4$, product i_1 is reassigned type t_2. A non-temporal star or snowflake schema will store $< i_1, t_2 >$, replacing the tuple $< i_1, t_1 >$, i.e., there will be no memory of the former description of an item. If the user poses the same query, as all the sales occurred before the revision, she would expect to get the same result. However, she gets the following:

spId	itemType	salesAmount
s_1	t_2	200
s_2	t_2	200

What happened is that the contribution of items of type t_1 is ignored, because now all items are of type t_2.

Notice that in order to issue the query above, the user needs to know the schema of the data warehouse, that is, what are the attributes in the fact and dimension tables. This schema may change over time. For instance, *itemId* may not always have been an attribute of the fact and/or dimension tables, if in the early days of this data warehouse data with granularity *itemId* was not available at the sources. In this case, the query above, which is ignorant of this schema change, will only consider total sales made since the time at which *itemId* was added to the fact table, although information is available to obtain the total sales over the whole lifespan of the data warehouse. All these situations must be handled ad-hoc by current OLAP systems, which have no built-in temporal capabilities.

1.2 Related Work

The problem of handling "slowly changing dimensions" was mentioned by Kimball [Kim96], who suggested some partial solutions (which neither take schema evolution into account, nor consider complex dimension updates). Based on this proposal, a *temporal star schema* was introduced [BSSJ98]. This work compares two different temporal implementations against the usual non-temporal star schema, and constitutes a first step toward recognizing the problem. More recently, a multidimensional model for handling complex data has been introduced [PJ99], where the temporal aspect is considered as a modeling issue, and is addressed in conjunction with another data modeling problems. None of these works propose a data warehouse evolution framework or a temporal query language for OLAP. Recent works on maintenance of temporal views [YW98, YW00] present a view definition language operating over non-temporal data sources, along with techniques for maintaining temporal views. Although dealing with temporal databases, these works are orthogonal to ours, as they focus on the data sources and on how a set of temporal views are obtained and maintained, while we focus on querying a temporal multidimensional database.

1.3 Our approach

In light of the above, we introduce a *temporal multidimensional data model* and a temporal query language supporting it, which we called *TOLAP* (*Temporal OLAP*.) TOLAP combines some of the temporal features of query languages like TSQL2 or SQL/TP [Sno95, Tom97] with some of the high-order features of languages like HiLog or SchemaLog [CKW89, LSS97], in the OLAP setting. We introduce *TOLAP* by means of examples, formally define its syntax and semantics, and discuss its expressive power. We show that *TOLAP* allows queries like (a) *"List the amount of sales by product type, using the categorization each item had at the time it was sold"*; (b) *"Were products categorized by brands two years ago?"*; or (c) *"How were customers classified three years ago?"*. Notice that the last two queries are performed over the evolving dimension's metadata. We also introduce an extension to *TOLAP*, allowing queries which *TOLAP* cannot express, like: *"list the total sales per item and region, using only the currently existing regions"*. Finally, we discuss different possible implementations, involving the translation of a *TOLAP* program to SQL.

1.4 Discussion

One might argue, at first sight, that a generic temporal query language like TSQL2 [Sno95] could be used instead of defining a special-purpose one like *TOLAP*. There are two reasons why we prefer to introduce a new language. First, a language designed specifically for the multidimensional model makes typical OLAP queries much more concise and elegant. In a generic language, queries would have to be laboriously encoded using detailed knowledge of the low-level relational structures used to encode the dimensional data. Second, the best-known temporal languages, such as TSQL2, support only a minimal level of schema versioning ([Sno95] p.29).

Another alternative would have been to add temporal features to other languages with schema management features, such as HiLog [CKW89] or SchemaLog [LSS97]. Again, using a language specifically designed for OLAP yields much simpler syntax and semantics, and just the high-order features that are needed to support schema evolution.

The remainder of the paper is organized as follows: in Section 2 we introduce the data model. In Section 3 we define the query language. We discuss its implementation alternatives in Section 4, and conclude in Section 5.

2 Temporal Multidimensional Model

In previous works [HMV99a, HMV99b], we introduced a multidimensional model supporting dimension updates. In that model, dimensions were non-temporal structures, like the ones of Section 1.1. In the *Temporal Multidimensional Model* we introduce here, dimension elements are timestamped at the schema or instance level (or both) in order to keep track of the updates that occur during the dimension's lifespan.

In the rest of the paper we will be dealing with what in temporal databases is called *valid* time [Sno95], that is, the timestamp represents the time when the fact recorded became valid, rather than the time when it was recorded (*transaction time*). The concepts presented here could be easily extended to handle transaction time too. We will consider time as discrete; that is, a point in the time-line, called a *time point*, will correspond to an integer.

2.1 Temporal Dimensions

The following sets are defined : a set of level names $\mathbf{L}$, where each level $l \in \mathbf{L}$ is associated with a set of values $dom(l)$; a set of attribute names $\mathbf{A}$, such that each attribute $a \in \mathbf{A}$ is associated with a set of values $dom(a)$; a set of temporal dimension names $\mathbf{TD}$; and a set of fact table names $\mathbf{F}$. We assume instant t_0 to be the dimension's creation instant.

Definition 1 (Temporal Dimension Schema) *A* temporal dimension schema *is a tuple (dname,L,λ,$\preceq$,A,$\gg$,μ) where: (a) dname $\in$ $\mathbf{TD}$ is the name of the temporal dimension; (b) μ is a level in the* Time *dimension. Intuitively, μ defines the granularity of the dimension* dname; *(c) $L \subseteq \mathbf{L}$ is a finite set of levels, which contains a distinguished level name* All, *s.t.* $dom(\text{All}) = \{\text{all}\}$, *where* $\{\text{all}\}$ *is considered valid during the complete lifespan of the dimension; (d) λ is a function with signature $dom(\mu) \to \mathbf{L}$, defining the instants when each level was part of the dimension; (e) $\preceq$ is a function with signature $dom(\mu) \to 2^{\mathbf{L}\times\mathbf{L}}$, such that for each $t \in dom(\mu)$, $\preceq_t$ is a relation s.t. $\preceq_t^*$, the transitive and reflexive closure of $\preceq_t$ is a partial order, with a unique bottom level, $l_{inf} \in \lambda(t)$, and a unique top level,* All, *where, for every level $l \in \lambda(t)$, $l_{inf} \preceq_t^* l$ and $l \preceq_t^*$* All *hold; (f)* A *is a finite set of attributes; (g)$\gg$ is a function with signature $dom(\mu) \times \mathbf{A} \to \mathbf{L}$, s.t. for every level $l \in \lambda(t)$,* a $\gg_t l$ *means that if the function is applied to an attribute* a, *it returns the level l, where attribute* a *belongs or belonged to level l at time t.*

Notice that l_{inf} is unique at any time instant t, although it may not be unique across time.

Note In the rest of the paper we will use the retail data warehouse introduced in Section 1.1, with temporal dimensions instead of dimension snapshots.

Example 1 *Let us add a dimension* Store *to our data warehouse, s.t. dname = Store, μ = month, $L = \{storeId, city, region, All\}, \lambda(t) = L, \forall t \geq t_0$, and $\preceq_t$ = $\{StoreId \preceq_t city, city \preceq_t region, \quad region \preceq_t All\}, \forall t \geq t_0$. Also, let us suppose a new level*

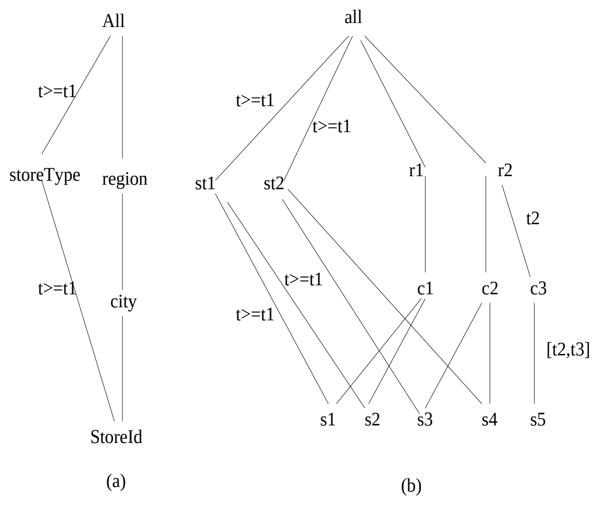


Figure 1: (a) Schema (b) Instance.

storeType is inserted above storeId, at time t_1. *Thus, the following holds:* $\{storeId \preceq_t StoreType_t$, $storeType \preceq_t All\}, \forall t \geq t_1$. *See figure 1(a)* [1].

Definition 2 (Temporal Dimension Instance) *A* temporal dimension instance *is a tuple (D,TRUP, TDESC), where D is a temporal dimension schema, and:*

- *TRUP (temporal rollup) is a set of functions, satisfying the following conditions: (a) for every instant* $t \in dom(\mu)$, *and for each pair of levels* $l_1, l_2 \in \lambda(t)$ *such that* $l_1 \preceq_t l_2$, *there exists in TRUP a rollup function* $\rho[t]_{l_1}^{l_2} : dom(l_1) \to dom(l_2)$; *thus, a function is defined for every snapshot taken at any instant* $t \in dom(\mu)$; *(b) for every instant t in the dimension's lifespan, and for every pair of paths in the graph with nodes in* $\lambda(t)$ *and edges in* $\preceq_t$, $\tau_1 =< l_1, l_2, \ldots, l_k, l_n >$, *and* $\tau_2 =< l_1, l'_2, \ldots, l'_k, l_n >$, *we have* $\rho[t]_{l_1}^{l_2} \circ \ldots \circ \rho[t]_{l_k}^{l_n} = \rho[t]_{l_1}^{l'_2} \circ \ldots \circ \rho[t]_{l'_k}^{l_n}$; *(c) at every instant t of the dimension lifespan, and for each triple of levels* $l_1, l_2, l_3 \in \lambda(t)$ *such that* $l_1 \preceq_t l_2$ *and* $l_2 \preceq_t l_3$,$ran(\rho[t]_{l_1}^{l_2}) \subseteq dom(\rho[t]_{l_2}^{l_3})$.
- *TDESC (temporal description) is a set of partial functions* $\gg$ *with signature* $dom(l) \to dom(a)$, *for each level* $l \in \lambda(t)$ *and attribute a s.t.* $a \gg_t l$.

We call condition (b) in the definition of temporal rollup, *snapshot consistency*, meaning that each element in each level must reach some element in every level above it in the hierarchy, and if there are different paths from one level to another, composing the rollup functions along the different paths must produce the same function.

[1]We will not be using level *storeType* in the following sections.

Example 2 *Figure 1(b) shows a temporal dimension instance for dimension* Store. *The rollup functions with no label are valid for the whole lifespan of Store, while* $\rho_{storeId}^{city}[t](s_5) = c_3, \forall t, t_2 \leq t \leq t_3$. *Rollup functions with label* $t \geq t_1$ *suggest that level storeType was created at time* t_1.

Clearly, the definitions we gave above are not applicable to the *Time* dimension. Thus, we will treat it in the usual way.

Definition 3 (Temporal Fact Table) *A* temporal fact table schema *is a tuple* $s = (fname, f, m, \mu)$, *where m is a level name, called the* measure *of the fact table,* μ *is a level in the* Time *dimension, and f is a function with signature* $dom(\mu) \to 2^{\mathbf{L}}$.

Given a temporal fact table schema $(fname, f, m, \mu)$, *a set of levels L in the range of f, and a level* μ *in the* Time *dimension, a mapping from each level* $l_i \in (L \cup \{\mu\})$ *to* $dom(l_i)$ *is called a* point.

Given a temporal fact table schema $s = (fname, f, m, \mu)$, *a* temporal fact table instance *over it is a partial function which maps points of s to elements in* $dom(m)$.

Definition 4 (Base Fact Table) *Given a set* $\mathbf{D}$ *of temporal dimensions, a* base fact table *is a fact table with schema* $(fname, f_D, m, \mu)$, *such that for each* $t \in dom(\mu)$, *every level in* $f_D(t)$ *is a bottom level of its dimension. Thus, a base fact table is a fact table such that its attributes are the bottom levels of each one of the dimensions in* $\mathbf{D}$.

Definition 5 (Multidimensional Database)

A temporal multidimensional database schema, *denoted* $\mathbf{B_s}$, *is a pair* $(\mathbf{D_s}, \mathbf{F_s})$, *where* $\mathbf{D_s}$ *is a set of temporal dimension schemas, and* $\mathbf{F_s}$ *is a set of temporal fact table schemas. A* temporal multidimensional database instance $I(\mathbf{B})$, *is a tuple* $(\mathbf{F_I}, \mathbf{D_I})$, *where* $\mathbf{D_I}$ *and* $\mathbf{F_I}$ *are dimension and fact table instances, respectively, defined as above.*

2.2 Temporal OLAP queries

Usually, in an OLAP environment, queries require the computation of aggregates over base fact tables. Moreover, in order to obtain good performance, some systems pre-compute aggregates over different groups of attributes. In the presence of dimensions with hierarchies of levels, queries computing aggregates over various dimension levels are often required. As we claimed that dimensions change over time, this must be taken into account, in order to give the user the desired answer to a query she poses to the system. We will denote by *temporal OLAP query,* a query over a set of temporal dimensions and fact tables. The example below will show the difference between a temporal OLAP query and a non-temporal one.

Example 3 *Consider a set of dimensions* $\mathbf{D} = \{Product, Store\}$ *from our retail data warehouse, and a base fact table with schema* $(Sales, f, sales, day)$. *Assume no schema update occurred (this will be studied in Section 3.2). Thus,* f *maps each instant to* $\{itemId, storeId\}$. *The instance of dimension Store is the one of figure 1, and for the instance of dimension* Product *we have* $\rho_{itemId}^{itemType}[t] = \{i_2 \rightarrow t_1, i_3 \rightarrow t_2\}, \forall t, t \geq d_0$; $\rho_{itemId}^{itemType}[t] = \{i_1 \rightarrow t_1\}, \forall t, d_0 \leq t \leq d_4$, *and* $\rho_{itemId}^{itemType}[t] = \{i_1 \rightarrow t_2\}, \forall t, t \geq d_5$*(a reclassification occurred at day* d_5*). Assume that in* Time *we have:* $\rho_{day}^{week} = \{d_1 \rightarrow w_1, d_2 \rightarrow w_1, d_3 \rightarrow w_2, d_4 \rightarrow w_2, d_5 \rightarrow w_2\}$*(and the rollups from week to All). Finally, we have the following instance for the* Sales *fact table (*Day *is displayed for the sake of clarity, but could have been omitted, like in TSQL2):*

itemId	storeId	day	sales
i_1	s_1	d_1	600
i_2	s_2	d_1	100
i_2	s_1	d_2	100
i_3	s_2	d_2	100
i_3	s_3	d_3	100
i_3	s_4	d_4	100
i_1	s_1	d_5	100

Let us now suppose we are given the query: " list the weekly total sum of sales, by city and item type". As in the example of Section 1, two interpretations could be given to this query. The first one, possibly the most usual one, would expect to get the sum of sales considering the type an item had when it was sold. In this case, for instance, item i_1 *would contribute to the aggregation in the following way: the first three tuples, with a total of 800, will add to the group* $\{t_1, c_1, w_1\}$, *while the last one will contribute to* $\{t_2, c_1, w_2\}$. *The result will be given by the following table:*

itemType	city	week	sales
t_1	c_1	w_1	800
t_2	c_1	w_1	100
t_2	c_2	w_2	200
t_2	c_1	w_2	100

The second interpretation, which is the **only** *one supported by non-temporal systems, would ask for the sum of the sales, considering that each sold item has the* current *type, regardless of the time the sale occurred. The result a user would get under this interpretation is given by the table below, which was computed in the following way: the rollup function for every occurrence of item* i_1 *is set to:* $\rho_{itemId}^{itemType}(i_1) = t_2$. *Thus, all the* i_1 *tuples will contribute to type* t_2. *For instance, the first tuple will now contribute to the group* $\{t_2, c_1, w_1\}$. *In the next Section we present a language that lets the user distinguish the two interpretations.*

itemType	city	week	sales
t_1	c_1	w_1	200
t_2	c_1	w_1	700
t_2	c_2	w_2	200
t_2	c_1	w_2	100

3 *TOLAP*: A Temporal Multidimensional Query Language

In this section we propose our multidimensional query language *TOLAP* (standing for Temporal OLAP). We introduce it first by means of examples, and then define its syntax and semantics.

3.1 TOLAP By Example

Let us consider again the set of dimensions $\mathbf{D} = \{Product, Store\}$ from our running example, and the corresponding base fact table named *Sales*, of Example 3. In the *Product* dimension, $\mu = day$. Also assume that there is a fact table $(Price, f_D, price, \mu : month)$, containing the price of each item each month($f_D(t) = \{itemId\}$ for each t).

3.1.1 Simple queries

We begin with queries not involving aggregates.

Example 4 *A query returning the sales for stores in Buenos Aires, on a daily basis, will be expressed in* TOLAP *as:*

$$\texttt{BASales(p,s,m,t)} \longleftarrow \texttt{Sales(p,s,m,t)}, \; \texttt{s} \xrightarrow{t_1} \texttt{city:'BA'}, \; \texttt{t} \longrightarrow \texttt{month:}t_1.$$

In TOLAP, *the query above returns the tuples in* Sales *such that* s *rolls up to* BA, *where* s *represents an element in the lowest level of the dimension* Store. *This query is expressed in a* point-based fashion *(see Section 3.4 for details).*

We assume a fixed ordering of the attributes in the base fact tables. For instance, in the base fact table *Sales*, the first position from the left will always correspond to dimension *Product.*

3.1.2 Queries with aggregates

In order to address queries involving aggregation, we adapt non-recursive Datalog with aggregate functions [CM90], which, in turn, was based on the approach of Klug's relational calculus with aggregates [Klu82].

Example 5 *Consider the query : "list the total sales per item, region and week," where we want aggregates to be computed using temporally consistent values (i.e., a sale in a given store must be credited to the region that corresponded to that store at the time of the sale).*

$$\begin{aligned}\texttt{WS(it,re,w,SUM(m))} \quad \longleftarrow \quad & \texttt{Sales(it,st,m,d),} \\ & \texttt{st} \xrightarrow{mo} \texttt{region:re,} \\ & \texttt{d} \longrightarrow \texttt{month:mo,} \\ & \texttt{d} \longrightarrow \texttt{week:w.}\end{aligned}$$

Note that although in Example 5 we made explicit the rollup between the time granularity of the *Sales* and *Store* dimension, (i.e. *day* and *month*), this could be easily avoided, allowing a limited form of "schema independence", as in Schemalog or SchemaSQL. Later examples show the use of variables that range over level names, pushing this independence farther.

Example 6 *We now introduce descriptive attributes of dimension levels. Suppose we want the total sales by store and brand, for stores with more than ninety employees. Assume that level* storeId *is described by an attribute* nbrEmp.

$$\begin{aligned}\texttt{SB(br,st,SUM(m))} \quad \longleftarrow \quad & \texttt{Sales(i,s,m,t),} \\ & \texttt{i} \xrightarrow{t} \texttt{brand:br,} \\ & \texttt{t} \longrightarrow \texttt{month:mo,} \\ & \texttt{s} \xrightarrow{mo} \texttt{storeId:st,} \\ & \texttt{s.nbrEmp} \geq \texttt{90.}\end{aligned}$$

3.1.3 Metaqueries

We would also like to query the system about the rollup functions themselves, regardless of the facts. Some examples of these kinds of queries are:

- *"Give me the time instants at which store s_1 belonged to the Southern region",* expressed as:

  $$\texttt{StoreTime(t)} \quad \longleftarrow \quad \texttt{Store:storeId:'}s_1\texttt{'} \xrightarrow{t} \texttt{region:'Southern'.}$$

  Note that we must specify the name of the dimension in the atom `Store:storeId:'`s_1`'`, because there is no fact table in the body of the rule to bind `s`.

- *"Were products categorized by brands two years ago?"* (this is the query of example (b) of Subsection 1.3).

  $$\texttt{ProdBrand()} \longleftarrow \texttt{Product:X:x} \xrightarrow{1/1/98} \texttt{brand:y.}$$

  In this example, X is a variable over level names. The expression above means that if any element, in any level in the *Product* dimension rolled up to an element in level *brand* at the required date, the answer to the query will be '*yes*'.

3.2 Data Warehouse Evolution in *TOLAP*

According to the definitions of Section 2, our model supports evolution of the schema over time (in temporal database terminology, this is called *schema versioning* or *schema evolution.* For instance, suppose in our running example that the bottom level of the *Store* dimension in Figure 1 was, initially *city*, and that, at time d_5, *storeId* was inserted below it. A fact table with attributes *itemId,city, sales,day* is in effect before d_5. After the update, the fact table attributes will be: *itemId,storeId, sales,* and *day.* In *TOLAP*, if an element was not defined at a given instant, it will not contribute to the result. For instance, given the query *"list the total sum of sales by brand and storeId",* we have:

$$\begin{aligned}\texttt{SB(br,st,SUM(m))} \quad \longleftarrow \quad & \texttt{Sales(i,s,m,t),} \\ & \texttt{i} \xrightarrow{t} \texttt{brand:br,} \\ & \texttt{t} \longrightarrow \texttt{month:mo,} \\ & \texttt{s} \xrightarrow{mo} \texttt{storeId:st.}\end{aligned}$$

The expression $\texttt{s}\xrightarrow{mo}\texttt{storeId:st}$ means that if an element in any level, which was once a component of a base fact table, rolled up to level *storeId* at time *mo*, it contributes to the aggregation. Thus, the sales made before the month corresponding to d_5 will not contribute to the aggregation in the head (condition $\texttt{s}\xrightarrow{mo}\texttt{storeId:st}$ will not be satisfied). Analogously, a query like *"total sales by store and itemId"* would return exactly the instance of the second fact table above.

Finally, suppose that at time d_9, level *brand* is deleted from the dimension *Product.* The remaining levels would be *itemId*,(bottom level), *itemType*, *company*, and *All*. The query *"total sales by brand and region"* would read in *TOLAP*:

$$\begin{aligned}\texttt{BR(br,reg,SUM(m))} \quad \longleftarrow \quad & \texttt{Sales(i,s,m,t),} \\ & \texttt{i} \xrightarrow{t} \texttt{brand:br,} \\ & \texttt{t} \longrightarrow \texttt{month:mo,} \\ & \texttt{s} \xrightarrow{mo} \texttt{region:reg.}\end{aligned}$$

Any sale taking after d_9 will not be considered, as *brand* is not a level of the dimension any more.

3.3 Syntax

In this section we will formally define the syntax of a *TOLAP* rule. We will first give some definitions which will be used below, and then formalize the concepts introduced in the previous section.

3.3.1 Preliminary definitions

Given a set T, and a discrete linear order $<$, with no endpoints, we define a *point based temporal domain* as the structure $T_P = (T, <)$. Analogously, given $T_P = (T, <)$, we define the set $I(T) = \{(a, b) | a \leq b, a, b \in T \cup \{-\infty, +\infty\}\}$, and let us denote as θ as the set of the usual interval comparison operators. Then, $T_I = (I(T), \theta)$ is an *Interval-based Temporal Domain* corresponding to T_P. These domains will are denoted *Temporal Domains*, and allow us to define the *abstract* and *concrete* rollup functions [Tom97]. We will define the rollup functions over T_P.

3.3.2 Atoms, Terms, Rules, and Programs

Assume $\mathbf{B_s}(\mathbf{D_s}, \mathbf{F_s})$, and $I(\mathbf{B})$ are a multidimensional database schema and instance, respectively, as defined in Section 2. Let $\mathbf{V_L}$ and $\mathbf{V_D}$ be a set of level and data variables, respectively. We have also the sets $\mathbf{C_L}$ and $\mathbf{C_D}$ of level and data constants, respectively. Let $\mathbf{P}$ be a set of intensional and extensional predicate symbols, and $\mathbf{F_F}$ is a set of aggregate function names.

Definition 6 (Terms) *(a) A* data term *is either a variable in* $\mathbf{V_D}$ *or a constant in* $\mathbf{C_D}$; *(b) a* rollup term *is an expression of the form* `d:X:x`, `X:x` *or* `x`, *where* `X` *is a level name variable in* $\mathbf{V_L}$ *or constant in* $\mathbf{C_L}$, `x` *is a data term, and* `d` *is a constant in* $\mathbf{C_L}$; *(c) a* descriptive term *is an expression of the form* `x.a` *where* `x` *and* `a` *are data terms (d) an* aggregate term *is an expression of the form* `f(d)` *s.t.* `f` *is a function name in* $\mathbf{F_F}$. *A* term *is a* data, rollup, descriptive *or* aggregate *term.*

Definition 7 (Atoms) *(a) A* fact atom *is an expression of the form* $\mathtt{F(X_1, \ldots, X_n, M, t)}$, *where* `F` *is a fact table in* $\mathbf{F_s}$, *and* $\mathtt{X_1, \ldots, X_n}$, `M` *and* `t` *are data terms; a* rollup atom *is an expression of the form* $\mathtt{X} \xrightarrow{t} \mathtt{Y}$, *or* $\mathtt{X} \longrightarrow \mathtt{Y}$, *where* `X` *and* `Y` *are rollup terms, and t is a data term; (c) a* descriptive atom *is an expression of the form* $\mathtt{x} \overset{t}{=} \mathtt{y}$, *where* `x` *is a descriptive term, and* `y` *and t are data terms; (d) an* aggregate atom *is of the form* $\mathtt{Q(R, \ldots, Z)}$ *s.t.* $\mathtt{Q} \in \mathbf{P}$, *and* `R,...,Z` *are data terms s.t. at least one is an aggregate term; (e) an expression* $\mathtt{t_1}\ \theta\ \mathtt{t_2}$, *where* $\mathtt{t_1}$ *and* $\mathtt{t_2}$ *are data terms, and* θ *is one of* $\{<, =\}$, *is a* constraint atom; *(f) if* $\mathtt{g} : \mathtt{N} \times .. \times \mathtt{N} \to \mathtt{N}$ *is a scalar function,* $\mathtt{g(n_1, \ldots n_m)}$, *where* $\mathtt{n_i}$ *are data terms, is a* scalar atom; *(g) an* intensional(extensional) atom *is an expression of the form* `p (X,..,Z)` *where* `X,Y,Z` *are data terms, and* `p` *is an intensional(extensional) predicate symbol.*

An atom *is a fact, rollup, descriptive, aggregate, constraint, scalar, intensional or extensional atom. An expression* $\neg\mathtt{t_1}$, *where* $\mathtt{t_1}$ *is an atom, is a* negated atom.

Definition 8 *(TOLAP rules) A* TOLAP-rule *is a formula of the form* $\mathtt{A} \longleftarrow \mathtt{A_1, A_2, \ldots A_n}$, *where A is an intensional(possibly aggregate) positive atom, and* $\mathtt{A_i}$, $\mathtt{i = 1 \ldots n}$ *are non-aggregate atoms. A TOLAP rule* Γ *satisfies the following conditions: (a) If a variable appears in the head of the rule, it must also appear in its body; (b) every position in a fact atom corresponds to the same dimension, with the rightmost position corresponding allways to the Time dimension. (c) for every level/data variable v, all the rollup terms where v appears are associated to the same dimension; (d) if there is an aggregate atom* $\mathtt{Q(a_1, \ldots, a_n)}$ *in the head of the rule, for all atoms in the body, of the form* $\mathtt{d{:}X{:}x} \xrightarrow{t} \mathtt{y : a_i}$, $\mathtt{d{:}X{:}x} \longrightarrow \mathtt{y : a_i}$, *or* $\mathtt{x.y} \overset{t}{=} \mathtt{a_i}$, `y` *is a constant data term; (e) if* `x.a` *is in the body of* Γ, *at least one rollup term in the body is of the form* `d:X:x`, `X:x` *or* `x`; *(f) every variable which appears in a negated, constraint or predicate atom in the body of a rule, must also appear in a positive rollup or fact atom. A* TOLAP Program *is a finite set of* TOLAP-rules.

From the rules above, it follows that there is a function, call it `dim`, *that maps each data variable* `x` *to a unique dimension* `dim(x)`, *and each level variable X to a unique dimension* `dim(X)`. *Furthermore, there is a function* `level`, *that maps each instant* `t` *to a unique level* `level(x,t)` *of the dimension* `dim(x)`, *and to a unique level* `level(X,t)` *of the dimension* `dim(X)`.

3.4 Semantics

We will use point-based semantics [Tom95] for the rollup functions. This means, for instance, that in a dimension such as *Store* of our running example, a value of a rollup function, say, $\mathtt{storeId{:}s} \xrightarrow{mo} \mathtt{city{:}c}$, exists for each month.

Let us assume that for each dimension instance we have a pair of relations, call them $\mathcal{R}_\mathcal{D}$ and $\mathcal{D}_\mathcal{D}$, representing the sets *TRUP* and *TDESC* of Definition 2, respectively. The multidimensional database is defined over three different domains: $\mathcal{D}$, N, T_P, where variables ranging over $\mathcal{D}$ belong to an uninterpreted sort, the ones ranging over N belong to an interpreted sort (numeric), and the temporal variables range over T_P, defined in 3.3.1.

A *valuation* θ for a *TOLAP* rule Γ, is a tuple (θ_s, θ_I), where θ_s is called a *schema valuation,* and θ_I is an *instance valuation.* Valuation θ_s maps the level and attribute variables in Γ to level and attribute names in $\mathbf{B_s}$, while θ_I maps domain variables to values in $I(\mathbf{B})$.

Definition 9 *A* schema valuation *for a rule* Γ, *denoted* $\theta_s(\Gamma)$ *maps level and attribute variables in the atoms of* Γ *as follows: (a) given a rollup atom of the form* $\mathtt{d{:}X{:}x} \xrightarrow{t} \mathtt{Y{:}y}$, θ_s *maps* `d` *to a dimension name in* $\mathbf{D}_s$, `t` *to a value* $w \in T_P$, *and* `X` *and* `Y` *to a pair of values* v, u *s.t.* $v \preceq_w^* u$ *holds in* $\mathtt{d} \in \mathbf{D}_s$; *(b) if the rollup atom is of the form* $\mathtt{X{:}x} \xrightarrow{t} \mathtt{Y{:}y}$,, θ_s *maps* `t` *to a value* $w \in T_P$, *and* `X` *and* `Y` *to a pair of values* v, u *s.t.* $v \preceq_w^* u$ *holds in* $\mathtt{dim(X)} \in \mathbf{D}_s$; *(c) if the rollup atom is of the form* $\mathtt{x} \xrightarrow{t} \mathtt{Y{:}y}$,, θ_s *maps* `t` *to a value* $w \in T_P$, *and* `Y` *to a value u s.t.* $\mathtt{level(x,w)} \preceq_w^* u$ *holds in* $\mathtt{dim(x)} \in \mathbf{D}_s$; *(d) for the rollup atoms of the form*

X:x $\longrightarrow$ Y:y, *θ_s maps* X *and* Y *to dimension levels in* dim(X) *s.t. v, u s.t. $v \preceq^* u$ holds in* dim(X)*; (e) given a descriptive atom of the form* x.A $\overset{t}{=}$ y*, θ_s maps* t *to a value $w \in T_P$, and* A *to an attribute name $u \in \mathbf{A}$, s.t. $u \gg_t$* level(x,w) *in* dim(x) $\in \mathbf{D}_s$;

Given a rule schema valuation $\theta_s(\Gamma)$ for a rule Γ, an instance valuation *is a function θ_I s.t.(a) it maps the domain variables* x *and* y *in the rollup atoms defined above, to values in $\mathcal{R}_\mathcal{D}$ over levels defined by θ_s; (b) θ_I maps variable* x *in the descriptive atoms defined as above, to values in $\mathcal{D}_\mathcal{D}$, over levels defined by θ_s; (c) θ_I maps a fact atom $F(x_1, .., x_n, M, t)$ as follows: F is mapped to a fact table name in $\mathbf{F_s}$, the rightmost term* t *in F, to a value $w \in T_P$, each domain variable x_i in F to a value in dom*(level(x_i,w))*, and the data term* M *in F to a value in N.*

A constraint atom x $\{<,=\}$ y *evaluates to true whenever $\theta_I(x)$ $\{<,=\}$ $\theta_I(y)$. A* negated atom *is evaluated using the Close World Assumption. Thus, $\neg($* x $\xrightarrow{t}$ Y:y) *is true if, given a valuation θ s.t. $\theta(x) = u$, $\theta(t) = w$, $\theta(Y) = l$, and $\theta(y) = v$, then it does not exist a rollup in* dim(x) *s.t. $\rho^l_{\texttt{level(x,w)}}[w](u) = v$. Predicate and function symbols are valuated as in standard datalog.*

Let AGG be the set of aggregate functions, with extension $AGG = \{MIN, MAX,\ COUNT, SUM\}$, and r a relation. The *aggregate operation* [CM90] $\gamma_{f\,A(X)}(r)$ is the relation

$\gamma_{f\,A(X)}(r) = \{t : t$ is an XA-tuple,$t[X] \in \pi_X(r), t[A] = f_A(\sigma_{X=t[X]}(r))\}$,

over XA, s.t. $XA \in schema(r), f \in AGG$, and $f_A(r)$ denotes the aggregation of the values in $t[A], t \in r$, using f. Thus, we can now define the semantics of a $TOLAP$ rule Γ of the form $Q(a_1, a_2, \ldots, a_n, AGG(m))$ $\longleftarrow A_1, \ldots, A_m$ as follows: For each level or data variable v_i in the body of Γ, and for a valuation θ of the variables in the rule's body, we have:

$r_\Gamma = \{< \theta(v_1), \ldots, \theta(v_n) > | \theta$ is a valuation of $\Gamma\}$.

Then

$Q = \gamma_{AGG_m(a_1,\ldots,a_n)}(r_\Gamma)$.

3.5 Expressive power

In this section we study, somewhat informally, $TOLAP$'s expressive power. We also define an extension to $TOLAP$ which will let us express queries like the fourth one of Section 1.3, which cannot be expressed in basic $TOLAP$.

3.5.1 What can be expressed in $TOLAP$?

Intuitively, it is not hard to see that $TOLAP$ has at least the power of first-order query languages with aggregation. However, in a sense it goes beyond this class. Note that in our data model, only the direct rollups are stored, and their indirect consequences are left implicit. Thus, to evaluate a rollup atom like d:X:x $\xrightarrow{t}$Y:y, we effectively need to compute the transitive closure of the rollup functions for dimension d. It is well-known that this cannot be done in first-order, even after adding aggregate functions [LW97]. However, as long as the dimension schema is fixed, this computation can be done in first order, because for a fixed schema, the number of joins needed to transitively close the rollup functions is known in advance.

Not only the structure of a dimension is subject to updates. There are common real-life situations in which the instance of a dimension may be modified in a non-trivial fashion. Suppose for instance, a company considers some country as divided into four regions, *north, south, east, west*, in order to assign representatives; at some time, it is decided that the northern region should be divided into two or more, for any given business reason. We call this change a *split.* As another example, several airlines could become a single one as a result of a corporate fusion, a common situation nowadays. We call this action a *merge* of elements.

Let us suppose the query *"total sales per item and region, using only the currently existing regions(or their descendants)"*. This query cannot be expressed in $TOLAP$. To show this, suppose a region r is split into r_1 and r_2. After that, r_1 is merged with another region r_4. In the meantime, maybe some region could have been deleted. With the tools defined so far, we could not find the "descendants" of r . There are two reasons for this: (a) so far, the model does not keep track of splits and merges, which can be solved by adding such information to the model, which we will do shortly; (b) this, is, again, a transitive closure problem, even if the schema remains fixed, so the extended language we define below is in some sense harder to evaluate than the basic one.

3.5.2 Extending $TOLAP$

We extend $TOLAP$ in order to be able to express the class of queries exemplified above. First, we add two predicates to the data model introduced in Section 2: *split*(x, y, L, t) and *merged*(x, y, L, t), with the following meanings: (a) *split*(x, y, L, t) is true if the element x in level L was split at time t, and y is one of the elements resulting from this splitting; (b) *merged*(x, y, L, t) is *true* if element x in level L was merged into element y at time t; these are *event* predicates, in the temporal database sense. The formal meaning of these predicates depends on the specification of the update operators *split* and *merge,* given in [HMV99b].

Using the *split* and *merged* predicates, we add to the syntax of $TOLAP$ defined in Section 3.3.2 a new kind of atom, d : $\mathtt{L}_1$: x $\xrightarrow{\mathtt{t}_1}$ $\mathtt{L}_2(\mathtt{t}_2)$: y. The valuation of this atom proceeds as in Section 3.4. The interpretation is as follows: the atom evaluates to $True$ whenever y is the element in level L_2 in dimension $\mathtt{d}$, to which an element $\mathtt{x}$ in level L_1 rolled up at time t_2, given that $\mathtt{y}$ is a successor(if $t_2 > t_1$) or predecessor (if $t_1 > t_2$) of an element $\mathtt{z}$ in L_2, s.t. $\mathtt{x}$ rolled up to $\mathtt{z}$ at time t_1.

In order to clarify the meaning of the expression $\mathtt{d : L_1 : x \xrightarrow{t_1} L_2(t_2) : y}$ let us explain it in terms of datalog with stratified negation expressions. Let us define a predicate $shift(x, y, L, t)$ as follows:

$$\begin{array}{rcl}
\mathtt{shift(x,y,L,t)} & \longleftarrow & \mathtt{split(x,y,L,t).} \\
\mathtt{shift(x,y,L,t)} & \longleftarrow & \mathtt{merged(x,y,L,t).} \\
\mathtt{shift(x,y,L,t_2)} & \longleftarrow & \mathtt{shift(x,z,L,t_1),} \\
 & & \mathtt{merged(z,y,L,t_2), t_2 > t_1.} \\
\mathtt{shift(x,y,L,t_2)} & \longleftarrow & \mathtt{shift(x,z,L,t_1),} \\
 & & \mathtt{split(z,y,L,t_2), t_2 > t_1.}
\end{array}$$

From predicate *shift* we derive another one, called $shiftPers(x, y, L, t)$, which extends the validity of the *split* or *merge*, to every instant t between updates For instance, if $shift(r, r_1, L, 10)$ and $shift(r_1, r_2, L, 13)$ hold, then, $shiftPers(r, r_1, L, 11)$ and $shiftPers(r, r_1, L, 12)$ also hold. This has been called *Persistence* [BWJ98]. Thus:

$$\begin{array}{rcl}
\mathtt{shiftPers(x,y,L,t)} & \longleftarrow & \mathtt{shift(x,y,L,t).} \\
\mathtt{shiftPers(x,y,L,s(t))} & \longleftarrow & \mathtt{shiftPers(x,y,L,t),} \\
 & & \mathtt{\neg shift(y,y_1,L,s(t)),} \\
 & & \mathtt{y \neq y_1, s(t) \leq} \ Now, \\
 & & \mathtt{\neg deleted(y,L,s(t)).} \\
\mathtt{shiftPers(x,y,L,s(t))} & \longleftarrow & \mathtt{shiftPers(x,y,L,t_1),} \\
 & & \mathtt{deleted(y,L,t_1),} \\
 & & \mathtt{\neg inserted(y,L,t_2),} \\
 & & \mathtt{inserted(Y,L,t),} \\
 & & \mathtt{t_1 < t_2, t_2 < t.}
\end{array}$$

Here, $s(t)$ stands for the successor of t. Predicates $deleted(y, L, t)$ and $inserted(y, L, t)$ represent the deletion of an element y from a level L containing it, or the insertion of an element into a level, respectively, and can be derived from the available data.

Now, we can define the meaning of $L_1 : x \xrightarrow{t_1} L_2(t_2) : y$ by means of the following datalog rules:

$$\begin{array}{rcl}
\mathtt{L_1 : x \xrightarrow{t_1} L_2(t_2) : y} & \longleftarrow & \mathtt{L_1 : x \xrightarrow{t_1} L_2 : y,} \\
 & & \mathtt{L_1 : x \xrightarrow{t_2} L_2 : y.} \\
\mathtt{L_1 : x \xrightarrow{t_1} L_2(t_2) : y} & \longleftarrow & \mathtt{L_1 : x \xrightarrow{t_1} L_2 : z,} \\
 & & \mathtt{L_1 : x \xrightarrow{t_2} L_2 : y,} \\
 & & \mathtt{shiftPers(z,y,L,t_2).} \\
\mathtt{L_1 : x \xrightarrow{t_1} L_2(t_2) : y} & \longleftarrow & \mathtt{L_1 : x \xrightarrow{t_1} L_2 : z,} \\
 & & \mathtt{L_1 : x \xrightarrow{t_2} L_2 : y,} \\
 & & \mathtt{shiftPers(y,z,L,t_2).}
\end{array}$$

We will denote this extension of $TOLAP$ as $TOLAP^+$. The meaning of a $TOLAP^+$ query is analogous to the meaning of a $TOLAP$ one. Now, we can express the query *"total sales per item and region, using only the currently existing regions"*, as:

$$\begin{array}{rcl}
\mathtt{IR(p,r,SUM(s))} & \longleftarrow & \mathtt{Sales(p,st,s,t),} \\
 & & \mathtt{t \rightarrow month{:}mo,} \\
 & & \mathtt{st \xrightarrow{mo} reg(}Now\mathtt{){:}r.}
\end{array}$$

In order to make things more clear, suppose that a sale such as $(p_1, s_1, 30, 10)$ occurred, and also suppose that at instant "1" store s_1 belonged to region r_2 which no longer exists at time "10", because it was split into r_{21} and r_{22}. After a series of updates(including the deletion of r_{21} and r_{22}), store s_1 currently belongs to region r_5. According to the semantics defined above, this sale will not contribute to the query result, because r_5 is not a descendant of r_1. However, if the rollup $\mathtt{st \xrightarrow{mo} reg(}Now\mathtt{){:}r}$ were replaced by $\mathtt{st \xrightarrow{Now} reg{:}r}$, the sale in question would be included in the aggregation. If we wanted this to occur, we must have asked for the *"total sales per item and region, using the current rollups from stores to regions"*, which can also be expressed in $TOLAP$.

4 Implementation

$TOLAP$ could be implemented in at least two different ways: (a) translating $TOLAP$ queries to SQL, or (b) using a temporal query language like TSQL2 [Sno95]. To make these ideas concrete, we use a data structure that encodes the dimensions into one schema relation and a set of instance relations. The schema relation describes all the dimensions in the data warehouse, with structure: $(Dimension, upLevel, loLevel, From, To)$, where $loLevel \preceq_{(From,To)} upLevel$. Although there is one instance relation for each dimension instance, two approaches can be followed: in the first one, the instance relation has the form (*upLevel,loLevel,upVal, loVal,From,To*), where $\rho_{loLevel}^{upLevel}[(From, To)](loVal) = upVal$. In the second approach, which we will call the "denormalized" representation, a dimension instance is stored as relation with a column for each dimension level, plus two columns *From,To*. This relation is updated every time a dimension update occurs, either at schema or instance level. In this way, the transitive closure of an instance is stored in a single tuple, which allows a more appropriate implementation of the rollup functions.

4.1 Translation into SQL.

The first alternative we analize consists in translating $TOLAP$ queries to SQL, using the data structure defined above. The following example will show that even simple $TOLAP$ queries can have non-trival SQL translations.

Suppose that in our running data warehouse, level *itemId* was deleted from the *Product* dimension at a certain time, causing the fact table schema to change. After that update, the query *"total sales by brand, region and month"* is issued . In $TOLAP$ this query would read:

$$\texttt{BRM(b,r,mo,SUM(m))} \longleftarrow \begin{array}{l}\texttt{Sales(it,st,m,t)},\\ \texttt{it} \xrightarrow{t} \texttt{brand:b},\\ \texttt{st} \xrightarrow{mo} \texttt{region:r},\\ \texttt{t} \longrightarrow \texttt{month:mo}.\end{array}$$

This *TOLAP* query will yield the following SQL code:

```
SELECT T.month,P.upVal,S2.upVal,SUM(sales)
FROM Sales_1 S, Products P , Store S1,Store S2,
Time T
WHERE S.itemId = P.loVal AND P.loLevel = 'itemId' AND
  P.upLevel = 'brand' AND S.storeId = S1.loVal
  AND S1.loLevel = 'storeId' AND S1.upLevel='city'
  AND S1.upLevel = S2.loLevel
  AND S1.upVal = S2.loVal AND S2.upLevel = 'region'
  AND T.day = S.day AND P.From <= S.day
  AND S.day <= P.To AND S1.From <= T.month
  AND T.month <= S1.To AND S2.From <= T.month
  AND T.month <= S2.To
GROUP BY T.month,P.upVal,S2,upVal
UNION
SELECT T.month,P.loVal,S2.upVal,SUM(sales)
FROM Sales_2 S, Products P , Store S1, Store
S2, Time T
WHERE S.brand = P.loVal AND P.loLevel = 'brand'   AND
S.storeId = S1.loVal AND S1.loLevel = 'storeId'
  AND S1.upLevel='city' AND S1.upVal = S2.loVal
  AND S1.upLevel = S2.loLevel
  AND S2.upLevel = 'region'
  AND T.day = S.day AND P.From <= S.day
  AND S.day <= P.To AND S1.From <= T.month
  AND T.month <= S1.To AND S2.From <= T.month
  AND T.month <= S2.To
GROUP BY T.month,P.loVal,S2.upVal
```

The consequences of the update in the generated query, are shown in italics in the WHERE clause. The attribute *P.loVal* in the second SELECT clause is needed because, after the update, *brand* became the bottom level of the dimension, being also one of the aggregation levels. Also note that the composition between levels *storeId* and *region* must be preformed explicitly.

Implementing the "denormalized" alternative delivers a better performance, at the expense of a more sophisticated update procedure. Here, instead of the *level,value* pairs, each column stores the corresponding values for the level it represents. Thus, the query above generates the following SQL code:

```
SELECT T.month,P.brand,S1.region,SUM(sales)
FROM Sales_1 S, Products P , Store S1, Time T
WHERE S.itemId = P.itemId
  AND S.storeId = S1.storeId
  AND T.day = S.day AND P.From <= S.day
  AND S.day <= P.To AND S1.From <= T.month
  AND T.month <= S1.To
GROUP BY T.month,P.brand,S1.region
UNION
SELECT T.month,P.brand,S1.region,SUM(sales)
FROM Sales_2 S, Products P , Store S1, , Time T
WHERE S.brand = P.brand
  AND S.storeId = S1.storeId
  AND T.day = S.day AND P.From <= S.day
  AND S.day <= P.To AND S1.From <= T.month
  AND T.month <= S1.To
GROUP BY T.month,P.brand,S1.region
```

4.2 Translation into TSQL2

Translating *TOLAP* into TSQL2 seems to be a natural choice, as we can avoid much of the explicit time manipulation, and the generated queries are simpler than their SQL equivalents. However, the translator must deal with the CAST, VALID, and other TSQL2 clauses. For instance, the first term of the query of Subsection 4.1 will look like:

```
SELECT VALID CAST (VALID(S) AS MONTH),P.upVal,
S2.upVal,SUM(sales)
FROM Sales_1 S, Products P , Stores S1, Stores
S2, Time T
WHERE S.ItemId = P.loVal AND S1.loLevel = 'storeId'
  AND P.loLevel = 'itemId' AND P.upLevel = 'Brand'
  AND S.storeId = S1.loVal AND S1.upVal = S2.loVal
  AND S2.upLevel = 'region' AND S1.upLevel='city'
  AND S1.upLevel = S2.loLevel
GROUP BY VALID(S) USING 1 MONTH ,P.upVal, S2.upVal
```

We are currently implementing *TOLAP* at the University of Buenos Aires. We choose the "denormalized" implementation presented in Subsection 4.1, based on the results of early tests performed over prototypes. We have also developed a visual interface which allows the user to visually browse the schema and instance of the dimensions in a multidimensional database, at any given instant . The screen is split into two windows. On the left one, the user browses the different schemas which the dimension goes through over time. In the right window, a tree-view of the rollup fuctions is displayed, synchronized with what is displayed in the other window. A bar in the lower part of the screen shows the validity intervals for these schemas and instances. We are developing using Java , connecting to an Oracle 8 database via JDBC drivers. The figures included in the Appendix will give the reader a better idea about the interface described above

5 Conclusion

We presented a temporal model for multidimensional OLAP, motivated by the observation that ignoring temporal issues leads to impoverished expressive power and questionable query semantics in many real-life scenarios. Our model supports changes both in the instances and in the structure of OLAP dimensions, supporting schema evolution. We also proposed *TOLAP*, a language that supports the model, allowing the expression of *temporal OLAP queries* in an elegant and intuitive fashion. We studied the expressive power of *TOLAP* and introduced an extension that allows transitive closure queries. Finally, we suggested how to translate a *TOLAP* query to different SQL dialects.

Although we presented *TOLAP* using a ruled-based framework, it is straightforward to translate it to an SQL-style if it is considered more expedient. For instance, the query we used as an example in Section 4 could be written as follows.

```
SELECT br, reg, m,SUM(S.sales)
FROM Sales S, Products P , Store ST, Time T
WHERE P.RUP(S.1,Brand,br)
   AND ST.RUP(S.2,Region,reg)
   AND T.RUP(S.time,Month,m).
```

Acknowledgements

This work was partially supported by the Institute of Robotics and Intelligent Systems, the Natural Sciences and Engineering Research Council, and the University of Buenos Aires.

References

[BSSJ98] R. Bliujute, S. Saltenis, G. Slivinskas, and G. Jensen. Systematic change management in dimensional data warehousing. *Time Center Technical Report TR-23*, 1998.

[BWJ98] R. Bettini, S. Wang, and S. Jajodia. Semantic assumptions and their use in databases. *IEEE Transactions on Knowledge and Data Engineering*, 1998.

[CKW89] W. Chen, M. Kifer, and D. S. Warren. Hilog as a platform for database language. In *Proceedings of the 2nd. International Workshop on Database Programming Languages*, pages 315–329, Oregon Coast,Oregon,USA, 1989.

[CM90] M. Consens and A.O. Mendelzon. Low complexity aggregation in Graphlog and Datalog. In *Proceedings of the 3rd International Conference on Database Theory, Lecture Notes in Computer Science n.470*, pages 379–394, 1990.

[CT98] L. Cabibbo and R. Torlone. A logical approach to multidimensional databases. In *EDBT'98: 6th International Conference on Extending Database Technology*, pages 253–269, Valencia, Spain, 1998.

[HMV99a] C. Hurtado, A.O. Mendelzon, and A. Vaisman. Maintaining data cubes under dimension updates. *Proceedings of IEEE/ICDE'99*, 1999.

[HMV99b] C. Hurtado, A.O. Mendelzon, and A. Vaisman. Updating OLAP dimensions. *Proceedings of ACM DOLAP'99*, 1999.

[Kim96] R. Kimball. *The Data Warehouse Toolkit.* J.Wiley and Sons, Inc, 1996.

[Klu82] A. Klug. Equivalence of relational algebra and relational calculus query languages having aggregate functions. *Journal of ACM, p.699-717*, 1982.

[Leh98] W. Lehner. Modeling large OLAP scenarios. In *Proceedings of the 1998 International Conference on Extending Database Technology*, Valencia, Spain, 1998.

[LSS97] L.V.S Lakshmanan, F. Sadri, and I.N. Subramanian. Logic and algebraic languages for interoperability in multidatabase systems. *Journal of Logic Programming 33(2),pp.101-149*, 1997.

[LW97] L. Libkin and L. Wong. On the power of aggregation in relational query languages. In *Database Programming Languages(DBPL'97)*, pages 270–280, 1997.

[PJ99] T.B Pedersen and C. Jensen. Multidimensional data modeling for complex data. *Proceedings of IEEE/ICDE'99*, 1999.

[Sno95] Richard Snodgrass. *The TSQL2 Temporal Query Language.* Kluwer Academic Publishers, 1995.

[Tom95] D. Toman. Point-based vs. interval-based temporal query languages. In *Proceedings of the ACM - PODS Conference*, 1995.

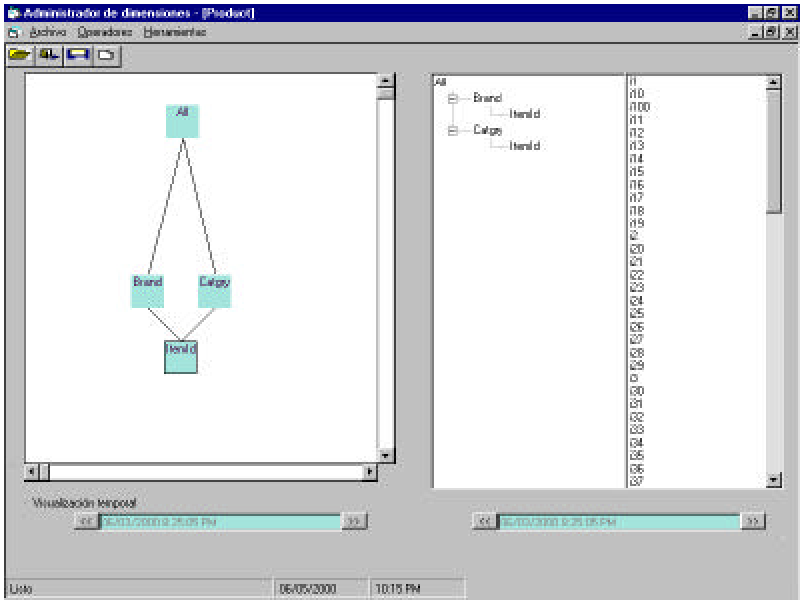
[Tom97] D. Toman. A point-based temporal extension to sql. In *Proceedings of DOOD'97*, Montreaux, Switzerland, 1997.

[YW98] J. Yang and J. Widom. Maintaining temporal views over non-temporal information sources for data warehousing. In *Proceedings of the Sixth International Conference on Extending Database Technology*, Valencia, Spain, 1998.

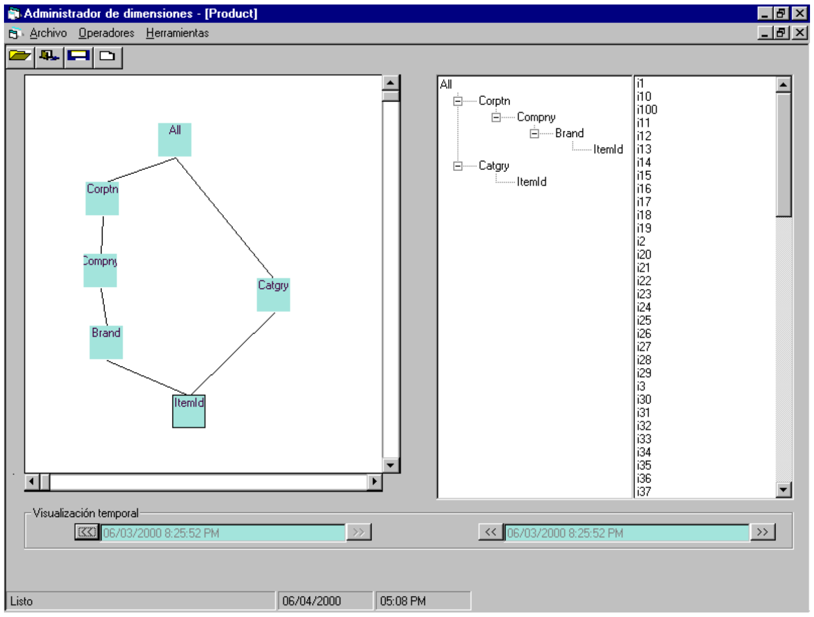
[YW00] J. Yang and J. Widom. Temporal view self-maintenance in a warehousing environment. *To appear in Proceedings of the Seventh International Conference on Extending Database Technology*, 2000.

APPENDIX

Here we show an example of our graphic interface. The left window shows a dimension schema for *Product*, with levels *Brand*, *Category* and *ItemId*. The window on the right displays the rollup functions, and the instance set of the highlighted level.



The figure below displays the dimension after a series of updates. We can see how the instances displayed in the right window change according to the schema. The bars on the lower part of the screen allow browsing through time.


Administrador de dimensiones - [Product]
Archivo Operadores Herramientas
All
Corptn
Compny
Brand
Catgry
ItemId
All
Corptn
Compny
Brand
ItemId
Catgry
ItemId
i1
i10
i100
i11
i12
i13
i14
i15
i16
i17
i18
i19
i2
i20
i21
i22
i23
i24
i25
i26
i27
i28
i29
i3
i30
i31
i32
i33
i34
i35
i36
i37
Visualización temporal
06/03/2000 8:25:52 PM
06/03/2000 8:25:52 PM
Listo
06/04/2000
05:08 PM