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 [Kim98, CT98, Le98]. 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 evolving 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:

\[ Q(x, \text{SUM}(p)) \leftrightarrow \text{Points}(x, p, t), \]

where \( x \rightarrow \text{franchise: 'Blazers'} \).

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
read:

\[ Q(x, \text{SUM}(p)) \leftarrow \text{Points}(x, p, t), \quad x \xrightarrow{f} \text{franchise: 'Blazers'}. \]

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

\[ Q(\text{SUM}(p)) \leftarrow \text{Points}(x, p, t), \quad x \xrightarrow{f} \text{franchise: 'Raptors'}, \quad x \xrightarrow{g} \text{name} \equiv \text{'Stoudamire'}. \]

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 \( \{ \text{itemId} \rightarrow \text{itemType}, \text{itemId} \rightarrow \text{brand} \} \); and the following rollup functions from \( \text{itemId} \) to \( \text{itemType} \) : \( \{(i_1, 1), (i_2, 1), (i_3, 2), (i_4, 2)\} \) (we will not be using \( \text{brand} \) at this time). The following fact table represents sales facts.

<table>
<thead>
<tr>
<th>itemId</th>
<th>spld</th>
<th>customerId</th>
<th>item Id</th>
<th>salesAmount</th>
</tr>
</thead>
<tbody>
<tr>
<td>d_1</td>
<td>s_1</td>
<td>c_1</td>
<td>i_1</td>
<td>100</td>
</tr>
<tr>
<td>d_2</td>
<td>s_2</td>
<td>c_2</td>
<td>i_1</td>
<td>100</td>
</tr>
<tr>
<td>d_3</td>
<td>s_1</td>
<td>c_2</td>
<td>i_2</td>
<td>100</td>
</tr>
<tr>
<td>d_4</td>
<td>s_2</td>
<td>c_4</td>
<td>i_4</td>
<td>100</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>spld</th>
<th>itemType</th>
<th>salesAmount</th>
</tr>
</thead>
<tbody>
<tr>
<td>s_1</td>
<td>t_1</td>
<td>100</td>
</tr>
<tr>
<td>s_2</td>
<td>t_1</td>
<td>100</td>
</tr>
<tr>
<td>s_1</td>
<td>t_2</td>
<td>100</td>
</tr>
<tr>
<td>s_2</td>
<td>t_2</td>
<td>100</td>
</tr>
</tbody>
</table>

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, \( \text{itemId} \) 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 \( \text{itemId} \) 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 \( \text{itemId} \) 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 [Kim06], 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 [BSS98]. 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 call 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 current 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 query 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 $L$, where each level $l \in L$ is associated with a set of values $dom(l)$; a set of attribute names $A$, such that each attribute $a \in A$ is associated with a set of values $dom(a)$; a set of temporal dimension names $TD$; and a set of fact table names $F$. We assume instant $t_0$ to be the dimension’s creation instant.

**Definition 1 (Temporal Dimension Schema)** A temporal dimension schema is a tuple $(dname, L, \lambda, \leq, A, \geq, \mu)$ where: (a) $dname \in TD$ is the name of the temporal dimension; (b) $\mu$ is a level in the Time dimension. Intuitively, $\mu$ defines the granularity of the dimension $dname$; (c) $L \subseteq L$ is a finite set of levels, which contains a distinguished level name All, s.t. $dom(All) = \{\text{all}\}$, where $\{\text{all}\}$ is considered valid during the complete lifespan of the dimension; (d) $\lambda$ is a function with signature $dom(\mu) \rightarrow L$, defining the instants when each level was part of the dimension; (e) $\leq$ is a function with signature $dom(\mu) \rightarrow L^2$, such that for each $t \in dom(\mu)$, $\leq_t$ is a relation s.t. $\leq_t$: the transitive and reflexive closure of $\leq_t$ is a partial order, with a unique bottom level, $l_{inf} \in L(t)$, and a unique top level, All, where, for every level $l \in \lambda(t)$, $l_{inf} \leq_t l$ and $l \leq_t All$ hold; (f) $\geq$ is a function with set of attributes; $(g \geq)$ is a function with signature $dom(\mu) \times A \rightarrow L$, s.t. for every level $l \in \lambda(t)$, $g \geq l$ means that if the function is applied to an attribute a, it returns the level l, where attribute a belongs 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. $dname = Store$, $\mu = month$, $L = \{storeId, city, region, All\}$, $\lambda = \lambda(t)$, $\forall t \geq t_0$, and $\leq_t = \{storeId \leq_t city, city \leq_t region, region \leq_t All\}$, $\forall t \geq t_0$. Also, let us suppose a new level
Definition 2 (Temporal Dimension Instance) A
temporal dimension instance is a tuple (D,TREP, 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 \( t \in \text{dom}(\mu) \), and for each pair of levels \( l_1, l_2 \in \lambda(t) \) such that \( l_1 \preceq_1 l_2 \), there exists in TRUP a rollup function \( \rho[l_1,l_2] : \text{dom}(l_1) \rightarrow \text{dom}(l_2) \); thus, a function is defined for every snapshot taken at any instant \( t \in \text{dom}(\mu) \); (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 \( \preceq_1 \), \( t_1 = < l_1, l_2, \ldots, k, l_3 > \), and \( t_2 = < l_1, l_2, \ldots, l_k, l_3 > \), we have \( \rho[l_1,l_2] \circ \cdots \circ \rho[l_1,l_k] \circ \rho[l_1,l_3] = \rho[l_1,l_3] \circ \cdots \circ \rho[l_1,l_k] \); (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 \( l_1 \preceq_1 l_2 \) and \( l_2 \preceq_1 l_3 \), \( \text{ran}(\rho[l_1,l_2]) \subseteq \text{dom}(\rho[l_1,l_3]) \).

- **TDESC (temporal description)** is a set of partial functions \( \Rightarrow \) with signature \( \text{dom}(l) \rightarrow \text{dom}(a) \), for each level \( l \in \lambda(t) \) and attribute \( a \) s.t. \( a \Rightarrow_1 l \).

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.

---

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 \( \rho_{\text{store}}^{\text{city}}(t) = c_3, \forall t, t_2 \leq t \leq t_3 \). 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 \( s = (f, m, \mu) \), where \( m \) is a level name, called the measure of the fact table, \( \mu \) is a level in the Time dimension, and \( f \) is a function with signature \( \text{dom}(\mu) \rightarrow \mathbb{R}^L \).

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

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

Definition 4 (Base Fact Table) Given a set \( D \) of temporal dimensions, a base fact table is a fact table with schema \( (f, m, \mu) \), such that for each \( t \in \text{dom}(\mu) \), every level in \( L \) 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 \( D \).

Definition 5 (Multidimensional Database) A temporal multidimensional database schema, denoted \( B_s \), 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 \( I(B) \), is a tuple \((\mathbf{F}_1, \mathbf{D}_1)\), where \( \mathbf{D}_1 \) and \( \mathbf{F}_1 \) 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 \( D = \{ \text{Product, Store} \} \) from our retail data warehouse, and a base fact table with schema \( (\text{Sales, f, sales, day}) \). Assume no schema update occurred (this will be studied in Section 3.2). Thus, \( f \) maps each instant to \( \{ \text{itemId, storeId} \} \). The instance of dimension Store is the one of figure 1, and for the instance of dimension Product we have \( \rho_{\text{itemId}} = t_2 \rightarrow t_1, i_3 \rightarrow t_2 \), \( \forall t \geq d_5 \), \( \rho_{\text{itemId}}^{\text{new}} = t_1 \rightarrow t_1 \), \( \forall t \leq d_5 \), and \( \rho_{\text{itemId}}^{\text{new}} = t_1 \rightarrow t_2 \), \( \forall t \geq d_5 \) (a reclassification occurred at day \( d_5 \)). Assume that in Time we have: \( \rho_{\text{itemId}}^{\text{new}} = \{ d_1 \rightarrow w_1, d_2 \rightarrow w_1, d_3 \rightarrow w_2, d_4 \rightarrow w_2, d_5 \rightarrow w_2 \} \) (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):

<table>
<thead>
<tr>
<th>itemId</th>
<th>storeId</th>
<th>day</th>
<th>sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>i_1</td>
<td>s_1</td>
<td>d_1</td>
<td>100</td>
</tr>
<tr>
<td>i_2</td>
<td>s_2</td>
<td>d_2</td>
<td>100</td>
</tr>
<tr>
<td>i_3</td>
<td>s_3</td>
<td>d_3</td>
<td>100</td>
</tr>
<tr>
<td>i_4</td>
<td>s_4</td>
<td>d_4</td>
<td>100</td>
</tr>
<tr>
<td>i_5</td>
<td>s_1</td>
<td>d_5</td>
<td>100</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>itemType</th>
<th>city</th>
<th>week</th>
<th>sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>t_1</td>
<td>c_1</td>
<td>w_1</td>
<td>800</td>
</tr>
<tr>
<td>t_2</td>
<td>c_1</td>
<td>w_1</td>
<td>100</td>
</tr>
<tr>
<td>t_2</td>
<td>c_2</td>
<td>w_2</td>
<td>200</td>
</tr>
<tr>
<td>t_2</td>
<td>c_1</td>
<td>w_2</td>
<td>100</td>
</tr>
</tbody>
</table>

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 \( \rho_{\text{itemId}}^{\text{new}}(i_1) = t_2 \). 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.

<table>
<thead>
<tr>
<th>itemType</th>
<th>city</th>
<th>week</th>
<th>sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>t_1</td>
<td>c_1</td>
<td>w_1</td>
<td>200</td>
</tr>
<tr>
<td>t_2</td>
<td>c_1</td>
<td>w_1</td>
<td>700</td>
</tr>
<tr>
<td>t_2</td>
<td>c_2</td>
<td>w_2</td>
<td>200</td>
</tr>
<tr>
<td>t_2</td>
<td>c_1</td>
<td>w_2</td>
<td>100</td>
</tr>
</tbody>
</table>

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 \( D = \{ \text{Product, Store} \} \) from our running example, and the corresponding base fact table named \( \text{Sales} \), of Example 3. In the Product dimension, \( \mu = \text{day} \). Also assume that there is a fact table \( (\text{Price, f}_D, \text{price}, \mu : \text{month}) \), containing the price of each item each month \( (f_D(t) = \{ \text{itemId} \} \) 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:

\[
\text{BA sales}(p,s,m,t) \leftarrow \text{Sales}(p,s,m,t), \quad s \rightarrow \text{city: 'BA'}, \quad m \rightarrow \text{month: t}.
\]

In TOLAP, the query above returns the tuples in Sales such that \( s \rightarrow \text{city: 'BA'} \), 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 \( \text{Sales} \), the first position from the left will always correspond to dimension \( \text{Product} \).

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).

\[
WS(it, re, w, SUM(m)) \leftarrow Sales(it, st, m, d),
\]

\[
st \rightarrow region: re,
\]

\[
d \rightarrow month: mo,
\]

\[
d \rightarrow week: w.
\]

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.

\[
SB(br, st, SUM(m)) \leftarrow Sales(i, s, m, t),
\]

\[
i \rightarrow brand: br,
\]

\[
t \rightarrow month: mo,
\]

\[
s \rightarrow storeId: st,
\]

\[
s. nbrEmp \geq 90.
\]

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 s₁ belonged to the Southern region”, expressed as:

\[
\text{StoreTime}(t) \leftarrow \text{Store}:\text{storeId}:'s₁',
\]

\[
\text{region}:'\text{Southern}'.
\]

Note that we must specify the name of the dimension in the atom \(\text{Store}:\text{storeId}:'s₁'\), 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).

\[
\text{ProdBrand()} \leftarrow \text{Product}:X \cdot x^{1/1998} \rightarrow \text{brand}: y.
\]

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 \(d₀\), \(storeId\) was inserted below it. A fact table with attributes \(itemId, city, sales, day\) is in effect before \(d₀\). 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:

\[
SB(br, st, SUM(m)) \leftarrow Sales(i, s, m, t),
\]

\[
i \rightarrow brand: br,
\]

\[
t \rightarrow month: mo,
\]

\[
s \rightarrow storeId: st.
\]

The expression \(s \rightarrow storeId: st\) 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 \(d₀\) will not contribute to the aggregation in the head (condition \(s \rightarrow storeId: st\) 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 \(d₀\) 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:

\[
BR(br, reg, SUM(m)) \leftarrow Sales(i, s, m, t),
\]

\[
i \rightarrow brand: br,
\]

\[
t \rightarrow month: mo,
\]

\[
s \rightarrow region: reg.
\]

Any sale taking after \(d₀\) 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 $T_P = (T, \leq)$. Analogously, given $T_P = (T, \leq)$, we define the set $I(T) = \{(a, b)|a \leq b, a, b \in T \cup \{-\infty, +\infty\}\}$, and let us denote as $\theta$ 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 be denoted Temporal Domains, and allow us to define the abstract and concrete rollup functions [Tom07]. We will define the rollup functions over $T_P$.

### 3.3.2 Atoms, Terms, Rules, and Programs

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

**Definition 6 (Terms)** (a) A data term is either a variable in $V_D$ or a constant in $C_D$; (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 $V_L$ or constant in $C_L$, $x$ is a data term, and $d$ is a constant in $C_L$; (c) a descriptive term is an expression of the form $a$, where $x$ is a data term, and $a$ is a descriptive term; (d) a tuple term is an expression of the form $t_1, t_2, \ldots, t_k$, where $t_1, t_2, \ldots, t_k$ are data terms.

**Definition 7 (Atoms)** (a) A fact atom is an expression of the form $F(x_1, \ldots, x_n, \text{rel}, \text{val})$, where $F$ is a fact table in $F_s$, and $x_1, \ldots, x_n$ are data terms; (b) a rollup atom is an expression of the form $x \rightarrow y$, where $x$ and $y$ are rollup terms; (c) a descriptive atom is an expression of the form $x \in y$, where $x$ is a descriptve term, and $y$ and $t$ are data terms; (d) an aggregate atom is of the form $Q(R, \ldots, Z)$ s.t. $Q \in P$, and $R, \ldots, Z$ are data terms s.t. at least one is an aggregate term; (e) an expression $\tau_1 \theta \tau_2$, where $\tau_1$ and $\tau_2$ are data terms, and $\theta$ is one of $\{<, =\}$, is a constraint atom; (f) if $g : N \times N \rightarrow N$ is a scalar function, $g(n_1, \ldots, n_k)$, $n_1$ are data terms, $a$ is a scalar atom; (g) an intensional atom is an expression of the form $p(x, \ldots, Z)$ s.t. $X, Y, Z$ are data terms, and $p$ is an intensional predicate symbol.

An atom is a fact, rollup, descriptive, aggregate, constraint, scalar, intensional or extensional atom. An expression $\neg \tau_1$, where $\tau_1$ is an atom, is a negated atom.

**Definition 8 (TOLAP rules)** A TOLAP rule is a formula of the form $A \leftarrow A_1, A_2, \ldots, A_n$, where $A$ is an intensional(possibly aggregate) positive atom, and $A_i$, $i = 1 \ldots n$ are non-aggregate atoms. A TOLAP rule $\Gamma$ 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 $Q(a_1, \ldots, a_n)$ in the head of the rule, for all atoms in the body, of the form $d : x : x \rightarrow y : a_i$, or $x, y \in a_i$, $y$ is a constant data term; (e) if $x, a$ is in the body of $\Gamma$, at least one rollup term in the body is of the form $d : x, x : x$ or $x$; (f) every variable which appears in a negated, constraint, or aggregate 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 $\text{dim}$, that maps each data variable $x$ to a unique dimension $\text{dim}(x)$, and each level variable $X$ to a unique dimension $\text{dim}(X)$. Furthermore, there is a function $\text{level}$, that maps each constant $\tau$ to a unique level $\text{level}(\tau)$ of the dimension $\text{dim}(x)$, and to a unique level $\text{level}(\tau)$ of the dimension $\text{dim}(X)$.

### 3.4 Semantics

We will use point-based semantics [Tom05] 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, $\text{storeid} : \leq _{\text{city}} \text{c}$, exists for each month.

Let us assume that for each dimension instance we have a pair of relations, call them $R_P$ and $P_P$, representing the sets $\text{TRIP}$ and $\text{DESC}$ of Definition 2, respectively. The multidimensional database is defined over three different domains: $P, N, T_P$, where variables ranging over $T_P$ 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 $\theta$ for a TOLAP rule $\Gamma$, is a tuple $(\theta_1, \theta_T)$, where $\theta_1$ is called a schema valuation, and $\theta_T$ is an instance valuation. Valuation $\theta_1$ maps the level and attribute variables in $\Gamma$ to level and attribute names in $B_s$, while $\theta_T$ maps domain variables to values in $I(B)$.

**Definition 9** A schema valuation for a rule $\Gamma$, denoted $\theta_1(\Gamma)$ maps level and attribute variables in the atoms of $\Gamma$ as follows: (a) given a rollup atom of the form $d : x : x \rightarrow y : y$, $\theta_1$ maps $d$ to a dimension name in $D_s$, $\tau$ to a value $w \in T_P$, and $X$ and $Y$ to a pair of values $v, u$ s.t. $v \leq w$, $w$ holds in $d \in D_s$; (b) if the rollup atom is of the form $x : x \rightarrow Y : y$, $\theta_1$ maps $\tau$ to a value $w \in T_P$, and $X$ and $Y$ to a pair of values $v, u$ s.t. $v \leq w$, $w$ holds in $d \in D_s$; (c) if the rollup atom is of the form $x \rightarrow y : y$, $\theta_1$ maps $\tau$ to a value $w \in T_P$, and $Y$ to a value $u$ s.t. $\text{level}(x, w) \leq u$, $u$ holds in $d \in D_s$; (d) for the rollup atoms of the form...
\[ x \rightarrow y : \theta, \] maps \( x \) and \( y \) to dimension levels in \( \text{dim}(X) \) s.t. \( v, u, s.t. v \sqsupset u \text{ holds in } \text{dim}(X) \); (c) given a descriptive atom of the form \( A \rightarrow \gamma \), \( \theta, \) maps \( \gamma \) to a value \( w \in T_P \), and \( A \) to an attribute name \( u \in A \), s.t. \( u \sqsupset \gamma \text{level}(x, w) \in \text{dim}(X) \in D_P \).

Given a rule schema valuation \( \theta, (\Gamma) \) for a rule \( \Gamma \), an instance valuation \( v \) is a function \( \theta, \) s.t. \( \theta(s) \) maps the domain variables \( x \) and \( y \) in the rollup atoms defined above, to values in \( \mathcal{R}_D \) over levels defined by \( \theta, \) (b) \( \theta, \) maps a fact \( F \) to \( \mathcal{F}_n \), the rightmost term \( t \in F \), to a value \( v \in T_P \), each domain variable \( x_i \) in \( F \) to a value in \( \text{dom}(\text{level}(x_i, w)) \), and the data term \( \text{N} \) in \( F \) to a value in \( N \).

A constraint atom \( \{ \gamma \} \) is true if, given a valuation \( \theta, \) s.t. \( \theta(s) \) maps the domain variables \( x \) and \( y \) in the rollup atoms defined above, to values in \( \mathcal{R}_D \) over levels defined by \( \theta, \) \( \theta(t) = w, \theta(y) = 1, \) and \( \theta[y] = v, \) then it does not exist a rollup in \( \text{dim}(X) \) s.t. \( \gamma(X, A, \{ x \}) = v \). Predicate and function symbols are evaluated as in standard datadog.

Let \( \text{AGG} \) be the set of aggregate functions, with extension \( \text{AGG} = \{ \text{MIN}, \text{MAX}, \text{COUNT}, \text{SUM} \} \), and \( r \) a relation. The aggregate operation \( \gamma \) over \( X A \), s.t. \( X A \in \text{schema}(r), f \in \text{AGG}, \) and \( f(A) \) denotes the aggregation of the values in \( \{ x \}, t \in r, \) using \( f \). Thus, we can now define the semantics of a \( \text{TOLAP} \) rule \( \Gamma \), of the form \( Q(a_1, a_2, \ldots, a_n, \text{AGG}(m)) \leftarrow A_1, \ldots, A_m \) as follows. For each level or data variable \( v_i \) in the body of \( \Gamma \), and for a valuation \( \theta \) of the variables in the rule’s body, we have:

\[ r_{\Gamma} = \{ \theta(v_1), \ldots, \theta(v_n) \} | \theta \text{ is a valuation of } \Gamma \} \]

Then

\[ Q = \gamma_{\text{AGG}}(a_1, \ldots, a_n)(r_{\Gamma}). \]

3.5 Expressive power

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

3.5.1 What can be expressed in \( \text{TOLAP} \)?

Intuitively, it is not hard to see that \( \text{TOLAP} \) 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 \( d: X : X \rightarrow Y : y \), we effectively need to compute the transitive closure of the rollup functions for dimension \( d \). It is well-known that this can 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 joints needed to transversely 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, \( \text{north, south, east, west} \), 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 \textit{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 \textit{merge} of elements.

Let us suppose the query “\textit{total sales per item and region}, using only the currently existing regions (or their descendents)”’. This query cannot be expressed in \( \text{TOLAP} \). To show this, suppose a region \( r \) is split into \( r_1 \) and \( r_2 \). After that, \( r_1 \) is merged with another region \( r_3 \). 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 \( \text{TOLAP} \)

We extend \( \text{TOLAP} \) 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: \textit{split}(\( x, y, L, t \)) and \textit{merged}(\( x, y, L, t \)), with the following meanings: (a) \textit{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) \textit{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 \textit{split} and \textit{merge}, given in [HMV99b].

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

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

\[
\begin{align*}
\text{shift}(x, y, L, t) & \leftarrow \text{split}(x, y, L, t). \\
\text{shift}(x, y, L, t) & \leftarrow \text{merged}(x, y, L, t). \\
\text{shift}(x, y, L, t_2) & \leftarrow \text{shift}(x, z, L, t_1). \\
\text{shift}(x, y, L, t_2) & \leftarrow \text{shift}(x, z, L, t_1), \quad t_2 > t_1. \\
\text{split}(x, z, L, t_1, t_2) & \leftarrow \text{merged}(x, y, L, t_2), \quad t_2 > t_1.
\end{align*}
\]

From predicate \( \text{shift} \) we derive another one, called \( \text{shiftPers}(x, y, L, t) \), which extends the validity of the split or merge, to every instant \( t \) between updates. For instance, if \( \text{shift}(r, r_1, L, 10) \) and \( \text{shift}(r_1, r_2, L, 13) \) hold, then, \( \text{shiftPers}(r, r_1, L, 11) \) and \( \text{shiftPers}(r, r_1, L, 12) \) also hold. This has been called Persistence [BWJS89]. Thus:

\[
\begin{align*}
\text{shiftPers}(x, y, L, t) & \leftarrow \text{shift}(x, y, L, t). \\
\text{shiftPers}(x, y, L, s(t)) & \leftarrow \text{shiftPers}(x, y, L, t), \\
& \quad \text{shift}(y, y, L, s(t)), \\
& \quad \text{shiftPers}(y, y, L, t), \\
& \quad \text{merged}(y, y, L, t), \\
& \quad \text{deleted}(y, L, t_1), \\
& \quad \text{inserted}(y, L, t_2). \\
\text{shiftPers}(x, y, L, s(t)) & \leftarrow \text{shiftPers}(x, y, L, t_1), \\
& \quad \text{deleted}(y, L, t_1), \\
& \quad \text{inserted}(y, L, t_2), \\
& \quad \text{inserted}(y, L, t_1), \\
& \quad \text{inserted}(y, L, t_2). \\
\end{align*}
\]

Here, \( s(t) \) stands for the successor of \( t \). Predicates \( \text{deleted}(y, L, t) \) and \( \text{inserted}(y, L, t) \) 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:

\[
\begin{align*}
L_1 : x & \xrightarrow{t_1} L_2(t_2) : y \quad \iff \quad L_1 : x \xrightarrow{t_1} L_2 : y, \\
L_1 : x & \xrightarrow{t_1} L_2 : z \quad \iff \quad \text{shiftPers}(z, y, L, t_2). \\
L_1 : x & \xrightarrow{t_1} L_2(t_2) : y \quad \iff \quad L_1 : x \xrightarrow{t_1} L_2(t_2) : z, \\
L_1 : x & \xrightarrow{t_1} L_2(t_2) : y \quad \iff \quad \text{shiftPers}(y, z, L, t_2).
\end{align*}
\]

We will denote this extension of \( \text{TOLAP} \) as \( \text{TOLAP}^+ \). The meaning of a \( \text{TOLAP}^+ \) query is analogous to the meaning of a \( \text{TOLAP} \) one. Now, we can express the query “total sales per item and region, using only the currently existing regions”, as:

\[
\begin{align*}
\text{IR}(p, r, \text{SUM}(s)) & \leftarrow \text{Sales}(p, s, t, s, t), \\
& \quad \text{month}:m, \\
& \quad \text{at} \xrightarrow{m} \text{reg}(\text{Now}):x.
\end{align*}
\]

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 “1F” store \( s_1 \) belonged to region \( r_2 \) which no longer exists at time “1(F”, 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 \( \text{at} \xrightarrow{m} \text{reg}(\text{Now}):x \) were replaced by \( \text{at} \xrightarrow{m} \text{reg}(\text{Now}):x \), 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 \( \text{TOLAP} \).

4 Implementation

\( \text{TOLAP} \) could be implemented in at least two different ways: (a) translating \( \text{TOLAP} \) queries to SQL, or (b) using a temporal query language like TSQL2 [Sho95]. 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: \( (\text{Dimension}, \text{upLevel}, \text{loLevel}, \text{From}, \text{To}) \), where \( \text{loLevel} \subseteq \{(\text{From}, \text{To})\} \text{upLevel} \). Although there is only one instance relation for each dimension instance, two approaches can be followed: in the first one, the instance relation has the form \( (\text{upLevel}, \text{loLevel}, \text{upVal}, \text{loVal}, \text{From}, \text{To}) \), where \( \text{rho}_{\text{loLevel}}[(\text{From}, \text{To})][\text{loVal}] = \text{upVal} \). 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 \( \text{From}, \text{To} \). 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 analyze consists in translating \( \text{TOLAP} \) queries to SQL, using the data structure defined above. The following example will show that even simple \( \text{TOLAP} \) queries can have non-trivial SQL translations.

Suppose that in our running data warehouse, level \( \text{itemId} \) was deleted from the \text{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 \( \text{TOLAP} \) this query would read:
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.itemld = P.indVal AND P.itemld = 'itemld'
  AND S1 loLevel = 'storeld' AND S1.10Level='city'
  AND S1.upVal = S2.10Level
  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.upVal,S1.upVal
UNION
```

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.itemld = P.indVal AND S1.10Level = 'storeld'
  AND P.10Level = 'itemld' AND P.upVal = 'Brand'
  AND S.storeld = S1.storeld
  AND S1.upVal = S2.upVal
  AND S1.10Level = 'city' AND S1.upVal = S2.10Level
  AND S2.10Level = '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
GROUP BY T.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 while the dimension goes through over time. In the right window, a tree-view of the rollup fucntions 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 rule-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


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.