Improving automatic SQL translation for ROLAP tools

Oscar Romero Alberto Abelló

Dept. de Llenguatges i Sistemes d'Informació, Universitat Politècnica de Catalunya

C/ Manuel Girona 1-3, E-08034 Barcelona. {oromero|aabello@lsi.upc.edu}

Abstract

In the last years, despite a vast amount of work have been devoted to modeling multidimensionality, multidimensional algebra translation to SQL have been overlooked. ROLAP tools automatically generate a cube-query according to the operations performed by the user. The SQL translation does not represent a problem when treating isolated operations but when mixing up together modifications brought about by a set of operations in the same cube-query, some conflicts could emerge depending on the operations involved. Therefore, if these problems are not detected and treated appropriately, the automatic translation can retrieve unexpected results. In this paper, we define and classify conflicts raised when automatically translating a multidimensional algebra to SQL, and analyze how to solve or minimize their impact.

Keywords: Multidimensional operations, ROLAP, Data Warehouse

1 Introduction

On-Line Analytical Processing (OLAP) tools are intended to ease information analysis and navigation all through the "Data Warehouse". OLAP functionality is characterized by dynamic multidimensional analysis of consolidated enterprise data supporting end user analytical and navigational activities. "Navigation" means to interactively explore a data cube by drilling, rotating and screening, and as presented in [4] we consider "roll-up" (increase the aggregation level), "drill-down" (decrease the aggregation level), "screening and scoping" (select by means of a criterion evaluated against the data of a dimension), "slicing" (specify a single value for one or more members of a dimension) and "pivot" (reorient the multidimensional view), as the typical end user operations performed on data cubes. Some authors, like [14] and [3], add "drill-across" (combine data from cubes sharing one or more dimensions) to these basic operations.

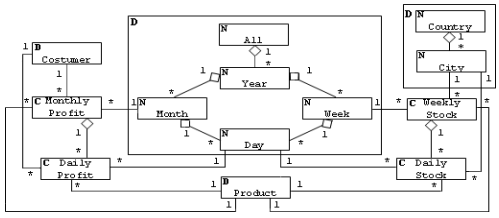

Figure 1: Example of multi-star schema

Multidimensional conceptual view of data is distinguished by the fact/dimension dichotomy. From here on, we will use **YAM**^{2} multidimensional terminology in [3], where a **Dimension** contains **Levels** representing different granularities (or levels of detail) to study data, and a **Level** contains **Descriptors**. Consonantly, a **Fact** contains **Cells** which contain **Measures**. One **Cell** represents those individual **cells** of the same granularity that show data regarding the same **Fact** (i.e. a **Cell** is a "Class" and **cells** are its instances). One **Fact** and several **Dimension**s to analyze it give rise to a **Star**. Moreover, as discussed in [3], we consider quite important to be able to relate different **Star**s not only sharing dimensions but defining semantic relationships between them like UML *Generali-*

zation, *Association*, *Derivation* or *Flow*; some already considered in other conceptual models as [7] and [12]. In figure 1, we find two **Fact**s containing two **Cell**s each one (`Daily Profit` and `Monthly Profit`; and `Daily Stock` and `Weekly Stock`), and sharing two **Dimension**s of analysis (`Time` and `Product`).

[8] shows how a **Star** should be implemented on a "Relational Database Management System" (RDBMS) through a *star* or a *snowflake* schema. The star schema consists of one table for the **Fact** and one denormalized table for every **Dimension** with the latter being pointed by "foreign keys" (FK) from the "fact table", which compose its "primary key" (PK). The normalized version of a star schema is a snowflake schema, getting a table for each **Level** with a FK pointing to each of its parents in the **Dimension** hierarchy. Both approaches can be conceptually generalized into a more generic one consisting in partially normalizing the **Dimension** tables according to our needs. Completely normalizing each **Dimension** we get a snowflake schema and not normalizing them at all results in a star schema. We choose this generic approach as we consider, like in [11], a **Fact** can contain not just one but several materialized **Cell**s ("Cell tables"). So that, each **Level** related to a materialized **Cell** must be also materialized as a table since a FK (each FK in the **Cell** pointing to **Level**s related to it) must be related to a PK, or at least, to a "unique" table field. If a certain **Level** is only related to non materialized **Cells** we can denormalize it. In figure 1, we have decided to materialize the four **Cell**s stated explicitly. Hence, those **Level**s directly related to them will be materialized, but, for instance, `Year` **Level** will not since no materialized **Cell** points to it.

Now, we present the standard SQL'92 template query (also known as *cube-query*) to retrieve a **Cell** of data, which conforms a **Cube** (a set of **cell**s placed in an n-dimensional space), from the RDBMS.

```
SELECT l1.ID, ..., ln.ID, c.Measure1, ...
FROM Cell c, Level1 l1, ..., Leveln ln
WHERE c.key1=l1.ID AND ...  AND c.keyn=ln.ID
GROUP BY l1.ID, ..., ln.ID
ORDER BY l1.ID, ..., ln.ID
```

"Cell" and "Level tables" in the FROM are appropriately linked in the WHERE clause. Finally, we talk about *atomic cube-query* when it retrieves a **Cube** of data not yet manipulated by multidimensional operations.

However, despite we already know how to implement a multidimensional schema in a RDBMS and how to retrieve data from it, there is not yet a standard multidimensional algebra accepted as reference point. As presented in section 2, some algebras have already been proposed but some of them do not directly map to SQL and, in general, none of them offers the translation of the operations to SQL, ignoring some problems that can arise due to SQL intrinsic restrictions. "ROLAP" (OLAP over RDBMS) tools automatically generate a cube-query according to the operations performed by the user. The SQL translation does not represent a problem when treating isolated operations but when mixing up together modifications brought about by a set of operations in the same cube-query, some conflicts could emerge depending on the operations involve. Therefore, if these problems are not detected and treated appropriately, the automatic translation can retrieve unexpected results. Our main contribution in this paper is to define and classify those conflicts when automatically translating a multidimensional algebra to SQL and analyze how to solve or minimize their impact.

To carry out our study we use **YAM**^{2} multidimensional algebra presented in section 3. There, we also present how this algebraic set of operations should be translated to SQL. In section 4 we introduce in detail problems arisen when translating these operations to SQL, which can be mainly classified as the *multiple aggregation*, the *fan-shaped* and the *selection granularity* problems. Former will be discussed in subsection 4.1, second one in 4.2 and the last one in subsection 4.3. Finally, section 5 concludes the paper.

2 Related Work

A vast amount of work have been devoted to modeling multidimensionality as have been gathered in several surveys like [1], [10], [13] and

Clause	ChangeBase	Drill-across	Selection	Roll-up	Projection	Union	
SELECT	Replace (LevelID)	Add (Measure)		Replace (LevelID)	Remove (Measure)		
FROM	Add (Levels)	Add (Cell)				Union (Cells and Levels)	
WHERE	Add (links)	Add (links)	AND (conditions)			Union (links)	OR (conditions)
GROUP BY	Replace (LevelID)			Replace (LevelID)			
ORDER BY	Replace (LevelID)			Replace (LevelID)			

Table 1: SQL query sentence modifications according to each multidimensional operation

[15]. Additionally, other multidimensional models have been presented later.

[16] presents an algebra over an XML and OLAP federation. "Selection Cube" selects data desired; the "Decoration" operator adds new **Dimension**s to the **Cube** and the "Federation Generalized Projection" "rolls-up" the **Cube** and removes unspecified **Dimension**s and **Measures** (meaning it is not an atomic operation). Finally, "drill-across" is not considered and the rest of its operations are intended to manipulate data in the federation. They also consider how these operations must be translated to SQL but, unlike us, conflicts when translating are not studied exhaustively.

An algebra with four operations is presented in [5]. "Derived Measures" derives new measures from already existent; "Join" is similar to "drill-across" but in a more restrictive way since it forces both **Cube**s to have the same **Dimension**s and **Level**s; "Slice" and "Multislice" select a single or a range of values; and finally, "Union", "Intersection" and "Difference" allow us to manipulate **Cube**s. In our case, we only consider **Union** since the same considerations can be done for "Intersection" and "Difference" just changing the OR operator by AND or AND NOT. "Roll-up" is introduced in the "Aggregated Cubes", an algebraic part of the schema. Anyhow, like models surveyed in above references, it does not offer the translation of their algebraic set of operations to SQL (rather they propose alternatives to SQL and relational algebra). Those models proposing alternatives to SQL argue that RDBMS are not well suited for multidimensional purposes. However, the importance of ROLAP tools in the market contradicts that, outlining relevance of research on improving the usage of SQL to query multidimensional data.

In [2] we find how to translate each isolated operation of $\mathbf{YAM}^2$ multidimensional algebra to SQL, although it does not consider combining a set of operations in the same cube-query. In addition, it states $\mathbf{YAM}^2$ algebra as complete, so that, any other multidimensional operation can be expressed in terms of it.

3 YAM^2 Multidimensional Algebra

In this section we present $\mathbf{YAM}^2$ operations introduced in detail in [2], intended to manipulate **Cube**s, and also how they should be translated to SQL (summarized in table 1). To better follow operations introduced below, see the sequence of examples in table 2:

Selection: By means of a logic clause C over a **Descriptor**, this operation allows to choose the subset of points of interest out of the whole n-dimensional space. In SQL, it means to *and* the corresponding clause to the WHERE clause. For instance, we can select those `Weekly Stock`s referring to `cookies` in the `Product` **Dimension**.

Roll-up: It groups **cell**s in the **Cube** based on an aggregation hierarchy. This operation modifies the granularity of data by means of a many-to-one relation which relates instances of two **Level**s in the same **Dimension**, corresponding to a part-whole relationship. In SQL, it changes the identifier in the GROUP BY clause by that of the parent **Level**. Thus, SELECT and ORDER BY clauses must be modified accordingly, so that the **Descriptor**s coincide in all three. Measures in the SELECT clause must also be summarized using an aggregation function. To roll up to **Level All**, all **Descriptor**s of a **Dimension** are removed from the GROUP BY, and "All" is placed in the the corresponding place in SELECT clause. Going on, we can **Roll-up** from `City` to **Level** `All` along `Place` **Dimension**.

```
SELECT p.ID, w.ID, c.ID, s.Stock
FROM weeklyStock s, Product p, Week w, City c
WHERE s.key1 = p.ID AND s.key2 = w.ID
AND s.key3 = c.ID AND p.name = 'cookies'
GROUP BY p.ID, w.ID, c.ID
ORDER BY p.ID, w.ID, c.ID
```

a) Selection

```
SELECT p.ID, w.ID, ''All'', SUM(s.Stock)
FROM weeklyStock s, Product p, Week w
WHERE s.key1 = p.ID AND s.key2 = w.ID
AND p.name = 'cookies'
GROUP BY p.ID, w.ID
ORDER BY p.ID, w.ID
```

b) Roll-up

```
SELECT p.ID, w.ID, SUM(s.Stock)
FROM weeklyStock s, Product p, Week w
WHERE s.key1 = p.ID AND s.key2 = w.ID
AND p.name = 'cookies'
GROUP BY p.ID, w.ID
ORDER BY p.ID, w.ID
```

c) ChangeBase

```
SELECT p.ID, d.ID, s.Stock, m.profit
FROM dailyStock s, dailyProfit m, Product p, Day d
WHERE s.key1 = p.ID AND s.key2 = d.ID
WHERE m.key1 = p.ID AND m.key2 = d.ID
AND p.name = 'cookies'
GROUP BY p.ID, d.ID
ORDER BY p.ID, d.ID
```

d) Drill-across

```
SELECT p.ID, d.ID, m.profit
FROM dailyProfit m, Product p, Day d
WHERE m.key1 = p.ID AND m.key2 = d.ID
AND p.name = 'cookies'
GROUP BY p.ID, d.ID
ORDER BY p.ID, d.ID
```

e) Projection

```
SELECT p.ID, d.ID, m.profit
FROM dailyProfit m, Product p, Day d
WHERE m.key1 = p.ID AND m.key2 = d.ID AND
(p.name = 'cookies' OR p.name = 'chocolate')
GROUP BY p.ID, d.ID
ORDER BY p.ID, d.ID
```

f) Union

Table 2: Example of $\mathbf{YAM}^2$ algebra translation to SQL

ChangeBase: This operation reallocates exactly the same instances of a **Cube** in a new n-dimensional space with exactly the same number of points, by means of a one-to-one relation. It really allows two different kinds of changes in the "base of the space" (those dimensions identifying each **cell**). Firstly, we can just rearrange the multidimensional space by reordering **Level** identifiers in ORDER BY and SELECT clauses (this would be equivalent to the "pivot" operation); and secondly, if there exist more than one set of **Dimension**s that identify the points in the space, we can change between those **Dimension**s modifying the analysis **Dimension**s used. It is, adding the new **Level** tables to the FROM and the corresponding links to the WHERE clause. Moreover, identifiers in the SELECT, ORDER BY and GROUP BY clauses must be replaced. Finally, Notice semantic relations rise new possibilities allowing us to replace a **Dimension** by one semantically related to it by a one-to-one relation. Following with the same example, we can change from ($\texttt{Time} \times \texttt{Product} \times 1$) to ($\texttt{Time} \times \texttt{Product}$) without losing **cell**s.

Drill-across: This operation changes the subject of analysis of the **Cube** by means of a one-to-one relation. The n-dimensional space remains exactly the same, only the **cell**s placed in it change. Like in the **ChangeBase** operation, semantic relations rise new possibilities as presented in [2]. In SQL, it means to add a new **Cell** table to the FROM, its **Measure**s to the SELECT, and the corresponding links to the WHERE clause. In general, if we are not using any *Relationship*, a new **Cell** table can always be added to the FROM clause if the attributes composing the identifier of the desired **Cell** point to the already used **Level** tables. For instance, in the same example, we could **Drill-down** to `Daily Stock` and directly **Drill-across** to `Daily Profit`.

Projection: This just selects a subset of **Measure**s from those available in the **Cube**. So that, it removes **Measures** from the SELECT clause. If there is not any **Measure** left, COUNT is assumed. Following our example, we can remove the `Stock` **Measure**.

Union: It unites two **Cube**s containing the same **Cell**s if both are defined over the same n-dimensional space. In SQL, we unite both FROM clauses, WHERE links, and finally *or* conditions of WHERE clauses. Hence, we can unite our example query to one identical but querying for `chocolate` instead of `cookies`.

The algebra composed by these operations is "closed" (applied to a cube-query, the result of all operations is another cube-query), "complete" (any valid cube-query can be computed as the combination of a finite set of operations applied to the appropriate **Cell**) and "minimal" (none can be expressed in terms of others, nor can any operation be dropped without affecting its functionality). Moreover, other operations can be derived by sequences of these operations. This is the case of **Slice** (which reduces the dimensionality of the original **Cube** by fixing a point in a **Dimension**) by means of **Selection** and **ChangeBase** operations. About **Drill-down** (i.e. the inverse of **Roll-up**), as argued in [6], it can only be applied if we previously performed a **Roll-up** and did not lose the correspon-

Ops/Source	∅	Selection	Roll-up	Drill-across
Roll-up	×	×	×	×
Drill-across	×		×	×

Table 3: Conflicts summary

dences between **cell**s. Losing correspondences can happen due to extra navigation between **Cube**s (through **Drill-across** or **Change-Base**) resulting that we do not have data in a lower aggregation **Level** for the target **Cube**.

4 SQL Translation Problems

In section 3 we have presented how an atomic cube-query should be modified when applying an isolated operation over it, but many times end users demand to navigate from **Cube** to **Cube** not just applying isolated operations but performing sequences of operations. Thus, a user chooses a source **Cube** from where starting to operate. Automatically, the ROLAP tool will conform a cube-query to retrieve this **Cube**. Notice this **Cube** is our start point so that it has not been yet manipulated by any operation. Consequently, it is placing a **Cell** of data on the n-dimensional space conformed by its analysis **Dimension**s. This **Cell**, as stated in section 1, could have been materialized or not. If it was, ROLAP tool will retrieve it from an atomic cube-query and if not, it will look for an appropriate **Cell**, in a lower aggregation **Level**, from where obtaining the needed **Cell** by means of **Roll-up**s. For instance, we could start our analysis from a materialized **Cell** (i.e. `Monthly Profit`) or from a non materialized one (i.e. `Annual Profit`). As `Annual Profit` is not materialized, we need to perform an implicit **Roll-up** over `Monthly Profit` from `Month` to `Year` to get needed data.

As presented in table 3, certain operations can pop up a conflict when combined with a concrete *source cube-query*. We refer to a source cube-query as an atomic cube-query modified by a sequence of operations. If no operation has been performed over the atomic cube-query we consider the empty sequence (∅). Hence, a cell is crossed (×) when the sequence of operations in the source cube-query contains a concrete operation that can cause a conflict with next one to be performed. For instance, it can happen if our source cube-query includes a **Selection** and next operation to be carried out is a **Roll-up**. Anyhow, any kind of conflict could be avoided using one subquery per multidimensional operation. But we only use subqueries if strictly necessary, shunning the materialization of partial results and easing the RDBMS query optimizer job.

Following, notice all conflicts are related to **Roll-up** and **Drill-across**, the rest of operations, except for **Selection**, propagate conflicts if already present in the cube-query but do not introduce new ones. **Projection**, **Union** and **ChangeBase** never arises a conflict. Intuitively, **Projection** removes **Measure**s from the SELECT clause and dropping a **Measure** just means to omit a "Cell table" column; **Union** *or*es conditions of two **Cube**s with the same n-dimensional space, selecting desired points prior to perform operations; and **ChangeBase** always asks for a one-to-one relation in order to be performed, avoiding conflicts due to its own nature.

Despite all conflicts are due to data aggregation anomalies (then, at least, a one-to-many relation is required), in our study we have classified conflicts raised in three groups; those caused by performing multiple aggregation functions in a query, those due to hidden many-to-many relationships and finally, those related to the selection granularity. Prior to present them in detail, we need to formalize some concepts that will help us to introduce our results. [9] presents three necessary conditions to warrant a correct data summarization:

Disjointness: *Sets of* **cell***s at a* **Level** *to be aggregated must be disjoint.*

Completeness: **cell***s at a certain* **Level** *must be constituted by all the* **cell***s of its child* **Level***s.*

Compatibility: **Dimension***, kind of measure aggregated and the aggregation function must be compatible.* Compatibility must be satisfied since certain functions are incompatible with some **Dimension**s and kind of measures. For instance, we can not aggregate `Stock` over `Time` **Dimension** by means of sum, as some repeated values would be counted.

When aggregating data we have to assure

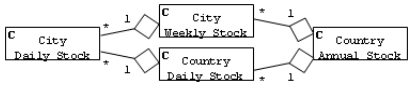

Figure 2: Example of a hierarchy of Cells

these conditions to avoid summarizability anomalies. If not, we will face duplicated values or find that some measurements at an aggregation **Level** cannot be used to obtain data at higher aggregation **Level**s, forcing us to go to finer granularities, maybe to the "atomic **Level**" (lower **Level** in a **Dimension** hierarchy that is always materialized), to obtain the source data for the calculation.

4.1 The Multiple Aggregation Problem

First problem is about functions used to aggregate data. This case typically arises when combining more than one **Roll-up** in the same cube-query. To analyze this problem, we conceptually divide a combination of two **Roll-up**s in two categories depending on whether both were performed over the same **Dimension** or over different ones.

In the first case, we can always solve the problem disregarding first **Roll-up** and just performing the second one, because in a certain moment of time, multidimensional data can only be showed in a certain aggregation **Level** for each **Dimension**. Notice it always can be assumed since, in the worst case, we can perform a **Roll-up** from the atomic **Level**. Oppositely, when performed over different **Dimension**s we have to compulsory aggregate data for each **Dimension**. Since SQL does not allow us to aggregate data by means of two different functions in the same query this conflict can not be solved in a single cube-query. For instance, if we carry out a **Roll-up** from `Week` to `Year` **Level** in the `Weekly Stock` **Cell**, and later we **Roll-up** from `Year` to **Level** `All`, the whole sequence of both **Roll-up**s can be directly expressed as:

```
SELECT p.ID, "All", c.ID, SUM(s.Stock)
FROM weeklyStock s, Product p, City c
WHERE s.key1 = p.ID AND s.key3 = c.ID
GROUP BY p.ID, c.ID
ORDER BY p.ID, c.ID
```

On the contrary, if we just carry out first **Roll-up**, and later another one from `City` to `Country` along the `Place` **Dimension**, nested queries are compulsory:

```
SELECT p.ID, co.ID, y.ID, SUM(s.Stock)
FROM (SELECT p.ID, c.ID, y.ID, AVG(s.Stock)
    FROM weeklyStock s, Product p,
    City c, Week w, Year y
    WHERE s.key1 = p.ID AND s.key2 = c.ID
    AND s.key3 = w.ID AND w.fkey = y.ID
    GROUP BY p.ID, c.ID, y.ID
    ORDER BY p.ID, c.ID, y.ID), Country co
WHERE s.key1 = p.ID AND s.key2 = c.ID
AND s.key3 = w.ID AND c.fkey = co.ID
GROUP BY p.ID, co.ID, y.ID
ORDER BY p.ID, co.ID, y.ID)
```

Even if SQL allowed us to perform more than one aggregation function in the same query, we would face another problem: the order between aggregation functions. Consider the `Stock` Cell hierarchy detailed in figure 2 extracted from the example presented in figure 1. In this case, `Stock` is analyzed through two **Dimension**s (`Place` and `Time`), and for each possible combination of its **Level**s we got a different **Cell**. For instance, `City Weekly Stock` (containing **cell**s on a `Week-City` granularity **Level**), `Country Annual Stock` (`Country-Year`), `City Daily Stock` (`City-Day`), etc. Thus, it is important to realize that our own multidimensional conceptual design fixes the order of aggregation functions when navigating along **Cell**s hierarchy. If we want to **Roll-up** from `City Daily Stock` to `Country Annual Stock` we have to first aggregate by means of sum (it means, **Roll-up** from `City` to `Country` **Level**) and later aggregate by means of average (**Roll-up** from `Day` to `Year`). So that, order does really matter since sum of averages is different from an average of sums (latter happens when navigating through `City Weekly Stock`). Both orders are possible, but semantics chosen when designing our schema forces us to follow a certain order.

As said, above conflict could be avoided if SQL allowed us to perform more than one aggregation function per query and set up an order between them. For instance, an SQL extension as showed below stating explicitly two GROUP BY's, very similar to SQL'99 GROUPING SETS modus operandi, would avoid using nested queries when combining more than one conflictive **Roll-up**. First GROUP BY would be related to first aggregation function and analogously to second one:

```
SELECT p.ID, co.ID, y.ID, AVG(SUM(s.Stock))
FROM weeklyStock s, Product p, City c, Week w, Year y, Country co
WHERE s.key1 = p.ID AND s.key2 = c.ID
AND s.key3 = w.ID AND w.fkey = y.ID AND c.fkey = co.ID
GROUP BY p.ID, c.ID, y.ID
GROUP BY p.ID, co.ID, y.ID
ORDER BY p.ID, c.ID, y.ID
```

Nevertheless, although this problem has been presented as a **Roll-up** plus **Roll-up** problem, it goes far beyond as it is crucial when obtaining non materialized **Cell**s from materialized ones. For instance, if we have to work with `City Weekly Stock` **Cell** that have not been materialized, ROLAP tools will have to perform a **Roll-up** from `Day` to `Week` over `City Daily Stock` to obtain needed data. So that, we have already performed an implicit **Roll-up** that could arise conflicts already presented if we next perform just one explicit **Roll-up**. Similarly, as presented in 4.2, implicit **Roll-up**s can also appear when carrying out a **Drill-across** (also in a **ChangeBase**, but in this case it is raised over the same **Dimension** avoiding any kind of conflict) from a non materialized **Cell**.

Meanwhile, best solution to minimize this problem is to choose with care appropriate **Cell**s to be materialized. An extreme solution would be to materialize all of them, but since it is an exponential space problem, it is not feasible. Hence, in addition to traditional criteria like how frequently would be a **Cell** queried, this problem defines another criterion to decide the usefulness of a given materialized view. According to semantics related to our **Cell**s hierarchy, those **Cell**s whose data can be used as pre-aggregated data to calculate above **Cell**s are good candidates (for instance, in case presented, to materialize `Country Daily Stock` instead of `City Weekly Stock`, since `Country Annual Stock` can only be calculated through the former).

4.2 The Fan-Shaped Problem

In this section we introduce a family of problems that are caused because disjointness is not preserved when aggregating data in certain situations. It typically appears related to **Drill-across**, either through semantic relationships or shared **Dimension**s. **Drill-across** asks for a one-to-one relationship, but sometimes a one-to-many relation is enough. For instance, after dropping the `Place` **Dimension** (by means of **Roll-up** and **ChangeBase**) we can **Drill-across** from `Annual Stock` to `Annual Profit`. Conceptually, the one-to-one relation is quite clear but in fact, we really have a one-to-many relation since both **Cell**s are not materialized and `Weekly Stock` and `Monthly Profit` are related to different **Level**s in the `Time` **Dimension**. We can get the needed one-to-one relation by means of internal **Roll-up**s (from `Month` to `Year` over both **Cells**). Since `Year` is not materialized, its descriptors are included along with its children **Level**s in the `Time` **Dimension** hierarchy, given raise to the following query:

```
SELECT p.ID, y.ID, AVG(s.Stock), SUM(s.Profit)
FROM weeklyStock s, monthlyProfit m, Product p
Month mo, Week w, Year y
WHERE m.key1 = p.ID AND m.key2 = mo.ID
AND s.key1 = p.ID AND s.key2 = w.ID
AND mo.yearID = w.yearID
GROUP BY p.ID, y.ID
ORDER BY p.ID, y.ID
```

As stated in section 4, the aggregation of data must be disjoint, and in this case, it is not. In fact, what should be a one-to-one relation turns into a many-to-many one calling up a fan-shaped matching. Thus, we should use a nested query performing first one **Roll-up** and later, the other one, being the "join" last performed. Hence, this problem could be solved if SQL allowed us to state a priority between "joins" and GROUP BY's. However, to minimize its impact is important, again, to choose with care which **Cell**s should be materialized. Therefore, this is another criterion to bear in mind when deciding the usefulness of a given materialized view.

Finally, also notice that when carrying out a **Drill-across** to a non materialized **Cell**, a ROLAP tool may need to perform internal **Roll-up**s to obtain data to where **Drill-across**. Internal **Roll-up**s followed by an explicit **Roll-up** can cause the same conflict stated in subsection 4.1.

4.3 The Selection Granularity Problem

This problem is closely tied to **Selection** and raises when completeness is not guaranteed. **Selection** allows us to reduce current n-dimensional space by means of a logic clause

over a certain **Descriptor**. For instance, selecting those **cell**s of `Daily Stock` related to `Barcelona` in the `Place` **Dimension**. Now, if we **Roll-up** from `Day` to `Week` we cannot change `Daily Stock` to `Weekly Stock` **Cell** in the cube-query to take advantage of pre-aggregated data, since aggregation in `Weekly Stock` is complete and in our current **Cell** it is not (we only have those points related to `Barcelona`). We cannot take profit of any pre-aggregated data in a materialized **Cell** when translating to SQL if a **Selection** has been carried out over a lower **Level Descriptor** in any of its analysis **Dimension**s. Using appropriate granularity **Cell** and performing internal **Roll-up**s is mandatory. Only way to solve this problem is considering it in the multidimensional schema. For instance, using semantic relationships and creating an specialization of `Daily Stock` (i.e. `Barcelona Daily Stock`) and another on `Weekly Stock` (i.e. `Barcelona Weekly Stock`). Between those **Cell**s, aggregation is complete and we can use pre-aggregated data without problems.

5 Conclusions

This paper analyzes conflicts that could arise when automatically translating to SQL a set of operations in a single cube-query. Despite all conflicts are due to data aggregation anomalies, we have classified conflicts raised in three groups; those caused by performing multiple aggregation functions in a query, those due to hidden many-to-many relationships and finally, those related to the selection granularity.

We have also presented how to solve these problems avoiding subqueries. First two problems can be shunned, or at least smoothed, choosing with care appropriate **Cell**s to be materialized. Both problems define two new criteria to decide the usefulness of a given materialized view: according to semantics related to our **Cell**s hierarchy and avoiding hidden many-to-many relationships. The selection granularity problem is more specific, and can only be solved considering it in the multidimensional conceptual schema.

References

[1] A. Abelló, J. Samos, and F. Saltor. A Framework for the Classification and Description of Multidimensional Data Models. In *Proc. of DEXA'2001*. Springer, 2001.

[2] A. Abelló, J. Samos, and F. Saltor. Implementing Operations to Navigate Semantic Star Schemas. In *Proc. of DOLAP'2003*. ACM, 2003.

[3] A. Abelló, J. Samos, and F. Saltor. **YAM**^{2} (Yet Another Multidimensional Model): An extension of UML. *Information Systems*, Elsevier, 2005. (In Press). Available at http://www.sciencedirect.com.

[4] U. S. E. Franconi, F. Baader and P. Vassiliadis. *Fundamentals of Data Warehousing*, chapter Multidimensional Data Models and Aggregation. Springer, 2000. M. Jarke, M. Lenzerini, Y. Vassilious and P. Vassiliadis editors.

[5] E. Franconi and A. Kamble. The GMD Data Model and Algebra for Multidimensional Information. In *Proc. of CAiSE'2004*. Springer, 2004.

[6] M.-S. Hacid and U. Sattler. An Object-Centered Multi-dimensional Data Model with Hierarchically Structured Dimensions. In *Proc. of KDEX'1997*. IEEE, 1997.

[7] J. G. J. C. Trujillo, M. Palomar and I.-Y. Song. Designing Data Warehouses with OO Conceptual Models. *IEEE Computer*, 34(12), IEEE, 2001.

[8] R. Kimball. *The Data Warehouse toolkit*. John Wiley & Sons, 1996.

[9] H. J. Lenz and A. Shoshani. Summarizability in olap and statistical data bases. In *Proc. of SSDBM'1997*. IEEE, 1997.

[10] G. H. M. Blaschka, C. Sapia and B. Dinter. Finding Your Way Through Multidimensional Data Models. In *Proc. of DEXA Workshops'1998*. IEEE, 1998.

[11] D. L. Moody and M. A. Kortink. From Enterprise Models to Dimensional Models: A Methodology for Data Warehouse and Data Mart Design. In *Proc. of DMDW'2000*. CEUR-WS, 2000.

[12] F. B. N. Tryfona and J. G. B. Christiansen. starER: A Conceptual Model for Data Warehouse Design. In *Proc. of DOLAP'1999*. ACM, 1999.

[13] T. B. Pedersen. *Aspects of Data Modeling and Query Processing for Complex Multidimensional Data*. PhD thesis, Faculty of Engineering and Science, 2000.

[14] T. B. Pedersen and C. S. Jensen. Multidimensional Database Technology. *IEEE Computer*, 34(12), IEEE, 2001.

[15] P. Vassiliadis and T. K. Sellis. A Survey of Logical Models for OLAP Databases. *SIGMOD Record*, 28(4), ACM, 1999.

[16] X. Yin and T. B. Pedersen. Evaluating XML-extended OLAP queries based on a physical algebra. In *Proc. of DOLAP'2004*. ACM, 2004.