Implementing Operations to Navigate Semantic Star Schemas

Alberto Abelló  
U. Politécnica de Catalunya  
C/ Manuel Girona 1-3  
E-08034 Barcelona  
aabello@lsi.upc.es

José Samos  
U. de Granada  
C/ Daniel Saucedo Aranda s/n  
E-18071 Granada  
jsamos@ugr.es

Félix Saltor  
U. Politécnica de Catalunya  
C/ Manuel Girona 1-3  
E-08034 Barcelona  
saltor@lsi.upc.es

ABSTRACT

In the last years, lots of work have been devoted to multidimensional modeling, star shape schemas and OLAP operations. However, “drill-across” has not captured as much attention as other operations. This operation allows to change the subject of analysis keeping the same analysis space we were using to analyze another subject. It is assumed that this can be done if both subjects share exactly the same analysis dimensions. In this paper, besides the implementation of an algebraic set of operations on a RDBMS, we are going to show when and how we can change the subject of analysis in the presence of semantic relationships, even if the analysis dimensions do not exactly coincide.

Categories and Subject Descriptors

H.2.8 [Database Management]: Database Applications

General Terms

Languages

Keywords

Star schema, OLAP operations, SQL, Drill-across, Semantic Relationships

1. INTRODUCTION

OLAP tools facilitate the extraction of information from the “Data Warehouse”. As defined in [19], OLAP functionality is characterized by dynamic multi-dimensional analysis of consolidated enterprise data supporting end user analytical and navigational activities. In this context, “navigation” means to interactively explore a data cube by drilling, rotating and screening. In [10], we can see that the typical end user operations performed on the data cubes are “roll-up” (increase the level of aggregation), “drill-down” (decrease the level of aggregation), “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 multi-dimensional view). Other authors, like [22] add “drill-across” (combine data cubes that share one or more dimensions) to those operations.

Figure 1: Example of multi-star schema

Multidimensional analysis is based on the separation of factual and dimensional data. Along this paper, we will use the terminology introduced in [2], where a Dimension (subclass of UML Classifier) contains Levels (subclass of UML Class) representing different granularities (or levels of detail) to study data, and a Level contains Descriptors (subclass of UML Attribute). On the other hand, Fact (subclass of UML Classifier) contains Cells (subclass of UML Class), which contain Measures (subclass of UML Attribute). One Cell represents those individual cells of the same granularity that show data regarding the same Fact. One Fact and several Dimensions to analyze it give raise to a Star. As already discussed in [1], we consider that it is important to be able to relate different Stars to facilitate the Drill-across operation. Thus, as we can see in figure 1, we could find two Facts (i.e. Production and Order) sharing Dimensions (i.e. Time and Product). However, this is not the only way to relate Stars. Semantic relationships (like Generalization, Association, Derivation, or Flow) could also appear between both Stars, so that they can be used to “drill-across”, as we will see.

[15] shows how a Star should be implemented on a “Relational Database Management System” (RDBMS), with one table for the Fact and one table for every Dimension, the latter being pointed by “foreign keys” (FK) from the “fact table”, which compose its “primary key” (PK). [18] goes further and shows how some kinds of multi-star schemas
should be implemented. Besides having FK from different “fact tables” pointing to the same “dimension table”, they also allow to have FK in a “fact table” pointing to another “fact table”. If that is the case, the FK between “fact tables” provide the ability to “drill-down” between levels of detail.

Once we have seen how to implement Stars, let’s see the standard SQL’92 template query as presented in [15] (from here on, we will refer to it as cube-query):

```
SELECT LevelID1, ..., LevelIDn, FUNCTION(f.Measure1), ...
FROM Fact f, Dimension d1, ..., Dimension dn
WHERE f.key = d1.ID AND ... AND f.key = dn.ID
AND d1.attr = value AND ...
GROUP BY LevelID1, ..., LevelIDn
ORDER BY LevelID1, ..., LevelIDn
```

The FROM clause contains the “fact table” and the “dimension tables”. These tables are linked in the WHERE clause, which also contains selection conditions defined over the columns of the “dimension tables”. The GROUP BY clause shows the identifiers of the Levels at which we want to aggregate data. Those columns in the grouping must also be in the SELECT clause, besides the Measures aggregated by some SQL function, in order to identify the values in the result. Finally, the ORDER BY clause is explicit to sort the output of the query by these same identifiers.

In spite of the fact that the basic structure of the cube-query is well known, there is not yet a well established set of end user operations to navigate multidimensional data. Some sets of operations have been proposed, as we will see in section 2. However, some of them do not directly map to SQL and, in general, none of them treat “drill-across” and “pivoting”. As shown in section 4, these operations can be smoothly translated to modifications on the cube-query. Finally, section 5 shows the implementation of new semantic possibilities to drill across, and section 6 concludes the paper.

2. RELATED WORK

In the last years, lots of work have been devoted to modeling multidimensionality (i.e. [17], [4], [11], [8], [12], [7], [27], [16], and [21]). Each one of these models offers an algebraic set of operations (some of them also offer a calculus). However, none of them offers the translation of the 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 “Relational OLAP” (ROLAP) tools in the market contradicts that, and outlines the importance of research on improving the usage of SQL to query multidimensional data.

[24] presents an end user oriented algebra of multidimensional operations. Nevertheless, it is neither translated to SQL nor considers drilling across, nor any kind of semantic relationship. An approach limited to operations over Dimensions is in [14]. In this case SQL is extended to facilitate handling dimensional data. Obviously, since it focuses on Dimensions, “drill-across” is not even mentioned.

Semantic relationships are often underestimated, as we can see in [5], whose methodology for multidimensional design proposes the transformation of generalizations into aggregations and classes. Some few conceptual models, [26] and [25], allow the representation of semantic relationships. However, these neither present a set of operations to manipulate data, nor study their usage to drill across.

Some models offer a “join” operation that would allow some kind of “drill across”. Nevertheless, this operation is far away from end user multidimensional concepts, and the benefits of semantic relationships are not explored in any case.

3. A MULTIDIMENSIONAL ALGEBRA

In this section we are going to see the algebraic operations of YAM$^2$ (a multidimensional model presented in [2]), which focus on identifying and uniformly manipulating sets of data, namely Cubes.

**Definition 1.** A Cube is an injective function from an n-dimensional finite space (defined by the cartesian product of n functionally independent Levels \{L$_1$,...,L$_n$\}, to the set of instances of a Cell (C$_c$).

$$c : L_1 \times \ldots \times L_n \rightarrow C_c, \text{ injective}$$

We generally say that a query is from (or over) its input schema to its output schema. Thus, there exists an input n-dimensional Cube (c$_i$), and we want to obtain an output n-dimensional Cube (c$_o$). Since, we defined a Cube as a function, operations must transform a function into another function.

![Figure 2: Multidimensional operations as composition of functions](image)

As shown in figure 2, we have three families of functions (i.e. f, g, and h), that can be used to transform a Cube. Obtaining c$_o$, from c$_i$ can be seen as mathematical composition of functions (c$_o$ = f o c$_i$ o g, with f and g belonging to the families of functions f and g, respectively). **Relations** in section 5 can be used for this purpose. Those functions of the family h define aggregation hierarchies and are used to roll data up.

**ChangeBase:** This operation reallocates exactly the same instances of a Cell in a new n-dimensional space with exactly the same number of points, by composing the Cube with a function of the family of functions f. Thus, it actually modifies the analysis dimensions used. Functions relating different Dimensions belong to the family f.

$$\phi : L_1 \times \ldots \times L_m \rightarrow L_1 \times \ldots \times L_m, \text{ injective}$$

$$c_o(x) = \gamma_\phi(c_i) = c_i(\phi(x))$$
Drill-across: This operation changes the image set of the Cube by means of an injective function \( \psi \) of the family \( g \). The n-dimensional space remains exactly the same, only the cells placed in it change. Functions relating instances of different Facts belong to the family \( g \).

\[ \psi : C^g_e \rightarrow C^g_s, \text{ injective} \]

\[ c_0(x) = \delta \psi (c_i) = \psi(c_i(x)) \]

Dice: By means of a predicate of points of interest out of the whole n-dimensional space.

\[ c_0(x) = \sigma P(c_i) = \begin{cases} 
   c_i(x) & \text{if } P(x) \\
   \text{undef} & \text{if } \neg P(x)
\end{cases} \]

Projection: This just selects a subset of Measures from those available in the Cube.

\[ c_0(x) = \pi_{m_1 \ldots m_k}(c_i) = c_i(x)[m_1, \ldots, m_k] \]

Roll-up: It groups cells in the Cube based on an aggregation hierarchy. This operation modifies the granularity of data, by means of an exhaustive function \( \varphi \) of the family \( h \) (i.e. \( \varphi \) relates instances of two Levels in the same Dimension, corresponding to a part-whole relationship).

\[ c_0(x) = \rho \varphi (c_i) = \bigcup_{\varphi(y)=x} c_i(y) \]

Union: Similar to operations between functions (if \( f \cdot g = f(x) \cdot g(x) \)), we can also define operations between Cubes, if both are defined over the same domain (n-dimensional space). By means of this operation we can recover the cells removed by means of Dice.

\[ c_1 \oplus c_2 = c_1(x) \oplus c_2(x) \]

In the sense of [3], these operations are conceptually a "procedural language", because queries are specified by a sequence of operations that construct the answer. For instance, with this set of operations, we can derive Slice (which reduces the dimensionality of the original Cube by fixing a point in a Dimension) by means of Dice and ChangeBase operations.

\[ c_0(x) = \text{slice}_{L_i=k}(c_i) = \gamma L_1 \times \ldots \times L_{i-1} \times L_{i+1} \times \ldots \times L_n(\sigma_{L_i=k}(c_i)) \]

Drill-down (i.e. the inverse of Roll-up) is not defined, because as argued in [12], we can only apply it, if we previously performed a Roll-up and did not lose the correspondences between cells. This can be expressed as an "undo" of Roll-up, or if we do not want to keep track of results, by means of views over the atomic data as in [27]. Therefore, it cannot be part of a true sequence of operations. The same could be said for Dice and Projection. If all we have to answer a query is the current Cube, we can neither recover cells (lost by dicing) nor Measures (lost by projecting). Nevertheless, while the only solution to Drill-down is to throw away the current Cube and go to the source, we can keep our Cube and add diced cells by means of Union and projected Measures by means of a sort of reflexive Drill-across to the same Fact.

4. TRANSLATING OPERATIONS TO SQL

In this section we are going to show the translation of those algebraic operations to modifications over the cube-query introduced in section 1.

\[ A := \sigma_{\text{Time.year}=2003} \]

\[
\text{SELECT} \ d_1\ . \ product,\ d_2\ . \ day,\ d_3\ . \ retailer,\ d_4\ . \ client,\ \text{Sum}(f\ . \ unitsSold) \\
\text{FROM} \ f,\ Product\ d_1, \ \text{Time}\ d_2,\ Retailer\ d_3,\ Client\ d_4 \\
\text{WHERE} \ f\ . \ product=d_1\ . \ product\ \text{AND} \ f\ . \ day=d_2\ . \ day \\
\text{GROUP BY} \ d_1\ . \ product,\ d_2\ . \ day,\ d_3\ . \ retailer,\ d_4\ . \ client \\
\text{ORDER BY} \ d_1\ . \ product,\ d_2\ . \ day
\]

\[ B := \sigma_{\text{Client.All}(\text{Retailer.All}(A))} \]

\[
\text{SELECT} \ d_1\ . \ product,\ d_2\ . \ day,\ \text{"All",\ "All"},\ \text{Sum}(f\ . \ unitsSold) \\
\text{FROM} \ f,\ \text{Product}\ d_1, \ \text{Time}\ d_2 \\
\text{WHERE} \ f\ . \ product=d_1\ . \ product\ \text{AND} \ f\ . \ day=d_2\ . \ day\ \text{AND} \ d_2\ . \ year=2003 \\
\text{GROUP BY} \ d_1\ . \ product,\ d_2\ . \ day \\
\text{ORDER BY} \ d_1\ . \ product,\ d_2\ . \ day
\]

\[ C := \gamma_{\text{Product}\times\text{Time}}(B) \]

\[
\text{SELECT} \ d_1\ . \ product,\ d_2\ . \ day,\ \text{Sum}(f\ . \ unitsSold),\ SUM(f\ . \ unitsProduced) \\
\text{FROM} \ f,\ \text{Product}\ d_1, \ \text{Time}\ d_2 \\
\text{WHERE} \ f\ . \ product=d_1\ . \ product\ \text{AND} \ f\ . \ day=d_2\ . \ day\ \text{AND} \ d_2\ . \ year=2003 \\
\text{GROUP BY} \ d_1\ . \ product,\ d_2\ . \ day \\
\text{ORDER BY} \ d_1\ . \ product,\ d_2\ . \ day
\]

\[ D := \delta_{\text{Product}}(C) \]

\[
\text{SELECT} \ d_1\ . \ product,\ d_2\ . \ day,\ \text{Sum}(f\ . \ unitsSold),\ \text{SUM}(f\ . \ unitsProduced) \\
\text{FROM} \ f,\ \text{Product}\ d_1, \ \text{Time}\ d_2 \\
\text{WHERE} \ f\ . \ product=d_1\ . \ product\ \text{AND} \ f\ . \ day=d_2\ . \ day\ \text{AND} \ d_2\ . \ year=2003 \\
\text{GROUP BY} \ d_1\ . \ product,\ d_2\ . \ day \\
\text{ORDER BY} \ d_1\ . \ product,\ d_2\ . \ day
\]

\[ E := \pi_{\text{unitsProduced}}(D) \]

\[
\text{SELECT} \ d_1\ . \ product,\ d_2\ . \ day,\ \text{SUM}(f\ . \ unitsProduced) \\
\text{FROM} \ f,\ \text{Product}\ d_1, \ \text{Time}\ d_2 \\
\text{WHERE} \ f\ . \ product=d_1\ . \ product\ \text{AND} \ f\ . \ day=d_2\ . \ day\ \text{AND} \ d_2\ . \ year=2003 \\
\text{GROUP BY} \ d_1\ . \ product,\ d_2\ . \ day \\
\text{ORDER BY} \ d_1\ . \ product,\ d_2\ . \ day
\]

\[ F := E \cap \gamma_{\text{Product}\times\text{Time}}(\text{Factory.All(\text{Order}(A))}) \]

\[
\text{SELECT} \ d_1\ . \ product,\ d_2\ . \ day,\ \text{SUM}(f\ . \ unitsProduced) \\
\text{FROM} \ f,\ \text{Product}\ d_1, \ \text{Time}\ d_2 \\
\text{WHERE} \ f\ . \ product=d_1\ . \ product\ \text{AND} \ f\ . \ day=d_2\ . \ day\ \text{AND} \ d_2\ . \ year=2003\ \text{OR} \ d_2\ . \ year=2002 \\
\text{GROUP BY} \ d_1\ . \ product,\ d_2\ . \ day \\
\text{ORDER BY} \ d_1\ . \ product,\ d_2\ . \ day
\]

Figure 3: Sequence of operations

Taking into account that end users desire to navigate from Cube to Cube, the idea is to consider that last query (or its partial results) has been materialized (or kept in memory), so that we can use it to solve the next one. In figure 3 we see a sequence of operations, and how they affect the cube-query step by step. Notice that one Cube could always be used in the obtaining of the next one.

- Dice selects the desired points by anding the corresponding predicate over Descriptors to the WHERE clause. The new predicate to be anded can only regard grouping attributes or attributes that functionally depend on them. In the example, \( d_2 \ . \ year=2003 \) is added to the WHERE clause.
- Roll-up changes the identifiers in the GROUP BY clause by those of the Levels above. The SELECT and ORDER BY clauses must be modified appropriately, so that the Descriptors coincide in all three. To roll
up to Level All, all Descriptors of a Dimension are removed from the GROUP BY, and “All” is placed in the corresponding position in SELECT clause. In the example, two Roll-ups are performed up to Level All along Retailer and Clients, so that no column of the corresponding tables is present neither in the GROUP BY nor in the ORDER BY nor SELECT clause, where they are substituted by ‘‘All’’.

- **ChangeBase** allows two different kinds of changes in the base of the space. Firstly, we can just rearrange the multidimensional space (B × A instead of A × B) by modifying the order of Level identifiers in ORDER BY and SELECT clauses (this would be equivalent to the “pivot” operation). Moreover, this operation extends “pivoting” functionality, because if there exist more than one set of Dimensions that identify the points in the space, we can change the Dimensions, by just adding the new “dimension tables” to the FROM and the corresponding links to the WHERE clause. Identifiers in the SELECT, ORDER BY and GROUP BY clauses must be replaced. For instance, if we are analyzing inventory transactions, our space would be defined by Product×Time×Order, but since one vendor only places one order per warehouse per day. In the example, since two Dimensions already contain only one point, we can just remove the ‘‘All’’ from the SELECT clause to convert a four-dimensional space into a two-dimensional one, we could also see data in the space Product×Time×Day×Vendor×Warehouse.

- **Drill-across** changes the subject of analysis by adding a new “fact table” to the FROM, its Measures to the SELECT, and the corresponding links to the WHERE clause. The links added will depend on the semantic relationship used to Drill-across, as we will see in section 5. In general, if we are not using any Relationship, a new “fact table” can always be added to the FROM clause if the attributes composing the identifier of the desired Cell point to the already used “dimension tables”. In the example, a new Measure unitsProduced is added to the SELECT clause, the “fact table” Production to the FROM, and the corresponding links to the WHERE clause.

- **Projection** removes Measures from the SELECT. If there is not any Measure left, COUNT is assumed. In the example, the Measure of Order table is removed (since the table is then useless, it is also removed).

- **Union** unites two Cubes if their spaces exactly coincide, which translated to the cube-query means that Levels in SELECT, GROUP BY, and ORDER BY clauses must coincide. Therefore, to unite two cube-queries both WHERE clauses just need to be used appropriately. In the example, by means of Dice, Roll-up, and ChangeBase, we obtain a Cube compatible to the existing one. Afterwards, we can or both select conditions in the same WHERE clause.

Let’s analyze now the properties of this set of operations regarding the cube-query:

**Property 1.** The algebra composed by these operations is closed (i.e. they operate on cube-queries and, since none of them modifies the structure of the query, the result of all operations is always a cube-query).

**Property 2.** The algebra composed by these operations is complete (i.e. since any clause can be modified, any valid cube-query can be computed as the combination of a finite set of operations applied to the appropriate Cube). Table 1 summarizes the effects of the different operations:

**SELECT Measures** can be added and removed. Descriptors actually need to be replaced to keep the size of the space. They can be replaced based on aggregation hierarchies or Dimension relationships.

**FROM** Dimension and fact tables can be added depending on the existing semantic relationships in the multidimensional schema. We consider that any table is automatically removed if after an operation it does not affect the result of the query (see figure 3, where Order is removed after Projection, and Client and Retailer are removed after Roll-up).

**WHERE** Links as well as conditions can be added. Unnecessary links are also removed when the corresponding table is. By means of semantic optimization techniques, unnecessary conditions over Descriptors can also be removed. Just notice that the predicate can be restricted by means of Dice and relaxed by means of Union.

**GROUP BY** Columns can be replaced and eventually removed (rolling up to All) from GROUP BY clause. The groups can always be fused, but never split, because as explained before we do not consider Drill-down. If we would consider such operation, they could.

**ORDER BY** Their columns exactly correspond to those Descriptors in the SELECT clause. Therefore, they are modified as the former are, being able to sort them by means of ChangeBase.

**Property 3.** The algebra composed by these operations is minimal (i.e. none can be expressed in terms of others, nor can any operation be dropped without affecting its functionality). Roll-up and Drill-across affect the same clauses, but the modifications are based on aggregation hierarchies and Dimension relationships respectively. Regarding the cube-query, since some operations affect more than one clause, these are not atomic. However,
they represent the basic end user multidimensional concepts, and if more than one clause is affected by the same operation, it is just to keep the cube-query structure (remember, for instance, that attributes in SELECT, GROUP BY and ORDER BY clauses must coincide in a cube-query, and tables must be linked).

5. NEW DRILL-ACROSS POSSIBILITIES

In [15], we can see that we can use two “fact tables” together if the common dimensions are exactly the same. In [1], we systematically showed how and which semantic relationships can be used to relate multidimensional constructs. Semantic relationships in the multidimensional schema define functions between Classes. By composing those functions appropriately, we can obtain the desired vision of data. If we want to analyze instances of a given Class in the space defined by the cartesian product of a set of Classes, all we have to do is to find the appropriate composition of functions. If that path of functions exists, we can analyze data in the desired way.

\[ X = \gamma \text{Product} \times \text{Retailer} \times \text{Client} (\sigma \text{Time: A}ll (\sigma \text{Time.year} = 2003 (\text{Order}))) \]

SELECT \text{d}_4 \text{product}, \text{d}_3 \text{retailer}, \text{d}_4 \text{client}, \text{Sum}(f \text{unitfield})
FROM \text{Order} l, \text{Product} d_1, \text{Time} d_2, \text{Retailer} d_3, \text{Client} d_4
WHERE f \text{product} = d_1 \text{product} AND f \text{day} = d_2 \text{ID}
AND f \text{retailer} = d_3 \text{retailer} AND f \text{client} = d_4 \text{client} AND d_2 \text{year} = 2003
GROUP BY d_4 \text{product}, d_3 \text{retailer}, d_4 \text{client}
ORDER BY d_4 \text{product}, d_3 \text{retailer}, d_4 \text{client}

Figure 4: Example of condition kept on otherwise unused Dimensions

Our approach is more powerful than just sharing “dimension tables”, because it allows to drill-across even if those tables do not exactly coincide. Moreover, since ChangeBase and Drill-across do not remove tables from the FROM clause, but link new tables to the existing ones, we can, for instance, keep conditions over Dimensions or Levels that do not participate in the definition of the space. As exemplified in figure 4, the Dice puts a condition on Time.year Level, and even after the data is rolled up above that Level and the Dimension is removed from the space by means of the ChangeBase, the condition is kept in the WHERE clause.

Figure 5: UML Relationships between model elements

UML, in [20], provides four different kinds of Relationships: Generalization, Flow, Association, and Dependency. As depicted in figure 5, Generalization relationships relate two GeneralizableElements, one with a more specific meaning than the other. Any kind of Classifier is a GeneralizableElement. Flow relationships relate two elements in the model, so that both represent different versions of the same thing. Association, as specified in UML, defines a semantic relationship between Classifiers. Finally, UML allows to represent different kinds of Dependency relationships between ModelElements like Binding, Usage, Permission, or Abstraction. We are not going to consider the three first, because they are rather used on application modeling. Moreover, due to the same reason, out of the different stereotypes of Abstraction we are only going to use Derivation. Derivability, also known as “Point of View”, helps to represent the relationships between model elements in different conceptions of the UoD.

We are going to see now how these kinds of Relationships would be implemented on a relational star schema, and how they would be used to either change the base of the space or drill across subjects (notice that we do not forbid to drill across when “dimension tables” exactly coincide, but open new possibilities to do it). On the one hand, if two sets of Dimensions are semantically related, we may be able to change the base. On the other hand, if two Facts are semantically related, we may be able to drill across.

5.1 Derivation

Derivation would be implemented on a RDBMS by means of views (in this section we only consider updatable views, so that we can identify each tuple in the view with its counterpart in the table). We can find that a “dimension table” is a view over either another “dimension table” or “fact table”, and a “fact table” could be a view over another “fact table”. A “fact table” cannot be a view over a “dimension table”, because Facts represent measured data.

Firstly, we could find that the “dimension table” (D\(_i\)) in the space of the input cube (c\(_i\)) is a view over the “dimension table” (D\(_o\)) in the space of the output cube (c\(_o\)). In this case, we can change the base of the space adding D\(_o\) to the FROM clause and linking it to D\(_i\) by appropriately equaling the identifiers of the table and the view (the PK of D\(_i\) should have been derived from attributes in D\(_o\)). However, if D\(_o\) was derived from D\(_i\), we would only be able to change the base of the space if the WHERE clause of the cube-query corresponding to c\(_i\) is subsumed by the view predicate. Otherwise, we will find points in the space of c\(_i\) without counterpart in the space of c\(_o\) (we would lose points in the analysis space).

As Dimensions, Facts can also be related by derivation. If the “fact table” (F\(_i\)) of c\(_i\) is a view over the “fact table” (F\(_o\)) of c\(_o\), we can add F\(_o\) to the FROM clause and link the identifiers of the table and the view (as before, the PK of F\(_i\) should have been derived from attributes in F\(_o\)). In the other way, if F\(_o\) is derived from F\(_i\), we can still link them. However, if some rows of F\(_i\) do not belong to its view F\(_o\), completely empty cells will appear in c\(_o\). We should perform an outer join to keep, at least, the Measures of F\(_i\) in the output.

Finally, the “Pull” operation in [4] could be obtained by ChangeBase, if D\(_i\) is a view over F\(_i\). This would allow to change to a new space based on the Measures in the current, by directly linking D\(_o\) to F\(_i\) in the WHERE clause. Notice that this Relationship can only be used if the new set of Dimensions form a base for the same space (we should probably change more than one Dimension at once). The counterpart “Push” operation would be obtained by rolling up to Level All along the pushed Dimension and drilling across to the Fact that was used in the derivation of the
Dimension. However, this is the classic Drill-across, where “dimension tables” must be shared, and would not really need the Derivation relationship to be performed.

5.2 Generalization

Even though an specific syntax has been defined in [13] and new techniques experimented in [6], without loss of generality, we assume that Generalizations would be implemented on a RDBMS with one table for the superclass, and another table for each of the subclasses. The PK of each subclass would point to that of the superclass. We argued in [1] that Generalizations can only be found between either two Dimensions or two Facts. Dimensions and Facts are so different, that they can only be related by Derivation or Association.

If D_o is a superclass of D_i, we will always be able to change the base of the space by adding the new table and linking it to its subclass. On the other hand, if D_i is superclass of D_o, we can only change the base if the specialization criterion of D_o subsumes the condition of the WHERE clause of c_i.

Regarding Generalization between Facts, we can always Drill-across from F_i to F_o, if F_o is superclass of F_i. If F_i is superclass of F_o and the specialization criterion does not subsume the WHERE condition in c_i, then it will be necessary to use an outer join to keep on obtaining the Measures in F_i. If the Generalization is part of a partition, an alternative to the outer join would be to unite c_o, to the result of drilling across to the other subclasses of F_i in the partition.

5.3 Association

The implementation of Associations on a RDBMS depends on their multiplicities. If the multiplicity is one-to-one or one-to-many, they can easily be implemented by means of a FK. If the multiplicity is many-to-many, they can be implemented using a “bridge table”. Associations exist between two Dimensions, two Facts or a Fact and a Dimension.

If there is a one-to-one Association between D_i and D_o, it will always be possible to link D_o to D_i, and substitute the corresponding attributes in the SELECT clause of the cube-query, and the set of Dimensions will still be a base of the space. If the multiplicity is one-to-many or many-to-many and we replace D_i by D_o, the size of the space would not be preserved. Nevertheless, these kinds of Associations could still be used if we replace more than one Dimension at once, and there exist such one-to-one relationship between both sets of Dimensions. For example, there is a one-to-many association between Day and Order, but a one-to-one between Day × Vendor × Warehouse and Order, as explained before.

Between two Facts, again, there is not any problem if the multiplicity of the Association is one-to-one. If not, we do not have an injective function as required to perform the Drill-across. If we have more than one instance of F_o per instance of F_i, we should Drill-across to an upper aggregation level of F_o, where the correspondence were one-to-one. On the other hand, if we have more than one instance of F_i per instance of F_o, we would get the same data more than once, placed at different points in the analysis space, giving rise to a double-counting problem. Moreover, if minimum multiplicity of the association is zero, i.e. if we could find instances of F_i associated with zero instances of F_o, we should use the outer join in order to keep the Measures of F_i in c_o.

The most common multiplicity between Dimension and Fact is one-to-many. However, in some special cases, we could find many-to-many Associations. [23] analyzes the different existing possibilities to implement such Associations between Dimensions and Facts on a RDBMS. Nevertheless, using them during navigation would mean that the same cell should be placed at different points in the space, giving rise again to a double-counting problem (our Cube would not be injective). This problem is similar to the Drill-down problem, where we should decide how cells are decomposed into different parts. [23] proposes a weighting factor to solve this case. Thus, we should place the “bridge table” and “fact table” in the FROM clause, link them appropriately, and weight the Measures in the SELECT clause.

5.4 Flow

This kind of Relationship should be implemented again by means of FK between old and new versions of tuples. As it was said before, a Dimension cannot eventually evolve into a Fact, nor vice-versa.

The simple evolution case is when every instance in the old Dimension evolved into exactly one instance in the new Dimension, and no new instances appeared. We just need to add D_o to the FROM clause and link both tables appropriately. If there is not such one-to-one correspondence between old and new instances, we should use “transformation matrices” (similar to the “weighting factor” of many-to-many Associations) as explained in [9] (notice that in this case we could be modifying the number of points in the space, nevertheless we consider this an exception to the rule, because the Dimension and Level do not actually change). If D_i is the old “dimension table” and some of its instances disappeared in the new version D_o, we need to assure that they are not selected before performing ChangeBase. The same happens if D_o is the old version of D_i and new instances appeared in the evolution, these instances should be removed from the space before the ChangeBase could be performed.

Drilling across by means of a Flow between two Facts means analyzing the old one from the new point of view, or vice-versa. If instances appear or disappear in the evolution, we should use the outer join appropriately to avoid loosing the Measures of F_i in c_o. Moreover, Drill-across using Flow between the Facts should only be used if there is a one-to-one correspondence between instances of new and old Facts. Notice that if there exists a one-to-many correspondence (instances were either fused or split during the evolution process), then it is due to the same happened to the Dimensions, because it is necessary to have new PK values to identify the new instances of the Fact. Thus, we should firstly change the base to that of F_o using Flow Relationships between the Dimensions, so that we would not need to use the Flow between the Facts to perform Drill-across.

6. CONCLUSIONS

This paper presents a set of algebraic operations to navigate multidimensional schemas. Each of these operations can be smoothly translated to SQL. Two operations stand out from the rest, i.e. Drill-across and ChangeBase, whose functionality has no counterpart in other models. They work on semantic relationships between different Stars and were not treated as first class citizens in any other multidimensional model before. ChangeBase operation extends the well known “pivoting” functionality, so that it can be used as
a step towards Drill-across. Thus, it is shown how we could drill across not only if “dimension tables” are shared, but also if either Dimensions or Facts are related by different kinds of UML Relationships (i.e. Derivation, Generalization, Association, and Flow).

In our navigational approach for building cube-queries, conditions in the WHERE clause are not explicitly removed. This allows to keep conditions when rolling-up and drilling-across, which offers the possibility of placing conditions on Levels and Dimensions that do not form the space of the analyzed cube. We assume that unnecessary conditions, links and tables are removed by means of semantic optimization mechanisms. As future work, we plan to study the implementation of such mechanisms, as well as how SQL’99 could improve the implementation of the Relationships.

Acknowledgements
This work has been partially supported by the Spanish Research Program PRONTIC under projects TIC2000-1723-C02-01, and TIC2000-1723-C02-02.

7. REFERENCES