

On Revising Data Cubes with Full Multidimensional Exceptions

Leonardo Terribile

IBM Argentina
terribil@ar.ibm.com

Mauricio Minuto Espil

Universidad de Buenos Aires
mminuto@ub.edu.ar

Keywords: Data Warehousing, Multidimensional Databases, OLAP, Belief Revision, Update Languages.

Abstract

Information in a data warehouse does not always reflect unquestionable facts in an organization. Sometimes, data should be considered, at least partially, as representing simple beliefs about the state of the world it intends to model. Thus, data in the warehouse may not be entirely reliable, and could be subject to revision. In this paper we address the *data cube revision problem* focusing into the situation when the definition of an exception to a roll-up function of one dimension is needed, and this exception actually depends on the values appearing in other dimensions of the cube. We present an extension to the revision language IRAH and introduce a modified version of an existing efficient revision algorithm for applying extended IRAH sentences to a data cube.

1 Introduction

A multidimensional database is usually represented as sets of facts and dimensions that help understanding the facts. Facts are seen as points in a multidimensional space, with one coordinate in each dimension, and an associated set of *measures*. Dimensions provide appropriate contextual meaning to facts, and are usually organized in hierarchies, thus supporting different levels of data aggregation. A *dimension schema* is represented as a directed acyclic graph, where the nodes are the dimension levels, with a unique bottom level, and a unique distinguished top level, denoted *All*. Each level is associated with a set of coordinate values. An *instance* of a dimension is given by a functional relationship defined extensionally between the coordinates in two consecutive levels in the graph.

Facts in a multidimensional database do not always reflect *actual* facts that have occurred in the real world. There are cases where data should better be considered as beliefs the organization responsables have about the state of the real world. Thus, data in the warehouse may not be entirely reliable, and could be subject to periodical revision based on the analyst’s information, or mere intuition. Functional dependencies between coordinates in consecutive levels may hold on almost the total number of paths in a dimension instance, although some exceptions may arise. As an example, let us consider a time dimension hierarchy, where Mondays are usually considered to be business days. Suppose, however, that a general failure in power supply occurred on February 4th 2002 –which was a non-holiday Monday, and hence an ordinary business day - affects some area in the city of Los Angeles, where store S1 of our company is located, preventing the normal operation of the store that day. If we have a category *non-business day* in the time hierarchy, an analyst would rather classify all February 4th 2002 measures for store S1 as aggregating on category *non-business day*, instead of the ordinary classification for Mondays on category *business day*. In this setting, we claim that assertions like “Mondays are business days” should be considered mere beliefs, not indisputable facts. Consequently, once a hierarchy is built upon this kind of inductive or imprecise knowledge and data have been loaded into dimension instances, exceptions are likely to appear. A mechanism of belief revision capable of altering the contents of dimension instances is therefore needed, preserving summarizability under uncertainty, in order to guarantee accurate analysis results. This revision will also affect the data cubes defined over such dimensions.

Let show another motivating example: Let us suppose, extending the example presented in [MEVT02], where a credit company which has offices nation wide, maintains a multidimensional database holding information about credit applications, including the loan identification, the borrower identification, the branch and the date on which the application has been issued, and the requested amount. The table in Figure 1 shows an instance of the database. Here, *custId*, *branchId* and *date* represent the bottom levels of dimensions Customer, Geography and Time, respectively. *Amount* is the measure. Figure 2 depicts the schema of dimension Customer, with its different levels of aggregation, and shows an instance of this dimension. Each level in the dimension is described by attributes. Thus, *custId* stands for the identification of the customer –SSN, for instance–; each customer has associated a value in level *delay* indicating the number of months elapsed from the last payment of any of her debts. Each value has a corresponding legal financial *grade* for the customer –with values S-standard, N-normal, R-risky, P-procrastinator, U-totally unreliable–. Finally, level *grade* is assigned a *status*, indicating whether or not the customer’s applications are usually accepted for consideration. Dimension Geography is constituted by levels *branchId* that identifies the brach of the company, the city where the branch is set and the region that includes the city. Dimension Time is made out of levels *Date*, *Month* and *Year* with the usual meaning.

appId	custId	branchId	date	amount
<i>a</i> ₁	<i>c</i> ₁	LA1425	02/11/2002	15,000
<i>a</i> ₂	<i>c</i> ₂	LA1425	01/10/2002	3,000
<i>a</i> ₃	<i>c</i> ₃	WA0854	02/07/2002	250,000
<i>a</i> ₄	<i>c</i> ₆	NY1545	01/28/2002	40,000
<i>a</i> ₅	<i>c</i> ₁₂	SD9854	01/30/2002	35,000
<i>a</i> ₆	<i>c</i> ₁₄	WA4566	02/04/2002	165,000
<i>a</i> ₇	<i>c</i> ₁₇	WA4566	02/04/2002	85,000

Figure 1: Credit Applications

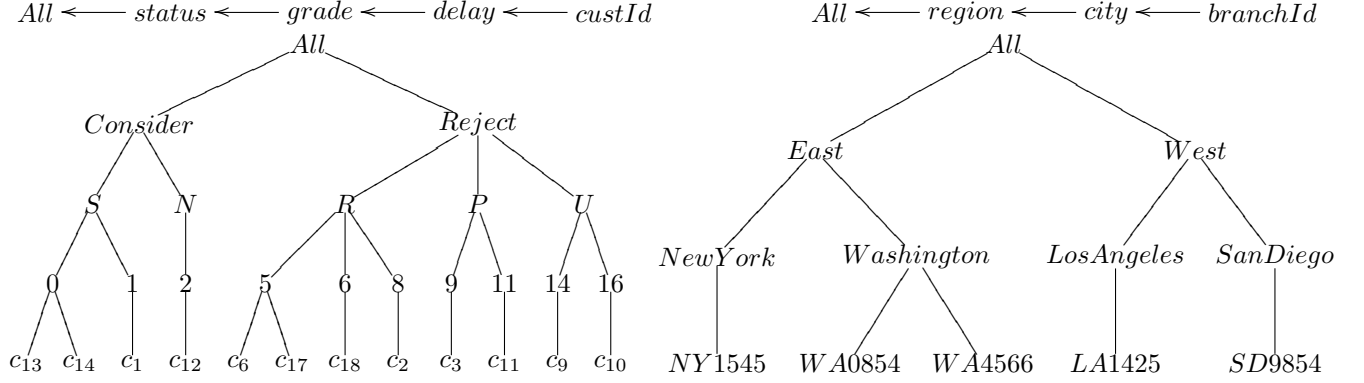


Figure 2: Dimensions *Customer* and *Geography*.

Example 1 Consider the materialized cube view –let us denote it *StatusByRegion*–, aggregated by status and region, shown in Figure 3(a) (Dimension Time has been suppressed by rolling it up to level *All*). Let us also suppose that a chief analyst with control over all operations traded in New York City is not actually convinced by the rule indicating that credit applications from customers with five months elapsed from their last payment should be considered for rejection, although she rather accept the rule that grades those customers as risky, because the rule is mandatory according to the law. An exception is therefore needed, asserting that applications from customers with a delay of 5 months should acquire the *Consider* status, but the exception applies only if the loan has been traded in the offices located in New York City. This exception affects the contents of the given cube. The new cube contents are shown in Figure 3(b). The modified customer dimension for a loan traded in New York City is depicted in Figure 4. As the reader can see, only the value 40,000, corresponding to a loan application done by customers with a five-month delay from a New York City’s office, has been aggregated with a consider status.

	West	East
Consider	50,000	165,000
Reject	3,000	375,000

	West	East
Consider	50,000	205,000
Reject	3,000	335,000

Figure 3: (a) Cube view aggregated by *status* and *region*. (b) Revised cube view.

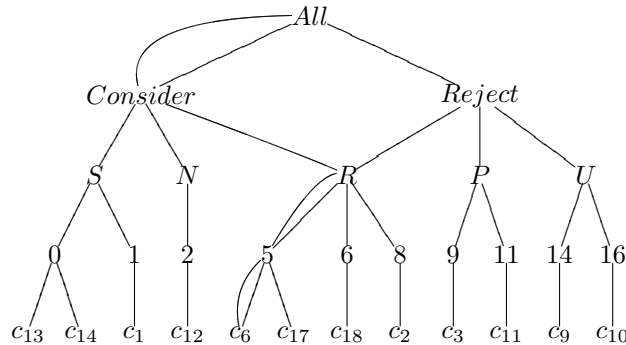


Figure 4: Revised dimension *Customer*.

Accepting an exception like the one presented above, once the design phase of the warehouse has been completed and data has been loaded, implies that an structural update ([HMOV99a]) is needed in order to modify the design of the multidimensional database, and these structural change may entail the whole

recomputation of every data cube present in the warehouse. Because the situations described in cases like the examples above are in fact exceptional, a redesign would result extremely expensive in terms of time needed for data downloading and reloading. Moreover, because the non-conclusiveness nature of the analyst attitude in the latter example, it turns out to be inappropriate to consider the new design as stable, thus the original cube views contents should be preserved. In [MEV01] the problem of computing revised dimension instances that incorporate single dimension exceptions, with the structural part (the schema) of the dimension hierarchy remaining unaltered, is addressed, treating the revised paths as exception paths, and materializing them separately from ordinary paths. In that work a language is introduced, called IRAH –standing for Intensional Redefinition of Aggregation Hierarchies–, suitable for defining exceptions in an intensional fashion. Later, in [MEVT02], the IRAH language is presented from the perspective of a language intended to maintain cube views under revision by single dimension exceptions introduced in the instance of the dimension hierarchy. Clear semantics are defined there for the data cube revision problem, based on a non-monotonic reasoning framework. In addition, a set of efficient algorithms – with low complexity each – is presented, that revise the contents of a given data cube, according to exceptions defined by IRAH statements, producing exception cubes. The algorithms identifies the cells in the cube which must be modified, and updates these cells –*i.e.* revises the cube– obtaining the aggregated data from the closest materialized cells. A more recent work [MEV03] thoroughly investigate how single dimension exceptions affects the contents of dimension instances and data cubes, that is, how dimension instances and exception cubes should be computed when exceptions defined on a single dimension arise. In the present work, we focus our attention on the situation occurring when an exception that involves more than one dimensions occurs, thus affecting a subset of the cells in the cube bounded by the coordinates of each dimensions mentioned in the exception, as the previous examples show. We call this new problem the *full dimension exception data cube revision problem*. The remainder of the paper is organized as follows: in Section 2 we briefly survey on related work in the belief revision area. Section 3 revisits the IRAH language, extending its syntax in order to allow defining multiple dimensions exceptions. Then, we present a rewriting technique that translates a multiple dimensions exception into a set of single dimension exceptions, and define a clear semantics for revision under this setting. A set of algorithms for revising dimension instances and data cubes, that modifies those presented in [MEVT02], is produced in Section 4. We conclude in Section 5.

2 Related Work

Rollup functions in dimension hierarchies have been extensively addressed in the data warehousing literature [HVM99a, HVM99b, Vai01, VRC02]. Efforts have been made to produce conceptual meaning for rollups, modeling them as parent-child relationships [LAW98], as logical containment (greater than) relationships [PJ99], or as whole-part relationships [ASS01]. ISA relationships, which are specially suited for modeling different levels at which a collection of facts can be viewed, are not addressed in these works. Uncertainty may appear due to, data in a data warehouse may sometimes represent mere beliefs, not unquestionable assertions. That is the reason why ISA relationships may exhibit exceptions [ER83].

Belief revision has been a topic of research in the area of non-monotonic reasoning for Artificial Intelligence. Formally, a revision is a function that maps an epistemic state (a logic theory or set of logic sentences) and a newly acquired belief (a sentence), to a new revised epistemic state. Revisions satisfy the well-known eight AGM postulates of minimal change [CGD85]. Brewka [Bre00] claims that the mechanics of revision should not be differentiated from a process consisting in giving the new belief to some (non-monotonic) inference system, letting the system work, contrasting it with the current epistemic state. This approach leads to simpler algorithms, and results useful for identifying common situations where the revision problem become tractable computationally.

In previous work [MEV01], we introduced IRAH, a rule-based language for maintenance of dimension hierarchies allowing defining exceptions. We view rollups as logical rules instead of functions. We presented a stratified semantics for the revision in order to obtain an algorithm for revision with low complexity (linear in most cases) while maintaining the least possible deviation from the AGM postulates [BFH98]. Previous works on belief revision [WA98, ABGM99], and theory repairing [ABC00], were taken into consideration for this approach. In a more recent work [MEVT02] we defined clear semantics for the data cube revision problem and presented a set of efficient algorithms that revise the contents of a given data cube.

3 Extending the IRAH Language

Dimension instances are usually presented in the form of rollup functions, *-i.e.* interpreted as mappings between consecutive level instances-. Rollup functions, can be regarded, from a rule based perspective, as ordinary aggregation rules. A rule language, called IRAH *-Intensional Redefinition of Aggregation Hierarchies-*, has been introduced in [MEV01], such that a revision operation on a single dimension can be defined as consisting of a set of exception rules with a priority defined that enforces their application over the application of ordinary aggregation rules (roll-ups).

IRAH is a typed language; variables and constants are typed. Types in IRAH are: **D** for dimension type; **L** for level type; **C** for coordinate type. Usual basic types like integers, booleans, characters, strings, dates, and so on, are also supported. A special symbol '=' denotes the equality predicate for each type. Analogously, the symbols '<=' and '>=' are given the usual meaning. A set of strongly typed function signatures is included in the language; and an implicit set of functions **Desc** (standing for level descriptors) is assumed, in order to admit expressions composed of references to level attributes. Variables in IRAH must be declared as it will be clear immediately.

A typical IRAH statement defining a set of single dimension exceptions has the form:

```
CREATE REVISION revision_name
ON DIMENSION d (l1 AS ν1, ..., lmax AS νmax)
SET lh1 = ch1
WHERE B11 AND ... AND Bk11
...
SET lhm = chm
WHERE B1m AND ... AND Bkmm
```

Here, *d* is a constant of type **D**, *revision_name* is the name of the redefinition, *l*_{*j*}, 1 ≤ *j* max, and *l*_{*h*}^{*j*}, 1 ≤ *i* ≤ *m*, are levels of *d* -constants of type **L**-; *ν*_{*i*} are variables of type **C**, and *c*_{*h*}^{*j*} are constants of type **C**. The SET ... WHERE ... expression constitutes the *redefinition rule j* in statement denoted *revision_name*, where *l*_{*h*}^{*j*} = *c*_{*h*}^{*j*} represents the head of rule *j*, and *B*₁^{*j*} AND ... AND *B*_{*k*_{*j*}}^{*j*} is the body of rule *j*. Elements *B*_{*i*}^{*j*} are called *level l_i - coordinate formulas*. Expressions *B*_{*i*}^{*j*} are made out of atomic constructs *C*^{*l_i*} (called *level l_i - coordinate expressions*), and propositional connectors ∧, ∨ and ¬, combined as in propositional logic. A *level l_i - coordinate expression C^{l_i}* is a condition intended to hold over the values of the attributes in level *l_i*. Formally, is an expression of the form *C*^{*l_i*} : φ(*ν*_{*i*}.*A*, *t*₁, ..., *t*_{*n*}), where φ is a predicate symbol (a boolean function), *ν*_{*i*} is the variable of type **C** associated with level *l_i* in the AS clause of the statement, *A* is a descriptor of level *l*, and the *t*_{*s*} are well-formed ground terms of type *T_i*, matching the signature of φ. A restriction is imposed to conditions *C*^{*l_i*} appearing in the body of a rule *j* : level *l_i* must immediately precede level *l_h*^{*j*} in the graph of the dimension hierarchy. Formally *l_i* <_{*d*} *l_h*^{*j*}.

The idea underlying an IRAH rule can be expressed as follows: if a path in the revised dimension instance satisfies the conditions present in the WHERE clause, it must satisfy the condition present in the SET clause. The process of revision consists in choosing a rule in a bottom-up fashion, according to the order of head levels in the dimension hierarchy, and isolating the paths in the dimension instance satisfying each one of the coordinate formulas in the WHERE clause. Then, a revision of those paths is performed, setting the value of the coordinate appearing in the head level of the rule to the value specified in the SET clause, and updating the values of the coordinates on the levels above in the paths.

In order to give the analyst the opportunity of specify which subset of coordinates of several dimensions will be affected by some exception rule, we introduce an extended kind of IRAH statement:

```
CREATE REVISION revision_name
ON DIMENSIONS d1 (l11 AS ν11, ..., lmax1 AS νmax1) AS D1,
...
dn (l1n AS ν1n, ..., lmaxn AS νmaxn) AS Dn
SET D11.lh11 = th11
...
Dp11.lhp11 = thp11
WHERE B11 AND ... AND Bk11
```

```

...
SET  $D_1^j.l_{h1}^j = t_{h1}^j$ 
...
 $D_{p_j}^j.l_{hp_j}^j = t_{h_j}^j$ 
WHERE  $B_1^l$  AND ... AND  $B_{k_j}^l$ 
...
SET  $D_1^q.l_{h1}^q = t_{h1}^q$ 
...
 $D_{p_q}^q.l_{hp_q}^q = t_{hp_q}^q$ 
WHERE  $B_1^m$  AND ... AND  $B_{k_m}^m$ 

```

Here the analyst can specify exceptions that affect more than one dimension. The dimensions involved are declared at the beginning of the statement and multiple heads for each exception may appear within the same SET clause. Each head of rule is specified by a expression like $D_{p_j}^j.l_{hp_j}^j = t_{h_j}^j$, where $t_{h_j}^j$ is a term that stands for a constant of type **C** or for a variable ν_h^j of type **C** that is associated with $D_{p_j}^j.l_{hp_j}^j$ in the declaration.

The use of this extended IRAH statement has the following restriction: if a level of a dimension appears within a SET clause, this dimension has to be mentioned at least in one of the restrictions in the WHERE clause.

Let's clarify the use of this kind of statement with an example:

Example 2 *The exception in Example 1 is expressed in IRAH as:*

```

CREATE REVISION  $R_1$ 
ON DIMENSIONS Customer (Delay AS  $\nu_1$ ) AS C
Geography (City AS  $w_1$ ) AS G
SET C.status = Consider,
( G.city =  $w_1$  )
WHERE  $\nu_1$ .month=5 AND  $w_1$ =New York

```

In the previous example, the SET clause that appear presented enclosed in parenthesis –an optional clause–, has the particularity that does not specify any actual change into the rollup function of its dimension. On the other hand, setting a level of a dimension with its corresponding variable, give us the chance to restrict the members of its dimension that will be affected by the exception. Here, the real change into the rollup function will take place into the Customer dimension, but will affect only the cells of the cube that belong to New York's offices.

The important issue derived from the syntax of this extended form of IRAH statement, is that it can be easily rewritten into a set of single dimension IRAH statements and, therefore, be used with a modified version of the algorithms presented in [MEVT02], version that will be presented in the next section. The statement is split into ordinary IRAH statements, each of them with the syntactical constructs corresponding to one dimension each, as it is shown below:

```

CREATE REVISION  $R_1^1$ 
ON DIMENSION Customer (Delay AS  $\nu_1$ )
SET status = Consider
WHERE  $\nu_1$ .month=5

CREATE REVISION  $R_1^2$ 
ON DIMENSION Geography (City AS  $w_1$ )
SET city =  $w_1$ 
WHERE  $w_1$ =New York

```

Once an extended IRAH statement as the above has been declared, it can be rewritten in ordinary IRAH statements, and these statements used to revise the contents of the dimensions involved. The contents of a materialized cube view can be revised in IRAH by means of the REVISE CUBE statement:

```

REVISE CUBE cube_name
WITH revision_name1, ..., revision_namek
INTO revised_cube [ [GAINED | LOST] VALUES ONLY] INTERSECTING CELLS

```

The view *revised_cube* is the name of a sparse cube view containing the modified cells, resulting from the revision process. The clause `[[GAINED | LOST] VALUES ONLY]` indicates that aggregate values that must be added or subtracted from the value in each modified cell may be computed separately. The new clause `INTERSECTING CELLS`, introduced in this work, indicates that the revision process will use multiple dimension exceptions, and that these ones were specified using the extended IRAH language presented here. This means that affected cells of the data cube to be computed, will be the result of the intersection of the affected cells of each of the multiple dimensions mentioned in the `SET` clause of the IRAH statement.

The following `REVISE CUBE` statement for Example 1 will produce a sparse cube containing the East column data affected and the gained and lost cubes presented in the figures below:

```

REVISE CUBE StatusByRegion
WITH R1
INTO Revised.StatusByRegion INTERSECTING CELLS

```

	West	East
Consider	50,000	205,000
Reject	3,000	335,000

	West	East
Consider	0	0
Reject	0	40,000

	West	East
Consider	0	40,000
Reject	0	0

Figure 5: (a) Revised cube view. (b) Cube with *Lost* information. (c) Cube with *Gained* information.

4 The Revision Algorithm

We have presented in [MEVT02] an algorithm that identifies the changing cells in the revised cube and for each one of them finds a set of cells –suitable for the computation of the new contents of the updated cells– in cubes preceding the cube under revision in the cube lattice [HRU96]. Our algorithm is slightly based on the linear algorithm which computes all the conclusions in a defeasible theory proposed by Maher *et al* [MRA⁺01]. Once the set has been determined, the revision can be applied.

We presented an algorithmic approach that takes as input a set of exception rules R , the dimension instances d_1, \dots, d_k , represented as tables T_{d_1}, \dots, T_{d_k} , and a cube C defined over some target space $\mathcal{S} : (d_1 : \text{targetLevel}_1, \dots, d_k : \text{targetLevel}_k)$; the algorithm produces a sparse exception cube C_{Lost} (alternatively C_{Gained}) over \mathcal{S} , containing all cells of cube C whose measures have been lost (gained) due to the application of the revisions in R , each cell with the corresponding computed measure.

In this work, we introduce a modified version of this algorithm, which can revise the data cube changed by exceptions that affect more than one dimension.

The revision process is divided in three parts. In the first part –outlined in Algorithm 1 below–, for each dimension d_i , with a set $R(d_i)$ of exception rules, the level targetLevel_i associated by \mathcal{S} to dimension d_i is isolated, in order to produce all the coordinates in targetLevel_i whose associated cells in cube C need to be modified. For each affected coordinates, sets of level members of the levels corresponding to a less aggregated materialized cube are identified, such that there exists a computation of the measures lost (gained) for the modified cells in cube C requiring aggregating the least number of cells.

Algorithm 1.

Input: A dimension table T_{d_i} , a set of exception rules $R(d_i)$, and a target level targetLevel_i , in dimension d_i .

Output: A function *DistCalcCoord* mapping modified coordinates in level targetLevel_i into sets *Lost* (*Gained*) of tuples t of the form $(t.\text{level}, t.\text{source})$.

Process.

For each dimension d_i

Put all **WHERE** clauses of exceptions in $R(d_i)$ in conjunctive normal form, and order conditions in the **WHERE** clause

according to the ordering of their levels in the dimension schema. For each rule r in R , $\text{Body}(r)$ denotes the highest level in the dimension hierarchy included in a condition defined in the **WHERE** clause of rule r ; $\text{Head}(r)$ denotes the level corresponding to the head of rule r .

Initialize a set *Paths* –which will contain tuples from table $T(d_i)$ – to the empty set.

Proceed bottom-up in the hierarchy [MEV01] on a level by level basis, with level j varying from l_{bottom} to targetLevel_i , as follows:

1. Modify a path p in set *Paths*, at level k , k varying from j to targetLevel_i , applying the coordinate values at the corresponding levels in a subpath p' which satisfies the condition in the head of some exception rule r with $\text{Head}(r)=j$, provided that: (a) all conditions in the body of rule r hold in p –in the course of previous iterations–, and r is the only rule in that situation; or (b) all rules satisfying condition (a) have all the same head.
2. Modify a path p in set *Paths*, applying null values to levels k in p , varying from j to targetLevel_i , provided that there exists at least two different rules r_1 and r_2 such that $\text{Head}(r_1)=\text{Head}(r_2)=j$, and all conditions in the body of both rules r_1 and r_2 hold in p –checked in previous iterations–.
3. We define the set of *active exceptions* of path p , denoted $A_p(\text{coord})$, where *coord* is the coordinate at the bottom level of path p , as the set containing all rules r that modified p during the revision. For each rule r that modifies some path p according to the previous steps, remove all rules s from A_p , such that the following holds: $\text{Head}(s)$ precedes $\text{Head}(r)$ and $\text{Body}(r)$ precedes $\text{Head}(s)$.
4. Add each rule r that modifies some path p according to previous steps, to the set A_p , such that there is no rule s in A_p such that there exists no rule s in A_p where the following condition holds: $\text{Head}(r)$ precedes $\text{Head}(s)$ or $\text{Body}(s)=\text{Head}(r)$.
5. Associate to each modified path p a tuple (*old*, *new*, *level_min*, *source*) such that: *old* contains the original coordinate value at level targetLevel_i of path p ; *new* contains the modified coordinate value at level targetLevel_i of path p ; *level_min* contains the level defined by $\min(\{\text{Body}(s) | s \in A_p\})$; *source* contains the coordinate value of path p at level *level_min*.
6. Add to set *Paths* all paths p in T_{d_i} , not already present in set *Paths*, that satisfy the first condition of some rule r on level j of path p . Associate rule r to path p . Initialize A_p to the empty set.
7. Remove from *Paths* all paths p not satisfying some condition on level j of any rule r associated to p .
8. With tuples (*old*, *new*, *level_min*, *source*) of each modified path p , build function *DistCalcCoord* mapping modified coordinates in level targetLevel_i into sets *Lost* and *Gained* of tuples t of the form $(t.\text{level}, t.\text{source})$. A tuple t in set *Lost* (*Gained*) of *DistCalcCoord(c)*, for a modified coordinate c , implies that any cell s in the cube, with $\text{targetLevel}_i : c$ as the coordinate for dimension d_i , will lose (gain) the measures of all cells with $t.\text{level} : t.\text{source}$ as a coordinate for dimension d_i , and the same coordinates of cell s for the rest of the dimensions. In the special case when *old* and *new* are the same coordinate value, a new entry into the *DistCalcCoord* is generated for that value, with empty sets of *Lost* and *Gained*.
9. Remove the tuples that appear in both *Lost* and *Gained* sets, associated to some coordinate c by function *DistCalcCoord*.

The change introduced into the Algorithm 1 is in the step number 8, when an entry into the *DistCalcCoord* function is generated for any special tuple where *old* and *new* are the same coordinate value. This indicates that it is not a common exception that modifies the path; otherwise, this exception comes from an **IRAH SET** clause where a level of a dimension is set with its correspondent variable.

The complexity of Algorithm 1 is $N \log N$, where N is the number of tuples in the dimension tables satisfying the first condition in the **WHERE** clauses of the exception rules.

After applying Algorithm 1 to all dimensions, the cells affected are identified by Algorithm 2.

Algorithm 2.

Input: A target space $\mathcal{S}(d_1 : \text{targetLevel}_1, \dots, d_k : \text{targetLevel}_k)$, and the function *DistCalcCoord* computed by Algorithm 1, for each dimension $d_i, 1 \leq i \leq k$.

Output: A set *AffectedCells* of cells in cube C for space \mathcal{S} such that their measures need to be modified due to the revision.

Process.

For each dimension $d_i, 1 \leq i \leq k$

For each coordinate c in the domain of *DistCalcCoord* for dimension d_i

$\text{AffectedCells}(d_i) = \bigcup \text{slice cube } C \text{ to level member } d_i : \text{targetLevel}_i : c, \text{ slice cube defined by Abello et al [ASS01]}$

$\text{AffectedCells} = \bigcap \{ \text{AffectedCells}(d_i) \mid \text{AffectedCells}(d_i) \neq \emptyset \}$

Algorithm 3 computes the measure amount lost or gained by each cell computed by Algorithm 2, proceeding in a top-down fashion, possibly in parallel.

Algorithm 3.

Input: An affected cell $z : (d_1 : l_1 : c_1, \dots, d_k : l_k : c_k)$ in a cube C on space $\mathcal{S} = (d_1 : l_1, \dots, d_k : l_k)$ –a member of the set *AffectedCells* computed by Algorithm 2–.

Output: A cell z' in an exception cube *CLost* (respectively *CGained*) on space \mathcal{S} , holding the measure lost (respectively gained) by cell z after the revision.

Process.

Let $\text{AggregateLost}(z) = \text{AggregateGained}(z) = \phi$.

Let sets $ALost_i$ and $AGained_i$ be $\text{DistCalCoord}(c_i).lost$ and $\text{DistCalCoord}(c_i).gained$ respectively, for dimension d_i of space \mathcal{S} , $1 \leq i \leq k$.

If some set $ALost_i$ ($AGained_i$) $\neq \phi$, let set $ALost_i$ ($AGained_i$) $= \{(l_i, c_i)\}$.

$\text{CellsLost} = ALost_1 \times \dots \times ALost_k$, and

$\text{CellsGained} = AGained_1 \times \dots \times AGained_k$.

With each tuple $t : ((l_1, coord_1), \dots, (l_k, coord_k))$ in CellsLost (CellsGained), generate cell $(d_1 : l_1 : coord_1, \dots, d_k : l_k : coord_k)$ and store it into set $\text{AggregateLost}(z)$ ($\text{AggregateGained}(z)$).

For each cell $(d_1 : l_1 : coord_1, d_k : l_k : coord_k)$ in set $\text{AggregateLost}(z)$ ($\text{AggregateGained}(z)$), such that no materialized cube exists on space $\mathcal{S}' = (d_1 : l_1, \dots, d_k : l_k)$, drill down to a space preceding \mathcal{S}' in the space lattice [HRU96], such that a materialized cube exists, and replace $(d_1 : l_1 : coord_1, d_k : l_k : coord_k)$ by the set obtained after the drill down operation. Generate a cell z' for each z in the exception cube *CLost* (*CGained*); aggregate measures of cells in $\text{AggregateLost}(z)$ ($\text{AggregateGained}(z)$), according to the aggregation function used to compute cube C , and put the result as the measure of cell z' .

Example 3 Let us consider again the instance of a dimension Customer in Example 1, and assume the following revision set holds over this instance:

$r_1 : (\nu_1 \text{ AS delay}) \text{ SET status} = \text{Consider} \text{ WHERE } \nu_1.month=5$
 $r_2 : (w_1 \text{ AS city}) \text{ SET city} = w_1 \text{ WHERE } w_1 = \text{New York}$

Figure 6 shows the contents of cubes *StatusByRegionLost* and *StatusByRegionGained*, generated by the modified revision algorithm.

	West	East
Consider	0	0
Reject	0	40,000

	West	East
Consider	0	40,000
Reject	0	0

Figure 6: (a) Cube *StatusByRegionLost*.

(b) Cube *StatusByRegionGained*.

Without this revision technique, analysts would not have the chance of using this kind of multiple dimension exceptions into their databases. In Figure 7(a), the reader can see what would have been the revised data cube for *StatusByRegion* view if the analyst had not had the opportunity to specify the constrain about New York offices. In that case, as well as the value 40,000, corresponding to a loan application done by customers with a five-month delay from a New York City’s office, the value of 85,000 from a requested loan done in Washington, has been aggregated with a *Consider* status. In Figure 7(b), we have the revised cube generated by the technique explained in this work.

5 Conclusions and Future Work

We have presented an extended IRAH statement to handle multiple dimension exceptions and a rewrite technique to translate it into a set of single dimension IRAH statements [MEVT02]. This new kind of IRAH

	West	East
Consider	50,000	290,000
Reject	3,000	250,000

	West	East
Consider	50,000	205,000
Reject	3,000	335,000

Figure 7: (a) Using single dimension exception. (b) Using multiple dimension exception.

statement gives the analyst the opportunity of specifying exceptions that change the hierarchy of one or more dimensions but their effects are only consider for some members of the other dimensions. We have shown a modified version of the revision algorithm (with complexity $N \log N$ on the number of tuples in the relational representation of the dimensions) introduced in [MEVT02], that uses these new rewritten IRAH statements as input to find the cells in the cube under revision whose measures need to be modified, and efficiently computes the new measures using materialized cube views at less aggregated levels.

In future research we will explore different alternatives of implementation for this technique. As this work covers the logical layer of the database, it will be very important to analyze how the use of different physical structures to store the data –MOLAP, ROLAP– will impact in the performance of the algorithm.

References

- [ABC00] M. Arenas, L. Bertossi, and J. Chomicki. Specifying and querying database repairs using logic programs with exceptions. In *Proceedings of the International Conference on Flexible Query Answering*, 2000.
- [ABGM99] G. Antoniou, D. Billington, G. Governatori, and M. Maher. Revising non-monotonic belief sets: The case of defeasible logic. In *Proceedings of the 23th German Conference in Artificial Intelligence, LNAI 1701*, Springer, 1999.
- [ASS01] A. Abello, J. Samos, and F. Saltor. Understanding analysis dimensions in a multidimensional object-oriented model. In *Proceedings of DMDW’01*, 2001.
- [BFH98] C. Boutilier, N. Friedman, and J. Halpern. Belief revision with unreliable observations. In *Proceedings of the AAAI*, 1998.
- [Bre00] G. Brewka. Declarative representation of revision strategies. In *Artificial Intelligence*, 2000.
- [CGD85] Alchouron C., P. Gardenfors, and Makinson D. On the logic of theory change: Partial meet functions for contraction and revision. In *Journal of Symbolic Logic* 50, 1985.
- [ER83] D. Etherington and R. Reiter. On inheritance hierarchies with exceptions. In *Proceedings of the AAAI*, 1983.
- [HMF99a] C. Hurtado, A.O. Mendelzon, and A. Vaisman. Maintaining data cubes under dimension updates. In *Proceedings of IEEE/ICDE’99*, 1999.
- [HMF99b] C. Hurtado, A.O. Mendelzon, and A. Vaisman. Updating olap dimensions. In *Proceedings of ACM DOLAP’99*, 1999.
- [HRU96] V. Harinarayan, A. Rajaraman, and J. Ullman. Implementing data cubes efficiently. In *Proceedings of the ACM-SIGMOD Conference*, 1996.
- [LAW98] W. Lehner, J. Albrecht, and H. Wedkind. Normal forms for multidimensional databases. In *Technical Report*, University of Erlangen-Nuremberg, Germany, 1998.
- [MEV01] M. Minuto Espil and A. Vaisman. Efficient intensional redefinition of aggregation hierarchies. In *Proceedings ACM-DOLAP’01*, 2001.
- [MEV03] M. Minuto Espil and A. Vaisman. Revising aggregation hierarchies in olap: a rule-based approach. In *Data and Knowledge Engineering, Special issue on “Advances in Data Warehousing”, Elsevier Journals, to appear*, North Holland, Amsterdam, 2003.
- [MEVT02] M. Minuto Espil, A. Vaisman, and L. Terribile. Revising data cubes with exceptions: A rule-based perspective. In *4th International Workshop on Design and Management of Data Warehouses (DMDW 2002)*, Toronto, Canada, 2002.
- [MRA⁺01] M. Maher, A. Rock, G. Antoniou, D. Billington, and T. Miller. Efficient defeasible reasoning systems. In *International Journal on Artificial Intelligence Tools, Vol.10(4)*, 2001.
- [PJ99] T.B. Pedersen and C. Jensen. Multidimensional data modeling for complex data. In *Proceedings of IEEE/ICDE’99*, 1999.
- [Vai01] A. Vaisman. Updates, view maintenance and time management in multidimensional databases. *Phd Thesis*, <http://www.cs.toronto.edu/avaisman/publications>, 2001.
- [VRC02] A.O. Vaisman, A. and Mendelzon, W. Ruaro, and S. Cymerman. Supporting dimension updates in an olap server. In *To appear Proceedings of CAISE’02*, Toronto, Canada, 2002.
- [WA98] M. Williams and G. Antoniou. A strategy for revising default theory extensions. In *Proceedings of the 6th International Conference On Principles of Knowledge Representation and Reasoning (KR)*, Toronto, Canada, 1998.