Revising Data Cubes with Exceptions: A Rule-Based Perspective

Mauricio M. Espil
mminuto@uca.edu.ar
Universidad Catolica Argentina

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

Leonardo Terribile
IBM Argentina
terribil@ar.ibm.com

Abstract

Information in a data warehouse does not always reflect unquestionable facts in an organization. Sometimes, data should be considered as representing just beliefs about the state of the world it intends to model, and could be subject to revision. We claim that a mechanism of belief revision capable of altering the contents of dimension instances is needed, in order to guarantee accurate analysis results. This revision will also affect the data cubes defined over such dimensions. In previous work we addressed the problem of computing revised *dimension instances*, introducing a language denoted IRAH in order to define exceptions intensionally. In this paper we present the IRAH language from the perspective of a language intended to maintain materialized *cube views* under revision by exceptions to the functional dependencies implied in the dimensions hierarchies. We define a clear semantics for the data cube revision problem, based on a non-monotonic reasoning framework, introducing the concept of *summarizability under uncertainty.* Finally, we introduce an efficient algorithm that applies the revision to a data cube. This algorithm identifies the cells in the cube which must be modified and updates them (*i.e.* revises the cube), obtaining the aggregated data from the closest materialized cells.

1 Introduction

A multidimensional database is usually represented as sets of dimensions and facts. Facts are seen as points in a multidimensional space with coordinates in each dimension, with an associated set of *measures.* Dimensions provide appropriate contextual meaning to facts, and are usually organized as hierarchies, 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.

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 based on the analyst's information or intuition. As an example, 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. For instance, 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 Monday) at some store, causing the closing of the store that day. In this setting, we claim that assertions like "Mondays are business days" should be considered merely 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. We called this the *data cube revision problem.*

Let us suppose a credit company maintains a multidimensional database holding information about credit applications, including the loan identification, the borrower identification, the date on which the application was issued, and the requested amount. The table in Figure 1 shows an instance of the

appId	custId	date	amount
a_1	c_1	02/11/2002	15,000
a_2	c_2	01/10/2002	3,000
a_3	c_3	02/07/2002	250,000
a_4	c_6	01/28/2002	40,000
a_5	c_{12}	01/30/2002	35,000
a_6	c_{14}	02/04/2002	165,000

Figure 1: Credit Applications

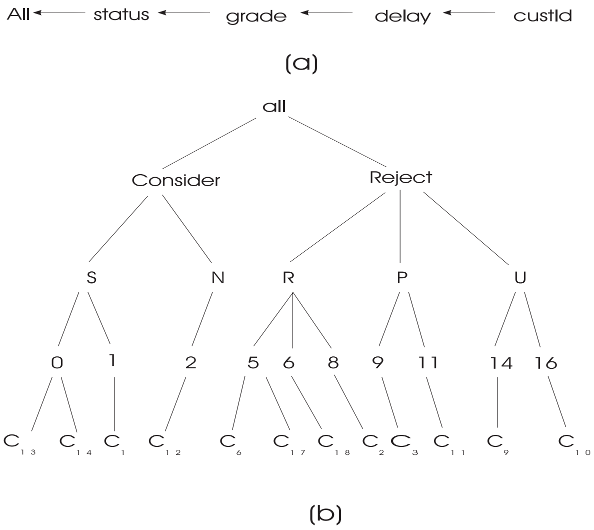

Figure 2: Dimension *Customer*

database. Here, *custId* and *date* represent the bottom levels of dimensions Customer and Time, respectively. *Amount* is the measure. Figure 2 (a) depicts the schema of dimension Customer, with its different levels of aggregation, and Figure 2 (b) 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.

Example 1 *Consider the materialized cube view (let us denote it* StatusByMonth*), aggregated by* status *and* month*, shown in Figure 3. Let us also suppose that the analyst is not totally convinced by the rule indicating that credit applications from customers with five months elapsed from their last payment should*

	Jan-2002	Feb-2002
Consider	35,000	166,500
Reject	43,000	250,000

Figure 3: Cube view aggregated by *status* and *month*

	Jan-2002	Feb-2002
Consider	75,000	166,500
Reject	3,000	250,000

Figure 4: Revised cube view aggregated by *status* and *month*

be considered for rejection, although she must accept the rule grading those customers as risky, because the rule is mandatory. An exception is therefore needed, asserting that applications from customers with a delay of 5 months should acquire the Consider *status. This exception affects the contents of the dimension hierarchy and the associated cube. The new cube contents are shown in Figure 4. The revised dimension is depicted in Figure 5.*

In previous work [MV01] we addressed the problem of computing revised dimension instances, treating the revised paths as exception paths. In that work we introduced a language, called IRAH (standing for Intensional Redefinition of Aggregation Hierarchies), suitable for defining exceptions in an intensional fashion. However, we did not deal with the problem of applying the revision process efficiently to a materialized cube view (*e.g.* by means of identifying the cells to be modified in a given materialized cube view, and defining from which materialized cube view the cube can be updated).

In this paper we present the IRAH language from the perspective of a language intended to maintain cube views under revision by exceptions to the functional dependencies implied in the dimensions hierarchies. We define a clear semantics for the data cube revision problem, based on a non-monotonic reasoning framework. This semantics allows introducing the concept of summarizability under uncertainty. Finally, we present an algorithm (with complexity NlogN on the number of tuples in the relational dimension instance) that applies the revision defined by IRAH statements to a given cube. The algorithm 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.

The remainder of the paper is organized as follows:

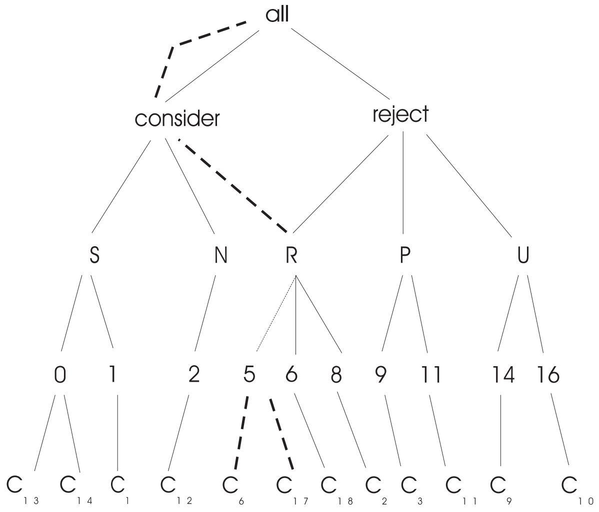

Figure 5: Revised dimension *Customer*.

in Section 2 we comment on related work. Section 3 presents our model, representing dimension and data cubes as sets of rules. Section 4 revisits the IRAH language, and defines the cube revision semantics. An algorithm for revising dimension instances and cubes is presented in Section 5. We conclude in Section 6.

2 Related Work

Rollup functions in dimension hierarchies have been extensively addressed in the data warehousing literature [CT98, HMV99]. These approaches, however, do not intend to give any insight on the conceptual semantics of the rollup functions. Moreover, although Hurtado *et al* [HMV99] address the issue of dimension updates, they do not consider the problem of revising dimension hierarchies. 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]. These works do not address the semantics of ISA relationships, specially suited for modeling different levels at which a collection of facts can be viewed. As we pointed out above, data in a data warehouse may sometimes represent mere beliefs, not unquestionable assertions. Consequently, uncertainty may appear, and 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 [AGM85]. 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 [MV01], 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 (different from the general semantics proposed by Brewka), 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. This approach takes into consideration previous works on belief revision [WA98, ADGM99], and theory repairing [ABC00]. However, we did not deal there with the cube view maintenance problem introduced by the revision process, which is the subject of the following sections.

3 The Model

In this section we present the multidimensional model from a formal, axiomatic, proof-theoretic perspective. Let us consider the following finite sets: a set $\mathbf{D}$ of dimension names, a set $\mathbf{L}$ of level names, and a set $\mathbf{C}$ of constants.

Given a dimension $d \in \mathbf{D}$, a *dimension schema* for d is a pair $(LevelSet(d), \prec_d)$, where $LevelSet(d)$ is a set included in $\mathbf{L}$, and $\prec_d$ is a relation in $LevelSet(d)^2$ such that $\prec^*$, its transitive closure, is a strict partial order, with a unique bottom, denoted generically l_{bottom}, and a unique top, denoted *All,* such that $l_{bottom} \neq All$. An element l in $LevelSet(d)$ is called a *level* of dimension d, and will be denoted as $d : l$.

Let *Level_Member* be a relation in $\mathbf{D} \times \mathbf{L} \times \mathbf{C}$. *Level_Member* associates constants (from now on coordinates) with levels in a dimension; we call a tuple (d, l, c) in *Level_Member* a *dimension level member,* and, for simplicity, we denote it as $d : l : c$, meaning that c is a coordinate in level l of dimension d. The set $Iset(d : l) = \{c | (d : l : c) \in Level\_Member\}$ is called the *Instance Set* of level l in dimension d. The following constraint holds: there exists a constant $all \in \mathbf{C}$ such that $Iset(d : All) = all, \forall d \in \mathbf{D}$.

Example 2 *Let us consider again dimensions* Customer *and* Time *in Section 1. A schema for dimension* Customer *is the pair (LevelSet(Customer),* $\prec_{Customer}$*), where LevelSet(Customer)=* {custId, delay, grade, status, All}, *and* $\prec_{Customer}$ *is defined according to the precedence relation depicted in Figure 2. A possible schema for the* Time *dimension is the pair (LevelSet(Time),* $\prec_{Time}$*), where LevelSet(Time)* = {day, month, year, All}, *and* $\prec_{Time}$ *is defined by the usual precedence relation day* $\prec_{Time}$ *month, month* $\prec_{Time}$ *year, year* $\prec_{Time}$ *All.*

Definition 1 (Space) *Let* $d_1, , d_k$ *be dimensions in* $\mathbf{D}$*, with their corresponding schemas, and let* $l_1, , l_k$ *be levels in* $\mathbf{L}$*, such that for all* $1 \leq i \leq k, l_i \in LevelSet(d_i)$*. We call the tuple* $((d_1, l_1), , (d_k, l_k))$ *a (multidimensional)* space*. In what follows we will denote a* space *as* $(d_1 : l_1, , d_k : l_k)$*.*

Definition 2 (Cell) *Let* $(d_1 : l_1, , d_k : l_k)$ *be a multidimensional space, and let* $c_1, , c_k$ *be coordinates such that for all* $1 \leq i \leq k, c_i \in ISet(d_i : l_i)$*. The tuple* $(d_1 : l_1 : c_1, , d_k : l_k : c_k)$ *is called a* cell *in the multidimensional space.*

3.1 Fact Aggregations and Cubes

Let us consider again the example presented in Section 1. As we pointed out before, facts (in this example, credit applications) have no meaning *per se.* In our approach we consider facts as abstract entities, no matter their nature and the measures which apply to them. We can find an analogy between abstract facts and object IDs. We consider abstract facts and measures as logical concepts (types). Thus, we define type **A** for abstract facts, and different abstract types **M** for measures. In our running example, an abstract fact is a credit application; a measure type is *Amount.*

Definition 3 (Fact Valuation) *We associate facts with measures, by means of a family of predicates* $Valuate_{\mathbf{M}}$*, with signature* $\mathbf{A} \times \mathbf{M} \rightarrow$ *Boolean. We denote a* fact valuation *a ground atom formed upon the predicate* $Valuate_{\mathbf{M}}$*, for some measure.*

Fact valuations give no hint on the nature of facts. In the multidimensional model, facts are given their real meaning when they are associated with the coordinate of a cell in a multidimensional space. As it has been previously defined, a cell in a multidimensional space is a tuple composed by level members, one for each dimension. Thus, we must attach abstract facts to level members.

Definition 4 (Fact Aggregation) *Let us consider a predicate* aggr *with signature* $\mathbf{A} \times \mathbf{D} \times \mathbf{L} \times \mathbf{C} \rightarrow$ *Boolean, associating an abstract fact to a level member. Ground atoms for predicate aggr are denoted* fact aggregations*. The informal meaning of an abstract aggregation of the form* aggr(a, d, l, c) *is the following: the abstract fact* a *contributes to the contents of some cell with coordinate* c *belonging to level* l *of dimension* d*.*

Example 3 *In Example 1,* a_1 *would be an abstract fact, and* $Valuate_{Amount}(a_1, 15.000)$ *would be a fact valuation. Moreover,* aggr$(a_1, Customer, custId, c_1)$ *and* aggr$(a_1, Time, day,$ *'2-11-2002') in our running example are abstract aggregations.*

Definition 5 (Cell Contents) *Let* $(d_1 : l_1, ..., d_k : l_k)$ *be a multidimensional space, and a function* contents *associating cells of the form* $(d_1 : l_1 : c_1, ..., d_k : l_k : c_k)$ *with the set of abstract facts* $\{a \in A | \forall i, i \in (1..k), aggr(a, d_i, l_i, c_i) is\ proved\}$*. The expression* contents$(d_1 : l_1 : c_k, ..., d_k : l_k : c_k)$ *defines the* contents *of cell* $(d_1 : l_1 : c_k, ..., d_k : l_k : c_k)$*.*

Notice that an abstract fact will be considered a member of the contents of the cell $(d_1 : l_1 : c_k, ..., d_k : l_k : c_k)$ if it is proved that for all coordinates of this cell, the corresponding fact aggregation $aggr(a, d_i, l_i, c_i)$ holds. The proof mechanism was presented in our previous work [MV01].

Definition 6 (Cube) *Let* $(d_1 : l_1, ..., d_k : l_k)$ *be a multidimensional space* $\mathcal{S}$*. Let* Cube *be a function that associates* $\mathcal{S}$ *with the function resulting from restricting the domain of function* contents *to cells in* $\mathcal{S}$ *only. We call expression* Cube$(d_1 : l_1, ..., d_k : l_k)$ *a* cube *over space* $\mathcal{S}$*.*

Thus, a cube turns out to be a function that associate cells in a given space to sets of abstract facts.

3.2 Axiom Bases

Multidimensional databases are built from base collections of concrete facts (*i.e.* facts with attributes and measures), stored in relational implementations in so-called *base fact tables.* From our logic perspective, base fact tables are sets of sentences denoted axiom bases, which provide a basis for logic inference. The sentences constituting axiom bases in our logic are fact valuations, and fact aggregations for bottom levels in all dimensions. It is necessary, however, to remark some important aspects:

(a) Our view of the model from a proof theoretic perspective is completely independent from the implementation. For example, the first row in the base fact table of Figure 1 is a synthesis of the assertions $Valuate_{Amount}(a_1, 15.000)$, $aggr(a_1, Customer, custId, c_1)$, and $aggr(a_1, Time, day,$ '2-11-2002'). (b) Incomplete information of a given fact is supported by the model. For instance, an application for some customer may exist, without storing the day this application has been issued. Missing measures are also supported. These features provide a clear conceptual meaning for null values in a classical base fact table, giving more freedom to model uncertainty on facts than traditional models for OLAP. (c) It is not mandatory to have unique fact identifiers in base fact tables. (d) Uniqueness guarantee ensures that an abstract fact can only aggregate to *at most* one cell in the cube. This introduces a new concept: summarizability under uncertainty, not addressed in other approaches [LS97, HM01]

3.3 Dimension Instances

We represent dimension instances as sets of normal default rule schemas [Rei80, Ant99], in order to provide an adequate framework for dealing with revisions.

Definition 7 (Aggregation Rule) *Given a pair of coordinates c_1 and c_2 in levels l_1 and l_2 of dimension instance d, such that $l_1 \prec_d l_2$, the following rule schema is called $l_2 - aggregation\ rule$.*

$$\frac{aggr(X, d, l_1, c_1)}{aggr(X, d, l_2, c_2) \wedge \mu(c_2)}$$

where
$\mu(c_2) = \forall \nu \in Iset(d : l), \nu \neq c_2 \wedge aggr(X, d, l, \nu)$.

The intuitive meaning of an aggregation rule is the following: if an abstract fact X aggregates over coordinate c_1 in level l_1 of dimension d, we infer that X should aggregate on coordinate c_2 in level l_2 of dimension d, unless some preferred evidence on the contrary is accepted.

The expression $\mu(c_2)$ is a finite conjunction of negative literals of the form $\neg aggr(d, X, l, \nu)$, for each coordinate value $\nu \neq c_2$ in set $Iset(d : l)$, denoted *uniqueness guarantee* for coordinate c_2 in level l_2 of dimension d, meaning that a fact in a dimension instance aggregates on no more than one coordinate for each dimension level.

Example 4 *The arc connecting coordinate U in level* grade *with coordinate* Reject *in level* status *in Figure 2, is represented by the rule:*

$$\frac{aggr(X, Customer, grade, U)}{aggr(X, Customer, status, Reject) \wedge \mu(Reject)}$$

where
$\mu(Reject) = \neg aggr(X, Customer, status, Consider)$

Definition 8 (Dimension instance) *A set of aggregation rules for dimension d is called a* dimension instance *of dimension d.*

4 The IRAH Language

Dimension instances are usually presented in the form of rollup functions, (*i.e.* interpreted as mappings between level instances). In order to consider exceptions to these rollup functions we introduced a rule language [MV01] such that a revision operation can be defined as a set of rules with a priority defined between them. We called this rule language IRAH (Intensional Redefinition of Aggregation Hierarchies).

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 '$\leq$', and '$\geq$' 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 an exception has the form:

```
CREATE REVISION revision_name
   ON DIMENSION d (l_1 AS ν_1,...,l_max AS ν_max)
   SET l_h^1 = c_h^1
   WHERE B_1^1 AND ...  AND B_{k_1}^1
        ...
   SET l_h^m = c_h^m
   WHERE B_1^m AND ...  AND B_{k_m}^m
```

Here, d is a constant of type **D**, *revision_name* is the name of the redefinition, $l_j, 1 \leq j$ max, and $l^i_h, 1 \leq i \leq m$, are levels of d (constants of type **L**); ν_i are variables of type **C**, and c^i_h 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 denoted *level* l_i *- coordinate formulas.* Expressions B_i^j are made out of atomic constructs C^{l_i} (denoted *level* l_i *- coordinate expressions*), and propositional connectors $\wedge, \vee$ and $\neg$, 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 $C^{l_i} : \varphi(\nu_i.A, t_1, ..., t_n)$, 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_is 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 precede level l_h^j in the dimension hierarchy. Formally $l_i \prec_d l_h^j$.

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. Once a redefinition has been declared, it can be used to revise the contents of the dimension. The following statement declares such revision:

```
REVISE DIMENSION d
   WITH revision_name1, ..., revision_namek
   INTO revised_instance
```

The instance *revised_instance* contains the modified paths. The contents of a materialized cube view can also be revised. This is declared with the statement

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

Again, 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 substracted from the value in each modified cell may be computed separately.

Example 5 *The exception in Example 1 is expressed in IRAH as:*

```
CREATE REVISION R1
   ON DIMENSION Customer (Delay AS ν1)
   SET status = Consider
   WHERE ν1.month=5
```

The following statement will produce an instance containing the paths marked with dashed lines in Figure 5.

```
REVISE DIMENSION Customer
   WITH R1
   INTO Customer_Exceptions
```

4.1 Redefinition Rules as Defaults

We have already shown how to map redefinition rules to default schemas [MV01]. The idea consists in (a) building the prerequisite of the default schema as the conjunction of all the formulas in the body of the rule (the WHERE clause) and ground atoms $aggr(X, d, l_i, \nu_i)$, one atom for each pair level-coordinate variable l_i, ν_i associated by the AS clause, such that variable ν_i appears in the body of the rule; (b) building the conclusion of the default schema (called justification-consequent in normal default logic terminology) as the conjunction of: (i) the ground atom $aggr(X, d, l_h^j, c_h^j)$, where l_h^j and c_h^j are the level and the coordinate appearing in the SET clause; (ii) an uniqueness guarantee for coordinate c_h^j.

Definition 9 *The default schema produced from mapping a redefinition rule with level l_h in the consequent will be called a l_h exception rule.*

Example 6 *The redefinition rule in Example 5 can be mapped to the normal default schema:*

$$\frac{aggr(X, Customer, delay, \nu_1) \wedge \nu_1.month = 5}{aggr(X, Customer, status, Consider) \wedge \mu(Consider)}$$

where
$\mu(Consider) = \neg aggr(X, Customer, status, Reject)$

Each rule with level l_h in the head appearing in a CREATE REVISION statement can be mapped therefore to an l_h-exception rule in our logic. We can build a revision set with all of those exception rules and name it N(d) after the name N appearing immediately following the `REVISION` keyword and the dimension name d appearing in the `ON` clause. There will be then a set R(Customer) of exception rules associated with the `CREATE REVISION` statement in Example 5.

4.2 Revision Semantics

Once a revision has been defined, we need to compute the contents of cells in a given cube, *i.e.,* the set of abstract facts present in those cube cells. Following Brewka [Bre00], we build a logic theory upon the axiom base, the dimension instances (sets of aggregation rules), and the revision sets (sets of exception rules), and let the mechanism of inference work with them, producing the derived fact aggregations. Due to space limitations we do not describe the inference mechanism introduced in previous work [MV01]. We will only define the cube revision semantics, as that work did not cover the cube revision problem.

Definition 10 (Theory) *Let d be a dimension; $\Gamma(d)$ the set of aggregation facts* $\mathrm{aggr}(X, d, l_{bottom}, c)$ *present in the axiom base for d; $\Delta(d)$ a set of (aggregation or exception) rules for d; $\ll$ a preference relation defined among rules in $\Delta(d)$; we call the triple* $\mathrm{T}(d)$=($\Gamma(d), \Delta(d), \ll$) *a* Theory *for dimension d.*

Definition 11 (Epistemic State) *Let $d_1, \ldots, d_k$ be dimensions, and let $\mathrm{T}(d_1), \ldots, \mathrm{T}(d_k)$, be theories for these dimensions. We call a tuple $\sigma(d_1, \ldots, d_k) =$ ($\mathrm{T}(d_1), \ldots, \mathrm{T}(d_k)$) an epistemic state for $d_1, \ldots, d_k$.*

Definition 12 (State Revision) *Let $\mathcal{D} = \{d_1, \ldots, d_k\}$ be a set of dimensions, and let* R $= \bigcup_{i:1..k} \mathrm{R}(d_i)$ *be the union of revision sets (sets of exceptions), one for each dimension. Moreover, let $\Gamma(d_1), \ldots, \Gamma(d_k)$ be axiom bases, $\Delta(d_1), \ldots, \Delta(d_k)$ sets of aggregation rules (an original dimension instance), and ϕ the empty preference relation. We call a function ρ_R that maps epistemic states of the form $\sigma(d_1, \ldots, d_n) = ((\Gamma(d_1), \Delta(d_1), \phi), \ldots, (\Gamma(d_k), \Delta(d_k), \phi))$, into epistemic states of the form $\sigma'(d_1, \ldots, d_n) = ((\Gamma(d_1), \Delta(d_1) \cup R(d_1), \ll (d_1)), \ldots, (\Gamma(d_k), \Delta(d_k) \cup R(d_k), \ll (d_k)))$, a* state revision under R *for dimensions $d_1, \ldots, d_k$ if and only if $\ll (d_1), \ldots, \ll (d_k)$ are preference relations that always prefer a rule r in $R(d_i)$ over a rule s in $\Delta(d_i)$ whenever rules r and s become applicable.*

Definition 13 (Cube Revision) *Let $d_1, ..., d_k$ be dimensions; $S(d_1, \ldots, d_k)$ the set of all spaces built upon $d_1, \ldots, d_k$; and the set* R *introduced in Definition 12. A function $revision_R$ mapping cubes on space $S(d_1, \ldots, d_k)$ induced by some epistemic state $\sigma(d_1, \ldots, d_k)$ into cubes of the same class, is called a* cube revision under R, *if and only if $revision_R$ ensures that the following holds: if $Cube_\sigma$ is the cube on space $s \in S(d1, ..., dk)$, induced by epistemic state σ, then $revision_R(Cube_\sigma) = Cube_{\sigma'}$, where $Cube_\sigma$ is the cube induced by epistemic state $\sigma' = \rho_R(\sigma)$.*

5 An Algorithmic Approach to Cube Revision

Definition 13 gives a clear formal semantics for cube revisions, from a proof-theoretic perspective. Nevertheless, from a practical point of view, this definition implies that the entire contents of the cells in the revised cube must be computed from scratch, a non-desirable approach. Moreover, in practice, cells do not contain abstract facts, but values resulting from measure summarization. Thus, a more efficient approach would only compute the contents of those cells in the revised cube that are likely to change due to the revision, not the contents of the whole cube. Further, the new contents of the cells in a revised cube can be computed from the contents of cells in other materialized cubes. For instance, in Example 1, suppose we need to compute the new contents of cells in the cube of Figure 3 defined over the space ($Customer : status$, $Time : month$), once the revision has been applied (Figure 4). A cube on a space ($Customer : grade$, $Time : month$) will not be suitable, because abstract facts a_2 and a_4 contribute to the value in cell ($Customer : grade : R$, $Time : month : January$), and, according to the hierarchy of Figure 2, the summarized contents of this cell will contribute to cell ($Customer : status : Reject$, $Time : month : January$). This would lead to an incorrect result, because after the revision abstract fact a_4 must contribute to the contents of cell ($Customer : status : Consider$, $Time : month : January$). On the other hand, a cube on space ($Customer : delay$, $Time : month$) can be used for computing the new contents of all modified cells in the revised cube.

Thus, we need 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 present an algorithmic approach that takes as input a set of exception rules R, the dimension instances $d_1, \ldots d_k$, represented as tables $T_{d_1}, \ldots, T_{d_k}$, and a cube C defined over some target space $\mathcal{S}$: $(d_1 : targetLevel_1, \ldots d_k : targetLevel_k)$; the algorithm produces a sparse exception cube CLost (alternatively CGained) 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 [1]. The structure of the tables T_{d_i} is described in previous work [MV01].

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 coordinate, 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.level, t.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, Body(r) denotes the highest level in the dimension hierarchy included in a condition defined in the `WHERE` clause of rule r; 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 [MV01] 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 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 Head(r_1)=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 $A_p(coord)$, 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: Head(s) precedes Head(r) and Body(r) precedes 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: Head(r) precedes Head(s) or Body(s)=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 $min(\{Body(s) | s \in A_p\})$; *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.level, t.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 $targetLevel_i : c$ as the coordinate for dimension d_i, will lose (gain) the measures of all cells with t.level: t.source as a coordinate for dimension d_i, and the same coordinates of cell s for the rest of the dimensions.
9. Remove the tuples that appear in both *Lost* and *Gained* sets, associated to some coordinate c by function *DistCalcCoord.*

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. We omit the proof for space reasons.

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

[1] We will only address cubes computed using distributive functions upon insertions and deletions; further, we will only admit dimensions schemas with a linear ordering relation.

Algorithm 2.
Input: A target space $\mathcal{S}(d_1 : targetLevel_1, \ldots, d_k : targetLevel_k)$, and the function *DistCalcCoord* computed by Algorithm 1, for each dimension $d_i, 1 \le i \le k$.
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 $d_i, 1 \le i \le k$
For each coordinate c in the domain of *DistCalcCoord* for dimension d_i

AffectedCells $= \bigcup$ slice cube C to level member $d_i : targetLevel_i : c$, *slice cube* defined by Abello *et al* [ASS01]

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 $z : (d_1 : l_1 : c_1, \ldots, d_k : l_k : c_k)$ in a cube C on space $\mathcal{S} = (d_1 : l_1, \ldots, d_k : l_k)$ (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 *AggregateLost(z)* = *AggregateGained(z)* = ϕ.
Let sets $ALost_i$ and $AGained_i$ be $DistCalCoord(c_i).lost$ and $DistCalCoord(c_i).gained$ respectively, for dimension d_i of space $\mathcal{S}, 1 \le i \le k$.
If some set $ALost_i$ ($AGained_i$)$=\phi$, let set $ALost_i$ ($AGained_i$) $= \{(l_i, c_i)\}$.
$CellsLost = ALost_1 \times \ldots ALost_k$, and
$CellsGained = AGained_1 \times \ldots AGained_k$.
With each tuple t : $((l_1, coord_1), \ldots, (l_k, coord_k))$ in *CellsLost* (*CellsGained*), generate cell $(d_1 : l_1 : coord1, \ldots, d_k : l_k : coord_k)$ and store it into set *AggregateLost(z)* (*AggregateGained(z)*).
For each cell $(d_1 : l_1 : coord1, , d_k : l_k : coord_k)$ in set *AggregateLost(z)* (*AggregateGained(z)*), such that no materialized cube exists on space $\mathcal{S}' = (d_1 : l_1, \ldots, d_k : l_k)$, drill down to a space preceding $\mathcal{S}'$ in the space lattice [HRU96], such that a materialized cube exists, and replace $(d_1 : l_1 : coord1, , d_k : l_k : coord_k)$ 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 *AggregateLost(z)* (*AggregateGained(z)*), according to the aggregation function used to compute cube C, and put the result as the measure of cell z'.

Example 7 *Let us consider the instance of dimension* Customer *in Example 1, and assume the following revision rule set holds over this instance:*

$r_1 : (\nu_2$ `AS` *delay* `SET` *status* = *Consider* `WHERE` $\nu_2.month{=}5$
$r_2 : (\nu_2$ `AS` *delay)* `SET` *status* = *Consider* `WHERE` $\nu_2.month = 6$
$r_3 : (\nu_2$ `AS` *custId)* `SET` *delay* = *6:0* `WHERE` $\nu_2 = c_3$
$r_4 : (\nu_2$ `AS` *custId)* `SET` *grade* = *P* `WHERE` $\nu_2 = c_6$

Let us denote p(c) the path with coordinate c at the bottom level CustID. *When applying Algorithm 1 to dimension* Customer, *and once the iteration on target level* status *had finished, the set* Paths *will contain paths p(c3), p(c6), p(c17), p(c18), with the contents depicted in Figure 6 (the dashed lines indicate the rules modifying the respective path), and the following sets of active exception rules associated with each one of them:* $Ap(c_3) = r_3, Ap(c_6) = r_1, Ap(c_{17}) = r_1, Ap(c_{18}) = r_2$. *Notice that* $Ap(c_3)$ *contains rule* r_3 *and not* r_2. *This occurs because* r_3 *never satisfied the condition in step 3 of Algorithm 1 (thus, once added it has not been removed), and* r_2 *did not satisfy the condition in step 4 of Algorithm 1 (thus it has never been added). A different situation arises with path* $p(c_6)$: *rule* r_4 *has been removed from set* $Ap(c_6)$, *because the condition in step 3 of Algorithm 1 holds for* r_4, *and* r_1 *has been added to* $Ap(c_6)$ *because the condition in step 4 holds for this rule. The contents of DistCalcCoord are: DistCalcCoord (Consider).lost={}; DistCalcCoord (Consider).gained* $= \{custId : c_3, delay : 6\}$; *DistCalcCoord (Reject).lost* $= \{CustId : c_3, delay : 6, delay : 5\}$; *DistCalcCoord (Reject).gained* $= \{delay : 5\}$; *Figures 7 and 8 show the contents of cubes StatusByMonthLost and StatusByMonthGained (Algorithms 2 and 3).*

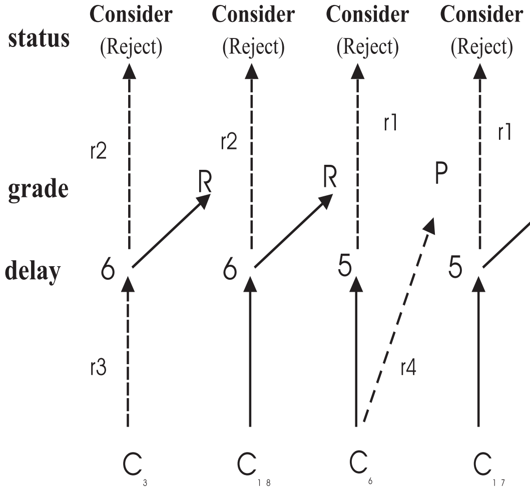

Figure 6: Modified paths after revision.

	Jan-2002	Feb-2002
Consider	0	0
Reject	40,000	250,000

Figure 7: Cube StatusByMonth*Lost*

	Jan-2002	Feb-2002
Consider	40,000	250,000
Reject	0	0

Figure 8: Cube StatusByMonth*Gained*

6 Conclusion

We have presented the IRAH language as a language not only suitable for intensionally defining exceptions to the functional dependencies implied in dimension hierarchies, but also for maintaining materialized cube views under these exceptions. We defined a clear semantics for data cube revision, based on a non-monotonic reasoning framework, introducing the concept of *summarizability under uncertainty.* We proposed an algorithm (with complexity NlogN on the number of tuples in the dimensions' relational representation) that, after finding the cells in the revised cube such that their measures need to be modified, efficiently computes the new measures using materialized cube views at less aggregated levels.

In future research we will explore the application of the rule-based model for data cubes and the notion of summarizability under uncertainty, for producing integrated views of summarized data in a data warehouse with base data provided by non-completely reliable sources, with different degrees of confidence.

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.

[ADGM99] G. Antoniou, Billington D., Governatori G., and Maher M. Revising non-monotonic belief sets: The case of defeasible logic. *Proceedings of the 23th German Conference in Artificial Intelligence, LNAI 1701, Springer*, 1999.

[AGM85] C. Alchouron, P. Gardenfors, and D. Makinson. On the logic of theory change: Partial meet functions for contraction and revision. *Journal of Symbolic Logic 50*, 1985.

[Ant99] G. Antoniou. A tutorial on default logics. *ACM Computing Surveys, Vol. 31, No. 4*, 1999.

[ASS01] A. Abello, J. Samos, and F. Saltor. Understanding analysis dimensions in a multidimensional object-oriented model. In *Proceedings of DMDW'01*, 2001.

[Bre00] G. Brewka. Declarative representation of revision strategies. In *Artificial Intelligence*, 2000.

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

[ER83] D. Etherington and R. Reiter. On inheritance hierarchies with exceptions. In *Proceedings of the AAAI*, 1983.

[HM01] C. Hurtado and A.O. Mendelzon. Reasoning about summarizability in heterogeneous multidimensional schemas. In *Proceedings of the International Conference on Database Theory (ICDT)*, 2001.

[HMV99] C. Hurtado, A.O. Mendelzon, and A. Vaisman. Maintaining data cubes under dimension updates. *Proceedings of IEEE/ICDE'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. Albretch, and H. Wedkind. Normal forms for multidimensional databases. In *Technical Report*, University of Erlangen-Nuremberg, Germany, 1998.

[LS97] H. Lenz and A. Shoshani. Summarizability in OLAP and statistical databases. In *Ninth International Conference on Scientific and Statistical Database Management*, 1997.

[MRA+01] M. Maher, A. Rock, G. Antoniou, D. Billington, and T. Miller. Efficient defeasible reasoning systems. *International Journal on Artificial Intelligence Tools, Vol.10(4)*, 2001.

[MV01] M. Minuto and A. Vaisman. Efficient intensional redefinition of aggregation hierarchies. In *Proceedings ACM-DOLAP'01*, 2001.

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

[Rei80] R. Reiter. A logic for default reasoning. *Artificial Inteligence 13*, 1980.

[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)*, 1998.