Repair-Oriented Relational Schemas for Multidimensional Databases[*]

Mahkameh Yaghmaie Leopoldo Bertossi[†] Sina Ariyan

Carleton University, School of Computer Science, Ottawa, Canada
{myaghmai,bertossi,mariyan}@scs.carleton.ca

ABSTRACT

Summarizability in a multidimensional (MD) database refers to the correct reusability of pre-computed aggregate queries (or views) when computing higher-level aggregations or roll-ups. A dimension instance has this property if and only if it is *strict* and *homogeneous*. A dimension instance may fail to satisfy either of these two semantics conditions, and has to be *repaired*, restoring strictness and homogeneity. In this work, we take a *relational approach* to the problem of repairing dimension instances. A dimension repair is obtained by translating the dimension instance into a relational instance, repairing the latter using established techniques in the relational framework, and properly inverting the process. We show that the common relational *star* and *snowflake* schemas for MD databases are not the best choice for this process. Actually, for this purpose, we propose and formalize the *path relational schema*, which becomes the basis for obtaining dimensional repairs. The path schema turns out to have useful properties in general, as a basis for a relational representation and implementation of MD databases and data warehouses. It is also particularly suitable for restoring MD summarizability through relational repairs. We compare the dimension repairs so obtained with existing repair approaches for MD databases.

Categories and Subject Descriptors

H.2.1 [**Database Management**]: Logical Design—*Data Models, Schema and subsechema*; H.2.7 [**Database Management**]: Database Administration—*Data warehouse and repository, Security, integrity, and protection*

General Terms

Design, Theory, Experimentation

Keywords

Multidimensional DBs, Semantic constraints, Repairs

[*]Research supported by NSERC Strategic Network on Business Intelligence (BIN,ADC02), and NSERC Discovery.
[†]Contact author.

1. INTRODUCTION

Multidimensional (MD) databases (MDDBs) [23] represent data at a high level of abstraction, using multiple *dimensions* to give sense and context to usually quantitative data, the so-called *facts* in data warehouses (DWHs).

Example 1. [9] We have data for a phone company about the cell phones usage depending on time and location. The `Location` dimension represents the hierarchy of the wireless network spots. Each cell phone number has an area code (`41` or `45`), and belongs to a city (`TCH` (Talcahuano), `TEM` (Temuco), or `CCP` (Concepcion)). Area codes and cities belong to a region (`VIII` or `IX`). Figure 1(a) shows the schema for the `Location` dimension, with *categories* `Number`, `Area Code`, etc.; and Figure 1(b) an instance of that schema.

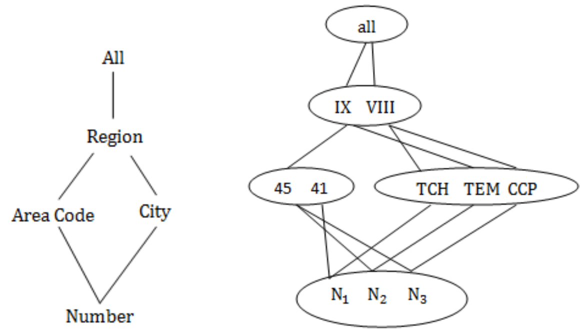
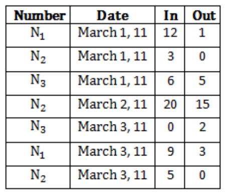


Figure 1: (a) `Location` schema. (b) `Location` instance.

`Region` is a *parent* for categories `Area Code` and `City`; and an *ancestor* of category `Number`. Element $\texttt{N}_2$ of category `Number` has element `IX` as an ancestor in category `Region`. Figure 2 shows data for network usage, linked to `Number` category. □



Number	Date	In	Out
N_1	March 1, 11	12	1
N_2	March 1, 11	3	0
N_3	March 1, 11	6	5
N_2	March 2, 11	20	15
N_3	March 3, 11	0	2
N_1	March 3, 11	9	3
N_2	March 3, 11	5	0

Figure 2: Cell phone traffic *facts* database.

In MDDBs, it should be possible to use results for an aggregate query at a lower levels of the category hierarchy to (correctly) calculate aggregations at higher levels of the same hierarchy. A dimension instance with this property is called *summarizable*, a notion introduced in [29] in the context of statistical databases. A non-summarizable MDDB will return incorrect query results when using pre-computed views, or will force users to compute aggregate query answers and views from scratch, losing efficiency [23, 27].

For a dimension to be summarizable, first it must be *strict*, meaning that each element in a category has at most one parent element in each upper category [23, 27, 29]. Secondly, it has to be *homogeneous*, i.e. each element in a category has at least one parent element in each parent category [23, 29]. We usually and informally refer to the combination of these two conditions as the *summarizability conditions*.

In Example 1, strictness is violated, because `N3` has two grandparents in the `Region` category (`IX` and `VIII`). Moreover, the `Location` dimension is non-homogeneous, i.e. heterogenous: element `41` has no parent in category `Region`.

For design reasons, a dimension instance may become non-summarizable [23]; also after dimension updates. Non-strictness and heterogeneity are common in MDDBs (and DWHs). Such MDDBs are said to be *inconsistent*. In such a case, one can try to restore the properties of strictness and homogeneity, through a *repair* process [2].

Repairs of relational databases that violate integrity constraints (ICs) have been investigated in the literature (cf. [7, 8] for surveys) . Intuitively, a repair of a relational instance D that does not satisfy a given set IC of ICs is an instance D' for the same schema, that does satisfy IC and minimally departs from D. Much work has been done in the area of relational repairs and *consistent query answering* [2].

A few recent approaches to repairing MDDBs have been formally proposed. Non-summarizability is resolved by changing either the dimension instance or the dimension schema. Instance-based repairs have been introduced and studied in [9, 11, 14, 16] (cf. also [28]). Schema-based repairs have been proposed in [21, 22], and formally defined and investigated in [5]. None of these proposals assume a relational representation of MDDBs or appeal to specific implementations of MDDBs. They work directly with the MD model.

It is common that MDDBs are implemented as relational databases (ROLAP), which facilitates optimized query answering and data storage [24, 30]. This is one reason why, in this work, we address non-summarizability, as caused by heterogeneity or non-strictness, through a relational representation of MDDBs. This choice also allows us to investigate the applicability of notions and mechanisms developed for *relational repairs* when dealing with inconsistent MDDBs, leveraging an existing rich body of research. We restore summarizability by repairing the underlying relational database.

To achieve this goal and propose a sensible characterization of MD repairs, we have to start by representing an initial, possibly inconsistent MDDB as a relational database, through an *MD2R mapping*. This mapping translates the multidimensional data model (MDM) into an *adequate* relational model. The latter includes a relational schema that allows for the adequate representation of the MDDB conditions of strictness and homogeneity as relational integrity constraints (ICs). The translation is such, that the original MDDB is inconsistent iff the resulting relational database is inconsistent wrt the created ICs.

The resulting inconsistent relational instance is repaired as a relational database, applying existing concepts and techniques. As a result, we obtain a set of *minimal relational repairs*. Finally, those repairs are translated back into MD repairs. As expected, the feasibility of this approach depends on the *invertibility* of the MD2R mapping [4, 6].

	A^{Number}	A^{AreaCode}	A^{Region}	A^{All}
1	N_1	41	NULL	NULL
2	N_2	45	IX	all
3	N_3	45	IX	all

	A^{Number}	A^{City}	A^{Region}	A^{All}
1	N_1	TCH	VIII	all
2	N_2	TEM	IX	all
3	N_3	CCP	VIII	all

Figure 3: *Path* instance with tables RP_1^{Loc}, resp., RP_2^{Loc}, for left, resp. right paths in `Location` dimension schema.

For this whole program to work, we need to identify an *expressive* relational representation for MDDBs that enables the efficient verification of summarizability conditions through an associated set of ICs. Moreover, there should be *no information loss* under the mapping and its inversion.

For motivation and comparison, we first show that the two well-known relational representations of MDDBs, *Star* and *Snowflake*, are not appropriate for our purpose. Next, we define a new, alternative relational representation of MDDBs, based on what we call the *path relational schema*.

Example 2. (example 1 continued) Figure 3 shows the `Location` dimension represented through the path schema. Each relational table represents a path from the bottom-most category to the top-most category in the dimension schema. The hierarchy in Figure 1(a) contains two category paths, which leads to two tables. Each path goes through several data elements in the dimension instance. The sequence of elements on a path creates a tuple for the table. A^c denotes the attribute corresponding to category c. □

Via the *relational path schema* we can express and efficiently check the strictness and homogeneity conditions through relational ICs. The MD2R mapping turns out to be uniquely invertible. Our results show the adequacy of our approach to MD inconsistency handling via repairs of the associated path relational instances. Notice that our MD repairs are instance-based, as opposed to schema-oriented: The original MD schema is not changed. Only the MD instance is changed via its transformation into a relational instance and subsequent repairs.

We obtain a class of MD repairs that differs from the class of MD (also instance-oriented) repairs proposed in [11, 16] ([9, 14] deal only with non-strictness, assuming homogeneity). This discrepancy is due to the kind of minimality of repairs that we impose on the relational side.

The relational repairs that we obtain can also be considered as simpler than those obtained by applying the same kind of process (relational transformation followed by relational repair) to more classic relational representations, like the star or snowflake (cf. Section 3). We use just changes of attribute values, whereas for the latter two cases we may need full tuple insertions or deletions. In Section 8 we provide experimental evidence in favor of using our path relational schemas as a basis for implementation of MDDBs and DWHs. The experiments focus on performance at aggregate query answering and inconsistency checking, independently from repair computation.

This paper is structured as follows. Section 2 describes the MD data model we use in our work. Section 3 shows that the star and snowflake schemas are not appropriate for dealing with inconsistency issues in MDDBs and DWHs. Section 4 proposes and formalizes the *path relational schema* as a new relational representation for MDDBs. Section 5 discusses the representation of summarizability conditions as ICs over a path schema. Section 6 provides the relational repair semantics for restoring consistency in path databases. Section 7 investigates the invertibility of the MD2R mapping. Section 8 presents a purely MD characterization of the repairs obtained through the relational route. Section 9 shows experiments in relation to the use of the path relational schema as a basis for MDDB and DWH implementation. Finally, Section 10 draws some conclusions and points to ongoing and future work. An extended version of this paper can be found at [31]. It contains, in particular, the proofs missing here, more examples and discussions, and also more experimentation details.

2. PRELIMINARIES

We use the Hurtado-Mendelzon formalization of multidimensional DBs [23]. In it, a *dimension schema* $\mathcal{S}$ is a directed acyclic graph, represented by a pair of the form $\langle \mathcal{C}, \nearrow \rangle$. $\mathcal{C}$ is a set of categories, and $\nearrow$ is a binary relation between categories, indicating the *parent-child relationship* in the schema. Its transitive and reflexive closure is $\nearrow^*$. There are no "shortcuts", i.e. if $c_i \nearrow c_j$, there is no (properly) intermediate category c_k with $c_i \nearrow^* c_k$ and $c_k \nearrow c_j$. A dimension schema has a top category, `All`, reachable from every other category: For every $c \in \mathcal{C}$, $c \nearrow^*$ `All` holds. There is also *a unique category* with no children, the so-called *base category.*

A *dimension instance*, $\mathcal{D}$, of the dimension schema is a pair $\langle \mathcal{M}, < \rangle$, where $\mathcal{M}$ is the finite set of data elements, and $<$ (or $<_{\mathcal{D}}$) is binary relation on $\mathcal{M}$, the *parent-child relationship*, that parallels relation $\nearrow$. More precisely, there is a mapping δ from $\mathcal{M}$ to $\mathcal{C}$ that assigns each data element to a unique category. If $\delta(m) = c$, we also say that $m \in c$. In consequence, $m_1 < m_2$ iff $\delta(m_1) \nearrow \delta(m_2)$. The transitive and reflexive closure of $<$ is $<^*$. Element `all` $\in \mathcal{M}$ is the only element of category `All`, and is expected to be reached from any other element, but this may not necessarily hold.

For a pair of categories c_i, c_j, with $c_i \nearrow^* c_j$, the *roll-up relation* $\mathcal{R}_{c_i}^{c_j}(\mathcal{D})$ contains the pairs $(m_i,\ m_j)$, with $m_i <^* m_j$, $m_i \in c_i, m_j \in c_j$. The roll-up relation is not necessarily a function. Nor has it to be *total*, i.e. there may be $m_i \in c_i$ that does not roll up to an element in c_j.

Example 3. The `Location` dimension schema in Figure 1(a) can be modeled through the following schema $\mathcal{S}$:

$$\mathcal{C} = \{\texttt{Number}, \texttt{AreaCode}, \texttt{City}, \texttt{Region}, \texttt{All}\}.$$

$$\nearrow = \{(\texttt{Number}, \texttt{AreaCode}), (\texttt{Number}, \texttt{City}), (\texttt{AreaCode}, \texttt{Region}), (\texttt{City}, \texttt{Region}), (\texttt{Region}, \texttt{All})\}.$$

For the corresponding dimension instance $\mathcal{D}$, we have:

$$\mathcal{M} = \{\texttt{N}_1, \texttt{N}_2, \texttt{N}_3, \texttt{41}, \texttt{45}, \texttt{TCH}, \texttt{TEM}, \texttt{CCP}, \texttt{VIII}, \texttt{IX}, \texttt{all}\}.$$

$$< = \{(\texttt{N}_1, \texttt{41}), (\texttt{N}_1, \texttt{TCH}), (\texttt{N}_2, \texttt{45}), (\texttt{N}_2, \texttt{TEM}), (\texttt{N}_3, \texttt{45}), (\texttt{N}_3, \texttt{CCP}), (\texttt{45}, \texttt{IX}), (\texttt{TEM}, \texttt{IX}), (\texttt{TCH}, \texttt{VIII}), (\texttt{CCP}, \texttt{VIII}), (\texttt{VIII}, \texttt{all}), (\texttt{IX}, \texttt{all})\}.$$

$\mathcal{R}_{\texttt{Number}}^{\texttt{Region}}(\mathcal{D}) = \{(\texttt{N}_1, \texttt{VIII}), (\texttt{N}_2, \texttt{IX}), (\texttt{N}_3, \texttt{VIII}), (\texttt{N}_3, \texttt{IX})\}$ computes ancestors in `Region` of base elements. □

The MD semantic conditions of strictness and homogeneity, together, guarantee the summarizability property for a dimension instance. They are usually global conditions, but they can also be imposed locally.

Definition 1. [11, 16] (a) For a dimension schema $\mathcal{S} = \langle \mathcal{C}, \nearrow \rangle$, a *local strictness constraint* is an expression of the form $c_i \rightarrow c_j$, where $c_i, c_j \in \mathcal{C}$, $c_i \neq c_j$, and $c_i \nearrow^* c_j$. It is satisfied by instance $\mathcal{D}$, denoted $\mathcal{D} \models\ c_i \rightarrow c_j$, iff $\mathcal{R}_{c_i}^{c_j}(\mathcal{D})$ is a (possibly partial) function.
(b) Instance $\mathcal{D}$ is *strict* if it satisfies the *full-strictness condition*, i.e. the whole set $FS^{\mathcal{S}} = \{c_i \rightarrow c_j \mid c_i, c_j \in \mathcal{C}, c_i \neq c_j, \text{ and } c_i \nearrow^* c_j\}$. □

Example 4. The instance for the `Location` dimension in Figure 1 is non-strict: The roll-up relation $\mathcal{R}_{\texttt{Number}}^{\texttt{Region}}(\mathcal{D})$ in Example 3 is not a function ($\texttt{N}_3$ has two grand parents in category `Region`). Thus, $\mathcal{D} \not\models \texttt{Number} \rightarrow \texttt{Region}$. □

Definition 2. [11, 16] (a) For a dimension schema $\mathcal{S} = \langle \mathcal{C}, \nearrow \rangle$, a *local homogeneity constraint* (a.k.a. *covering constraint*) is an expression of the form $c_i \Rightarrow c_j$, with $c_i, c_j \in \mathcal{C}$, $c_i \neq c_j$, and $c_i \nearrow c_j$. It is satisfied by instance $\mathcal{D}$, denoted $\mathcal{D} \models\ c_i \Rightarrow c_j$, iff $\mathcal{R}_{c_i}^{c_j}(\mathcal{D})$ is *total.*
(b) Instance $\mathcal{D}$ is *homogenous* iff it satisfies the *full-homogeneity condition*, i.e. the whole set $FH^{\mathcal{S}} = \{c_i \Rightarrow c_j \mid c_i, c_j \in \mathcal{C},\ c_i \neq c_j, \text{ and } c_i \nearrow c_j\}$. □

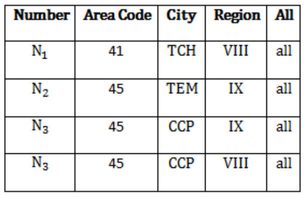
Example 5. For the instance in Figure 1, $\mathcal{R}_{\texttt{AreaCode}}^{\texttt{Region}}$ is $\{(\texttt{45, IX})\}$. Element `41` does not appear as a first argument. Then, the relation is not total, and the instance is heterogenous: $\mathcal{D} \not\models \texttt{AreaCode} \Rightarrow \texttt{Region}$. □

Remark 1. We will make some common assumptions [23] that make the presentation easier: (a) The existence of a single base category. (b) Dimension instances are *complete*, i.e. elements that do not have children are all base elements. (c) Although we will use the null value, `NULL`, in the relational representation of the original MD instance, the latter does not contain `NULL`. Actually, `NULL` $\notin \mathcal{M}$. The semantics of `NULL` will be as in SQL relational databases, with a semantics *à la* SQL, as logically captured in [10]. □

3. ROLAP REVISITED AND MDDBS

The star schema is the most common relational representation of an MD database, with a fact table with measurements that is directly joined to dimension tables that contain descriptive attributes.

We can obtain an example of this representation from Figures 1(b) and 2, if we create a referential IC from the latter to a *single relation* representing the former [21, 25]. In it, the categories are captured as attributes, and each base element with its ancestors generates a tuple for the relational table. Figure 4 shows this representation of the `Location` dimension as a relational instance for a star schema. Base element $\texttt{N}_3$ appears in two tuples, because it has multiple ancestors at the higher category `Region`.



Number	Area Code	City	Region	All
N_1	41	TCH	VIII	all
N_2	45	TEM	IX	all
N_3	45	CCP	IX	all
N_3	45	CCP	VIII	all

Figure 4: Star representation of dimension in Figure 1.

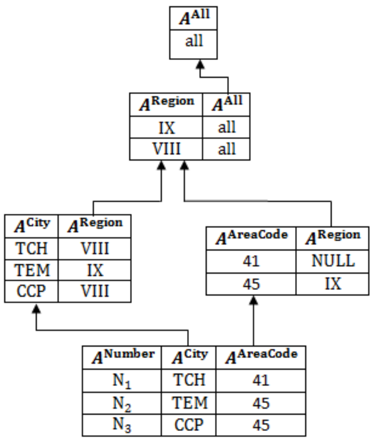
A weaknesses of the star schema as a relational representation of MD summarizability conditions (cf. Section 1) is that checking homogeneity through relational ICs is problematic. We could think that homogeneity is captured through the absence of `NULL`. However, our running example shows that this does not hold: The instance for `Location` in Figure

1 is heterogenous (check 41), but its star representation in Figure 4 has no NULL.

However, checking a strictness constraint between categories can be done via a *functional dependency* (FD) between the corresponding attributes in the star representation.

An additional problem with the star representation is its lack of invertibility: Moving back from a star representation to a MD representation is uncertain. Due to the flat structure of the star schema, we lose information about the original MD2R mapping, and inverting the mapping may not generate a unique MD instance (cf. Section 7).

While the star schema captures a dimension in a flat relational structure, the *snowflake* schema provides a *hierarchical* relational representation. Being a normalized version of a star schema, its hierarchical structure makes query answering more complex [25]. Under this schema, each category c in a dimension schema is represented by a separate table, with A^c as first attribute. The other attributes in that table correspond to the c's parent categories. Each of them points to or *references* the same attribute in its own table [21, 25]. Figure 5 shows the snowflake relational representation of the Location dimension instance.



A^{All}
all

A^{Region}	A^{All}
IX	all
VIII	all

A^{City}	A^{Region}
TCH	VIII
TEM	IX
CCP	VIII

$A^{AreaCode}$	A^{Region}
41	NULL
45	IX

A^{Number}	A^{City}	$A^{AreaCode}$
N_1	TCH	41
N_2	TEM	45
N_3	CCP	45

Figure 5: Snowflake representation of dimension in Figure 1.

The hierarchical structure of snowflake complicates capturing and checking strictness through relational ICs: Since each category is mapped to a single table, this requires executing several joins. For example, if we want to check strictness between categories Number and Region through the schema in Figure 5, we can see from the MD schema that there are two ways to reach category Region from category Number. On the relational side, we have to check each path by joining the corresponding tables, to discover that N_3 is related to different elements in Region.

More generally, checking local strictness conditions via the snowflake representation amounts to checking relational *equality generating dependencies* (EGD) [1], which can be expressed in relational calculus by sentences of the form

$$\forall \bar{x}(\varphi(\bar{x}) \rightarrow x_j^1 = x_j^2), \qquad (1)$$

where φ is a formula that captures the required (and possible multiple) joins, and $x_j^1, x_j^2 \in \bar{x}$.

Unlike strictness, checking homogeneity in a snowflake instance is simple: The presence of NULL reflects the missing parents, like the NULL for Region in Figure 5. Thus, we can check homogeneity through NOT NULL relational constraints (assuming that the original MD instance does not contain null values). Local homogeneity, i.e. MD constraints of the form $c_i \Rightarrow c_j$, can be checked by relational ICs of the form

$$\forall \bar{x}(\psi(\bar{x}) \rightarrow \mathit{NotNull}(x_j)), \qquad (2)$$

with $x_j \in \bar{x}$, and *NotNull* a built-in predicate that is true only when its argument is (symbolically) different from NULL.

The hierarchical structure of the snowflake schema makes the MD2R mapping invertible. For example, the snowflake instance in Figure 5 is uniquely invertible to the Location MD dimension.

4. MDDBS AS PATH INSTANCES

We now propose a relational representation of MDDBs that allows for: (a) a simple representation and verification of summarizability conditions via relational ICs, (b) invertibility of the MD2R mapping; and (c) a simple characterization of relational repairs (leading to MD repairs).

We formulated strictness and homogeneity conditions in terms of roll-up relations, whose elements are pairs of data elements that are connected by a *path* traversing the category hierarchy. Then, in order to better express the summarizability conditions in relational terms, the relational representation will explicitly store those paths. We propose a *path-based mapping* from MDDBs to relational databases (cf. [32] for a somehow similar XML-to-Relational mapping).

Definition 3. For a dimension schema $\mathcal{S} = \langle \mathcal{C}, \nearrow \rangle$, a *base-to-all path* (B2A) P is a list of categories $\langle c_1, \ldots, c_n \rangle$, where c_1 is the base category, c_n is All, and $c_i \nearrow c_{i+1}$. □

The *path relational schema* will represent each B2A path by a *single* predicate. The categories in the path are mapped to attributes for that predicate. A category can be mapped to more than one attribute, in separate tables.

Definition 4. For a dimension schema $\mathcal{S} = \langle \mathcal{C}, \nearrow \rangle$ and dimension instance $\mathcal{D} = \langle \mathcal{M}, < \rangle$, a *p-instance* for a B2A path $P = \langle c_1, \ldots, c_n \rangle$ is a list of elements $p = \langle e_1, \ldots, e_n \rangle$, with:[1] (a) $e_i \in \mathcal{M}$ and $\delta(e_i) = c_i$, or $e_i =$ NULL. (b) When e_i and e_{i+1} are not NULL, $e_i < e_{i+1}$. (c) There is no p-instance p' that can be obtained from p by replacing a NULLs by a non-NULL e_i, and p' satisfies (a)-(b). The set of p-instances for P is denoted by $\mathit{Inst}^{\mathcal{D}}(P)$. □

Condition (c) enforces the use of non-null data elements if possible; equivalently, of NULL only when strictly needed. Notice that NULL is incomparable via $<$ with elements in $\mathcal{M}$. If the base category is non-empty (which we may assume), there will be no p-instance starting with NULL.

Now we describe the ***relational transformation*** $\mathcal{T}$ (or path mapping) of the dimension schema and instance:

(A) (Schema) For each $c \in \mathcal{C}$, create a relational attribute A^c. For each B2A path P of the form $\langle c_1, \ldots, c_n \rangle$, create a relational predicate $RP[A^{c_1}, \ldots, A^{c_n}]$.

(B) (Instance) For each p-instance $p \in \mathit{Inst}^{\mathcal{D}}(P)$ of the form $\langle e_1, \ldots, e_n \rangle$, create the relational tuple $RP(e_1, \cdots, e_n)$.

Example 6. (example 2 continued) Figure 3 shows the application of the *path mapping rules* to the Location dimension. The Location schema in Figure 1 has two B2A paths: $\mathtt{P}_1^{\mathtt{Loc}}$: $\langle$Number, AreaCode, Region, All$\rangle$ (Figure 3 left), and $\mathtt{P}_2^{\mathtt{Loc}}$: $\langle$Number, City, Region, All$\rangle$ (Figure 3,

[1]We use the term "p-instance", because later on we will talk about "path instances", which will be instances of the relational path schema.

right). Each of these paths is mapped to a separate relational table using rule (A), and has 3 associated p-instances. Then, by rule (B), each relation has 3 tuples. For example, the tuple $(\texttt{N}_2, 45, \texttt{IX}, \texttt{all})$ in Figure 3 (left) originates in the p-instance $\langle \texttt{N}_2\texttt{,45, IX, all}\rangle \in \mathit{Inst}^{\mathcal{D}}(P_1^{\texttt{Loc}})$. □

Notice that the active domain $\mathit{Act}(D)$ of the generated relational instance D is contained in $\mathcal{M} \cup \{\texttt{NULL}\}$; and the domain $\mathit{Dom}(A^c)$ of attribute A^c is $\delta^{-1}(c) \cup \{\texttt{NULL}\}$. Then, $\mathit{Dom}(A^c) \subseteq \mathit{Act}(D) \cup \{\texttt{NULL}\}$. We assume $\texttt{all} \in \mathit{Dom}(A^{\texttt{All}})$. Even with $\texttt{all} \in \mathcal{M}$, NULL may be a value for $A^{\texttt{All}}$ if all is not reached by lower-level elements in the MD instance.

Notice that the generated relational schema depends only on the MD schema. In particular, the number of relational tables depends on the number of paths *in the MD schema*, and not on the MD instance.

5. MD CONSTRAINTS AS PATH ICS

An MD schema $\mathcal{S}$ will come endowed with a set $\mathcal{K}$ of (local) strictness and homogeneity constraints (cf. Definitions 1 and 2). An MD instance $\mathcal{D}$ may not satisfy $\mathcal{K}$, which should be reflected in the violation by the associated relational instance D of a corresponding set Σ of relational ICs. We show now the generation of Σ from $\mathcal{K}$.

Checking a strictness condition $c_i \rightarrow c_j$ under the path mapping depends on the set of B2A paths where both c_i, c_j appear. Each such *single path* has to be checked, and also *pairs of paths* containing c_i, c_j. Thus, we need functional dependencies (FDs) for single paths (cf. Rule (C) below), and also an IC for each pairs of paths (cf. Rule (D) below). The latter are simple cases of *equality generating dependencies* (much simpler than (1)). (FDs are also EGDs [1].)

(C) (FD generation) $c_i \rightarrow c_j \;\;\mapsto$

$$\{RP\colon A^{c_i} \rightarrow A^{c_j} \mid P \text{ is a B2A path with } c_i, c_j \in P\}.$$

(D) (EGD generation) $c_i \rightarrow c_j \;\;\mapsto$

$$\{RP_1.A^{c_i} = RP_2.A^{c_i} \rightarrow RP_1.A^{c_j} = RP_2.A^{c_j} \mid P_1, P_2 \text{ is a pair of B2A paths with } c_i, c_j \in P_1 \cap P_2\}.[2]$$

Rule (C) can be obtained as a special case of Rule (D).

Example 7. (example 2 continued) Global strictness conditions for the Location dimension lead to the following (reduced) set of FDs:

$$RP_1^{\texttt{Loc}}: \quad \{A^{\texttt{Number}} \rightarrow A^{\texttt{AreaCode}},\ A^{\texttt{AreaCode}} \rightarrow A^{\texttt{Region}}\}. \quad (3)$$
$$RP_2^{\texttt{Loc}}: \quad \{A^{\texttt{Number}} \rightarrow A^{\texttt{City}},\ A^{\texttt{City}} \rightarrow A^{\texttt{Region}}\}. \quad (4)$$

Now, Number and Region are the only categories that require Rule (D), because they reside on two different paths. We generate the following EGD:

$$RP_1^{\texttt{Loc}}.A^{\texttt{Number}} = RP_2^{\texttt{Loc}}.A^{\texttt{Number}} \;\rightarrow\; RP_1^{\texttt{Loc}}.A^{\texttt{Region}} = RP_2^{\texttt{Loc}}.A^{\texttt{Region}}. \quad (5)$$

These ICs are checked on the generated path instance. □

Since we may have introduced NULL in a path relational instance, we have to take them into account when checking IC satisfaction in it. Several semantics have been proposed for relational databases with null values. Here, we are using a single null value, NULL, which we expect to behave as in SQL databases. In [10] a precise definition of IC satisfaction in relational DBs with SQL NULL was given, through a reconstruction in classical predicate logic. We adopt it here, without going into the details. The main element in this approach is the separation of attributes into *relevant* and *non-relevant*, depending on their occurrence or not in dependencies, and their possibility of taking value NULL. For example, a join attribute is relevant.

More precisely, a relational dependency ψ is rewritten into a sentence ψ^N, such that, for a relational instance D,

$$D \models_N \psi \;:\Longleftrightarrow\; D^{rel} \models \psi^N. \quad (6)$$

Here, $\models_N$ denotes the (new) notion of IC satisfaction in databases with NULL. On the RHS, we have usual first-order satisfaction with NULL treated as any other constant.[3] Instance D^{rel} is obtained by restricting D to its relevant attributes, and ψ^N is a syntactic rewriting of ψ that takes into account the possible occurrence of NULL.

Example 8. (example 7 continued) Dependency (5) written as a first-order sentence becomes:

$$\psi\colon\; \forall n \forall a \forall r \forall x \forall c \forall r' \forall y\; (RP_1^{\texttt{Loc}}(n,a,r,x) \wedge RP_2^{\texttt{Loc}}(n,c,r',y) \rightarrow r = r').$$

This formula does not take the possible occurrence of NULL (with its intended semantics) into account. This is done by its rewriting into ψ^N, through ψ's set of relevant attributes [10], namely, $\mathit{Rel} = \{RP_1^{\texttt{Loc}}.A^{\texttt{Number}},\ RP_1^{\texttt{Loc}}.A^{\texttt{Region}},\ RP_2^{\texttt{Loc}}.A^{\texttt{Number}},\ RP_2^{\texttt{Loc}}.A^{\texttt{Region}}\}$:

$$\psi^N\colon\; \forall n \forall r \forall r' (RP_1^{\texttt{Loc},Rel}(n,r) \wedge RP_2^{\texttt{Loc},Rel}(n,r')) \;\wedge\; \mathit{NotNull}(n) \wedge \mathit{NotNull}(r) \wedge \mathit{NotNull}(r') \;\rightarrow\; r = r'). \quad (7)$$

This IC is checked on the instance D^{rel} that results from restricting the instance D in Figure 3 to the attributes in Rel. D^{rel} has predicates $RP_1^{\texttt{Loc},rel}$ and $RP_2^{\texttt{Loc},rel}$. It is easy to check that for $n = \texttt{N}_3$, $r = \texttt{IX}$ and $r' = \texttt{VIII}$, (7) is not true in D^{rel}, with evaluation done classically and treating NULL as any other constant. Then, by (6), $D \not\models_N \psi$, which agrees with the local non-strictness of the original MD instance. □

Homogeneity can be checked via the path instance D with NOT NULL constraints (Rule (E) below). A NULL in a tuple of D shows that the preceding elements in the corresponding p-instance do not have ancestors all the way up. The two NULLs in table $RP_1^{\texttt{Loc}}$ (Figure 3) show that the path is *discontinued* after element 41.

(E) (NOT NULL generation) $c_i \Rightarrow c_j \;\;\mapsto$

$$\{\texttt{NOT NULL}\ RP.A^{c_j} \mid P \text{ is a B2A path with } c_i, c_j \in P\}.$$

All the ICs introduced in (C)-(E) can be easily written as first-order sentences of the forms (1) or (2) (cf. Example 8).

Example 9. (example 2 continued) Homogeneity of the Location instance can be checked with the NOT NULL constraints on the path instance:[4]

$$\texttt{NOT NULL} \quad RP_1^{\texttt{Loc}}.\{A^{\texttt{AreaCode}}, A^{\texttt{Region}}, A^{\texttt{All}}\}, \quad (8)$$
$$\texttt{NOT NULL} \quad RP_2^{\texttt{Loc}}.\{A^{\texttt{City}}, A^{\texttt{Region}}, A^{\texttt{All}}\}. \quad (9)$$

As expected, the path instance in Figure 3 violates the constraints NOT NULL $RP_1^{\texttt{Loc}}.A^{\texttt{Region}}$ and NOT NULL $RP_1^{\texttt{Loc}}.A^{\texttt{All}}$. □

6. REPAIRING PATH INSTANCES

The mappings we introduced are such, that the MD instance is non-summarizable iff the generated path instance is inconsistent wrt the relational ICs. If the latter happens, we

[2]Slightly abusing notation, here we are treating paths as sets of categories.

[3]In particular, the *unique names assumption* applies to it.

[4]The first element in a p-instance is never NULL (cf. Def. 4).

can use relational repairs. This requires introducing appropriate relational repair operations for path instances; and a notion of distance between instances, to capture minimality.

The relational ICs of Section 5 are *denial constraints*, which can be enforced through tuple deletions or changes of attribute values. Deleting a tuple from a path instance implicitly removes a p-instance from the dimension, which would lead to the loss of MD data, making inversion problematic. We adopt here repairs that are obtained via a *minimum number of changes of attribute values*. These *attribute-based repairs* have been used and investigated before, for denial constraints and FDs (cf. [8] for references).

Definition 5. Let D be a relational instance, possibly with NULL. (a) An *atomic update* on D is represented by a triplet $\langle R(\bar{t}), A, v\rangle$, where v is a *new* value in $Dom(A) \smallsetminus \{\texttt{NULL}\}$ assigned to attribute A in the database atom $R(t) \in D$.[5] (b) An *update* on D is a finite set ρ of atomic updates (not assigning more than one new value to an existing attribute value $\bar{t}[A]$). $\rho(D)$ denotes the instance resulting from applying (simultaneously all the updates in) ρ to D. (c) For a set Σ of denial constraints (for D's schema), an update ρ on D is a *minimal repair* iff: 1. $\rho(D) \models_N \Sigma$, and 2. there is no ρ', such that $\rho'(D) \models_N \Sigma$, and $|\rho'| < |\rho|$. □

An atomic update changes an existing value in the database by a new non-null value that is already present in the database. For our purposes, in Definition 5 we can consider only sets Σ of denial constraints of the forms (C), (D), or (E), i.e. EGDs and NOT-NULL constraints. In the following we will assume that this is the case. For a repair ρ of D, we also call (the result) $\rho(D)$ a repair of D.

Example 10. (examples 7 and 9 continued) For the path instance D in Figure 3, and the relational ICs in Examples 7 and 9, the following are repair candidates, among others (for simplicity we use only the tuple ids of Figure 3):

$$\begin{aligned}\rho_1 \;&=\; \{\langle RP_1^{\texttt{Loc}}(1), A^{\texttt{Region}}, \texttt{VIII}\rangle,\ \langle RP_1^{\texttt{Loc}}(1), A^{\texttt{All}}, \texttt{all}\rangle,\\ &\qquad \langle RP_2^{\texttt{Loc}}(3), A^{\texttt{Region}}, \texttt{IX}\rangle\},\\ \rho_2 \;&=\; \{\langle RP_1^{\texttt{Loc}}(1), A^{\texttt{Region}}, \texttt{VIII}\rangle,\ \langle RP_1^{\texttt{Loc}}(1), A^{\texttt{All}}, \texttt{all}\rangle,\\ &\qquad \langle RP_1^{\texttt{Loc}}(3), A^{\texttt{AreaCode}}, \texttt{41}\rangle,\ \langle RP_1^{\texttt{Loc}}(3), A^{\texttt{Region}}, \texttt{VIII}\rangle\}.\end{aligned}$$

Both of these (and other) updates restore the consistency of D. However, ρ_1 is the only minimal repair as of Definition 5. It changes the value of $A^{\texttt{Region}}$ in the third tuple of $RP_2^{\texttt{Loc}}$, from VIII to IX. On the MD side, this change amounts to modifying the link from element CCP to category Region. Still on the MD side, ρ_1 also creates an originally missing link, by assigning VIII as the parent of 41. This is done by ρ_1 on the relational side by updating the first tuple in table $RP_1^{\texttt{Loc}}$. Figure 6 shows this minimal repair of the original path instance.

ρ_2 is not minimal, but it still restores both strictness and homogeneity on the MD side, by indirectly modifying the link between $\texttt{N}_3$ and category AreaCode, and also creating a link from 41 to VIII.

Directly on the MD side, changing the parent of $\texttt{N}_3$ from CCP to TEM restores strictness of the Location dimension. If we also add a link between 41 and VIII, we obtain an MD "repair" corresponding to the following relational update:

$$\begin{aligned}\rho' \;&=\; \{\langle RP_1^{\texttt{Loc}}(1), A^{\texttt{Region}}, \texttt{VIII}\rangle,\ \langle RP_1^{\texttt{Loc}}(1), A^{\texttt{All}}, \texttt{all}\rangle,\\ &\qquad \langle RP_2^{\texttt{Loc}}(3), A^{\texttt{City}}, \texttt{TEM}\rangle,\ \langle RP_2^{\texttt{Loc}}(3), A^{\texttt{Region}}, \texttt{IX}\rangle\}.\end{aligned}$$

[5]As usual in relational DBs, we denote the value for attribute A in a tuple $R(\bar{t})$ with $R(\bar{t})[A]$, or simply $\bar{t}[A]$ when predicate R is clear from the context.

A^{Number}	A^{AreaCode}	A^{Region}	A^{All}
N_1	41	~~NULL~~ VIII	~~NULL~~ all
N_2	45	IX	all
N_3	45	IX	all

A^{Number}	A^{City}	A^{Region}	A^{All}
N_1	TCH	VIII	all
N_2	TEM	IX	all
N_3	CCP	~~VIII~~ IX	all

Figure 6: Minimal repair of the path instance.

ρ', compared with ρ_1, makes an *unnecessary* update on $A^{\texttt{City}}$, in the third tuple of $RP_2^{\texttt{Loc}}$. Hence, according to Definition 5, ρ' is not a minimal relational repair. □

Notice that NULLs are updated in the derived path instance only when there is a NOT NULL constraint violation. Since we might be interested in checking some of, but not necessarily all, the possible homogeneity constraints, we would have some attributes that are not restricted by a NOT NULL constraint. In this case, a minimal relational repair might still have NULLs (cf. Example 11 in [31]). On the other side, restoring consistency wrt strictness constraints via relational repairs does not modify the NULL values in the database. The minimal repairs of a path instance associated to an MD instance that violates homogeneity are NULL-free if homogeneity is globally imposed.

The semantics we are using for evaluating FDs and EGDs in presence of NULL does not consider NULL as a source of IC violation. Since strictness is violated when we have more than one parent for an element in an upper category, strictness seen from the relational side is not violated if an element rolls up to a non-null value and NULL in a parent category. The latter is reached on the relational side due to a *missing parent* on the MD side.

Example 11. (example 8 continued) Consider the EGD (5) obtained from the MD strictness constraint Number → Region. Sentence (7) can be used for checking (5) in the presence of NULL.

In order to illustrate the effect of NULL on the evaluation of Number → Region via the instance in Figure 3, we instantiate (7) on the first tuples of tables $RP_1^{\texttt{Loc}}$ and $RP_2^{\texttt{Loc}}$, obtaining

$$\begin{aligned}&RP_1^{\texttt{Loc},Rel}(\texttt{N}_1, \texttt{NULL}) \wedge RP_2^{\texttt{Loc},Rel}(\texttt{N}_1, \texttt{VIII})) \wedge NotNull(\texttt{N}_1) \wedge\\ &\qquad NotNull(\texttt{NULL}) \wedge NotNull(\texttt{VIII}) \;\rightarrow\; \texttt{NULL} = \texttt{VIII}.\end{aligned}$$

Due to the occurrence of NULL in relevant attributes, the antecedent of the implication has the false conjunct $NotNull($ NULL), which makes the whole sentence true. Hence, the instance in Figure 3, even having $\texttt{N}_1$ associated to both NULL and VIII does not violate the EGD. This makes sense, because NULL was introduced as an auxiliary relational element to deal with heterogeneity; and it does not appear on the MD side. In the corresponding MD instance $\texttt{N}_1$ is connected only to VIII (cf. Figure 1(b)). □

Theorem 1. For a relational path instance D and a set Σ of relational ICs associated to an MD instance $\mathcal{D}$ with a set $\mathcal{K}$ of MD strictness and homogeneity constraints, there always exists a minimal repair wrt Σ.[6] □

[6]Cf. [31] and Example 13 there for a proof and example.

7. BACK TO MD INSTANCES: INVERSION

We will define MD repairs (cf. Definition 6) via the translation of the MD database into a relational instance subject to relational ICs that are derived from semantic MD constraints. The generated relational instance is repaired wrt the ICs. In Section 8 we will address this question about the kind of repairs that are obtained going through the relational route. In this section, we will concentrate on the *invertibility* of the relational mapping $\mathcal{T}$ (introduced in Section 4), i.e. on how to obtain an MD instance from a given (relational) path instance. This is a question about schema mappings and their invertibility [4, 6].

Mapping $\mathcal{T}$ has two components, for the schema and the instance. The former includes a transformation of a set of MD constraints into relational ICs. We expect this two-part mapping $\mathcal{T}$ to have an inverse $\mathcal{T}^{-1}$ with good properties:

Expected properties:

(E1) $\mathcal{T}^{-1}(\mathcal{T}(\mathcal{S})) = \mathcal{S}$, where $\mathcal{S}$ is the MD schema.

(E2) $\mathcal{T}^{-1}(\mathcal{T}(\mathcal{D})) = \mathcal{D}$, where $\mathcal{D}$ is the MD instance.

(E3) For an MD instance $\mathcal{M}$, and a set of MD constraints $\mathcal{K}$, if D and Σ are their relational counterparts, and $\rho(D)$ is a repair of D wrt Σ, then $\mathcal{T}^{-1}(\rho(D)) \models \mathcal{K}$ holds.

We proceed as follows. First, we define the domain of $\mathcal{T}^{-1}$, next we define $\mathcal{T}^{-1}$ with transformation rules, and finally, we check that $\mathcal{T}^{-1}$ has the expected properties. Mapping $\mathcal{T}^{-1}$ is defined on triples $\langle \mathcal{R}, \Sigma, D\rangle$, where $\mathcal{R}$ is a path schema, Σ is a set of ICs over $\mathcal{R}$, and D is an instance for $\mathcal{R}$, s.t.:

1. For every predicate $R[A_1, \cdots, A_n] \in \mathcal{R}$, $A_n = A^{\texttt{All}}$, and $Dom(A^{\texttt{All}}) = \{\texttt{all}, \texttt{NULL}\}$. All predicates R share the "first attribute" A_1, and $\texttt{NULL} \notin Dom(A_1)$ (we assume there is a single base category). Other attributes are allowed to take the value NULL. Different predicates R_i may share attributes other than A_1 and $A^{\texttt{All}}$.
2. The elements of Σ are of the form: (a) NOT NULL $R_i.A_j$, or (b) $R_i.A_k = R_j.A_l \rightarrow R_i.A_m = R_j.A_n$, with R_i, R_j not necessarily distinct predicates in $\mathcal{R}$.
3. D satisfies the basic conditions in item 1. However, it may be that $D \not\models_N \Sigma$. We can also assume, but this is not crucial, that for every tuple $R(e_1, \ldots, e_n) \in D$, if $e_i = \texttt{NULL}$, then $e_j = \texttt{NULL}$ for every $i \leq j \leq n$.

For the definition of $\mathcal{T}^{-1}(\mathcal{R})$, we associate attributes to categories. The joint and consecutive occurrence of attributes in a relational predicate generates direct links between categories (cf. rule (F) below). The definition of $\mathcal{T}^{-1}(D)$ is given by considering each tuple separately, creating a corresponding p-instance for a dimension instance $\mathcal{M}$ for MD schema $\mathcal{S} := \mathcal{T}^{-1}(\mathcal{R})$.[7] This creates elements in categories and links between elements belonging to directly connected categories (cf. rule (G) below).

(F) (Schema inversion) For an attribute A appearing in an $R \in \mathcal{R}$, create a category (name) c^A. The set of so-created categories is denoted with $\mathcal{C}$. For each relational predicate $R[A_1, \ldots, A_n]$ in $\mathcal{R}$ and $1 \leq i \leq n-1$, create an edge from c^{A_i} to $c^{A_{i+1}}$ in the dimension schema.

[7]Since the attributes A_j in $\mathcal{R}$ may not have names of the form A^c, for c a category name, we will obtain an MD schema that will be "isomorphic" to the original $\mathcal{S}$, if any. We will still denote with $\mathcal{S}$ the MD schema resulting from the inversion.

(G) (Instance inversion) For each atom $R(e_1,\ldots, e_n)$, with $R[A_1, \ldots, A_n] \in \mathcal{R}$, and $1 \leq i \leq n-1$, if $e_i \neq \texttt{NULL}$, introduce e_i as an element of (the extension of) c^{A_i} (i.e. make $\delta(e_i) := c^{A_i}$). If $e_{i+1} \neq \texttt{NULL}$, create an edge from e_i to e_{i+1}.

Example 12. (example 10 continued) Figure 6 shows the minimal repair of our path instance. Applying to it the inversion rules just introduced, we obtain the dimension instance in Figure 7. Dashed lines correspond to inserted edges. More specifically, using rule (F) on $RP_1^{\texttt{Loc}}$, the top table in Figure 6, creates a set of categories:[8] $RP_1^{\texttt{Loc}}[A^{\texttt{Number}}, A^{\texttt{AreaCode}}, A^{\texttt{Region}}, A^{\texttt{All}}] \mapsto \{\texttt{Number}, \texttt{AreaCode}, \texttt{Region}, \texttt{All}\} \subseteq \mathcal{C}$, and also a set of $\nearrow$-links: $\{\langle\texttt{Number}, \texttt{AreaCode}\rangle, \langle\texttt{AreaCode}, \texttt{Region}\rangle, \langle\texttt{Region}, \texttt{All}\rangle\} \subseteq \nearrow$.

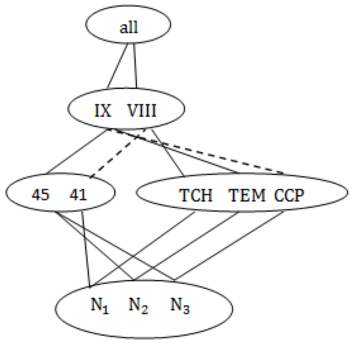


Figure 7: Dimension instance obtained via inversion.

For instance inversion with rule (G), we map, e.g., the updated (first) tuple $(\texttt{N1}, \texttt{41}, \texttt{VIII}, \texttt{all})$ in the top table:

$$RP_1^{\texttt{Loc}}(\texttt{N1}, \texttt{41}, \texttt{VIII}, \texttt{all}) \mapsto \{\texttt{N}_1, \texttt{41}, \texttt{VIII}, \texttt{all}\} \subseteq \mathcal{M},$$
$$\delta(\texttt{41}) = \texttt{AreaCode},\ \delta(\texttt{VIII}) = \texttt{Region},\ \delta(\texttt{all}) = \texttt{All},$$
$$\texttt{N}_1 < \texttt{41},\ \texttt{41} < \texttt{VIII},\ \texttt{VIII} < \texttt{all}.$$

Applying rule (G) to other tuples in the relational (minimal repair) instance D, we obtain an MD instance $\mathcal{D}$, the one in Figure 7. It turns out to be (globally) strict and homogeneous. In fact, for D in Figure 6, since $D \models \Sigma$ (with Σ as in Example 10), it holds $\mathcal{D} \models \mathcal{K}$. Here, $\mathcal{K}$ is the original set of MD constraints that gave rise to Σ. □

It easy to verify that a *unique* dimension instance is obtained as a result of applying the inversion rules, and that the expected properties (E1)-(E3) hold. Furthermore, it is also easy to verify that both $\mathcal{T}$ and $\mathcal{T}^{-1}$ can be computed in polynomial time.

8. A PURELY MD REPAIR SEMANTICS

With all the above elements we are ready to define *a repair semantics for MD databases wrt summarizability constraints.* The definition is *indirect*, in the sense that we first map the MD schema $\mathcal{S}$ and instance $\mathcal{D}$ to a relational schema $\mathcal{R}$ and instance D, resp. Furthermore, the set $\mathcal{K}$ of *local summarizability constraints*, i.e. local strictness and homogeneity constraints, on the MD side is translated into a set Σ of relational ICs. So as $\mathcal{D}$ may not satisfy $\mathcal{K}$, D may not satisfy Σ. Then, we consider relational repairs for D wrt Σ. These are *attribute-based repairs* [8] that change a minimum number of attribute values. Those relational minimal repairs form a class denoted by $Rep(D, \Sigma)$. Inverting the relational repairs takes us to a class of MD instances $\mathcal{D}'$ that satisfy $\mathcal{K}$.

[8]Naturally identifying the generated category c^{A^c} with c.

Definition 6. Let $\mathcal{D}$ be an MD instance, $\mathcal{K}$ a set of *local summarizability constraints* (LSCs) (i.e. local strictness and local homogeneity constraints), D the instance $\mathcal{T}(\mathcal{D})$, and Σ the class of relational ICs obtained from $\mathcal{K}$. An MD instance $\mathcal{D}'$ is a *path repair* of $\mathcal{D}$ wrt $\mathcal{K}$ iff there is $D' \in Rep(D, \Sigma)$ with $\mathcal{D}' = \mathcal{T}^{-1}(D')$. $Rep(\mathcal{D}, \mathcal{K})$ is the class of path repairs of $\mathcal{D}$ wrt $\mathcal{K}$. $\square$

Since the relational repairs in $Rep(D, \Sigma)$ only change data (not the relational schema), the elements of $Rep(\mathcal{D}, \mathcal{K})$ also change only data, i.e. instance $\mathcal{D}$ (and not the MD schema $\mathcal{S}$). So, they are *instance-oriented repairs* of $\mathcal{D}$. A path-repair of $\mathcal{D}$ does not add new elements to categories. Actually, the MD data repair operations are insertions or deletions of edges in the dimension instance. Violations of `NOT NULL` ICs on the relational side (associated to the lack of homogeneity on the MD side) result in edge insertions for $\mathcal{D}$. Operations tackling non-strictness (EGD violations on the relational side) may be insertions and deletions of links.

There have been previous approaches to MD instance-based repairs [14, 28, 9].[9] In [9], repairs assume homogeneity, and only address strictness. The closest approach to ours is [11] (cf. also [16]). They define MD repairs directly on the MD instance, wrt both local strictness and homogeneity constraints. In [11] a minimal repair is a new dimension instance that is consistent wrt the summarizability constraints, and is obtained by applying a minimal number of updates to the original dimension instance (edge insertions or deletions between existing data elements). Let us denote with $Rep^{bch}(\mathcal{D}, \mathcal{K})$ their class on minimal MD repairs, which can be compared to our class $Rep(\mathcal{D}, \mathcal{K})$.

Example 13. (examples 10 and 12 continued) Figure 8 shows $Rep^{bch}(\mathcal{D}, \mathcal{K})$, i.e. the minimal MD repairs according to [11] for the `Location` dimension in Figure 1. $\mathcal{D}_3$ corresponds to the one obtained in Example 12 via relational translation, which produces only this single MD repair. So, in this example, $Rep(\mathcal{D}, \mathcal{K}) = \{\mathcal{D}_3\} \subsetneqq Rep^{bch}(\mathcal{D}, \mathcal{K})$.

It is easy to check that $\mathcal{D}_2$ can be obtained with the inversion rules after applying the update ρ_2 in Example 10. We saw that ρ_2 is not a minimal update, then $\mathcal{D}_2 \notin Rep(\mathcal{D}, \mathcal{K})$. Similarly for $\mathcal{D}_1$, that corresponds to the update ρ' also discussed in Example 10 (it does not produce a minimal relational repair). Thus, $\mathcal{D}_1 \notin Rep(\mathcal{D}, \mathcal{K})$ either. $\square$

The previous example might suggest that always $Rep(\mathcal{D}, \mathcal{K}) \subseteq Rep^{bch}(\mathcal{D}, \mathcal{K})$. This is not true: there are examples of MD schemas and instances where repairs in $Rep(\mathcal{D}, \mathcal{K})$ are not elements of $Rep^{bch}(\mathcal{D}, \mathcal{K})$ (cf. Example 16 in [31]). Despite the incomparability of these repair classes under set inclusion, it is still worth comparing $Rep(\mathcal{D}, \mathcal{K})$ and $Rep^{bch}(\mathcal{D}, \mathcal{K})$ in more detail in the case of our ongoing example. This will allow us to gain additional insight that will lead us to a purely MD characterization of our MD repairs.

Example 14. (example 13 continued) Let us focus on the edges inserted or deleted by each of the repairs in Figure 8. They all agree on the insertion of a link between `41` and `VIII`, to enforce homogeneity. However, they differ on strictness enforcement. Notice that the edges deleted or inserted by $\mathcal{D}_1$ and $\mathcal{D}_2$ belong to the first level of the dimension hierarchy, while repair $\mathcal{D}_3$ makes changes on its second level. $\mathcal{D}_1$ changes the link between element $\mathtt{N}_3$ and category `City`, updating the p-instance $\langle \mathtt{N}_3$,`CCP`,`VIII`,`all`$\rangle$ into $\langle \mathtt{N}_3$,`TEM`,`IX`,`all`$\rangle$. With $\mathcal{D}_3$ that p-instance is changed into $\langle \mathtt{N}_3$,`CCP`,`IX`,`all`$\rangle$. So, $\mathcal{D}_1$ causes more changes on this p-instance in comparison to $\mathcal{D}_3$. A similar comparison applies to $\mathcal{D}_2$ and $\mathcal{D}_3$. $\square$

[9]For repairs based on changes on the MD schema cf. [21, 22], and more formally [5].

This example shows that modifying different edges may be reflected in different ways on the underlying relational database. Hence, a notion of MD repair minimality (we already have a notion of minimality on the relational side, in Definition 5) should not be solely based on the number of modified edges, but also on which edges are being modified and at which level. We define an MD measure that considers this.

Definition 7. Let $\mathcal{D}$ and $\mathcal{D}'$ be dimension instances over the same MD schema $\mathcal{S}$ and active domain $\mathcal{M}$ and category association function δ. (a) The sets of *insertions*, *deletions* and *modifications* as a result of updating $\mathcal{D}$ into $\mathcal{D}'$ are:

$$\begin{aligned} ins(\mathcal{D}, \mathcal{D}') &= \{ (e_1, e_2) \in (<_{\mathcal{D}'} \smallsetminus <_{\mathcal{D}}) \mid \text{there is no } e_3 \\ &\qquad \text{with } (e_1, e_3) \in (<_{\mathcal{D}} \smallsetminus <_{\mathcal{D}'})\}. \\ del(\mathcal{D}, \mathcal{D}') &= \{ (e_1, e_2) \in (<_{\mathcal{D}} \smallsetminus <_{\mathcal{D}'}) \mid \text{there is no } e_3 \\ &\qquad \text{with } (e_1, e_3) \in (<_{\mathcal{D}'} \smallsetminus <_{\mathcal{D}})\}. \\ mod(\mathcal{D}, \mathcal{D}') &= \{ (e_1, e_2, e_3) \mid (e_1, e_2) \in (<_{\mathcal{D}'} \smallsetminus <_{\mathcal{D}}) \text{ and} \\ &\qquad (e_1, e_3) \in (<_{\mathcal{D}} \smallsetminus <_{\mathcal{D}'})\}. \end{aligned}$$

(b) The *cost of updating* $\mathcal{D}$ into $\mathcal{D}'$, denoted $ucost(\mathcal{D}, \mathcal{D}')$ is:

$$\begin{aligned} ucost(\mathcal{D}, \mathcal{D}') &= \sum_{(e_1,e_2) \,\in\, (ins(\mathcal{D},\mathcal{D}') \cup del(\mathcal{D},\mathcal{D}'))} |\alpha(e_1, e_2)| \times |\beta(e_2)| \\ &\quad + \sum_{(e_1,e_2,e_3) \,\in\, mod(\mathcal{D},\mathcal{D}')} |\alpha(e_1, e_2)| \times |\gamma(e_2, e_3)|, \text{ with:} \end{aligned}$$

$\alpha(e_1, e_2) = \{p \mid p \in Inst^{\mathcal{D}}(P),\ \{\delta(e_1), \delta(e_2)\} \subseteq P,\ e_1 \in p\}$, $\beta(e) = \{\ e' \in \mathcal{M} \mid e <^{\star}_{\mathcal{D}} e'\}$, and $\gamma(e_2, e_3) = \{e' \in \mathcal{M} \mid e_2 <^{\star}_{\mathcal{D}} e',$ but not $e_3 <^{\star}_{\mathcal{D}} e'\}$. $\square$

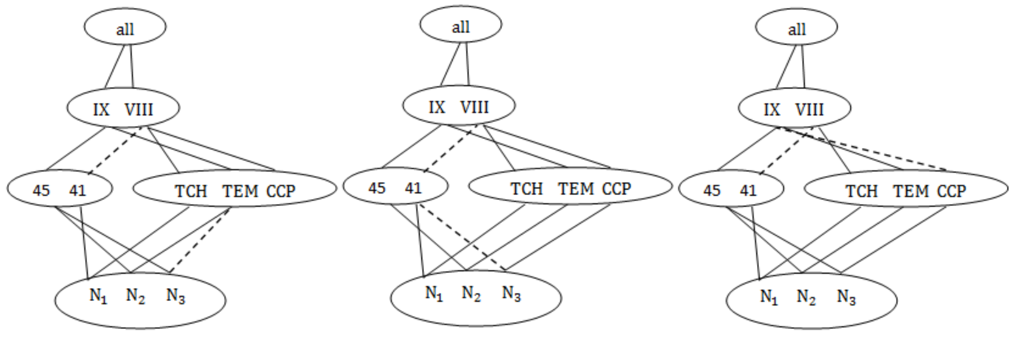
Intuitively, $ucost(\mathcal{D}, \mathcal{D}')$ captures the number of changes made to the elements of p-instances belonging to $\mathcal{D}$. From the corresponding relational point of view, this is the number of changes of attribute-values, as an indirect result of the MD updates. Each number of updated attribute values as a result of each edge change is computed separately. The sum of those values for all changed edges (e_1, e_2) gives the total number of attribute value updates needed for updating $\mathcal{D}$ into $\mathcal{D}'$.

Example 15. (example 14 continued) We want $ucost(\mathcal{D}, \mathcal{D}_i)$ for the MD repairs $\mathcal{D}_i$, $i = 1, 2, 3$.
$ins(\mathcal{D}, \mathcal{D}_1) = \{(\mathtt{41}, \mathtt{VIII})\}$, $del(\mathcal{D}, \mathcal{D}_1) = \emptyset$, $mod(\mathcal{D}, \mathcal{D}_1) = \{(\mathtt{N}_3, \mathtt{TEM}, \mathtt{CCP})\}$,
$ins(\mathcal{D}, \mathcal{D}_2) = \{(\mathtt{41}, \mathtt{VIII})\}$, $del(\mathcal{D}, \mathcal{D}_2) = \emptyset$, $mod(\mathcal{D}, \mathcal{D}_2) = \{(\mathtt{N}_3, \mathtt{41}, \mathtt{45})\}$,
$ins(\mathcal{D}, \mathcal{D}_3) = \{(\mathtt{41}, \mathtt{VIII})\}$, $del(\mathcal{D}, \mathcal{D}_3) = \emptyset$, $mod(\mathcal{D}, \mathcal{D}_3) = \{(\mathtt{CCP}, \mathtt{IX}, \mathtt{VIII})\}$.

The sets α, β and γ are associated to instance $\mathcal{D}$. They are: $\alpha(\mathtt{41}, \mathtt{VIII}) = \{\langle \mathtt{N}_1,\ \mathtt{41},\ \mathtt{NULL},\ \mathtt{NULL}\ \rangle\}$, $\alpha(\mathtt{N}_3, \mathtt{TEM}) = \{\langle \mathtt{N}_3,\ \mathtt{CCP}, \mathtt{VIII}, \mathtt{all}\ \rangle\}$, $\alpha(\mathtt{N}_3, \mathtt{41}) = \{\langle \mathtt{N}_3, \mathtt{45}, \mathtt{IX}, \mathtt{all}\ \rangle\}$, $\alpha(\mathtt{CCP}, \mathtt{IX}) = \{\langle \mathtt{N}_3, \mathtt{CCP}, \mathtt{VIII}, \mathtt{all}\ \rangle\}$; $\beta(\mathtt{VIII}) = \{\mathtt{VIII}, \mathtt{All}\}$; $\gamma(\mathtt{TEM}, \mathtt{CCP}) = \{\mathtt{TEM},\ \mathtt{IX}\}$, $\gamma(\mathtt{41}, \mathtt{45}) = \{\mathtt{41},\ \mathtt{VIII}\}$, $\gamma(\mathtt{IX}, \mathtt{VIII}) = \{\mathtt{IX}\}$.

With these elements we can compute each update cost:

$$\begin{aligned} ucost(\mathcal{D}, \mathcal{D}_1) &= |\alpha(\mathtt{41}, \mathtt{VIII})| \times |\beta(\mathtt{VIII})| + |\alpha(\mathtt{N}_3, \mathtt{TEM})| \\ &\quad \times |\gamma(\mathtt{TEM}, \mathtt{CCP})| = 1 \times 2 + 1 \times 2 = 4, \\ ucost(\mathcal{D}, \mathcal{D}_2) &= |\alpha(\mathtt{41}, \mathtt{VIII})| \times |\beta(\mathtt{VIII})| + |\alpha(\mathtt{N}_3, \mathtt{41})| \\ &\quad \times |\gamma(\mathtt{41}, \mathtt{45})| = 1 \times 2 + 1 \times 2 = 4, \\ ucost(\mathcal{D}, \mathcal{D}_3) &= |\alpha(\mathtt{41}, \mathtt{VIII})| \times |\beta(\mathtt{VIII})| + |\alpha(\mathtt{CCP}, \mathtt{IX})| \\ &\quad \times |\gamma(\mathtt{IX}, \mathtt{VIII})| = 1 \times 2 + 1 \times 1 = 3. \end{aligned}$$



(a) $\mathcal{D}_1$ (b) $\mathcal{D}_2$ (c) $\mathcal{D}_3$

Figure 8: Minimal repairs in $Rep^{bch}(\mathcal{D}, \mathcal{K})$ for the `Location` dimension

$\mathcal{D}_3$ provides the least update cost for $\mathcal{D}$, i.e. the minimum number of changes to the underlying relational database, which is consistent with the discussion in Example 12: $\mathcal{D}_3$ is the only MD repair for $\mathcal{D}$ that also corresponds to a minimal relational repair. Notice that the update cost for each of $\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3$ is the same as the number of changes made to the p-instances of $\mathcal{D}$, and also the same as the number of attribute-value updates (the $|\rho|$s) performed via the corresponding relational repairs. $\square$

Lemma 1. Let $\mathcal{D}$ be an instance for the MD schema $\mathcal{S}$, and $\mathcal{K}$ be a set of LSCs over $\mathcal{S}$ (cf. Definition 6). Let D and Σ be the corresponding elements on the path relational side, and ρ an update of D. For the MD instance $\mathcal{D}'$ for $\mathcal{S}$ with $\mathcal{D}' = \mathcal{T}^{-1}(\rho(D))$, it holds: $ucost(\mathcal{D}, \mathcal{D}') = |\rho|$. $\square$

From this lemma (cf. [31] for a proof), we obtain a characterization of our MD repairs in pure MD terms.

Theorem 2. Let $\mathcal{D}$ be an instance for the MD schema $\mathcal{S}$, and $\mathcal{K}$ be a set of LSCs over $\mathcal{S}$. For every instance $\mathcal{D}'$ for $\mathcal{S}$, it holds: $\mathcal{D}' \in Rep(\mathcal{D}, \mathcal{K})$ iff $\mathcal{D}' \models \mathcal{K}$ and $ucost(\mathcal{D}, \mathcal{D}')$ is minimum (among the consistent $\mathcal{S}$-instances). $\square$

This characterization of MD repairs implicitly takes into consideration the effect of MD repair operations (edge insertions and deletions) on the underlying relational instance. As we seen above, not all the repairs proposed by [11] have to be MD repairs in our sense, nor the other way around. This is due to the fact that, in [11] edges are modified without considering (neither explicitly nor implicitly) the underlying relational side effects of the MD operations.

The repair approach in [11] applies minimality without considering any sort of priority on edge changes. In contrast, ours implicity does so, via the underlying relational side-effects of MD edge changes. In particular, for our MD repair process, edges with fewer connections to the base elements are good candidates for change: they affect fewer tuples in the underlying database. Among those edges, those that reside at higher levels of the hierarchy are optimal choices for change, because they update fewer attribute values in the affected tuples. As a result, in our approach, those MD repairs causing minimal side-effects are preferred.

9. EXPERIMENTS AND COMPUTATION

We had two main motivations behind the introduction of the path schema for MDDBs, and ROLAP for DWHs, in particular. The first one is the proposal of a new relational representation (schema) that is more suitable than the most popular ones for capturing, representing, checking, and enforcing summarizability constraints on MDDBs. As discussed above, the path schema is particularly appropriate for these tasks. In Section 9.2 below, we report on quite satisfactory experiments on inconsistency detection. However, it would be difficult to "sell" this new form of relational representation if a MDDB or a DWH built according to it had a poor performance wrt aggregate query answering. Being this a crucial issue, we decided to concentrate our first experiments on this particular task. The results are quite encouraging and are reported in Section 9.1 below.

We also think that the path relational representation of MDDBs is quite natural, and particularly appropriate for representing the hierarchically dimensional structure of data, particularly those subject to *non-linear*, i.e. lattice-like, dimension hierarchies. Unlike the star schema, the path schema allows to clearly and unambiguously read off the multi-dimensional instance from the path database instance (cf. Section 3). Compared with the snowflake representation, the path representation is equally good at capturing dimension hierarchies, without incurring, as snowflake, in the extra cost for query answering and inconsistency detection.

The second motivation is the use of an appropriate relational representation, the path relational schema, to support actual MD repairs, directly on the relational representation, or indirectly on the MD representation. We have introduced in detailed and precise terms the *repair semantics* and the operations that are needed to move and transfer results between the direct MD representation and the relational representation. Experimenting with repairs has been left for a second phase of experiments that corresponds to ongoing work. Since experiments of this kind require *algorithms for repair computation*, in Section 9.3 below we briefly describe a couple of approaches we are currently developing.

9.1 Query answering under the path schema

Our repairing of MD databases is done via relational transformations. However, if the original MD database is not implemented as a path relational (PR) database, and we wanted to use the latter for MD repairing, the translation would introduce an additional cost. In consequence, it is natural to ask about the possibility of using upfront *path relational schemas* as the basis for the implementation of MD databases or DWHs.

We claim that *the path relational schema is a promising alternative to consider for a relational approach to MDDBs and DWHs*, independently from its virtues wrt MD repairing. We ran some experiments supporting this claim, comparing a PR implementation with relational implementations based on the star and snowflake schemas. We addressed performance at aggregate query answering, on SQL Server 2008. We illustrate them using our running example (cf. Example 1).

In addition to the `Location` dimension (cf. Figure 1(a)), we consider a `Time` dimension with a simple linear hierarchy: `Date` $\nearrow$ `Month`, `Month` $\nearrow$ `Year`, and `Year` $\nearrow$ `All`. We count `Incoming` and `Outgoing` calls (the facts) (cf. Figure 2).

The relational representations of the `Location` dimension in the star, snowflake and path schemas that we implemented are in Figures 4, 5 and 3, resp. The relational representations of the `Time` dimension are simple, and done as described in Sections 3 and 4.

We generated and loaded a sizeable amount of test data into star, snowflake, and path databases. The generator creates elements for each category in the dimension schema, taking into account the hierarchy levels. The lower the category level, the larger the set of elements, with the base category containing 100,000 elements. For each of the three relational DBs we tested several queries. Due to the space limitations, we discuss in detail only one of them, namely a query, $\mathcal{Q}$, asking for the `Incoming` calls made from each region (i.e. category `Region`) in year 2010. $\mathcal{Q}$ takes 3 different forms in SQL depending on the underlying relational schema (cf. [31, sec. 9] for many more experimental details). The final results of the experiment, including all queries, are shown in Figure 9.

For the star schema, we need to join `Traffic-Fact-Table` (the fact table in Figure 2) with the tables for the `Location` dimension ($\mathtt{R}^{\mathtt{Loc}}$) in Figure 4, and the `Time` dimension ($\mathtt{R}^{\mathtt{Time}}$).

For the path schema, the structure of the SQL query depends on the categories used in the aggregate query (`Region` in this case). A category may belong to several B2A paths, and we may have to use several path tables. Since `Region` belongs to two B2A paths, in order to compute the roll-up $\mathcal{R}^{\mathtt{Region}}_{\mathtt{Number}}$ from `Number` to `Region`, we need first the union of sets of tuples in either path tables, with selected attributes $A^{\mathtt{Number}}$ and $A^{\mathtt{Region}}$. This intermediate result is joined with the fact table and the relational `Time` table $RP^{\mathtt{Time}}$. As expected, the query posed to the path DB is more complex than the one posed to the star DB. For this reason, the "star query" executes faster than the "path query".

As is well-known (cf. also Section 3), the hierarchical structure of the snowflake schema requires several joins, which are costly operations. As expected, the number of joins in the query posed to the snowflake DB is considerably higher than in the first two cases, and accordingly, the execution time is also much higher. The execution times for the 3 SQL queries are: 155.2 ms for the star DB, 193.8 ms for the path DB, and 427.3 ms for the snowflake DB.

We considered other queries, involving other categories of the `Location` dimension in the aggregation. They are about the sum of `Incoming` calls made in year `2010` from each cell phone number, area code, city, and finally, from all of the numbers. The results are shown in Figure 9. Query $\mathcal{Q}$ corresponds to category `Region` in Figure 9.

Notice that when the category we are rolling-up to in the query is `City` or `AreaCode`, instead of `Region`, the structure of the SQL query under the path schema is similar to the one for the same query under the star schema. In this case, only one B2A path has to be considered, and there is no need to merge the tuples of several path tables. In cases like this, we have similar query answering times under the star and path schema, as shown in Figure 9.

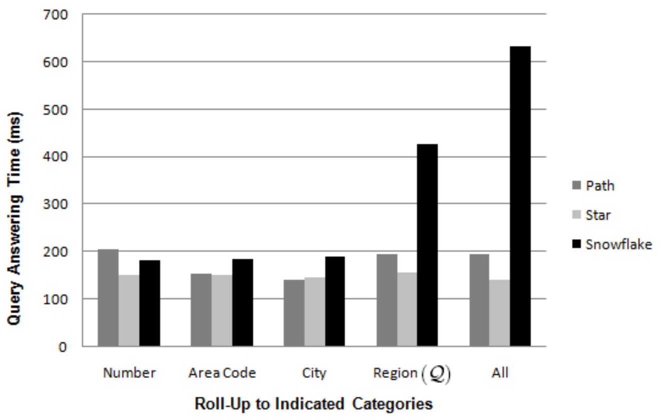


Figure 9: Comparing path, star and snowflake schemas.

We observe that, as the level of the category in the aggregate query increases, the query answering time for the snowflake schema increases significantly, while the performance for the star and path schemas is not considerably affected by this factor. This is due to the hierarchical structure of snowflake, which requires a larger number of joins for rolling-up to higher levels of the hierarchy. For the star and path schemas, the number of joins is independent from the level of the category used in the aggregate query. Our results show that, as expected, the star schema provides the best query answering performance, with the path schema being the second best, by a small difference.

9.2 Inconsistency detection

In Section 3, we discussed that one of the important criteria for a relational representation of a MD database is its efficiency in checking the MD summarizability conditions, as expressed through relational ICs. We made a case for the path schema in this regard. We ran experiments on the time it takes to check strictness and homogeneity of the `Location` dimension in each of the star, snowflake and path instances. Sections 3 and 4 describe how to represent summarizability conditions with ICs over the star, snowflake and path databases, respectively. We generated 70,000 p-instances for the `Location` MD dimension instance, including around 600,000 cases of non-strictness; and 20,000 cases of heterogeneity, expressed by the occurrence of `NULL` in one of the p-instance elements.

We generated a considerable proportion of cases of non-strictness, to compare the difference between FD and EGD evaluation under the star or path schemas, and observe the impact of the several joins for snowflake. The times for non-strictness detection for star, snowflake and path databases are 17.007 sec, 1200 sec, and 15.686 sec, respectively, which are in line with our discussions in Sections 3 and 5.

The considerably higher execution time for snowflake is due to the its explicit hierarchical structure, which complicates strictness checking due to a series of joins. On the other side, simple constraints are required for the star and path databases; and the negligible difference between their execution times is due to different dimension table schemas and the number of tuples belonging to each of those tables. The additional EGDs used with the path schema for detecting non-strictness in more than one path tables do not considerably effect the performance.

For homogeneity, the kinds of ICs used for all 3 schemas are the same, i.e. NOT NULL constraints. Hence, our main concern in checking homogeneity was to verify the weakness of star schema in completely capturing the heterogeneous instances (cf. Section 3). In a first phase, detecting heterogeneity took 510 ms for the star schema, 866 ms for the path schema, and 686 ms for the snowflake schema. The difference between the first and the last two numbers is due to the fact that only half of the heterogeneity cases were captured under the star schema. On the other hand, since in the path schema a category may be represented by more than one table (like Region and All in the Location dimension), we need more NOT NULL constraints for the path schema when compared with the snowflake schema. For this reason, under snowflake the heterogeneity instances are retrieved faster than with the path schema. However, the impact of this factor is not particularly high.

Since the other path and snowflake do support the detection of all kinds of heterogeneity, they were competing in handicapped terms with the star schema. For this reason, we compared the performances with the same number of cases of heterogeneity present in each of the three DBs, and all of them *detectable* in the star DB. We generated 20,000 such instances of heterogeneity; and the times obtained were 750 ms for the star schema, 962 ms for the path schema, and 758 ms for the snowflake schema. As expected, the performance of the star schema deteriorated.

9.3 Repair computation

Earlier in this section, we have reported on some experiments with the proposed relational schema in the direction of aggregate query answering and inconsistency detection. Experimentation with MD repairs is ongoing work. This requires developing the corresponding algorithms, for repair computation. In this direction, we are exploring two alternative methodologies, whose comparison will be interesting to make.

The first one uses "answer set programs" (ASPs), similar to the repairs programs in [11, 16], that work directly with the MD representation. In our case, given the different MD repair semantics (as in Theorem 2), our programs include "weak constraints with weights", that extend the ASP paradigm [12]. They are used to minimize the *number* of violations of program constraints. For this reason, they can be used to capture our *numerical* MD distance. (The distance in [11, 16] is not numerical, but set-theoretical).

The other way to go consists in repairing directly the relational instances obtained via the MD2R mapping wrt relational ICs. ASPs have been also successfully used to compute relational repairs and do consistent query answering (cf. [15] and references therein). In this case, the ASPs also require the use of weak constraints, because the number of changes of attribute values is minimized. ASPs of this kind have been used in [19].

10. CONCLUSIONS

In this paper we addressed two important problems in relation to multidimensional (MD) databases. Each of them has several aspects:

1. We proposed and analyzed a new relational representation for MDDBs, and data warehouses, in particular. This so-called *path relational schema* (PRS) was also compared with the traditional relational schemas for ROLAP, the star and the snowflake schemas. Our experiments suggest that the path relational schema has promising properties in terms of query answering time when compared with the star and snowflake schemas.

The PRS, enriched with natural relational ICs, is suitable for handling inconsistency in MDDBs wrt MD summarizability constraints. It allows for: (a) a natural relational representation of those semantic requirements, (b) the possibility of *completely* and *efficiently* checking their satisfaction. Our experiments support these claims.

2. We proposed a new characterization of repairs of MD databases that fail to satisfy summarizability constraints. We did this by first translating the problem into a similar one formulated in purely relational terms. The proposed relational schema and translation have nice invertibility properties.

Going the relational route has two main advantages: (a) We can take advantage of much existing work on relational repairs and consistent query answering. (b) The repair process could be implemented directly on relational platforms.

As an alternative to the relational characterization of MD repairs, we also offered a new definition of minimal MD repairs in purely MD terms. As opposed to previous approaches, the new notion is sensitive to the levels in an MD hierarchy where changes are made to restore consistency.

In this paper we haven't considered consistent query answering for aggregate queries. It would be natural to explore the *range semantics* [3, 20]. We should also investigate the existence of a corresponding notion of *canonical instance* introduced in [9], that was used to do and approximate CQA. Still around CQA, it would be interesting to fully exploit our relational translation and the existing (or new) results and techniques for relational repairs and CQA [8].

We think that the *path relational* route to MD repairs can be particularly useful for doing MD database repairing and CQA under updates [26]. In the MD scenario, considering, in addition to summarizability constraints, also *aggregation constraints* would be quite natural [18].

The relational framework offered by the path schema together with NOT NULL and EGDs deserves investigation *per se*. In this vein, there are also interesting issues to explore in relation to *data exchange* and *schema mappings* [4, 6] (from one relational MD schema into another, like the path schema), and also *schema evolution* [17].

We have briefly compared our MD repair semantics with other MD data-oriented repairs. A deeper investigation is still necessary. It is also interesting to compare our instance-based MD repairs with the structural MD repairs that are based on changes on the schema, or category lattice, and the distribution of data elements [5]. They leave data and their links unchanged, something that might be welcome and relevant in some applications. On the other hand, schema-based repairs may have a drawback that, since categories are changed, queries have to be rewritten, and cached aggregations partially recomputed. Virtual approaches to these problems can be attempted, using MDX views [5]. For heterogeneity, structural repairs should be favored in the absence of predefined queries or when the rewriting is affordable.

Data-oriented repairs are well suited for addressing heterogeneity, and also non-strictness, specially when inconsistencies occur at higher levels of the schema. In this latter case, solving non-strictness via schema-based repairs tends to enforce multiple schema changes.

An issue with our proposed approach is that it could require a *new relational layer* for an already existing MDDB that has been implemented directly as such or via a star or snowflake relational database. Instead of building a materialized relational layer on top, we should explore defining the proposed path schema as a *virtual view* of the existing representation/implementation (as in [5]). This view could be seen as a *logical layer* [13] on top.

11. REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison-Wesley, 1995.

[2] M. Arenas, L. Bertossi, and J. Chomicki. Consistent Query Answers in Inconsistent Databases. Proc. PODS 1999, pp. 68–79.

[3] M. Arenas, L. Bertossi, J. Chomicki, X. He, V. Raghavan, and J. Spinrad. Scalar Aggregation in Inconsistent Databases. *Theoretical Computer Science*, 2003, 296(3):405-434.

[4] M. Arenas, J. Perez, J. Reutter, and C. Riveros. Composition and Inversion of Schema Mappings. *Sigmod Record*, 2010, 38:17-28.

[5] S. Ariyan and L. Bertossi. Structural Repairs of Multidimensional Databases. Proc. Alberto Mendelzon International WS of Foundations of Data Management, 2011, CEUR Workshop Proc., Vol. 749.

[6] Ph. A. Bernstein and S. Melnik. Model Management 2.0: Manipulating Richer Mappings. Proc. SIGMOD, 2007, pp. 1-12.

[7] L. Bertossi. Consistent Query Answering in Databases. *Sigmod Record*, 2006, 35:68-76.

[8] L. Bertossi. *Database Repairing and Consistent Query Answering*, Morgan & Claypool, Synthesis Lectures on Data Management, 2011.

[9] L. Bertossi, L. Bravo, and M. Caniupan. Consistent Query Answering in Data Warehouses. Proc. 3rd Alberto Mendelzon International Workshop on Foundations of Data Management, 2009, CEUR Workshop Proc., Vol. 450.

[10] L. Bravo and L. Bertossi. Semantically Correct Query Answers in the Presence of Null Values. Proc. EDBT 2006 Workshops, 2006, Springer LNCS 4254, pp. 336-357.

[11] L. Bravo, M. Caniupan, and C. A. Hurtado. Logic Programs for Repairing Inconsistent Dimensions in Data Warehouses. *Proc. 4th Alberto Mendelzon International Workshop on Foundations of Data Management*, 2010, *CEUR Workshop Proc.*, Vol. 619.

[12] F. Buccafurri, N. Leone, P. Rullo. Enhancing Disjunctive Datalog by Constraints. *IEEE Trans. Knowl. Data Eng.*, 12(5): 845–860, 2000.

[13] L. Cabibbo and R. Torlone. The Design and Development of a Logical System for OLAP. Proc. DaWaK, 2000, Springer LNCS 1874, pp. 1-10.

[14] M. Caniupan. Handling Inconsistencies in Data Warehouses. In *Current Trends in Database Technology*, 2004, Springer LNCS 3268, pp. 166-176.

[15] M. Caniupan, L. Bertossi. The Consistency Extractor System: Answer Set Programs for Consistent Query Answering in Databases. *Data and Knowledge Engineering*, 69(6): 545–572, 2010.

[16] M. Caniupan, L. Bravo, and C. A. Hurtado. Logic Programs for Repairing Inconsistent Dimensions in Data Warehouses. Journal submission, 2010.

[17] C. Curino, H-J. Moon, A. Deutsch, and C. Zaniolo. Update Rewriting and Integrity Constraint Maintenance in a Schema Evolution Support System: PRISM++. *PVLDB*, 2010, 4(2):117-128.

[18] S. Flesca, F. Furfaro, and F. Parisi. Querying and Repairing Inconsistent Numerical Databases. *ACM Trans. Database Systems*, 2010, 35(2).

[19] E. Franconi, A. Laureti-Palma, N. Leone, S. Perri, and F. Scarcello. Census Data Repair: A Challenging Application of Disjunctive Logic Programming. Proc. LPAR, 2001, Springer LNCS, pp. 561-578.

[20] A. Fuxman, E. Fazli, and R. Miller. Conquer: Efficient Management of Inconsistent Databases. Proc. SIGMOD, 2005, pp. 155-166.

[21] C. A. Hurtado and C. Gutierrez. Handling Structural Heterogeneity in OLAP. In *Data Warehouses and OLAP: Concepts, Architectures and Solutions.* Idea Group, Inc, 2007.

[22] C. A. Hurtado, C. Gutierrez, and A. Mendelzon. Capturing Summarizability with Integrity Constraints in OLAP. *ACM Transactions on Database Systems*, 2005, 30:854-886.

[23] C. A. Hurtado and A. Mendelzon. Reasoning about Summarizability in Heterogeneous Multidimensional Schemas. Proc. ICDT, 2001, Springer LNCS 1973, pp. 375-389.

[24] M. Jarke, M. Lenzerini, Y. Vassiliou, and P. Vassiliadis. *Fundamentals of Data Warehouses.* Springer, 1997.

[25] M. Levene and G. Loizou. Why is the Snowflake Schema a Good Data Warehouse Design? *Information Systems*, 2003, 28:225-240.

[26] A. Lopatenko and L. Bertossi. Complexity of Consistent Query Answering in Databases under Cardinality-Based and Incremental Repair Semantics. Proc. ICDT, 2007, Springer LNCS 4353, pp. 179-193.

[27] T. B. Pedersen and C. S. Jensen. Multidimensional Data Modeling for Complex Data. Proc. ICDE, 1999, pp. 336-345.

[28] T. B. Pedersen, C. S. Jensen, and C. E. Dyreson. Extending Practical Pre-Aggregation in On-Line Analytical Processing. Proc. VLDB, 1999, pp. 663-674.

[29] M. Rafanelli and A. Shoshani. STORM: A Statistical Object Representation Model. Proc. SSDM, 1990, pp. 14-29.

[30] P. Vassiliadis and T. K. Sellis. A Survey of Logical Models for OLAP Databases. *SIGMOD Record*, 1999, 28(4):64-69.

[31] M. Yaghmaie, L. Bertossi, and S. Ariyan. Repair-Oriented Relational Schemas for Multidimensional Databases (extended version). `http://people.scs.carleton.ca/~bertossi/papers/subm2011Extended.pdf`

[32] M. Yoshikawa, T. Amagasa, T. Shimura, and S. Uemura. XRel: A Path-based Approach to Storage and Retrieval of XML Documents using Relational Databases. *ACM Trans. Internet Techn.*, 2001, 1(1):110-141.