# Repair-Oriented Relational Schemas for Multidimensional Databases*

(second extended version)

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) [29] represent data at a higher level of abstraction than relational databases, using multiple *dimensions* to give and make sense to/of usually quantitative data, the so-called *facts* in data warehouses (DWHs) [35, 33].

EXAMPLE 1. [9] The following represents data for a phone company about the cell phones usage as depending on time and location. The `Location` dimension represents the hierarchy of the wireless network spots. Each cell phone number has a specific area code (`41` or `45`), and belongs to a city, `TCH` (Talcahuano), `TEM` (Temuco), or `CCP` (Concepcion). Area codes and cities themselves belong to a region, `VIII` or `IX`. The `Location` dimension schema is in Figure 1a, and an instance of this schema is shown in Figure 1b. Figure 1c shows data about the network usage by phone numbers.

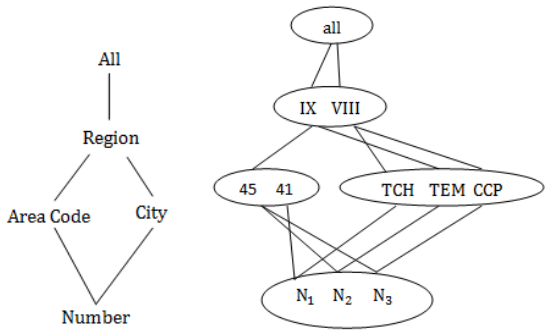
In Figure 1a, `Number`, `Area Code`, etc., are *categories*. For example, `Region` is a *parent* category for categories `Area Code` and `City`; and an *ancestor* of category `Number`. Similarly, element $\mathtt{N}_2$ of category `Number` has `IX` as an ancestor element in category `Region`. □

Due to large volumes of data in MDDBs, computation from scratch of aggregate queries should be avoided whenever possible. Ideally, aggregate query results at lower levels should be used to calculate results at higher levels of the hierarchy. A dimension instance that allows this is called *summarizable*.

The notion of summarizability was introduced in [43], in the context of statistical databases. A multidimensional database is summarizable, when all of its dimensions allow for summarization. A non-summarizable MDDB will either return incorrect query results when using pre-computed views, or lose efficiency by having to compute the answers from scratch [43, 37, 36, 41, 29].

For a dimension to be summarizable, two conditions must be satisfied. First, it must be *strict*, meaning that each element in a category has at most one parent in each upper category [29, 41, 43]. Secondly, it has to be *homogeneous*, meaning that each element in a category has at least one parent element in each parent category [29, 35, 43]. For this reason, we usually and informally refer to the combination of the homogeneity and strictness conditions as the *summarizability conditions*.

In Example 1, strictness is violated, because $\mathtt{N}_3$ has two grandparents in the `Region` category, namely `IX` and `VIII`. Moreover, the `Location` dimension in this example is non-homogeneous (or heterogenous), because element `41` has no




(a) `Location` schema (b) `Location` instance

| 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 |

(c) Fact table

Figure 1: Cell phone traffic database

parent in category `Region`.

There are design reasons that commonly make a dimension instance non-summarizable [29]. Furthermore, a dimension instance may become non-summa-rizable after dimension updates [32]. In particular, non-strictness and heterogeneity are common in MDDBs. An MDDB that has any of these two properties is said to be *inconsistent*. Confronted with inconsistency, one can try to restore the properties of strictness and homogeneity, through what is usually called a *database repair* process.

Repairs of relational databases that violate integrity constraints (ICs) have been investigated in the literature [2]. 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 (see [7, 20] for recent surveys).

Several approaches for repairing MDDBs have been proposed recently. Non-summarizability is resolved by changing either the dimension instance or the dimension schema. Instance-based repairs have been introduced and studied in [9, 13, 17, 19], cf. also [42]. Schema-based repairs have been proposed in [27, 30, 28]. MD schema-based repairs have been formally defined and investigated in [5].

The MD repairs proposed so far do not assume any relational representation of MDDBs, and work directly with/on the multidimensional model. Nor they appeal to any kind specific implementation of MDDBs. However, MDDBs can be implemented either as relational systems (ROLAP), proprietary multidimensional systems (MOLAP), or a hybrid of both (HOLAP) [34, 35, 45]. Based on [35], MDDBs are usually implemented as/on relational databases, which facilitates optimized query answering and data storage [45].

In this work we address non-summarizability, as caused by heterogeneity or non-strictness, through a relational representation of MDDBs. In particular, we investigate the applicability of notions and mechanisms related to *relational repairs* when dealing with inconsistent MDDBs. Our goal is to restore summarizability by repairing the underlying relational database. In this way, we can take advantage of an already existing rich body of research.

To achieve our goal, we have to start by representing our original 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 schema that allows for the 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 database is inconsistent wrt the created ICs.

Next, the resulting inconsistent relational instance is repaired as a relational database, using existing techniques. As a result, we obtain a set of *minimal relational repairs*. The final step consists of translating these repairs into MDDB repairs. As expected, the feasibility of this approach depends on the *invertibility* of MD2R mappings [22, 4].

For all this program to work, we need to properly confront our first challenge, the proposal of an *expressive* relational representation for an MDDB. The ideal relational representation should enable the efficient check of the summarizability conditions through the associated set of integrity constraints. Moreover, there should be *no information loss* under this mapping and its inversion, otherwise we might have an incomplete or incorrect retrieval of repaired dimension instances from the repaired relational instances.

In this direction, we first investigate the two well-known relational representations of MDDBs, the *Star* and *Snowflake*, showing that they are not appropriate for our purpose. Next, we define a new alternative relational representation of MDDBs. It uses a *path-based* approach, and the associated relational schema is called *the path schema*.

Example 2. (example 1 continued) Figure 2 shows how the `Location` dimension is represented according to the path schema. Each relational table represents a path from the bottommost category to the topmost category in the dimension schema. The hierarchy in Figure 1a contains two examples of the aforementioned paths. Hence, we have two tables representing them. Each path goes through several data elements in the dimension instance. The sequence of elements on each path creates a tuple for the corresponding table. For an MD category $c$, $A^c$ denotes the corresponding relational attribute. □

Our results show the adequacy of our approach to MD inconsistency handling via repairs of the associated path relational instances. By using the relational path schema, we can efficiently check the strictness and homogeneity conditions through relational ICs. Furthermore, the MD2R mapping turns out to be uniquely invertible.

| | $A^{\text{Number}}$ | $A^{\text{AreaCode}}$ | $A^{\text{Region}}$ | $A^{\text{All}}$ |
|---|---|---|---|---|
| 1 | $N_1$ | 41 | NULL | NULL |
| 2 | $N_2$ | 45 | IX | all |
| 3 | $N_3$ | 45 | IX | all |

(a) Table $RP_1^{Loc}$ for left path $P_1^{Loc}$ in `Location` schema

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

(b) Table $RP_2^{Loc}$ for right path $P_2^{Loc}$ in `Location` schema

Figure 2: `Location` dimension instance represented according to path schema

Notice that our MD repairs are instance-based, as opposed to schema-oriented: The original MD schema is not changed into a new MD schema, but only the MD instance is changed via its transformation into a relational instance and subsequent repairs. However, we obtain a class of MD repairs that differs from the class of (also instance-oriented) MD repairs proposed in [13, 19] ([9, 17] deal only with non-strictness, assuming homogeneity). This discrepancy is due to the 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). The former require changes of attribute values, whereas the latter two cases, may require full tuple insertions or deletions. In Section 8 a discussion about the so-obtained MD repairs can be found.

The rest of this paper is structured as follows. Section 2 briefly describes the multidimensional data model we use in our work. Section 3 explains why the well-known star and snowflake relational schemas are not appropriate for dealing with consistency 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 integrity constraints over path schema. Section 6 provides the relational repair semantics for restoring consistency in path databases. 7 investigates the invertibility of the proposed MD to relational mapping. 8 presents a purely MD characterization of the repairs obtained through our 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 on what we have achieved and about relevant ongoing and future work.

## 2. PRELIMINARIES

Graph-theoretic representations of MDDBs have been proposed in the literature [15, 31]. In this work we adopt the Hurtado-Mendelzon formalization [29, 26].

A *dimension schema* $\mathcal{S}$ is a directed acyclic graph (DAG), 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 dimension schema. The transitive and reflexive closure of this binary relation is denoted with $\nearrow^*$. We make the usual assumption that there are no "shortcuts" in an MD schema, i.e. if $c_i \nearrow c_j$, then there is no (properly) intermediate category $c_k$ with $c_i \nearrow^* c_k$ and $c_k \nearrow c_j$.

Every dimension schema has a distinguished top category, `All`, which is reachable from every other category: For every category $c \in \mathcal{C}$, $c \nearrow^* \texttt{All}$ holds. In addition, in every dimension schema, there is *a unique category* that has no child. It is called the *base category*.

Complying to the dimension schema, a *dimension instance*, $\mathcal{D}$, is modeled as a pair $\langle \mathcal{M}, < \rangle$, where $\mathcal{M}$ is the finite set of data elements, and $<$ (sometimes denoted $<_{\mathcal{D}}$) is binary relation on $\mathcal{M}$, the *parent-child relationship*, that parallels relation $\nearrow$. More precisely, there is a mapping $\delta$ from $\mathcal{M}$ to $\mathcal{C}$ that assigns each data element to a unique category. If $\delta(m) = c$, then 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 denoted with $<^*$. Element $\texttt{all} \in \mathcal{M}$ is the only element of category `All`. In general, `all` is expected to be reached from any other category element, but there may be instances where this is not the case.

A roll-up relation, $\mathcal{R}_{c_i}^{c_j}(\mathcal{D})$, can be built to any pair of categories $c_i, c_j$ in the schema. It contains the pairs of the form $(m_i, m_j)$, with $c_i \nearrow^* c_j$, $m_i \in c_i, m_j \in c_j$, $m_i <^* m_j$. The roll-up relation is not necessarily a function. We do not assume this relation to be *total*, i.e. there may be $m_i \in c_i$ that does not rolls up to an element in $c_j$.

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

$$\begin{aligned} \mathcal{C} &= \{\texttt{Number}, \texttt{AreaCode}, \texttt{City}, \texttt{Region}, \texttt{All}\}. \\ \nearrow &= \{(\texttt{Number}, \texttt{AreaCode}), (\texttt{Number}, \texttt{City}), \\ &\quad (\texttt{AreaCode}, \texttt{Region}), (\texttt{City}, \texttt{Region}), \\ &\quad (\texttt{Region}, \texttt{All})\}. \end{aligned}$$

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

$$\begin{aligned} \mathcal{M} &= \{N_1, N_2, N_3, 41, 45, \texttt{TCH}, \texttt{TEM}, \texttt{CCP}, \texttt{VIII}, \texttt{IX}, \texttt{all}\}. \\ < &= \{(N_1, 41), (N_1, \texttt{TCH}), (N_2, 45), (N_2, \texttt{TEM}), (N_3, 45), \\ &\quad (N_3, \texttt{CCP}), (45, \texttt{IX}), (\texttt{TEM}, \texttt{IX}), (\texttt{TCH}, \texttt{VIII}), \\ &\quad (\texttt{CCP}, \texttt{VIII}), (\texttt{VIII}, \texttt{all}), (\texttt{IX}, \texttt{all})\}. \end{aligned}$$

The ancestors in the `Region` category of all base elements can be obtained via the roll-up relation

$$\mathcal{R}_{\texttt{Number}}^{\texttt{Region}}(\mathcal{D}) = \{(N_1, \texttt{VIII}), (N_2, \texttt{IX}), (N_3, \texttt{VIII}), (N_3, \texttt{IX})\}. \ \square$$

As mentioned above, the MD semantic conditions of strictness and homogeneity have received much attention. They, together, guarantee the summarizability property for a dimension instance. These are global conditions that can also be made local (or enforced locally).

Definition 1. [13, 19] (a) For a dimension schema $\mathcal{S} = \langle \mathcal{C}, \nearrow \rangle$, a *strictness constraint* is an expression of the form $c_i \to c_j$, where $c_i, c_j \in \mathcal{C}$, $c_i \neq c_j$, and $c_i \nearrow^* c_j$. This

constraint is satisfied by a dimension instance $\mathcal{D}$, denoted $\mathcal{D} \models c_i \to c_j$, iff the roll up relation $\mathcal{R}^{c_j}_{c_i}(\mathcal{D})$ is a (possibly partial) function.
(b) The dimension instance $\mathcal{D}$ is *strict* if it satisfies the *full-strictness condition*, namely the set, $FS^{\mathcal{S}} = \{c_i \to c_j \mid c_i, c_j \in \mathcal{C}, c_i \neq c_j, \text{ and } c_i \not\nearrow^* c_j\}$, of all strictness constraints. □

Notice that an instance $\mathcal{D}$ is strict when every roll-up relation is a function.

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

DEFINITION 2. [13, 19] (a) For a dimension schema $\mathcal{S} = \langle \mathcal{C}, \nearrow \rangle$, a *homogeneity constraint* (a.k.a. *covering 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 $c_i \not\nearrow c_j$. This constraint is satisfied by a dimension instance $\mathcal{D}$, denoted $\mathcal{D} \models c_i \Rightarrow c_j$, iff the roll-up relation $\mathcal{R}^{c_j}_{c_i}(\mathcal{D})$ is *total*.
(b) A dimension instance $\mathcal{D}$ is *homogenous* iff it satisfies the *full-homogeneity condition*, namely the set, $FH^{\mathcal{S}} = \{c_i \Rightarrow c_j \mid c_i, c_j \in \mathcal{C}, c_i \neq c_j, \text{ and } c_i \not\nearrow c_j\}$, of all homogeneity constraints. □

Notice that an instance $\mathcal{D}$ is homogeneous when every roll-up relation is total.

EXAMPLE 5. For the `Location` dimension instance in Figure 1, the roll up relation $\mathcal{R}^{\texttt{Region}}_{\texttt{AreaCode}}$ is {`(45, IX)`}. Since element `41` *does not appear* in this roll up relation as a first argument, the relation is not total. This makes the dimension instance heterogenous: $\mathcal{D} \not\models \texttt{AreaCode} \Rightarrow \texttt{Region}$. □

REMARK 1. In this work we will make some common assumptions [26]. They do not trivialize our problems, but make the presentation easier to follow. Our results can be easily modified in order to take into account situations where those assumptions are not met. They are: (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, $\texttt{NULL} \notin \mathcal{M}$. The semantics of `NULL` will be as in SQL relational databases, with a semantics *à la* SQL, as logically captured in [12]. □

# 3. ROLAP AND MDDB SEMANTICS

## 3.1 Star schema revisited

The relational model based on the star schema is the most common (relational) representation of a dimensional database. In this case, a fact table consisting of numerical measurements is directly joined to a set of dimension tables that contain descriptive attributes [35].

An example of this structure could be obtained from Figures 1b and 1c, if we create a referential IC from the latter to a *single relation* representing the former [35, 40, 27, 38]. In it, the categories are captured as attributes, and each base element with its parents and grand parents generates a tuple for the relational table.

Figure 3 shows the representation of the `Location` dimension as a relational database instance for a star schema. Notice that in case of multiple parents in an upper category for a given base element, more than one tuple are needed to represent the base element with its ancestors. That is why $\texttt{N}_3$ appears in two tuples in Figure 3.

| Number | Area Code | City | Region | All |
|---|---|---|---|---|
| $\texttt{N}_1$ | 41 | TCH | VIII | all |
| $\texttt{N}_2$ | 45 | TEM | IX | all |
| $\texttt{N}_3$ | 45 | CCP | IX | all |
| $\texttt{N}_3$ | 45 | CCP | VIII | all |

Figure 3: Star representation of dimension in Figure 1

The star schema does not meet the criteria mentioned in Section 1 for an adequate relational representation of MDDB when summarizability conditions are being considered. One of its weaknesses is that checking homogeneity through relational ICs is problematic. One could think of checking homogeneity through the absence of null values. However, as our running example shows, this may not be possible. For instance, although the `Location` dimension in Figure 1 is heterogenous (check the parents for element `41`), its star representation in Figure 3 does not contain any `NULL`.

On the other side, checking a strictness constraint between two categories in the dimension schema can be done by checking a *functional dependency* (FD) between the corresponding attributes in the star representation.

An important problem with the star relational representation has to do with the 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. As a result, inverting this mapping might not generate a unique MD instance. (Cf. Section 7 for a detailed discussion of invertibility.)

## 3.2 Snowflake schema revisited

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 [38]. 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 [35, 27, 38]. Figure 4 shows the snowflake relational representation of the `Location` dimension instance.

Due to the hierarchical structure of the snowflake schema, capturing strictness through relational ICs is complicated: Since each category is mapped to a single table, in order to check strictness, several joins must be executed . For example, if we want to check strictness between categories `Number` and `Region` through the schema in Figure 4, we can see from the MD schema that there are two ways to reach category `Region` from category `Number`. At the relational level, we have to check each path by joining the corresponding tables. Through these paths and joins we can discover that $\texttt{N}_3$ is related to different elements in `Region`. In more general terms, and from the relational point of view, checking local strictness conditions, i.e. of the form $c_i \to c_j$, amounts to checking relational *equality generating dependencies* (EGD)

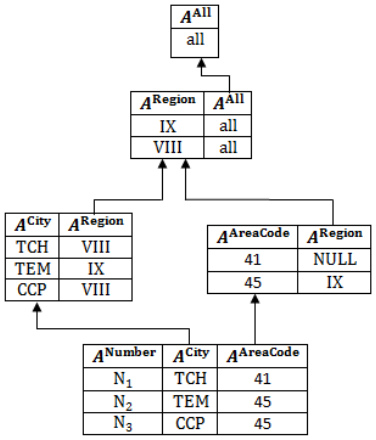



Figure 4: Snowflake representation of dimension in Figure 1

[1] in the snowflake database. They can be expressed in the relational calculus by universally quantified sentences of the form

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

where $\varphi$ is a formula that captures the required (and possible multiple) join, 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` value for `Region` in Figure 4. 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 simple relational ICs of the form

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

where $x_j \in \bar{x}$, and *NotNull* is 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, it is easy to check that the snowflake instance in Figure 4 is uniquely invertible to the `Location` MD dimension.

Due to the complexities involved in the representation (and also checking) of the strictness condition by relational ICs, we can not use snowflake relational representation in our approach.

# 4. MDDBS AS PATH INSTANCES

In this section we propose a relational representation for MDDBs that is well-behaved wrt our needs, namely: Simple relational representation and verification of MD summarizability conditions via relational ICs, invertibility of the MD2R mapping; and, as mentioned above, obtaining a simpler class of relational repairs, which will also make the final MD inversion process easier.

In Section 2, we formulated the strictness and homogeneity conditions on the basis of the roll up relations. The occurrence of a pair of data elements in a roll up relation indicates the existence of a *path* between those two elements in the hierarchy.

In order to better express the summarizability conditions in relational terms, the relational representation must store these paths in simple terms, allowing for their efficient verification. Inspired by existing XML-to-Relational mappings [44, 47], we propose a *path-based mapping* from MDDBs to relational databases.

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

The *path schema* we will introduce next represents each base-to-all path by means of a *single* database predicate. The categories along each path will be mapped to attributes of the corresponding relational table. Hence, it is possible to have a category mapped to *more than one* attribute in separate tables.

Definition 4. Given a dimension schema $\mathcal{S} = \langle \mathcal{C}, \nearrow \rangle$ and a dimension instance $\mathcal{D} = \langle \mathcal{M}, < \rangle$, a *p-instance* for a B2A path $P = \langle c_1, \ldots, c_n \rangle$ is an ordered list of elements $p = \langle e_1, \ldots, e_n \rangle$, with:[1]
(a) $\delta(e_i) = c_i$ or $e_i =$ `NULL`.
(b) Whenever $e_i$ and $e_{i+1}$ are both different from `NULL`, $e_i < e_{i+1}$.
(c) There is no p-instance $p'$ that can be obtained from $p$ by replacing `NULL`s in positions $i$ by non-`NULL` $e_i$s, that satisfies the first two conditions above.
The set of all p-instances for path $P$ is denoted by $\mathit{Inst}^{\mathcal{D}}(P)$. □

Condition (c) above enforces the use of non-null data elements whenever possible; or, equivalently, the use of `NULL` only when strictly needed. Notice that `NULL` is incomparable via $<$ with elements in $\mathcal{M}$. We are also assuming that `NULL` does not belong to $\mathcal{M}$. Also notice that if the base category is non-empty (something natural to assume), then there will be no p-instance starting with `NULL`.

Now we describe in precise terms the ***relational transformation*** $\mathcal{T}$ of both the dimensional schema and dimensional instance:

(A) (Schema transformation) 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 transformation) 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 2 shows the result of these *path mapping rules* applied to the `Location` dimension. The `Location` schema in Figure 1 has two B2A paths: $\mathtt{P}_1^{\mathtt{Loc}}$: ⟨`Number, AreaCode, Region, All`⟩ (Figure 2a), and $\mathtt{P}_2^{\mathtt{Loc}}$: ⟨`Number, City, Region, All`⟩ (Figure 2b). Each of these paths has 3 associated p-instances, and is mapped to a separate relational table using rule (A). Each of the resulting relations contains 3 tuples based on rule (B). As an example of instance mapping, the tuple $(\mathtt{N}_2, \mathtt{45}, \mathtt{IX}, \mathtt{all})$ in Figure 2a represents the p-instance ⟨$\mathtt{N}_2$`,45, IX, all`⟩, which belongs to $\mathit{Inst}^{\mathcal{D}}(P_1^{\mathtt{Loc}})$. □

[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.

Notice that the active domain, $Act(D)$, of the generated relational instance $D$ is contained in $\mathcal{M} \cup \{\texttt{NULL}\}$; and the domain, $Dom(A^c)$, of the generated attribute $A^c$ is $\delta^{-1}(c) \cup \{\texttt{NULL}\}$. Then, $Dom(A^c) \subseteq Act(D) \cup \{\texttt{NULL}\}$. We assume that $\texttt{all} \in Dom(A^{\texttt{All}})$, because we assumed that $\texttt{all} \in \mathcal{M}$. Also notice that even having $\texttt{all} \in \mathcal{M}$, for $A^{\texttt{All}}$ we may have the value NULL if all is not reached by lower-level elements in the given MD instance.

Finally, notice that the relational schema generated 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

A given MD instance $\mathcal{D}$ will come endowed with a set $\mathcal{K}$ of (local) strictness and homogeneity constraints as those in Definitions 1 and 2. $\mathcal{D}$ may not satisfy $\mathcal{K}$, which should be reflected in the violation by the associated relational instance $D$ of a corresponding set $\Sigma$ of relational ICs. Now we describe how to generate such a set $\Sigma$ from $\mathcal{K}$.

Checking a strictness condition, $c_i \to c_j$, between categories $c_i$ and $c_j$ under the path schema transformation depends on the set of B2A paths where both of these categories appear. Actually, this strictness condition must be checked within each single path, and also among pairs of paths containing $c_i$ and $c_j$.

To this end, we need functional dependencies (FDs) to check strictness on each path (cf. Rule (C) below). We also need an integrity constraint for each pair of paths. This becomes a very simple case of *equality generating dependencies* [1] (cf. Rule (D) below). They are much simpler than those needed in Section 3.2 (cf. (1)). (FDs form a particular class of EGDs.)

(C) (FD generation)

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

(D) (EGD generation)

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

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

EXAMPLE 7. (example 2 continued) In order to check global strictness for the Location dimension through the corresponding path instance, the following (reduced) set of FDs can be generated and verified:

$$RP_1^{\texttt{Loc}}: \quad \{A^{\texttt{Number}} \to A^{\texttt{AreaCode}},\ A^{\texttt{AreaCode}} \to A^{\texttt{Region}}\}. \quad (3)$$

$$RP_2^{\texttt{Loc}}: \quad \{A^{\texttt{Number}} \to A^{\texttt{City}},\ A^{\texttt{City}} \to A^{\texttt{Region}}\}. \quad (4)$$

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

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

□

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

Since we have introduced NULL in the relational representation, we may have to check the above relational dependencies against instances containing NULL. This is not an uncontroversial issue since several semantics offer themselves for relational database with null values. In this work, we are using a single null value, NULL. And we expect it to behave as in SQL databases. In consequence, we have to characterize in precise terms IC evaluation in the presence of such a null value. This was done in [12], where a characterization in classical predicate logic was provided. We cannot go into the details here. The essence of this logical reconstruction is the separation of attributes into *relevant* and *non-relevant*, depending on their occurrence or not in dependencies, and on their possibility of taking value NULL. For example, a join attribute is relevant, because it makes an important difference if it takes the null value or not.

In more precise terms, given a dependency $\psi$, it is transformed into a new first-order sentence $\psi^N$, 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 right-hand side we have usual first-order satisfaction where NULL is treated as any other constant.[3] The classical notion is applied to a new relational instance $D^{rel}$ obtained by restricting $D$ to its relevant attributes, and $\psi^N$ is a syntactic rewriting of $\psi$ that takes into account the possible occurrence of NULL. This is illustrated in the next example.

EXAMPLE 8. (example 7 continued) Consider dependency (5). Written as a usual first-order sentence it becomes:

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

However, this expression does not take into account the presence of NULL with its intended semantics. The set of relevant attributes for its evaluation is [12]:

$$Rel = \{RP_1^{\texttt{Loc}}.A^{\texttt{Number}},\ RP_1^{\texttt{Loc}}.A^{\texttt{Region}}, RP_2^{\texttt{Loc}}.A^{\texttt{Number}}, RP_2^{\texttt{Loc}}.A^{\texttt{Region}}\}.$$

The rewriting of $\psi$ into $\psi^N$ takes these relevant attributes into account and the possible presence of NULL in them. It becomes:

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

This sentence is checked against the instance $D^{rel}$ that results from restricting instance $D$ in Figure 2 to the attributes in *Rel*. This instance 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 $r' = \texttt{VIII}$, constraint (7) is not true in $D^{rel}$, where evaluation is done classically and taking NULL as any other constant in the domain. Hence, on the basis of (6), we have $D \not\models_N \psi$, which is in line with the local non-strictness of the original MD instance. □

Now, the homogeneity condition can be checked under the path schema using NOT NULL constraints (Rule (E) below).

[3]In particular, the *unique names assumption* applies to it, making it different from other constants, and also equal to itself.

A NULL as attribute value in a tuple shows that the preceding elements in the corresponding p-instance do not have ancestors all the way up. In Figure 2, we have two NULLs in table $RP_1^{\text{Loc}}$, which shows that the path is *disconnected* after element 41.

(E) (NOT NULL generation)

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

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

Example 9. (example 2 continued) We can check the Location dimension homogeneity through the following set of NOT NULL constraints:

$$\texttt{NOT NULL} \qquad RP_1^{\text{Loc}}.\{A^{\text{AreaCode}}, A^{\text{Region}}, A^{\text{All}}\}, \tag{8}$$

$$\texttt{NOT NULL} \qquad RP_2^{\text{Loc}}.\{A^{\text{City}}, A^{\text{Region}}, A^{\text{All}}\}. \tag{9}$$

We recall from Definition 4, that the first element in a p-instance never takes value NULL. In the path database of Figure 2, constraints NOT NULL $RP_1^{\text{Loc}}.A^{\text{Region}}$ and NOT NULL $RP_1^{\text{Loc}}.A^{\text{All}}$ are violated. □

# 6. REPAIRING PATH INSTANCES

The mappings introduced in the last two sections are such, that the MD instance is non-summarizable iff the generated path instance is inconsistent wrt the relational dependencies. This is where the idea of relational repairs comes into the picture.

First, we have to introduce appropriate relational repair operations for path instances; and, next, a notion of distance between instances that can be used to characterize the minimal repairs.

The relational dependencies we introduced in Section 5 are particular cases of *denial constraints*. For these constraints, consistency can be restored through tuple deletions or changes of attribute values [7]. Deleting a whole tuple from an inconsistent path instance implicitly removes a p-instance from the dimension. This would lead to an important loss of MD data. In consequence, we adopt here repairs that are obtained by changes of attribute values. A minimal relational repair will minimize the number of changes of attribute values (the update operations). This class of attribute-based repairs, the A-repairs, has been used and investigated before [24, 46, 11, 10, 39, 23], specifically for denial constraints in [24, 10], and for FDs in [46, 11].

Definition 5. Consider a relational instance $D$, possibly containing 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 $Dom(A) \setminus \{\texttt{NULL}\}$ assigned to attribute $A$ in the database atom $R(t) \in D$.[4]
(b) An *update* on $D$ is a finite set, $\rho$, of atomic updates on $D$ (that does not assign more than one new value to an existing attribute value $\bar{t}[A]$). The instance that results from applying $\rho$, i.e. simultaneously all the updates in $\rho$, to $D$ is denoted with $\rho(D)$.
(c) For a set $\Sigma$ of denial constraints (for the schema of $D$), an update $\rho$ on $D$ is a (minimal) repair if and only if: 1. $\rho(D) \models_N \Sigma$, and 2. there is no $\rho'$, such that $\rho'(D) \models_N \Sigma$, and $|\rho'| < |\rho|$. □

[4]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.

As we can see, 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 application of relational repairs to MDDBS, in Definition 5 we can restrict ourselves to sets $\Sigma$ of relational denial constrains of the form (C), (D), or (E) introduced above, i.e. just EGDs and NOT-NULL constraints. In the following we will assume that this is the case.

Example 10. (examples 7 and 9 continued) Consider the path instance $D$ in Figure 2, and the relational ICs obtained in Examples 7 and 9. The following are candidates to be repairs of $D$ (for simplicity we use only the tuple numbers (ids) shown in Figure 2):

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

Both of these updates restore the consistency of $D$. In addition to the ones shown above, there are other ways of repairing $D$. However, $\rho_1$ is the only minimal repair according to Definition 5. This repair changes the value for attribute Region in the third tuple of $RP_2^{\text{Loc}}$ from VIII to IX. On the corresponding MD side, this change implies modifying the link from element CCP to category Region. Still on the MD side, $\rho_1$ also, indirectly via a relational repair, creates an originally missing link, by assigning element VIII as the parent of element 41. This is done on the relational side by updating the first tuple in table $RP_1^{\text{Loc}}$. The impact of relational repair operations at the MD level is discussed in more detail in the next section.

Although $\rho_2$ is not a minimal relational repair, it can restore summarizability at MD level. This repair modifies the link between $\text{N}_3$ and category AreaCode, and also creates a link from 41 to VIII. These MD updates resolve both non-strictness and heterogeneity.

From multidimensional perspective, changing the parent of $\text{N}_3$ from CCP to TEM will also restore strictness in the Location dimension. Together with the insertion of a link between 41 and VIII, this MD repair is corresponded to the following relational updates:

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

Comparing $\rho_1$ with $\rho'$ reveals that, the latter performs an *unnecessary* update on $A^{\text{City}}$ in the third tuple of $RP_2^{\text{Loc}}$. Hence, based on Definition 5, $\rho'$ is not considered as a relational repair. In other words, although the aforementioned MD update restores summarizability in Location dimension, it does not correspond to a 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. That is, if the set $\Sigma$ of relational denial constraints is derived from a non-full set of homogeneity constraints on

the corresponding MD schema (cf. Definition 2(b)), a relational repair wrt $\Sigma$ may still have NULLs.

EXAMPLE 11. (example 9 continued) Instead of imposing *global* homogeneity as before, we are now interested in checking only a local homogeneity constraint, between categories Number and City, i.e. Number $\Rightarrow$ City. In this case, according to mapping rule (E), the only NOT NULL constraint generated is NOT NULL $RP_2^{\text{Loc}}.A^{\text{City}}$ (instead of all the ICs in (8) and (9)). This single NOT NULL constraint is satisfied by the path instance in Figure 2b.

With this single homogeneity constraint, if we assume that the EGDs and FDs for checking strictness are the same as those generated in Example 7, the path instance in Figure 2 would be minimally repaired as shown in Figure 5. The NULL values in the first tuple of table $RP_1^{\text{Loc}}$ are not updated (5a), since they do no violate any of the imposed NOT NULL constraints.

Notice that homogeneity constraints of the form $c_i \Rightarrow c_j$, as introduced in Definition 2, require that $c_i \nearrow c_j$ holds in the schema, i.e. a single link connects $c_i$ to $c_j$. Thus, if we wanted to impose the (non-official) local homogeneity constraint Number $\Rightarrow$ Region, we would have to do it by imposing four additional (local) homogeneity constraints: Number $\Rightarrow$ AreaCode, AreaCode $\Rightarrow$ Region, Number $\Rightarrow$ City and City $\Rightarrow$ Region. This would lead to additional relational constraints:

NOT NULL $RP_1^{\text{Loc}}.A^{\text{AreaCode}}$, NOT NULL $RP_2^{\text{Loc}}.A^{\text{City}}$,
NOT NULL $RP_1^{\text{Loc}}.A^{\text{Region}}$, and NOT NULL $RP_2^{\text{Loc}}.A^{\text{Region}}$.

□

Restoring consistency wrt strictness constraints through relational repairs does not modify the NULL values in the database. The semantics introduced for evaluating FDs and EGDs in presence of NULL values does not consider NULLs as a source of IC violation. Strictness is violated when we have more than one parent for an element in an upper category. Hence, strictness is not violated if an element rolls up to a non-null value and NULL in a parent category. The latter being reached due to a *missing parent* in the parent category.

EXAMPLE 12. (example 8 continued) Consider the EGD (5) obtained from the MD strictness constraint Number $\rightarrow$ Region. The first-order sentence (7) can be used for checking the satisfaction of the relational EGD in the presence of NULL.

In order to illustrate the effect of NULL on the evaluation of strictness constraints on the instance in Figure 2, we instantiate the universal sentence (7) on the first tuples of tables $RP_1^{\text{Loc}}$ and $RP_2^{\text{Loc}}$, obtaining

$$RP_1^{\text{Loc},Rel}(\text{N}_1, \text{NULL}) \wedge RP_2^{\text{Loc},Rel}(\text{N}_1, \text{VIII})) \wedge \mathit{NotNull}(\text{N}_1)$$
$$\wedge \mathit{NotNull}(\text{NULL}) \wedge \mathit{NotNull}(\text{VIII}) \;\rightarrow\; \text{NULL} = \text{VIII}.$$

Due to the occurrence of NULL in relevant attributes, the antecedent of this implication has the false conjunct *NotNull*( NULL), which makes the whole sentence true. Hence, the instance in Figure 2, even having $\text{N}_1$ 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 $\text{N}_1$ is connected only to VIII (cf. Figure 1b). □

It should be clear by now that, in general, repairs of relational path instances associated to MD instances that violate homogeneity will be guaranteed to be NULL-free iff the homogeneity constraint is imposed globally, i.e. to all pairs of parent-child categories. Since the violation of strictness has nothing to do with NULL values (as shown in the previous example), the question of occurrence of NULL in a repaired path instance depends only on the scope of the homogeneity conditions.

THEOREM 1. For a relational path instance $D$ and a set $\Sigma$ 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 $\Sigma$.

**Proof:** It suffices to build a path instance $D'$ obtained by an update $\rho$ applied to $D$, such that $D' \models_N \Sigma$. If such an update $\rho$ exists, it immediately follows that there is a minimal one.

For each attribute $A^c$, except the first column in each path table, select an arbitrary value $v^{A^c} \in \mathit{Dom}(A^c) \setminus \{\text{NULL}\}$. Since we assume that every category $c$ has at least one element for a given instance, $\mathit{Dom}(A^c) \setminus \{\text{NULL}\}$ is non-empty.

The relational update $\rho$ contains a set of atomic updates such that, in each table $RP_i$ that $A^c$ appears, the value of this attribute is updated to $v^{A^c}$ in all the tuples. We can show that the instance resulting from applying $\rho$ to $D$, $D' = \rho(D)$, satisfies ICs of the form (C)-(E), i.e. the set $\Sigma$.

Since the values of all of the attributes (except the first one in each table) are updated to non-null values, the NOT NULL constraints of the form (E) are all satisfied. (Notice that, according to Definition 4, the first attribute of a path table cannot be NULL.) Moreover, because of the unique value selected for each attribute, the FDs and EGDs of the form (C) and (D) cannot be violated. As a result, it holds $D' \models_N \Sigma$.

Since the set of repairs for $D$ is finite, and there is a partial order for comparing two repairs (see Definition 5), we can conclude that, there is always a minimal relational repair for the inconsistent path instance $D$. □

EXAMPLE 13. (example 10 continued) Figure 6 shows a possible non-minimal repair, $\rho$, obtained as described in the proof of Theorem 1. Here, we are choosing $v^{A^{\text{AreaCode}}} = \text{41}$, $v^{A^{\text{Region}}} = \text{VIII}$, $v^{A^{\text{All}}} = \text{all}$, and $v^{A^{\text{City}}} = \text{TEM}$.

Notice that, for the shared attribute $A^{\text{Region}}$, we selected one value to be applied to both tables. Comparing $\rho$ to the minimal repair $\rho_1$ obtained in Example 10 reveals that $\rho$ performs 6 more attribute updates than $\rho_1$, which executes only 3 updates. As a result, according to Definition 5, $\rho$ cannot be considered a minimal repair. □

## 7. BACK TO MD INSTANCES: INVERSION

We have defined MD repairs via the translation of the MD database into a relational instance that is subject to relational ICs. The latter are derived from semantic MD constraints. The generated relational instance is repaired wrt the ICs. Now, it is natural to ask about the kind of repairs that are obtained going through the relational route. We will address this question in more detail in Section 8. 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 clear situation of schema mappings and their invertibility [22, 4].

| $A^{\text{Number}}$ | $A^{\text{AreaCode}}$ | $A^{\text{Region}}$ | $A^{\text{All}}$ |
|---|---|---|---|
| $N_1$ | 41 | NULL | NULL |
| $N_2$ | 45 | IX | all |
| $N_3$ | 45 | IX | all |

(a) Repaired $RP_1$

| $A^{\text{Number}}$ | $A^{\text{City}}$ | $A^{\text{Region}}$ | $A^{\text{All}}$ |
|---|---|---|---|
| $N_1$ | TCH | VIII | all |
| $N_2$ | TEM | IX | all |
| $N_3$ | CCP | ~~VIII~~ IX | all |

(b) Repaired $RP_2$

Figure 5: An example of a repaired path instance containing NULL

| $A^{\text{Number}}$ | $A^{\text{AreaCode}}$ | $A^{\text{Region}}$ | $A^{\text{All}}$ |
|---|---|---|---|
| $N_1$ | 41 | VIII | all |
| $N_2$ | 41 | VIII | all |
| $N_3$ | 41 | VIII | all |

(a) A repair for $RP_1$

| $A^{\text{Number}}$ | $A^{\text{City}}$ | $A^{\text{Region}}$ | $A^{\text{All}}$ |
|---|---|---|---|
| $N_1$ | TEM | VIII | all |
| $N_2$ | TEM | VIII | all |
| $N_3$ | TEM | VIII | all |

(b) A repair for $RP_2$

Figure 6: A non-minimal repair according to Theorem 1

Notice that mapping $\mathcal{T}$ has two components, the schema and the instance transformations; 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 the usual good properties. More precisely, the following should hold:

**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 $\Sigma$ are their (generated) relational counterparts, and $\rho(D)$ is a repair of $D$ wrt $\Sigma$, then $\mathcal{T}^{-1}(\rho(D)) \models \mathcal{K}$ holds.

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

1. For every predicate $R[A_1, \cdots, A_n] \in \mathcal{R}$, it holds that $A_n = A^{\texttt{All}}$. In particular, all the relational predicates share this "final" attribute. We also assume that $Dom(A^{\texttt{All}}) = \{\texttt{all}, \texttt{NULL}\}$. Furthermore, we will assume that all the predicates $R$ share the "first attribute" $A_1$ whose domain does not contain NULL. The reason is that, we are assuming there is a single base category at the MD level. All the other attributes are allowed to take the value NULL. Notice also that different predicates $R_i$ may share attributes other than $A_1$ and their last ones.

2. The elements of $\Sigma$ 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. Being $D$ an instance of schema $\mathcal{R}$, it does satisfy the basic conditions about the attribute values expressed in item 1. However, we do not require at this stage that it also satisfies the relational constraints in $\Sigma$.

   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 $i \leq j \leq n$.

Now, for the definition of $\mathcal{T}^{-1}(\mathcal{R})$ we associate attributes to categories. And the joint and consecutive occurrence of attributes in a relational predicate generates direct links between categories (cf. rule (F) below). On the other side, the definition of $\mathcal{T}^{-1}(D)$ can be done by considering each tuple separately, and creating the corresponding p-instance for a dimension instance $\mathcal{M}$ of MD schema $\mathcal{S} := \mathcal{T}^{-1}(\mathcal{R})$.[5] This creates elements in categories and links between elements in directly connected categories (cf. rule (G) below).

(F) (Schema inversion) For each attribute $A$ appearing in some $R \in \mathcal{R}$, 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.

(G) (Instance inversion) For each relational tuple $R(e_1, \ldots, e_n)$, 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) $c^{A_i}$ (or, more precisely, make $\delta(e_i) := c^{A_i}$). Furthermore, if $e_{i+1} \neq \texttt{NULL}$, create an edge from $e_i$ to $e_{i+1}$. In this way we create an MD instance $\mathcal{M}$.

Example 14. (example 10 continued) Figure 7a shows the minimal repair of a path instance that we had obtained. Applying the above inversion rules, this relational repair generates the dimension instance in Figure 7b. The dashed lines correspond to inserted edges.

For example, using rule (F) on $RP_1^{\text{Loc}}$, the top table in Figure 7a, we obtain a set of categories:[6]

$$RP_1^{\text{Loc}}[A^{\text{Number}}, A^{\text{AreaCode}}, A^{\text{Region}}, A^{\text{All}}] \mapsto \{\texttt{Number}, \texttt{AreaCode}, \texttt{Region}, \texttt{All}\} \subseteq \mathcal{C},$$

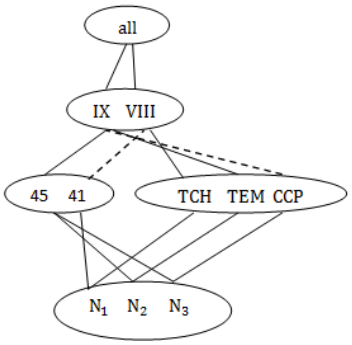
[5] Since the attributes $A_j$ in $\mathcal{R}$ may not have names of the form $A^c$, for $c$ a category name, we will obtain an MD schema that will be "isomorphic" to the original $\mathcal{S}$, if any. We will still denote with $\mathcal{S}$ the MD schema resulting from the inversion.

[6] Naturally, identifying the generated category $c^{A^c}$ simply with $c$.

| $A^{\text{Number}}$ | $A^{\text{AreaCode}}$ | $A^{\text{Region}}$ | $A^{\text{All}}$ |
|---|---|---|---|
| $N_1$ | 41 | ~~NULL~~ VIII | ~~NULL~~ all |
| $N_2$ | 45 | IX | all |
| $N_3$ | 45 | IX | all |

| $A^{\text{Number}}$ | $A^{\text{City}}$ | $A^{\text{Region}}$ | $A^{\text{All}}$ |
|---|---|---|---|
| $N_1$ | TCH | VIII | all |
| $N_2$ | TEM | IX | all |
| $N_3$ | CCP | ~~VIII~~ IX | all |

(a) Path instance




(b) Dimension via inversion

Figure 7: Dimension instance obtained by inverting a relational path instance.

and also a set of $\nearrow$-links:

$$\{\texttt{Number} \nearrow \texttt{AreaCode}, \texttt{AreaCode} \nearrow \texttt{Region}, \texttt{Region} \nearrow \texttt{All}\}.$$

For an example of instance inversion with rule (G), the updated (first) tuple $(\texttt{N1}, \texttt{41}, \texttt{VIII}, \texttt{all})$ in the top table is mapped as follows:

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

Applying the same inversion rule to other tuples in the relational instance, we obtain an MD instance $\mathcal{D}$. In this example, the MD instance in Figure 7b, obtained via the inversion of the relational path repair, turns out to be (globally) strict and homogeneous. In fact, since $D \models \Sigma$ for the path instance in Figure 7a (with $\Sigma$ 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 $\Sigma$. □

It should be clear from these rules that a *unique* dimension instance is obtained as a result of the application of these 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.

As the previous example shows, we can obtain a repair for the original MD instance via the detour through minimal repairs of relational path instances. The results and properties of this strategy are explored in the next section.

# 8. A PURELY MD REPAIR SEMANTICS

We have defined *a repair semantics for MD databases wrt summarizability constraints*. This 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 (LSCs), i.e. strictness and homogeneity constraints, on the MD side is translated into a set of relational constraints $\Sigma$. So, as $\mathcal{D}$ may not satisfy $\mathcal{K}$, $D$ may not satisfy $\Sigma$. In consequence, we consider relational repairs for $D$ wrt $\Sigma$. These are *attribute-based repairs* [8] that restore the consistency of $D$ wrt $\Sigma$ by changing attribute values and minimizing the number of these changes. These relational repairs form a class denoted by $Rep(D, \Sigma)$. We showed that inverting these relational repairs takes us to a class of MD instances $\mathcal{D}'$ that satisfy $\mathcal{K}$.

Definition 6. Let $\mathcal{D}$ be an MD instance, $\mathcal{K}$ be a set of LSCs, $D$ be the relational instance $\mathcal{T}(\mathcal{D})$, and $\Sigma$ be 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 $D' \in Rep(D, \Sigma)$, such that $\mathcal{D}' = \mathcal{T}^{-1}(D')$. $Rep(\mathcal{D}, \mathcal{K})$ denotes the class of path repairs of $\mathcal{D}$ wrt $\mathcal{K}$. □

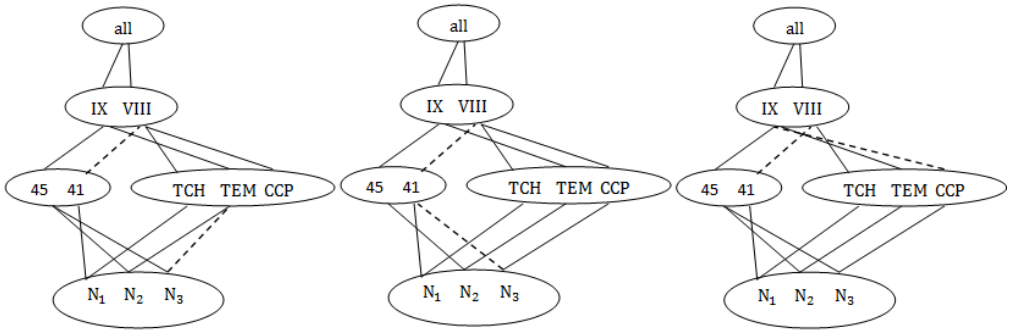
Since the relational repairs in $Rep(D, \Sigma)$ only change data (as opposed to the relational schema), the MD repairs in $Rep(\mathcal{D}, \mathcal{K})$ also only change data, i.e. instance $\mathcal{D}$, keeping the MD schema $\mathcal{S}$ intact. As such, they are *data-based repairs* of $\mathcal{D}$. More specifically, an MD repair does not add new elements to categories. Actually, the MD data repair operations are only insertions or deletions of edges in the dimension instance. More specifically, violations of `NOT NULL` relational constraints (associated to the lack of homogeneity in the MD counterpart) result in insertions of edges in $\mathcal{D}$. The operations that tackle non-strictness (EGD violations on the relational side) may produce both insertions and deletions of links.

There have been previous approaches to MD data-based repairs.[7] In [9], data-based repairs assume homogeneity, and only address strictness. In one way or another, MD data-based repairs have also been considered in [17, 42]. An approach that is closest to ours can be found in [13] (cf. also [19]). They define MD repairs directly on the MD instance, wrt both local strictness and homogeneity constraints. Their data operations are also and only insertions and deletions of edges between existing data elements. Actually, in [13], a minimal repair is defined as a new dimension that is consistent with respect to the summarizability constraints, and is obtained by applying a minimal number of updates (edge insertions or deletions) to the original dimension. In that work, they obtain a class of MD repairs, let us denote it with $Rep^{bch}(\mathcal{D}, \mathcal{K})$, that could be compared to our class $Rep(\mathcal{D}, \mathcal{K})$.

Example 15. (examples 10 and 14 continued) Figure 8 shows the minimal repairs generated according to [13], i.e. the elements of $Rep^{bch}(\mathcal{D}, \mathcal{K})$, for the `Location` dimension in Figure 1.

The MD repair $\mathcal{D}_3$ corresponds to the one obtained in Example 14 via our relational translation approach, which produces only this single MD repair. Consequently, in this example, $Rep(\mathcal{D}, \mathcal{K}) = \{\mathcal{D}_3\} \subsetneq Rep^{bch}(\mathcal{D}, \mathcal{K})$.

[7] Repairs based on changes on the MD schema have also been considered [27, 30, 28], and more formally in [5].




(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

For $\mathcal{D}_2$ in Figure 8a, it is easy to check that it can be obtained with the inversion rules after applying the update $\rho_2$ in Example 10, where we saw that $\rho_2$ is not a minimal update. This is why $\mathcal{D}_2$ is not obtained as an element of $Rep(\mathcal{D}, \mathcal{K})$. Something similar happens with $\mathcal{D}_1$. It corresponds to the update $\rho'$ also discussed in Example 10, where we found that it does not produce a minimal relational repair. Thus, $\mathcal{D}_1$ does not belong to $Rep(\mathcal{D}, \mathcal{K})$ either.

Summarizing, $\mathcal{D}_1$ and $\mathcal{D}_2$ are MD repairs with the notion of minimality used in [13], but they are not minimal when seen as repairs of the underlying and associated relational path instance. $\mathcal{D}_3$ is the only repair that is minimal in both ways. □

The previous example might suggest that it is always the case that $Rep(\mathcal{D}, \mathcal{K}) \subseteq Rep^{bch}(\mathcal{D}, \mathcal{K})$. However, this is not true in general since there are examples of MD schemas and instances where repairs in $Rep(\mathcal{D}, \mathcal{K})$ are not elements of $Rep^{bch}(\mathcal{D}, \mathcal{K})$.

Example 16. Figure 9 shows an alternative instance $\mathcal{D}$ for the dimension schema in Figure 1a. For simplicity, bold edges are used to denote multiple edges, connecting each element at the bottom end to the single and same element at the top end. Consider the set of relational ICs obtained in Examples 7 and 9 for global strictness and homogeneity, resp. Here, $\mathcal{D} \not\models \texttt{Number} \rightarrow \texttt{Region}$, because $\texttt{N}_1, \cdots, \texttt{N}_5$ have each two parents, `VIII` and `IX`, in category `Region`.

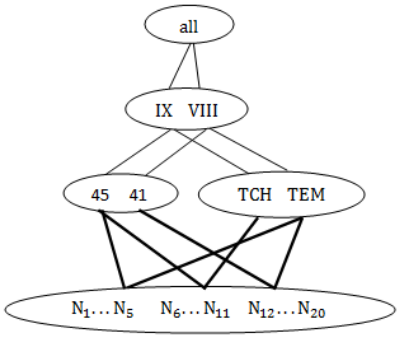



Figure 9: A non-strict instance for the `Location` dimension

There are two minimal repairs for the path relational instance corresponding to $\mathcal{D}$, via the following updates:

$$\begin{aligned}\rho_1 &= \{\langle RP_1^{\texttt{Loc}}(1, \cdots, 5), A^{\texttt{Region}}, \texttt{VIII}\rangle,\\ &\qquad \langle RP_1^{\texttt{Loc}}(1, \cdots, 5), A^{\texttt{AreaCode}}, \texttt{41}\rangle\},\\ \rho_2 &= \{\langle RP_2^{\texttt{Loc}}(1, \cdots, 5), A^{\texttt{Region}}, \texttt{IX}\rangle,\\ &\qquad \langle RP_2^{\texttt{Loc}}(1, \cdots, 5), A^{\texttt{City}}, \texttt{TCH}\rangle\}.\end{aligned}$$

It holds $|\rho_1| = |\rho_2| = 10$. The corresponding MD repairs, $\mathcal{D}_1$ and $\mathcal{D}_2$, resp., are those in Figure 10.

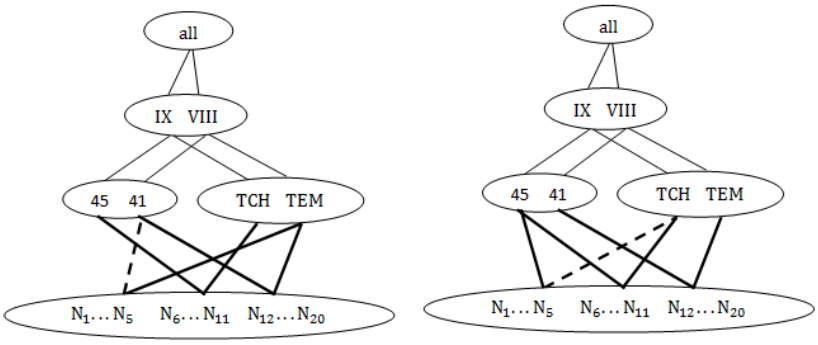
On the other side, there are two minimal repairs obtained according to [13], those in Figure 11. Each of them performs two edge modifications. However, the effect of those MD changes on the relational side is not minimal: $\mathcal{D}_1^{bch}$ causes 17 attribute changes, and $\mathcal{D}_2^{bch}$ 23 attribute changes. Thus, in this example, the classes $Rep(\mathcal{D}, \mathcal{K})$ and $Rep^{bch}(\mathcal{D}, \mathcal{K})$ are disjoint. □

Despite this incomparability of repair classes under set inclusion, it is still worth comparing in more detail $Rep(\mathcal{D}, \mathcal{K})$ and $Rep^{bch}(\mathcal{D}, \mathcal{K})$ for our ongoing example, to gain additional insight that could lead us to a purely MD characterization for our MD repairs.

Example 17. (example 15 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` for enforcing homogeneity. However, each of them has a different approach to the enforcement of strictness. Actually, the edges deleted or inserted by repairs $\mathcal{D}_1$ and $\mathcal{D}_2$ belong to the first level of the dimension hierarchy, while repair $\mathcal{D}_3$ makes changes on its second level.

Repair $\mathcal{D}_1$ changes the link between element $\texttt{N}_3$ and category `City`, updating the p-instance $\langle \texttt{N}_3,\texttt{CCP},\texttt{VIII},\texttt{all}\rangle$ into $\langle \texttt{N}_3,\texttt{TEM},\texttt{IX},\texttt{all}\rangle$. However, with repair $\mathcal{D}_3$ the aforementioned p-instance is changed into $\langle \texttt{N}_3,\texttt{CCP},\texttt{IX},\texttt{all}\rangle$. So, $\mathcal{D}_1$ causes more changes on this p-instance in comparison to $\mathcal{D}_3$. We find a similar relationship between repairs $\mathcal{D}_2$ and $\mathcal{D}_3$. □

We can see from this example that, modifying different edges may be reflected in different ways on the underlying relational database. Hence, the notion of MD 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). In the following, we will define an MD measure that takes this into consideration.




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

Figure 10: Minimal repairs in $Rep(\mathcal{D}, \mathcal{K})$ for the `Location` dimension

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 $\delta$.
(a) The sets of *insertions*, *deletions* and *modifications* as a result of updating $\mathcal{D}$ into $\mathcal{D}'$ are, respectively:

$$ins(\mathcal{D}, \mathcal{D}') = \{ (e_1, e_2) \in (<_{\mathcal{D}'} \setminus <_{\mathcal{D}}) \mid \text{there is no } e_3 \text{ with } (e_1, e_3) \in (<_{\mathcal{D}} \setminus <_{\mathcal{D}'})\}.$$

$$del(\mathcal{D}, \mathcal{D}') = \{ (e_1, e_2) \in (<_{\mathcal{D}} \setminus <_{\mathcal{D}'}) \mid \text{there is no } e_3 \text{ with } (e_1, e_3) \in (<_{\mathcal{D}'} \setminus <_{\mathcal{D}})\}.$$

$$mod(\mathcal{D}, \mathcal{D}') = \{ (e_1, e_2, e_3) \mid (e_1, e_2) \in (<_{\mathcal{D}'} \setminus <_{\mathcal{D}}) \text{ and } (e_1, e_3) \in (<_{\mathcal{D}} \setminus <_{\mathcal{D}'})\}.$$

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

$$ucost(\mathcal{D}, \mathcal{D}') = \sum_{(e_1,e_2) \in (ins(\mathcal{D},\mathcal{D}') \cup del(\mathcal{D},\mathcal{D}'))} |\alpha(e_1, e_2)| \times |\beta(e_2)| + \sum_{(e_1,e_2,e_3) \in mod(\mathcal{D},\mathcal{D}')} |\alpha(e_1, e_2)| \times |\gamma(e_2, e_3)|,$$

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

Intuitively, the cost of updating $\mathcal{D}$ to $\mathcal{D}'$ corresponds to the number of changes made to the elements of p-instances belonging to $\mathcal{D}$. From the corresponding relational point of view, this value is the number of changes of attribute-values, as a result of the MD updates. We compute the number of updated attribute values as a result of each edge change separately. The sum of these values for all changed edges $(e_1, e_2)$ captures the total number of attribute value updates needed for updating $\mathcal{D}$ to $\mathcal{D}'$. This is the quantity computed by *ucost*.

More technically, $\alpha(e_1, e_2)$ in Definition 7 reflects the set of p-instances that will be updated by changing edge $(e_1, e_2)$. On the relational side, the size of this set is equal to the number of tuples that will be affected as a result of this edge change. On the other hand, as discussed around Example 17, the level of the modified edge is an important factor in measuring the number of changes made to the p-instances. In Definition 7, the functions $\beta$ and $\gamma$ are used as an indicator of the level of edge $(e_1, e_2)$. More specifically, for each aforementioned p-instance in $\alpha(e_1, e_2)$, $|\beta|$ and $|\gamma|$ represent the number of changes made to the elements preceding $e_2$, depending on whether the edge is inserted/deleted or modified. The resulting value associated to such an edge $(e_1, e_2)$ equals to the number of attribute value updates made to the tuple corresponding to $p$ in the underlying path instance. Thus, $ucost(\mathcal{D}, \mathcal{D}')$ shows the total number of changes made to the set of p-instances, i.e. the total number of attribute value updates that are needed on an underlying path instance for updating $\mathcal{D}$ into $\mathcal{D}'$.

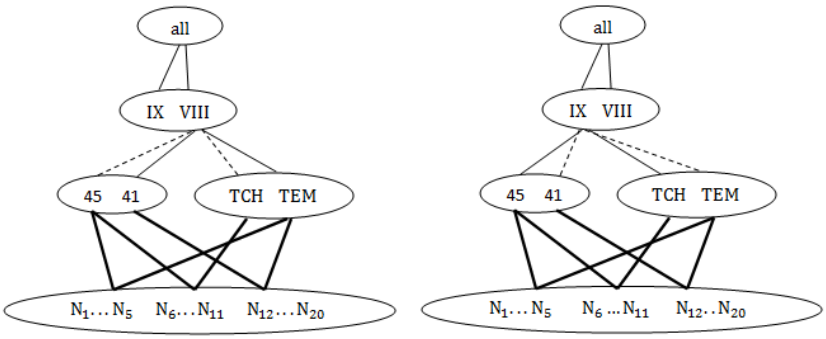
EXAMPLE 18. (example 17 continued) Let us calculate the cost of updating $\mathcal{D}$ into each of the repairs $\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3$. First, we have:
$ins(\mathcal{D}, \mathcal{D}_1) = \{(\texttt{41}, \texttt{VIII})\}$, $del(\mathcal{D}, \mathcal{D}_1) = \emptyset$, $mod(\mathcal{D}, \mathcal{D}_1) = \{(\texttt{N}_3, \texttt{TEM}, \texttt{CCP})\}$,
$ins(\mathcal{D}, \mathcal{D}_2) = \{(\texttt{41}, \texttt{VIII})\}$, $del(\mathcal{D}, \mathcal{D}_2) = \emptyset$, $mod(\mathcal{D}, \mathcal{D}_2) = \{(\texttt{N}_3, \texttt{41}, \texttt{45})\}$,
$ins(\mathcal{D}, \mathcal{D}_3) = \{(\texttt{41}, \texttt{VIII})\}$, $del(\mathcal{D}, \mathcal{D}_3) = \emptyset$, $mod(\mathcal{D}, \mathcal{D}_3) = \{(\texttt{CCP}, \texttt{IX}, \texttt{VIII})\}$,
The sets $\alpha$, $\beta$ and $\gamma$ are associated to instance $\mathcal{D}$. Here, they are as follows:
$\alpha(\texttt{41}, \texttt{VIII}) = \{\langle \texttt{N}_1,\ \texttt{41},\ \texttt{NULL},\ \texttt{NULL}\ \rangle\}$, $\alpha(\texttt{N}_3, \texttt{TEM}) = \{\langle \texttt{N}_3, \texttt{CCP}, \texttt{VIII}, \texttt{all}\ \rangle\}$, $\alpha(\texttt{N}_3, \texttt{41}) = \{\langle \texttt{N}_3, \texttt{45}, \texttt{IX}, \texttt{all}\ \rangle\}$, $\alpha(\texttt{CCP}, \texttt{IX}) = \{\langle \texttt{N}_3, \texttt{CCP}, \texttt{VIII}, \texttt{all}\ \rangle\}$;
$\beta(\texttt{VIII}) = \{\texttt{VIII}, \texttt{All}\}$;
$\gamma(\texttt{TEM}, \texttt{CCP}) = \{\texttt{TEM}, \texttt{IX}\}$, $\gamma(\texttt{41}, \texttt{45}) = \{\texttt{41}, \texttt{VIII}\}$, $\gamma(\texttt{IX}, \texttt{VIII}) = \{\texttt{IX}\}$.
With these elements we can compute the update costs for each case:

$$ucost(\mathcal{D}, \mathcal{D}_1) = |\alpha(\texttt{41}, \texttt{VIII})| \times |\beta(\texttt{VIII})| + |\alpha(\texttt{N}_3, \texttt{TEM})| \times |\gamma(\texttt{TEM}, \texttt{CCP})| = 1 \times 2 + 1 \times 2 = 4,$$

$$ucost(\mathcal{D}, \mathcal{D}_2) = |\alpha(\texttt{41}, \texttt{VIII})| \times |\beta(\texttt{VIII})| + |\alpha(\texttt{N}_3, \texttt{41})| \times |\gamma(\texttt{41}, \texttt{45})| = 1 \times 2 + 1 \times 2 = 4,$$

$$ucost(\mathcal{D}, \mathcal{D}_3) = |\alpha(\texttt{41}, \texttt{VIII})| \times |\beta(\texttt{VIII})| + |\alpha(\texttt{CCP}, \texttt{IX})| \times |\gamma(\texttt{IX}, \texttt{VIII})| = 1 \times 2 + 1 \times 1 = 3.$$

We can see that $\mathcal{D}_3$ provides the least update cost for the original instance $\mathcal{D}$, i.e. the minimum number of changes to the relational database. This fact is consistent with the observations made in Example 14: $\mathcal{D}_3$ is the only minimal MD repair for $\mathcal{D}$ that also corresponds to a minimal relational repair.




(a) $\mathcal{D}_1^{bch}$ (b) $\mathcal{D}_2^{bch}$

Figure 11: Minimal repairs in $Rep^{bch}(\mathcal{D}, \mathcal{K})$ for the `Location` dimension

Notice that, the update cost for each of the MD repairs $\mathcal{D}_1$, $\mathcal{D}_2$ and $\mathcal{D}_3$ is the same as the number of changes made to the p-instances belonging to $\mathcal{D}$, and also the same as number of attribute-value (column) updates (the $|\rho|$s) performed on the corresponding relational repairs. □

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

**Proof:** The number of attribute value updates caused by $\rho$ is equal to the number of changed tuples multiplied by the number of attribute values updated in each of them. In the following, we verify that the *ucost* function computes these values.

Obviously, attribute values change only as a result of inserting/deleting/modyfing an edge in the MD instance (cf. Definition 7). *ucost* considers each edge change separately, computes the aforementioned values for it and then sums up the computed values for all edge changes occurred in the MD update.

Now, for every such edge change, the number of p-instances containing that edge represents the number of tuples that will be modified in the path database. The set $\alpha$ for a given edge in Definition 7 contains those p-instances. For each affected tuple, we need to compute the number of attributes in the path database that will be updated as a result of such edge change. This computation depends on whether the edge was inserted, deleted or modified. Multiplying this value by the number of affected tuples (those in its $|\alpha|$-set) equals the total number of changed columns for such edge change.

In case of inserting or deleting an edge $(e_1, e_2)$, the number of attribute value updates for each tuple is equal to the number of ancestors of element $e_2$, which is represented through $|\beta|$. On the other hand, for the case of an edge modification, say $(e_1, e_2)$ to $(e_1, e_3)$, we need to exclude common ancestors of $e_2$ and $e_3$ from the computation, which is taken care of by function $|\gamma|$.

Finally, $ucost(\mathcal{D}, \mathcal{D}')$ represents the sum of all of the above values computed for all edge changes, and is therefore equal to the total number of column updates needed to update dimension $\mathcal{D}$ into $\mathcal{D}'$. □

From this lemma, we easily obtain a characterization of our MD repairs, that were defined through our relational translation, 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). □

**Proof:** In one direction, we have to prove that, for an MD instance $\mathcal{D}'$ satisfying $\mathcal{K}$ with minimum *ucost*, there exists $D' \in Rep(D, \Sigma)$, such that $\mathcal{D}' = \mathcal{T}^{-1}(D')$ (cf. Definition 6). From Lemma 1 we have that, for any MD update $\mathcal{D}'$, *ucost* $= |\rho|$, where $\mathcal{T}^{-1}(\rho(D)) = \mathcal{D}'$. So, a minimum *ucost* implies a minimum $|\rho|$, which itself leads to $\rho(D) \in Rep(D, \Sigma)$. In other words, minimizing *ucost* implies minimizing the number of atomic updates performed at relational level.

In the other direction, we need to show that, for any MD repair $\mathcal{D}' \in Rep(\mathcal{D}, \mathcal{K})$, $ucost(\mathcal{D}, \mathcal{D}')$ will be minimum. According to Definition 6, for every $\mathcal{D}' \in Rep(\mathcal{D}, \mathcal{K})$, there exists a $D' \in Rep(D, \Sigma)$, such that $\mathcal{D}' = \mathcal{T}^{-1}(D')$. From Definition 5, it holds that $D'$ corresponds to a relational repair $\rho$ which performs a minimum number of attribute value updates on $D$, i.e. $|\rho|$ is minimum. Hence, based on Lemma 1, the *ucost* must also be minimum. □

This characterization of MD repairs implicitly takes into consideration the effect of MD repair operations (edge insertions and deletions) on the underlying relational layer. As we showed above, not all of the repairs proposed by [13] have to be MD repairs in our sense, nor the other way around. This is due to the fact that, in [13] edges are modified without considering (neither explicitly nor implicitly) the underlying relational side effects of the MD operations.

To conclude this section, let us emphasize that the repair approach in [13] applies minimality without considering any sort of priority of change on edges. In contrast, our MD characterization of minimal repairs implicity does, via the underlying relational side-effects of MD edge modifications. In particular, in an MD repair process, edges with fewer connections to the base elements are good candidates for modification, since they affect fewer tuples in the underlying database. Among these edges, those that reside at the higher levels of the hierarchy are optimal choices for change during MD repair, since they update fewer attributes in the

affected tuples. As a result, following our approach, those MD repairs causing minimal side effects will be preferred.

# 9. EXPERIMENTS

## 9.1 Query answering

According to our approach, repairing MD databases is done via relational transformations that have some clear conceptual and theoretical advantages (cf. Section 8). 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 a non-negligible 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 an interesting alternative to consider for a ROLAP approach to MDDBs and DWHs.* And this is independent from its virtues in relation to MD repairing.

Accordingly, in this section we provide experimental support for this claim, comparing a PR implementation with relational implementations based on the star and snowflake schemas. We consider the performance at aggregate query answering, and we use SQL Server 2008 for our experiments. They are based on the running example about cell phone usage (cf. Example 1).

We consider two dimensions, `Location` and `Time`, and we measure the numerical facts `Incoming` and `Outgoing` (calls) (cf. Figure 1c). The `Time` dimension has a simple linear hierarchy: `Date` $\nearrow$ `Month`, `Month` $\nearrow$ `Year`, and `Year` $\nearrow$ `All`. The hierarchy (lattice) for the `Location` dimension is given in Figure 1a.

The relational representations of the `Location` dimension in the star, snowflake and path schemas that we implemented can be found in Figures 3, 4 and 2, respectively. The corresponding relational representations of the `Time` dimension are simple, and are done as described in Sections 3.1, 3.2 and 4, respectively.

We implemented a data generator in Java, to load a sizeable amount of test data into star, snowflake, and path databases. Starting with initial random data, the program generates elements for each category in the dimension schema, but taking into account the hierarchy levels. That is, the lower the category level the bigger the set of generated elements. For the base category, the number of generated elements was up to 100,000. For each of the relational implementations, we tested several queries. However, due to space limitations, we will discuss in detail only one of them. The final results of the experiment, including all queries, are shown at the end of this section.

We focus on the aggregate query, $\mathcal{Q}$, asking for the total `Incoming` calls made from each region (in `Region`) in year `2010`. This query takes 3 different forms in SQL depending on the underlying relational schema. We show them in Table 1.

As we can see, for the star schema, we need to join the fact table (called `Traffic-Fact-Table`) in Figure 1c, and the tables for the `Location` dimension ($\mathtt{R}^{\mathtt{Loc}}$) in Figure 3, 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 our example). A category might belong to more than one

(a) Star schema:

```
SELECT R^Loc.A^Region, SUM(F.In) FROM
   Traffic-Fact-Table F, R^Loc, R^Time WHERE
   R^Time.A^Year = 2010 GROUP BY R^Loc.A^Region;
```

(b) Path schema:

```
SELECT RP^Loc.A^Region, SUM(F.In)
   FROM Traffic-Fact-Table F, RP^Time,
   ((SELECT A^Number, A^Region FROM RP_1^Loc ) UNION
   (SELECT A^Number, A^Region FROM RP_2^Loc )) as RP^Loc
   WHERE RP^Time.A^Year = 2010 GROUP BY RP^Loc.A^Region;
```

(c) Snowflake schema:

```
SELECT R^Loc.A^Region, SUM(F.In) FROM
   Traffic-Fact-Table F, R^Day, R^Month, R^Year,
   ((SELECT A^Number, A^Region FROM R^Number, R^AreaCode,
   R^Region ) UNION (SELECT A^Number, A^Region FROM
   R^Number, R^City, R^Region ) ) as R^Loc
   WHERE R^Year.A^Year = 2010 GROUP BY R^Loc.A^Region;
```

Table 1: Query for star, path and snowflake schemas, resp.

B2A path, and hence, we might have to search more than one path table. The SQL version of our query is shown in Table 1(b). 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 the union of sets of tuples that belong to either path tables, and on the basis of the selected attributes $A^{\mathtt{Number}}$ and $A^{\mathtt{Region}}$. The result of this query (the temporary table $RP^{\mathtt{Loc}}$) is eventually joined to the fact table (`Traffic-Fact-Table`), and the `Time` dimension table ($RP^{\mathtt{Time}}$). Notice that, due to the linear structure of the `Time` dimension, its corresponding path table is similar to the one used for the star schema.

A comparison of queries (a) and (b) reveals that the latter requires an inner query for retrieving the specified roll-up relation $\mathcal{R}^{\mathtt{Region}}_{\mathtt{Number}}$, while this complication is avoided in the former. For this reason, query (a) is expected to execute faster than query (b).

As discussed in Section 3.2, and also widely known, the hierarchical structure of the snowflake schema requires several joins, which are costly operations, and can be used as an indicator of query performance. The SQL version of our query over the snowflake schema is shown in Table 1(c). In this case, in order to compute the roll-up $\mathcal{R}^{\mathtt{Region}}_{\mathtt{Number}}$, we have to traverse both of the paths that lead us from `Number` to `Region` in the dimension schema. Each path is traversed via a series of join operations, and the results of each path are merged as the temporary table $R^{\mathtt{Loc}}$. Due to the linear structure of the `Time` dimension, the computation of the roll-up $\mathcal{R}^{\mathtt{Year}}_{\mathtt{Day}}$ is more straightforward.

It is clear that the number of joins in query (c) is considerably higher than the number of joins in queries (a) or (b). As a consequence, we expect a noticeable difference in the execution time of query (c) in comparison to queries (a) or (b).

We executed each of the above SQL queries on their corresponding relational schemas. The response times revealed are 155.2 ms for query (a), 193.8 ms for query (b), and 427.3 ms for query (c). These findings are in correspondence to our expectations based on the discussions above on the query structure, including the effect of join operations.

As mentioned before, here we have limited our discussion to a single aggregate query, $\mathcal{Q}$. It involves the computation of the roll-up relations $\mathcal{R}^{\mathtt{Year}}_{\mathtt{Day}}$ and $\mathcal{R}^{\mathtt{Region}}_{\mathtt{Number}}$. In order to

have a more general view of query answering performance under the path schema, we considered other queries similar to $\mathcal{Q}$, but with other categories from the `Location` dimension in the aggregate query. More specifically, we executed queries that retrieve the sum of `Incoming` calls made in year `2010` from each cell phone number, area code, city, and finally, from all of the numbers. (Since the `Time` dimension is linear, queries involving its categories are simpler.) The results of our experiments are shown in Figure 12. The SQL queries for each of the aforementioned cases can be found in Appendix A. Our query $\mathcal{Q}$ corresponds to the bars shown for the `Region` category in it.

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 the graph. Here, the negligible difference between the query answering time under these two schemas is due to different dimension table schemas and number of tuples belonging to each of these table.

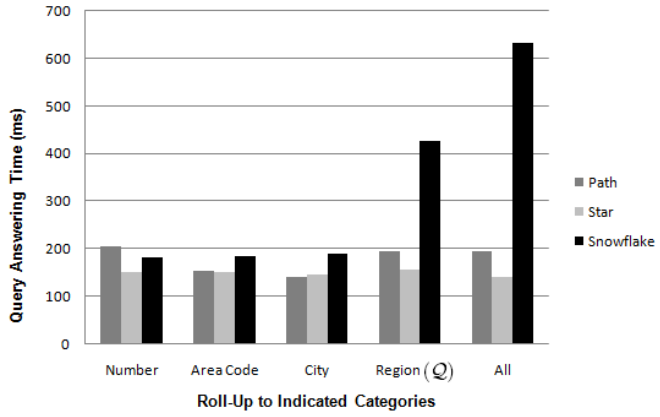



Figure 12: Comparison of query answering for path, star and snowflake schemas

Another interesting phenomenon that can be observed from the graph is that, as the level of the category used 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 an expected feature of the snowflake hierarchical structure, which requires more number of joins for rolling-up to higher levels of the hierarchy. However, for the star and path schemas, the number of joins in a query is independent from the level of the category used in the aggregate query.

In general, 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. Our experiments suggest that the path relational schema exhibits promising properties in terms of query answering time when compared with the most common relational approaches to MD modeling and implementation, namely the star and snowflake schemas.

In this section we have not reported on experiments related to MD repairs as implemented in the path relational schema. This is left for an extended version of this paper.

## 9.2 Inconsistency detection

Relational databases are subject to a repair process when they fail to satisfy certain integrity constraints (ICs). In one way or another, the violations of the ICs have to be captured and represented. On this basis repairs can be computed. As a consequence, an important property of a relational schema is its efficiency in capturing those violations.

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. In this section, we will report on the times it takes to check strictness and homogeneity of the `Location` dimension in each of the star, snowflake and path instances.

We used the data generator program to randomly generate 70,000 p-instances for the `Location` dimension instance, i.e. an MD instance. Around 600,000 cases of non-strictness were created with the generated data; and 20,000 cases of heterogeneity, expressed by the occurrence of `NULL` in one of the p-instance elements. Recall from Section 3 that the star schema is not expressive enough to capture all kinds of the heterogeneity. In the first phase of our experiment, with the given data, only half of the heterogeneity cases were detected under the star schema, by using using `NOT NULL` constraints. Since the other two schemas do support the detection of all kinds of heterogeneity, they were competing in handicapped terms with the star schema (more on this issue below).

We generated a considerable proportion of cases of non-strictness, to clearly observe the difference between evaluating FDs and EGDs under the star or path schema, and observing the impact of the several join operations for this task under the snowflake schema. In the case of homogeneity, the kinds of ICs used for the star, snowflake and path schemas are all 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.

Sections 3.1, 3.2 and 4 described how to represent summarizability conditions with ICs over the star, snowflake and path databases, respectively. We defined a stored procedure for each of the three relational instances corresponding to the original MD instance. Each procedure checks the aforementioned ICs using SQL queries. More specifically, for each of the ICs, we wrote an SQL query which returns the set of tuples violating the IC. The stored procedures can be found in Appendix B.

The running times for non-strictness detection for the star, snowflake and path databases are 17.007 sec, 1200 sec, and 15.686 sec, respectively. These results are in line with our discussions, in Sections 3.1, 3.2 and 5, around the efficiency of checking strictness in these schemas . The considerably difference in execution time for snowflake in comparison with the star and path schemas is due to the explicit hierarchical structure of the snowflake database, which complicates strictness checking. Unlike simple constraints for the star and path databases, for the snowflake it is necessary to perform a series of join operations between different tables. The negligible difference between the execution times of star and path database is due to different dimension table schemas

and the number of tuples belonging to each of those tables. Notice that the additional EGDs used with the path schema for detecting non-strictness in more than one path tables do not considerably effect the performance of inconsistency detection.

The time measured for detecting heterogeneity was 510 ms for the star schema, 866 ms for the path schema, and 686 ms for the snowflake schema. The difference between the first number and the last two is, as discussed above, due to the fact that the queries executed over the star schema capture only half of the heterogeneity cases. 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.

As also mentioned above, these results are somehow unfair, in the sense that they do not reflect the weakness of the star schema for capturing heterogeneity. For this reason, we decided to test the performances of the three schemas when the same number of heterogeneity cases were present in the corresponding databases. Hence, in the second phase of our experiments, we narrowed down the heterogeneity instances of our data set to those that are detectable in a star database. More specifically, we generated 20,000 instances of heterogeneity, which could all be captured using `NOT NULL` constraints on the star schema. In this case, we were interested in checking only heterogeneity, the controversial point, as we just mentioned, expecting to obtain more realistic results. The time measured for detecting heterogeneity was 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. Notice that the difference in execution times between the path schema and the star or snowflake schemas comes from the fact that for the former we need more `NOT NULL` constraints than for the star or snowflake schemas. However, the impact on this issue on performance is not particularly high.

Our experiments support our claims that the path schema is a relational representation of MD databases that we can use for *completely* and *efficiently* checking summarizability constraints through ICs.

# 10. CONCLUSIONS

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

1. We have proposed and analyzed a new relational representation for multidimensional databases (MDDBs), and data warehouses in particular. This so-called *path relational schema* (PRS) has also been compared with the traditional relational schemas for ROLAP, the star and the snowflake schemas.

   The PRS, enriched with natural relational ICs, has some advantages that make it particularly suitable for handling inconsistency issues in MDDBs in relation to MD summarizability constraints. More concretely, it allows for: (a) a natural relational representation of those semantic requirements, (b) the possibility of easily checking their satisfaction, and (c) a simple specification and application of consistency repair policies. We also verified that it offers a reasonable performance in terms of aggregate query answering.

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

   Proceeding in this way has two main advantages: (a) Much work has been done around relational repairs and consistent query answering, and we could take advantage of it. (b) The repair process could be implemented directly on relational platforms.

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

There are many interesting research directions that are opened by this proposal. Some of them are fully open, and others are subject of ongoing research. We mention some of them.

1. In this paper we haven't considered consistent query answering for aggregate queries. We could naturally explore the *range semantics* [3, 25]. The main problem is about doing CQA under summarizability constraints without explicitly computing all repairs.

   In this direction, we should investigate the existence of a corresponding notion of *canonical instance* introduced in [9], that was used to do and approximate consistent answers (under their repair semantics).

2. Still in the direction of 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].

3. We have the impression that the relational route to MD repairs can be particularly useful for doing MD database repairing and CQA under updates. Except for preliminary work in [39] on dynamic (relational) CQA, very little has been done in this important direction.

4. In the MD scenario, considering, in addition to summarizability constraints, also *aggregation constraints* is quite natural. At this place existing results on attribute-based repairs under aggregation constraints [23] would be a very nice fit, and also very useful.

5. 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* [6] and *schema mappings* [4] (from one relational MD schema into another, like the path schema), and also *schema evolution* [21].

6. In this paper we have reported on some experiments with the proposed relational schema in the direction of aggregate query answering. Currently, we are also

running experiments in relation to the repair aspects of this relational approach.

On the repair computation side, we have two alternative ways to go, and comparing them would be interesting. One of them consists in using "answer set programs" (ASPs), similar to those in [13, 19]. They work directly with the MD representation. In our case, given the different MD repair semantics (the one in Theorem 2), our program would include "weak constraints with weights", that extend the ASP paradigm [14]. They are used to minimize numbers of violations of program constraints. For that reason, they can be used to capture our *numerical* MD distance. (The distance in [13, 19] is set-theoretical, not numerical).

The other way to go consists in repairing directly the relational instances obtained via the MD2R mapping wrt relational ICs. ASP have been also successfully used to compute relational repairs and do consistent query answering (cf. [18] and references therein). In this case, the ASP would also require the use of weak constraints, since we would be minimizing the number of changes of attribute values. ASP of this kind have been used in [24].

7. A deeper investigation of the relationship between our MD repairs and other approaches to the same or similar problem is still necessary. In particular, it would be interesting to compare our data-based approach to repairs to MD repairs that are based on changes in the schema [5]. The combination of both approaches should also be investigated.

8. 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. (This idea has been explored with schema-based repairs [5] using MDX views). This view could be seen as a *logical layer* [16] 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. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS99)*, pages 68–79, 1999.

[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. *ACM SIGMOD Record*, 38:17–28, 2010.

[5] S. Ariyan and L. Bertossi. Structural Repairs of Multidimensional Databases. In *Proc. Alberto Mendelzon International WS of Foundations of Data Management (AMW'11)*, volume 749 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2011.

[6] P. Barcelo. Logical Foundations of Relational Data Exchange. *SIGMOD Record*, 2009, 38(1):49-58.

[7] L. Bertossi. Consistent Query Answering in Databases. *ACM SIGMOD Record*, 35:68–76, 2006.

[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. In *Proceedings of the 3rd Alberto Mendelzon International Workshop on Foundations of Data Management (AMW'09*, volume 450 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2009.

[10] L. Bertossi, L. Bravo, E. Franconi, and A. Lopatenko. The Complexity and Approximation of Fixing Numerical Attributes in Databases under Integrity Constraints. *Information Systems*, 33(4-5):407–434, 2008.

[11] Ph. Bohannon, M. Flaster, W. Fan, and R. Rastogi. A Cost-Based Model and Effective Heuristic for Repairing Constraints by Value Modification. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 143–154. ACM, 2005.

[12] L. Bravo and L. Bertossi. Semantically Correct Query Answers in the Presence of Null Values. In *Proceedings of the EDBT 2006 Workshops*, volume 4254 of *Lecture Notes in Computer Science*, pages 336–357. Springer, 2006.

[13] L. Bravo, M. Caniupan, and C. A. Hurtado. Logic Programs for Repairing Inconsistent Dimensions in Data Warehouses. In *Proceedings of the 4th Alberto Mendelzon International Workshop on Foundations of Data Management (AMW10)*, volume 619 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2010.

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

[15] L. Cabibbo and R. Torlone. Querying Multidimensional Databases. In *Proceedings of the 6th International Workshop on Database Programming Languages (DBLP1997)*, volume 1369 of *Lecture Notes in Computer Science*, pages 319–335. Springer, 1997.

[16] L. Cabibbo and R. Torlone. The Design and Development of a Logical System for OLAP. In *Proceedings of the Second International Conference on Data Warehousing and Knowledge Discovery (DaWaK2000)*, volume 1874 of *Lecture Notes in Computer Science*, pages 1–10. Springer, 2000.

[17] M. Caniupan. Handling Inconsistencies in Data Warehouses. In *Current Trends in Database Technology*, Springer LNCS 3268, pages 166–176, 2004.

[18] 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.

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

[20] J. Chomicki. Consistent Query Answering: Five Easy Pieces. In *Proceedings of the International Conference on Database Theory (ICDT07)*, volume 4353 of

*Lecture Notes in Computer Science*, pages 1–17. Springer, 2007.

[21] 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.

[22] R. Fagin. Inverting Schema Mappings. In *ACM Transactions on Database Systems (TODS)*, volume 32, 2007.

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

[24] E. Franconi, A. Laureti-Palma, N. Leone, S. Perri, and F. Scarcello. Census Data Repair: A Challenging Application of Disjunctive Logic Programming. In *Proceedings of the 8th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, Lecture Notes in Computer Science, pages 561–578. Springer, 2001.

[25] A. Fuxman, E. Fazli, and R. Miller. Conquer: Efficient Management of Inconsistent Databases. In *Proc. SIGMOD05*, pp. 155-166.

[26] C. A. Hurtado. *Structurally Heterogeneous OLAP Dimensions*. PhD thesis, University of Toronto, CA, 2002.

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

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

[29] C. A. Hurtado and A. Mendelzon. Reasoning about Summarizability in Heterogeneous Multidimensional Schemas. In *Proceedings of the International Conference on Database Theory (ICDT01)*, volume 1973 of *Lecture Notes in Computer Science*, pages 375–389. Springer, 2001.

[30] C. A. Hurtado and A. Mendelzon. OLAP Dimension Constraints. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS02)*, pages 169–179, 2002.

[31] C. A. Hurtado, A. Mendelzon, and A. Vaisman. Maintaining Data Cubes under Dimension Updates. In *Proceedings of the 15th International Conference on Data Engineering (ICDE99)*, pages 346–355. IEEE Computer Society, 1999.

[32] C. A. Hurtado, A. Mendelzon, and A. Vaisman. Updating OLAP Dimensions. In *Proceedings of the ACM 2nd International Workshop on Data Warehousing and OLAP (DOLAP)*, pages 60–66. ACM, 1999.

[33] W. H. Inmon. *Building the Data Warehouse*. Wiley, 4 edition, September 2005.

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

[35] R. Kimball and M. Ross. *The Data Warehouse Toolkit: the Complete Guide to Dimensional Modeling*. Wiley, 2 edition, April 2002.

[36] W. Lehner, J. Albrecht, and H. Wedekind. Normal Forms for Multidimensional Databases. In *Proceedings of the 10th International Conference on Scientific and Statistical Database Management*, pages 63–72. IEEE Computer Society, 1998.

[37] H. Lenz and A. Shoshani. Summarizability in OLAP and Statistical Data Bases. In *Proceedings of the 9th International Conference on Scientific and Statistical Database Management*, pages 132–143. IEEE Computer Society, 1997.

[38] M. Levene and G. Loizou. Why is the Snowflake Schema a Good Data Warehouse Design? *Inf. Syst.*, 28:225–240, 2003.

[39] A. Lopatenko and L. Bertossi. Complexity of Consistent Query Answering in Databases under Cardinality-Based and Incremental Repair Semantics. In *Proceedings of the International Conference on Database Theory (ICDT07)*, volume 4353, pages 179–193. Springer, 2007.

[40] P. E. O'Neil and G. Graefe. Multi-Table Joins through Bitmapped Join Indices. *SIGMOD Record*, 24(3):8–11, 1995.

[41] T. B. Pedersen and C. S. Jensen. Multidimensional Data Modeling for Complex Data. In *Proceedings of the International Conference on Data Engineering (ICDE99)*, pages 336–345. IEEE Computer Society, 1999.

[42] T. B. Pedersen, C. S. Jensen, and C. E. Dyreson. Extending Practical Pre-Aggregation in On-Line Analytical Processing. In *Proceedings of the International Conference on Very Large Databases (VLDB99)*, pages 663–674. Morgan Kaufmann, 1999.

[43] M. Rafanelli and A. Shoshani. STORM: A Statistical Object Representation Model. In *Proceedings of the 5th International Conference on Statistical and Scientific Database Management*, volume 420 of *Lecture Notes in Computer Science*, pages 14–29. Springer, 1990.

[44] T. Shimura, M. Yoshikawa, and S. Uemura. Storage and Retrieval of XML Documents Using Object-Relational Databases. In *Proceedings of the 10th International Conference on Database and Expert Systems Applications (DEXA99)*, volume 1677 of *Lecture Notes in Computer Science*, pages 206–217. Springer, 1999.

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

[46] J. Wijsen. Database Repairing Using Updates. *ACM Transactions on Database Systems (TODS)*, 30(3):722–768, 2005.

[47] 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.*, 1(1):110–141, 2001.

# APPENDIX

## A. QUERIES USED IN THE EXPERIMENT

$Q_1$: Return the sum of `Incoming` calls made from each cell phone `Number` in year 2010.

(a) Star schema:

```
SELECT R^Loc.A^Number, SUM(F.In) FROM
  Traffic-Fact-Table F, R^Loc, R^Time
  WHERE R^Time.A^Year = 2010 GROUP BY R^Loc.A^Number;
```

(b) Path schema[a]:

```
SELECT RP_1^Loc.A^Number, SUM(F.In) FROM
  Traffic-Fact-Table F, RP^Time, RP_1^Loc
  WHERE RP^Time.A^Year = 2010
  GROUP BY RP_1^Loc.A^Number;
```

(c) Snowflake schema:

```
SELECT R^Number.A^Number, SUM(F.In) FROM
  Traffic-Fact-Table F, R^Day, R^Month, R^Year, R^Number
  WHERE R^Year.A^Year = 2010
  GROUP BY R^Number.A^Number;
```

[a]For path schema, the values for attribute $RP_1^{Loc}.A^{Number}$ is equal to $RP_2^{Loc}.A^{Number}$. So in this query, we can either choose $RP_1^{Loc}$ or $RP_2^{Loc}$.

$Q_2$: Return the sum of `Incoming` calls made from each cell phone `AreaCode` in year 2010.

(a) Star schema:

```
SELECT R^Loc.A^AreaCode, SUM(F.In) FROM
  Traffic-Fact-Table F, R^Loc, R^Time
  WHERE R^Time.A^Year = 2010 GROUP BY R^Loc.A^AreaCode;
```

(b) Path schema:

```
SELECT RP_1^Loc.A^AreaCode, SUM(F.In) FROM
  Traffic-Fact-Table F, RP^Time, RP_1^Loc
  WHERE RP^Time.A^Year = 2010
  GROUP BY RP_1^Loc.A^AreaCode;
```

(c) Snowflake schema:

```
SELECT R^AreaCode.A^AreaCode, SUM(F.In) FROM
  Traffic-Fact-Table F, R^Day, R^Month, R^Year,
  R^Number, R^AreaCode
  WHERE R^Year.A^Year = 2010
  GROUP BY R^AreaCode.A^AreaCode;
```

$Q_3$: Return the sum of `Incoming` calls made from each cell phone `City` in year 2010.

(a) Star schema:

```
SELECT R^Loc.A^City, SUM(F.In) FROM
  Traffic-Fact-Table F, R^Loc, R^Time
  WHERE R^Time.A^Year = 2010 GROUP BY R^Loc.A^City;
```

(b) Path schema:

```
SELECT RP_2^Loc.A^City, SUM(F.In) FROM
  Traffic-Fact-Table F, RP^Time, RP_2^Loc
  WHERE RP^Time.A^Year = 2010
  GROUP BY RP_2^Loc.A^City;
```

(c) Snowflake schema:

```
SELECT R^City.A^City, SUM(F.In) FROM
  Traffic-Fact-Table F, R^Day, R^Month, R^Year,
  R^Number, R^City
  WHERE R^Year.A^Year = 2010
  GROUP BY R^City.A^City;
```

$Q_4$: Return the sum of `Incoming` calls made from each cell phone `Region` in year 2010.

(a) Star schema:

```
SELECT R^Loc.A^Region, SUM(F.In) FROM
  Traffic-Fact-Table F, R^Loc, R^Time WHERE
  R^Time.A^Year = 2010 GROUP BY R^Loc.A^Region;
```

(b) Path schema:

```
SELECT RP^Loc.A^Region, SUM(F.In)
  FROM Traffic-Fact-Table F, RP^Time,
  ((SELECT A^Number, A^Region FROM RP_1^Loc ) UNION
  (SELECT A^Number, A^Region FROM RP_2^Loc )) as RP^Loc
  WHERE RP^Time.A^Year = 2010 GROUP BY RP^Loc.A^Region;
```

(c) Snowflake schema:

```
SELECT R^Loc.A^Region, SUM(F.In) FROM
  Traffic-Fact-Table F, R^Day, R^Month, R^Year,
  ((SELECT A^Number, A^Region FROM R^Number, R^AreaCode,
  R^Region ) UNION (SELECT A^Number, A^Region FROM
  R^Number, R^City, R^Region ) ) as R^Loc
  WHERE R^Year.A^Year = 2010 GROUP BY R^Loc.A^Region;
```

$Q_5$: Return the sum of `Incoming` calls made from `All` cell phone numbers in year 2010.

(a) Star schema:

```
SELECT R^Loc.A^All, SUM(F.In) FROM
  Traffic-Fact-Table F, R^Loc, R^Time
  WHERE R^Time.A^Year = 2010 GROUP BY R^Loc.A^All;
```

(b) Path schema:

```
SELECT RP^Loc.A^All, SUM(F.In)
  FROM Traffic-Fact-Table F, RP^Time,
  ((SELECT A^Number, A^All FROM RP_1^Loc ) UNION
  (SELECT A^Number, A^All FROM RP_2^Loc )) as RP^Loc
  WHERE RP^Time.A^Year = 2010 GROUP BY RP^Loc.A^All;
```

(c) Snowflake schema:

```
SELECT R^Loc.A^All, SUM(F.In) FROM
  Traffic-Fact-Table F, R^Day, R^Month, R^Year,
  ((SELECT A^Number, A^All FROM R^Number, R^AreaCode,
  R^Region, R^All ) UNION (SELECT A^Number, A^All FROM
  R^Number, R^City, R^Region, R^All ) ) as R^Loc
  WHERE R^Year.A^Year = 2010 GROUP BY R^Loc.A^All;
```

## B. STORED PROCEDURES USED IN THE EXPERIMENT

```
CREATE PROCEDURE [dbo].[checkStarSchema]
AS
BEGIN
-- num -> areacode
select * from dbo.LocationStar as L1 inner join dbo.LocationStar as L2
on L1.Number = L2.Number where L1.AreaCode <> L2.AreaCode

-- num -> city
select * from dbo.LocationStar as L1 inner join dbo.LocationStar as L2
on L1.Number = L2.Number where L1.City <> L2.City

-- areacode->region
select * from dbo.LocationStar as L1 inner join dbo.LocationStar as L2
on L1.AreaCode = L2.AreaCode where L1.Region <> L2.Region

-- city->region
select * from dbo.LocationStar as L1 inner join dbo.LocationStar as L2
on L1.City = L2.City where L1.Region <> L2.Region

-- num->region
select * from dbo.LocationStar as L1 inner join dbo.LocationStar as L2
on L1.Number = L2.Number where L1.Region <> L2.Region

-- areacode not null
select * from dbo.LocationStar where AreaCode Is NULL

-- city not null
select * from dbo.LocationStar where City Is NULL

-- region not null
select * from dbo.LocationStar where Region Is NULL

-- all not null
select * from dbo.LocationStar where AllCol Is NULL
END
```

```
CREATE PROCEDURE [dbo].[checkSnowflakeSchema]
AS
BEGIN
-- num -> city
select * from
(select N.Number,C.City from dbo.NumberSnowflake as N inner join dbo.CitySnowflake as C
on N.City = C.City ) as t1 ,
(select N.Number,C.City from dbo.NumberSnowflake as N inner join dbo.CitySnowflake as C
on N.City = C.City ) as t2
where t1.Number = t2.Number and t1.City <> t2.City

-- num -> areacode
select * from
(select N.Number,A.AreaCode from dbo.NumberSnowflake as N inner join dbo.AreaCodeSnowflake as A
on N.AreaCode = A.AreaCode ) as t1 ,
(select N.Number,A.AreaCode from dbo.NumberSnowflake as N inner join dbo.AreaCodeSnowflake as A
on N.AreaCode = A.AreaCode ) as t2
where t1.Number = t2.Number and t1.AreaCode <> t2.AreaCode

-- num -> region
select * from
((select N.Number,R.Region from dbo.NumberSnowflake as N inner join dbo.AreaCodeSnowflake as A
    on N.AreaCode = A.AreaCode inner join dbo.RegionSnowflake as R on A.Region = R.Region) union
(select N.Number,R.Region from dbo.NumberSnowflake as N inner join dbo.CitySnowflake as C
    on N.City = C.City inner join dbo.RegionSnowflake as R on C.Region = R.Region)) as t1,
((select N.Number,R.Region from dbo.NumberSnowflake as N inner join dbo.AreaCodeSnowflake as A
    on N.AreaCode = A.AreaCode inner join dbo.RegionSnowflake as R on A.Region = R.Region) union
(select N.Number,R.Region from dbo.NumberSnowflake as N inner join dbo.CitySnowflake as C
    on N.City = C.City inner join dbo.RegionSnowflake as R on C.Region = R.Region)) as t2
where t1.Number = t2.Number and t1.Region <> t2.Region

-- city -> region
select * from
(select C.City,R.Region from dbo.CitySnowflake as C inner join dbo.RegionSnowflake as R
on C.Region = R.Region) as t1 ,
(select C.City,R.Region from dbo.CitySnowflake as C inner join dbo.RegionSnowflake as R
on C.Region = R.Region) as t2
where t1.City = t2.City and t1.Region <> t2.Region

-- areacode -> region
select * from
(select A.AreaCode,R.Region from dbo.AreaCodeSnowflake as A inner join dbo.RegionSnowflake as R
on A.Region = R.Region) as t1 ,
(select A.AreaCode,R.Region from dbo.AreaCodeSnowflake as A inner join dbo.RegionSnowflake as R
on A.Region = R.Region) as t2
where t1.AreaCode = t2.AreaCode and t1.Region <> t2.Region

-- city not null
select * from dbo.NumberSnowflake where City Is NULL

-- areacode not null
select * from dbo.NumberSnowflake where AreaCode Is NULL

-- region not null
select * from dbo.AreaCodeSnowflake where Region Is NULL
select * from dbo.CitySnowflake where Region Is NULL

-- all not null
select * from dbo.RegionSnowflake where AllCol Is NULL
END
```

```
CREATE PROCEDURE [dbo].[checkPathSchema]
AS
BEGIN
-- num->areacode
select * from dbo.LocationPath1 as P1 inner join dbo.LocationPath1 as P2
on P1.Number = P2.Number where P1.AreaCode <> P2.AreaCode

-- areacode->region
select * from dbo.LocationPath1 as P1 inner join dbo.LocationPath1 as P2
on P1.AreaCode = P2.AreaCode where P1.Region <> P2.Region

-- numb->city
select * from dbo.LocationPath2 as P1 inner join dbo.LocationPath2 as P2
on P1.Number = P2.Number where P1.City <> P2.City

-- city->region
select * from dbo.LocationPath2 as P1 inner join dbo.LocationPath2 as P2
on P1.City = P2.City where P1.Region <> P2.Region

-- num->region
select * from dbo.LocationPath1 as P1 inner join dbo.LocationPath1 as P2
on P1.Number = P2.Number where P1.Region <> P2.Region

select * from dbo.LocationPath2 as P1 inner join dbo.LocationPath2 as P2
on P1.Number = P2.Number where P1.Region <> P2.Region

select * from dbo.LocationPath1 as P1 inner join dbo.LocationPath2 as P2
on P1.Number = P2.Number where P1.Region <> P2.Region

-- areacode not null
select * from dbo.LocationPath1 where AreaCode Is NULL

-- city not null
select * from dbo.LocationPath2 where City Is NULL

-- region not null
select * from dbo.LocationPath1 where Region Is NULL
select * from dbo.LocationPath2 where Region Is NULL

-- all not null
select * from dbo.LocationPath1 where AllCol Is NULL
select * from dbo.LocationPath2 where AllCol Is NULL
END
```