Repair-Oriented Relational Schemas for Multidimensional Databases

by

Mahkameh Yaghmaie

A Thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfilment of
the requirements for the degree of
Master of Computer Science
in

Computer Science
Carleton University
Ottawa, Ontario, Canada
August 2011

Abstract

In data warehouses (DWs), summarizability is the key factor in query answering optimization. This property allows re-using pre-computed aggregate queries or views, to compute correct query results. In multidimensional (MD) modeling, a dimension instance is called summarizable if and only if it is strict and homogeneous. In reality, dimensions might not satisfy these two semantic conditions. In such cases, strictness and homogeneity must be restored by *repairing* the non-summarizable dimension instance.

This thesis addresses the issue of non-summarizability with a novel *relational* approach. Unlike existing direct MD repair mechanisms, we take advantage of the relational repair approaches to integrity constraint (IC) satisfaction. Our methodology consists of three phases: 1) finding an *expressive* relational model for multidimensional data models (MDMs), and translating the MD instance into a relational instance, 2) repairing the inconsistent relational database (RDB) using relational repair approaches wrt relational ICs, and 3) inverting the mapping used in the first step, to obtain a repaired MD instance from a repaired relational instance.

In this process, our biggest challenge is to find an expressive and adequate relational representation for multidimensional databases (MDDBs). To this end, we first investigate the two well-known relational implementations of MDDBs, *Star* and *Snowflake*. Our discussions show that, these schemas are not the best choice for our purpose. Hence, we propose a different relational reconstruction of the multidimensional database. It takes into account the paths belonging to the dimension schema and instance. Hence, the new relational representation of MDMs is referred to as *path schema*. Our MD repair approach takes advantage of the unique properties of this relational schema.

We show that using path schema, we can efficiently check strictness and homogeneity through relational integrity constraints, and can easily obtain minimal relational repairs. Moreover, the path-based mapping from MDDB to RDB is uniquely invertible. Besides, as a new relational representation of MDDBs, the path schema has some interesting properties, such as efficient query answering and IC checking, compared to star and snowflake schemas.

We compare the dimension repairs obtained through our approach, with existing MD repair approaches. Our relational solution towards the non-summarizability problem generates dimension repairs, that are minimal from *both* multidimensional and relational perspectives. We manage to characterize our class of repairs in pure MD terms.

Keywords: Multidimensional data model, Semantic constraints, Repairs

Acknowledgments

First and foremost, I offer my sincerest gratitude to my supervisor, Dr Leopoldo Bertossi. Working under supervision of Professor Bertossi was an invaluable experience, that allowed me to learn many aspects of science and life. He has supported me throughout my thesis, with his patience and knowledge, whilst allowing me the room to work in my own way. I attribute the level of my Masters degree to his encouragement and effort.

I am indebted to my parents, Manoochehr and Mahvash, and my sister Mahshad, whose love, support and understanding helped me to pursue my graduate studies. My final thought is always with them.

Finally, I wish to thank my friends with whom I share great memories in Ottawa, Amin Fereidooni, Sina Arian, Aida Malaki, Saba Rahimi, Zeinab Bahmani and Mostafa Khaghani.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Data warehouses (DWs) provide analytical information that can be used in decision making process. They support multidimensional (MD) analysis of data, in which multiple perspectives are used to view quantitative data. Multidimensional data models (MDMs) provide the logical foundation for this data organization and representation. Dimensions and fact tables are the basic components in this type of data modeling. In a MDM, a dimension is modeled by two hierarchies: a dimension schema and a dimension instance. A dimension schema is a hierarchy composed of a number of levels or categories, each of which represents a different level of granularity. Instantiating the dimension schema by values at each level forms the dimension instance. An instance or extension of the multidimensional data model is called a MD instance or a multidimensional database (MDDB).

DWs are basically the implementation of MDDBs along with some tools for manipulating the MD data. DWs implement/represent the MD instance either as relational systems (ROLAP), proprietary multidimensional systems (MOLAP), or a hybrid of both (HOLAP).

In ROLAP (relational online analytical processing) systems, the multidimensional data model is a relational model and it is represented as a relational database (RDB) instance. Although ROLAP uses a relational database to represent the MD instance, the database must be carefully designed for analytical processing.

Star and *Snowflake* are the two best-known relational schemas that are specifically designed for ROLAP MD databases. ROLAP maps operations on multidimensional data model to standard relational operations. An advantage of these systems is that they can be easily integrated into other relational information systems [49, 50, 70].

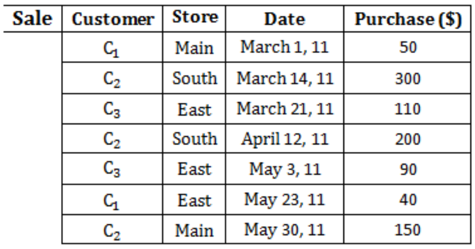
If the MD instance is represented as a relational database, it can be viewed multidimensionally, but only by successively accessing and processing the tables corresponding to different dimensions. An alternative to this relational storage, is to directly represent and manipulate the MDDB in the form of multidimensional arrays.

A MOLAP (multidimensional online analytical processing) system processes data that is already stored in a multidimensional array, in which all possible combinations of data are reflected, each in a cell that can be accessed directly. For this reason, MOLAP is, for most uses, faster and more user-responsive than ROLAP. However, in terms of storage efficiency, ROLAP systems are more preferred to MOLAP systems [49, 50, 70].

A HOLAP (hybrid online analytical processing) system is a combination of ROLAP and MOLAP. In HOLAP, part of the MD instance is stored in a MOLAP system, and another part of the MD instance in a ROLAP system. This storage strategy allows a tradeoff of the advantages of both ROLAP and MOLAP architecture [49].

Based on [50], ROLAPs are the most common implementation of the MD instance in DWs. The rich infrastructure of RDBs enables using standard, common and established techniques [70].

Example 1.1. We want to analyze product sales in a company, considering different factors, such as the customer who bought the product, the store location where the product was sold, and the purchase date. In multidimensional modeling terms, `Sales` is the fact, which is measured relative to dimensions `Customer`, `Store` and `Date`. The table in Figure 1.1 shows the purchase data based on these dimensions.



Sale	Customer	Store	Date	Purchase ($)
	C_1	Main	March 1, 11	50
	C_2	South	March 14, 11	300
	C_3	East	March 21, 11	110
	C_2	South	April 12, 11	200
	C_3	East	May 3, 11	90
	C_1	East	May 23, 11	40
	C_2	Main	May 30, 11	150

Figure 1.1: `Sales` fact table

The hierarchy of the `Store` location dimension is shown in Figure 1.2. In this hierarchy, each `Store` belongs to a `City` , which itself belongs to a `Country` . Figure 1.3 shows a similar hierarchy for the `Date` dimension.

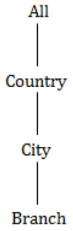


Figure 1.2: `Store` dimension schema

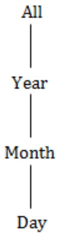
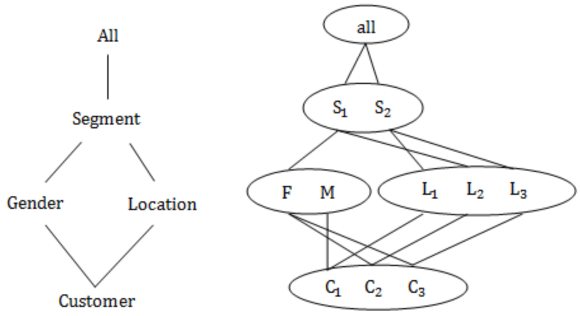


Figure 1.3: `Date` dimension schema

The `Customer` dimension has a non-linear structure. Figure 1.4a shows the hierarchy for categorizing the customer data. An instance of this categorization is brought in Figure 1.4b. Customers can be grouped based on different demographic information, such as location and gender. On this basis, customers can be divided into groups of individuals that are similar in specific ways that are relevant to marketing. This segmentation allows companies to target groups effectively, and allocate marketing resources to best effect.

In Figure 1.4a, `Customer`, `Gender`, `Location`, `Segment` and `All` are categories. The given dimension instance represents three customers, $\mathtt{C}_1$, $\mathtt{C}_2$ and $\mathtt{C}_3$. The male and female genders are denoted by `M` and `F`, respectively. Customers may reside in either of the three locations, $\mathtt{L}_1$, $\mathtt{L}_2$ or $\mathtt{L}_3$. For marketing purposes, and based on the gender



(a) Dimension schema (b) Dimension instance

Figure 1.4: `Customer` dimension

and location, each customer is grouped into either of the segments $\texttt{S}_1$ or $\texttt{S}_2$.

Parent-child relationships are denoted by edges in the dimension schema, and the dimension instance. It can be seen in Figure 1.4a that, `Segment` is the *parent* category for categories `Gender` and `Location`; and an *ancestor* of category `Customer`. A similar parent-child relationship can be found in the dimension instance of Figure 1.4b. For instance, element $\texttt{C}_2$ of category `Customer` has $\texttt{S}_1$ as an ancestor element in the `Segment` category, and `F` and $\texttt{L}_2$ as parents from categories `Gender` and `Location`, respectively. □

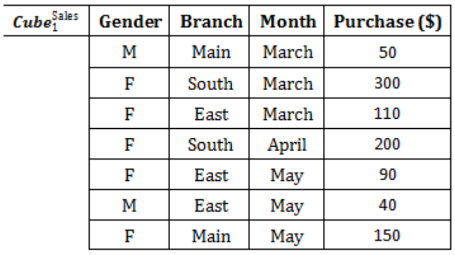
In ROLAPs, the fact table stores data associated to the elements of lowest category of each dimension hierarchy. It can be seen in Figure 1.1 that, the `Sales` data is stored based on the `Customer`, `Branch` and `Day` categories, which are all the bottommost category of their dimension schemas. This linkage between the fact and dimensions in ROLAPs enables the computation of fact data also at higher granularity levels of the dimension hierarchies.

Cube views are simple aggregate queries used as basic component of an MD query. They compute the fact data relative to a given granularity level in a dimension hierarchy. Due to large volumes of data in MDDBs, aggregate query results should not be computed from scratch. A key strategy for speeding up cube view processing is to reuse pre-computed cube views. To this end, a cube view should be rewritten as another query, that refers to pre-computed cube views for lower levels of the hierarchy. A dimension instance that correctly supports this kind of accumulative query computation is called *summarizable*.

The notion of summarizability was first introduced by [64], in the context of statistical databases. A multidimensional database is summarizable if all of its dimensions allow summarization. Non-summarizability results in incorrect query results because of the wrong use of pre-computed views; or in inefficiency due to computation of the query answers from scratch [64, 53, 52, 61, 43].

Example 1.2. (example 1.1 continued) The table in Figure 1.1 stores the fact data, `Sales`, based on the base category of `Customer`, `Store` and `Date` hierarchies (Figures 1.4a, 1.2 and 1.3). Hence, this table is referred to as the *base cube* for our example, $BaseCube^{\texttt{Sales}}(\texttt{Customer}, \texttt{Branch}, \texttt{Day})$.

An example of a non-base cube view would be the case where we want to measure the product `Sales` categorized by customer `Gender`, for each `Month`, and `Branch`, i.e. $Cube_1^{\texttt{Sales}}(\texttt{Gender}, \texttt{Branch}, \texttt{Month})$. Notice that categories `Gender` and `Month` are the parent categories of `Customer` and `Day`, respectively. Hence, this cube can be easily computed using $BaseCube^{\texttt{Sales}}$ as shown in Figure 1.5.

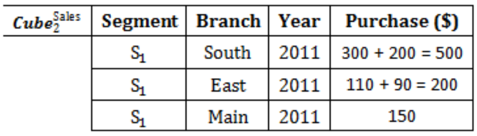


Cube $_1^{Sales}$	Gender	Branch	Month	Purchase ($)
	M	Main	March	50
	F	South	March	300
	F	East	March	110
	F	South	April	200
	F	East	May	90
	M	East	May	40
	F	Main	May	150

Figure 1.5: The cube view $Cube_1^{\texttt{Sales}}(\texttt{Gender}, \texttt{Branch}, \texttt{Month})$

Assume we want to compute the profitability of each customer segment by measuring the total purchases made by the customers in the segment in each `Year`, and

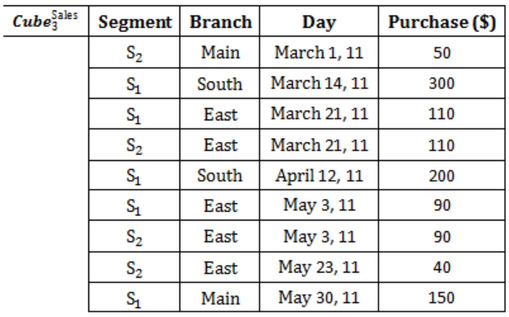
`Branch`, i.e. $Cube_2^{\texttt{Sales}}(\texttt{Segment}, \texttt{Branch}, \texttt{Year})$. Notice that both categories `Segment` and `Year` are parent categories of `Gender` and `Month` , respectively. Consider using the pre-computed cube view $Cube_1^{\texttt{Sales}}$ for computing this cube view. Since gender `M` is not related to any customer segment (see Figure 1.4b), the purchase data for male customers will not be included in the final result. In other words, $Cube_2^{\texttt{Sales}}$ will only contain the purchase data of female customers as shown in Figure 1.6. This is an example of missing data for cube view computation.



$Cube_2^{Sales}$	Segment	Branch	Year	Purchase ($)
	S_1	South	2011	300 + 200 = 500
	S_1	East	2011	110 + 90 = 200
	S_1	Main	2011	150

Figure 1.6: The cube view $Cube_2^{\texttt{Sales}}(\texttt{Segment}, \texttt{Branch}, \texttt{Year})$

Now consider cube view $Cube_3^{\texttt{Sales}}(\texttt{Segment}, \texttt{Branch}, \texttt{Day})$, which we would like to compute based on the base cube $BaseCube^{\texttt{Sales}}$. Figure 1.7 represents this cube view. Since, customer $\texttt{C}_3$ is related to two customer segments (see Figure 1.4b), the purchase data for this customer will be considered twice in computing $Cube_3^{\texttt{Sales}}$. This is an example of double counting in cube view computation. □



$Cube_3^{Sales}$	Segment	Branch	Day	Purchase ($)
	S_2	Main	March 1, 11	50
	S_1	South	March 14, 11	300
	S_1	East	March 21, 11	110
	S_2	East	March 21, 11	110
	S_1	South	April 12, 11	200
	S_1	East	May 3, 11	90
	S_2	East	May 3, 11	90
	S_2	East	May 23, 11	40
	S_1	Main	May 30, 11	150

Figure 1.7: The cube view $Cube_3^{\texttt{Sales}}(\texttt{Segment}, \texttt{Branch}, \texttt{Day})$

A summarizable dimension must satisfy two conditions. The first one, *strictness*, says that each element in a category should have at most one parent in each upper category [43, 61, 64]. The second condition, *homogeneity*, requires each element in a category to have at least one parent element in each parent category [43, 50, 64]. We refer to these two semantic conditions as *summarizability constraints*.

Example 1.3. (examples 1.1 and 1.2 continued) In the `Customer` dimension, strictness is violated, since $\mathtt{C}_3$ has two ancestors, $\mathtt{S}_1$ and $\mathtt{S}_2$, in the `Segment` category. In other words, this customer belongs to two different segments at the same time, which represents data inconsistency in the MD instance. That is why we had the issue of double counting in computing $\mathit{Cube}_3^{\mathtt{Sales}}$ in Example 1.2.

Moreover, the `Customer` dimension is non-homogeneous, i.e. heterogeneous, since element `M` has no parent in category `Segment`. In fact, this hierarchy does not categorize male customers into any segment, which suggests that some data is missing. The missing data in computing $\mathit{Cube}_2^{\mathtt{Sales}}$ in Example 1.2 results from this fact. □

It is common to have dimension instances that are non-summarizable due to their specific design [43]. In addition, a dimension instance may violate summarizability constraints after some dimension updates [46]. In particular, non-strict and heterogeneous dimensions are common in MDDBs. In this case, the MDDB is said to be *inconsistent*. Hence, for the sake of correct and efficient query answering, non-summarizability must be resolved or restored.

In relational databases, *forcing* integrity constraints (ICs) is an approach for avoiding data inconsistencies. By keeping the ICs satisfied all the time, the database instance will never become inconsistent. On the other hand, database consistency can be restored through what is usually called a *database repair* process [4]. Data inconsistencies are detected through IC *checking* over the given relational instance. In relational databases, a repair for database instance D, which is inconsistent wrt a set of integrity constraints ICs, is a new instance D' with the same schema, which satisfies the ICs, and minimally departs from D (see [12, 26] for surveys).

Similarly, in MDDBs, a few approaches have been proposed for keeping the MD instance summarizable all the time. In [16, 25, 59], strictness and homogeneity are imposed through relational integrity constraints. In these studies, non-summarizability is avoided by making sure that the aforementioned ICs are always satisfied. A similar approach is taken in [41, 42, 44], proposing some multidimensional constraints to assure summarizability. In addition, multidimensional or relational normalization is another approach for assuring summarizability in MDDBs [51, 52, 56, 50].

On the other hand, several approaches to directly repairing non-summarizable MDDBs have been proposed recently. They restore strictness and/or homogeneity by changing either the dimension schema [8, 41, 44, 42], or the dimension instance [13, 18, 22, 24, 62]. The idea is to modify the dimension schema or the instance in a minimal way, so that the resulting dimension instance satisfies the summarizability constraints.

The aforementioned MD repair mechanisms focus solely on the multidimensional data model, and its properties. More specifically, they do not take into account any relational/multidimensional implementation of the MDDB in addressing non-summarizability. However, in the case of ROLAPs, the MD repair operations have some side effects on the relational representation. Hence, the issue of non-summarizability should be addressed by taking these side effects into account.

1.1 Problem Statement and Contributions

In this thesis, unlike existing direct MD repair approaches, we are interested in handling non-summarizability by first considering a relational representation of MDDBs. We take a novel *relational* approach to the problem of non-summarizability. Our goal is to investigate the feasibility of using *relational repair* operations for dealing with non-summarizability, i.e. to restore strictness and homogeneity by repairing the underlying relational database. We achieve this goal in three steps (see Figure 1.8):

1. We propose an *expressive* relational representation for MDDB: As a preliminary step, we first formalize the MDM using the graph-based notation proposed in [43, 40] (see Chapter 2). We refer to this formalization as the *H-M model*. Next, we propose a mapping from MDDB to RDB, an *MD2R mapping*, which translates the multidimensional data model into a relational model. We will elaborate on the properties of this mapping later in this section.

   Since our approach is based on relational repairs, we must also detect the case of non-summarizability in the relational database. Hence, beside mapping the dimension schema and the dimension instance, we also propose relational *integrity constraints* that correspond to the summarizability conditions, strictness and homogeneity (see Chapters 3 and 4).

2. We introduce repairs for the inconsistent relational instance: Using the aforementioned mapping, non-summarizability in the MD instance is reflected as inconsistencies in the relational instance. In this phase, we use existing relational repair approaches for restoring consistency at the relational level. In particular, we define the repair operation and minimality criteria for obtaining the set of *minimal relational repairs*. Much work has been done in the area of relational repairs (see [12, 26] for a survey). We take advantage of this already existing rich body of research, and apply it to the non-summarizability issue in MDDBs (see Chapter 5).

3. We invert the mapping to obtain a summarizable dimension: The final step is to obtain the set of repaired dimension instances from the set of minimal relational repairs. To this end, we must translate back the relational instance into a multidimensional instance. The feasibility of this phase depends on the *invertibility* of MD2R mapping [29, 7] (see Chapter 6).

The whole process is described in Figure 1.8.

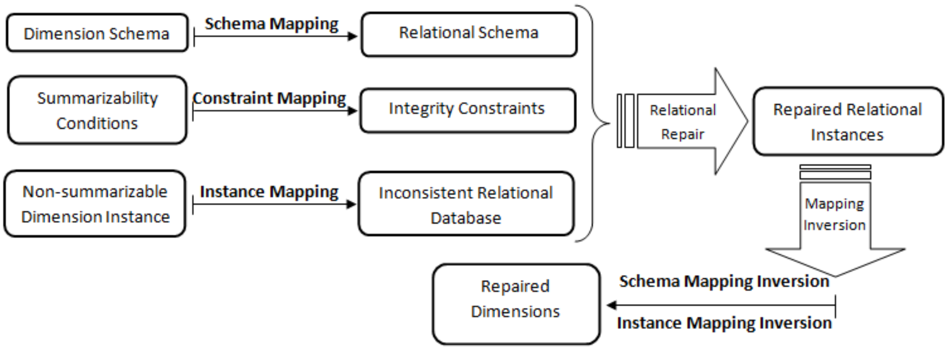


Figure 1.8: An overview of our proposed solution for non-summarizability

As shown in Figure 1.8, we start by translating the MD instance into an *expressive* relational instance [1]. This expressivity is reflected in the following properties:

(I) In order to take advantage of the relational repair approaches for handling non-summarizability, we must be able to check the summarizability conditions through a set of *integrity constraints* over the MDDB relational representation. Checking summarizability through ICs should be as efficient as possible, since these ICs are the basis for repairing the corresponding relational database.

(II) The inconsistent relational instance should be easily repairable. Otherwise, the cost of repairing the relational database would overcome the benefit of using this approach to deal with non-summarizability.

(III) The multidimensional database should be fully retrievable from the relational database. To this end, there should be *no information loss* in mapping the multidimensional database into a relational database, or the other way around. Intuitively, any information loss in this mapping results in an incomplete retrieval of repaired dimension instance from repaired relational instance [29, 7].

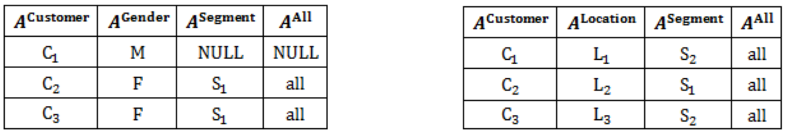
[1] Notice that in this translation we are considering only a snapshot of the MD instance, i.e. we do not consider *slowly changing dimensions* in the MD2R mapping and the historical changes in the MDDB are not translated.

According to [18], it is not feasible to use the relational repair techniques over the existing ROLAP approaches like star or snowflake to obtain dimension repairs. Based on the aforementioned study, it is either not possible to represent the summarizability constraints with the relational constraints in star and snowflake databases, or the minimal relational repairs obtained in these databases do not coincide with minimal multidimensional repairs. That is why in this thesis, we propose a relational reconstruction of MDDBs which enables the use of relational repairs for resolving non-summarizability.

In order to identify a suitable relational representation of MDDBs to be used in our approach, we first investigate the two best-known relational implementations of MDDBs and DWs, *Star* and *Snowflake*. We show that none of these schemas meet expressivity properties (I)-(III) mentioned above (cf. Chapter 3). This leads us to define a *new relational representation* for the MD instances; a *path-based* approach for mapping the multidimensional data model into a relational model. The new relational schema is called a *path relational schema*. The path schema will be used as the basis of our repair approach.

Example 1.4. (example 1.1 continued) Figure 1.9 shows how the `Customer` dimension is represented in path schema. Each relational table represents a path from the bottommost category to the topmost category in the dimension schema. The hierarchy in Figure 1.4a contains two examples of the aforementioned path, and hence we have two tables representing them. Each tuple in these tables resembles an instance of that path in the dimension instance. Intuitively, a path instance is a sequence of elements that reside on that path. In this figure, each category c of the `Customer` dimension is associated to the relational attribute A^c.

Representing the `Store` and `Date` dimensions in path database is more straightforward. Due to the linear structure of these hierarchies, only one table is needed to represent each of them in the path database. In case of `Store` dimension, the relational



A^{Customer}	A^{Gender}	A^{Segment}	A^{All}
C_1	M	NULL	NULL
C_2	F	S_1	all
C_3	F	S_1	all

(a) Table $RP_1^{\texttt{Cus}}$ for the left path $\texttt{P}_1^{\texttt{Cus}}$ in the `Customer` dimension schema

A^{Customer}	A^{Location}	A^{Segment}	A^{All}
C_1	L_1	S_2	all
C_2	L_2	S_1	all
C_3	L_3	S_2	all

(b) Table $RP_2^{\texttt{Cus}}$ for the right path $\texttt{P}_2^{\texttt{Cus}}$ in the `Customer` dimension schema

Figure 1.9: `Customer` dimension represented in path schema

table in path database contains attributes: $A^{\texttt{Branch}}$, $A^{\texttt{City}}$, $A^{\texttt{Country}}$, $A^{\texttt{All}}$; and the table representing `Date` dimension has $A^{\texttt{Day}}$, $A^{\texttt{Month}}$, $A^{\texttt{Year}}$, $A^{\texttt{All}}$ as its attributes. □

The contributions of this thesis are as follows:

1. We propose a new relational representation for MDDBs, the path schema (cf. Chapter 4). This relational schema is expressive and adequate enough to be the basis of our MD repair approach. In particular, path schema enables efficient checking of strictness and homogeneity through ICs. Furthermore, it is easily repairable compared to star or snowflake (cf. Chapter 5). The former requires simple updates at the attribute level, while the latter cases might require tuple-based operations. Finally, the invertibility of the proposed MD2R mapping enables the generation of MD repairs from repaired path database (cf. Chapter 6). In general, the path schema has the expressivity properties mentioned before, and hence is a good choice to be used for tackling non-summarizability through relational repairs.

   Besides its virtues in MD repairing, path schema has some useful data-related properties compared to star and snowflake schemas (cf. Chapter 8). Our experiments reveal that, path has the second best performance in query answering, with star schema being the first. The negligible difference between the path schema and the star schema in this ranking comes from the fact that, in a path instance a dimension might be represented in more than one relational table, while in the star schema, there is always a single relational table corresponding to each dimension. This difference in MD data representation results in different relational query structure. In the case of the star schema, this relational query is more optimized compared to that for the path schema.

We also conduct some experiments around performance of star, snowflake and path schemas in detecting violation of ICs corresponding to summarizability conditions. Among these schemas, path schema is the only MDDB relational representation that enables efficient and complete checking of the summarizability constraints through ICs. Hence, it can be said that, the path schema is not only a repair-oriented schema, but also an efficient implementation of MDDBs.

2. The MD repairs generated through our approach are a form of *instance-based* repairs, as opposed to schema-based repairs. As discussed above, we obtain the MD repairs by inverting the MD2R mapping. Since the relational repair operations modify the data and leave the database schema unchanged, the new MD instance obtained as a result of mapping inversion has the same schema as the original instance.

   We compare our MD repairs with repairs obtained from other instance-based MD repair approaches. In particular, we analyze the correspondence between our repairs and those generated by [18], as a well-known instance-based repair mechanism. An interesting observation is that, our class of MD repairs and the class of MD repairs proposed by [18] are *disjoint*. That is, in general there is no correspondence between the set of minimal MD repairs produced through our relational approach and those proposed in [18]. This is due to the fact that we generate MD repairs that correspond to minimal relational repairs in the MDDB relational layer. More specifically, we force the minimality of repairs on the relational side as well as multidimensional side, while in [18], minimal MD repairs are obtained by performing minimum changes to the original dimension instance.

We propose a pure MD characterization of the dimension repairs that are obtained through our relational approach. Unlike existing approaches, our approach considers the effect of MD repair operations on the MDDB relational implementation. In MD terms, this consideration is reflected in a prioritization of dimension instance edges for modification during MD repair process.

This thesis is organized as follows. Chapter 2 presents a formalization of the multidimensional data model which will be used throughout the thesis. In Chapter 3, we explain why star and snowflake schemas are not a perfect choice to represent or restore summarizability through relational repairs. We propose the path schema as a new relational representation for MDDB, and formalize the MD2R mapping in Chapter 4 (first step of Figure 1.8). Chapter 5 provides the relational repair semantics which is used in the second phase of our approach for restoring consistency in the path database. As the third step of our solution, we obtain dimension repairs through MD2R mapping inversion in Chapter 6. A pure MD characterization of the dimension repairs generated by our relational approach is given in Chapter 7. Chapter 8 analyzes the performance of the path schema in query answering and inconsistency detection, compared to star and snowflake schemas. Finally, in Chapter 9, we conclude and highlight some future research directions.

Chapter 2

Preliminaries

2.1 Basic Notions

Multidimensional data model is composed of logical cubes, facts and dimensions.

A *data cube* is a data structure that allows fast analysis of data. It can also be defined as the capability of manipulating and analyzing data from multiple perspectives. The arrangement of data into cubes in MDMs overcomes some limitations of relational databases. Data cube consists of numeric facts called measures which are categorized by dimensions.

Facts are business performance measurements, which are typically numeric and additive. Measures populate the cells of a logical cube with the facts collected about business operations.

In order to measure the facts at different granularity levels, each *dimension* is represented by a hierarchy of categories. In other words, a dimension is a data element that categorizes each item in a data set into non-overlapping categories. Dimensions form the edges of a logical cube, and thus of the measures within the cube. They provide structured labeling information to, otherwise unordered, numeric measures.

In multidimensional modeling, several *categories* at different granularity levels can be defined for each dimension. These categories form a hierarchy, which is referred to as *dimension schema.* The hierarchy of elements belonging to these categories is called *dimension instance.*

The key feature of multidimensional modeling is its simplicity, which allows readability, and efficient database navigation. Beside that, multidimensional models are known to be understandable, predictable and extendable [50, 47, 43].

2.2 Formalizing the Multidimensional Data Model

Several formalizations for MDMs have been proposed in literature. In some of these works, the entity-relationship (ER) model in relational databases is extended, in order to capture multidimensional semantics. Description logic (DL) can be used to give a formal model-theoretic semantics to the resulting data model. Through this semantics, one can employ reasoning to verify the multidimensional data model specification [37, 38, 69].

Unified Modeling Language (UML) is another well-known modeling language, that can be extended to represent multidimensional data models. Several studies extended the UML Class Diagram in order to enhance its multidimensional expressivity [1, 68]

.

[35, 63] are among the studies in which a logic-based formalization for the multidimensional data models is proposed. This formalization builds the basis of a logic-based MD query language as well.

In [20, 45], a graph-based formalization of multidimensional data models is proposed (cf. [43, 40] for precise details). Unlike other works, this formalization approach does not require any complex modeling language. Due to its simplicity and readability, this formalization approach is used in several works, including [13, 22, 24, 41, 42, 43].

In our case, it is not necessary to have a complex logic-based infrastructure, or provide a reasoning engine for the multidimensional data model. Hence, among all of the above approaches, we will choose the MDM formalization proposed in [20, 45], for modeling the non-summarizability problem. This formal model is explained in more details in Section 2.2.1.

Since we are addressing non-summarizability, we also need to formalize the summarizability conditions. Section 2.2.1 provides formal definitions for strictness and homogeneity. This symbolic representation will be used throughout this thesis.

2.2.1 Hurtado-Mendelzon Model

A dimension schema, $\mathcal{S}$, is a *directed acyclic graph* (DAG), formalized by a pair of the form $\langle \mathcal{C}, \nearrow \rangle$. In this pair, $\mathcal{C}$ is the set of all categories, and $\nearrow$ is the *parent-child relationship* between these categories. The transitive and reflexive closure of this binary relation is indicated 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 $c_k \nearrow c_j$.

As an standard, in every dimension schema, there is a distinguished top category named `All`, which is reachable from every other category, i.e. for all categories c: $c \nearrow^*$ `All` holds. In addition, every dimension schema has a *unique category*, that does not have any children. This category is referred to as *base category.*

Similar to the dimension schema, the dimension (also called dimension instance), $\mathcal{D}$, is represented by a pair of the form $\langle \mathcal{M}, < \rangle$. In this pair, $\mathcal{M}$ represents the set of all elements (ground atoms), and $<$ (sometimes denoted by $<_{\mathcal{D}}$) is the *parent-child relationship* between these elements. The transitive and reflexive closure of $<$ is shown by $<^*$. For the distinguished category `All`, the only element defined is `all`. Notice that under this assumption and in a given instance, `all` may still not be reached by elements of lower-level categories.

A function named δ: $\mathcal{M} \mapsto \mathcal{C}$, is used to map each element to the category it belongs to. If $\delta(e) = c$, then we also say that $e \in c$. Consequently, $e_i < e_j$, if and only if, $\delta(e_i) \nearrow \delta(e_j)$.

A *complete* dimension instance is the one, in which elements that do not have children, are all base elements. In an incomplete dimension, we have elements that are not reachable from the base category. These elements are ignored in aggregate query computation. In this thesis, we assume the dimension instances to be complete.

In this MDM formalization, we also have a roll up relation which can be built to any pair of categories c_i, c_j in the schema, $\mathcal{R}_{c_i}^{c_j}(\mathcal{D})$. This relation returns a set of pairs of the form $(e_i,\ e_j)$, in which the first element belongs to c_i, and the second element is a member of c_j. These two elements are related through the $<^*$ relation, i.e. $e_i <^* e_j$. In general, this roll up relation is not necessarily a *function*, nor is it *total.*

Definition 2.1. The roll up relation $\mathcal{R}_{c_i}^{c_j}(\mathcal{D})$: $\mathcal{M} \mapsto \mathcal{M}$ is said to be a *function*, if it maps every element of its domain (elements belonging to c_i) to at most one element in c_j. □

Definition 2.2. The roll up relation $\mathcal{R}_{c_i}^{c_j}(\mathcal{D})$: $\mathcal{M} \mapsto \mathcal{M}$ is said to be *total*, if the set $\{a \in \mathcal{M} \mid$ exists b, *(a,b)* $\in \mathcal{R}\}$ is equal to the set of elements belonging to c_i, i.e. $\delta^{-1}(c_i)$. □

Example 2.1. (example 1.1 continued) Using the above notation, the `Customer` dimension schema, $\mathcal{S}_{\texttt{Customer}}$, in Figure 1.4a is formalized as follows:

$$\begin{aligned}
\mathcal{C} &= \{\texttt{Customer}, \texttt{Gender}, \texttt{Location}, \texttt{Segment}, \texttt{All}\}.\\
\nearrow &= \{(\texttt{Customer}, \texttt{Gender}), (\texttt{Customer}, \texttt{Location}), (\texttt{Gender}, \texttt{Segment}), (\texttt{Location}, \texttt{Segment}),\\
&\quad (\texttt{Segment}, \texttt{All})\}.
\end{aligned}$$

The `Customer` dimension instance, $\mathcal{D}_{\texttt{Customer}}$ is represented by the pair $\langle\mathcal{M},<\rangle$. The set of elements, $\mathcal{M}$, and the relationship between them, $<$, are as follows:

$$\begin{aligned}
\mathcal{M} &= \{\mathtt{C}_1, \mathtt{C}_2, \mathtt{C}_3, \mathtt{F}, \mathtt{M}, \mathtt{L}_1, \mathtt{L}_2, \mathtt{L}_3, \mathtt{S}_1, \mathtt{S}_2, \mathtt{all}\}. \\
< &= \{(\mathtt{C}_1, \mathtt{M}), (\mathtt{C}_2, \mathtt{F}), (\mathtt{C}_3, \mathtt{F}), (\mathtt{C}_1, \mathtt{L}_1), (\mathtt{C}_2, \mathtt{L}_2), (\mathtt{C}_3, \mathtt{L}_3), (\mathtt{F}, \mathtt{S}_1), (\mathtt{L}_1, \mathtt{S}_2), (\mathtt{L}_2, \mathtt{S}_1), (\mathtt{L}_3, \mathtt{S}_2), \\
&\quad (\mathtt{S}_1, \mathtt{all}), (\mathtt{S}_2, \mathtt{all})
\end{aligned}$$

The set of elements belonging to each category can be determined by inverting the δ function. For example, $\delta^{-1}(\mathtt{Location})$ contains elements $\mathtt{L}_1$, $\mathtt{L}_2$ and $\mathtt{L}_3$. Similar approach can be taken for the rest of categories in the `Customer` dimension schema.

$$\begin{aligned}
\delta^{-1}(\mathtt{Customer}) &= \{\mathtt{C}_1, \mathtt{C}_2, \mathtt{C}_3\} \\
\delta^{-1}(\mathtt{Gender}) &= \{\mathtt{F}, \mathtt{M}\} \\
\delta^{-1}(\mathtt{Location}) &= \{\mathtt{L}_1, \mathtt{L}_2, \mathtt{L}_3\} \\
\delta^{-1}(\mathtt{Segment}) &= \{\mathtt{S}_1, \mathtt{S}_2\} \\
\delta^{-1}(\mathtt{All}) &= \{\mathtt{all}\}
\end{aligned}$$

In order to find the parents and ancestors of dimension instance elements, we can use the roll up relation. For instance, the grand parents of all base elements in the `Segment` category, are as follows:

$$\mathcal{R}_{\mathtt{Customer}}^{\mathtt{Segment}} = \{(\mathtt{C}_1, \mathtt{S}_2), (\mathtt{C}_2, \mathtt{S}_1), (\mathtt{C}_3, \mathtt{S}_1), (\mathtt{C}_3, \mathtt{S}_2)\}.$$

It can be easily checked that $\mathcal{R}_{\mathtt{Customer}}^{\mathtt{Segment}}$ is not a function. As explained before, by default, the roll up relation in neither total, nor a function. An example of a partial roll up relation is $\mathcal{R}_{\mathtt{Gender}}^{\mathtt{Segment}}$, which does not map element `M` to any element in the `Segment` category:

$$\mathcal{R}_{\mathtt{Gender}}^{\mathtt{Segment}} = \{(\mathtt{F}, \mathtt{S}_1)\}$$

However, it is possible to have a total roll up relation, which is also a function. Consider $\mathcal{R}_{\mathtt{Customer}}^{\mathtt{Location}}$ as an example of such roll up relation:

$$\mathcal{R}_{\mathtt{Customer}}^{\mathtt{Location}} = \{(\mathtt{C}_1, \mathtt{L}_1), (\mathtt{C}_2, \mathtt{L}_2), (\mathtt{C}_3, \mathtt{L}_3)\}$$

□

A *granularity* over a list of dimension schemas, $\langle \mathcal{S}_1,\cdots,\mathcal{S}_n\rangle$, is a list of categories of the form $\langle\ c_1,\cdots,c_n\ \rangle$, in which each c_i belongs to schema $\mathcal{S}_i$.

A *fact table* consists of a granularity and a measure. Measures are formalized by a set of variables. Each variable has its specific domain of values. A *base fact table* is the one, in which all of the categories in the granularity are base categories of their corresponding dimension schema.

Example 2.2. (example 2.1 continued) In Example 1.1, `Sales` is the only measure introduced. However, we can have other measures, such as the discount amount offered for the purchase, the time customer spent on shopping the product and etc.

The granularity of the fact table in Figure 1.1 is ⟨`Customer`, `Branch`, `Day`⟩. This granularity is defined over the list of dimension schemas ⟨`Customer`, `Store`, `Date`⟩.

It can be easily checked in Figures 1.4a, 1.2 and 1.3 that, the categories `Customer`, `Branch` and `Day` are all base categories of their corresponding dimension schemas. Hence, the fact table in Figure 1.1 is an example of base fact table. □

Summarizability Constraints

Similar to relational databases, several types of constraints have been proposed to capture different kinds of semantics in MDDBs. Among these, strictness and homogeneity have received lots of attention, since they together can assure summarizability in any dimension. These constraints are global conditions that can also be imposed locally.

Definition 2.3. [18, 24] (a) For a dimension schema $\mathcal{S} = \langle \mathcal{C}, \nearrow\rangle$, a *strictness constraint* is an expression of the form $c_i \rightarrow c_j$, where $c_i,\ c_j \in \mathcal{C},\ c_i \neq c_j$, and $c_i \nearrow^* c_j$. This constraint is satisfied by a dimension instance $\mathcal{D}$, denoted $\mathcal{D} \models c_i \rightarrow c_j$, iff the roll up relation $\mathcal{R}_{c_i}^{c_j}$ is a (possibly partial) function.

(b) The dimension instance $\mathcal{D}$ is *strict* if it satisfies the *full-strictness condition*, namely the set, $FS^{\mathcal{S}} = \{c_i \rightarrow c_j \text{ — } c_i,\, c_j \in \mathcal{C},\, c_i \neq c_j, \text{ and } c_i \nearrow^* c_j\}$, of all strictness constraints. □

Example 2.3. (example 2.1 continued) According to Example 2.1, the roll up relation $\mathcal{R}_{\texttt{Customer}}^{\texttt{Segment}}$ is not a function. Hence, based on Definition 2.3, the `Customer` dimension is non-strict, i.e. $\mathcal{D}_{\texttt{Customer}} \not\models \texttt{Customer} \rightarrow \texttt{Segment}$. □

Example 2.4. (example 2.1 continued) The `Customer` dimension is strict if it satisfies the following local constraints:

$$\begin{aligned} FS^{\mathcal{S}} \;=\; & \{\texttt{Customer} \rightarrow \texttt{Location}, \texttt{Customer} \rightarrow \texttt{Gender}, \texttt{Customer} \rightarrow \texttt{Segment}, \\ & \texttt{Customer} \rightarrow \texttt{All}, \texttt{Location} \rightarrow \texttt{Segment}, \texttt{Location} \rightarrow \texttt{All}, \\ & \texttt{Gender} \rightarrow \texttt{Segment}, \texttt{Gender} \rightarrow \texttt{All}, \texttt{Segment} \rightarrow \texttt{All}\} \end{aligned} \tag{2.1}$$

Notice that, in a strict dimension, every roll up relation is a function. □

Definition 2.4. [18, 24] (a) For a dimension schema $\mathcal{S} = \langle \mathcal{C}, \nearrow \rangle$, a *homogeneity constraint* (also known as covering) is an expression of the form $c_i \Rightarrow c_j$, where $c_i,\, c_j \in \mathcal{C}$, $c_i \neq c_j$, and $c_i \nearrow c_j$. This constraint is satisfied by a dimension instance $\mathcal{D}$, denoted $\mathcal{D} \models c_i \Rightarrow c_j$, iff the roll up relation $\mathcal{R}_{c_i}^{c_j}$ is total.

(b) The dimension instance $\mathcal{D}$ is *homogeneous* if it satisfies the *full-homogeneity condition*, namely the set, $FH^{\mathcal{S}} = \{c_i \Rightarrow c_j \text{ — } c_i,\, c_j \in \mathcal{C},\, c_i \neq c_j, \text{ and } c_i \nearrow c_j\}$, of all homogeneity constraints. □

Example 2.5. (example 2.1 continued) According to Example 2.1, the roll up relation $\mathcal{R}_{\texttt{Gender}}^{\texttt{Segment}}$ is not total. Hence, based on Definition 2.4, the `Customer` dimension is heterogeneous, i.e. $\mathcal{D}_{\texttt{Customer}} \not\models \texttt{Gender} \Rightarrow \texttt{Segment}$. □

Example 2.6. (example 2.1 continued) The `Customer` dimension is homogeneous if it satisfies the following local constraints:

$$\begin{aligned} FH^{\mathcal{S}} &= \{\texttt{Customer} \Rightarrow \texttt{Location}, \texttt{Customer} \Rightarrow \texttt{Gender}, \texttt{Location} \Rightarrow \texttt{Segment}, \\ &\quad \texttt{Gender} \Rightarrow \texttt{Segment}, \texttt{Segment} \Rightarrow \texttt{All}\} \end{aligned} \tag{2.2}$$

Notice that, in a homogeneous dimension, every roll up relation is total.

It can be easily checked that, the set of roll up relations to check for homogeneity is a *subset* of the relations that should be checked for strictness. □

Although we will use the null value, `NULL` , in the relational representation of the original MD instance, we assume that `NULL` can not be an element in the dimension instance, i.e. $\texttt{NULL} \notin \mathcal{M}$. The semantics of `NULL` will be as in SQL relational databases, which is logically captured in [17].

The assumptions we made about the existence of a single base category, and completeness of dimension instance are just for simplifying the representation [40]. Obviously, our approach can be easily extended to address cases where these assumptions are not met.

Chapter 3

ROLAP and MDDB Semantics

3.1 Introduction

In ROLAPs, MDDBs are mostly represented as a star or snowflake database. This chapter studies the feasibility of using either of these relational representations in our approach. We start by formalizing star and snowflake schema as a mapping from MDDB to RDB. Next, we evaluate these mappings based on the criteria brought in Chapter 1, for an expressive relational representation of MDDBs.

3.2 Star Schema Revisited

Star schema is the generic representation of a multidimensional instance as a relational database [50]. In star schema, a fact table consisting of numeric measurements is joined to a set of dimension tables containing descriptive attributes. This starlike structure was referred to as *star join schema* in the earliest days of relational databases [60]. In this schema, dimension tables have a simple primary key, while for the fact tables a set of foreign keys makes up a composite key. This composite key consists of the relevant dimension tables keys.

The main advantage of star schema is its simplicity. The query answering process is never complex in this schema. The only joins and conditions involve a fact table and a single level of dimension tables. Query answering does not deal with indirect dependencies to other tables, which are possible in a better normalized snowflake schema (cf. Section 3.4). In general, it is not clear that, the overheads in further normalizing the dimension tables of a star schema, in order to obtain a snowflake schema, outweigh the simplicity of the star schema [54].

Star and snowflake have the same structure for storing the fact table [50]. An example of this structure is shown in Figure 1.1. As a result, our major focus is on how dimensions are represented by these schemas.

Star schema has a *flat* structure for storing the dimensions. Here, the whole dimension is represented by a *single table*. In addition, the set of all categories constitutes the attributes of that table. Star mapping requires the dimension schema to have a single base category. Each base element along with its parents and ancestors represent a tuple in the dimension table [50, 41]. A precise relational formalization of star schema can be found in [54].

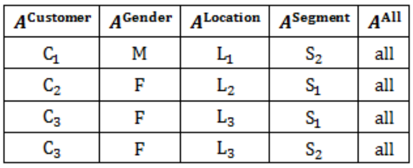
Based on the above discussions, mapping a dimension to a star database is formalized as follows. Part (I) specifies how star schema represents dimension schema, and Part (II) is about translating the instances.

(I) For each $c \in \mathcal{C}$, create an attribute A^c.

For the set of all categories $\mathcal{C}$, create a relational predicate $R[A^{c_1}, \cdots, A^{c_n}]$.

(II) For each $e \in \mathcal{M}$, where $\delta(e)$ is a base category, create tuples of the form $R(e, e_1, \cdots, e_n)$, where $e_i =$ `NULL` , or $e <^* e_i$ for $1 \leq i \leq n$.

Applying the above rules to the `Customer` dimension results in the star database of Figure 3.1. In this relational instance, the only table is $R[A^{\texttt{Customer}}, A^{\texttt{Gender}}, A^{\texttt{Location}}, A^{\texttt{Segment}}, A^{\texttt{All}}]$. The set of attributes of this table is generated using rule (I). `Customer` is the base category for this dimension. Hence, for generating the tuples belonging to



A^{Customer}	A^{Gender}	A^{Location}	A^{Segment}	A^{All}
C_1	M	L_1	S_2	all
C_2	F	L_2	S_1	all
C_3	F	L_3	S_1	all
C_3	F	L_3	S_2	all

Figure 3.1: `Customer` dimension represented in star schema

R, we have to calculate the following roll up relations:

$$\begin{aligned}
\mathcal{R}_{\mathtt{Customer}}^{\mathtt{Gender}} &= \{(\mathtt{C}_1,\mathtt{M}),(\mathtt{C}_2,\mathtt{F}),(\mathtt{C}_3,\mathtt{F})\}\\
\mathcal{R}_{\mathtt{Customer}}^{\mathtt{Location}} &= \{(\mathtt{C}_1,\mathtt{L}_1),(\mathtt{C}_2,\mathtt{L}_2),(\mathtt{C}_3,\mathtt{L}_3)\}\\
\mathcal{R}_{\mathtt{Customer}}^{\mathtt{Segment}} &= \{(\mathtt{C}_1,\mathtt{S}_2),(\mathtt{C}_2,\mathtt{S}_1),(\mathtt{C}_3,\mathtt{S}_1),(\mathtt{C}_3,\mathtt{S}_2)\}\\
\mathcal{R}_{\mathtt{Customer}}^{\mathtt{All}} &= \{(\mathtt{C}_1,\mathtt{all}),(\mathtt{C}_2,\mathtt{all}),(\mathtt{C}_3,\mathtt{all})\}
\end{aligned}$$

Using the above roll up relations, we must find all of the parents and grand parents for each base element, i.e. $\mathtt{C}_1$, $\mathtt{C}_2$ and $\mathtt{C}_3$ (rule (II)).

$$\begin{aligned}
\mathtt{C}_1 &: \{\mathtt{M},\mathtt{L}_1,\mathtt{S}_2,\mathtt{all}\}\\
\mathtt{C}_2 &: \{\mathtt{F},\mathtt{L}_2,\mathtt{S}_1,\mathtt{all}\}\\
\mathtt{C}_3 &: \{\mathtt{F},\mathtt{L}_3,\mathtt{S}_1,\mathtt{S}_2,\mathtt{all}\}
\end{aligned}$$

The sets obtained for elements $\mathtt{C}_1$ and $\mathtt{C}_2$ are mapped to the first two tuples in R[`Customer`, `Gender`, `Location`, `Segment`, `All`] using rule (II).

$$\begin{aligned}
\mathtt{C}_1 \in \mathcal{M}\ \textit{and}\ \delta(\mathtt{C}_1) = \mathtt{Customer}: \ \textit{base category} &\mapsto R(\mathtt{C}_1,\mathtt{M},\mathtt{L}_1,\mathtt{S}_2,\mathtt{all})\\
\mathtt{C}_2 \in \mathcal{M}\ \textit{and}\ \delta(\mathtt{C}_2) = \mathtt{Customer}: \ \textit{base category} &\mapsto R(\mathtt{C}_2,\mathtt{F},\mathtt{L}_2,\mathtt{S}_1,\mathtt{all})
\end{aligned}$$

However, the third set can not be translated to a single tuple. We need two tuples for representing multiple parents of element $\mathtt{C}_3$ in the `Segment` category. As a result, although the `Customer` dimension contains three base elements, we have four tuples in the dimension table of Figure 3.1.

$$\mathtt{C}_3 \in \mathcal{M}\ \textit{and}\ \delta(\mathtt{C}_3) = \mathtt{Customer}: \ \textit{base category} \ \mapsto \ \{R(\mathtt{C}_3,\mathtt{F},\mathtt{L}_3,\mathtt{S}_1,\mathtt{all}),R(\mathtt{C}_3,\mathtt{F},\mathtt{L}_3,\mathtt{S}_2,\mathtt{all})\}$$

Figure 3.2 shows the relation between the dimension tables `Customer`, `Store` and `Date` and the `Sales` fact table in star database.

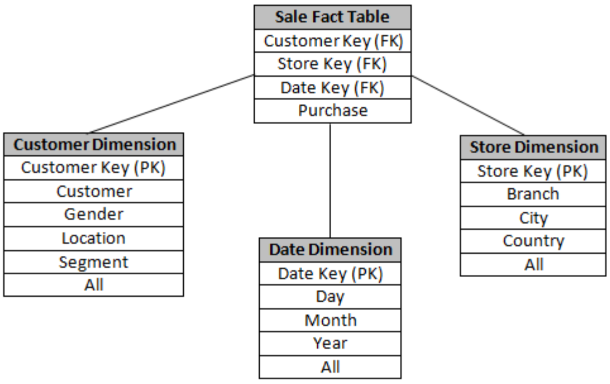


Figure 3.2: Example 1.1 implemented as star database

3.3 Why Star is not a Good Choice?

Now that star mapping is precisely formalized, we evaluate this MD2R mapping to see if it can be used in our approach. Our discussions show that, star schema does not have the properties of an expressive MDDB relational representation, mentioned in Chapter 1.

3.3.1 Summarizability Constraints in Star Schema

In this section, we show that although star schema enables detecting non-strictness through functional dependencies (FDs), it is not expressive enough to represent heterogeneity. This fact makes star schema an inappropriate choice for our approach.

Homogeneity in Star Schema

One of the problems with star schema is that checking homogeneity through ICs in this relational schema is not straightforward. In fact, star schema is *not expressive* *enough* to capture heterogenous dimensions in general. In some cases, the condition of having no parent in a specific category is reflected by a `NULL` value for the corresponding attribute in the dimension table. However, there are situations in which heterogenous instances *can not* be detected through inspecting `NULL` values.

Example 3.1. (example 2.6 continued) In Example 2.5, we showed that, due to the missing parent for element `M` in category `Segment`, the `Customer` dimension is heterogeneous. However, the database instance in Figure 3.1 does not have any `NULL` value for the $A^{\texttt{Segment}}$ attribute.

The reason is that, in star schema, we are storing the parents and grand parents of each base element as separate tuples. In our case, since $\texttt{C}_1$ rolls up to segment $\texttt{S}_2$ through location $\texttt{L}_1$, the $A^{\texttt{Segment}}$ attribute is not `NULL` . In star mapping, we do not consider the fact that $\texttt{C}_1$ does not have any ancestor in the `Segment` category, through gender `M`.

Star failure in representing heterogeneous instances does not happen all the time. For instance, in `Customer` dimension, if $\texttt{C}_1$ had no parent in category `Gender`, heterogeneity would have been reflected by a `NULL` value for attribute $A^{\texttt{Gender}}$ in the first tuple of dimension table. □

It can be concluded that, in star schema, imposing `NOT NULL` constraint to dimension table attributes, helps detecting *some, but not all,* heterogeneous dimensions.

Strictness

In star schema, a local strictness constraint between two categories in the dimension schema is represented by a *functional dependency* between the mapped attributes of the dimension table (rule (III)). In fact, functional dependency is the natural translation of strictness into integrity constraints [13, 18, 22, 24, 41, 42] (see Section 4.5 for a detailed discussion on FD evaluation in presence of `NULL` values).

(III) (FD Generation)

$(c_i \rightarrow c_j) \mapsto_{Star} (R\colon\ A^{c_i} \rightarrow A^{c_j}).$

Example 3.2. (example 2.4 continued) In this example, we generate the set of integrity constraints needed for checking strictness in the `Customer` dimension, through the star database of Figure 3.1. According to Definition 2.3, strictness should be checked for every pair of categories that are related through the $\nearrow^*$ relation. Since `All` is a special category, which has only one pre-defined element, `all`, there is no need to check whether it is in a strict relation with other lower categories or not. Using rule (III), and the set of constraints in Example 2.4 $\mathit{FS}^{\mathcal{S}}$, we obtain the following minimized set of functional dependencies for imposing global strictness to the `Customer` dimension:

$$R : \{A^{\texttt{Customer}} \rightarrow A^{\texttt{Gender}}, A^{\texttt{Customer}} \rightarrow A^{\texttt{Location}}, A^{\texttt{Gender}} \rightarrow A^{\texttt{Segment}}, A^{\texttt{Location}} \rightarrow A^{\texttt{Segment}}\} \tag{3.1}$$

It can be easily checked that the database in Figure 3.1 is violating the FDs R : $\{A^{\texttt{Gender}} \rightarrow A^{\texttt{Segment}}\ , A^{\texttt{Location}} \rightarrow A^{\texttt{Segment}}\ \}$. □

3.3.2 Invertibility of Star Mapping

Moving back from star database to dimension instance is not straightforward. In this section, we show that, this MD2R mapping is not uniquely invertible. Due to the information loss in mapping MDDB to RDB, we can not obtain a single MD instance from a star database instance. For a complete discussion about invertibility of MD2R mapping see Chapter 6.

Due to the flat structure of the star schema, we are *losing* some information in mapping a dimension to a relational table. In other words, star schema does not store every property of the multidimensional hierarchies. In fact, we might obtain several possible multidimensional instances from inverting a single star database. Hence, we can say that, inverting this MD2R mapping is *non-deterministic.*

In the process of mapping inversion, the set of categories are re-generated from the set of attributes of the dimension table (rule (IV)). However, the edges between these categories can not be retrieved clearly from the star schema. That is why, we might end up having *several possible* dimension schemas from inverting a single star schema.

The aforementioned problem can also be seen from an instance-based perspective. In star mapping, we store the *roll up relations* in each tuple, and ignore the *direct edges* between the elements. As a result, we might have several dimension instances, that have the same roll up relation but differ in the < members (rule (V)). The star mapping inversion is formalized as follows.

(IV) For each attribute A appearing in some $R \in \mathcal{R}$, create a category (name) $c^A \in \mathcal{C}$. The set of so-created categories is denoted with $\mathcal{C}$.

(V) For each relational tuple $R(e_1, \cdots, e_n)$, with $R[A_1,\ldots, A_n]$, add elements $e_i \neq$ `NULL` , for $1 \leq i \leq n$ to the set of elements $\mathcal{M}$, where $\delta(e_i) = c^{A_i}$.

Notice that, the above rules can only determine the sets $\mathcal{C}$ and $\mathcal{M}$, and do not generate the parent-child relations in the dimension schema and the dimension instance.

Example 3.3. (examples 3.2 and 3.1 continued) Consider the star database in Figure 3.1. Using rule (IV), we obtain a unique set of categories, $\mathcal{C}$. This set is equal to what we obtained in Example 2.1. However, the set of edges between these categories, $\nearrow$, is not unique. In fact, each of the following sets is a potential candidate for the $\nearrow$ relation. In other words, $\mathcal{C}$ along with each of these sets is represented by the same table as in Figure 3.1 in the star schema.

1. $\{($`Customer`,`Gender`$)$, $($`Gender`,`Location`$)$, $($`Location`,`Segment`$)$, $($`Segment`,`All`$)\}$.

2. {(Customer,Gender), (Gender,Location), (Gender,Segment), (Location,All), (Segment,All)}.

3. {(Customer,Gender), (Customer,Location), (Gender,Segment), (Location,Segment), (Segment,All)}.

4. {(Customer,Gender), (Customer,Location), (Location,Segment), (Gender,All), (Segment,All)}.

5. {(Customer,Gender), (Customer,Location), (Gender,Segment), (Location,All), (Segment,All)}.

6. {(Customer,Gender), (Customer,Location), (Customer,Segment), (Gender,All), (Location,All), (Segment,All)}.

Notice that in generating these sets, we consider the order of attributes in the relational table. Without this consideration, the number of possible dimension schemas would be much larger. It can be checked, that the third set is equal to what we had in Example 2.1 for the Customer dimension. □

Based on our discussions, the weakness of star schema in representing heterogeneity at relational level, and the non-invertibility of the MDDB to star mapping make star schema an inappropriate choice for our solution.

3.4 Snowflake Schema Revisited

While star schema represents the dimension in a *flat* relational structure, snowflake provides a *hierarchical* relational representation for MDDB. In snowflake, the flat, single-table dimension in star schema is decomposed into a tree structure with potentially many nesting levels [50]. [54] considers star as a special case of the snowflake schema, in which the height of the hierarchical structure is equal to *one*.

Snowflake, is said to be the normalized version of the star schema. For the sake of space saving, the dimensions are represented in third normal form (cf. [3] for various normalization approaches in relational databases). However, this space saving is often disadvantageous. The reason is that typically, dimension tables are geometrically smaller than fact tables. Hence, improving storage efficiency by normalizing these tables has virtually no impact on the overall database size [50].

Despite the above fact, there are some cases in which normalization is needed. In dimensions with a huge number of base elements, repeating some values in each tuple seems inefficient. In such cases, snowflake is more preferred to star schema. Based on [50], we mostly trade off dimension table space for simplicity and accessibility.

The hierarchical structure of snowflake schema causes some complications in query processing. In contrast to the star schema, in snowflake we need to execute several join operations over the tables, in order to retrieve tuples for browsing and aggregate query processing. Numerous tables and joins in this process translate into lower query answering performance [50, 41] (cf. [54] for a more detailed evaluation of star and snowflake schemas from relational databases aspect). The performance of query answering in star and snowflake are analyzed in Chapter 8.

In the snowflake 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 parent categories of c. Each of these attributes points to or *references* the same attribute in the parent table [50, 41]. A precise relational formalization of snowflake schema is given in [54].

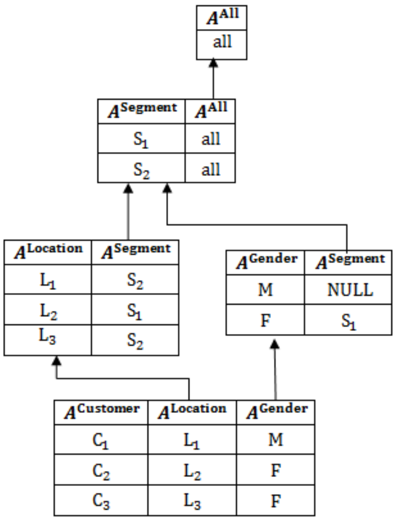
As mentioned in Section 3.2, the fact tables in both star and snowflake schemas are identical (see Figure 1.1). Hence, we are more interested in comparing the relational representation of dimensions in these two schemas. A formalization of snowflake mapping is given below. In this set of rules, part (I) represents schema mapping rules, and part (II) addresses instance mapping.

(I) For each $c \in \mathcal{C}$, create an attribute A^c.

For each $c \in \mathcal{C}$, create a relational predicate $R_c[A^c, A^{c_1}, \cdots, A^{c_n}]$, where $c \nearrow c_i$, for $1 \leq i \leq n$.

For all x $\big($if $R_c(\cdots, x, \cdots)$, then $\big(R_{c_1}(x, \cdots)$ or $R_{c_2}(x, \cdots)$ or $\cdots$ or $R_{c_n}(x, \cdots)\big)\big)$, where $c \nearrow c_i$, for $1 \leq i \leq n$.

(II) For each $e \in \mathcal{M}$, where $\delta(e)$=c, create a tuple $R_c(e, e_1, \cdots, e_n)$, where e_i = `NULL` , or $e < e_i$ for $1 \leq i \leq n$.



A^{All}
all

$A^{Segment}$	A^{All}
S_1	all
S_2	all

$A^{Location}$	$A^{Segment}$
L_1	S_2
L_2	S_1
L_3	S_2

A^{Gender}	$A^{Segment}$
M	NULL
F	S_1

$A^{Customer}$	$A^{Location}$	A^{Gender}
C_1	L_1	M
C_2	L_2	F
C_3	L_3	F

Figure 3.3: `Customer` dimension represented in snowflake schema

Applying the above rules to the `Customer` dimension results in the snowflake database of Figure 3.3. The tables in this database instance are generated using rule (I):

$$
\begin{aligned}
(\texttt{Customer} \in \mathcal{C}) &\mapsto (R_{\texttt{Customer}}[A^{\texttt{Customer}}, A^{\texttt{Location}}, A^{\texttt{Gender}}] \ \textit{where} \ \texttt{Customer} \nearrow \texttt{Location} \\
&\quad \textit{and} \ \texttt{Customer} \nearrow \texttt{Gender}). \\
(\texttt{Location} \in \mathcal{C}) &\mapsto (R_{\texttt{Location}}[A^{\texttt{Location}}, A^{\texttt{Segment}}] \ \textit{where} \ \texttt{Location} \nearrow \texttt{Segment}). \\
(\texttt{Gender} \in \mathcal{C}) &\mapsto (R_{\texttt{Gender}}[A^{\texttt{Gender}}, A^{\texttt{Segment}}] \ \textit{where} \ \texttt{Gender} \nearrow \texttt{Segment}). \\
(\texttt{Segment} \in \mathcal{C}) &\mapsto (R_{\texttt{Segment}}[A^{\texttt{Segment}}, A^{\texttt{All}}] \ \textit{where} \ \texttt{Segment} \nearrow \texttt{All}). \\
(\texttt{All} \in \mathcal{C}) &\mapsto (R_{\texttt{All}}[A^{\texttt{All}}]).
\end{aligned}
$$

Rule (I) also specifies the referential constraint between the child category table and its parents tables. In snowflake schema, each tuple of table R_c stores the direct parents of an element belonging to c. For the sake of database integrity, each attribute representing a parent of the category c should refer to the first attribute in the parent table. In our case, the following referential constraints must hold in the snowflake database of Figure 3.3:

$$
\begin{aligned}
\forall x, y, z \ (R_{\texttt{Customer}}(x, y, z) &\rightarrow (\exists \ w \ R_{\texttt{Location}}(y, w))). \\
\forall x, y, z \ (R_{\texttt{Customer}}(x, y, z) &\rightarrow (\exists \ w \ R_{\texttt{Gender}}(z, w))). \\
\forall x, y \ (R_{\texttt{Location}}(x, y) &\rightarrow (\exists \ z \ R_{\texttt{Segment}}(y, z))). \\
\forall x, y \ (R_{\texttt{Gender}}(x, y) &\rightarrow (\exists \ z \ R_{\texttt{Segment}}(y, z))). \\
\forall x, y \ (R_{\texttt{Segment}}(x, y) &\rightarrow (R_{\texttt{All}}(y))).
\end{aligned}
$$

The last step of MD2R mapping is translating the dimension instance. Here, we must apply rule (II) to all of the members of the set $\mathcal{M}$ (see Example 2.1). Since the methodology is the same for each element, we only demonstrate how the first tuple in each table is obtained:

$$\begin{array}{rcl}
(\mathtt{C}_1 \in \mathcal{M} \ \mathit{and} \ \delta(\mathtt{C}_1) = \mathtt{Customer}) & \mapsto & (R_{\mathtt{Customer}}(\mathtt{C}_1, \mathtt{L}_1, \mathtt{M}) \ \mathit{where} \ \mathtt{C}_1 < \mathtt{L}_1 \ \mathit{and} \ \mathtt{C}_1 < \mathtt{M}) \\
(\mathtt{L}_1 \in \mathcal{M} \ \mathit{and} \ \delta(\mathtt{L}_1) = \mathtt{Location}) & \mapsto & (R_{\mathtt{Location}}(\mathtt{L}_1, \mathtt{S}_2) \ \mathit{where} \ \mathtt{L}_1 < \mathtt{S}_2) \\
(\mathtt{M} \in \mathcal{M} \ \mathit{and} \ \delta(\mathtt{M}) = \mathtt{Gender}) & \mapsto & (R_{\mathtt{Gender}}(\mathtt{M}, \mathrm{NULL})) \\
(\mathtt{S}_1 \in \mathcal{M} \ \mathit{and} \ \delta(\mathtt{S}_1) = \mathtt{Segment}) & \mapsto & (R_{\mathtt{Segment}}(\mathtt{S}_1, \mathtt{all}) \ \mathit{where} \ \mathtt{S}_1 < \mathtt{all}) \\
(\mathtt{all} \in \mathcal{M} \ \mathit{and} \ \delta(\mathtt{all}) = \mathtt{All}) & \mapsto & (R_{\mathtt{All}}(\mathtt{all}))
\end{array}$$

Notice the NULL value in the tuple generated for element M. Since this element has no parent in category Segment, the attribute representing its parent $A^{\mathtt{Segment}}$ is set to NULL

.

Figure 3.4 shows the hierarchical structure of the snowflake database for the MDDB in Example 1.1. It can be seen that, the fact table structure in this Figure is similar to the one in Figure 3.2.

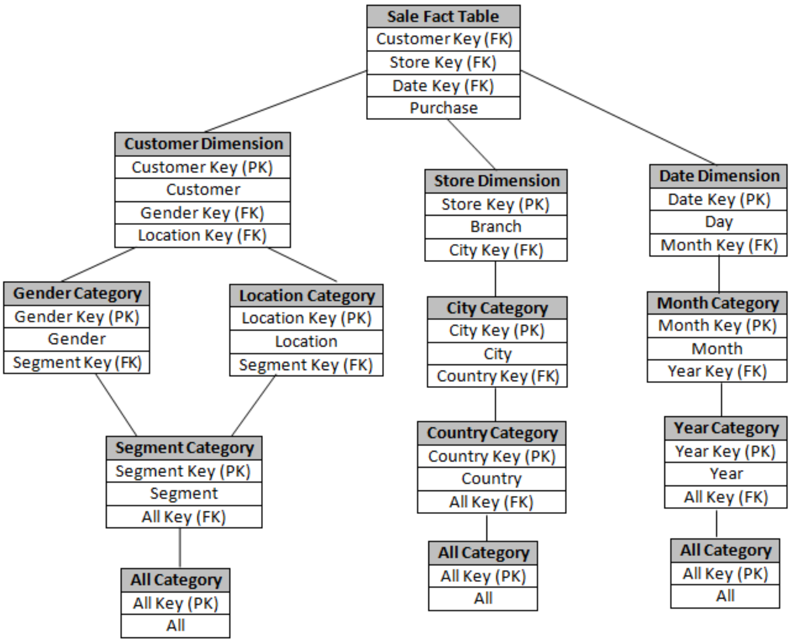


Figure 3.4: Example 1.1 implemented as snowflake database

3.5 Why Snowflake is not a Good Choice?

Now that snowflake mapping is precisely formalized, we evaluate this MD2R mapping to see if it can be used in our approach. Our discussions show that, snowflake schema fails to meet the criteria introduced in Chapter 1 for an expressive MDDB relational implementation.

3.5.1 Summarizability Constraints in Snowflake Schema

In this section, we show that although we can detect heterogeneity through `NOT NULL` constraints over snowflake database, we can not check the case of non-strictness in this database through ICs efficiently. This fact makes this schema an inappropriate choice for our approach.

Strictness

The hierarchical structure of the snowflake schema results in some complications in checking strictness through ICs. Since each category is mapped to a single table, *several joins* must be executed in order to check strictness. Example 3.4 demonstrates this complicated process.

Example 3.4. (example 2.4 continued) Assume that we want to check strictness between categories `Customer` and `Segment`, i.e. `Customer` → `Segment`. Based on the Definition 2.3, for each element of `Customer` category, there should be one grand parent in the `Segment` category. In the snowflake schema of Figure 3.3, the only way to find these grand parents is by joining the intermediate tables.

However, there are two ways to reach category `Segment` from category `Customer`. One goes through `Location`, and the other passes `Gender`. Strictness implies that, no matter how `Customer` rolls up to `Segment`, there should be one parent for each base element.

For finding the grand parents reached through `Location` category, we must execute three join operations, $R_{\texttt{Customer}} \bowtie R_{\texttt{Location}} \bowtie R_{\texttt{Segment}}$. The results of this query are { ($\texttt{C}_1$, $\texttt{L}_1$, $\texttt{S}_2$), ($\texttt{C}_2$, $\texttt{L}_2$, $\texttt{S}_1$), ($\texttt{C}_3$, $\texttt{L}_3$, $\texttt{S}_2$)}.

The same procedure must be applied, in order to find the grand parents reached through `Gender` category. Here, the query is $R_{\texttt{Customer}} \bowtie R_{\texttt{Gender}} \bowtie R_{\texttt{Segment}}$. The set of tuples retrieved as the answer is { ($\texttt{C}_2$, `F`, $\texttt{S}_1$), ($\texttt{C}_3$, `F`, $\texttt{S}_1$)}.

Strictness implies that, after merging these query results, there should be at most one distinct `Segment` value for each element of the `Customer` category. If we consider each of these result sets as a separate temporary table, there should be an *equality generating dependency* (EGD) [3] (cf. Section 4.5.1) between attributes `Customer` and `Segment` of these two tables.

It can be checked that, in the merged set, $\texttt{C}_1$ is related to $\texttt{S}_2$, $\texttt{C}_2$ corresponds to $\texttt{S}_1$, and $\texttt{C}_3$ is related to both $\texttt{S}_1$ and $\texttt{S}_2$ in the `Segment` category. Here, element $\texttt{C}_3$ is violating the equality generating dependency, and hence the strictness constraint is not satisfied. □

The aforementioned EGD can be expressed in the relational calculus by universally quantified sentences of the form

$$\forall \bar{x}(\varphi(\bar{x}) \rightarrow x_j^1 = x_j^2), \tag{3.2}$$

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

In general, there is no simple and unique mapping strategy for expressing strictness in terms of relational ICs on top of snowflake database. As Example 3.4 shows, checking strictness in snowflake schema might not be a straightforward process. In fact, this complication depends on the structure of the dimension schema, and the categories between which strictness is imposed. For example, in order to check whether the relation between categories `Customer` and `Location` is strict, we need to execute one simple join on the corresponding tables.

The general approach for checking strictness of a dimension is to perform the procedure elaborated in Example 3.4, for *every pair of paths* between *any two categories.* An inefficiency of this process is that, if categories `Customer` and `Location` have strict relationship, and the relation between `Location` and `Segment` is also strict, we can not infer that `Customer` and `Segment` have a strict relation.

Based on the above discussions, we can not efficiently check strictness through ICs over snowflake database, since the hierarchical structure of this schema causes some complications in this process.

Homogeneity

Unlike strictness, homogeneity can efficiently be checked in snowflake schema. When the relation between categories c_i and c_j is covering, $c_i \Rightarrow c_j$, the attribute A^{c_j} in table R_{c_i} should not take any `NULL` value (rule (III)). Imposing this constraint is due to the fact that, in snowflake schema, `NULL` references resemble *missing parents* (assuming that the original MD instance does not contain null values). Hence, homogeneity constraint between two categories can be checked using `NOT NULL` constraints over snowflake database (cf. Section 4.5.2).

(III) (NOT NULL Generation)

$$(c_i \Rightarrow c_j) \mapsto_{Snowflake} (\texttt{NOT NULL } R_{c_i}.A^{c_j}).$$

The `NOT NULL` constraint can be expressed in relational calculus as follows:

$$\forall \bar{x}(\psi(\bar{x}) \rightarrow \mathit{NotNULL}(x_j)), \qquad (3.3)$$

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` .

Example 3.5. (example 2.6 continued) In this example, we generate the set of integrity constraints needed for checking homogeneity in the `Customer` dimension, through the snowflake database of Figure 3.3. According to Definition 2.4, homogeneity should be checked only for those pairs of categories that are directly connected to each other through an edge.

Based on rule (III), the set of constraints in $FH^{\mathcal{S}}$ is represented by the following set of ICs:

$$\begin{aligned}\texttt{NOTNULL}\quad & \{R_{\texttt{Customer}}.A^{\texttt{Gender}},\ R_{\texttt{Customer}}.A^{\texttt{Location}},\ R_{\texttt{Gender}}.A^{\texttt{Segment}}, R_{\texttt{Location}}.A^{\texttt{Segment}},\\ & R_{\texttt{Segment}}.A^{\texttt{All}}\} \end{aligned} \tag{3.4}$$

It can be easily checked that, the `Customer` dimension in Figure 1.4 is not homogeneous, since the constraint `NOT NULL` $R_{\texttt{Gender}}$.`Segment` is violated in the snowflake database of Figure 3.3. □

3.5.2 Invertibility of Snowflake Mapping

Re-generating a multidimensional instance from a snowflake database is an easy and straightforward process. In fact, the hierarchical structure of the snowflake database simplifies mapping inversion. Here, each relational table is mapped to a category in the dimension schema. The edges between these categories can be retrieved from the referential constraints between the attributes (rule (IV)).

Instance mapping inversion has a similar procedure. Each tuple stores an element and its direct parents. Hence, the set of elements, and the links between them can be easily generated from a snowflake database instance (rule (V)). Based on our discussions, the snowflake mapping inversion is formalized as follows.

(IV) For each attribute A appearing in some $R \in \mathcal{R}$, create a category (name) $c^A \in \mathcal{C}$. The set of so-created categories is denoted with $\mathcal{C}$.

For each relational predicate $R_c[A_1, \cdots, A_n]$, create an edge from c^{A_1} to c^{A_i}, for $2 \leq i \leq n$, in the dimension schema.

(V) For each relational tuple $R_c(e_1, \cdots, e_n)$, with $R_c[A_1,\ldots, A_n]$, add element e_1 to the set of elements $\mathcal{M}$, where $\delta(e_1){=}c^{A_1}$, and insert an edge from e_1 to e_i, where $e_i \neq$ `NULL` , for $2 \leq i \leq n$, in the dimension instance.

Let's see how the relational tables in the snowflake schema of Figure 3.3 represent a dimension schema:

$$\begin{aligned}
R_{\texttt{Customer}}[A^{\texttt{Customer}}, A^{\texttt{Location}}, A^{\texttt{Gender}}] &\mapsto \texttt{Customer} \in \mathcal{C}, \texttt{Customer} \nearrow \texttt{Location},\\
&\quad\ \texttt{Customer} \nearrow \texttt{Gender}\\
R_{\texttt{Location}}[A^{\texttt{Location}}, A^{\texttt{Segment}}] &\mapsto \texttt{Location} \in \mathcal{C}, \texttt{Location} \nearrow \texttt{Segment}\\
R_{\texttt{Gender}}[A^{\texttt{Gender}}, A^{\texttt{Segment}}] &\mapsto \texttt{Gender} \in \mathcal{C}, \texttt{Gender} \nearrow \texttt{Segment}\\
R_{\texttt{Segment}}[A^{\texttt{Segment}}, A^{\texttt{All}}] &\mapsto \texttt{Segment} \in \mathcal{C}, \texttt{Segment} \nearrow \texttt{All}\\
R_{\texttt{All}}[A^{\texttt{All}}] &\mapsto \texttt{All} \in \mathcal{C}
\end{aligned}$$

Instance mapping inversion is achieved by applying rule (V) to all of the relational tuples in snowflake database of Figure 3.3. Since the procedure for each tuple is the same, we only show how the first tuple in each table is mapped to its multidimensional counterpart:

$$\begin{aligned}
R_{\texttt{Customer}}(\texttt{C}_1, \texttt{L}_1, \texttt{M}) &\mapsto \texttt{C}_1 \in \mathcal{M},\ \delta(\texttt{C}_1) = \texttt{Customer},\ \texttt{C}_1 < \texttt{L}_1,\ \texttt{C}_1 < \texttt{M}\\
R_{\texttt{Location}}(\texttt{L}_1, \texttt{S}_2) &\mapsto \texttt{L}_1 \in \mathcal{M},\ \delta(\texttt{L}_1) = \texttt{Location},\ \texttt{L}_1 < \texttt{S}_2\\
R_{\texttt{Gender}}(\texttt{M}, \texttt{NULL}) &\mapsto \texttt{M} \in \mathcal{M},\ \delta(\texttt{M}) = \texttt{Gender}\\
R_{\texttt{Segment}}(\texttt{S}_1, \texttt{all}) &\mapsto \texttt{S}_1 \in \mathcal{M},\ \delta(\texttt{S}_1) = \texttt{Segment},\ \texttt{S}_1 < \texttt{all}\\
R_{\texttt{All}}(\texttt{all}) &\mapsto \texttt{all} \in \mathcal{M},\ \delta(\texttt{all}) = \texttt{All}
\end{aligned}$$

It can be easily checked that by continuing the above procedure for all of the relational tuples in the snowflake database, the `Customer` dimension is re-constructed.

Despite this fact, due to its weakness in representing non-strictness at relational level, snowflake is not the perfect choice for our approach.

3.6 Summary

In this chapter, we showed that neither star nor snowflake schema is the perfect choice for our purpose. In particular, representing homogeneity through ICs and invertibility are the two expressivity properties that star schema does not have. On the other hand, snowflake does not allow efficient detection of non-strictness through ICs. Although the weaknesses of these schemas have been discussed a lot, only a few studies focused on alternatives for MDDB relational implementation [9], or star/snowflake enhancement [48, 50]. None of these relational representations have the expressivity properties mentioned in Chapter 1. This fact leads us to think of a relational reconstruction of the MD instances.

Chapter 4

MDDBs as Path Instances

4.1 Introduction

Our discussions of Chapter 3 show that neither the star nor the snowflake schema is the perfect choice for restoring summarizability through relational repairs. Hence, in this chapter we propose a relational reconstruction of the MD instance that satisfies our criteria, namely: a simple relational representation and verification of MD summarizability conditions via relational ICs, invertibility of the MD2R mapping; and a simple relational repair approach.

The hierarchical structure of dimensions in MDMs is similar to the tree-like structure of the XML documents. Hence, we start by studying the existing approaches for mapping an XML document to a relational database. Inspired by these approaches, we propose a new relational representation for MDDBs, path schema. We give precise formalization for this MD2R mapping. As part of our relational schema proposal, we explain how to translate the multidimensional queries to relational queries posed to path database. We also evaluate the efficiency of checking summarizability constraints through ICs over the new relational schema.

4.2 Dimension Schemas as XML Trees

Due to the similarity between the hierarchical structure of the dimensions in multidimensional data models, and the *tree-like* structure of XML documents, it would be interesting to study the existing mechanisms for representing an XML tree-like structure as a relational model. According to [73], two main approaches exist for storing an XML document in a relational database:

1. *Structure-mapping approach*: In this approach, database schemas are designed based on the *logical structure* or the document type definition (DTD) of the XML document. In [27, 2], a database table is created for each element type in the XML document. In [65], a more sophisticated approach based on a detailed analysis of DTD is presented. A separate relational table is created for those elements that have in-degree 0, or can occur multiple times. Nodes with in-degree 1 are in-lined in the parent nodes' relation.

   In MDDBs, the relationship between parent and child categories in the dimension schema is many-to-many. Hence, if we apply this mapping to MDDBs, according to the approach proposed in [65], each child category should be represented in a single table. More specifically, the result of applying this mapping to a dimension is very much similar to the hierarchical structure of snowflake schema, since for each category in the hierarchy, a separate table is created.

2. *Model-mapping approach*: It generates a *fixed* database schema to store the structure of every XML document. Here, the main problem is how to map basic constructs in the tree model to relational model. In [34], separate relations are used to store edges and leaves of the XML tree. This approach requires many joins for querying the database.

   In [66, 73], nodes receive the same treatment. Here, all paths in the XML tree

> are uniquely identified, and stored in a database table. Moreover, a table is designed for storing all of the inner nodes in the graph, along with their *path identifiers*. Although this approach is independent of the original XML document structure, the XML tree can be easily *reconstructed* from the relational database, using the stored path expressions.
>
> Although this path-based approach works fine for storing XML data in relational databases, it is not applicable to our problem. The reason is that, having a generic database schema for representing all of the dimensions in a MDDB complicates the process of aggregate query answering. Moreover, checking summarizability constraints in this generic relational schema is not a straightforward process.

However, the idea of using path identifiers as in [66, 73], to store the tree structure in a relational database, guarantees full reconstruction of the XML document, i.e. a path-based mapping can assure invertibility. To us, this property seems interesting, since we also expect our MD2R mapping to be invertible.

4.3 Path Schema

As discussed before, the optimal relational representation of MDDB must enable efficient checking of summarizability conditions through ICs. In Section 2.2.1, we explained that these constraints are defined on the basis of the *roll up relation*. This binary relation itself contains elements in the dimension instance, that are connected to each other through a sequence of edges. In other words, the occurrence of a pair of elements in a roll up relation indicates the existence of a *path* between those two elements in the dimension schema.

In order to better express the summarizability constraints as integrity constraints, the optimal relational representation must store the aforementioned paths efficiently.

Inspired by the XML-to-RDB mapping proposed in [66, 73], we suggest a path-based mapping from multidimensional data model to relational model. In order to introduce this mapping, we have to first define some preliminary terms.

Definition 4.1. 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 $1 \leq i \leq n-1$. □

Definition 4.2. 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 $P = \langle\ c_1,\cdots,c_n\ \rangle$, is an *ordered list* of elements $p = \langle\ e_1,\cdots,e_n\ \rangle$[1]. Each p-instance is characterized by the following conditions:

(a) $\delta(e_i) = c_i$, or $e_i =$ `NULL` for $1 \leq i \leq n$.

(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` in positions i by non-`NULL` e_is, that satisfies the first two conditions above.

The set of all instances of the path P is denoted by $Inst^{\mathcal{D}}(P)$. □

In Definition 4.2, condition (c) enforces the use of non-null data elements whenever possible; or, equivalently, the use of `NULL` only when strictly needed. As mentioned before, we assume that `NULL` does not belong to $\mathcal{M}$, hence, it is incomparable via $<$ with dimension instance elements (condition (b)). Notice that, if the base category is non-empty (something natural to assume), then there will be no p-instance starting with `NULL`.

Definition 4.3. A p-instance, $p = \langle\ e_1,\ldots,\ e_n\ \rangle$, is said to be *broken*, if there exists at least one member e_i such that $e_i =$ `NULL` for $2 \leq i \leq n$.

[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.

Note that, based on Definition 4.2, the first element of a p-instance *can not* be NULL. □

Example 4.1. (example 1.4 continued) The Customer dimension schema in Figure 1.4a has two B2A paths:

$$\begin{aligned}
\mathtt{P}_1^{\mathtt{Cus}} &= \langle \mathtt{Customer}, \mathtt{Gender}, \mathtt{Segment}, \mathtt{All} \rangle \\
\mathtt{P}_2^{\mathtt{Cus}} &= \langle \mathtt{Customer}, \mathtt{Location}, \mathtt{Segment}, \mathtt{All} \rangle
\end{aligned}$$

The set of instances for these B2A paths are as follows:

$$\begin{aligned}
\mathit{Inst}^{\mathcal{D}}(\mathtt{P}_1^{\mathtt{Cus}}) &= \{\langle \mathtt{C}_1, \mathtt{M}, \mathtt{NULL}, \mathtt{NULL} \rangle, \langle \mathtt{C}_2, \mathtt{F}, \mathtt{S}_1, \mathtt{all} \rangle, \langle \mathtt{C}_3, \mathtt{F}, \mathtt{S}_1, \mathtt{all} \rangle\} \\
\mathit{Inst}^{\mathcal{D}}(\mathtt{P}_2^{\mathtt{Cus}}) &= \{\langle \mathtt{C}_1, \mathtt{L}_1, \mathtt{S}_2, \mathtt{all} \rangle, \langle \mathtt{C}_2, \mathtt{L}_2, \mathtt{S}_1, \mathtt{all} \rangle, \langle \mathtt{C}_3, \mathtt{L}_3, \mathtt{S}_2, \mathtt{all} \rangle\}
\end{aligned}$$

As can be seen, the first p-instance of $\mathtt{P}_1^{\mathtt{Cus}}$ is broken, since it contains two NULL elements. □

For schema mapping, we consider all of the B2A paths in the dimension schema. Each path is mapped to a *single* database table. The categories along each path are 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.

Similar to the schema mapping, instance mapping is performed by translating each p-instance to a database tuple in the corresponding table. The following rules formalize path mapping (transformation), $\mathcal{T}$ in two sections: part (I) specifies schema mapping rules, and part (II) represents the rules for instance mapping.

(I) 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 $RP[A^{c_1}, \ldots, A^{c_n}]$.

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

As presented before, Figure 4.1 is the result of applying path mapping rules to `Customer` dimension. $\mathtt{P}_1^{\mathtt{Cus}}$ is mapped to the table shown in Figure 4.1a, and the result of mapping $\mathtt{P}_2^{\mathtt{Cus}}$ is the table in Figure 4.1b. For each of these tables, the set of tuples shows the set of p-instances obtained in Example 4.1 for the corresponding B2A path.

	A^{Customer}	A^{Gender}	A^{Segment}	A^{All}
1	C_1	M	NULL	NULL
2	C_2	F	S_1	all
3	C_3	F	S_1	all

(a) Table $RP_1^{\mathtt{Cus}}$ for the left path $\mathtt{P}_1^{\mathtt{Cus}}$ in the `Customer` dimension schema

	A^{Customer}	A^{Location}	A^{Segment}	A^{All}
1	C_1	L_1	S_2	all
2	C_2	L_2	S_1	all
3	C_3	L_3	S_2	all

(b) Table $RP_2^{\mathtt{Cus}}$ for the right path $\mathtt{P}_2^{\mathtt{Cus}}$ in the `Customer` dimension schema

Figure 4.1: `Customer` dimension represented in path schema

The active domain, $\mathit{Act}(D)$, of the generated path instance D is contained in $\mathcal{M} \cup$ `NULL`; and the domain, $\mathit{Dom}(A^c)$, of the generated attribute A^c is $\delta^{-1}(c) \cup$ `NULL`. In other words, $\mathit{Dom}(A^c) \subseteq \mathit{Act}(D) \cup$ `NULL`. Notice that the domain of the attribute corresponding to base category does not include `NULL`, i.e. $\mathit{Dom}(A^{base}) = \delta^{-1}(base)$.

For category `All` , we assume that `all` $\in$ *Dom*($A^{\mathtt{All}}$), since we have `all` $\in \mathcal{M}$. However, $A^{\mathtt{All}}$ might still contain `NULL` value, since category `All` might not be reached by lower-level elements in the given MD instance.

Notice that, the relational schema generated depends only on the MD schema. In particular, the number of path tables generated is equal to the number of B2A paths in the *MD schema*, and not the MD instance.

Theorem 4.1. In each B2A path P of the form $\langle\ c_1,\cdots,c_n\ \rangle$, the set of first elements of all p-instances *F(P)*, is equal to the set of elements belonging to c_1, which is the base category.

$$\{\bigcup_i \{e_{i,1} | \langle e_{i,1}, \cdots, e_{i,n}\rangle \in \mathit{Inst}^{\mathcal{D}}(P)\}\} \quad = \quad \delta^{-1}(c_1)$$

□

Proof of Theorem 4.1: We discussed previously that NULL $\notin F(P)$, and $F(P) \subseteq \delta^{-1}(c_1)$. Now, we will prove by contradiction that $F(P) = \delta^{-1}(c_1)$.

Assume there is an element e in set *F(P)*, such that e does not belong to c_1. So, e might belong to categories other than c_1, or it might be NULL . The first case is in contradiction with the first property of p-instances in Definition 4.2. Moreover, the second situation is not also possible, since according to Definition 4.3, the first element of a p-instance *can not* be NULL .

On the other hand, assume that, there is an element e in c_1, which is not a member of *F(P)*. This statement implies that, e is a base element, which does not belong to any p-instance. This situation happens when there is no outgoing edge from the base element in the dimension instance. However, according to Definition 4.3, this case is considered as a broken p-instance, in which all of the elements except the first one is equal to NULL. Since this situation is formalized as a broken p-instance, e must belong to set *F(P)*. This fact contradicts our initial assumption that e does not belong to *F(P)*. □

As can be seen in Example 4.1, for $\mathtt{P}_1^{\mathtt{Cus}}$ or $\mathtt{P}_2^{\mathtt{Cus}}$, the set of first elements of all p-instances is equal to $\{\mathtt{C}_1, \mathtt{C}_2, \mathtt{C}_3\}$. This set, itself, is equal to the set of elements belonging to category Customer (see Example 2.1).

Figure 4.2 shows how path schema represents the dimensions Customer, Store and Date, and the Sales fact table in Example 1.1. Theorem 4.1 implies that the the set of values the attribute $A^{\mathtt{Customer}}$ takes in the two path tables $RP_1^{\mathtt{Cus}}$ and $RP_2^{\mathtt{Cus}}$ are equal. Hence, the fact table can refer to either of these two attributes. For the sake of database integrity, we also add a referential constraint between the $A^{\mathtt{Customer}}$ of the path tables $RP_1^{\mathtt{Cus}}$ and $RP_2^{\mathtt{Cus}}$, i.e. enforcing the constraint brought in Theorem 4.1.

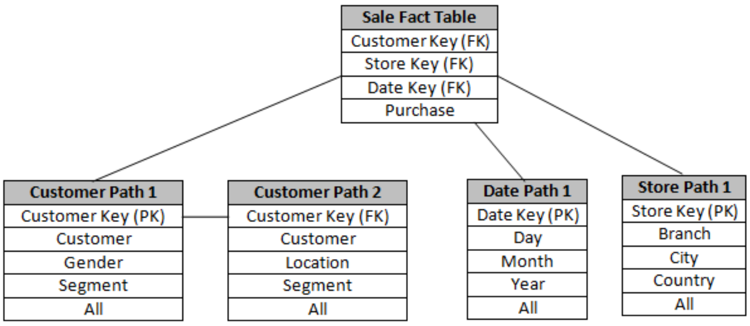


Figure 4.2: Example 1.1 implemented as path database

4.4 Query Answering in Path Schema

In ROLAP systems, queries posed to the MD instance are translated to queries posed to the MDDB relational representation. In this section we start by a brief introduction on the MDDB query language, and then elaborate on how in ROLAPs a MD query is translated to a SQL query, considering star, snowflake and path schemas.

4.4.1 Multidimensional Expressions

Most commonly, the aggregate queries over MDDBs group the query results by the values of a set of attributes (i.e. categories), and then calculate an aggregate function (*f*) for each group. This function could be for example *Min*, *Max*, *Sum* or *Count* [53]. The aggregation is achieved by upward navigation through a category path, i.e. computing the roll up relation. The query *conditions* are imposed on the table obtained from joining the fact table and some roll up relations. These aggregate queries are also known as cube views.

SELECT $A_i, \cdots, A_j, f(A)$ FROM $fact\ table, \mathcal{R}_k, \cdots, \mathcal{R}_m$

WHERE $conditions$ GROUP BY $A_i, \cdots, A_j$

Multidimensional Expressions (MDX) is the query language commonly used for processing data cubes in MDDBs, much like SQL is a query language for relational databases. SQL is designed to retrieve data in two dimensions: a column dimension and a row dimension. Hence, it can not efficiently navigate the cubes of multidimensional databases. In general, certain common forms of multidimensional data analysis are difficult with SQL aggregation constructs [39]. That is why, MDDBs have their own specific query language. MDX is also a calculation language, with syntax similar to spreadsheet formulas. MDX was first introduced as part of the *OLE DB for OLAP* specification in 1997 by Microsoft [58, 67].

The basic form of the two-dimensional query in MDX is as follows:

`SELECT` {*member selection*} `ON AXIS(0)`, {*member selection*} `ON AXIS(1) FROM` [*source cube*]

The query syntax uses some keywords that are common in SQL, such as `SELECT` and `WHERE`. Even though there are apparent similarities in the two languages, there are also significant differences. A prominent difference is that the output of an MDX query, which uses a cube as a data source, is another cube, whereas the output of an SQL query (which uses a columnar table as a source) is typically columnar.

Each cube view is built upon a pre-computed cube. The keyword `FROM` is used to specify the base cube for computing a cube view in MDX. A cube view consists of one or more axes, each identified using the `AXIS` keyword. The query above defines two, `AXIS(0)` and `AXIS(1)`. Basically, each axis of a cube view contains a set of elements. Normally, one of the axes of a cube view is used to represent the fact data (measure). Besides, MDX allows us to place any dimension from the source cube onto any axis of the query's result cube.

We elaborate more on the MDX syntax in the following example.

Example 4.2. (example 1.2 continued) Consider the base cube in Example 1.2 (which corresponds to the fact table in Figure 1.1) and the `Store`, `Date` and `Customer` dimensions in Figures 1.2, 1.3 and 1.4a, respectively. Suppose we want to compare the monthly purchases made by our female and male customers during year 2010 in our Canadian stores. The following MDX query computes this cube view:

`SELECT {[Measures].[Sale]}  ON AXIS(0),`

`{[Customer].[All].[Segment].[Gender].MEMBERS}  ON AXIS(1),`

`{[Store].[All].[Country].[CANADA]}  ON AXIS(2)`

`{[Date].[All].[Year].[2010].CHILDREN}  ON AXIS(3),`

`FROM` $\mathit{BaseCube}^{\mathtt{Sales}}$

The keyword Measures resembles the set of our fact data. In our running example (Example 1.1), we introduced only one measure for our MDM, Sales. Hence, in our case, the set [Measures] contains only one element. In the above MDX query, we choose the only member of this set, Sales, for the first axis of the cube view.

The second axis of this cube view should represent the set of elements belonging to Gender category in the Customer dimension, i.e. F and M. To this end, we have to first navigate in the Customer dimension schema to reach the category Gender. MDX provides a top-down navigation in the dimension hierarchy starting from the topmost category, All. In the above query, the expression [Customer].[All].[Segment].[Gender] resembles the navigation from category All to category Gender in the Customer dimension. The second step is to place the members of category Gender on the second axis. In MDX, the keyword MEMBERS is used to retrieve the elements/members of a given category. We use this keyword to put elements F and M on AXIS(1).

On the third axis of this cube view, we should represent a set with a single member, the country CANADA. To achieve this, our first step is to navigate to the category Country in the Store dimension. The expression [Store].[All].[Country] on AXIS(2) performs this navigation. The term [CANADA] narrows down the elements of category Country to country CANADA. In other words, the expression [Store].[All].[Country].[CANADA] resembles the element CANADA in the Store dimension.

On the fourth axis, we need the set of months in year 2010. Similar to the previous axes, we first use the expression [Date].[All].[Year] to reach the desired category in the Date dimension. By applying the term [2010] to the expression [Date].[All].[Year], we obtain the element 2010 in the `Date` dimension. The months in year 2010 are simply the children of this element, and they can be retrieved by using the `CHILDREN` keyword on the element 2010.

Notice that, the above cube view is built upon $\mathit{BaseCube}^{\texttt{Sales}}(\texttt{Customer}, \texttt{Branch}, \texttt{Day})$ of Example 1.2, as the base data cube. The computation of this cube view involves finding the roll up relations $\mathcal{R}_{\texttt{Customer}}^{\texttt{Gender}}$, $\mathcal{R}_{\texttt{Branch}}^{\texttt{Country}}$ and $\mathcal{R}_{\texttt{Day}}^{\texttt{Year}}$, and calculating the aggregate function `SUM` for the fact data, `Sales`. □

4.4.2 MD Queries as SQL Queries over Path Database

In ROLAP systems, MDX queries posed to a MDDB must be translated into SQL queries to be posed to the MDDB relational representation. This translation depends on the schema of the MDDB relational representation. Depending on the relational schema used to represent the MDDB, the SQL query corresponding to a MD query has different structures. In this section, we elaborate on how MDX queries are translated to SQL queries over path database. We also provide a comparison between star, snowflake and path schemas in this query translation. The experiments supporting our discussions are brought in Chapter 8.

In the path database, the format of this SQL query varies depending on the cube view. In other words, the structure of the SQL query depends on the categories used in the MDX query. A category might belong to more than one B2A paths in the dimension schema. This category is mapped to two or more attributes in separate tables. So, depending on the category queried on each axis of the cube view, we might have to search one or more path tables. This explains why, in the path schema, we do not have a fixed SQL query structure for all MDX queries.

Example 4.3. (example 4.2 continued) Assume we want to find the total purchases made by our female and male customers. The MDX query posed to the MDDB is shown in part (a) below. Based on Figure 4.1, `Gender` appears in only one table in the path schema, $RP_1^{\texttt{Cus}}$. Hence, we can retrieve MDX query results by simply joining $RP_1^{\texttt{Cus}}$ with the fact table of Figure 1.1. Part (b) shows the translated query into SQL.

```
(a) SELECT {[Measures].[Sale]} ON AXIS(0),
    {[Customer].[All].[Segment].
    [Gender].MEMBERS} ON AXIS(1),
    {[Date].[All].[all]} ON AXIS(2),
    {[Store].[All].[all]} ON AXIS(3)
    FROM  BaseCube^Sales
```

```
(b) SELECT A^Gender, SUM(Purchase)
    FROM Sale-Fact-Table, RP_1^Cus
    GROUP BY A^Gender
```

□

Example 4.4. (example 4.3 continued) In this example, we are interested in analyzing the profitability of each of our customer segments. Through this analysis, companies can set their marketing strategies more efficiently.

The MDX query for finding the total product sale for each customer segment is shown in part (a) below. Based on Figure 1.4a, category `Segment` resides on two B2A paths. As a result, it is mapped to two separate attributes in the path schema of Figure 4.1.

Due to the fact explained above, the structure of the translated query is different from what we had in Example 4.3. In order to find the element in `Segment` category that each `Customer` rolls up to, we have to search both of the path tables. In particular, we have to first find the union of tuples belonging to both path tables on the selected attribute ($A^{\texttt{Segment}}$), and then join this result with the fact table of Figure 1.1. The equivalent SQL query for our example is brought in part (b).

(a) SELECT {[Measures].[Sale]} ON AXIS(0),
{[Customer].[All].[Segment].
MEMBERS} ON AXIS(1),
{[Date].[All].[all]} ON AXIS(2),
{[Store].[All].[all]} ON AXIS(3)
FROM $BaseCube^{\texttt{Sales}}$

(b) SELECT $A^{\texttt{Segment}}$, SUM(Purchase)
FROM Sale-Fact-Table, (
(SELECT $A^{\texttt{Customer}}$, $A^{\texttt{Segment}}$
FROM $RP_1^{\texttt{Cus}}$) UNION
(SELECT $A^{\texttt{Customer}}$, $A^{\texttt{Segment}}$
FROM $RP_2^{\texttt{Cus}}$))
GROUP BY $A^{\texttt{Segment}}$

□

It would be interesting to provide a generic structure for the SQL queries over path database that correspond to MDX queries over MD database. In other words, we would like to formulate the SQL equivalent of MDX queries in a path database. For simpler representation, we assume only two-dimensional cube views in MDX. This MDX-to-SQL formulation is shown below. Although there are other possible options for the equivalent SQL query, the one in part (b) is the optimized version of these alternatives. In this query, the multidimensional results for MDX query are obtained by: 1) performing projection on each path table based on the queried category, 2) finding the union of the selected columns, 3) joining the fact table and the result of union operation, and 4) grouping the tuples based on the given category.

(a) SELECT {[Measures].[*Measure*]} ON AXIS(0),
{[*Dimension*].[*All*]$\cdots$[*Category*].MEMBERS}
ON AXIS(1)
FROM [*Base-Cube*]

(b) SELECT $A^{Category}$, SUM (*Measure*)
FROM *Fact-Table,* (
$\texttt{UNION}_i$
(SELECT $A^{BaseCategory}$, $A^{Category}$
FROM $RP_i^{\texttt{Cus}}$)
) GROUP BY $A^{Category}$

Now that we have precisely translated MD queries into SQL queries over path databases, it is natural to ask how different is this translation in case of star or snowflake schemas. In other words, we need to study the difference of star, snowflake and path database in terms of MDX-to-SQL query formulation (see Chapter 8 for an experimental study on the query answering performance of these schemas).

Example 4.5. (example 4.4 continued) In this example, we show the SQL equivalent queries for the MDX query given in Example 4.4 part (a), as posed to the star (part (c)) and the snowflake (part (d)) schemas. As we can see, for the star schema, we need to join the fact table `Sale-Fact-Table`) in Figure 1.1 with the table for the `Customer` dimension R^{Star} in Figure 3.1.

A comparison of query (b) in Example 4.4 and query (c) in this example reveals that, the former requires an inner query for retrieving the specified roll up relation $\mathcal{R}_{\texttt{Customer}}^{\texttt{Segment}}$, while this complication is avoided in the latter. For this reason, query (c) is expected to execute faster than query (b).

We discussed in Section 3.4 that query answering in the snowflake schema requires several joins. In relational databases, the join operation is known for being costly, slowing down the query answering process. Hence, the snowflake schema is expected to have the worst performance in query answering compared to the star and path schemas.

The SQL version of query (a) in Example 4.4 over a snowflake database is shown below (query (d)). In this case, in order to compute the roll up $\mathcal{R}_{\texttt{Customer}}^{\texttt{Segment}}$, we have to traverse both of the paths that lead from `Customer` to `Segment` in the dimension schema. Each path is traversed via a series of join operations, and the results of each path are then merged together. It is clear that the number of joins in query (d) is considerably higher than the number of joins in queries (b) or (c).

(c) `SELECT` $A^{\texttt{Segment}}$, `SUM(Purchase)`
`FROM Sale-Fact-Table,` R^{Star}
`GROUP BY` $A^{\texttt{Segment}}$

(d) `SELECT` $A^{\texttt{Segment}}$, `SUM(Purchase)`
`FROM Sale-Fact-Table, (`
`(SELECT` $A^{\texttt{Customer}}$, $A^{\texttt{Segment}}$
`FROM` $R_{\texttt{Customer}}$, $R_{\texttt{Gender}}$, $R_{\texttt{Segment}}$)
`UNION`
`(SELECT` $A^{\texttt{Customer}}$, $A^{\texttt{Segment}}$
`FROM` $R_{\texttt{Customer}}$, $R_{\texttt{Location}}$, $R_{\texttt{Segment}}$))
`GROUP BY` $A^{\texttt{Segment}}$

□

4.5 Summarizability Constraints in the Path Schema

In order to be able to take advantage of the relational repairs, we must express the summarizability constraints in terms of relational integrity constraints. In this section, we will show that, unlike star and snowflake, we can efficiently check strictness and homogeneity in path database. Given an MD instance $\mathcal{D}$ and a set $\mathcal{K}$ of (local) strictness and homogeneity constraints as those in Definitions 2.3 and 2.4, we will define the set of corresponding ICs, Σ, such that the violation of $\mathcal{K}$ by $\mathcal{D}$ is reflected by the violation of Σ in the underlying path database D.

4.5.1 Strictness

Assume we want to check strictness between categories c_i and c_j, that is $c_i \rightarrow c_j$. According to Definition 2.3, every element of c_i should roll up to at most one element in c_j. In dimension schema, there might be more than one possible path for rolling up the elements in c_i to their parents in c_j. Strictness implies that, no matter which path is taken, there should be no more than one parent for each element in c_i.

Each *local path* between categories c_i and c_j belongs to a separate B2A path in the dimension schema. Hence, it can be said that, checking strictness between categories c_i and c_j in path database depends on the *number* of B2A paths these categories reside in. In other words, strictness must be checked within each single path, and also among all paths from c_i to c_j. Hence, we need separate integrity constraints for each local path, and also a constraint to assure strictness among each pair of paths.

Strictness within each local path between categories c_i and c_j can be checked by a *functional dependency* between the attributes, A^{c_i} and A^{c_j} in the corresponding path table. In addition, in order to assure strictness among each pair of paths, we must impose a functional dependency on the union of tuples in both path tables. More specifically, in the set of tuples belonging to both of these path tables, each value of A^{c_i} is related to exactly one value of A^{c_j}. This constraint can be expressed using *equality generating dependencies* [3]. The EGDs we need here are much simpler than those needed for snowflake schema (cf. Section 3.5.1).

Definition 4.4. [3, 10, 31] In relational databases, an equality generating dependency between two relational tables is formalized as follows:

$$R_1.X_1 = R_2.X_2 = R_1.Y_1 = R_2.Y_2 \tag{4.1}$$

In Equation 4.1, $R1$ and R_2 are two relational tables. X_i and Y_i are attributes of table R_i (i=1,2). For these attributes, we have $dom(X_1) \cap dom(X_2) \neq \emptyset$, and $dom(Y_1) \cap dom(Y_2)$ $\neq \emptyset$. A database instance satisfies the above constraint, if the following expression is always true.

$$\forall t_1, t_2((R_1(t_1) \wedge R_2(t_2)) \quad \rightarrow \quad ((t_1[X_1] = t_2[X_2]) \rightarrow (t_1[Y_1] = t_2[Y_2])))$$

□

Basically, equality generating dependencies say that, if a certain pattern of entries appears, then a certain equality must hold. Functional dependencies are considered as a special case of EGDs.

Based on the above discussion, the following rules map strictness to ICs in path database. Rule (III) shows how to check strictness for each local path between categories c_i and c_j, using functional dependencies. Rule (IV) represents strictness for every pair of these local paths with an equality generating dependency.

(III) (FD generation)

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

(IV) (EGD generation)

$(c_i \rightarrow c_j) \mapsto_{path} \{(RP_m.A^{c_i}{=}RP_n.A^{c_i} \rightarrow RP_m.A^{c_j} = RP_n.A^{c_j}) \mid P_m, P_n \text{ is a pair of B2A paths with } c_i,\, c_j \in P_m \cap P_n\}.$ [2]

Notice that rule (III) can be obtained as a special case of Rule (IV).

Example 4.6. (examples 1.4 and 2.4 continued) In this example, we generate the set of integrity constraints needed for checking strictness in the `Customer` dimension, through the path database of Figure 4.1. According to Definition 2.3, strictness should be checked for every pair of categories in the dimension schema. As mentioned before, we are not concerned with checking strictness between `All` and other categories. Using rule (III), the set $FS^{\mathcal{S}}$ is translated to the following set of functional dependencies:

$$RP_1^{\texttt{Cus}} \;:\; \{A^{\texttt{Customer}} \rightarrow A^{\texttt{Gender}}, A^{\texttt{Customer}} \rightarrow A^{\texttt{Segment}}, A^{\texttt{Gender}} \rightarrow A^{\texttt{Segment}}\}.$$

$$RP_2^{\texttt{Cus}} \;:\; \{A^{\texttt{Customer}} \rightarrow A^{\texttt{Location}}, A^{\texttt{Customer}} \rightarrow A^{\texttt{Segment}}, A^{\texttt{Location}} \rightarrow A^{\texttt{Segment}}\}.$$

The above set can be minimized to the following *irreducible* set of FDs:

$$RP_1^{\texttt{Cus}} \;:\; \{A^{\texttt{Customer}} \rightarrow A^{\texttt{Gender}}, A^{\texttt{Gender}} \rightarrow A^{\texttt{Segment}}\}. \tag{4.2}$$

$$RP_2^{\texttt{Cus}} \;:\; \{A^{\texttt{Customer}} \rightarrow A^{\texttt{Location}}, A^{\texttt{Location}} \rightarrow A^{\texttt{Segment}}\}. \tag{4.3}$$

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

Except for (`Customer`,`Segment`) and (`Customer`,`All`), all pairs of category in the `Customer` dimension reside in only one B2A path. Since `All` is a special category with only one element, `all`, there is no need to check strictness between `Customer` and `All` through different paths. Hence, `Customer` and `Segment` are the only categories that must be applied to rule (IV).

$$RP_1^{\texttt{Cus}}.A^{\texttt{Customer}} = RP_2^{\texttt{Cus}}.A^{\texttt{Customer}} \rightarrow RP_1^{\texttt{Cus}}.A^{\texttt{Segment}} = RP_2^{\texttt{Cus}}.A^{\texttt{Segment}}. \tag{4.4}$$

□

According to mapping rule (II), a broken p-instance generates `NULL` values in the path database. As a result, we might be faced with evaluation of the above ICs against instances containing `NULL` . This is not a straightforward process, since several semantics exist for relational database with null values. In this thesis, we use a single null value, `NULL` , with similar behavior to the one in commercial database management systems. In order to characterize IC evaluation in presence of such a null value, we use the logic-based semantics proposed in [17].

In this study, a logical reconstruction of the relational database D is suggested. The basis of this reconstruction is categorizing database attributes into *relevant* and *irrelevant* groups. Attributes appearing in a given IC, ψ, are considered as relevant, and others as irrelevant. By restricting D to its relevant attributes, a new relational instance D^{Rel}, is obtained. The next step is to rewrite ψ as a new first-order sentence ψ^N, that takes into account the possible occurrence of `NULL` . Based on this approach, evaluating ψ against D is semantically equivalent to evaluation of ψ^N against D^{Rel}, where `NULL` is treated as any other constant [3].

$$D \models_N \psi \ :\Longleftrightarrow \ D^{Rel} \models \psi^N. \tag{4.5}$$

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

A^{Customer}	A^{Segment}
C_1	NULL
C_2	S_1
C_3	S_1

(a) $RP_1^{\texttt{Cus},Rel}$

A^{Customer}	A^{Segment}
C_1	S_2
C_2	S_1
C_3	S_2

(b) $RP_2^{\texttt{Cus},Rel}$

Figure 4.3: D^{Rel}: Path database restricted to relevant attributes for evaluating EGD

Here, $\models_N$ denotes the (new) notion of IC satisfaction in databases in presence of `NULL`. On the right-hand side of equation 4.5, we have usual first-order satisfaction, in which `NULL` is considered as a constant.

Example 4.7. (example 4.6 continued) In this example, we demonstrate how to evaluate EGD (4.4) against the path database D, in Figure 4.1. By rewriting this IC as a usual first-order sentence, we obtain ψ:

$$\forall c\ \forall g\ \forall s\ \forall a\ \forall l\ \forall s'\ \forall a'((RP_1^{\texttt{Cus}}(c,g,s,a) \wedge RP_2^{\texttt{Cus}}(c,l,s',a')) \rightarrow s = s'). \qquad (4.6)$$

The above first-order expression does not take into account the presence of `NULL` with its intended semantics. In order to logically reconstruct D, we have to restrict it to attributes relevant for evaluating ψ. The following set contains these attributes for our example:

$$Rel = \{RP_1^{\texttt{Cus}}.A^{\texttt{Customer}}, RP_1^{\texttt{Cus}}.A^{\texttt{Segment}}, RP_2^{\texttt{Cus}}.A^{\texttt{Customer}}, RP_2^{\texttt{Cus}}.A^{\texttt{Segment}}\} \qquad (4.7)$$

Figure 4.3 shows D^{Rel} for our example.

As the next step, ψ is rewritten as ψ^N, which is imposed to D^{Rel}. This transformation takes the relevant attributes into account, and the possible presence of `NULL` in them.

$$\begin{aligned}\forall c\ \forall s\ \forall s'((RP_1^{\texttt{Cus},Rel}(c,s)\ \wedge\ RP_2^{\texttt{Cus},Rel}(c,s')\ \wedge\ NotNULL(c)\\ \wedge\ NotNULL(s)\ \wedge\ NotNULL(s'))\ \rightarrow\ s = s'). \qquad (4.8)\end{aligned}$$

As explained before, the *NotNULL* is a built-in predicate, that is true only when its argument is (symbolically) different from `NULL` . It is easy to check that, for $c =\texttt{C}_3$, $s =\texttt{S}_1$ and $s' =\texttt{S}_2$, ψ^N is not true in D^{Rel}. In other words, the third tuples of tables $RP_1^{\texttt{Cus}}$ and $RP_2^{\texttt{Cus}}$ are violating this rule. Due to the aforementioned violation and based on rule (4.5), we have $D \nvDash_N \psi$, which is in line with the local non-strictness of the `Customer` dimension.

On the other hand, since FDs are a special form of EGDs, it is easy to check that, based on the given semantics, the set of FDs in 4.2 and 4.3 are not violated in our example. □

4.5.2 Homogeneity

Assume we want to check homogeneity between category c_i and its parent c_j, that is $c_i \Rightarrow c_j$. According to Definition 2.4, every element of c_i should have at least one direct parent in c_j. If there is no parent for an element in c_i , the p-instance to which this element belongs to will be *broken*.

Using the path mapping rule (II), a broken p-instance (see Definition 4.3) in a dimension results in `NULL` values for some attributes in the generated database tuple. Now that we have characterized the evaluation of ICs corresponding to strictness (FDs and EGDs) in presence of `NULL`, homogeneity can be checked in path database through avoiding these `NULL` values.

Consider P as the B2A path that categories c_i and c_j belong to, $P = \langle c_1,\cdots,c_i,c_j,\cdots,c_n \rangle$. Based on Definition 4.2, in any instance of path P, $p = \langle e_1,\cdots,e_n \rangle$, e_{i+1} is the parent for element e_i. If the roll up relation between categories c_i and c_j is not total, e_j will be equal to `NULL` in at least one of the instances of path P.

In order to check homogeneity between categories c_i and c_j, we must first find all of the B2A paths that these categories belong to. If any of these paths has a broken instance, the relation between these categories is heterogeneous. Rule (V) shows how to detect such heterogeneous relations using NOT NULL constraints.

(V) (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 of the ICs introduced in (III)-(V) can be easily written as first-order sentences of the forms (3.2) or (3.3) (as in Example 4.7).

Example 4.8. (examples 1.4 and 2.6 continued) In this example, we generate the set of integrity constraints needed for assuring homogeneity in the Customer dimension, through the path database of Figure 4.1. Unlike strictness, homogeneity should be checked only for those pairs of categories that are directly connected to each other through an edge.

According to Example 2.6, except for (Segment,All), all such pairs of categories belong to one B2A path in the Customer dimension. For these pairs, the NOT NULL constraint is imposed to either of the tables $RP_1^{\texttt{Cus}}$ or $RP_2^{\texttt{Cus}}$. However, based on rule (V), homogeneity between categories Segment and All should be checked in both tables of Figure 4.1.

$$\texttt{NOTNULL}\ RP_1^{\texttt{Cus}}.\{\texttt{Gender}, \texttt{Segment}, \texttt{All}\}. \tag{4.9}$$

$$\texttt{NOTNULL}\ RP_2^{\texttt{Cus}}.\{\texttt{Location}, \texttt{Segment}, \texttt{All}\}. \tag{4.10}$$

According to Definition 4.2, the first element of p-instance never takes value NULL . Hence, we do no need to impose the NOT NULL constraint to the first attribute of path tables.

Based on the path database of Figure 4.1, the constraints NOT NULL $\{RP_1^{\texttt{Cus}}.\texttt{Segment}, RP_1^{\texttt{Cus}}.\texttt{All}\}$ are violated in the first tuple of table $RP_1^{\texttt{Cus}}$. □

4.6 Summary

This chapter showed that, path schema has the first property of an expressive relational representation for MDDBs. Now that we have successfully translated the multidimensional database and summarizability constraints into their relational counterpart, we can move on to the second phase of our approach, which is repairing an inconsistent path instance. We expect to repair path database via simple relational repair operations.

Chapter 5

Repairing Path Instances

5.1 Introduction

In the previous chapter, we discussed how to represent dimension schema, dimension instance and summarizability conditions in relational databases using path mapping. This mapping assures that non-summarizability at multidimensional level causes some inconsistencies in the path database. This is where the idea of relational repairs comes into the picture. In this chapter, we aim at restoring the path database consistency using conventional database repair approaches.

Problem Statement 5.1. Given a path relational schema $\mathcal{R}$ and an instance D, which is inconsistent with respect to a set of integrity constraints Σ; what is the appropriate semantics for obtaining a repaired path instance, D', where $D' \models \Sigma$. In particular, what are the appropriate relational repair operations for the path instance; and, how are the minimal repairs characterized (based on the notion of distance between path instances). □

5.2 Relational Repair Operation

Repairs of relational database instances have been systematically investigated (cf. [12, 26]). Generally, there are three main approaches for repairing inconsistent relational databases:

- S-repair, which minimizes under *set inclusion* the set of insertions or deletions of database *tuples* [4].
- C-repair, which minimizes the *cardinality* of the set of whole *tuples*, by which the repaired instance differs from the original database instance [5].
- A-repair, which changes some *attribute values* in existing tuples. This repair mechanism minimizes a numerical aggregation function over the differences between attribute values in the original tuples and their repaired versions [71]. Typically, this function represents the number of attribute value updates, and hence, A-repair minimizes the *number* of attribute value changes [32].

The optimal repair semantics for a problem depends on several criteria, such as the application domain and database properties. Here, we investigate the applicability of the aforementioned relational repair strategies to an inconsistent path database. The integrity constraints we introduced in Section 4.5 are particular cases of *denial constraints*. For these constraints, consistency can be restored through tuple deletions or attribute-level updates [12].

Deleting a tuple from an inconsistent path database is a *feasible* repair mechanism, i.e. it can resolve the inconsistencies. However, this operation might not be the best solution in all cases. As discussed in Chapter 4, tuples in path database resemble p-instances in the dimension. Tuple deletion in path database implies removing the corresponding p-instance from the dimension, which in some cases leads to a considerable amount of information loss at both relational and multidimensional level.

From an attribute-based perspective, updating the value of an attribute in an inconsistent path database is a *feasible* repair approach. This repair operation does not have the information loss side effect corresponding to tuple deletion.

Example 5.1. (examples 4.6 and 4.8 continued) This example compares attribute-based vs. tuple-based repairs for repairing a path database. Here, we are interested in checking strictness and homogeneity between some, but not all, of the categories in the `Customer` dimension. In other words, we want to have *local* summarizability, as opposed to global summarizability.

Assume we want to check 1) strictness between categories `Location` and `Segment` (`Location` → `Segment`), 2) homogeneity between categories `Gender` and `Segment` (`Gender` ⇒ `Segment`), and 3) strictness between categories `Customer` and `Segment` (`Customer` → `Segment`).

Based on the discussions we had in Section 4.5, the following integrity constraints must be checked to ensure the aforementioned summarizability conditions:

1. $RP_2^{\texttt{Cus}}{:}A^{\texttt{Location}} \rightarrow A^{\texttt{Segment}}$.

2. `NOT NULL` $RP_1^{\texttt{Cus}}.A^{\texttt{Segment}}$.

3. $RP_1^{\texttt{Cus}}{:}\ A^{\texttt{Customer}} \rightarrow A^{\texttt{Segment}}$.

4. $RP_2^{\texttt{Cus}}{:}\ A^{\texttt{Customer}} \rightarrow A^{\texttt{Segment}}$.

5. $RP_1^{\texttt{Cus}}.\ A^{\texttt{Customer}} = RP_2^{\texttt{Cus}}.\ A^{\texttt{Customer}} \rightarrow RP_1^{\texttt{Cus}}.\ A^{\texttt{Segment}} = RP_2^{\texttt{Cus}}.\ A^{\texttt{Segment}}$.

Checking the above ICs in the path database of Figure 4.1 reveals that, constraints (2) and (5) are violated. In particular, the first tuple in table $RP_1^{\texttt{Cus}}$ is violating the `NOT NULL` constraint, and the third tuples of tables $RP_1^{\texttt{Cus}}$ and $RP_2^{\texttt{Cus}}$ are violating the equality generating dependency. Now, we will discuss how to restore the database consistency through attribute updates or tuple deletions.

At attribute level, constraint (2) can be satisfied by updating the $A^{\texttt{Segment}}$ attribute in the violating tuple to $\texttt{S}_2$. Notice that, changing the $A^{\texttt{Segment}}$ value to $\texttt{S}_1$ will violate the strictness constraint between Customer and Segment for element $\texttt{C}_1$ (constraint (5)).

A C-repair or S-repair for this constraint violation would be to delete the violating tuple ($\texttt{C}_1$,M,NULL ,NULL). As a result of this operation, at multidimensional level, the pair ($\texttt{C}_1$,M) will be removed from the set $<$ of the dimension instance.

Constraint (5) is violated in tuples ($\texttt{C}_3$, F,$\texttt{S}_1$,all) and ($\texttt{C}_3$, $\texttt{L}_3$,$\texttt{S}_2$,all). Updating the $A^{\texttt{Segment}}$ value in the first tuple to $\texttt{S}_2$, or changing the $A^{\texttt{Segment}}$ value in the second tuple to $\texttt{S}_1$ are the two possible attribute-based repair operations for this constraint violation.

A tuple-based repair for this inconsistency would be to delete one of the aforementioned violating tuples. Either of these two tuple-based repair operations implies removing the corresponding p-instance from the Customer dimension. For instance, deleting the first violating tuple results in removing the pairs ($\texttt{C}_3$,F), (F,$\texttt{S}_1$), ($\texttt{S}_1$,all) from the set $<$ of the dimension instance. Obviously, we are losing a considerable amount of data by performing this tuple-based repair. □

Based on the above discussions, A-repairs seem to be the most appropriate mechanism for restoring consistency in path database. This class of relational repairs has been used and investigated before [36, 72, 15, 14, 55, 33], specifically for denial constraints in [36, 14], and for FDs in [72, 15].

5.3 Minimality of Relational Repairs

As discussed before, in this thesis we aim at restoring summarizability through relational repairs. An important factor in this process is to issue *minimum* number of changes to both relational and multidimensional databases. It is obvious that, through MD2R mapping inversion, the updates made to the physical database are reflected as changes to the multidimensional instance. Intuitively, a minimal multidimensional repair is obtained by a minimum number of changes to the original dimension. Hence, the relational repair that causes the minimum number of updates to the path database is preferred to other possible repairs. In our approach, we need a cardinality-based repair semantics, as opposed to the set-inclusion-based, which is more common in relational databases [12].

Definition 5.1. 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 $Dom(A) \smallsetminus \{\texttt{NULL}\}$ assigned to attribute A in the database atom $R(t) \in D$.[1]

(b) An *update* on D is a finite set, ρ, 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 ρ, i.e. simultaneously all the updates in ρ, to D is denoted with $\rho(D)$.

(c) For a set Σ of denial constraints (for the schema of D), an update ρ on D is a (minimal) repair if and only if: 1) $\rho(D) \models_N \Sigma$, and 2) there is no ρ', such that $\rho'(D)$ $\models_N \Sigma$, and $|\rho'| < |\rho|$. □

In our case, the set Σ in Definition 5.1 is restricted to relational denial constrains of the form (C), (D), or (E), i.e. just EGDs and `NOT NULL` constraints. In the rest of this thesis, we will rely on this assumption.

According to the above definition, an atomic update changes an existing value in the database by a new non-null value, that is already present in the database. Obviously, this condition does not imply that, the repaired instances are `NULL`-free.

Example 5.2. (examples 4.6 and 4.8 continued) Consider the path database in Example 1.4, and the integrity constraints obtained in Examples 4.6 and 4.8 for checking

[1] 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.

`Customer` dimension summarizability in this database. The followings are candidates to be repairs of D (for simplicity we use the tuple numbers(ids) shown in Figure 4.1):

$$\begin{aligned}
\rho_1^{Path} &= \{\langle RP_1^{\texttt{Cus}}(\mathit{1}), A^{\texttt{Segment}}, \texttt{S}_2\rangle,\ \langle RP_1^{\texttt{Cus}}(\mathit{1}), A^{\texttt{All}}, \texttt{all}\rangle,\ \langle RP_2^{\texttt{Cus}}(\mathit{3}), A^{\texttt{Segment}}, \texttt{S}_1\rangle\}\\
\rho_2^{Path} &= \{\langle RP_1^{\texttt{Cus}}(\mathit{1}), A^{\texttt{Segment}}, \texttt{S}_2\rangle,\ \langle RP_1^{\texttt{Cus}}(\mathit{1}), A^{\texttt{All}}, \texttt{all}\rangle,\ \langle RP_1^{\texttt{Cus}}(\mathit{3}), A^{\texttt{Gender}}, \texttt{M}\rangle,\\
&\quad\ \langle RP_1^{\texttt{Cus}}(\mathit{3}), A^{\texttt{Segment}}, \texttt{S}_2\rangle\}\\
\rho_3^{Path} &= \{\langle RP_1^{\texttt{Cus}}(1), A^{\texttt{Segment}}, \texttt{S}_1\rangle,\ \langle RP_1^{\texttt{Cus}}(\mathit{1}), A^{\texttt{All}}, \texttt{all}\rangle,\ \langle RP_2^{\texttt{Cus}}(1), A^{\texttt{Segment}}, \texttt{S}_1\rangle\\
&\quad\ \langle RP_2^{\texttt{Cus}}(\mathit{3}), A^{\texttt{Segment}}, \texttt{S}_1\rangle\}\\
\rho_4^{Path} &= \{\langle RP_1^{\texttt{Cus}}(1), A^{\texttt{Segment}}, \texttt{S}_2\rangle,\ \langle RP_1^{\texttt{Cus}}(\mathit{1}), A^{\texttt{All}}, \texttt{all}\rangle,\ \langle RP_1^{\texttt{Cus}}(2), A^{\texttt{Segment}}, \texttt{S}_2\rangle\\
&\quad\ \langle RP_1^{\texttt{Cus}}(\mathit{3}), A^{\texttt{Segment}}, \texttt{S}_2\rangle,\ \langle RP_2^{\texttt{Cus}}(\mathit{2}), A^{\texttt{Segment}}, \texttt{S}_2\rangle\}\\
\rho_5^{Path} &= \{\langle RP_1^{\texttt{Cus}}(1), A^{\texttt{Segment}}, \texttt{S}_1\rangle,\ \langle RP_1^{\texttt{Cus}}(\mathit{1}), A^{\texttt{All}}, \texttt{all}\rangle,\ \langle RP_1^{\texttt{Cus}}(2), A^{\texttt{Gender}}, \texttt{M}\rangle\\
&\quad\ \langle RP_1^{\texttt{Cus}}(\mathit{3}), A^{\texttt{Segment}}, \texttt{S}_2\rangle,\ \langle RP_2^{\texttt{Cus}}(\mathit{1}), A^{\texttt{Segment}}, \texttt{S}_1\rangle\}\\
\rho_6^{Path} &= \{\langle RP_1^{\texttt{Cus}}(1), A^{\texttt{Segment}}, \texttt{S}_1\rangle,\ \langle RP_1^{\texttt{Cus}}(\mathit{1}), A^{\texttt{All}}, \texttt{all}\rangle,\ \langle RP_1^{\texttt{Cus}}(2), A^{\texttt{Segment}}, \texttt{S}_2\rangle\\
&\quad\ \langle RP_1^{\texttt{Cus}}(\mathit{3}), A^{\texttt{Segment}}, \texttt{S}_2\rangle,\ \langle RP_2^{\texttt{Cus}}(\mathit{1}), A^{\texttt{Segment}}, \texttt{S}_1\rangle,\ \langle RP_2^{\texttt{Cus}}(\mathit{2}), A^{\texttt{Segment}}, \texttt{S}_2\rangle\}
\end{aligned}$$

All of these updates restore the consistency of the path database. However, ρ_1^{Path} has the minimum cardinality among the above sets. Hence, according to Definition 5.1, ρ_1^{Path} is the only minimal repair for restoring consistency in the path database of Figure 4.1. This repair updates attribute $A^{\texttt{Segment}}$ in the third tuple of $RP_2^{\texttt{Cus}}$ from $\texttt{S}_2$ to $\texttt{S}_1$. The effect of this update on the corresponding MD side is to change the link from element $\texttt{L}_3$ to category `Segment`. This repair also assigns element $\texttt{S}_2$ as the parent of element `M`, by updating the `NULL` values in the first tuple of table $RP_1^{\texttt{Cus}}$. The impact of relational repair operations at the MD level is discussed in more detail in the next chapter.

Obviously, the aforementioned MD update is not the only possible approach for repairing the `Customer` dimension. Another possible repair could be to modify the parent of element $\texttt{C}_3$ from `F` to `M`, and also create an edge from element `M` to $\texttt{S}_2$. This

MD update corresponds to relational repair ρ_2^{Path}. It can be seen that, although this repair is not minimal, it can restore summarizability at MD level.

Changing the parent of element $\texttt{C}_3$ from $\texttt{L}_3$ to $\texttt{L}_2$ will also resolve non-strictness in the Customer dimension. Together with the insertion of a link between elements M and $\texttt{S}_2$, this MD repair corresponds to the following relational update:

$$\begin{aligned}\rho' \quad = \quad & \{\langle RP_1^{\texttt{Cus}}(\mathit{1}), A^{\texttt{Segment}}, \texttt{S}_2\rangle,\ \langle RP_1^{\texttt{Cus}}(\mathit{1}), A^{\texttt{All}}, \texttt{all}\rangle,\ \langle RP_2^{\texttt{Cus}}(\mathit{3}), A^{\texttt{Location}}, \texttt{L}_2\rangle,\\ & \langle RP_2^{\texttt{Cus}}(\mathit{3}), A^{\texttt{Segment}}, \texttt{S}_1\rangle\}\end{aligned}$$

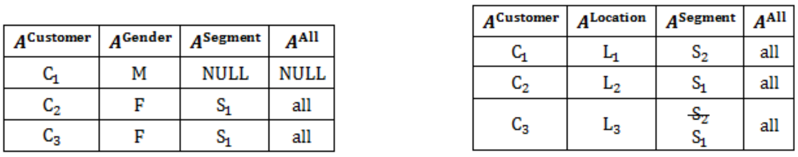
Notice that ρ' performs an *unnecessary* update on $A^{\texttt{Location}}$ in the third tuple of $RP_2^{\texttt{Cus}}$, compared to ρ_1^{Path}. Hence, based on Definition 5.1, ρ' is not considered as a relational repair. In other words, although the aforementioned MD update restores summarizability in the Customer dimension, it does not have minimal effects on the relational side. (cf. Section 7.3) □

As mentioned before, repaired path instances might not be necessarily NULL-free. NULL values are updated as a result of NOT NULL constraint violation. Since we might originally be interested in checking homogeneity between some, but not all, parent-child categories, we might have attributes that are not restricted by the NOT NULL constraint. Hence, the existence of NULL values in the repaired instance depends on the scope of homogeneity constraint. That is, if the set Σ of relational denial constraints is derived from a non-full set of homogeneity constraints on the corresponding MD schema (cf. Definition 2.4(b)), a relational repair wrt Σ may still have NULLs.

Example 5.3. (example 4.8 continued) Unlike Example 4.8, here, we are interested in checking homogeneity constraint only between categories Customer and Location, i.e. Customer $\Rightarrow$ Location. In other words, we want to impose *local* as opposed to *global* homogeneity. In this case, according to mapping rule (V), the only NOT NULL constraint generated is NOT NULL $RP_2^{\texttt{Cus}}.A^{\texttt{Location}}$, but not all of the ICs in sets (4.9) and (4.10) of

Example 4.8. This single `NOT NULL` constraint is satisfied in the path instance of Figure 4.1.

Assuming that the EGDs and FDs for checking strictness are the same as those generated in Example 4.6, the path instance can be minimally repaired as shown in Figure 5.1. The `NULL` values in the first tuple of table $RP_1^{\texttt{Cus}}$ are not updated, since they do no violate any of the imposed `NOT NULL` constraints.



A^{Customer}	A^{Gender}	A^{Segment}	A^{All}
C_1	M	NULL	NULL
C_2	F	S_1	all
C_3	F	S_1	all

(a) Repaired $RP_1^{\texttt{Cus}}$

A^{Customer}	A^{Location}	A^{Segment}	A^{All}
C_1	L_1	S_2	all
C_2	L_2	S_1	all
C_3	L_3	~~S_2~~ S_1	all

(b) Repaired $RP_2^{\texttt{Cus}}$

Figure 5.1: An example of a repaired path instance containing `NULL`

Consider the case where we want to impose the (non-official) homogeneity constraint between `Customer` and `Segment`. According to Definition 2.4, homogeneity constraints of the form $c_i \Rightarrow c_j$ require that $c_i \nearrow c_j$ holds in the schema, i.e. a single link connects c_i to c_j. Hence, in our case, we have to check the following local homogeneity constraints: `Customer` $\Rightarrow$ `Gender`, `Gender` $\Rightarrow$ `Segment`, `Customer` $\Rightarrow$ `Location`, `Location` $\Rightarrow$ `Segment`. These local constraints would be translated to the following ICs: `NOT NULL` $\{RP_1^{\texttt{Cus}}.A^{\texttt{Gender}}$, $RP_1^{\texttt{Cus}}.A^{\texttt{Segment}}$, $RP_2^{\texttt{Cus}}.A^{\texttt{Location}}$, $RP_2^{\texttt{Cus}}.A^{\texttt{Segment}}$ $\}$. It can be easily checked that, in this case, the first tuple of table $RP_1^{\texttt{Cus}}$ is violating the constraint `NOT NULL` $RP_1^{\texttt{Cus}}.A^{\texttt{Segment}}$. Hence, the repaired instance would contain a non-null value for attribute $A^{\texttt{Segment}}$ in this table. However, the `NULL` value for column $A^{\texttt{All}}$ in table $RP_1^{\texttt{Cus}}$ would remain unchanged in the repaired instance. □

Unlike homogeneity, ICs checking strictness are not violated by the `NULL` values. The semantics introduced in Section 4.5.1 for evaluating FDs and EGDs in presence of `NULL` values reveals that, `NULL`s are not a source of violation for these ICs. As a result, restoring strictness through relational repairs does not modify the NULL values in the path database.

Example 5.4. (example 4.7 continued) In order to clarify the above discussion, consider the EGD (4.4) obtained from the MD strictness constraint Customer → Segment, and the first-order sentence (4.8), obtained in Example 4.7, for checking the satisfaction of this IC in the presence of NULL.

In order to see the effect of NULL on the evaluation of strictness constraints, we instantiate the universal sentence (4.8) on the first tuples of tables $RP_1^{\texttt{Cus}}$ and $RP_2^{\texttt{Cus}}$, obtaining:

$$\begin{aligned}((RP_1^{\texttt{Cus},Rel}(\texttt{C}_1,\texttt{NULL}) \ \wedge \ RP_2^{\texttt{Cus},Rel}(\texttt{C}_1,\texttt{S}_2) \ \wedge \ \mathit{NotNULL}(\texttt{C}_1) \\ \wedge \ \mathit{NotNULL}(\texttt{NULL}) \ \wedge \ \mathit{NotNULL}(\texttt{S}_2)) \ \rightarrow \ \texttt{NULL} = \texttt{S}_2).\end{aligned}$$

Due to the occurrence of NULL in relevant attributes, the antecedent of the above implication has the false conjunct, *NotNULL* (NULL). Hence, the whole sentence becomes true. In other words, although in the instance of Figure 4.1, $\texttt{C}_1$ is associated to both NULL and $\texttt{S}_2$, the EGD (4.4) is not violated. This is due to the fact that NULL was introduced as an *auxiliary relational* element to represent missing parents; and it does not appear on the MD side.

In general, having NULL in the relevant attributes of rule (4.8) makes the *NotNULL* predicate false, and hence the whole implication becomes true. So, it can be concluded that, NULL values do not violate EGD (4.4). Since FDs are a special form of EGD, we can generalize our statement; the ICs for assuring strictness in path database are not violated by NULL values. Recall from Example 4.7 that, the aforementioned EGD is violated in the third tuples of tables $RP_1^{\texttt{Cus}}$ and $RP_2^{\texttt{Cus}}$. □

In general, the repairs introduced in Definition 5.1 do not generate any new NULLs, either for satisfying constraint (III) or (IV). Every NULL value is counted as a source of heterogeneity in the dimension. By generating new `NULL` values, we are creating another instance of homogeneity violation in the dimension. Obviously, generating such instances is undesirable, even if imposing homogeneity to the whole dimension is not our concern at the moment.

It should be clear by now that, in general, repairs of a relational path instance associated to a MD instance will 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 5.1. For a relational path instance D and a set Σ of relational ICs associated to an MD instance $\mathcal{D}$ with a set $\mathcal{K}$ of MD strictness and homogeneity constraints, there always exists a minimal repair wrt Σ. □

Proof: It suffices to build a path instance D' obtained by an update ρ applied to D, such that $D' \models_N \Sigma$. If such an update ρ 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, $v^{A^c} \in \mathit{Dom}(A^c) \smallsetminus \texttt{NULL}$. Assuming that every category c has at least one element e for a given instance, the set $\mathit{Dom}(A^c) \smallsetminus \texttt{NULL}$ is not empty.

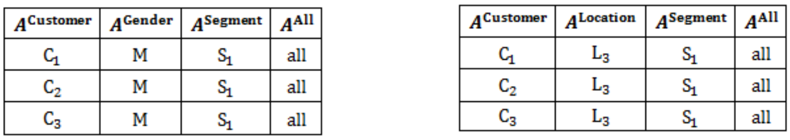
The relational update ρ contains a set of atomic updates such that, in each table $RP_i^{\texttt{Cus}}$ that A^c appears, the value of this attribute is updated to v^{A^c} in all of the tuples. We can show that, the instance resulted from applying ρ to D, $D' = \rho(D)$, satisfies ICs of the form (C)-(E),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

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 (Note that according to Definition 4.2, the first attribute of a path table cannot be

$\texttt{NULL}$). Moreover, because of the unique value selected for each attribute, the FDs and EGDs of the form (C) and (D) cannot be violated. In particular, FDs are satisfied because the value of each attribute (except the first one) in each table is the same in all of the tuples. EGDs can not be violated since the attributes shared between several path tables (except the first attribute) has the same value is all tables. 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 5.1), we can conclude that, there is always a minimal relational repair for the inconsistent path instance D. □

Example 5.5. (example 5.2 continued) Figure 5.2 shows one possible repair obtained as indicated in the proof of Theorem 5.1. In this example, we assign the following values to each of the attributes in the path instance of Figure 4.1: $v^{A^{\texttt{Gender}}} = \texttt{M}$, $v^{A^{\texttt{Location}}}$ $= \texttt{L}_3$, $v^{A^{\texttt{Segment}}} = \texttt{S}_1$ and $v^{A^{\texttt{All}}} = \texttt{all}$. Notice that, for the shared attribute $A^{\texttt{Segment}}$, we select one value to be applied to both tables. It can be easily checked that, the path instance in Figure 5.2 satisfies the denial constraints defined in Examples 4.6 and 4.8.



A^{Customer}	A^{Gender}	A^{Segment}	A^{All}
C_1	M	S_1	all
C_2	M	S_1	all
C_3	M	S_1	all

(a) A repair for $RP_1^{\texttt{Cus}}$

A^{Customer}	A^{Location}	A^{Segment}	A^{All}
C_1	L_3	S_1	all
C_2	L_3	S_1	all
C_3	L_3	S_1	all

(b) A repair for $RP_2^{\texttt{Cus}}$

Figure 5.2: A non-minimal repair according to Theorem 5.1

Comparing ρ to the minimal repair ρ_1^{Path} obtained in Example 5.2 reveals that, the former performs eight attribute updates while the latter executes only three updates. As a result, according to Definition 5.1, ρ can not be considered as a minimal relational repair. □

5.4 Summary

This chapter provided the semantics for obtaining the set of minimal repairs for an inconsistent path database. In particular, we defined attribute-based updates as the relational repair operation, and used cardinality-based comparison in order to find minimal repairs.

Now that we have successfully repaired the path database, it is natural to ask about the kind of MD repairs obtained through this relational approach. We will address this question in Chapter 7. Before that, we should propagate the relational changes made to path database up to the multidimensional layer. In order to obtain multidimensional repairs, our next step is inversion of path mapping. In other words, we should translate back the repaired path instance into a new MD instance.

Chapter 6

Back to MD Instances: Inverting Path Repairs

6.1 Introduction

In the previous chapter, we explained how to repair an inconsistent path database based on the given integrity constraints for assuring summarizability. Now, it is the time to retrieve a new summarizable dimension from the repaired relational instance. To achieve this goal, we must study the invertibility of the MD2R mapping, $\mathcal{T}$, introduced in Chapter 4. Any information loss in mapping the MDDB to the path database results in an incomplete retrieval of a dimension from the repaired path instance [30].

Problem Statement 6.1. Given a path schema $\mathcal{R}$ and a repaired path database D, what are the mapping inversion rules for obtaining a dimension schema $\mathcal{S}$ and a repaired dimension instance, $\mathcal{D}$. □

6.2 Schema Mapping Inversion

A schema mapping is a specification that describes how data structured under one schema is to be transferred to another schema. The first schema is referred to as source, and the second one is called target schema. Recently, a lot of attention has been paid to the specification and manipulation of schema mappings [11, 57]. Merge, composition and inverse are among the operators defined for manipulating the schema mappings [11, 29].

An inverse of a schema mapping $\mathcal{T}^{-1}$ is an ideal mapping to bring the data exchanged through $\mathcal{T}$ back to the source. Intuitively, existence of such mapping depends on whether or not information is lost through $\mathcal{T}$ [29, 30].

In general, in schema mapping, we do not assume to have a single source, or a single target schema. In other words, there is a many-to-many relationship between the source and target instances. This assumption complicates the semantics for inverse operator.

The standard algebraic definition of inverse operator is given in [57]. According to this definition, if a pair of instances ($\mathcal{S}_1$, $\mathcal{S}_2$) are related to each other through a schema mapping, the inverse of this mapping contains the pair ($\mathcal{S}_2$, $\mathcal{S}_1$). However, this definition was intended for a generic model management context, and it has some weaknesses in the schema mapping context [29].

In [29], the notion of schema mapping inversion is based on another algebraic property of inverses, that the composition of a function with its inverse is the identity mapping. In particular, if a mapping $\mathcal{T}^{-1}$ is an inverse of a mapping $\mathcal{T}$, then $\mathcal{T} \circ \mathcal{T}^{-1}$ should be the identity mapping. By identity mapping, we mean the one in which every element is mapped to itself. A more precise formalization of this notion can be found in [7].

In our case, the MD2R mapping $\mathcal{T}$ has two components, the schema and the instance transformations; the former also including a transformation of a set of constraints. We expect this mapping to have an inverse $\mathcal{T}^{-1}$ with the following properties:

1. The mapping $\mathcal{T}$ should be *uniquely* invertible. In MD2R mapping $\mathcal{T}$, there is a one-to-one relationship between the multidimensional instances and relational instances. In other words, for each source instance, there is only one target instance. Hence, we expect to obtain a unique multidimensional instance by applying $\mathcal{T}^{-1}$ to the relational instance. Recall from Chapter 3 that, one of the weaknesses of star schema was its inability to meet this criterion, i.e. the inversion of star database could generate more than one multidimensional instance.

2. The composition of the MD2R mapping $\mathcal{T}$ with its inverse $\mathcal{T}^{-1}$ should generate the identity mapping. In particular, it should hold that $\mathcal{T}^{-1}\ (\mathcal{T}\ (\mathcal{S})) = \mathcal{S}$, where $\mathcal{S}$ is the dimension schema, and $\mathcal{T}^{-1}\ (\mathcal{T}\ (\mathcal{D})) = \mathcal{D}$, where $\mathcal{D}$ is the dimension instance.

3. Inverting the path schema and instance, i.e. computing $\mathcal{T}^{-1}\ (\mathcal{R})$ and $\mathcal{T}^{-1}\ (D)$ should be done in polynomial time in the sizes of the relational schema $\mathcal{R}$ and the relational instance D.

4. If $\mathcal{T}^{-1}$ is applied to a repaired path instance, the resulting MD instance should satisfy the original summarizability constraints $\mathcal{K}$. More precisely, since for a repair ρ, it holds $\rho(D) \models \Sigma$, where Σ is the set of ICs corresponding to $\mathcal{K}$, we expect that $\mathcal{T}^{-1}\ (\rho(D)) \models \mathcal{K}$.

In the next section, we will first determine the domain of $\mathcal{T}^{-1}$, then specify the transformation rules in $\mathcal{T}^{-1}$, and finally show that $\mathcal{T}^{-1}$ meets the aforementioned invertibility criteria.

6.3 Inverting the Path Mapping

Inverting the path mapping $\mathcal{T}$ is a simple process. More precisely, the mapping $\mathcal{T}^{-1}$ is defined on a triples $\langle\ \mathcal{R},\ \Sigma,\ D\ \rangle$, where D is an instance over the path schema $\mathcal{R}$, and Σ is a set of ICs such that:

1. For every relation in $\mathcal{R}$, such as $R[A^{c_1}, \cdots, A^{c_n}]$, it holds that $A^{c_n} = A^{\texttt{All}}$. In other words, the last attribute of all of the relations in $\mathcal{R}$ is the same. In addition, the relations in $\mathcal{R}$ also share the first attribute, A^{c_1}. This is due to the assumption of a single base category in the MD instance. Notice that, relations in $\mathcal{R}$ may share attributes other than A^{c_1} and A^{c_n} as well.

2. For every relation in $\mathcal{R}$, such as $R[A^{c_1}, \cdots, A^{c_n}]$, it holds that $\texttt{NULL} \notin Dom(A^{c_1})$, $\texttt{NULL} \in Dom(A^{c_i})$ $(2 \leq i \leq n)$, and $Dom(A^{c_n}) = \{\texttt{NULL}, \texttt{all}\}$.

3. For every tuple in D, such as $R(e_1, \cdots, e_n)$, if $e_i = \texttt{NULL}$, it holds that $e_j = \texttt{NULL}$ for $i < j \leq n$.

4. The ICs in Σ are of the form a) $\texttt{NOT NULL}\ R_i.A_j$ or b) $R_i.A_k = R_j.A_l \rightarrow R_i.A_m = R_j.A_n$, where R_i and R_j are two not necessarily distinct relations in $\mathcal{R}$.

5. The relational instance D does not necessarily satisfy Σ. Although, in our approach, it holds that $D \models \Sigma$ (since D is a repaired path instance); in general, for the definition of $\mathcal{T}^{-1}$, we do not assume that the constraints in Σ are all satisfied by D.

Intuitively, inverting the path schema is a *straightforward* process. The set of attributes of all path tables resembles the set of categories for the dimension schema. Since the list of attributes in each table represents a B2A path in the dimension schema, the edges between these categories can be retrieved according to Definition 4.1 (rule (VI) defining $\mathcal{T}^{-1}$ $(\mathcal{R})$).

Similarly, the inversion of instance mapping, $\mathcal{T}^{-1}$ (D) can be done by considering each tuple as a p-instance in the dimension instance. Each p-instance might add new elements, or links to the dimension instance (rule (VII)). Obviously a *unique dimension* is obtained as a result of this mapping inversion. In other words, there is no uncertainty in any stage of path mapping inversion. This mapping inversion is formalized as follows.

(VI) For each attribute A appearing in some $R \in \mathcal{R}$, create a category (name) $c^A \in \mathcal{C}$. The set of so-created categories is denoted with $\mathcal{C}$.

For each relational predicate $R[A_1,\ldots, A_n]$, create an edge from c^{A_i} to $c^{A_{i+1}}$ for $1 \leq i \leq n-1$, in the dimension schema.

(VII) For each relational tuple $\mathrm{R}(e_1,\ldots, e_n)$, with $R[A_1,\ldots, A_n]$, if $e_i \neq \texttt{NULL}$, add e_i to the set of elements $\mathcal{M}$ for $1 \leq i \leq n$, where $\delta(e_i)$=c^{A_i}, and insert an edge from e_i to e_{i+1} for $1 \leq i \leq n-1$, in the dimension instance, where e_i,$e_{i+1} \neq \texttt{NULL}$.

Example 6.1. (example 5.2 continued) Figure 6.1a shows the repaired path database according to the minimal repair obtained in Example 5.2. Using the above inversion rules, this repaired database instance is mapped to the summarizable dimension in Figure 6.1b.

Inversion of schema mapping can be achieved by applying rule (VI) to the relational tables of Figure 4.1[1]:

$$RP_1^{\texttt{Cus}}[A^{\texttt{Customer}}, A^{\texttt{Gender}}, A^{\texttt{Segment}}, A^{\texttt{All}}] \mapsto \{\texttt{Customer}, \texttt{Gender}, \texttt{Segment}, \texttt{All}\} \subseteq \mathcal{C},$$

$$\texttt{Customer} \nearrow \texttt{Gender}, \texttt{Gender} \nearrow \texttt{Segment},\ \texttt{Segment} \nearrow \texttt{All}.$$

$$RP_2^{\texttt{Cus}}[A^{\texttt{Customer}}, A^{\texttt{Location}}, A^{\texttt{Segment}}, A^{\texttt{All}}] \mapsto \{\texttt{Customer}, \texttt{Location}, \texttt{Segment}, \texttt{All}\} \subseteq \mathcal{C},$$

$$\texttt{Customer} \nearrow \texttt{Location}, \texttt{Location} \nearrow \texttt{Segment},\ \texttt{Segment} \nearrow \texttt{All}.$$

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

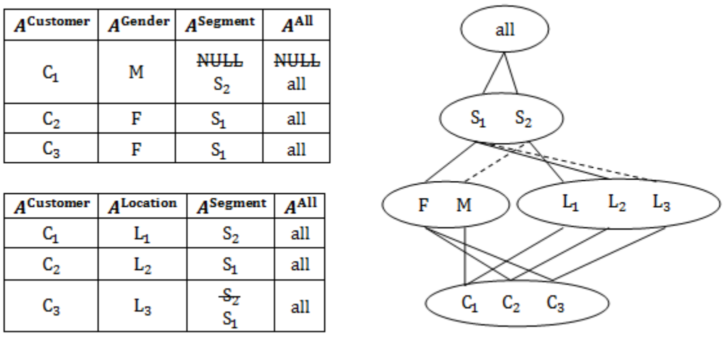
By applying rule (VII) to the set of tuples in each path table, the dimension instance in Figure 6.1b is generated.

$$
\begin{aligned}
RP_1^{\texttt{Cus}}(\texttt{C}_1,\texttt{M},\texttt{S}_2,\texttt{all}) &\mapsto \{\texttt{C}_1,\texttt{M},\texttt{S}_2,\texttt{all}\} \subseteq \mathcal{M},\ \delta(\texttt{C}_1)=\texttt{Customer},\ \delta(\texttt{M})=\texttt{Gender},\\
&\quad\ \delta(\texttt{S}_2)=\texttt{Segment},\ \delta(\texttt{all})=\texttt{All},\ \texttt{C}_1<\texttt{M}<\texttt{S}_2<\texttt{all}.\\
RP_1^{\texttt{Cus}}(\texttt{C}_2,\texttt{F},\texttt{S}_1,\texttt{all}) &\mapsto \{\texttt{C}_2,\texttt{F},\texttt{S}_1,\texttt{all}\} \subseteq \mathcal{M},\ \delta(\texttt{C}_2)=\texttt{Customer},\ \delta(\texttt{F})=\texttt{Gender},\\
&\quad\ \delta(\texttt{S}_1)=\texttt{Segment},\ \delta(\texttt{all})=\texttt{All},\ \texttt{C}_2<\texttt{F}<\texttt{S}_1<\texttt{all}.\\
RP_1^{\texttt{Cus}}(\texttt{C}_3,\texttt{F},\texttt{S}_1,\texttt{all}) &\mapsto \{\texttt{C}_3,\texttt{F},\texttt{S}_1,\texttt{all}\} \subseteq \mathcal{M},\ \delta(\texttt{C}_3)=\texttt{Customer},\ \delta(\texttt{F})=\texttt{Gender},\\
&\quad\ \delta(\texttt{S}_1)=\texttt{Segment},\ \delta(\texttt{all})=\texttt{All},\ \texttt{C}_3<\texttt{F}<\texttt{S}_1<\texttt{all}.\\
RP_2^{\texttt{Cus}}(\texttt{C}_1,\texttt{L}_1,\texttt{S}_2,\texttt{all}) &\mapsto \{\texttt{C}_1,\texttt{L}_1,\texttt{S}_2,\texttt{all}\} \subseteq \mathcal{M},\delta(\texttt{C}_1)=\texttt{Customer},\ \delta(\texttt{L}_1)=\texttt{Location},\\
&\quad\ \delta(\texttt{S}_2)=\texttt{Segment},\ \delta(\texttt{all})=\texttt{All},\ \texttt{C}_1<\texttt{L}_1<\texttt{S}_2<\texttt{all}.\\
RP_2^{\texttt{Cus}}(\texttt{C}_2,\texttt{L}_2,\texttt{S}_1,\texttt{all}) &\mapsto \{\texttt{C}_2,\texttt{L}_2,\texttt{S}_1,\texttt{all}\} \subseteq \mathcal{M},\delta(\texttt{C}_2)=\texttt{Customer},\ \delta(\texttt{L}_2)=\texttt{Location},\\
&\quad\ \delta(\texttt{S}_1)=\texttt{Segment},\ \delta(\texttt{all})=\texttt{All},\ \texttt{C}_2<\texttt{L}_2<\texttt{S}_1<\texttt{all}.\\
RP_2^{\texttt{Cus}}(\texttt{C}_3,\texttt{L}_3,\texttt{S}_1,\texttt{all}) &\mapsto \{\texttt{C}_3,\texttt{L}_3,\texttt{S}_1,\texttt{all}\} \subseteq \mathcal{M},\delta(\texttt{C}_3)=\texttt{Customer},\ \delta(\texttt{L}_3)=\texttt{Location},\\
&\quad\ \delta(\texttt{S}_1)=\texttt{Segment},\ \delta(\texttt{all})=\texttt{All},\ \texttt{C}_3<\texttt{L}_3<\texttt{S}_1<\texttt{all}.
\end{aligned}
$$

This example shows that, path instance is uniquely invertible to a MD instance using rules (VI) and (VII). In other words, there is a one-to-one relationship between the relational (source) instance and multidimensional (target) instance (property (1) of invertibility).

Furthermore, it can be easily checked that the generated dimension in Figure 6.1b is summarizable, i.e. globally strict and homogeneous. Since it holds $D \models \Sigma$ for the path instance of Figure 6.1a, it follows that $\mathcal{D} \models \mathcal{K}$ for the repaired `Customer` dimension. This fact is in compliance with property (4) of invertibility mentioned in the previous section. □

Theorem 6.1. For a relational instance D over a path schema $\mathcal{R}$, computing $\mathcal{T}^{-1}$ $(\mathcal{R})$ and $\mathcal{T}^{-1}$ (D) takes polynomial time in the sizes of the path schema and instance



$A^{Customer}$	A^{Gender}	$A^{Segment}$	A^{All}
C_1	M	~~NULL~~ S_2	~~NULL~~ all
C_2	F	S_1	all
C_3	F	S_1	all

$A^{Customer}$	$A^{Location}$	$A^{Segment}$	A^{All}
C_1	L_1	S_2	all
C_2	L_2	S_1	all
C_3	L_3	~~S_2~~ S_1	all

(a) Repaired path database

(b) Repaired `Customer` dimension

Figure 6.1: Retrieving a summarizable dimension from the repaired path schema (dashed lines resemble inserted edges)

(property (3) of invertibility).

Proof: We need to show that $\mathcal{T}^{-1}$ $(\mathcal{R})$ and $\mathcal{T}^{-1}$ (D) are computable through rules (VI) and (VII) in polynomial time in the sizes of the path schema and instance. In particular, for rule (VI), assuming that $arity(R_i)$ denotes the number of attributes for relation R_i, the total cost of inverting the schema is equal to $2 * \Sigma_i arity(R_i)$. Here, generating the set of categories $\mathcal{C}$, and converting the attributes of relation R_i to a B2A path in dimension schema have the same cost, which is $\Sigma_i arity(R_i)$. In case of instance mapping (rule (VII)), the total cost would be equal to $\Sigma_i card(R_i) * arity(R_i)$, where the number of tuples for relation R_i is denoted by $card(R_i)$. The term $card(R_i) * arity(R_i)$ is equal to the total time needed for creating the p-instances in the dimension instance from the set of tuples belonging to R_i. In general, the total time needed for inverting a path instance is $2 * \Sigma_i arity(R_i) + \Sigma_i card(R_i) * arity(R_i)$. Hence, it can be concluded that, the MD model is generated in polynomial time in size of the relational schema and instance. □

Example 6.2. (example 6.1 continued) In this example, we verify property (2) mentioned at the beginning of this section for invertibility. Consider the schema of the `Customer` dimension, $\mathcal{S} = \langle \mathcal{C}, \nearrow \rangle$ as formalized in Example 2.1. In order to simplify the calculations, we introduce a smaller dimension instance for this schema, $\mathcal{D} = \langle \mathcal{M}, < \rangle$:

$$\begin{aligned} \mathcal{M} &= \{\mathtt{C_1}, \mathtt{F}, \mathtt{L_1}, \mathtt{S_1}, \mathtt{all}\} \\ < &= \{(\mathtt{C_1}, \mathtt{F}), (\mathtt{C_1}, \mathtt{L_1}), (\mathtt{F}, \mathtt{S_1}), (\mathtt{L_1}, \mathtt{S_1}), (\mathtt{S_1}, \mathtt{all})\} \end{aligned}$$

As elaborated in Example 4.1, the `Customer` dimension is transformed to the relational database of Figure 1.9 via path mapping (rules (I) and (II)):

$$\begin{aligned} \mathcal{T}(\mathcal{S}) = \mathcal{T}(\langle \mathcal{C}, \nearrow \rangle) &= \{RP_1^{\mathtt{Cus}}[\mathtt{Customer}, \mathtt{Gender}, \mathtt{Segment}, \mathtt{All}], \\ &\quad RP_2^{\mathtt{Cus}}[\mathtt{Customer}, \mathtt{Location}, \mathtt{Segment}, \mathtt{All}]\} \\ \mathcal{T}(\mathcal{D}) = \mathcal{T}(\langle \mathcal{M}, < \rangle) &= \{RP_1^{\mathtt{Cus}}(\mathtt{C_1}, \mathtt{F}, \mathtt{S_1}, \mathtt{all}), RP_2^{\mathtt{Cus}}(\mathtt{C_1}, \mathtt{L_1}, \mathtt{S_1}, \mathtt{all})\} \end{aligned}$$

Now, applying rules (VI) and (VII) to the path database of Figure 1.9 would regenerate the `Customer` dimension:

$$\begin{aligned} \mathcal{T}^{-1}(\mathcal{T}(\mathcal{S})) &= \mathcal{T}^{-1}(\{RP_1^{\mathtt{Cus}}[\mathtt{Customer}, \mathtt{Gender}, \mathtt{Segment}, \mathtt{All}], \\ &\quad RP_2^{\mathtt{Cus}}[\mathtt{Customer}, \mathtt{Location}, \mathtt{Segment}, \mathtt{All}]\}) = \mathcal{S} \\ \mathcal{T}^{-1}(\mathcal{T}(\mathcal{D})) &= \mathcal{T}^{-1}(\{RP_1^{\mathtt{Cus}}(\mathtt{C_1}, \mathtt{F}, \mathtt{S_1}, \mathtt{all}), RP_2^{\mathtt{Cus}}(\mathtt{C_1}, \mathtt{L_1}, \mathtt{S_1}, \mathtt{all})\}) = \mathcal{D} \end{aligned}$$

As a result, we have $\mathcal{T}^{-1}$ $(\mathcal{T}$ $(\mathcal{S})) = \mathcal{S}$ and $\mathcal{T}^{-1}$ $(\mathcal{T}$ $(\mathcal{D})) = \mathcal{D}$. In other words, composition of path mapping and its inverse would result in the identity mapping. It can be easily checked that, this property holds for the dimension instance of Figure 1.4b as well. More specifically, the inverse of path mapping introduced by the set of rules (VI) and (VII) is in compliance with the notion of mapping inversion proposed by [29, 7] (property (2)). □

6.4 Summary

This chapter showed that, the mapping from the MD instance to the path database is uniquely invertible. In other words, the path-based approach in moving from multidimensional to relational layer assures no information loss.

Now that we have obtained new summarizable dimensions, we have to talk about the characteristics of these MD repairs that are obtained going through the relational route. Our hypothesis is that, the relational repair semantics proposed in Chapter 5 results in repairs that are minimal from both multidimensional and relational perspectives. We will investigate on this hypothesis in the next chapter.

Chapter 7

A Purely MD Repair Semantics

7.1 Introduction

So far, we have defined *a repair semantics for MD databases wrt summarizability constraints.* However, our approach is *indirect*, i.e. we do not address non-summarizability at MD level. In particular, we introduced a mapping $\mathcal{T}$ to translate a non-summarizable dimension instance $\mathcal{D}$ over a schema $\mathcal{S}$ wrt summarizability constraints $\mathcal{K}$, to its relational counterparts, i.e. a path instance D over relational schema $\mathcal{R}$ which is inconsistent wrt the set of ICs Σ. We introduced the relational repair semantics for obtaining the set of minimal attribute-based repairs for D, i.e. $\mathit{Rep}(D, \Sigma)$. In the previous chapter, we showed that, through applying $\mathcal{T}^{-1}$ to $\mathit{Rep}(D, \Sigma)$, we get a set of MD instances $\mathcal{D}'$ that satisfy $\mathcal{K}$. Here, we will talk about the properties of $\mathcal{D}'$. In particular, we will discuss the kind of repairs that are obtained going through the relational route.

7.2 MD Repair Semantics

Definition 7.1. Let $\mathcal{D}$ be an MD instance, $\mathcal{K}$ be a set of local summarizability constraints, D be the relational instance $\mathcal{T}$ $(\mathcal{D})$, and Σ 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 $D' \in \mathit{Rep}(D, \Sigma)$, such that $\mathcal{D}' = \mathcal{T}^{-1}\ (D')$. $\mathit{Rep}(\mathcal{D}, \mathcal{K})$ denotes the class of path repairs of $\mathcal{D}$ wrt $\mathcal{K}$. □

In our approach, non-summarizability is tackled by repairing the underlying relational database. The relational repairs used here are those that update the *database tuples*, and leave the database schema *unchanged*. As a result, the new dimension obtained from inverting the MD2R mapping has the *same* schema as the non-summarizable original dimension. More specifically, since we do not introduce any new virtual element or category for repairing the dimension, the database repair operations are reflected only as insertion or deletion of edges in the dimension instance. In particular, the relational repairs that address the `NOT NULL` constraint violation, i.e. heterogeneity, result in insertions of edges in $\mathcal{D}$, and those that tackle non-strictness (EGD violation), might cause both insertions and deletions of links. Hence, it can be said that, our approach is a special type of *instance-based* (data-based) repair mechanism. In the next section, we will compare our dimension repairs with results from existing instance-based MD repair approaches like [13, 18, 22, 24, 62].

7.3 Correspondence to Other MD Repairs

In [62, 41], non-summarizability is tackled by adding new virtual elements to the dimension instance. This approach creates virtual parents for elements causing heterogeneity. On the other hand, non-strictness is resolved by combining multiple parents belonging to a single element. For the `Customer` dimension, inserting element $\mathtt{S}'$ as the virtual parent for element `M`, restores homogeneity. Regarding non-strictness, we can merge elements $\mathtt{S}_1$ and $\mathtt{S}_2$ into one virtual element, to resolve the issue of multiple parents. Obviously, this repaired dimension is not comparable to our results; since in our approach, we do not introduce any virtual value either at relational or multidimensional level.

In order to address non-strictness, [22] introduced an approach for repairing the dimension instance through minimal deletion of edges between the elements. This study suggests a *level-based* deletion of edges starting from the lowest level in the dimension instance. In other words, the removal priority is given to edges that are in the lower levels of the hierarchy. Although this approach resolves non-strictness, it might create a heterogenous dimension, which itself violates summarizability.

Example 7.1. (example 6.1 continued) Figure 7.1 shows one possible repair suggested by [22] for the `Customer` dimension. Since this approach assumes the original dimension to be homogeneous, it does not resolve heterogeneity between categories `Gender` and `Segment`. As can be seen in Figure 7.1, removing the link between $\mathtt{C_3}$ and $\mathtt{L_3}$ creates another partial roll up relation, $\mathcal{R}^{\mathtt{Location}}_{\mathtt{Customer}}$, and also makes the `Customer` dimension *incomplete*. From a path-based perspective, this approach changes the p-instance $\langle \mathtt{C_3,L_3,S_2,all} \rangle$ to the broken p-instance $\langle\ \mathtt{C_3,NULL,NULL,NULL} \rangle$. Propagating this change to the underlying path schema violates the `NOT NULL` constraints obtained in Example 4.8 for checking homogeneity. This simple example shows that, in general there might be no correspondence between the minimal repairs obtained in [22], and those generated by our approach. □

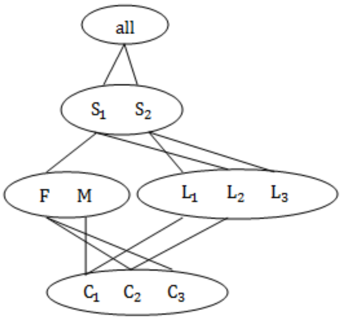


Figure 7.1: Minimal repair obtained in [22] for the `Customer` dimension

Deletion or insertion of edges between the elements in the dimension instance is another repair mechanism suggested by [13, 18, 24]. [13] uses this approach to restore strictness, assuming that the original dimension is homogeneous. [18] (cf. [24]) proposes a more general repair mechanism without making any assumption about the original dimension. Here, 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 these studies, edges are deleted or inserted in an *iterative* manner, until summarizability is satisfied in the dimension instance.

Definition 7.2. [18, 24] Let $\mathcal{D}$ be an MD instance over schema $\mathcal{S}$ and $\mathcal{K}$ be a set of local summarizability constraints. An MD instance $\mathcal{D}'$ over schema $\mathcal{S}$ is said to be a repair for $\mathcal{D}$ wrt $\mathcal{K}$ iff $\mathcal{D}' \models \mathcal{K}$.

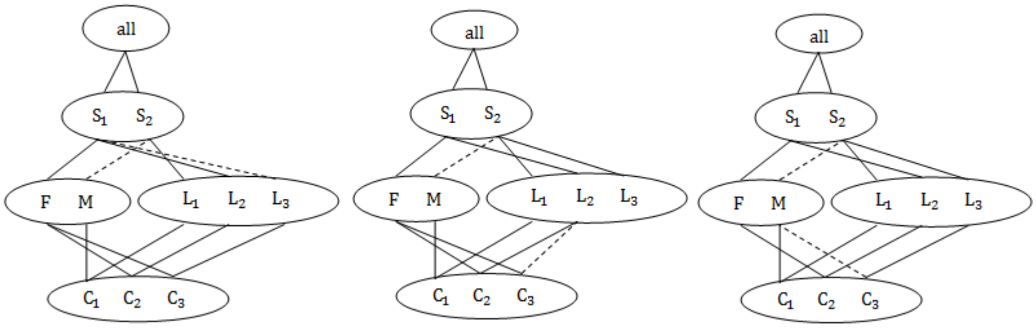
A minimal repair for $\mathcal{D}$ wrt $\mathcal{K}$ is a repair $\mathcal{D}'$, such that the distance between $\mathcal{D}$ and $\mathcal{D}'$, $\mathit{dist}(\mathcal{D},\mathcal{D}')$, is minimum among all of the repairs for $\mathcal{D}$, where the distance $\mathit{dist}(\mathcal{D},\mathcal{D}')$ is the size of the symmetric difference between the child/parent relations of the two dimensions. □

In consequence, the class of repairs in [18], denoted by $\mathit{Rep}^{bch}(\mathcal{D},\mathcal{K})$, is compared to our class $\mathit{Rep}(\mathcal{D},\mathcal{K})$.

Example 7.2. (examples 5.2 and 6.1 continued) Figure 7.2 shows the set of minimal repairs generated by [18], for the `Customer` dimension in Figure 1.4.

Dimension repair $\mathcal{D}_1$ in Figure 7.2a is similar to the dimension we obtained in Example 6.1. This dimension corresponds to the only minimal relational repair we found in Example 5.2, ρ_1^{Path}. However, our approach does not generate repairs $\mathcal{D}_2$ and $\mathcal{D}_3$ for the `Customer` dimension.

Consider repair $\mathcal{D}_3$ in Figure 7.2c. It can be checked that using repair ρ_2^{Path} in Example 5.2, and the inversion rules (VI) and (VII) in Chapter 6, we can obtain the



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

Figure 7.2: Minimal repairs obtained in [18] for the `Customer` dimension, $Rep^{bch}(\mathcal{D}, \mathcal{K})$ (dashed lines resemble inserted edges)

same dimension as $\mathcal{D}_3$. However, since ρ_2^{Path} is not a minimal relational repair, we don't generate $\mathcal{D}_3$ as a repair for the `Customer` dimension.

For $\mathcal{D}_2$ the situation is the same. This MD repair corresponds to the relational update ρ' obtained in Example 5.2. Recall from this example that, ρ' can not be considered as a relational repair for path database, since it performs unnecessary updates. As a result, $\mathcal{D}_2$ is not generated by our approach as a minimal MD repair.

Although repairs $\mathcal{D}_2$ and $\mathcal{D}_3$ are obtained through a minimum number of changes to the `Customer` dimension, their effect on the underlying path schema is not minimal. In fact, $\mathcal{D}_1$ is the only repair that is minimal *from both multidimensional and relational perspectives*. □

So far, we have compared the relational effects of MD repairs $\mathcal{D}_1$, $\mathcal{D}_2$ and $\mathcal{D}_3$ on the path schema. However, it would be interesting to consider these repairs effects on star and snowflake as well.

Due to the flat structure of the star schema, the multidimensional repair operations may have *no effect* on the underlying database. For example, the insertion of an edge between elements `M` and `S`$_2$ does not change the star schema shown in Figure 3.1. As discussed before (see Section 3.3.1), star schema is not expressive enough to check heterogeneity through ICs. An obvious outcome of this property is that, star can not properly reflect the repair operations for restoring homogeneity. However, star schema might be affected by *some* multidimensional repair operations. In general, insertion of a new edge may result in *column updates*, and the deletion of an edge can cause *tuple deletion* or *column updates.*

Example 7.3. (example 7.2 continued) Dimension repairs $\mathcal{D}_1$, $\mathcal{D}_2$ and $\mathcal{D}_3$ result in the following relational repairs for the star schema of Figure 3.1. It can be checked that, based on Definition 5.1, ρ_1^{Star} is the minimal relational repair for this database.

$$\begin{aligned}
\mathcal{D}_1 \Rightarrow \rho_1^{Star} &= \{\langle R(\mathit{4}), A^{\mathtt{Segment}}, \mathtt{S}_1\rangle\}\\
\mathcal{D}_2 \Rightarrow \rho_2^{Star} &= \{\langle R(\mathit{3}), A^{\mathtt{Location}}, \mathtt{L}_2\rangle,\ \langle R(\mathit{4}), A^{\mathtt{Location}}, \mathtt{L}_2\rangle,\ \langle R(\mathit{4}), A^{\mathtt{Segment}}, \mathtt{S}_1\rangle\}\\
\mathcal{D}_3 \Rightarrow \rho_3^{Star} &= \{\langle R(\mathit{3}), A^{\mathtt{Gender}}, \mathtt{M}\rangle,\ \langle R(\mathit{4}), A^{\mathtt{Gender}}, \mathtt{M}\rangle,\ \langle R(\mathit{3}), A^{\mathtt{Segment}}, \mathtt{S}_2\rangle\}
\end{aligned}$$

□

In the snowflake schema, the columns with referential constraint resemble the edges in the dimension instance. Generally, adding a new edge, results in a *reference update*, and deleting an edge, causes either *tuple deletion* or *reference update.*

Example 7.4. (example 7.2 continued) Dimension repairs in Figure 7.2 correspond to the following relational repairs for the snowflake database of Figure 3.3. According to Definition 5.1, all of these relational repairs are minimal.

$$\begin{aligned}
\mathcal{D}_1 \Rightarrow \rho_1^{Snowflake} &= \{\langle R_{\mathtt{Location}}(\mathit{3}), A^{\mathtt{Segment}}, \mathtt{S}_1\rangle, \langle R_{\mathtt{Gender}}(\mathit{1}), A^{\mathtt{Segment}}, \mathtt{S}_2\rangle\}\\
\mathcal{D}_2 \Rightarrow \rho_2^{Snowflake} &= \{\langle R_{\mathtt{Customer}}(\mathit{3}), A^{\mathtt{Location}}, \mathtt{L}_2\rangle, \langle R_{\mathtt{Gender}}(\mathit{1}), A^{\mathtt{Segment}}, \mathtt{S}_2\rangle\}\\
\mathcal{D}_3 \Rightarrow \rho_3^{Snowflake} &= \{\langle R_{\mathtt{Customer}}(\mathit{3}), A^{\mathtt{Gender}}, \mathtt{M}\rangle, \langle R_{\mathtt{Gender}}(\mathit{1}), A^{\mathtt{Segment}}, \mathtt{S}_2\rangle\}
\end{aligned}$$

□

The above discussions suggest that, Definition 7.2 introduces MD repairs that might not have minimal effects on the underlying path database. We showed that, this statement can be generalized to the star schema as well, i.e. not all of the MD repairs in $Rep^{bch}(\mathcal{D},\mathcal{K})$ result in minimal changes to the star database.

This fact implies a difference between the properties of MD repairs in [18] and our approach. The next example shows that, in general, the classes of $Rep(\mathcal{D},\mathcal{K})$ and $Rep^{bch}(\mathcal{D},\mathcal{K})$ are *disjoint*.

Example 7.5. (example 7.2 continued) Consider the dimension schema for the `Customer` dimension shown in Figure 1.4a. In this example, we introduce an alternative instance for this schema, $\mathcal{D}$, which is brought in Figure 7.3. 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. The same set of ICs obtained in Examples 4.6 and 4.8 will be used here for checking global strictness and homogeneity. It can be easily checked that $\mathcal{D}$ is non strict, since $\mathcal{D} \not\models$ `Customer` $\rightarrow$ `Segment`. Notice that elements $\texttt{c}_1, \cdots, \texttt{c}_5$ have two grand parents, $\texttt{s}_1$ and $\texttt{s}_2$ in the `Segment` category.

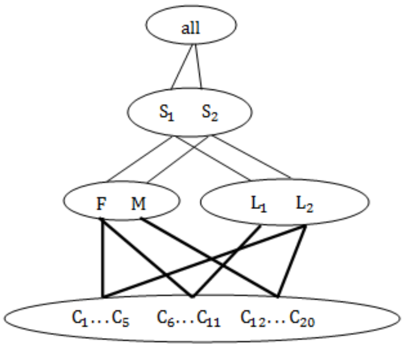


Figure 7.3: A non-strict instance for the `Customer` dimension

The corresponding path database for $\mathcal{D}$ is shown in Figure 7.4. Notice that, the number of tuples in both $RP_1^{\texttt{Cus}}$ and $RP_2^{\texttt{Cus}}$ is 20, but for simplicity, we demonstrate the tables in a somehow compact mode.

According to Definition 5.1, there are two minimal repairs for the path relational

	A^{Customer}	A^{Gender}	A^{Segment}	A^{All}
1...5	$C_1 \ldots C_5$	F	S_1	all
6...11	$C_6 \ldots C_{11}$	F	S_1	all
12...20	$C_{12} \ldots C_{20}$	M	S_2	all

(a) Table $RP_1^{\texttt{Cus}}$

	A^{Customer}	A^{Location}	A^{Segment}	A^{All}
1...5	$C_1 \ldots C_5$	L_2	S_2	all
6...11	$C_6 \ldots C_{11}$	L_1	S_1	all
12...20	$C_{12} \ldots C_{20}$	L_2	S_2	all

(b) Table $RP_2^{\texttt{Cus}}$

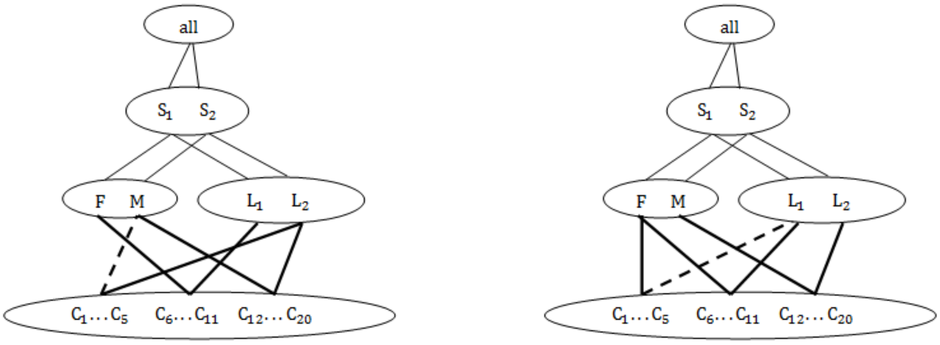
Figure 7.4: Path database D

instance D:

$$\rho_1 = \{\langle RP_1^{\texttt{Cus}}(1,\cdots,5), A^{\texttt{Segment}}, \texttt{S}_2\rangle,\ \langle RP_1^{\texttt{Cus}}(1,\cdots,5), A^{\texttt{Gender}}, \texttt{M}\rangle\}$$

$$\rho_2 = \{\langle RP_2^{\texttt{Cus}}(1,\cdots,5), A^{\texttt{Segment}}, \texttt{S}_1\rangle,\ \langle RP_2^{\texttt{Cus}}(1,\cdots,5), A^{\texttt{Location}}, \texttt{L}_1\rangle\}$$

It holds $|\rho_1| = |\rho_2| = 10$. Using inversion rules (VI) and (VII), the corresponding MD repairs, $\mathcal{D}_1$ and $\mathcal{D}_2$, respectively, are those in Figure 7.5. Notice that, each of these MD repairs performs 10 edge insertions/deletions to the original dimension instance of Figure 7.3.



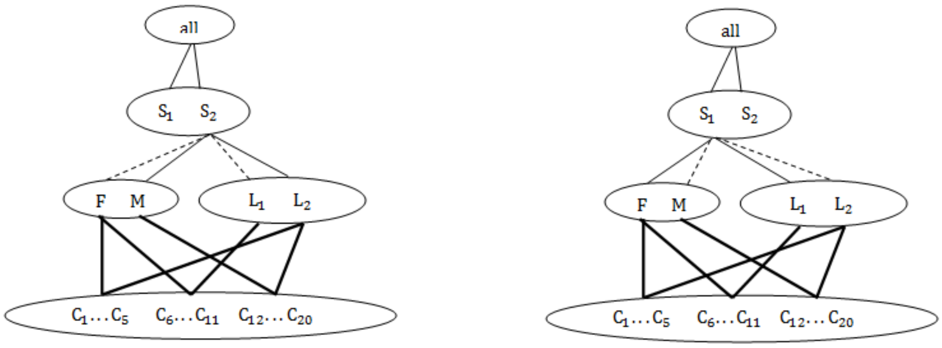
(a) $\mathcal{D}_1$

(b) $\mathcal{D}_2$

Figure 7.5: Minimal repairs in $\mathit{Rep}(\mathcal{D},\mathcal{K})$ for the `Customer` dimension

On the other hand, the MD repairs belonging to $\mathit{Rep}^{bch}(\mathcal{D},\mathcal{K})$ are shown in Figure 7.6.

Each of these MD repairs is obtained by performing 4 edge insertions/deletions. Hence, the repairs in Figure 7.5, $\mathcal{D}_1$ and $\mathcal{D}_2$, can not be considered as minimal based



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

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

Figure 7.6: Minimal repairs in $\mathit{Rep}^{bch}(\mathcal{D},\mathcal{K})$ for the `Customer` dimension

on [18], because they modify more links in the dimension instance.

On the other hand, the effect of repairs $\mathcal{D}_1^{bch}$ and $\mathcal{D}_2^{bch}$ on the underlying path instance is not minimal:

$$\begin{aligned}
\mathcal{D}_1^{bch} \Rightarrow \rho_1^{bch} &= \{\langle RP_1^{\texttt{Cus}}(1,\cdots,5), A^{\texttt{Segment}}, \texttt{S}_2\rangle,\ \langle RP_1^{\texttt{Cus}}(6,\cdots,11), A^{\texttt{Segment}}, \texttt{S}_2\rangle,\\
&\quad\ \langle RP_2^{\texttt{Cus}}(6,\cdots,11), A^{\texttt{Segment}}, \texttt{S}_2\rangle\}\\
\mathcal{D}_2^{bch} \Rightarrow \rho_2^{bch} &= \{\langle RP_1^{\texttt{Cus}}(12,\cdots,20), A^{\texttt{Segment}}, \texttt{S}_1\rangle,\ \langle RP_2^{\texttt{Cus}}(1,\cdots,5), A^{\texttt{Segment}}, \texttt{S}_1\rangle,\\
&\quad\ \langle RP_2^{\texttt{Cus}}(12,\cdots,20), A^{\texttt{Segment}}, \texttt{S}_1\rangle\}
\end{aligned}$$

It can be easily checked that $|\rho_1^{bch}| = 17$, and $|\rho_2^{bch}| = 23$. Hence, these relational updates can not be considered as minimal relational repairs for the instance in Figure 7.4. Thus, in this example, the classes $\mathit{Rep}(\mathcal{D},\mathcal{K})$ and $\mathit{Rep}^{bch}(\mathcal{D},\mathcal{K})$ are disjoint. □

Although the repair classes $\mathit{Rep}(\mathcal{D},\mathcal{K})$ and $\mathit{Rep}^{bch}(\mathcal{D},\mathcal{K})$ are incomparable under set inclusion, it is still worth comparing these two classes for our ongoing example, in order to be able to characterize our MD repairs in pure MD terms.

7.4 Modified Notion of Minimality for Dimension Repairs

In this section, we would like to find the difference between $\mathcal{D}_1$, and the other two repairs in Figure 7.2. By characterizing this difference in multidimensional terms, we can propose a new definition for MD repairs.

Example 7.6. (example 7.2 continued) In this example, we investigate on the edges inserted or deleted by each of the repairs in Figure 7.2. In all of these repairs, an edge is inserted between elements $\texttt{M}$ and $\texttt{S}_2$ for resolving heterogeneity. However, each of these repairs resolve non-strictness in its own unique way. Notice that, the edges modified by repairs $\mathcal{D}_2$ and $\mathcal{D}_3$ belong to the first level of the dimension instance, while in repair $\mathcal{D}_1$ the modified links belong to the second level of the hierarchy.

Dimension repair $\mathcal{D}_2$ changes the link between element $\texttt{C}_3$ and category $\texttt{Location}$. In path mapping terms, this repair modifies the p-instance $\langle \texttt{C}_3,\texttt{L}_3,\texttt{S}_2,\texttt{all}\rangle$ to $\langle \texttt{C}_3,\texttt{L}_2,\texttt{S}_1,\texttt{all}\rangle$. On the other hand, MD repair $\mathcal{D}_1$, updates the same p-instance to $\langle \texttt{C}_3,\texttt{L}_3,\texttt{S}_1,\texttt{all}\rangle$. In other words, repair $\mathcal{D}_2$ causes more changes to the p-instances of the $\texttt{Customer}$ dimension, compared to $\mathcal{D}_1$. For dimension repair $\mathcal{D}_3$ the story is the same. □

The above example suggests that, modifying different edges in dimension instance might have different effects on the underlying database. More specifically, edges should be prioritized for modification in a repair process. The notion of minimality should not only include the number of edges modified, but also the effect of that edge on the relational side. So, we need some a mechanism to measure this relational effect with MD parameters.

Definition 7.3. Let $\mathcal{D}$ and $\mathcal{D}'$ be dimension instances over the same MD schema $\mathcal{S}$ and active domain $\mathcal{M}$ and category association function δ.

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

$$\begin{aligned}
ins(\mathcal{D},\mathcal{D}') &= \{\ (e_1,e_2) \in (<_{\mathcal{D}'} \setminus <_{\mathcal{D}}) \mid thereisnoe_3with(e_1,e_3) \in (<_{\mathcal{D}} \setminus <_{\mathcal{D}'})\}.\\
del(\mathcal{D},\mathcal{D}') &= \{\ (e_1,e_2) \in (<_{\mathcal{D}} \setminus <_{\mathcal{D}'}) \mid thereisnoe_3with(e_1,e_3) \in (<_{\mathcal{D}'} \setminus <_{\mathcal{D}})\}.\\
mod(\mathcal{D},\mathcal{D}') &= \{\ (e_1,e_2,e_3) \mid (e_1,e_2) \in (<_{\mathcal{D}'} \setminus <_{\mathcal{D}})\ and(e_1,e_3) \in (<_{\mathcal{D}} \setminus <_{\mathcal{D}'})\}.
\end{aligned}$$

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

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

with:

$\alpha(e_1,e_2) = \{p \mid p \in Inst^{\mathcal{D}}(P),\ \{\delta(e_1),\delta(e_2)\} \subseteq P,\ e_1 \in p\}$,

$\beta(e) = \{\ e' \in \mathcal{M} \mid e <^*_{\mathcal{D}} e'\}$, and $\gamma(e_2,e_3) = \{e' \in \mathcal{M} \mid e_2 <^*_{\mathcal{D}} e'$, but not $e_3 <^*_{\mathcal{D}} e'\}$. □

Intuitively, the cost of updating $\mathcal{D}$ to $\mathcal{D}'$ can be interpreted as the number of changes made to the elements of p-instances belonging to $\mathcal{D}$. In relational terms, this value is equal to the number of attribute updates, as a result of the MD updates. To calculate this value, we consider each edge change separately, as described in the next paragraph. The sum of the number of attribute modifications for all changed edges (e_1,e_2) equals the total number of attribute value updates needed for updating $\mathcal{D}$ to $\mathcal{D}'$, which is captured by $ucost$.

For computing the number of attribute updates resulted from each edge modification (e_1,e_2), we consider two parameters: the number of tuples affected by this edge change, and the number of attribute values in each of these tuples that will be updated. In Definition 7.3, $\alpha(e_1,e_2)$ represents the former parameter, and β and γ capture the latter.

In particular, α reflects the set of p-instances that will be updated by changing edge (e_1,e_2). The size of this set shows the number of tuples that will be affected as a result of this edge change. Now, for measuring the number of changes made to each p-instance, we take into account the discussion in Example 7.6; based on the level of the edge being modified, the amount of changes made to the p-instances varies. Hence, Definition 7.3 introduces two factors β and γ such that, for each aforementioned p-instance in $\alpha(e_1, e_2)$, $|\beta|$ and $|\gamma|$ represent the number of changes made to the elements proceeding e_2 in the p-instance, depending on whether the edge is inserted/deleted or modified. 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 7.7. (example 7.6 continued) In this example, we will calculate the cost of updating the `Customer` dimension to each of the dimensions $\mathcal{D}_1$, $\mathcal{D}_2$ and $\mathcal{D}_3$ in Figure 7.2. The first step in this process is to find the set of edge changes, namely, insertions, deletions and modifications:

$$\begin{aligned}
&ins(\mathcal{D}, \mathcal{D}_1) = \{(\mathtt{M}, \mathtt{S}_2)\}, \quad del(\mathcal{D}, \mathcal{D}_1) = \emptyset, \quad mod(\mathcal{D}, \mathcal{D}_1) = \{(\mathtt{L}_3, \mathtt{S}_1, \mathtt{S}_2)\},\\
&ins(\mathcal{D}, \mathcal{D}_2) = \{(\mathtt{M}, \mathtt{S}_2)\}, \quad del(\mathcal{D}, \mathcal{D}_2) = \emptyset, \quad mod(\mathcal{D}, \mathcal{D}_2) = \{(\mathtt{C}_3, \mathtt{L}_2, \mathtt{L}_3)\},\\
&ins(\mathcal{D}, \mathcal{D}_3) = \{(\mathtt{M}, \mathtt{S}_2)\}, \quad del(\mathcal{D}, \mathcal{D}_3) = \emptyset, \quad mod(\mathcal{D}, \mathcal{D}_3) = \{(\mathtt{C}_3, \mathtt{M}, \mathtt{F})\},
\end{aligned}$$

Now, *ucost* for each of these MD repairs can be calculated as follows:

$$\begin{aligned}
ucost(\mathcal{D}, \mathcal{D}_1) &= |\alpha(\mathtt{M}, \mathtt{S}_2)| \times |\beta(\mathtt{S}_2)| + |\alpha(\mathtt{L}_3, \mathtt{S}_1)| \times |\gamma(\mathtt{S}_1, \mathtt{S}_2)|\\
ucost(\mathcal{D}, \mathcal{D}_2) &= |\alpha(\mathtt{M}, \mathtt{S}_2)| \times |\beta(\mathtt{S}_2)| + |\alpha(\mathtt{C}_3, \mathtt{L}_2)| \times |\gamma(\mathtt{L}_2, \mathtt{L}_3)|\\
ucost(\mathcal{D}, \mathcal{D}_3) &= |\alpha(\mathtt{M}, \mathtt{S}_2)| \times |\beta(\mathtt{S}_2)| + |\alpha(\mathtt{C}_3, \mathtt{M})| \times |\gamma(\mathtt{M}, \mathtt{F})|
\end{aligned}$$

So, we need to compute the sets α, β and γ for each of the specified edge changes:

$$\begin{aligned}
\alpha(\texttt{M},\texttt{S}_2) &= \{\langle \texttt{C}_1,\texttt{M},\texttt{NULL},\texttt{NULL}\rangle\},\\
\alpha(\texttt{L}_3,\texttt{S}_1) &= \{\langle \texttt{C}_3,\texttt{L}_3,\texttt{S}_2,\texttt{all}\rangle\},\\
\alpha(\texttt{C}_3,\texttt{L}_2) &= \{\langle \texttt{C}_3,\texttt{L}_3,\texttt{S}_2,\texttt{all}\rangle\},\\
\alpha(\texttt{C}_3,\texttt{M}) &= \{\langle \texttt{C}_3,\texttt{F},\texttt{S}_1,\texttt{all}\rangle\}\\
\beta(\texttt{S}_2) &= \{\texttt{S}_2,\texttt{all}\}\\
\gamma(\texttt{S}_1,\texttt{S}_2) &= \{\texttt{S}_1\}\\
\gamma(\texttt{L}_2,\texttt{L}_3) &= \{\texttt{L}_2,\texttt{S}_1\}\\
\gamma(\texttt{M},\texttt{F}) &= \{\texttt{M},\texttt{S}_2\}
\end{aligned}$$

With these elements, we can compute the update costs for each case:

$$\begin{aligned}
\mathit{ucost}(\mathcal{D},\mathcal{D}_1) &= |\alpha(\texttt{M},\texttt{S}_2)| \times |\beta(\texttt{S}_2)| + |\alpha(\texttt{L}_3,\texttt{S}_1)| \times |\gamma(\texttt{S}_1,\texttt{S}_2)| = 1\times 2 + 1 \times 1 = 3,\\
\mathit{ucost}(\mathcal{D},\mathcal{D}_2) &= |\alpha(\texttt{M},\texttt{S}_2)| \times |\beta(\texttt{S}_2)| + |\alpha(\texttt{C}_3,\texttt{L}_2)| \times |\gamma(\texttt{L}_2,\texttt{L}_3)| = 1\times 2 + 1 \times 2 = 4,\\
\mathit{ucost}(\mathcal{D},\mathcal{D}_3) &= |\alpha(\texttt{M},\texttt{S}_2)| \times |\beta(\texttt{S}_2)| + |\alpha(\texttt{C}_3,\texttt{M})| \times |\gamma(\texttt{M},\texttt{F})| = 1\times 2 + 1 \times 2 = 4.
\end{aligned}$$

We can see that $\mathcal{D}_1$ 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 6.1: $\mathcal{D}_1$ is the only minimal MD repair for $\mathcal{D}$ that also corresponds to a minimal relational repair.

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 by the corresponding relational repairs (see Example 5.2). □

Lemma 7.1. Let $\mathcal{D}$ be an instance for the MD schema $\mathcal{S}$, and $\mathcal{K}$ be a set of local summarizability constraints over $\mathcal{S}$. Let D and Σ be the corresponding elements on the path relational side, and ρ an update of D. For the MD instance $\mathcal{D}'$ for $\mathcal{S}$ with $\mathcal{D}'$ $= \mathcal{T}^{-1}(\rho(D))$, it holds: $\mathit{ucost}(\mathcal{D},\mathcal{D}') = |\rho|$.

Proof: The number of attribute value updates caused by ρ 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 7.3). *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 α for a given edge in Definition 7.3 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 the MD repairs generated by our relational approach, in pure MD terms.

Theorem 7.1. Let $\mathcal{D}$ be an instance for the MD schema $\mathcal{S}$, and $\mathcal{K}$ be a set of local summarizability constraints over $\mathcal{S}$. For every instance $\mathcal{D}'$ of $\mathcal{S}$, it holds: $\mathcal{D}' \in \mathit{Rep}(\mathcal{D}, \mathcal{K})$ iff $\mathcal{D}' \models \mathcal{K}$ and $\mathit{ucost}(\mathcal{D}, \mathcal{D}')$ 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 $D' \in \mathit{Rep}(D, \Sigma)$, such that $\mathcal{D}' = \mathcal{T}^{-1}(D')$ (cf. Definition 7.1). From Lemma 7.1 we have that, for any MD update $\mathcal{D}'$, $\mathit{ucost} = |\rho|$, where $\mathcal{T}^{-1}(\rho(D)) = \mathcal{D}'$. So, a minimum ucost implies a minimum $|\rho|$, which itself leads to $\rho(D) \in \mathit{Rep}(D, \Sigma)$. 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 $\mathcal{D}' \in \mathit{Rep}(\mathcal{D}, \mathcal{K})$, $\mathit{ucost}(\mathcal{D}, \mathcal{D}')$ will be minimum. According to Definition 7.1, for every $\mathcal{D}' \in \mathit{Rep}(\mathcal{D}, \mathcal{K})$, there exists a $D' \in \mathit{Rep}(D, \Sigma)$, such that $\mathcal{D}' = \mathcal{T}^{-1}(D')$. From Definition 5.1, it holds that D' corresponds to a relational repair ρ which performs a minimum number of attribute value updates on D, i.e. $|\rho|$ is minimum. Hence, based on Lemma 7.1, the *ucost* must also be minimum. □

Based on the above discussion, the proposed characterization of MD repairs implicitly takes into consideration the effect of MD repair operations (edge insertions and deletions) on the underlying relational layer. We discussed that, our approach and the one in [18] generate two disjoint classes of MD repairs. The reason is that, in the latter edges are treated the same for modification, while the former considers priority of change on edges. In particular, in our repair process, edges with fewer connections to the base elements are considered as 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 attribute values in the affected tuples.

7.5 Summary

In this chapter, we studied the correspondence between our results and other direct MD repair approaches. In particular, we compared our dimension repairs with those obtained from [18], as the most generic instance-based repair mechanism. Our results showed that, in general, there is no correspondence between the our repairs and those generated by [18]. Through this comparison, we were able to propose an MD characterization of our MD repairs.

Chapter 8

Experiments

8.1 Introduction

Our discussions in the previous chapter show the conceptual and theoretical advantages of our relational repair approach, clearly. However, as a new relational implementation for MDDBs, path schema should still be analyzed in more detail. In other words, an in-depth comparison of path with the existing star and snowflake schemas is crucial.

Consider the case where the original MD instance is not implemented as a path relational database. In order to apply the proposed repair mechanism, we need to translate the MD instance into a path instance. This translation introduces a non-negligible cost. Such common situations lead us to investigate on the possibility of using upfront *path relational schemas* as the basis for the implementation of MD databases. We claim that, the path relational schema is an interesting alternative to consider for a ROLAP approach to MDDBs, and this advantage is independent from its virtues in relation to MD repairing.

In order to support our claim, we demonstrate several experiments comparing path, star and snowflake schemas. In particular, these schemas are analyzed based on their performance at aggregate query answering, and inconsistency detection. We use SQL Server 2008 for running our experiments.

These experiments are based on our running example about product sales (cf. Example 1.1). We consider dimensions `Customer`, `Store` and `Date` for measuring the `Sales` amount. The schema for these dimensions can be found in Figures 1.4a, 1.2 and 1.3, respectively.

To achieve our goal, we needed three independent representations of the MDDB in Example 1.1, as a star, snowflake and path databases. Figures 3.1, 3.3 and 4.1 show the representation of the `Customer` dimension in these schemas, respectively. In case of `Store` and `Date` dimensions, the MD2R mapping is more straightforward, and is done as described in Sections 3.2, 3.4 and 4.3. The relational tables representing these two dimensions in star, snowflake and path databases are as follows:
Star:

- $\mathcal{S}_{\texttt{Store}} \mapsto_{Star} \{\mathrm{R}^{\texttt{Store}}[A^{\texttt{Branch}}, A^{\texttt{City}}, A^{\texttt{Country}}, A^{\texttt{All}}]\}$
- $\mathcal{S}_{\texttt{Date}} \mapsto_{Star} \{\mathrm{R}^{\texttt{Date}}[A^{\texttt{Day}}, A^{\texttt{Month}}, A^{\texttt{Year}}, A^{\texttt{All}}]\}$

Snowflake:

- $\mathcal{S}_{\texttt{Store}} \mapsto_{Snowflake} \{R_{\texttt{Branch}}[A^{\texttt{Branch}}, A^{\texttt{City}}],\ R_{\texttt{City}}[A^{\texttt{City}}, A^{\texttt{Country}}],\ R_{\texttt{Country}}[A^{\texttt{Country}}, A^{\texttt{All}}],\ R_{\texttt{All}}[A^{\texttt{All}}]\}$
- $\mathcal{S}_{\texttt{Date}} \mapsto_{Snowflake} \{R_{\texttt{Day}}[A^{\texttt{Day}}, A^{\texttt{Month}}],\ R_{\texttt{Month}}[A^{\texttt{Month}}, A^{\texttt{Year}}],\ R_{\texttt{Year}}[A^{\texttt{Year}}, A^{\texttt{All}}],\ R_{\texttt{All}}[A^{\texttt{All}}]\}$

Path:

- $\mathcal{S}_{\texttt{Store}} \mapsto_{Path} \{RP^{\texttt{Store}}[A^{\texttt{Branch}}, A^{\texttt{City}}, A^{\texttt{Country}}, A^{\texttt{All}}]\}$
- $\mathcal{S}_{\texttt{Date}} \mapsto_{Path} \{RP^{\texttt{Date}}[A^{\texttt{Day}}, A^{\texttt{Month}}, A^{\texttt{Year}}, A^{\texttt{All}}]\}$

8.2 Query Answering

In this section, we analyze the query answering time of star, snowflake and path schemas, based on the category used in the aggregate query. More specifically, we investigate how the query answering time changes as the level of the category used in the query increases in the dimension hierarchy. This criterion shows how much the query answering performance is dependant on the category used. Obviously, if the query answering performance changes significantly as the category used in the query varies, the average query answering time for the relational schema increases.

Our primary goal in this experiment was to define the MDDB of Example 1.1 in SQL Server 2008. To this end, we needed three MD instances, with the same set of dimensions (`Customer`, `Store` and `Date` dimensions), each implemented differently, i.e. as a star, snowflake or path instance. Due to some limitations in SQL Server for mapping the MD instance to the relational instance, we could not achieve this goal. SQL Server 2008 only supports a one-to-one mapping from a category in dimension schema to an attribute in relational table. Despite this restriction, star and snowflake schemas can be defined as the MDDB relational implementation. However, path mapping might map a category to more than one attribute in the relational schema (like `Segment` and `All` in the `Customer` dimension). As a result, due to this SQL Server limitation, we could not define path schema as the underlying relational layer of the MDDB in Example 1.1.

Based on the above discussions, we could not analyze the query answering performance of the star, snowflake and path schemas by posing MDX queries to the upper MD layer. So, we decided to run our experiments on the relational layer, i.e. posing SQL queries to these databases and analyzing their performance in query answering.

After defining the dimension tables in star, snowflake and path databases, we needed to load these instances with test data. Since we aim at testing the performance of these schemas, the database instance must be initiated with a sizable amount of data. To this end, we implemented a data generator in Java.

Since the data generator program fills the star, snowflake and path databases with random values, it must be given the relational schema of these databases. Hence, the program takes as input the set of tables along with their attributes for each of the star, snowflake and path schemas. Based on the given input, the program generates random values for each attribute and inserts the generated tuples to the database tables. In order to simulate the real cases, this data generator program takes into account the hierarchy levels. In reality, the number of elements belonging to a category decreases as the level of the category in the dimension schema increases. Hence, the number of the values generated for an attribute depends on the level of the corresponding category in the dimension schema.

We designed a data structure representing a relational table which contains the table name and its attributes. For each attribute, we specify its name and the level of the corresponding category in the dimension schema. For example, the set of attributes of table $RP_1^{\texttt{Cus}}$ in Figure 4.1 are defined for the data generator program as $\{(A^{\texttt{Customer}}$, 1), $(A^{\texttt{Gender}}$, 2), $(A^{\texttt{Segment}}$, 3), $(A^{\texttt{All}}$, -1)$\}$. Notice that, for attribute $A^{\texttt{All}}$ the level is defined as -1. The reason for this setting is that, we do not want to generate any value for this attribute, except for the pre-defined element `all`. Hence, the value -1 is used as an indicator to skip the data generation phase for an attribute. In general, the maximum number of values generated for an attribute with level l is equal to $\frac{100{,}000}{3^{l-1}}$. For an attribute representing the base category of the dimension hierarchy, the number of generated values is equal to 100,000. Table 8.1 shows the size of the generated values for an attribute based on the level of its corresponding category.

As discussed above, the query answering performance of star, snowflake and path schemas is analyzed using several queries. In this section, we start by explaining one

Level of Category c in Hierarchy	Number of Generated Values for Attribute A^c
1	100,000
2	33,333
3	11,111
-1 (category `All`)	1

Table 8.1: The size of the generated values for a category based on its level

of the aggregate queries used in our experiments.

In this query, $\mathcal{Q}$, we are interested in retrieving the total amount of product `Sales` for each Canadian store in year `2010`, based on customer `Segment` . This query takes 3 different forms in SQL depending on the relational schema (cf. Section 4.4). The equivalent SQL queries for the aforementioned query $\mathcal{Q}$ is shown in Table 8.2.

Section 4.4 explains the difference between the structure of the above queries. Notice that, queries (a) and (b) have the same number of join operations. However, query (b) has an additional inner query compared to query (a). Hence, it is expected to take more time for execution than query (a). On the other hand, query (c) requires 11 number of join operations, while in case of queries (a) or (b) this number is equal to 3. Obviously, this difference implies a considerably longer execution time for query (c).

We executed each of the above SQL queries on their corresponding relational schemas. Our experiment results support the above statements. More specifically, the response times obtained are 155.2 ms for query (a), 193.8 ms for query (b), and 427.3 ms for query (c). Notice the non-negligible difference between execution times of query (c) and the first two queries.

(a) Star schema:

SELECT $\mathtt{R}^{\mathsf{Customer}}.A^{\mathsf{Segment}}$, SUM(F.Purchase) FROM

Sale-Fact-Table F, $\mathtt{R}^{\mathsf{Customer}}$, $\mathtt{R}^{\mathsf{Date}}$, $\mathtt{R}^{\mathsf{Store}}$ WHERE

$\mathtt{R}^{\mathsf{Date}}.A^{\mathsf{Year}} = 2010$ AND $\mathtt{R}^{\mathsf{Store}}.A^{\mathsf{Country}} =$ Canada

GROUP BY $\mathtt{R}^{\mathsf{Customer}}.A^{\mathsf{Segment}}$;

(b) Path schema:

SELECT $RP^{\mathsf{Cus}}.A^{\mathsf{Segment}}$, SUM(F.Purchase) FROM

Sale-Fact-Table F, RP^{Date}, RP^{Store},

((SELECT A^{Customer} , A^{Segment} FROM RP_1^{Cus}) UNION

(SELECT A^{Customer} , A^{Segment} FROM RP_2^{Cus})) as RP^{Cus}

WHERE $RP^{\mathsf{Date}}.A^{\mathsf{Year}} = 2010$ AND $RP^{\mathsf{Store}}.A^{\mathsf{Country}} =$ Canada

GROUP BY $RP^{\mathsf{Cus}}.A^{\mathsf{Segment}}$;

(c) Snowflake schema:

SELECT $R^{\mathsf{Cus}}.A^{\mathsf{Segment}}$, SUM(F.Purchase) FROM

Sale-Fact-Table F, R_{Day}, R_{Month}, R_{Year}, R_{Branch}, R_{City}, R_{Country},

((SELECT A^{Customer} , A^{Segment} FROM R_{Customer}, R_{Gender}, R_{Segment}) UNION

(SELECT A^{Customer} , A^{Segment} FROM R_{Customer}, R_{Location}, R_{Segment})) as R^{Cus}

WHERE $R_{\mathsf{Year}}.A^{\mathsf{Year}} = 2010$ AND $R_{\mathsf{Country}}.A^{\mathsf{Country}} =$ Canada

GROUP BY $R^{\mathsf{Cus}}.A^{\mathsf{Segment}}$;

Table 8.2: Equivalent SQL query for $\mathcal{Q}$ in star, path and snowflake schemas, resp.

As discussed before, the MD query $\mathcal{Q}$ involves the computation of the roll up relations $\mathcal{R}_{\texttt{Customer}}^{\texttt{Segment}}$, $\mathcal{R}_{\texttt{Day}}^{\texttt{Year}}$ and $\mathcal{R}_{\texttt{Branch}}^{\texttt{Country}}$. However, this example is not a strong support for our original claim. We need to extend our experiment to include other queries. We are interested in analyzing the effect of the roll up relations used in the MD query on the answering time of the equivalent SQL query. More specifically, we would like to try other roll up relations than those mentioned above, in our sample queries as well. To this end, we considered the roll up relations of the `Customer` dimension, namely $\mathcal{R}_{\texttt{Customer}}^{\texttt{Customer}}$, $\mathcal{R}_{\texttt{Customer}}^{\texttt{Gender}}$, $\mathcal{R}_{\texttt{Customer}}^{\texttt{Location}}$ and $\mathcal{R}_{\texttt{Customer}}^{\texttt{All}}$ in the following MD queries, $\mathcal{Q}_{\texttt{Customer}}$, $\mathcal{Q}_{\texttt{Gender}}$, $\mathcal{Q}_{\texttt{Location}}$ and $\mathcal{Q}_{\texttt{All}}$, respectively. Due to the linear structure of the `Date` and `Store` dimensions, the equivalent SQL queries for these dimensions would be simpler. That is why, we tried different roll up relations of the `Customer` dimension, rather than `Date` or `Store` dimensions, in our experiments.

- $\mathcal{Q}_{\texttt{Customer}}$: Return the sum of product `Sales` made in the `Canadian` stores during year `2010` for each `Customer` .

- $\mathcal{Q}_{\texttt{Gender}}$: Return the sum of product `Sales` made in the `Canadian` stores during year `2010` for female and male customers.

- $\mathcal{Q}_{\texttt{Location}}$: Return the sum of product `Sales` made in the `Canadian` stores during year `2010` for each `Location`, namely $\texttt{L}_1$, $\texttt{L}_2$ and $\texttt{L}_3$.

- $\mathcal{Q}_{\texttt{Segment}}$: Return the sum of product `Sales` made in the `Canadian` stores during year `2010` for each customer `Segment`, namely $\texttt{S}_1$ and $\texttt{S}_2$.

- $\mathcal{Q}_{\texttt{All}}$: Return the sum of product `Sales` made in the `Canadian` stores during year `2010` for `All` of the customers.

Notice that $\mathcal{Q}_{\texttt{Segment}}$ is the same as $\mathcal{Q}$, which we already used in our experiment.

Tables 8.3, 8.4, 8.5 and 8.6 represent the equivalent SQL queries for the above MD queries over star, snowflake and path database.

(a) Star schema:

SELECT $\mathtt{R}^{\texttt{Customer}}.A^{\texttt{Customer}}$, SUM(F.Purchase) FROM

Sale-Fact-Table F, $\mathtt{R}^{\texttt{Customer}}$, $\mathtt{R}^{\texttt{Date}}$, $\mathtt{R}^{\texttt{Store}}$ WHERE

$\mathtt{R}^{\texttt{Date}}.A^{\texttt{Year}} = 2010$ AND $\mathtt{R}^{\texttt{Store}}.A^{\texttt{Country}} = \texttt{Canada}$

GROUP BY $\mathtt{R}^{\texttt{Customer}}.A^{\texttt{Customer}}$;

(b) Path schema[a]:

SELECT $RP_1^{\texttt{Cus}}.A^{\texttt{Customer}}$, SUM(F.Purchase) FROM

Sale-Fact-Table F, $RP^{\texttt{Date}}$, $RP^{\texttt{Store}}$, $RP_1^{\texttt{Cus}}$

WHERE $RP^{\texttt{Date}}.A^{\texttt{Year}} = 2010$ AND $RP^{\texttt{Store}}.A^{\texttt{Country}} = \texttt{Canada}$

GROUP BY $RP_1^{\texttt{Cus}}.A^{\texttt{Customer}}$;

(c) Snowflake schema:

SELECT $R_{\texttt{Customer}}.A^{\texttt{Customer}}$, SUM(F.Purchase) FROM

Sale-Fact-Table F, $R_{\texttt{Day}}$, $R_{\texttt{Month}}$, $R_{\texttt{Year}}$, $R_{\texttt{Branch}}$, $R_{\texttt{City}}$, $R_{\texttt{Country}}$, $R_{\texttt{Customer}}$

WHERE $R_{\texttt{Year}}.A^{\texttt{Year}} = 2010$ AND $R_{\texttt{Country}}.A^{\texttt{Country}} = \texttt{Canada}$

GROUP BY $R_{\texttt{Customer}}.A^{\texttt{Customer}}$;

[a] For path schema, the values for attribute $RP_1^{\texttt{Cus}}.A^{\texttt{Customer}}$ is equal to $RP_2^{\texttt{Cus}}.A^{\texttt{Customer}}$. So in this query, we can either choose $RP_1^{\texttt{Cus}}$ or $RP_2^{\texttt{Cus}}$ (cf. Section 4.3).

Table 8.3: Equivalent SQL query for $\mathcal{Q}_{\texttt{Customer}}$ in star, path and snowflake schemas, resp.

(a) Star schema:

SELECT $\texttt{R}^{\texttt{Customer}}.A^{\texttt{Gender}}$, SUM(F.Purchase) FROM
Sale-Fact-Table F, $\texttt{R}^{\texttt{Customer}}$, $\texttt{R}^{\texttt{Date}}$, $\texttt{R}^{\texttt{Store}}$ WHERE
$\texttt{R}^{\texttt{Date}}.A^{\texttt{Year}} = \texttt{2010}$ AND $\texttt{R}^{\texttt{Store}}.A^{\texttt{Country}} = \texttt{Canada}$
GROUP BY $\texttt{R}^{\texttt{Customer}}.A^{\texttt{Gender}}$;

(b) Path schema:

SELECT $RP_1^{\texttt{Cus}}.A^{\texttt{Gender}}$, SUM(F.Purchase) FROM
Sale-Fact-Table F, $RP^{\texttt{Date}}$, $RP^{\texttt{Store}}$, $RP_1^{\texttt{Cus}}$
WHERE $RP^{\texttt{Date}}.A^{\texttt{Year}} = \texttt{2010}$ AND $RP^{\texttt{Store}}.A^{\texttt{Country}} = \texttt{Canada}$
GROUP BY $RP_1^{\texttt{Cus}}.A^{\texttt{Gender}}$;

(c) Snowflake schema:

SELECT $R_{\texttt{Gender}}.A^{\texttt{Gender}}$, SUM(F.Purchase) FROM
Sale-Fact-Table F, $R_{\texttt{Day}}$, $R_{\texttt{Month}}$, $R_{\texttt{Year}}$,
$R_{\texttt{Branch}}$, $R_{\texttt{City}}$, $R_{\texttt{Country}}$, $R_{\texttt{Customer}}$, $R_{\texttt{Gender}}$
WHERE $R_{\texttt{Year}}.A^{\texttt{Year}} = \texttt{2010}$ AND $R_{\texttt{Country}}.A^{\texttt{Country}} = \texttt{Canada}$
GROUP BY $R_{\texttt{Gender}}.A^{\texttt{Gender}}$;

Table 8.4: Equivalent SQL query for $\mathcal{Q}_{\texttt{Gender}}$ in star, path and snowflake schemas, resp.

(a) Star schema:

SELECT $\mathtt{R}^{\mathtt{Customer}}.A^{\mathtt{Location}}$, SUM(F.Purchase) FROM

Sale-Fact-Table F, $\mathtt{R}^{\mathtt{Customer}}$, $\mathtt{R}^{\mathtt{Date}}$, $\mathtt{R}^{\mathtt{Store}}$ WHERE

$\mathtt{R}^{\mathtt{Date}}.A^{\mathtt{Year}} = \mathtt{2010}$ AND $\mathtt{R}^{\mathtt{Store}}.A^{\mathtt{Country}} = \mathtt{Canada}$

GROUP BY $\mathtt{R}^{\mathtt{Customer}}.A^{\mathtt{Location}}$;

(b) Path schema:

SELECT $RP_2^{\mathtt{Cus}}.A^{\mathtt{Location}}$, SUM(F.Purchase) FROM

Sale-Fact-Table F, $RP^{\mathtt{Date}}$, $RP^{\mathtt{Store}}$, $RP_2^{\mathtt{Cus}}$

WHERE $RP^{\mathtt{Date}}.A^{\mathtt{Year}} = \mathtt{2010}$ AND $RP^{\mathtt{Store}}.A^{\mathtt{Country}} = \mathtt{Canada}$

GROUP BY $RP_2^{\mathtt{Cus}}.A^{\mathtt{Location}}$;

(c) Snowflake schema:

SELECT $R_{\mathtt{Location}}.A^{\mathtt{Location}}$, SUM(F.Purchase) FROM

Sale-Fact-Table F, $R_{\mathtt{Day}}$, $R_{\mathtt{Month}}$, $R_{\mathtt{Year}}$,

$R_{\mathtt{Branch}}$, $R_{\mathtt{City}}$, $R_{\mathtt{Country}}$, $R_{\mathtt{Customer}}$, $R_{\mathtt{Location}}$

WHERE $R_{\mathtt{Year}}.A^{\mathtt{Year}} = \mathtt{2010}$ AND $R_{\mathtt{Country}}.A^{\mathtt{Country}} = \mathtt{Canada}$

GROUP BY $R_{\mathtt{Location}}.A^{\mathtt{Location}}$;

Table 8.5: Equivalent SQL query for $\mathcal{Q}_{\mathtt{Location}}$ in star, path and snowflake schemas, resp.

(a) Star schema:

SELECT R$^{\text{Customer}}.A^{\texttt{All}}$, SUM(F.Purchase) FROM
Sale-Fact-Table F, R$^{\text{Customer}}$, R$^{\text{Date}}$, R$^{\text{Store}}$ WHERE
R$^{\text{Date}}.A^{\text{Year}} = 2010$ AND R$^{\text{Store}}.A^{\text{Country}} =$ Canada
GROUP BY R$^{\text{Customer}}.A^{\texttt{All}}$;

(b) Path schema:

SELECT $RP^{\text{Cus}}.A^{\texttt{All}}$, SUM(F.Purchase) FROM
Sale-Fact-Table F, RP^{Date}, RP^{Store},
((SELECT A^{Customer} , $A^{\texttt{All}}$ FROM RP_1^{Cus}) UNION
(SELECT A^{Customer} , $A^{\texttt{All}}$ FROM RP_2^{Cus})) as RP^{Cus}
WHERE $RP^{\text{Date}}.A^{\text{Year}} = 2010$ AND $RP^{\text{Store}}.A^{\text{Country}} =$ Canada
GROUP BY $RP^{\text{Cus}}.A^{\texttt{All}}$;

(c) Snowflake schema:

SELECT $R^{\text{Cus}}.A^{\texttt{All}}$, SUM(F.Purchase) FROM
Sale-Fact-Table F, $R_{\texttt{Day}}$, $R_{\texttt{Month}}$, $R_{\texttt{Year}}$, $R_{\texttt{Branch}}$, $R_{\texttt{City}}$, $R_{\texttt{Country}}$,
((SELECT A^{Customer} , $A^{\texttt{All}}$ FROM $R_{\texttt{Customer}}$, $R_{\texttt{Gender}}$, $R_{\texttt{Segment}}$, $R_{\texttt{All}}$) UNION
(SELECT A^{Customer} , $A^{\texttt{All}}$ FROM $R_{\texttt{Customer}}$, $R_{\texttt{Location}}$, $R_{\texttt{Segment}}$, $R_{\texttt{All}}$)) as R^{Cus}
WHERE $R_{\texttt{Year}}.A^{\text{Year}} = 2010$ AND $R_{\texttt{Country}}.A^{\text{Country}} =$ Canada
GROUP BY $R^{\text{Cus}}.A^{\texttt{All}}$;

Table 8.6: Equivalent SQL query for $\mathcal{Q}_{\texttt{All}}$ in star, path and snowflake schemas, resp.

As previously discussed, the queries over star database are very similar to each other, in the sense that, they all join the fact table to the relevant dimension tables (see part (a) of Tables 8.3, 8.4, 8.5 and 8.6).

On the other hand, the queries over path database has different structure based on the roll up relation used in the MD query. Notice that, queries in part (b) of Tables 8.4 and 8.5 have the same structure. While the query for path database in Table 8.6 looks totally different.

In case of snowflake schema, the number of joins in the SQL query depends on the roll up relation used in the MD query. It can be easily checked that, the query in part (c) of Table 8.3 executes 7 join operations, while the snowflake queries in Tables 8.4 and 8.5 perform 8 number of joins. In case of the query in Table 8.6(c), the number of joins required is equal to 12.

The result of executing these queries over the star, snowflake and path databases, is shown in Figure 8.1.

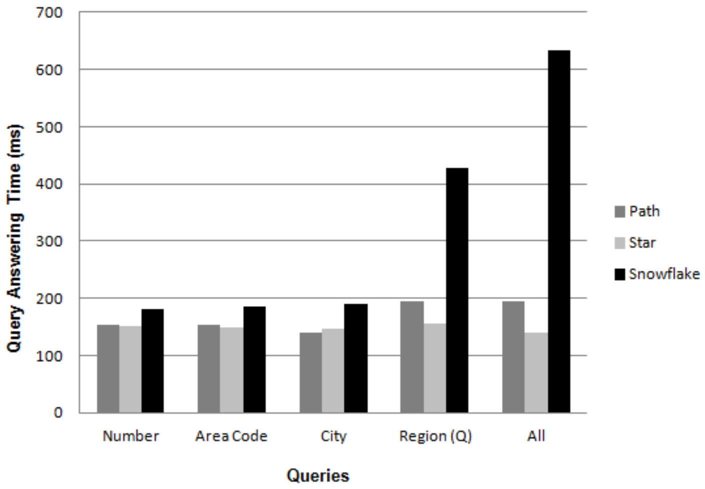


Figure 8.1: Comparison of query answering for path, star and snowflake schemas

An interesting phenomenon that can be observed from the graph in Figure 8.1 is that, the queries corresponding to $\mathcal{Q}_{\texttt{Customer}}$, $\mathcal{Q}_{\texttt{Gender}}$ and $\mathcal{Q}_{\texttt{Location}}$ over path database have almost the same response time as their counterpart for star database. This is due to the fact that, queries over path database in Tables 8.3, 8.4 and 8.5 do not contain an inner query as the one in Table 8.2(b). In these queries, only one B2A path has to be considered, and there is no need to merge the tuples of several path tables. In other words, the aforementioned queries have similar structure as the queries over star database. Hence, in these cases, the query answering time for path and star schemas are the same. 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.

Also notice that, as the level of the category used in the aggregate query increases, the query answering time for the snowflake schema increases significantly. However, the performance for the star and path schemas is not considerably dependant on this factor. This is an expected property 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 does not change by the level of the category used in the aggregate query.

In general, Figure 8.1 shows that, as expected, the star schema has the best query answering performance compared to path and snowflake schemas. In this ranking, path schema is in the second place with a slight difference to the star schema. This shows that, the new MDDB implementation shows some useful properties regarding the query answering performance, compared to the existing relational representations of MDDBs, namely star and snowflake schemas.

8.3 Inconsistency Detection

Relational databases are repaired when integrity constraints are violated. Hence, when it comes to repairs, an important property of a relational schema is its efficiency in capturing these violations. The performance of the relational schema in inconsistency detection depends on several factors such as the relational schema design, the type of ICs and the size of the relational instance.

We discussed that one of the important properties of an expressive MDDB relational representation is its efficiency in checking the summarizability conditions through ICs. In this section, we will investigate the time it takes to check strictness and homogeneity of the `Customer` dimension in each of the star, snowflake and path instances.

We use the data generator program to generate random p-instances for the `Customer` dimension instance. The number of p-instances is limited to 70,000. Around 600,000 cases of non-strictness can be found in the generated data. On the other hand, 20,000 of these p-instances resemble a case of heterogeneity, i.e. they contain a `NULL` in one of their elements. Recall from Section 3.2 that the star schema is not expressive enough to capture all cases of heterogeneity. In the first phase of our experiment, only half of these can be detected in star schema using `NOT NULL` constraints. In this phase, we are simulating a common situation, in which the sample MD instance contains heterogeneity cases that might not be detectable in the star schema.

We generated a considerable number of non-strictness instances, so that we can clearly observe the difference between evaluating FDs and EGDs in star or path schema, and observing the impact of performing a series of join operations in snowflake schema. In the case of homogeneity, the types of ICs used in star, snowflake and path schemas are all the same, i.e. `NOT NULL` constraints. Hence, our only concern in checking homogeneity is to verify the weakness of the star schema in capturing heterogeneous instances completely. In other words, we would like to see the difference in performance of the star schema compared to the path or snowflake schemas in detecting heterogeneity cases.

Chapters 3 and 4 described the mechanism for translating summarizability conditions to ICs over star, snowflake and path databases, respectively. We defined a stored procedure for each of these databases, to check the aforementioned ICs by using SQL queries. More specifically, for each of these ICs, we wrote an SQL query which returns the tuples violating that IC.

The ICs in Example 3.2 detect non-strictness of the `Customer` dimension in the star schema. We can also use `NOT NULL` constraints to detect some of the heterogeneity cases. The stored procedure for checking summarizability of the `Customer` dimension in star database is shown below.

In general, a functional dependency $A \rightarrow B$ can be checked by searching for tuples in which the value for attribute A is the same, but the values for attribute B differ. This idea is used for detecting non-strictness in the star schema. On the other hand, heterogeneity can be detected by searching for tuples that have `NULL` value for attributes that are restricted by `NOT NULL` constraint.

```
CREATE PROCEDURE checkStarSchema
AS
BEGIN
--CUSTOMER
-- cus -> gen
select * from dbo.CustomerStar as c1, dbo.CustomerStar as c2
where (c1.CustomerCol = c2.CustomerCol and c1.GenderCol <> c2.GenderCol)

--cus -> loc
select * from dbo.CustomerStar as c1, dbo.CustomerStar as c2
where (c1.CustomerCol = c2.CustomerCol and c1.LocationCol <> c2.LocationCol)
```

```
--cus -> seg
select * from dbo.CustomerStar as c1, dbo.CustomerStar as c2
where (c1.CustomerCol = c2.CustomerCol and c1.SegmentCol <> c2.SegmentCol)

--gen->seg
select * from dbo.CustomerStar as c1, dbo.CustomerStar as c2
where (c1.GenderCol = c2.GenderCol and c1.SegmentCol <> c2.SegmentCol)

--loc->seg
select * from dbo.CustomerStar as c1, dbo.CustomerStar as c2
where (c1.LocationCol = c2.LocationCol and c1.SegmentCol <> c2.SegmentCol)

--gen not null
select * from dbo.CustomerStar where GenderCol Is NULL

--loc not null
select * from dbo.CustomerStar where LocationCol Is NULL

--seg not null
select * from dbo.CustomerStar where SegmentCol Is NULL

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

The ICs for checking strictness and homogeneity of the `Customer` dimension over the path schema can be found in Examples 4.6 and 4.8, respectively. The following stored procedure searches for any tuple that violates these ICs in the path instance. The procedure for checking FDs, EGDs and `NOT NULL` constraints are similar to the one explained for the star database.

```
CREATE PROCEDURE checkPathSchema
```

```
AS
BEGIN
--CUSTOMER
--cus->gen
select * from dbo.CustomerPath1 as c1, dbo.CustomerPath1 as c2
where (c1.CustomerCol = c2.CustomerCol and c1.GenderCol <> c2.GenderCol)

--gen->seg
select * from dbo.CustomerPath1 as c1, dbo.CustomerPath1 as c2
where (c1.GenderCol = c2.GenderCol and c1.SegmentCol <> c2.SegmentCol)

--cus->loc
select * from dbo.CustomerPath2 as c1, dbo.CustomerPath2 as c2
where (c1.CustomerCol = c2.CustomerCol and c1.LocationCol <> c2.LocationCol)

--loc->seg
select * from dbo.CustomerPath2 as c1, dbo.CustomerPath2 as c2
where (c1.LocationCol = c2.LocationCol and c1.SegmentCol <> c2.SegmentCol)

--cus->seg
select * from dbo.CustomerPath1 as c1, dbo.CustomerPath1 as c2
where (c1.CustomerCol = c2.CustomerCol and c1.SegmentCol <> c2.SegmentCol)

select * from dbo.CustomerPath2 as c1, dbo.CustomerPath2 as c2
where (c1.CustomerCol = c2.CustomerCol and c1.SegmentCol <> c2.SegmentCol)

select * from dbo.CustomerPath1 as c1, dbo.CustomerPath2 as c2
where (c1.CustomerCol = c2.CustomerCol and c1.SegmentCol <> c2.SegmentCol)

--gen not null
select * from dbo.CustomerPath1 where GenderCol Is NULL
```

```
--loc not null
select * from dbo.CustomerPath2 where LocationCol Is NULL

--seg not null
select * from dbo.CustomerPath1 where SegmentCol Is NULL
select * from dbo.CustomerPath2 where SegmentCol Is NULL

--all not null
select * from dbo.CustomerPath1 where AllCol Is NULL
select * from dbo.CustomerPath2 where AllCol Is NULL
END
```

For the snowflake schema, the ICs in Examples 3.4 and 3.5 are used to check summarizability of the `Customer` dimension. The satisfaction of these ICs are checked by the following stored procedure. Notice that, as explained in Example 3.4, non-strictness can be detected by performing a series of join operations in the snowflake database. More specifically, for checking strictness between categories c_i and c_j, we first have to join the tables to obtain the roll up relation $\mathcal{R}_{c_i}^{c_j}$, and then join the roll up relation with itself in order to find those tuples in which A^{c_i} is the same but A^{c_j} is different. The process for checking homogeneity is similar to what we had for the star and path schemas.

```
CREATE PROCEDURE checkSnowflakeSchema
AS
BEGIN
--CUSTOMER
--cus -> loc
select * from
(select c.CustomerCol,l.LocationCol from
dbo.CustomerSnowflake as c inner join dbo.LocationSnowflake as l
on c.LocationFk = l.LocationCol ) as t1 ,
(select c.CustomerCol,l.LocationCol from
dbo.CustomerSnowflake as c inner join dbo.LocationSnowflake as l
on c.LocationFk = l.LocationCol ) as t2
where t1.CustomerCol = t2.CustomerCol and t1.LocationCol <> t2.LocationCol

--cus -> gen
select * from
(select c.CustomerCol,g.GenderCol from
dbo.CustomerSnowflake as c inner join dbo.GenderSnowflake as g
on c.GenderFK = g.GenderCol ) as t1 ,
(select c.CustomerCol,g.GenderCol from
dbo.CustomerSnowflake as c inner join dbo.GenderSnowflake as g
on c.GenderFK = g.GenderCol ) as t2
where t1.CustomerCol = t2.CustomerCol and t1.GenderCol <> t2.GenderCol

--cus -> seg
select * from
((select c.CustomerCol,s.SegmentCol from dbo.CustomerSnowflake as c
inner join dbo.GenderSnowflake as g on c.GenderFK = g.GenderCol
inner join dbo.SegmentSnowflake as s
on g.SegmentFK = s.SegmentCol) union
(select c.CustomerCol,s.SegmentCol from dbo.CustomerSnowflake as c
inner join dbo.LocationSnowflake as l on c.LocationFk = l.LocationCol
inner join dbo.SegmentSnowflake as s
on l.SegmentFK = s.SegmentCol)) as t1,
((select c.CustomerCol,s.SegmentCol from dbo.CustomerSnowflake as c
inner join dbo.GenderSnowflake as g on c.GenderFK = g.GenderCol
inner join dbo.SegmentSnowflake as s
on g.SegmentFK = s.SegmentCol) union
(select c.CustomerCol,s.SegmentCol from dbo.CustomerSnowflake as c
inner join dbo.LocationSnowflake as l on c.LocationFk = l.LocationCol
inner join dbo.SegmentSnowflake as s
on l.SegmentFK = s.SegmentCol)) as t2
```

```
where t1.CustomerCol = t2.CustomerCol and t1.SegmentCol <> t2.SegmentCol

--loc -> seg
select * from
(select l.LocationCol,s.SegmentCol from
dbo.LocationSnowflake as l inner join dbo.SegmentSnowflake as s
on l.SegmentFK = s.SegmentCol) as t1 ,
(select l.LocationCol,s.SegmentCol from
dbo.LocationSnowflake as l inner join dbo.SegmentSnowflake as s
on l.SegmentFK = s.SegmentCol) as t2
where t1.LocationCol = t2.LocationCol and t1.SegmentCol <> t2.SegmentCol

--gen -> seg
select * from
(select g.GenderCol,s.SegmentCol from
dbo.GenderSnowflake as g inner join dbo.SegmentSnowflake as s
on g.SegmentFK = s.SegmentCol) as t1 ,
(select g.GenderCol,s.SegmentCol from
dbo.GenderSnowflake as g inner join dbo.SegmentSnowflake as s
on g.SegmentFK = s.SegmentCol) as t2
where t1.GenderCol = t2.GenderCol and t1.SegmentCol <> t2.SegmentCol

--loc not null
select * from dbo.CustomerSnowflake where LocationFk Is NULL

--gen not null
select * from dbo.CustomerSnowflake where GenderFK Is NULL

--seg not null
select * from dbo.GenderSnowflake where SegmentFK Is NULL
select * from dbo.LocationSnowflake where SegmentFK Is NULL
```

```
--all not null
select * from dbo.SegmentSnowflake where AllFK Is NULL
END
```

The times obtained for retrieving cases of non-strictness in star, snowflake and path database are 17.007 sec, 1200 sec and 15.686 sec, respectively. These results are in line with our discussions on the efficiency of checking strictness in these schemas, in Sections 3.2, 3.4 and 4.5. Notice the huge difference between the execution time for snowflake compared to the star and path schemas. Recall that the hierarchical structure of the snowflake database complicates the process of checking strictness. Unlike simple constraints for star and path databases, here we have to perform a series of join operations between different tables to detect non-strictness. The negligible difference between the execution times of star and path database is due to different dimension table schemas and number of tuples belonging to each of these table. Notice that the additional EGDs used for the path schema for detecting non-strictness do not considerably affect the performance of inconsistency detection.

The time taken for detecting cases of heterogeneity is 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 cases is due to the fact that the queries executed over star schema can return only half of the heterogeneity instances, because the star schema cannot *completely* represent heterogeneity. On the other hand, since in path schema a category might be represented by more than one table (like `Segment` and `All` in the `Customer` dimension as represented by the path schema of Figure 4.1), we need more `NOT NULL` constraints for the path compared to the snowflake schema. That is why, in the snowflake schema, cases of heterogeneity are retrieved faster than for the path schema.

The previous results are somehow unfair, in the sense that they do not reflect the weakness of the star schema in capturing heterogeneity. In other words, we would like to see the performance of star, snowflake and path schemas when equal number of heterogeneity cases are represented in these databases. Hence, on the second phase of our experiment, we narrowed down the heterogeneity cases of our data set to those that are detectable in a star database. More specifically, we generated 20,000 cases of heterogeneity, which could all be captured using `NOT NULL` constraints in star schema. At this phase, we are only interested in checking heterogeneity in star, snowflake and path instances. As a result, we expect to have more realistic results. The time measured for detecting heterogeneity instances in this phase is 750 ms for star schema, 962 ms for path schema, and 758 ms for snowflake schema. As expected, the performance of the star schema deteriorated. Notice that the difference between the execution time in path and the execution time in star or snowflake schemas comes from the fact that we need more `NOT NULL` constraints for the path schema when capturing heterogeneity, compared to star and snowflake schemas.

In general, it can be said that the path schema is a MDDB relational representation that can be used for checking summarizability constraints through ICs *completely* and *efficiently*.

8.4 Summary

This chapter studied the path schema as a relational implementation of MDDBs. We showed that it has an acceptable query answering performance compared to the star and snowflake schemas. In particular, it in in the second best place, with a negligible difference behind star schema. We also investigated the performance of path schema in checking summarizability constraints through ICs. In particular, we compared the times it takes to detect IC violations in star, snowflake and path instances. The experimental results showed that the path approach can efficiently find all cases of non-strictness and heterogeneity through relational ICs.

Chapter 9

Conclusions

In this thesis, we were concerned with using relational repairs for restoring summarizability in multidimensional databases. Our methodology was to represent the MD instance as a relational instance, repair the inconsistent relational database in case of non-summarizability, and then retrieve a new dimension instance from the repaired relational database.

The feasibility of our approach depends heavily on how a multidimensional database is represented as a relational database. Summarizability constraints were captured through relational integrity constraints which can be *efficiently* checked. Moreover, our MD2R mapping has *no information loss* when translating the multidimensional database into the relational database.

We studied the two well-known relational implementations for MDDBs, star and snowflake. Our discussions showed that, based on the aforementioned criteria, star and snowflake are not the perfect choice for our purposes. As a result, we proposed a relational reconstruction of the multidimensional database via path relational schema. We argued that the dimension schema, dimension instance, and summarizability conditions can all be efficiently represented in this new relational representation. In particular, we ran some experiments on the performance of the path schema in detecting non-summarizability through ICs. The results showed that, unlike star or snowflake, path can completely and efficiently find cases of inconsistencies at the relational level.

As part of our proposal for a new relational implementation of MDDBs, we studied query answering performance of the path schema. In particular, we ran several experiments comparing query answering time in star, snowflake and path databases. Our results revealed that path comes second closely after star in query answering performance.

In the future, we would like to study path schema in more details. In particular, we are interested in comparing the star, snowflake and path schemas using data-related metrics, such as database size.

Based on the aforementioned studies, we can propose different optimization techniques for the path schema, such as normalization. Due to the specific design of the path schema, we might have data redundancy in the relational database. The B2A paths of a dimension schema might have several categories in common. In the best case, they only share the first and last element in a path, i.e. base category and category `All`. In the relational side, these common categories are mapped to attributes that are common between more than one path table. Due to these common attributes, we might have data redundancy in the path database. For instance, it can be easily checked that, if the number of base elements in a dimension instance is huge, this data redundancy can become a serious issue. Notice that, this problem does not occur in star or snowflake databases.

A possible solution to the aforementioned data redundancy problem would be to keep the shared attributes in separate tables. For instance, we can keep the attribute representing the base category in a separate table, and relate the path tables to this attribute with referential constraints. However, we might have some complications in the summarizability checking process due to this specific database design.

Having found a good relational representation for MDDBs, we focused on repairing the inconsistent path database in case of non-summarizability. We introduced the repair semantics used in our methodology for obtaining the set of minimal relational repairs. To this end, we took advantage of the numerous works done in the area of relational database repairs (see [12, 26] for a survey). The new MDDB relational implementation is proved to be easily repairable via attribute updates.

The feasibility of using relational repairs for restoring MD consistency depends on the invertibility of MD2R mapping, since a repaired database instance must be translated back to a multidimensional instance. The existence of such inverse mapping was determined by the fact that, no information is lost when mapping the MD instance into the relational instance. We argued that, path mapping is uniquely invertible. This fact assures that the repair process could be implemented directly on relational platforms.

By comparing the MD repairs obtained through our approach with those generated by [18], we showed that the proposed relational approach produces MD repairs that correspond to minimal relational repairs. We managed to characterize these repairs in pure MD terms. This characterization shows that, unlike [18], we somehow prioritize the dimension instance edges for modification during the repair process.

Most of the existing MD repair approaches address only non-strictness or heterogeneity, assuming that the other summarizability condition is satisfied in the original dimension [13, 22, 41, 42, 44]. Instead, we proposed a *general* solution to handle non-summarizability, without making any initial assumptions.

The concept of *local* summarizability constraints was introduced in [18, 24]. The idea is that, in some cases, we want to check strictness and homogeneity between certain categories, and not necessarily in the whole dimension. Fortunately, our approach is not restricted to global constraint satisfaction, since the path mapping translates each local constraint separately. In fact, our methodology is *flexible* enough to be used for restoring summarizability both locally and globally.

Notice that, although the proposed instance-based repair process performs some changes to the dimension instance, it does not invalidate *all* of the pre-computed aggregate query results on the dimension. Inserting/deleting a number of edges in the dimension instance during the MD repair process, affects a set of roll up relations in the dimension instance. Hence, only those pre-computed aggregate query results that depend on the affected roll up relations should be re-computed. In other words, the proposed MD repair approach does not cause any inefficiency in maintaining aggregate query results.

One interesting problem in non-summarizable MDDBs is *consistent query answering* (CQA). CQA was first introduced by [4], in the context of relational databases. It refers to the process of retrieving query results that are *consistent* with respect to a given set of integrity constraints; although the relational database as a whole may be *inconsistent.* CQA for aggregate queries with scalar functions under the range semantics was introduced and analyzed in [6]. These queries are similar to the aggregate queries posed to ROLAP systems.

In order to adopt this concept in MDDBs, [13] proposed the notion of *canonical instance*; a dimension which is obtained by somehow merging the minimal repairs of a dimension. This dimension is used for finding consistent answers to queries in a non-summarizable MDDB.

For future work remains the study of consistent query answering in MDDBs. Unlike [13], we are interested in taking advantage of the notions of repair and consistent query answering in relational databases, to solve this problem. In particular, we can apply the CQA techniques to the set of minimal relational repairs for an inconsistent path database, and then translate the consistent relational query answers into consistent multidimensional results. In the context of CQA and inspired by [13], we are also interested in characterizing a canonical instance for the set of minimal relational repairs obtained for an inconsistent path database. It would be interesting to compare the dimension instance representing this relational canonical instance with the MD canonical instance proposed in [13].

Updating the dimension instances is very common in MDDBs. An outcome of this process is that, the computed MD repairs can no longer be used in the CQA process. Obviously, it is not efficient to frequently re-compute the MD repairs. One of our future goals is to maintain the MD repairs under dimension updates. In particular, we would like to study how we can update the MD repairs, based on the original dimension instance update operation.

It would be interesting to consider *aggregation constraints* in the MD scenario, in addition to summarizability constraints. To this end, we can take advantage of the existing results on attribute-based repairs under aggregation constraints [33].

Currently, we are also running experiments in relation to the repair aspects of our 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* (ASP) similar to those in [18, 24]. They work directly with the MD representation. In our case, given the different MD repair semantics (the one in Theorem 7.1), our program would include *weak constraints with weights*, that extend the ASP paradigm [19]. 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 [18, 24] is set-theoretical, not numerical).

The other way to go is based on the repairing of the relational instances obtained via the MD2R mapping wrt relational integrity constraints. ASP have been also successfully used to compute relational repairs and do consistent query answering (cf. [23] 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. ASPs of this kind have been used in [36].

A problem with the proposed solution is that, it creates a *new* relational layer on top of the MDDB. In other words, it can not be applied to MDDBs that are already implemented, either as a star or snowflake database. Due to the large number of dimensions, re-creating the relational layer of MDDB using path schema seems to be costly. In particular, migrating the data from the star or snowflake tables into the path tables is not a straightforward process. So, we would like to see if the proposed path schema can be defined as a *virtual view* on top of the existing star or snowflake schema. This view is somehow similar to the *logical layer* proposed in [21], where the logical layer separates the MDDB from its physical implementation.

The aforementioned goal can be formalized as a *schema evolution* problem [7, 28]. In this problem, we know how to map schema $\mathcal{S}_1$ to schemas $\mathcal{S}_2$ and $\mathcal{S}_3$ separately, and we are interested in finding the mapping rules for moving from $\mathcal{S}_2$ to $\mathcal{S}_3$. According to our discussions in Chapters 3 and 4, we already have the rules for mapping the multidimensional database to a star, snowflake or path database. In order to have the aforementioned virtual view, we must figure out how to map a star or snowflake database to a path database.

We believe that a deeper investigation of the relationship between our MD repairs and other existing approaches to MD repairs is still crucial. In particular, we are interested in comparing our data-based repair mechanism to schema-based MD repairs [8]. One interesting aspect of this comparison can be the application of these MD repairs in the CQA process. In particular, it would be interesting to compare the consistent answers obtained using our instance-based repairs and those generated based on existing schema-based repairs. In addition, the combination of both instance-based and schema-based approaches for restoring summarizability should also be investigated.

Bibliography

[1] A. Abelló, J. Samos, and F. Saltor. YAM^2: a Multidimensional Conceptual Model Extending UML. *Information Systems*, 31(6):541–567, 2006.

[2] S. Abiteboul, S. Cluet, V. Christophides, T. Milo, G. Moerkotte, and J. Siméon. Querying Documents in Object Databases. *International Journal on Digital Libraries*, 1(1):5–19, 1997.

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

[4] 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.

[5] M. Arenas, L. Bertossi, and J. Chomicki. Scalar Aggregation in FD-Inconsistent Databases. In *Proceedings of the International Conference on Database Theory (ICDT01)*, volume 1973 of *Lecture Notes in Computer Science*, pages 39–53. Springer, 2001.

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

[7] M. Arenas, J. Perez, J. Reutter, and C. Riveros. Foundations of Schema Mapping Management. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS10)*, pages 227–238, 2010.

[8] S. Ariyan and L. Bertossi. Structural Repairs of Multidimensional Databases. In *Proceedings of the 5th Alberto Mendelzon International Workshop on Foundations of Data Management (AMW11)*, volume 749 of *CEUR Workshop Proceedings.* CEUR-WS.org, 2011.

[9] A. Bauer, W. Hümmer, and W. Lehner. An Alternative Relational OLAP Modeling Approach. In *Proceedings of the Second International Conference on Data Warehousing and Knowledge Discovery (DaWaK2000)*, volume 1874 of *Lecture Notes in Computer Science*, pages 189–198. Springer, 2000.

[10] C. Beeri and M. Y. Vardi. A Proof Procedure for Data Dependencies. *Journal of the ACM*, 31(4):718–741, 1984.

[11] P. A. Bernstein. Applying Model Management to Classical Meta Data Problems. Presented at the First Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 5-8, 2003.

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

[13] 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 (AMW09*, volume 450 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2009.

[14] 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.

[15] 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.

[16] K. Boulil, S. Bimonte, H. Mahboubi, and François Pinet. Towards the Definition of Spatial Data Warehouses Integrity Constraints with Spatial OCL. In *Proceedings of the 13th International Workshop on Data Warehousing and OLAP (DOLAP10)*, pages 31–36. ACM, 2010.

[17] 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.

[18] 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.

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

[20] 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.

[21] 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.

[22] M. Caniupan. Handling Inconsistencies in Data Warehouses. In *Current Trends in Database Technology*, volume 3268 of *Lecture Notes in Computer Science*, pages 166–176. Springer, 2004.

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

[24] M. Caniupan, L. Bravo, and C. A. Hurtado. Logic Programs for Repairing Inconsistent Dimensions in Data Warehouses. Submitted to Journal, Jan 2010.

[25] F. Carpani and R. Ruggia. An Integrity Constraints Language for a Conceptual Multidimensional Data Model. Presented at the Thirteenth International Conference on Software Engineering and Knowledge Engineering (SEKE'2001), Sheraton Buenos Aires Hotel, Buenos Aires, Argentina, June 13-15, 2001.

[26] 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.

[27] V. Christophides, S. Abiteboul, S. Cluet, and M. Scholl. From Structured Documents to Novel Query Facilities. *ACM SIGMOD Record*, 23:313–324, 1994.

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

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

[30] R. Fagin, P. G. Kolaitis, L. Popa, and W-C. Tan. Reverse Data Exchange: Coping with Nulls. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS09)*, pages 23–32, 2009.

[31] R. Fagin and M. Y. Vardi. The Theory of Data Dependencies - an Overview. In *Proceedings of the 11th Colloquium on Automata, Languages and Programming*, volume 172 of *Lecture Notes in Computer Science*, pages 1–22. Springer, 1984.

[32] S. Flesca, F. Furfaro, and F. Parisi. Consistent Query Answers on Numerical Databases Under Aggregate Constraints. In *Proceedings of the 10th Database Programming Languages (DBLP05)*, volume 3774 of *Lecture Notes in Computer Science*, pages 279–294. Springer, 2005.

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

[34] D. Florescu and D. Kossmann. Storing and Querying XML Data using an RDMBS. *IEEE Data Eng. Bull.*, 22(3):27–34, 1999.

[35] E. Franconi and A. Kamble. The GMD Data Model and Algebra for Multidimensional Information. In *Proceedings of the 16th International Conference on Advanced Information Systems Engineering (CAiSE)*, volume 3084 of *Lecture Notes in Computer Science*, pages 446–462. Springer, 2004.

[36] 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*, volume 2250 of *Lecture Notes in Computer Science*, pages 561–578. Springer, 2001.

[37] E. Franconi and G. Ng. The i.com Tool for Intelligent Conceptual Modeling. In *Proceedings of the 7th International Workshop on Knowledge Representation meets Databases (KRDB2000)*, volume 29 of *CEUR Workshop Proceedings*, pages 45–53. CEUR-WS.org, 2000.

[38] E. Franconi and U. Sattler. A Data Warehouse Conceptual Data Model for Multidimensional Aggregation. In *Proceedings of the International Workshop on Design and Management of Data Warehouses (DMDW)*, volume 19 of *CEUR Workshop Proceedings*. CEUR-WS.org, 1999.

[39] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M.i Venkatrao, F. Pellow, and H. Pirahesh. Data Cube: A Relational Aggregation Operator

Generalizing Group-by, Cross-Tab, and Sub Totals. *Data Mining and Knowledge Discovery*, 1(1):29–53, 1997.

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

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

[42] 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.

[43] 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.

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

[45] 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.

[46] 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.

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

[48] H. V. Jagadish, L. V. S. Lakshmanan, and D. Srivastava. What can Hierarchies Do for Data Warehouses? In *Proceedings of 25th International Conference on Very Large Data Bases (VLDB99)*, pages 530–541. Morgan Kaufmann, 1999.

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

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

[51] J. Lechtenbörger and G. Vossen. Multidimensional Normal Forms for Data Warehouse Design. *Information Systems*, 28(5):415–434, 2003.

[52] 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.

[53] 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.

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

[55] 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 of *Lecture Notes in Computer Science*, pages 179–193. Springer, 2007.

[56] J. Mazón, J. Lechtenbörger, and J. Trujillo. Solving Summarizability Problems in Fact-dimension Relationships for Multidimensional Models. In *Proceedings of the 11th International Workshop on Data Warehousing and OLAP (DOLAP08)*, pages 57–64. ACM, 2008.

[57] S. Melnik, P. A. Bernstein, A. Y. Halevy, and E. Rahm. Supporting Executable Mappings in Model Management. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 167–178. ACM, 2005.

[58] Microsoft Corporation. Multidimensional Expressions (MDX) Reference. `http://msdn.microsoft.com/en-us/library/ms145506.aspx`, 2008.

[59] T. Niemi, J. Nummenmaa, and P. Thanisch. Logical Multidimensional Database Design for Ragged and Unbalanced Aggregation. In *Proceedings of the 3rd International Workshop on Design and Management of Data Warehouses (DMDW01)*, volume 39 of *CEUR Workshop Proceedings*, page 7. CEUR-WS.org, 2001.

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

[61] 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.

[62] 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.

[63] T. B. Pedersen, C. S. Jensen, and C. E. Dyreson. A Foundation for Capturing and Querying Complex Multidimensional Data. *Information Systems*, 26(5):383–423, 2001.

[64] 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.

[65] J. Shanmugasundaram, K. Tufte, C. Zhang, G. He, D. J. DeWitt, and J. F. Naughton. Relational Databases for Querying XML Documents: Limitations and Opportunities. In *Proceedings of 25th International Conference on Very Large Data Bases (VLDB99)*, pages 302–314. Morgan Kaufmann, 1999.

[66] 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.

[67] G. Spofford, S. Harinath, C. Webb, D. H. Huang, and F. Civardi. *MDX Solutions: with Microsoft SQL Server Analysis 2005 and Hyperion Essbase.* Wiley, 2nd edition, 2006.

[68] J. Trujillo, M. Palomar, J. Gómez, and Il-Yeol Song. Designing Data Warehouses with OO Conceptual Models. *IEEE Computer*, 34(12):66–75, 2001.

[69] N. Tryfona, F. Busborg, and J. G. Christiansen. StarER: A Conceptual Model for Data Warehouse Design. In *Proceeding of the Second International Workshop on Data Warehousing and OLAP (DOLAP99)*, pages 3–8. ACM, 1999.

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

[71] J. Wijsen. Condensed Representation of Database Repairs for Consistent Query Answering. In *Proceedings of the International Conference on Database Theory*

(ICDT03), volume 2572 of *Lecture Notes in Computer Science*, pages 375–390. Springer, 2003.

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

[73] 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.