Discovering and merging related analytic datasets

Rutian Liu, Eric Simon, Bernd Amann, Stéphane Gançarski

Version of Record: https://www.sciencedirect.com/science/article/pii/S0306437920300065
Manuscript_9569ef8131d38bb44723071ffbd55797

Discovering and Merging Related Analytic Datasets*

Rutian Liu[a,*], Eric Simon[b], Bernd Amann[c], Stéphane Gançarski[d]

[a]SAP France, LIP6, Paris, France
[b]SAP France, Paris, France
[c]LIP6, Sorbonne Université, CNRS, Paris, France
[d]LIP6, Sorbonne Université, CNRS, Paris, France

Abstract

The production of analytic datasets is a significant big data trend and has gone well beyond the scope of traditional IT-governed dataset development. Analytic datasets are now created by data scientists and data analysts using big data frameworks and agile data preparation tools. However, despite the profusion of available datasets, it remains quite difficult for a data analyst to start from a dataset at hand and customize it with additional attributes coming from other existing datasets. This article describes a model and algorithms that exploit automatically extracted and user-defined semantic relationships for extending analytic datasets with new atomic or aggregated attribute values. Our framework is implemented as a REST service in SAP HANA and includes a careful theoretical analysis and practical solutions for several complex data quality issues.

Keywords: schema augmentation, schema complement, data quality, SAP HANA

1. Introduction

1.1. Context and motivation

Enterprise Business Intelligence (BI) provides business users with solutions for managed data reporting, analysis, aggregation and visualization. These solutions heavily rely on trusted and well documented analytic datasets comprising multidimensional *facts* that hold *measures* and refer to one or more hierarchical *dimensions* [1]. Traditionally, these analytic datasets are created by the IT department in the form of data warehouses and data marts [2] or by enterprise application software vendors in the form of predefined and customizable analytic models. For example, SAP provisions thousands of predefined and customizable analytic datasets, also known as "virtual data models" for various business application domains (e.g., SCM, CRM, ERP) [3, 4]. These datasets are defined as views over the transactional data stored and managed by the SAP S4/HANA business suite and carry information, including sophisticated measures, which is easily understandable by business users, and ready for consumption by BI tools. Recently, agile data preparation tools [5, 6, 7] emerged to empower business users and data scientists to easily create their own high-quality analytic datasets from transactional data and existing analytic datasets and to build insightful and interactive personalized data visualizations.

Data preparation tools have the ability to partially automate data cleaning and transformation operations, perform record linkage and duplicate elimination, and even extract structured information from unstructured data. Another typical requirement of business users or data scientists is to *augment* the schema of an existing analytic dataset with new attributes, coming from one or more semantically related datasets that may represent additional details on dimensions or new measures. This is a critical need in many scenarios such as the creation of data mashups (e.g., add demographics to sales information), or the engineering of features within datasets used to train predictive models (e.g., features engineering to learn identifying customers that are likely to sign up for a credit card in the following quarter). Despite the importance of this task, existing data preparation tools poorly support users in augmenting the schema of an analytic dataset. This lack of assistance compels business users to redefine multiple times similar analytic datasets in a possibly inconsistent manner or to depend on their IT department to create their customized datasets. This creates an important bottleneck on the IT organization and pushes forward the deadline for making the desired datasets available to business users.

Addressing the problem of agile schema augmentation in the context of building trusted analytic datasets for specific BI tasks is a major business opportunity for several reasons. First, companies generally manage large collections of analytic datasets which keep growing with the need for new business data analysis tasks. At the scale of a large company, the facility offered by agile data preparation

*This work was supported by the French Association Nationale de la Recherche Technique (ANRT) [grant CIFRE number 2016/0644];
*Corresponding author
Email addresses: rutian.liu@lip6.fr (Rutian Liu), eric.simon@sap.com (Eric Simon), bernd.amann@lip6.fr (Bernd Amann), stephane.gancarski@lip6.fr (Stéphane Gançarski)

tools increases considerably the number of available analytic datasets with respect to those created by the IT organization or those predefined by enterprise software application vendors. Second, semantic relationships between analytic datasets, which are essential to support schema augmentation, can be accurately and automatically extracted from the dataset definitions. Indeed, analytic datasets are often defined over other datasets using queries, scripts, or views. For example, in BI applications supported by SAP, it is common to find analytic datasets with five or more levels of nested view definitions. By parsing these view definitions it is possible to discover the dimensions shared by different analytic datasets. Third, analytic datasets are generally complemented by rich and carefully designed metadata (e.g., about which attributes represent levels of hierarchical dimensions, or about the units and currencies of measure attributes) to support complex data analysis scenarios, that can be exploited during the schema augmentation process. Finally, IT organization and business users invest time to create analytic datasets containing high quality data (i.e., clean data) for business process optimization and decision making. Reusing these datasets and their metadata for schema augmentation is therefore worthwhile. For all these reasons, analytic datasets are a "gold mine" of high-quality and interrelated data that is relevant to business users and data scientists, although under-exploited by current data preparation tools.

Table 1: Sales

ORG_ID	CITY	COUNTRY	YEAR	REVENUE
SALESORG			*TIME*	(M)
Oh_01	Dublin	USA	2018	3.2
Ca_01	Dublin	USA	2018	5.3
Ie_01	Dublin	Ireland	2018	45.1

1.2. Main challenges

The manual augmentation of an analytic dataset is often a cumbersome and error-prone process, raising multiple challenges illustrated hereafter. Imagine a simple use case with two analytic datasets represented by fact tables Sales and Dem(ographics) as shown in Tables 1 and 2. Attributes ORG_ID, CITY and COUNTRY in table Sales come from dimension table *SALESORG*, attribute YEAR in tables Sales and Dem is from dimension table *TIME*, and attributes CITY, STATE, COUNTRY in table Dem are from dimension table *REGION*. Dimension table names are in italic font to distinguish them from fact tables. Assume that attribute ORG_ID is unique in Sales while attributes CITY, STATE, COUNTRY, and YEAR are unique in Dem. In addition, assume that *SALESORG* and *REGION* are also related through common attributes CITY and COUNTRY which means that these attributes have the same meaning in both tables. Then, tables Sales and Dem are naturally related through their common attribute YEAR which comes from the same dimension and attributes CITY and COUNTRY which come from different dimensions but are semantically equivalent.

Table 2: Dem (Demographics)

CITY	STATE	COUNTRY	YEAR	POP	UNEMP
REGION			*TIME*	(K)	(%)
Dublin	Ohio	USA	2018	61	2.5
Dublin	California	USA	2018	42	3.1
Dublin	-	Ireland	2018	527	5.7
San Jose	California	USA	2018	1,035	2.3

In our example, all tables are defined as views over transactional data as shown on Figure 1. Dimension tables are represented by rounded rectangles, fact tables by square rectangles. Plain edges represent data dependencies, expressing for instance that the definition of Sales depends on *SALESORG*, *TIME*, and some transactional data. A data analyst now might want to build a new analytic table SalesDem by augmenting the schema of dataset Sales with the measure attributes POP(ulation) and UNEMP(loyment rate) of dataset Dem. This augmentation then corresponds to yet another fact table (view) SalesDem defined by a left-outer join with Dem on the common attributes CITY, COUNTRY and YEAR.

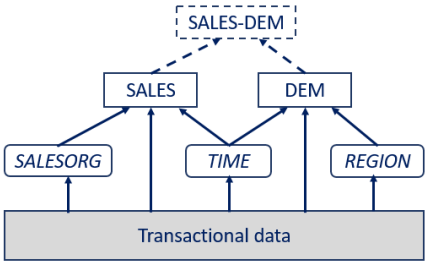


Figure 1: The construction of table SALES – DEM

The goal of this article is to propose a solution for assisting the user in the definition of this new augmentation view. In particular, we will show that by exploiting all available information on analytic datasets, possibly extended by some other user-defined metadata, it is possible to automatically generate for a given dataset, a set of useful and correct schema augmentations. This is done by solving several challenges we will describe below.

Relationship extraction. The first challenge is to discover the relationships between analytic datasets that can be used for schema augmentation. As explained before, many useful relationships can be extracted from the definitions of analytic datasets. For example, the view definition of fact table SALES in Figure 1 uses attributes from dimensions *SALESORG* and *TIME*. Similarly, fact table DEM uses attributes from dimension *TIME*. Thus, relationships can be extracted to identify semantically equivalent attributes of *TIME* in both fact tables and to generate left-outer

join queries like the query for the augmentation SalesDem of table Sales.

Row multiplication. In our example, the outer-join operation increases the number of rows in Sales (*e.g.*, generates two rows with ORG_ID = 'Oh_01') because the join attributes do not constitute a unique identifier in dataset Dem. However, for some application scenarios in feature engineering, data enrichment or data analysis, such a multiplication of rows in Sales is undesirable. There, schema augmentation should be controlled to keep the number of rows in Sales constant, leading to the notion of *schema complement* introduced in [8]. This raises the challenge of identifying row multiplication by computing the unique, and possibly minimal, identifiers of analytic datasets.

Incorrect and ambiguous reduction. A simple solution to avoid row multiplication is to transform table Dem by *reducing* partitions identified by the common attributes with Sales into single tuples. For instance, a possible reduction operation is to pre-aggregate the measures of table Dem along attribute STATE before performing a left outer join. However, this raises two new challenges. The first challenge is to choose the correct aggregation functions that can be applied to the aggregated attributes. For this we need the knowledge that POP can be summed and averaged while only a maximal or minimal value can be computed on the unemployment rate values of attribute UNEMP. Furthermore, if an averaged POP is used to augment Sales, we must determine what aggregation functions are applicable on this new attribute in the augmented Sales dataset to avoid future incorrect aggregation operations. The second challenge is the *ambiguity* of the aggregated measure attributes added to Sales which, for instance, contain an aggregated value for all cities 'Dublin' in country 'USA', thereby destroying the distinction between cities in different states and sharing the same name. Ambiguous measures to should be detected and controlled, *e.g.*, by assigning a *null* value to POP and UNEMP for tuples with ORG_ID = 'Oh_01' and ORG_ID = 'Ca_01'.

Incomplete merge. Another solution to avoid row multiplication would be to add STATE in the schema of Sales (although this is not necessary since ORG_ID was supposed to be unique in Sales). Then the common join attributes would form a unique identifier in Dem and a schema complement for dataset Sales can be obtained without any pre-aggregation. However, even in that case, summing POP in the augmented Sales dataset along CITY will not return the same result as summing POP along CITY in Dem, even for the same values of COUNTRY and STATE in Sales (*e.g.*, POP value of city 'San Jose' in 'California' will not be counted). This can be a problem if in the augmented Sales dataset, REVENUE should be compared with POP in each STATE of a COUNTRY. This issue arises because the merge of Sales with Dem is *incomplete* with respect to Dem and detecting incomplete merge is necessary to avoid such an erroneous analysis.

1.3. Research contributions

This paper presents a new solution to assist business users and data scientists in augmenting the schema of an analytic dataset with attributes coming from other datasets. This is achieved by automatically discovering related analytic datasets and suggesting ways of building a schema augmentations, including schema complements, that resolve the challenges presented before.

More specifically, we make the following technical contributions.

- We introduce attribute graphs as a novel concise and natural way to define literal functional dependencies over the level types of hierarchical dimensions from which we can easily infer unique identifiers in both dimension and fact tables, and minimal identifiers in the case of dimensions.
- We give the formal definitions for schema augmentation, schema complement and merge query in the context of analytic tables. We then present several reduction operations for transforming schema augmentations with row multiplication into schema complements extending each row in the source table by a single row in the augmented table. These operations extend previous contributions on schema augmentation and schema complement (e.g., [8], [9], [10]) to the case of analytic datasets.
- We also present formal quality criteria for schema augmentations, schema complements and merge queries. These criteria are used to define automatic repair operations (1) to notify the generation of ambiguous attributes, (2) to infer applicable aggregation functions on new attributes, and (3) to supplement merge results obtained by incomplete schema augmentation.
- We describe the implementation of our solution as a REST service within the SAP HANA platform and provide a detailed description of our algorithms. We separate the generic part of the algorithms from the specific implementation optimizations done by leveraging the capabilities of SAP HANA.
- We evaluate the performance of our algorithms to compute unique identifiers in dimension and fact tables, and analyze the effectiveness of our REST service using two application scenarios.

1.4. Paper outline

The rest of this paper is structured as follows. In Section 2, we describe the multi-dimensional data model for analytic datasets, which is used to capture all usual concepts of hierarchical dimensions, facts, measures, and cubes [1]. This makes our results easily applicable to any other database system implementing the concepts of multi-dimensional models. We also introduce the new concept of

attribute graph as a way to define *literal functional* dependencies between dimension and fact table attributes and to compute dimension and fact table identifiers. In Section 3, we describe the relationships between analytic tables that are used to support schema augmentation and that can be automatically extracted from the definition of fact and dimensions tables. We then provide the formal definitions of schema augmentation and of natural and reduction-based schema complement. Section 4 introduces formal quality criteria for schema augmentations, schema complements and merge queries. Section 5 describes the implementation of our solution as a REST application service, not in production yet, in the SAP HANA platform while Section 6 details our algorithms. Section 8 presents our experiments to evaluate the performance and effectiveness of our solution. Section 7 compares our research results with previous work on schema augmentation as well as work on the querying of single or multiple OLAP data cubes. We conclude in Section 9.

2. Data Model

This section presents our extension of the relational data model to model *analytic tables* and their semantic relationships. We assume as input to our model any relational database containing a set of *non-analytic tables*. Analytic tables, *i.e.*, *dimension* and *fact* tables, are then defined bottom-up as views over these non-analytic tables and other analytic tables. We use conventional relational database notations [11]. Each *table* T is a finite multiset of tuples over a set of domains of values $\mathrm{S} = \{\mathrm{A}_1, \ldots, \mathrm{A}_n\}$, called *attributes*, where each domain may contain a *null* marker. We call S the *schema* of T.

2.1. Hierarchies and dimension tables

We consider a multidimensional data model with dimensions, *i.e.* hierarchies of values defined by hierarchy types. A *hierarchy type* $\mathbf{H} = (\mathsf{L}, \preccurlyeq)$ is a set of level types $\mathsf{L} = \{L_1, \cdots, L_n\}$ that is organized by a partial order $\preccurlyeq$. L_i is called a child level type of L_j if there exists an edge $L_i \preccurlyeq L_j$ and $L_i \preccurlyeq^* L_j$ denotes that L_i is a descendant level type of L_j. We call all types L_i where there exists no type L_j such that $L_j \preccurlyeq L_i$ or $L_i \preccurlyeq L_j$, respectively the lower and the upper bounds of $\mathbf{H}$.

Example 1. *Two examples of hierarchy types are shown in Figure 2 where an arc from* A *to* B *means that* $A \preccurlyeq B$.

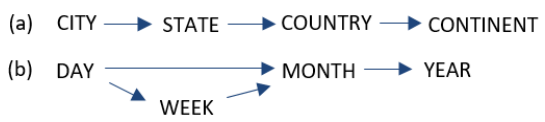


Figure 2: Hierarchy types **GEOGRAPHY** (a) and **TIME** (b)

Each level type L_i represents a domain N_i of values related to the values of the domains N_j of other level types L_j. More precisely, a *hierarchy* $H = (N, \leq)$ of hierarchy type $\mathbf{H} = (\mathsf{L}, \preccurlyeq)$ is a set of values N and a partial order $\leq$ where N contains for each level type $L_i \in \mathsf{L}$ a non empty subset of values $N_i \subseteq N$ such that each order relation $v_i \leq v_j$ preserves the ancestor/descendant relation $\preccurlyeq^*$ between the corresponding hierarchy types L_i and L_j, *i.e.*, $v_i \in N_i, v_j \in N_j \Rightarrow L_i \preccurlyeq^* L_j$. We also assume that $(N, \leq)$ is *transitively reduced*, *i.e.*, there is no pair of nodes that is connected by an edge and a sequence of two or more edges.

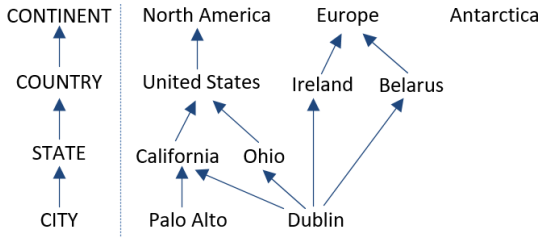


Figure 3: Hierarchy *REGION* of type **GEOGRAPHY**

Example 2. *Figure 3 shows a hierarchy REGION (right part) of type* **GEOGRAPHY** *(left part).*

Hierarchies can naturally be represented in relational tables, called *dimension tables* or, more simply, *dimensions*.

Definition 1 (Dimension table)**.** *Any hierarchy* $H = (N, \leq)$ *of type* $\mathbf{H} = (\mathsf{L}, \preccurlyeq)$ *defines a* dimension *table* $D(\mathrm{S})$ *with a one-to-one mapping* $\phi : \mathsf{L} \to \mathrm{X}$ *from level types* $L_i \in \mathsf{L}$ *to a subset of attributes* $\mathrm{X} \subseteq \mathrm{S}$ *in the schema of* T *such that for each* maximal path $v_1.v_2.\cdots.v_k$ *in* $N_1 \times N_2 \times \cdots \times N_k$ *in* H *there exists a tuple* $t \in T$ *where* $t.\phi(L_1) = v_1, t.\phi(L_2) = v_2, \cdots, t.\phi(L_k) = v_k$ *and* $t.A_j = null$ *for all other attributes in* X.

Example 3. *Dimension table REGION in Table 3 represents the hierarchy REGION of hierarchy type* **GEOGRAPHY** *in Figure 3. Each level type is translated into an attribute of the same name and each tuple represents a maximal path in the hierarchy (null markers are denoted with "-").*

Table 3: Dimension table *REGION*

CITY	STATE	COUNTRY	CONTINENT
Dublin	Ohio	United States	North America
Dublin	California	United States	North America
Dublin	-	Ireland	Europe
Dublin	Ontario	Canada	North America
Dublin	-	Belarus	Europe
Palo Alto	California	United States	North America
-	-	-	Antarctica

All attributes of a dimension table D corresponding to level types are called the *dimension attributes* of D and the

hierarchy type $\mathbf{H} = (\mathsf{L}, \preccurlyeq)$ can naturally be mapped into a *dimension attribute hierarchy* $\mathcal{A} = (\mathrm{S}, \preccurlyeq)$ where $\mathrm{A}_i \preccurlyeq \mathrm{A}_j$ iff $L_i \preccurlyeq L_j$. The remaining attributes are called *detail attributes* and functionally depend on one or more dimension attributes (dimension table *REGION* only contains dimension attributes).

Null markers in dimension tables represent regular values yielding the same interpretation for equality as null values in SQL unique constraints [11]: two attribute values $t_1.\mathrm{A}$ and $t_2.\mathrm{A}$ are *literally equal*, denoted by $t_1.\mathrm{A} \equiv t_2.\mathrm{A}$, if $t_1.\mathrm{A}$ and $t_2.\mathrm{A}$ are equal or both null markers. Literal equality naturally extends to sets of attributes and leads to the notion of *literal functional dependencies (LFD)* [12]. A literal functional dependency LFD $\mathrm{X} \mapsto Z$ holds between two subsets of attributes if for any two tuples t_1 and t_2 of T, when $t_1.\mathrm{X} \equiv t_2.\mathrm{X}$ then $t_1.Z \equiv t_2.Z$. Note that if X does not contain any nullable attribute, which is for instance enforced in SQL by declaring X as a *primary key*, an LFD $\mathrm{X} \mapsto Z$ is equivalent to a functional dependency with nulls.

Definition 2 (Dimension identifier)**.** *Let* $\mathrm{X} \subseteq \mathrm{S}$ *be the set of dimension attributes in a schema* S *and* $\mathcal{L}$ *be a set of LFDs defined on* X*. Then,* $\mathrm{K} \subseteq \mathrm{X}$ *is a* dimension identifier *of* S *if* K *is a minimal set such that* $\mathrm{K} \mapsto \mathrm{S}$ *holds for any instance* T *over* S *satisfying all LFDs in* $\mathcal{L}$*.*

The set of LFDs defined on a dimension table schema characterizes the set of all valid schema instances. However, defining this set of LFDs may be difficult for the designer of a dimension table. We introduce the notion of *attribute graphs*, which are defined over attribute hierarchies and which allow the schema designer to characterize the set of all valid dimension tables by additional semantic properties in a simple and natural way.

Definition 3 (Attribute graph)**.** *An attribute graph over some attribute hierarchy* $\mathcal{A} = (\mathrm{S}, \preccurlyeq)$ *is a directed labeled graph* $\mathcal{D} = (\mathrm{S}, \mathsf{R}, \lambda_{\mathsf{R}}, \bot, \top)$*, where* S *is the set of attributes in* $\mathcal{A}$*,* $\bot$ *and* $\top$ *are two special attributes with empty domains (by definition,* $t.\bot \equiv t.\top \equiv$ *null for all tuples),* $\mathsf{R} \subseteq (\mathrm{S} \cup \{\bot, \top\})^2$ *is a set of edges such that there exists an edge:*

1. $(A_i, A_j) \in \mathsf{R}$ *for each edge* $A_i \preccurlyeq A_j$ *in* $\mathcal{A}$*;*
2. $(\bot, A_i) \in \mathsf{R}$ *for each lower bound in* $\mathcal{A}$ *and*
3. $(A_i, \top) \in \mathsf{R}$ *for each upper bound in* $\mathcal{A}$*.*

There might exist also other edges $(A_i, A_j) \in \mathsf{R}$ *between any two nodes connected by a path in* $\mathcal{D}$*.*

In the following we denote by $\mathsf{R}(\mathrm{A}_i, \mathrm{A}_j) = l$ an edge $(\mathrm{A}_i, \mathrm{A}_j) \in \mathsf{R}$ labeled by $l = \lambda_{\mathsf{R}}(\mathrm{A}_i, \mathrm{A}_j)$. The edge labeling function $\lambda_{\mathsf{R}} : \mathsf{R} \to \{+, \mathbf{1}, \mathbf{f}\}$ assigns to each edge a unique label encoding the presence of functional and literal functional dependency constraints between the connected attributes of a dimension table.

Definition 4 (Valid dimension table)**.** *A dimension table* T *with schema* S *is* valid *with respect to some attribute graph* $\mathcal{D} = (\mathrm{X}, \mathsf{R}, \lambda_{\mathsf{R}})$ *where* $\mathrm{X} \subseteq \mathrm{S}$*, if all following conditions hold in* T*:*

1. *If there exists a tuple* $t \in T$ *such that* $t.A_i$ *is not null, then there exists either an edge* $\mathsf{R}(\bot, A_i)$ *in* $\mathcal{D}$ *or an edge* $\mathsf{R}(A_k, A_i)$ *in* $\mathcal{D}$ *such that* $t.A_k$ *is not null.*
2. *For all tuples* t_1, t_2 *in* T *and all edges* $\mathsf{R}(A_i, A_j)$ *in* $\mathcal{D}$ *the following holds:*
   (a) *If* $\mathsf{R}(A_i, A_j) = \mathbf{f}$*, then* $t_1.A_i \equiv t_2.A_i$ *implies* $t_1.A_j \equiv t_2.A_j$*;*
   (b) *If* $\mathsf{R}(A_i, A_j) = \mathbf{1}$*, then* $t_1.A_i = t_2.A_i$ *implies* $t_1.A_j \equiv t_2.A_j$*.*

In Definition 4, $t_1.\mathrm{A} = t_2.\mathrm{A}$ is *true* if both tuples $t_1.\mathrm{A}$ and $t_2.\mathrm{A}$ are equal non-null values, *false* if both $t_1.\mathrm{A}$ and $t_2.\mathrm{A}$ are different non-null values and *unknown* otherwise. From this definition, it follows naturally that $t_1.\mathrm{A} = t_2.\mathrm{A}$ implies $t_1.\mathrm{A} \equiv t_2.\mathrm{A}$ but not the opposite. Observe also that by Definition 4, $\mathsf{R}(\mathrm{A}_i, \mathrm{A}_j) = \mathbf{f}$ is equivalent to $\mathrm{A}_i \mapsto \mathrm{A}_j$ and $\mathrm{A}_i \to \mathrm{A}_j$ implies $\mathsf{R}(\mathrm{A}_i, \mathrm{A}_j) = \mathbf{1}$ whereas $\mathsf{R}(\mathrm{A}_i, \mathrm{A}_j) = \mathbf{1}$ does not imply $\mathrm{A}_i \to \mathrm{A}_j$. Consequently, if some dimension table T is valid w.r.t. some attribute graph $\mathcal{D}$, it is also valid w.r.t. to all attribute graphs obtained by replacing $\mathbf{f}$ edge labels by $\mathbf{1}$ edge labels and $\mathbf{1}$ edge labels by $+$ edge labels. We can also see that all edges $\mathsf{R}(\bot, \mathrm{A}_i)$ are labeled by $+$ or $\mathbf{f}$. Indeed, $t_i.\bot \equiv t_j.\bot$ holds for all couples (t_i, t_j), and either $t_i.\mathrm{A}_i \equiv t_j.\mathrm{A}_i$ also holds for all couples (t_i, t_j), *i.e.* $\mathsf{R}(\bot, \mathrm{A}_i) = \mathbf{f}$ or not, *i.e.* $\mathsf{R}(\bot, \mathrm{A}_i) = +$ ($t_i.\mathrm{A}_i \equiv t_j.\mathrm{A}_i$ does not hold for at least one couple). Symmetrically, all edges $\mathsf{R}(\mathrm{A}_i, \top)$ are labeled by $\mathbf{f}$ since $t_i.\top \equiv t_j.\top$ for all tuples t_i and t_j.

Example 4. *Figure 4 shows an attribute graph for hierarchy type* **GEOGRAPHY** *that validates dimension table REGION. The lower and upper bound attributes are respectively* CITY *(connected to node* $\bot$*) and* CONTINENT *(connected to node* $\top$*). The arc labels of attribute graph have the following semantics. First, since* $\mathsf{R}(STATE, COUNTRY) = \mathbf{1}$*, for each non-null value of attribute* STATE*, we can determine a unique value of attribute* COUNTRY*. Second,* $\mathsf{R}(COUNTRY, CONTINENT) = \mathbf{f}$ *states that the attribute* COUNTRY *literally determines the attribute* CONTINENT*. Third, the same value of* CITY *can have multiple values for* STATE *(*$\mathsf{R}(CITY, STATE) = +$*) or multiple values for* COUNTRY *(*$\mathsf{R}(CITY, COUNTRY) = +$*) and no value for* STATE*. Finally, there exists a single continent without countries, states and cities, which is represented by the arc* $\mathsf{R}(\bot, CONTINENT)$ *with label* $\mathbf{f}$*.*

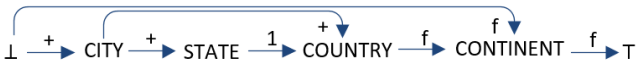


Figure 4: Attribute graph for dimension *REGION*

Proposition 1. *Let* $\mathcal{D} = (\mathrm{S}, R, \lambda_R, \bot, \top)$ *be an attribute graph. Then, the subset* $\mathrm{X}_D \subseteq \mathrm{S}$ *of all attributes in* S *with at least one* + *labeled in-edge and no* **f** *labeled in-edge is a dimension identifier for all valid dimension tables with attributes* S.

The proofs of the previous and the following propositions can be found in the Appendix A.

Attribute graphs concisely describe well-know properties of hierarchies, as described for instance in [1]. Edges $\mathsf{R}(\bot, \mathrm{A})$ can model "unbalanced" hierarchies, edges $\mathsf{R}(\mathrm{A}_i, \mathrm{A}_j)$ such that $\mathrm{A}_i \not\leftarrow \mathrm{A}_j$ in $\mathcal{A}$ model "non-covering hierarchies", and edges $\mathsf{R}(\mathrm{A}_j, \mathrm{A}_i)$ with label **1** represent "non-strict" hierarchies.

Example 5. *By Proposition 1, the dimension identifier for dimension REGION is* (*CITY*, *STATE*, *COUNTRY*). *Figure 5 shows an attribute graph for the schema of a dimension table WAREHOUSE. It expresses that attribute WH_ID literally determines CITY and STATE but not COUNTRY. By Proposition 1, the identifier of WAREHOUSE is* (*WH_ID*, *COUNTRY*).

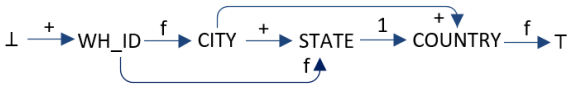


Figure 5: Attribute graph for dimension *WAREHOUSE*

2.2. Fact tables

Fact tables associate measures with dimension values. A *fact table over a set of dimensions* $D_1, \cdots, D_n$ is a table T whose schema S contains a non-empty subset X_i of dimension attributes from each dimension D_i, and a non-empty set of attributes Z representing one or more *measures* where $\mathrm{X}_1 \cup \cdots \cup \mathrm{X}_n \mapsto \mathrm{S}$. The active domain of each dimension attribute $T.\mathrm{A} \in \mathrm{X}_i$ is a subset of the active domain of $D_i.\mathrm{A}$. Each measure is represented by a group of attributes where one attribute has the role of *Value* and other attributes have the role *Detail*. *Value* attributes carry the actual measure values while *Detail* attributes provide auxiliary information on the measure. In particular, each measure attribute can be associated with a *Unit* or *Currency* detail attribute to control the application of aggregate functions. For simplicity, the term *measure attribute* will refer in the following to a *Value* attribute.

Definition 5 (Fact identifier)**.** *Let* $\mathrm{K} \subseteq \mathrm{S}$ *be a set of dimension attributes in the schema of a fact table* $T(\mathrm{S})$ *and* $\mathcal{L}$ *be a set of LFDs defined on* S. *Then,* K *is a* fact identifier *of* S *if* $\mathrm{K} \mapsto \mathrm{S}$ *holds for any instance* T *over* S *satisfying all LFDs in* $\mathcal{L}$.

Each attribute graph of some dimension represents a set of LFDs and can be used to compute fact identifiers for some schema S as stated in the following proposition.

Proposition 2. *Let* $T(\mathrm{S})$ *be a fact table defined over a set of dimensions* $D_1, \cdots, D_n$ *and* $\mathrm{K}_1, \cdots, \mathrm{K}_n$ *be the dimension identifiers of* $D_1 \cap \mathrm{S}, \cdots, D_n \cap \mathrm{S}$ *respectively. Then* $\mathrm{K} = \mathrm{K}_1 \cup \cdots \cup \mathrm{K}_n$ *is a fact identifier of* T, *and* K *is a minimal identifier if all dimensions in* T *are mutually independent.*

Example 6. *Consider fact tables* SALES *and* INVENTORY *below.* SALES *contains the sales of products, is defined over dimensions STORE, TIME, and PROD.* INVENTORY *is defined over dimensions WAREHOUSE, TIME, TAX and PROD. Measure attributes are in italics. Figure 6 shows the attribute graphs for dimensions STORE, PROD and TAX with identifiers STORE_ID, PROD_SKU and TAX_NO respectively.*

SALES(*PROD_SKU*, *BRAND*, *MONTH*, *YEAR*, *CITY*, *STATE*, *COUNTRY*, *AMOUNT*, *CURRENCY*)

INVENTORY(*PROD_SKU*, *BRAND*, *YEAR*, *WH_ID*, *CITY*, *COUNTRY*, *TAX_NO*, *RATE*, *TAX_DESC*, *QTY_ON_HAND*)

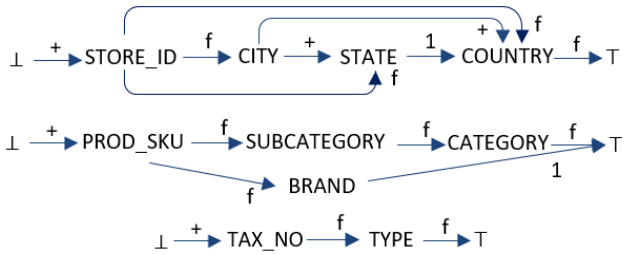


Figure 6: Attribute graphs of dimensions *STORE*, *PROD* and *TAX*

The fact identifiers of fact tables SALES *and* INVENTORY *are defined by the union of the corresponding dimension identifier attributes. If all dimensions are mutually independent, they form a minimal set.*

2.3. Aggregable attributes

In analytic tables, attributes are *aggregable*, but an attribute does not necessarily aggregate with any aggregate function along any dimension, as shown by extensive previous work in statistical and OLAP databases [13].

Example 7. *In the schema of fact table* SALES, *measure AMOUNT is aggregable using SUM along all dimension attributes. However, measure QTY_ON_HAND in fact table* INVENTORY *is not aggregable along the TIME dimension,* i.e. *summing the quantity of products QTY_ON_HAND over all months produces is considered senseless. So, a* SUM *over QTY_ON_HAND must at least be grouped by YEAR.*

Our model enables the designer of an analytic table to declare for each attribute the aggregate function that is applicable and the maximal set of dimension attributes along which this aggregation can be computed. Clearly, if some attribute A is aggregable along a set of dimension attributes Z, then it is also aggregable along any subsets

of Z. In the following we denote by $\mathbf{agg}_{\mathrm{A}}(\mathrm{F},\mathrm{Z})$ the *aggregable property* of A and state that property $\mathbf{agg}_{\mathrm{A}}(\mathrm{F},\mathrm{Z})$ holds in T if Z is the maximal set of attributes along which A is aggregable using F in T. More precisely, we give the following definition for aggregable properties $\mathbf{agg}_{\mathrm{A}}(\mathrm{F},\mathrm{Z})$:

Definition 6. *Let* X *be the set of dimension attributes in an analytic table T over schema* S*, A be an aggregable attribute in* S *and* F *be an aggregate function on A. Aggregable property* $\mathbf{agg}_A(\mathrm{F},\mathrm{Z})$ *holds in T if*

1. *all aggregations along any subsets of* Z *are considered as meaningful by the user;*
2. Z *only contains dimension attributes which determine A,* i.e. *for all dimension attributes* $B \in \mathrm{Z} \cap \mathrm{X}$ *there exists a minimal subset of dimension attributes* $\mathrm{U} \subseteq \mathrm{X}$ *such that* $\mathrm{U} \mapsto A$ *and* $B \in \mathrm{U}$*;*
3. F *is applicable to A.*

Example 8. *Considering Example 7, attribute QTY_ON_HAND depends on all dimension attributes and can be aggregated along these attributes except attribute YEAR of dimension TIME which is considered meaningless to the user (item 1 in Definition 6). We then state that* $\mathbf{agg}_{QTY\_ON\_HAND}(\mathrm{SUM}, Z)$ *holds in* INVENTORY *where* Z *contains all dimension attributes of* INVENTORY *except YEAR.*

Aggregable properties generalize the approach of [14] and [2], which only focused on SUM aggregations and categorized measures as being additive, fully-additive, non-additive, or semi-additive.

For determining valid aggregate function for new attributes, we provide a default categorization of attributes. Two categories, **NUM** and **DESC**, are obtained from the (SQL) data type of attributes. A third category **STAT** captures numerical values resulting from the use of some aggregate functions. Table 4 describes the five common SQL aggregate functions applicable to each category.

Table 4: Categories of aggregable attributes

Category	Properties
NUM	• Numerical values • Applicable functions: SUM, AVG, COUNT, MIN, MAX
DESC	• Descriptive or categorical values • Applicable functions: COUNT
STAT	• Numerical statistical values • Applicable functions: COUNT, MIN, MAX

Example 9. *Attribute STORE_ID of data type string in dimension table STORE is of category* **DESC** *and can only be counted. Attributes AMOUNT and QTY_ON_HAND in tables* SALES *and* INVENTORY *respectively are of category* **NUM***. Thus, by default, both attributes can be summed up unless a user-defined aggregable property is specified, as in Example 8 for attribute QTY_ON_HAND.*

If A is an attribute of T for which no aggregable property was defined by a user, we first use the default category of A to define which aggregate function F is applicable to A and then state that a default aggregable property $\mathbf{agg}_{\mathrm{A}}(\mathrm{F},\mathrm{Z})$ holds in T where Z is the set of all dimension attributes of T (by default, A literally depends on all dimension attributes).

2.4. Summary of metadata acquisition

To conclude this section, we summarize the data and related metadata introduced by our data model in Table 5 and indicate how they are obtained. We distinguish between data and metadata that can be *user-defined*, *automatically computed* from other data or metadata and derived by *default rules*. Most human effort is required on the definition of dimension and fact tables (analytic schemas) with a careful description of their attribute graphs. We shall see in Section 6.1 that attribute graphs can also be automatically computed from dimension tables or samples thereof. This essentially reduces the human-effort for enabling schema augmentations and complements to defining analytic schemas and aggregable properties.

Table 5: Summary of data model concepts

Metadata	Source
Dimension and fact tables	User-defined
Attribute graphs	User-defined or computed
Dimension identifiers	Computed
Fact identifiers	Computed
Aggregable properties	User-defined or default rule

3. Schema Augmentation and Complement

In this section we first describe the relationships that can be automatically extracted from the definitions of dimension and fact tables and formally define the notion of schema augmentation between tables connected by such relationships. We then introduce natural and reduction-based schema complements as a special case of schema augmentation where the tuples of a table are augmented with new attribute values obtained from another table *without generating new tuples in the result.*

3.1. Relationships

We distinguish between join relationships and attribute-mapping relationships. Both kinds of relationships link semantically equivalent attributes and can be extracted from the view definitions of analytic tables. New join relationships between tables that are not related through views can also be added by the business user if necessary.

Definition 7 (Join relationship)**.** *A* join relationship $\mathsf{R}(T_1, T_2)$ *between two tables T_1 and T_2 is defined by a non-empty set of equality atoms $\{P_1, \cdots, P_k\}$ where each P_i is*

of the form $T_1.A = T_2.B$. It relates all tuples of T_1 and T_2 satisfying the join predicate $P_1 \wedge \cdots \wedge P_k$, i.e.*, the set of all related tuples is defined by $T_1 \bowtie_{P_1 \wedge \cdots \wedge P_k} T_2$.*

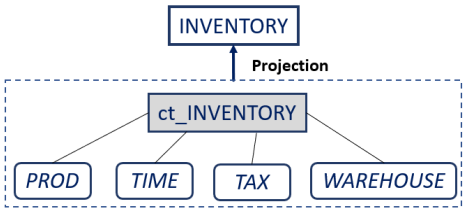


Figure 7: The construction of INVENTORY

Example 10. *Fact table* INVENTORY *of Example 6 is a view over dimensions PROD, TIME, TAX and WAREHOUSE, the view is defined by a star join between a non-analytical table* ct_INVENTORY *and these dimensions as shown in Figure 7. The analysis of the view definition yields four join relationships between* ct_INVENTORY *and the dimension tables.*

Join relationships might also exist in the database schema between non-analytic tables (*i.e.*, tables that are neither dimension nor fact tables) through foreign key constraints which can be easily extracted from the schema definition.

Definition 8 (Attribute mapping relationship)**.** *Let T_1 and T_2 be two tables such that T_1 is derived from T_2 using a query $\mathcal{Q}$ and auxiliary tables: $T_1 = \mathcal{Q}(T_2, T_i, \cdots, T_n)$ with $n \geq 0$. Query $\mathcal{Q}$ defines an* attribute mapping *from $T_2.B$ to $T_1.A$, denoted by $T_2.B \twoheadrightarrow T_1.A$, if for all possible values $x \in dom(A)$ and all possible instances of $T_2, T_i, \cdots, T_n$:*

$$\sigma_{A=x}(\mathcal{Q}(T_2, T_i, \cdots, T_n)) = \sigma_{A=x}(\mathcal{Q}(\sigma_{B=x}(T_2), T_i, \cdots, T_n))$$

An attribute mapping relationship $\mathsf{R}(T_1, T_2)$ *between T_1 and T_2 is defined by a non-empty set of attribute mappings from T_1 to T_2.*

Example 11. *Dimension PROD from Example 6 is defined as a view over three non-analytic tables, where* X *is the set of attributes in PROD, and P_1, P_2 are join predicates: PROD* $= \pi_{\mathrm{X}}$(PRODUCT $\bowtie_{P_1}$ SUBCATEGORY $\bowtie_{P_2}$ CATEGORY)*. Predicate P_1 defines a join relationship between* PRODUCT *and* SUBCATEGORY *on SUBCATEGORY_ID whereas predicate P_2 defines a join relationship between* SUBCATEGORY *and* CATEGORY *on CATEGORY_ID. An attribute mapping relationship between PROD and each non-analytic table providing an attribute in* X *is also extracted from the view definition.*

Example 12. *In the view definition of Example 10, since* INVENTORY *is a projection of the star join result, there exists an attribute mapping relationship between table* INVENTORY *and the other five tables.*

Each relationship $\mathsf{R}(T_1, T_2)$ between two tables $T_1(\mathrm{S}_1)$ and $T_2(\mathrm{S}_2)$ defines a partial mapping μ_{R} from attributes A of T_1 to attributes B of T_2 where $\mu_{\mathsf{R}}(T_1.\mathrm{A}) = T_2.\mathrm{B}$ if (i) R is a join relationship containing an atom $T_1.\mathrm{A} = T_2.\mathrm{B}$ or (ii) R contains an attribute mapping $T_1.\mathrm{A} \twoheadrightarrow T_2.\mathrm{B}$ or $T_2.\mathrm{B} \twoheadrightarrow T_1.\mathrm{A}$. Relationship R is *well-formed* if μ_{R} is a *one-to-one mapping*. For the sake of simplicity, unless specified differently, we assume in the following that all relationships R define a *natural* mapping $\mu_{\mathsf{R}}(T_1.\mathrm{A}) = T_2.\mathrm{A}$ for some attributes A of $\mathrm{S}_1 \cap \mathrm{S}_2$ shared by table T_1 and T_2 and the set of attributes A is called the *common attributes* of T_1 and T_2.

Example 13. *In Example 11, the common attributes for join relationships* R(CATEGORY, SUBCATEGORY) *and* R(PRODUCT, SUBCATEGORY) *are respectively* {*CATEGORY_ID*} *and* {*SUBCATEGORY_ID*}*. Similarly, the attribute mapping relationship* R(*PROD*, SUBCATEGORY) *between dimension table PROD and non-analytical table* SUBCATEGORY *has common attributes* {*PROD_SKU*, *BRAND*}*.*

Consider now a fact table ORDER *recording detail descriptions for orders from different stores and customers.* ORDER *has two attributes ORDER_DAY, SHIP_DAY, each one referring to attribute DAY of dimension TIME. A single attribute mapping relationship* R(ORDER, *TIME*) *mapping the two* ORDER *attributes to the same attribute TIME.DAY,* μ_{R}(ORDER.*ORDER_DAY*) = *TIME.DAY,* μ_{R}(ORDER.*SHIP_DAY*) = *TIME.DAY, is* not a one-to-one mapping *and therefore* not well-formed. *Thus, two distinct relationships* R_1(ORDER, *TIME*) *and* R_2(ORDER, *TIME*) *are needed, one for ORDER_DAY and the other for SHIP_DAY.*

Figure 8 shows a graph of relationships issued from previous examples. Dimension tables are represented by rounded rectangles, fact tables by bold square rectangles, and non-analytic tables by square rectangles. Each node contains its identifier (ID), and edges indicate relationships labelled with their common attributes. Edges with solid lines are extracted relationships while edges with bold dashed lines are user-defined relationships. For instance, a join relationship is explicitly defined by a designer between dimensions *STORE* and *WAREHOUSE*, using equality atoms on attributes CITY, STATE and COUNTRY.

3.2. Schema augmentation

We first introduce the notion of schema augmentation using well-formed relationships and the notion of merge query.

Definition 9 (Schema augmentation)**.** *Let $T_0(\mathrm{S}_0)$ and $T(\mathrm{S})$ be two tables related by a relationship* R *with a set of common attributes* $\mathrm{Y} = \mathrm{S}_0 \cap \mathrm{S}$*. Then table T is a* schema augmentation *to T_0 with respect to* R *if* $\mathrm{Y} \subset \mathrm{S}$*.*

Observe that, by definition, common attributes Y is not empty and T must bring new attributes to T_0.

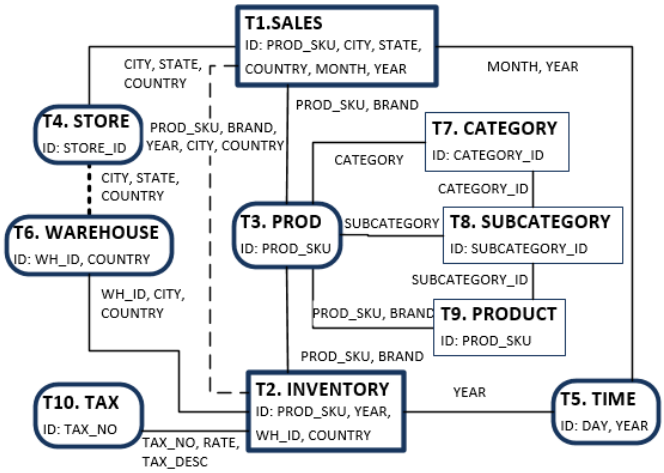


Figure 8: Examples of relationships

Definition 10 (Merge query)**.** *Let* $T_0(\mathrm{S}_0)$ *and* $T(\mathrm{S})$ *be two tables related by a relationship* R *with a set of common attributes* Y*. Then the* merge *of* T_0 *and* T *is a* left-outer join query $Q = \Pi_{\mathrm{X}}(T_0 \rtimes_{P_1 \wedge \cdots \wedge P_k} T)$*, where* $\mathrm{K} = |\mathrm{Y}|$ *is the number of common attributes of* R*,* Π *is a duplicate elimination projection.* Q *is defined as follows:*

1. *If all common attributes are dimension attributes, and there exists a pair of common attributes* $A_1, A_2 \in \mathrm{Y}$ *in* T_0 *such that* $T_0.A_1 \preccurlyeq T_0.A_2 \wedge T.A_1 \not\preccurlyeq T.A_2$*, then* $\mathrm{X} = \mathrm{S}_0 \uplus \mathrm{S}$ *else* $\mathrm{X} = \mathrm{S}_0 \cup \mathrm{S}$*.*

2. *For each* $A_i \in \mathrm{Y}$*,* $\exists P_i \in P$ *such that* P_i *is of the form:* $(T_0.A_i = T.A_i)$ *OR* $(T_0.A_i = null$ *AND* $T.A_i = null)$*.*

In the following, we will abbreviate $Q = \Pi_{\mathrm{X}}(T_0 \rtimes_{P_1 \wedge \cdots \wedge P_k} T)$ *to* $Q = T_0 \rtimes_{\mathrm{Y}} T$*.*

Item 1 checks if the attribute hierarchy is preserved by relationship R and keeps the common attributes separately (disjoint union $\uplus$) for each table if this is not the case. Item 2 manages the SQL semantics of joins in presence of nulls by adapting the join predicates.

Example 14. *Table* SALES *is a schema augmentation to* INVENTORY *with respect to common attributes:* $\mathrm{Y} = \{$*PROD_SKU, BRAND, YEAR, CITY, COUNTRY*$\}$ *and adds MONTH, STATE and AMOUNT to* INVENTORY*. The merge query is* INVENTORY $\rtimes_{\mathrm{Y}}$ SALES *and the schema of the resulting merge does not duplicate the common attributes.*

After merging a table with a schema augmentation, the identifier of the new augmented table might change. The following proposition describes how to compute the identifiers of augmented tables.

Proposition 3. *Let* $T_0'(\mathrm{S}_0')$ *be a merge of table* $T_0(\mathrm{S}_0)$ *with a target schema augmentation* $T(\mathrm{S})$*. Let* K *be the identifier of* T_0 *and* $\mathrm{S}_{new} \subseteq \mathrm{S}_0' - \mathrm{S}_0$ *be the set of dimension attributes added to* T_0 *in the merge. Then, for all minimal subsets* $\mathrm{K}_{new} \subseteq \mathrm{S}_{new}$ *where* $\mathrm{K}_{new} \mapsto \mathrm{S}_{new}$*,* $\mathrm{K} \cup \mathrm{K}_{new}$ *is an identifier of* T_0'*.*

Example 15. *Consider the merged result* INVENTORY$'$ *in Example 14. Then,* $\mathrm{K} = \{$*PROD_SKU, YEAR, WH_ID, COUNTRY*$\}$ *and* $\mathrm{S}_{new} = \{$*MONTH, STATE*$\}$*. The set* S_{new} *is also the minimal subset where* $\mathrm{S}_{new} \mapsto \mathrm{S}_{new}$*. Thus, the identifier of* INVENTORY$'$ *is* $\mathrm{K} \cup \mathrm{S}_{new} = \{$*PROD_SKU, YEAR, WH_ID, COUNTRY, MONTH, STATE*$\}$*.*

Schema augmentations define no restriction on the common attributes in underlying table relationship R. The consequence of this unrestricted setting is that many augmentations lead to tuple duplication and unexpected results. For example, in the merge results of Example 14, tuples in INVENTORY will be duplicated because of the newly added dimension attributes MONTH and STATE. Then an aggregation with functions SUM or AVG of QTY_ON_HAND could easily produce wrong results. To better control the schema augmentation process, we introduce new restrictions that can be applied for filtering and transforming schema augmentations before merging them with the source tables.

3.3. Natural schema complement

We extend the original definition of natural schema complement [8] by considering LFD and relationships including attribute mappings.

Definition 11 (Natural schema complement)**.** *Let* $T(\mathrm{S})$ *be a schema augmentation of* $T_0(\mathrm{S}_0)$ *with a set of common attributes* $\mathrm{Y} = \mathrm{S}_0 \cap \mathrm{S}$*. Table* T *is a* natural schema complement *to a table* T_0 *with respect to* R *if* $\mathrm{Y} \mapsto \mathrm{S}$*.*

Condition $\mathrm{Y} \mapsto \mathrm{S}$ guarantees that by joining T_0 and T on their common attributes Y, each tuple of T_0 will match at most one tuple of T. The merge of T_0 and T through natural schema complement follows Definition 10 for merging schema augmentations and, by Definition 11 and Proposition 3, the identifier of the merge result is equal to the identifier of the source table T_0. We refer to the merge with a schema augmentation and natural schema complement by *augmented merge* and *natural merge* respectively.

Example 16. *The non-analytical table* PRODUCT*(PROD_SKU, BRAND, ...) is a natural schema complement to dimension table PROD since the common attribute PROD_SKU is a primary key in* PRODUCT*. This constraint guarantees that the corresponding merge query PROD* $\rtimes_{PROD\_SKU}$ PRODUCT *generates exactly one tuple for each tuple in PROD. The schema of the natural merge only has one occurrence of the common attribute PROD_SKU (natural outer join).*

3.4. Reduction queries

When the common dimension attributes of T_0 and T do not literally determine all attributes of T, a natural schema complement can still be obtained by transforming T before applying the outer join merge. These transformations can be obtained by queries which reduce each partition of T identified by the shared attributes to a single tuple. We refer to those queries as *reduction queries*.

Definition 12 (Reduction query)**.** *Let* $T(\mathrm{S})$ *be a table with identifier* K*. A query* $Q(T)$ *producing the table* $T'(\mathrm{S}')$ *is called a* reduction *of* T *on a subset of attributes* $\mathrm{K}' \subseteq \mathrm{S}'$ *if* $\mathrm{K}' \subset \mathrm{K}$ *is a proper subset of identifier* K *and* $\mathrm{K}' \mapsto \mathrm{S}'$ *holds in* T' *(*K' *is an identifier of* T'*). The attributes in* $\mathrm{K} - \mathrm{K}'$ *are called the* reduced attributes*.*

Example 17. *Consider the fact table* SALES *with identifier* K = {*PROD_SKU, MONTH, YEAR, CITY, STATE, COUNTRY*}*. A query* SALES′ = Q(SALES) *that filters* *COUNTRY = 'USA' and projects on all other attributes is a reduction of* SALES *on attributes* $\mathrm{K}' = \mathrm{K} - \{$*COUNTRY*$\}$*.* K' *is the key of* SALES′ *and COUNTRY is the reduced attribute.*

Each reduction query $Q(T)$ of table T with key K on attributes $\mathrm{K}' \subset \mathrm{K}$ reduces each partition of T with the same values for attributes K' and different values for the reduced attributes $\mathrm{K} - \mathrm{K}'$ into a single tuple (K' is the key of the result).

For a given table and set of attributes, there might exist a great variety of reduction queries. In our work, we focus on three types of reduction queries: aggregate, filter and pivot.

Aggregate reduction. Let $T(\mathrm{S})$ be an analytic table, A be an aggregable attribute in S, and F be an aggregate function such that aggregable property $\mathbf{agg}_{\mathrm{A}}(\mathrm{F}, \mathrm{Z})$ holds in T. We denote by $Q(T) = \mathit{Agg}_T(\mathrm{F}(\mathrm{A}) \mid \mathrm{X})$, $\mathrm{S} - \mathrm{Z} \subseteq \mathrm{X} \subseteq \mathrm{S}$, an aggregate query on table T that aggregates A using aggregate function F with group-by attributes X. When $\mathrm{K}' \subset \mathrm{K}$, where K is the identifier of T, we call query $Q(T) = \mathit{Agg}_T(\mathrm{F}(\mathrm{A}) \mid \mathrm{K}')$ an *aggregate reduction* of T on attributes K', and $\mathrm{K} - \mathrm{K}'$ are the reduced attributes. Note that the SQL group-by operator implements literal equality semantics for *null* values (*null* values are not distinct).

Filter reduction. Let $T(\mathrm{S})$ be a table, we denote by $Q(T) = \mathit{Filter}_T(P \mid \mathrm{X})$, $P = \{P_1 \wedge \cdots \wedge P_i\}$, a filter query that filters T by a set of predicates P on attributes X of T. When $\mathrm{K}' \subset \mathrm{K}$, where K is the identifier of table T, and for each attribute $\mathrm{A} \in \mathrm{K} - \mathrm{K}'$, there exists a predicate $P_i \in P$ of the form $\mathrm{A} = v_i, v_i \in dom(\mathrm{A})$, we call query $Q(T) = \mathit{Filter}_T(P \mid \mathrm{K} - \mathrm{K}')$ a *filter reduction* of T on attributes K', and $\mathrm{K} - \mathrm{K}'$ are the reduced attributes.

Pivot reduction. Let $T(\mathrm{S})$ be a table, and A be an attribute in S. We denote by $Q(T) = \mathit{Pivot}_T(\mathrm{A} \mid \mathrm{X})$, $\mathrm{X} \subset \mathrm{S}$, a pivot query with a set of attributes X whose values are transformed into columns, and the values of attribute A are pivoted into the new columns. The query replaces attributes A and X by converting every distinct tuple $t \in \pi_{\mathrm{X}}(T)$ in the projection of T to a *new attribute* $T.t$ storing the corresponding values of A. When $\mathrm{K}' \subset \mathrm{K}$, where K is the identifier of T, we call query $\mathit{Pivot}_T(\mathrm{A} \mid \mathrm{K} - \mathrm{K}')$ a *pivot reduction* of T on attributes K', and $\mathrm{K} - \mathrm{K}'$ are the reduced attributes.

The above definitions of aggregate and pivot reductions can be easily generalized by replacing attribute A with a set of attributes.

Example 18. *Consider a table* $T(\mathrm{S})$ *in Table 6a with two attributes* A_1, A_2 *forming an identifier of* T*. The result of a pivot reduction query of* T *on attribute* A_1*, as* $\mathit{Pivot}_T(M \mid \{A_2\})$*, is shown in Table 6b. Domain values of* A_2 *are pivoted into two new attributes* M_b_1, M_b_2 *storing the value of* M*.*

Table 6: Example of pivot reduction

(a) Table T

T	A_1	A_2	M
t_1	a_1	b_1	x_1
t_2	a_2	b_1	x_2
t_3	a_1	b_2	x_3

(b) Table $\mathit{Pivot}_T(\mathrm{M} \mid \{\mathrm{A}_2\})$

T'	A_1	M_b_1	M_b_2
	a_1	x_1	x_3
	a_2	x_2	-

Proposition 4. *Let* $T_0(\mathrm{S}_0)$ *and* $T(\mathrm{S})$ *be two tables such that* T *is a schema augmentation to* T_0 *with common attributes* Y*. Let* K *be an identifier of* T*. The result of a reduction query* $Q(T)$ *on attributes* $\mathrm{K}' = \mathrm{K} \cap \mathrm{Y}$*, is a natural schema complement to* T_0*.*

Example 19. *Fact tables* SALES *and* INVENTORY *in the relationship graph of Figure 8 have common attributes* Y = {*PROD_SKU, BRAND, YEAR, CITY, COUNTRY*}*.* INVENTORY *is a schema augmentation to* SALES *since* Y *is not a fact identifier of* INVENTORY *(it misses* {*WH_ID, TAX_NO*}*). We can then define several reduction queries which transform* INVENTORY *into a natural schema complement of SALES. A first reduction query is* Q(INVENTORY) = $\mathit{Agg}_{\mathrm{INVENTORY}}$(AVG(*QTY_ON_HAND* | X), X = {*PROD_SKU, YEAR, COUNTRY*} *which aggregates measure attribute QTY_ON_HAND on* X*. It produces a natural schema complement to* SALES*. Similarly, a pivot query* $\mathit{Pivot}_{\mathrm{INVENTORY}}$(*QTY_ON_HAND* | {*WH_ID, TAX_NO*}) *or a filter query* $\mathit{Filter}_{\mathrm{INVENTORY}}$(*WH_ID = '1234' ∧ TAX_NO = 'abcd'*) *reduces attributes WH_ID, TAX_NO and transforms* INVENTORY *into a natural schema complement of* SALES*. Finally, a reduction query* Q(SALES) *that first applies a filter* $\mathrm{SALES}_{FIL} = \mathit{Filter}_{\mathrm{SALES}}(\{STATE =' Ohio'\})$ *followed by a pivot query* $\mathit{Pivot}_{\mathrm{SALES}_{FIL}}(\{AMOUNT\} \mid \{MONTH\})$ *reduces attributes MONTH and STATE and produces a natural schema complement to* INVENTORY*.*

3.5. Derived relationships

We derive new relationships by composition and fusion of relationships which respectively perform the intersection and union of common attributes. Derived relationships offer more schema augmentation opportunities, especially between fact tables. For example, in Figure 8, there is no direct relationship between fact tables INVENTORY and SALES (edge with plain line). However, both tables share dimension tables *PROD* and *TIME* which can be used to derive a new relationship shown by the edge with a dashed line.

Proposition 5. Composition of relationships. *Let* $\mathsf{R}_1(T_1, T_2)$ *and* $\mathsf{R}_2(T_2, T_3)$ *be two well-formed relationships between tables* T_1, T_2 *and* T_3 *with respective common attributes* Y_1 *and* Y_2. *If* $\mathrm{Y}_3 = \mathrm{Y}_1 \cap \mathrm{Y}_2 \neq \emptyset$, *then there exists a well-formed relationship* $\mathsf{R}_3(T_1, T_3)$ *that is a* composition *of* $\mathsf{R}_1(T_1, T_2)$ *and* $\mathsf{R}_2(T_2, T_3)$ *with common attributes* Y_3.

Proposition 6. Fusion of relationships. *Let* $\mathsf{R}_1(T_1, T_2)$ *and* $\mathsf{R}_2(T_1, T_2)$ *be two well-formed relationships between two tables* T_1 *and* T_2 *with respective common attributes* Y_1 *and* Y_2. *If* $\forall A \in \mathrm{Y}_1 \cap \mathrm{Y}_2$, $\mu_{\mathsf{R}_1}(A) = \mu_{\mathsf{R}_2}(A)$ *then there exists a well-formed relationship* $\mathsf{R}_3(T_1, T_2)$ *that is a* fusion *of* $\mathsf{R}_1(T_1, T_2)$ *and* $\mathsf{R}_2(T_1, T_2)$ *with common attributes* $\mathrm{Y}_3 = \mathrm{Y}_1 \cup \mathrm{Y}_2$.

Given a set of well-formed relationships $\mathsf{R}_0(T_0, T), \cdots, \mathsf{R}_n(T_0, T), n \geq 0$ between two tables T_0, T with respective common attributes $\mathrm{Y}_0, \cdots, \mathrm{Y}_n$. The *common attributes of* T_0 *and* T is the union of $\mathrm{Y}_0, \cdots, \mathrm{Y}_n$.

Example 20. *In Figure 8, relationships* $\mathsf{R}(SALES, PROD)$ *and* $\mathsf{R}(PROD, INVENTORY)$ *are composed to yield* $\mathsf{R}(SALES, INVENTORY)$ *with common attributes* {*PROD_SKU*, *BRAND*}. *Similarly,* $\mathsf{R}(SALES, TIME)$ *and* $\mathsf{R}(TIME, INVENTORY)$ *are composed to generate* $\mathsf{R}(SALES, INVENTORY)$ *with common attribute YEAR. Finally,* $\mathsf{R}(SALES, STORE)$,$\mathsf{R}(STORE, WAREHOUSE)$ *and* $\mathsf{R}(WAREHOUSE, INVENTORY)$ *are composed to produce* $\mathsf{R}(SALES, INVENTORY)$ *with common attributes* {*CITY*, *COUNTRY*}. *Then the fusion of* $\mathsf{R}(SALES, INVENTORY)$, $\mathsf{R}(SALES, INVENTORY)$ *and* $\mathsf{R}(SALES, INVENTORY)$ *yields* $\mathsf{R}'(SALES, INVENTORY)$ *with common attributes* {*YEAR*, *PROD_SKU*, *BRAND*, *CITY*, *COUNTRY*}. *However, the fusion of the two relationships* $\mathsf{R}_1(ORDER, TIME)$ *and* $\mathsf{R}_2(ORDER, TIME)$ *in Example 13 is not possible because* $\mu_{\mathsf{R}_1}(DAY) \neq \mu_{\mathsf{R}_2}(DAY)$ *(both relationships map two different attributes ORDER.ORDER_DAY and ORDER.SHIP_DAY to the same attribute TIME.DAY.*

Consider a table $T'_0(\mathrm{S}'_0)$ which is the result of merging table $T_0(\mathrm{S}_0)$ with a target schema augmentation $T(\mathrm{S})$ through a *path* of relationships reaching T from T_0. Then, each relationship $\mathsf{R}(T, T_i)$ (resp. $\mathsf{R}(T_0, T_i)$) with common attributes Y yields a relationship $\mathsf{R}(T'_0, T_i)$ if $\mathrm{S}'_0 \cap \mathrm{Y} \neq \emptyset$.

4. Quality Guarantees

We now describe the quality guarantees managed by our system. Section 4.1 evaluates the correctness of aggregated attribute values produced by reduction operations and merge queries. Sections 4.2 and 4.3 deal with the generation of *ambiguous* and *incomplete* attribute values during the construction of schema augmentations (and schema complements).

4.1. Propagation of aggregable properties

When the schema of a table T_0 is augmented with a new attribute A after a merge with a table T, stating the aggregable property of A that holds in the augmented table is critical to avoid incorrect aggregations on that augmented table. This raises two problems:

- The first problem is to define the *aggregable properties* of all new attributes which are computed by some reduction query. These properties include the identification of the applicable aggregate functions and the set of dimension attributes along which each new attribute can be aggregated in the query result (before merging).
- The second problem is to define the aggregable properties of the new attributes in the augmented table obtained after merging the source table with the reduction query result.

To address the first problem we determine which aggregate functions are applicable to A in the result $T = \mathcal{Q}(T')$ of a reduction query $\mathcal{Q}$ over T'. This falls into one of the following cases:

1. A is an attribute of T', filtered or not. Then if F is applicable to A in T', it is also applicable to A in T.
2. $\mathrm{A} = F(\mathrm{A}')$ for some A' in T', i.e., A' has been aggregated. Then if F is a *distributive* aggregate function (see Definition 13 below), F is also applicable to A in T. Other default functions applicable to A are determined by the co-domain categories of function F (**NUM**, **DESC** or **STAT**) using Table 4.
3. A holds pivoted values of an attribute A' of T'. Then, if F is applicable to A' in T', it is also applicable to A in T.

Definition 13 (Distributive aggregate function)**.** *Let* F *be an aggregate function applicable to a set of domain values* V. *If for any partition* $\mathrm{V}_1, \mathrm{V}_2$ *of* V, $\mathrm{F}(\mathrm{V}_1 \cup \mathrm{V}_2) = \mathrm{F}(\mathrm{F}(\mathrm{V}_1) \cup \mathrm{F}(\mathrm{V}_2))$ *then* F *is said to be a* distributive *aggregate function.*

Let us illustrate the previous case analysis. Suppose that an attribute A contains values that are aggregated from attribute A' using the average function AVG (case 2). First, we can easily see that AVG is not distributive, *i.e.* cannot be applied to A. Then, according to Table 4, statistical functions like AVG and STDEV have the domain category **NUM**, the co-domain category **STAT** and the only functions that can be applied are COUNT, MIN and MAX. We can conclude that the default aggregate functions that are applicable on A are COUNT, MIN and MAX.

Now, if $\mathbf{agg}_{\mathrm{A}'}(\mathrm{F}, \mathrm{Z}')$ holds in T' and F is applicable to A in the reduction query result $T = Q(T')$, the aggregable property $\mathbf{agg}_{\mathrm{A}}(\mathrm{F}, \mathrm{Z})$ holds in T for some subset $\mathrm{Z} \subseteq \mathrm{Z}'$ of attributes that are not reduced by the reduction query on

T'. For determining Z, we use Definition 6, which states that it is not possible to aggregate an attribute A along dimension attributes which do not determine A (item 2). The following example illustrates how Z can be validated using Definition 6.

Example 21. *Suppose that the fact table* PRICE_LIST*(PROD_SKU, YEAR, PRICE) is a natural schema complement to* SALES *with common attributes PROD_SKU and YEAR. Let* SALES′ *be the natural merge of* SALES *with* PRICE_LIST *adding a new attribute PRICE to* SALES*. PRICE is of category* **NUM** *and aggregable along some set of variables* X *using* SUM*,* i.e. $\mathbf{agg}_{PRICE}(\text{SUM}, \text{X})$*. Suppose that the user decides to define* $\text{X} = \{CITY, COUNTRY\}$*. Then, it is easy to see that an aggregation* $\mathcal{A}gg_{\text{SALES}'}(\text{SUM}(PRICE) \mid \{PROD\_SKU, YEAR, REGION\})$ *would wrongly sum up the PRICE values for all products sold in all cities of the same REGION. Using Definition 6 for aggregable properties, we can see that since attribute PRICE depends on attributes PROD_SKU and YEAR of table* PRICE_LIST*, these two attributes are the only dimension attributes that might appear in* X *and* $\mathbf{agg}_{PRICE}(\text{SUM}, \text{X})$ *does not hold in* SALES′ *for* $\text{X} = \{CITY, COUNTRY\}$*.*

We now consider the problem of determining the aggregable properties of the new attributes after merge. The following proposition states which aggregable properties hold for attribute A in the augmented table T'_0, knowing the aggregable properties of A that hold in the used schema augmentation T.

Proposition 7. *Let* $T'_0(\text{S}'_0)$ *be a merge of table* $T_0(\text{S}_0)$ *with a target schema augmentation* $T(\text{S})$*. Then the following aggregable properties hold for all aggregable attributes* $A \in \text{S}'_0$*:*

1. *If* $\mathbf{agg}_A(\text{F}, \text{V}_0)$ *holds in* T_0 *and* $A \in \text{S}'_0 \cap \text{S}_0$ *then* $\mathbf{agg}_A(\text{F}, \text{V}_0)$ *holds in* T'_0*.*
2. *If* $\mathbf{agg}_A(\text{F}, \text{V})$ *holds in* T *and* $A \in \text{S}'_0 - \text{S}_0$ *, then* $\mathbf{agg}_A(\text{F}, \text{V} \cap \text{S}'_0)$ *holds in* T'_0*.*

Example 22. *Suppose* SALES′ *is the complete natural merge of* SALES *and* INVENTORY *in Figure 8 reduced by the aggregate query* $\mathcal{A}gg_{\text{INVENTORY}}(\text{AVG}(A) \mid \text{X})$ *where* $A = QTY\_ON\_HAND$ *and* $\text{X} = \{PROD\_SKU, YEAR, COUNTRY\}$*. Function* AVG *is applied on A that transforms A from category* **NUM** *to attribute* $AVG(A)$ *of category* **STAT***. Then, only* COUNT, MIN *and* MAX *are applicable on* $AVG(A)$ *by default. Because* $\mathbf{agg}_A(\text{AVG}, \text{V})$ *holds in* INVENTORY *for* $\text{V} = \{PROD\_SKU, YEAR, WH\_ID, CITY, COUNTRY, TAX\_NO\}$*, then by Item 2 in Proposition 7,* $\mathbf{agg}_{\text{AVG}(A)}(\text{COUNT}, \text{V}')$ *holds in* SALES′ *where* $\text{V}' = \{YEAR, COUNTRY\}$ *(*V' *only contains dimension attributes which are necessary to determine attribute* $AVG(A)$*).*

4.2. Non-ambiguous aggregable attributes

As illustrated in Section 1, merge queries may produce attribute values that are ambiguous with respect to a non-strict dimension hierarchy. The next definition states that a schema augmentation $T(\text{S})$ *may* contain ambiguous attribute values with respect to a dimension D if S misses some attributes of D which are necessary to distinguish two tuples of T that are literally equal on their D attributes.

Definition 14 (Ambiguous analytic table)**.** *Let* $T(\text{S})$ *be an analytic table over a dimension* $D(\text{S}_D)$*. Let* $\text{X} = \text{S} \cap \text{S}_D$ *be the set of attributes in* S *from dimension* D *and* $\text{X}^* = \{A_j \in \text{S}_D \mid \exists A_i \in \text{X}, A_i \preccurlyeq^* A_j\}$ *be the ancestors of all attributes in* X *in the hierarchy type of* D*. Table* T *is said to be a* non-ambiguous *with respect to* D *if the literal functional dependency* $\text{X} \mapsto \text{X}^*$ *holds in* $T \bowtie_{\text{X}} D$ *and* ambiguous *otherwise.*

In the following, X^* is called the closure of X in D. We also say that a table T is *ambiguous* if it is ambiguous w.r.t. at least one dimension and *non-ambiguous* otherwise.

Example 23. *Fact table* SALES *contains attributes* $\text{X} = \{CITY, STATE, COUNTRY\}$ *from dimension STORE (Figure 8).* X *is closed w.r.t. the ancestor relationship,* i.e. $\text{X}^* = \text{X}$*. Then* $\text{X} \mapsto \text{X}^*$ *holds in table* SALES $\bowtie_{\text{X}}$ *STORE and* SALES *is not ambiguous with respect to STORE. Now suppose that* SALES′ *only contains two attributes* $\text{X}_1 = \{CITY, COUNTRY\}$ *from dimension STORE. Then* $\text{X}_1^* = \{CITY, STATE, COUNTRY\}$ *and* $\text{X}_1 \mapsto \text{X}_1^*$ *does not hold in* SALES′ $\bowtie_{\text{X}}$ *STORE (the city of Dublin exists in two different states of country USA). In this case,* SALES′ *is ambiguous with respect to dimension STORE.*

We can show that merge queries over two non-ambiguous tables generate non-ambiguous results. Consider two tables T_1 and T_2 that are non-ambiguous w.r.t. to some common dimension D, *i.e.* both literal functional dependencies $\text{X}_1 \mapsto \text{X}_1^*$ and $\text{X}_2 \mapsto \text{X}_2^*$ hold in D. Then, by composition of literal functional dependencies, $\text{X}_1 \cup \text{X}_2 \mapsto \text{X}_1^* \cup \text{X}_2^*$ holds in $T_1 \rtimes_P T_2$.

In the case of reduced schema augmentations, it may happen that a reduction query generates an ambiguous schema complement from a non-ambiguous input table (by removing some dimension attributes). The next proposition defines a sufficient condition for reduction queries to produce non-ambiguous schema augmentations.

Proposition 8. *Let* $T(\text{S})$ *be a non-ambiguous analytic table w.r.t. a dimension* D *of schema* S_D*. Let* $T'(\text{S}') = Q(T)$ *be a reduction of* T*. Let* $\text{X} = \text{S} \cap \text{S}_D$ *and* $\text{X}' = \text{S}' \cap \text{S}_D$*. If* $\text{X}' \mapsto \text{X}$ *in* D*, then* T' *is* non-ambiguous *w.r.t.* D*.*

From Proposition 8 we can directly conclude that any filter reduction over a non-ambiguous tables generates a non-ambiguous table (for filter reduction $\text{X}' = \text{X}$ and $\text{X}' \mapsto \text{X}$ holds in all dimensions D). However, for pivot and aggregate reduction queries we must check if $\text{X}' \mapsto \text{X}'^*$ holds in $T' \bowtie_{\text{X}'} D$ (Definition 14) to guarantee non-ambiguity ($\text{X}' \mapsto \text{X}$ holds in D is a sufficient but not necessary condition for non-ambiguity).

Example 24. *Fact table* INVENTORY *contains attributes* $\text{X} = \{WH\_ID, CITY, COUNTRY\}$ *from dimension*

WAREHOUSE. $\mathrm{X}^* = \{$*WH_ID, CITY, STATE, COUNTRY*$\}$ *is the schema of WAREHOUSE and* $\mathrm{X} \mapsto \mathrm{X}^*$ *holds in* INVENTORY $\bowtie_{\mathrm{X}}$ *WAREHOUSE. Therefore, table* INVENTORY *is not ambiguous with respect to WAREHOUSE. Now, let T be the result of an aggregate reduction as* $\mathcal{Agg}_{\mathrm{INVENTORY}}(\mathrm{AVG}(QTY\_ON\_HAND) \mid \mathrm{X}')$ *where* $\mathrm{X}' = \{$*CITY, COUNTRY*$\}$*. Using Proposition 8,* $\mathrm{X}' \mapsto \mathrm{X}$ *does not hold in D. Thus, T might be ambiguous with respect to WAREHOUSE. We then check whether the condition* $\mathrm{X}' \mapsto \mathrm{X}'^*$ *holds in* $T \bowtie_{\mathrm{X}'}$ *WAREHOUSE, where* $\mathrm{X}'^* = \{$*CITY, STATE, COUNTRY*$\}$*. Based on the attribute graph in Figure 5,* $\mathrm{X}' \mapsto \mathrm{X}'^*$ *does not hold, and T is ambiguous with respect to WAREHOUSE*

4.3. Complete merge results

A third quality problem we presented in the introduction concerns merge queries which might produce results which are incomplete with respect to the original schema augmentation table.

Example 25. *Consider the two analytic tables* T_0 *and T below related by a well-formed relationship* R *with common attributes* $A_1 \preccurlyeq A_2$ *from dimension* D_1*, and* $B_1 \preccurlyeq B_2$ *from dimension* D_2*. Identifiers of* T_0 *and T are* $\mathrm{K}_0 = \{A_1, A_2, B_1, B_2\}$ *and* $\mathrm{K} = \{A_1, A_2, A_3, B_1, B_2\}$ *respectively. Clearly, T is a schema augmentation to* T_0 *with respect to* R*. However, merging* T_0 *with T with a left outer join yields a table* T_0' *containing tuples* t_5 *and* t_6 *augmented by two new null valued attributes* A_3 *and* M *(there's no matching tuple in T). Then, by Proposition 7, if aggregable property* $\mathbf{agg}_M(F, \{A_1, A_3, B_1\})$ *holds in T, it also holds in* T_0'*. Then, some valid aggregate query* $\mathcal{Agg}(\mathrm{F}(M) \mid \{A_2, B_2\})$ *will generate a null value for partition* (b_1, e_1) *in the merge* T_0' *while the same query has value* $\mathrm{F}(\{y_1, y_4\})$ *for the same partition in T. This possibly undesirable situation occurs because* T_0' *is not a* complete merge *with respect to T.*

Table 7: Example of incomplete merge

T_0	A1	A2	B1	B2
t_5	a_1	b_1	d_1	e_1
t_6	a_2	b_3	d_3	e_3

T	A1	A2	A3	B1	B2	M
t_1	a_1	b_1	c_1	d_2	e_1	y_1
t_2	a_3	b_2	c_1	d_2	e_1	y_2
t_3	a_2	b_1	c_2	d_2	e_3	y_3
t_4	a_1	b_1	c_1	d_3	e_1	y_4

Definition 15 (Candidate completion tuples)**.** *Let* T_0' *be the merge of* T_0 *and T that are related by a relationship* R *with a set of common attributes* Y*. Let* $\mathrm{Y}^{top} \subseteq \mathrm{Y}$ *be the set of the "highest" attributes of* Y *in the corresponding attribute hierarchies. We can define the following two tables* $T^{ct} \subseteq T$ *and* $T^{cand} \subseteq T$*:*

- *the natural semi-join* $T^{ct} = T \ltimes_{\mathrm{Y}} T_0'$*, where* $\mathrm{Y} = \wedge_i(T.A_i = T_0'.A_i)$ *for each* $A_i \in \mathrm{Y}$*, is called the* completion table *of T with respect to* T_0'*.*
- *the natural semi-join* $T^{cand} = T \ltimes_{\mathrm{Y}^{top}} T_0'$*, where* $\mathrm{Y}^{top} = \wedge_i(T.A_i = T_0'.A_i)$ *for each* $A_i \in \mathrm{Y}^{top}$*, is called the* candidate completion table *of T with respect to* T_0'*.*

Example 26. *In Example 25, we have* $\mathrm{Y} = \{A_1, A_2, B_1, B_2\}$ *and* $\mathrm{Y}^{top} = \{A_2, B_2\}$*. Then completion table and candidate completion of T with respect to* T_0' *are* $T^{ct} = T \ltimes_{\mathrm{Y}} T_0' = \emptyset$ *and* $T^{cand} = T \ltimes_{\mathrm{Y}^{top}} T_0' = \{t_1, t_4\}$*.*

It is easy to see that $T^{ct} \subseteq T^{cand}$ for all schema augmentations tables T of some table T_0. The completeness of merge queries can now be defined by comparing the candidate completion table with the completion table.

Definition 16 (Complete merge)**.** *Let T be a schema augmentation to* T_0 *with respect to a relationship* R *having a set of common attributes* Y*. The merge result* T_0' *of* T_0 *and T is said to be a complete merge with respect to T if* $T^{ct} = T^{cand}$*.*

Example 27. *Continuing with Example 26, since* $T^{cand} \neq T^{ct}$*, we can conclude that* T_0' *is not a complete merge with respect to T.*

The following proposition states how it is possible to complete the result of an incomplete merge query.

Proposition 9. *Let* $T_0(\mathrm{S}_0)$ *and* $T(\mathrm{S})$ *be two analytic tables with a relationship* R *and common dimension attributes* Y*. Let* $T_0'(\mathrm{S}_0')$ *be the merge of* T_0 *and T and* $T^{miss} = T^{cand} - T^{ct}$ *be the set of all tuples in T lost in the merge with* T_0*. We can define a completion table* $T^{com}(S_0')$ *for all dimensions* D_i *of schema* S_{D_i} *in* T_0*,* $0 \leq i \leq n$*, where* $(\mathrm{S}_{D_i} \cap \mathrm{S}) \subset (\mathrm{S}_{D_i} \cap \mathrm{S}_0)$*:*

$$T^{com} = \Pi_{\mathrm{S}_0'}(T^{miss} \rtimes_{\mathrm{S}_{D_0} \cap \mathrm{Y}} D_0 \rtimes \cdots \rtimes_{\mathrm{S}_{D_n} \cap \mathrm{Y}} D_n) \quad (1)$$

If $\mathrm{Y} \mapsto \mathrm{S}_0$ *(*non-ambiguous merge condition*), then*

$$\mathcal{Q}_m(T_0, T) = T_0' \cup T^{com} \quad (2)$$

is a complete merge *of* T_0 *and T with respect to T.*

In the above proposition, Table T^{miss} identifies the tuples of T^{cand} that are missing in T_0' to get a complete merge. Expression 1 augments T^{miss} with all dimension attributes and values that exist in T_0 but not in T. This step avoids generating *null* values for these attributes and can be skipped if Y already contains all the dimension attributes in T_0. The *non-ambiguous merge condition* for Expression 2 checks if Y contains an identifier of T_0. If this condition is true and T_0 is not ambiguous, the merge result is also not ambiguous.

We can apply Proposition 7 to compute the aggregable property of the completed merge $\mathcal{Q}_m(T_0, T)$. However, when the merge T_0' of T_0 with T is not complete, for all aggregable attributes A where $\mathbf{agg}_{\mathrm{A}}(F, V)$ holds in T (A is an attribute obtained from T) the aggregable property $\mathbf{agg}_{\mathrm{A}}(F, \emptyset)$ holds in T_0' (A is not aggregable in T_0').

Example 28. *Resuming Example 25, we obtain* $T^{miss} = T^{cand} - T^{ct} = \{t_1, t_4\}$. *Thus, using the notations of Definition 16,* $T^{com} = \{t_1, t_4\}$ *and* Q_m *returns tuple* t_5, t_6 *augmented with* $M = null$ *and* $A_3 = null$ *completed by tuples* t_1 *and* t_4 *from* T. Q_m *is a complete merge with respect to* T.

5. Architecture Overview in SAP HANA

The contributions presented in this paper have been implemented and published within the SAP HANA platform [15] as a REST application service. The extended HANA architecture is depicted in Figure 9. White boxes are new components that implement the algorithms described in Section 6 whereas grey boxes are existing HANA components that have been extended.

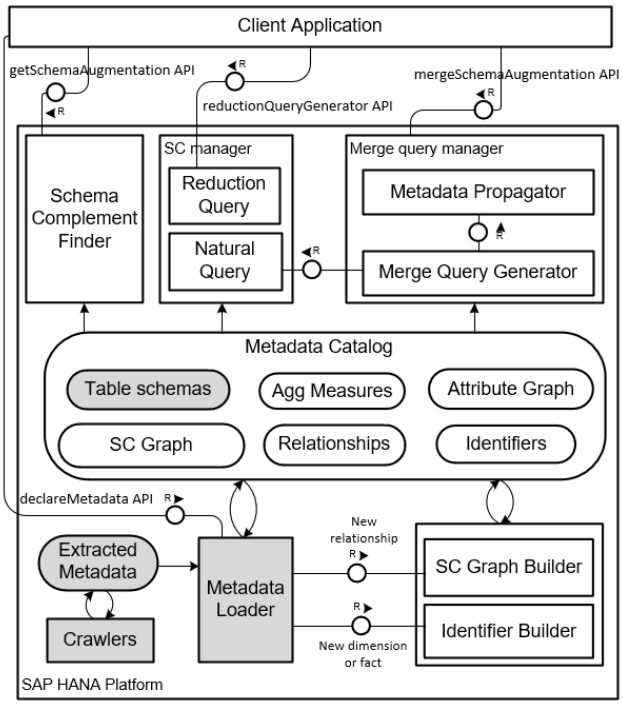


Figure 9: Architecture overview

Analytic tables are defined in SAP HANA as non-recursive *information views* [16] over non-analytic tables and other information views using a set of relational operators (*e.g.*, union, join, projection, aggregate). For example, all dimension and fact tables in Section 3.1 can be expressed as information views. Dimension tables can be either defined as arbitrary views over non-analytic tables or as project-union views over other dimension tables. Dimension attribute hierarchies are part of the dimension table definitions. The schemas of these information views, including the role played by dimension attributes or measure attributes as defined in Sections 2.1 and 2.2, are stored in the Metadata Catalog. We extended the metadata of information views by enabling a user to declare the aggregable properties of measure attributes and the attribute graphs of dimension tables. Default values are used for missing aggregable properties (see Section 2.3). All these metadata are also stored in the Metadata Catalog (Agg Measures and Attribute Graph).

The SAP HANA platform provides a framework of schedulable metadata crawlers that asynchronously extract metadata from the definition of tables and information views [17]. Extracted metadata are periodically stored by the Metadata Loader in the Metadata Catalog. We extended the Metadata Loader as follows. When a new or updated dimension table is crawled, the Metadata Loader calls the Identifier Builder which uses its attribute graph to compute and store its dimension identifier. Similarly, the Identifier Builder is called to compute and store the fact identifier of new fact tables from the attribute graphs of their dimensions. In both cases, the Metadata Loader stores the attribute mapping and join relationships extracted from the information views. When new non-analytic tables are crawled, primary and foreign keys (PK/FK) are extracted from the Table schemas and new relationships are created by the Metadata Loader. The Metadata Loader also recursively composes the relationships and adds them to the set of relationships. If new relationships are derived for a pair of tables, they are recursively fused, removed from the set of relationships, and kept separately for maintenance. New relationships are passed to the SC Graph Builder that updates a Schema Complement (SC) Graph.

Our new service offers four main REST APIs. The Declare Metadata API enables designers to declare relationships between tables which are passed to the SC Graph Builder. It also supports the declaration of all metadata (attribute graph, aggregable attributes, and relationships) generated during the merge of an analytic table with a schema complement, which are then passed to the Identifier Builder and the SC Graph Builder accordingly. The Get Schema Augmentation API takes as input a start table name T_0 and returns a list of schema complements and augmentations T' with their relationships from T_0 to T'. The Schema Complement Finder component processes a Get Schema Augmentation request using the SC Graph and the Table schemas (see Section 6.2). The Reduction Query Generator API takes an input a target schema augmentation table, and a set of user actions to reduce the identifier of the target table. It returns a reduction query and a description of aggregated attributes in the query. Heterogeneous units and currencies of aggregate reduction queries are managed using existing core functionalities of SAP HANA. Finally, the Merge Schema Augmentation API takes as input a start table and a target schema augmentation table, or a reduction query, and returns as output a final merge query with a list of computed metadata properties (see Section 6.4). The Schema Complement (SC) Manager uses a Reduction Query sub-manager to create a reduction query. In case of aggregate reduction, it guarantees that heterogeneous units and currencies are not aggregated together or alternatively converts all units and currencies to a common one before performing an aggrega-

Table 8: A tuple from hierarchy table HT

Attribute	Value
NODE	[North America].[United States].[Ohio].[Dublin]
PRED_NODE	[North America].[United States].[Ohio]
NODE_VALUE	Dublin
ATT_NAME	CITY
IS_LEAF	0
LEVEL_NUM	4

tion. The SC Manager uses a Natural Query sub-manager to compute a query for a path of natural schema complements, detect ambiguous results and create a complement query if needed. The Merge Query Generator is responsible for producing the final merge query. It calls the Natural Query component and the Metadata Propagator to compute the list of metadata properties associated with the augmented table using metadata propagation techniques presented earlier in Section 6.4.

6. Algorithms and REST Service Implementation

6.1. Dimension and fact identifiers

In this section we describe how to compute and maintain dimension and fact identifiers which play a central role in our work. We encode attribute graphs into two tables ATtribute Graph Node (ATGN) and ATtribute Graph Edge (ATGE). Their schema are respectively:

```
ATGN (DT_ID, ATT_NAME, LEVEL_NUM, OPTIONAL)
ATGE (DT_ID, ATT_NAME, PARENT_ATT_NAME, LABEL)
```

Each hierarchy in a dimension table is identified by a distinct DT_ID value. Table ATGN stores attribute nodes, with their level in the hierarchy and whether they can take null values (OPTIONAL = 1). Table ATGE stores all edges in attribute graphs where LABEL can take values: '+', '1', 'f'. In both tables, attribute names with value '__bot' or '__top' represent the two special attributes $\bot$ or $\top$ respectively.

Example 29. *Table 9 shows tuples that encode the attribute graph of dimension WAREHOUSE in ATGN and ATGE.*

When an attribute graph has not been defined by a user over the hierarchy of a dimension table, Algorithm 1 provides the option to efficiently compute it from a sample of the dimension table using the indexing scheme of SAP HANA, called *Hierarchy Table* (HT) [18], which encodes the nodes of a hierarchy instance represented in a dimension or a fact table.

Example 30. *Node 'Dublin' of 'Ohio' in the hierarchy of REGION in Figure 3 is encoded as one tuple in HT (attribute names are on the left) shown in Table 8. It contains information about its path from the top-level node, its parent node, its attribute name, whether it is a leaf node, and its level number in the hierarchy.*

Algorithm 1 Attribute Graph Generation (ATG)

input: *HT* hierarchy table
output: attribute graph

1. *Initialization.* All attributes in ATGN are initialized with OPTIONAL = 0; edges having PARENT_ATT_NAME = $\top$ are initialized with LABEL = f; edges having ATT_NAME = $\bot$ are initialized with LABEL = +.
2. Find all attributes (except bottom level attributes $\bot$) in HT that have 'IS_LEAF = 0' and add an additional edge from $\bot$.
3. Update ATGN by marking all attributes that are nullable with OPTIONAL = 1.
4. For each (ATT_NAME, PARENT_ATT_NAME) pair in ATGE, if for a NODE_VALUE value for the same attribute ATT_NAME, there are more than one PRED_NODE values for PARENT_ATT_NAME, then mark the edge with label +. If there is only one single value in PARENT_ATT_NAME then mark the edge with label **f**.
5. For each + labeled edge (ATT_NAME, PARENT_ATT_NAME) in ATGE where ATT_NAME is optional in ATGN, if for all non-null NODE_VALUE values for the same attribute ATT_NAME, there is only one PRED_NODE value for PARENT_ATT_NAME then update the edge with label **1**.

Given an attribute graph over some attribute hierarchy, a dimension identifier for all valid tables with respect to that attribute graph, is computed using the CDI algorithm. The worst-case time complexity of this algorithm is linear in the size (number of nodes and edges) of attribute graph $\mathcal{D}$.

Example 31. *We apply function CDI on the attribute graph of REGION in Figure 4. dimId is initialized with all attributes of REGION. Only CONTINENT is removed, because it has a label* **f** *in-edge. In Line 16, the result dimension identifier dimId is* {*CITY, STATE, COUNTRY*}.

By Proposition 2, the union of the identifiers of each group of dimension attributes occurring in the fact table defines a fact identifier of a fact table. However, this identifier might not be minimal. For example, when the fact table is defined as a view from a non-analytic table for which we know a key, if all attributes of this key have an attribute mapping relationship into a (strict) subset of the fact identifier attribute, this subset is also a fact identifier.

Example 32. *Recall the fact table* INVENTORY *of Example 6. Suppose that this table is defined as a view over non-analytic table* INVENTORY_ct *with primary key* K ={*PROD_SKU, YEAR, WH_ID, COUNTRY*}.

Table 9: Attribute graph for dimension WAREHOUSE

(a) ATGN

DT_ID	ATT_NAME	LEVEL_NUM	OPTIONAL
1	__bot	0	0
1	WH_ID	1	0
1	CITY	2	0
1	STATE	3	1
1	COUNTRY	4	0
1	__top	5	0

(b) ATGE

DT_ID	ATT_NAME	PARENT_ATT_NAME	LABEL
1	__bot	WH_ID	'+'
1	WH_ID	CITY	'f'
1	CITY	STATE	'+'
1	WH_ID	STATE	'f'
1	STATE	COUNTRY	'1'
1	CITY	COUNTRY	'+'
1	COUNTRY	__top	'f'

Algorithm 2 Compute Dimension Identifier (CDI)

input:
  $\mathcal{D}$: Attribute graph of dimension table with schema S
  X: A subset of S
output:
  dimId: Dimension identifier for S
begin
  Initialize: $dimId \leftarrow$ X
  for Attribute A in *dimId* **do**
    if (A has one label **f** in-edge from X in $\mathcal{D}$) **or** (A only has label **1** in-edges from X in $\mathcal{D}$) **then**
      $dimId \leftarrow dimId - \{A\}$
    end if
  end for
  return *dimId*
end

INVENTORY_ct*(PROD_SKU, YEAR, WH_ID, COUNTRY, TAX_NO, QTY_ON_HAND, TAX_AMT)*

Each attribute of the primary key has a corresponding attribute mapping relationship with one attribute of the initial fact identifier K′ ={*PROD_SKU, YEAR, WH_ID, COUNTRY, TAX_NO* } *computed by function CDI as given in Example 6.* K *is strictly included in* K′ *and can be used instead of* K. *It implicitly follows that dimension TAX depends on the other dimensions of that fact table.*

An update of a dimension table may violate the attribute graph constraints defined for that table, and hence the dimension identifier generated from that attribute graph. Thus, when a non-analytic table is updated, we must check if the attribute graph is still satisfied and the dimension identifier is still valid in T. If the uniqueness test fails, the update must be rejected. Since dimension tables are not frequently updated, the uniqueness test entails a small overhead in the transaction processing workload. In addition, the uniqueness test can be efficiently implemented using Hierarchy Tables.

6.2. Computing schema augmentations and complements

The Metadata Catalog contains the *native* relationships that are extracted from the definition of analytic tables or explicitly declared by the user, and the *derived* relationships obtained using composition and fusion. From these relationships, schema augmentations and complements are discovered by exploring a Schema Complement graph defined as follows.

Definition 17 (Schema Complement graph)**.** *Let* $\mathcal{T} = \{T_0, T, \cdots, T_n\}$ *be a set of tables and* $\mathcal{R} = \{\mathsf{R}_{jk}, \cdots, \mathsf{R}_{nm}\}$ *be a set of relationships between tables in* $\mathcal{T}$*, where* $\mathsf{R}_{jk} \in \mathcal{R}$ *is a relationship between* T_j *and* T_k*. A* Schema Complement ($\mathcal{SC}$) graph *for* $\mathcal{T}$ *and* $\mathcal{R}$ *is a directed property graph* $\mathcal{SC} = (\mathcal{T}, \mathcal{E})$ *connecting the tables in* $\mathcal{T}$ *by a set of labeled edges* $\mathcal{E}$ *where:*

1. *for each* $\mathsf{R}_{jk} \in \mathcal{R}$ *there exist two edges* $\mathcal{E}(T_j, T_k)$ *and* $\mathcal{E}(T_k, T_j)$.
2. *each edge* $\mathcal{E}(T_j, T_k)$ *is labeled by a* complement type $\mathcal{E}.CT$*: if the common attributes of* R_{jk} *contain the identifier of* T_k *then,* $\mathcal{E}.CT$ = 'NAT' *(natural edge) else* $\mathcal{E}.CT$ = 'AUG' *(augmentation edge).*

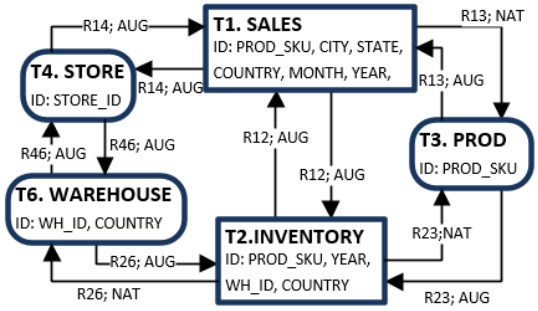


Figure 10: SC graph example for Figure 8

Example 33. *Figure 10 shows a partial* $\mathcal{SC}$ *graph for the relationships of Figure 8. Each edge is labelled with a reference to a relationship and its CT label. Nodes represent tables with their identifiers (ID).*

The Get Schema Augmentation API is implemented by the function *CSA* in Algorithm 3. Given a table T_0,

Algorithm 3 Compute Schema Augmentation (CSA)

```
1: input:
2:     T_0: start node (table)
3:     SC: schema complement graph
4: output:
5:     result: [(T, CT, newAttrsrs, rid)] :
6:     – T: destination table
7:     – CT: complement type
8:     – newAttrs: new attributes
9:     – rid: relationship with destination T
10:    altEdges: [(T, CT, rid)] : list of optional paths
11: begin
12:    result and altEdges are empty
13:    mark T_0 as visited
14:    for all edges e(T_0, T_i) in SC do
15:        if T_i is not visited then
16:            mark T_i as visited
17:            newAttrs ← new attributes in T_i w.r.t. T_0
18:            add (T_i, e.CT, newAttrs, e.rid) to result
19:            if e.CT = 'NAT' then
20:                add CSA(T_i, SC) to (result, altEdges)
21:            end if
22:        else
23:            add (T_i, e.CT, e.rid) to altEdges
24:        end if
25:    end for
26:    return (result, altEdges)
27: end
```

the function CSA explores a possibly cyclic SC graph and returns an array of schema augmentations for T_0. Lines 13 and 16 avoid infinite loops during traversal by marking visited nodes. Also, if multiple paths exist between T_0 and a schema complement T, a single edge from T_0 to T is added to *result* and alternative edges are added to *altEdges*. Thus, a single path of relationships from T_0 to T is built in *result*, which depends on the edge $\mathcal{E}(T_0, T)$ selected at the first iteration. Line 20 recursively continues the exploration for 'NAT' edges, possibly adding new arcs to *result*. Line 17 compares the attributes of T_0 and T as follows. If T_0 has never been augmented with schema augmentations before, any attribute of T that is not in common with T_0 is considered to be new. Otherwise, the attributes of T_0 that come from previous schema augmentations with T have metadata that indicate their provenance from T and *newAttrs* is empty. In both cases, a new entry is added to *result* to construct the full path from T_0 to a subsequent schema augmentation T.

Our implementation of the CSA algorithm leverages the SAP HANA graph engine [19] which supports various graph algorithms over graph data stored in a columnar table storage (using a table of nodes and a table of edges). First, we use the GET_NEIGHBORHOOD algorithm of the graph engine to compute the set *Reachable* of all tables that are *reachable* from T_0 through a sequence of natural 'NAT' edges followed by at most one augmentation 'AUG' edge. We use the term *connected* for nodes which are connected by any path without taking account of this restriction. The GET_NEIGHBORHOOD performs a breadth-first search and returns all reachable tables. Second, we select all edges whose start node and end node are in: $T_0 \cup Reachable$ and order them by respecting the ascending depth with respect to T_0. These edges are then annotated with their set of new attributes which is retrieved using SQL queries over the table schemas in the Metadata Catalog. Finally, the resulting $\mathcal{SC}$ graph is built and traversed in the main-memory of the application server engine. Given the edge ordering in the $\mathcal{SC}$ graph according to their depth with respect to T_0, algorithm CSA always returns the shortest path to a given table in *Result*. The use of a clever exploration strategy is left open for future work. However, in *Result*, schema augmentations and complements are returned sorted according to the depth-first search exploration of the $\mathcal{SC}$ graph.

Example 34. *We illustrate our* iterative implementation *of algorithm CSA with input T_1 on Figure 10. First, CSA computes all tables which are reachable from T_1 in $\mathcal{SC}$ by a sequence of NAT edges and followed by at most one AUG edge. We obtain, Reachable $= \{T_2, T_4, T_3\}$. The next step retrieves the list of edges connecting nodes in $T_0 \cup Reachable$, ordered by their distance from T_1 : Edges $= [\mathcal{E}(T_1, T_2), \mathcal{E}(T_1, T_4), \mathcal{E}(T_1, T_3), \mathcal{E}(T_3, T_2)]$. The algorithm then processes each edge $\mathcal{E}(T_k, T_i)$ in Edges as follows. If the target table T_i has not been visited yet, the set of new attributes $newAttrs_i$ is computed for T_i and the edge is added to Result. All other edges are added to altEdges. In our example, this produces $(T_2, AUG, newAttrs_2, \mathsf{R}_{12})$ for edge $\mathcal{E}(T_1, T_2)$, $(T_4, AUG, newAttrs_4, \mathsf{R}_{14})$ for edge $\mathcal{E}(T_1, T_4)$ and $(T_3, NAT, newAttrs_3, \mathsf{R}_{13})$ for edge $\mathcal{E}(T_1, T_3)$. For edge $\mathcal{E}(T_3, T_2)$, table T_2 is marked as visited and $(T_2, AUG, \mathsf{R}_{23})$ is added to altEdges.*

Suppose now that $(T_2, AUG, newAttrs, \mathsf{R}_{12})$ is selected for augmenting the schema of table T_1, resulting in a new table T_1'. Then there might appear new tables that are reachable from T_1' in the graph of Figure 10, which were not reachable from the original table T_1. For example T_6 is only accessible through two AUG edges from T_1, but by a single AUG edge from T_1'. However, we can show that T_1' will never connect to more tables than to those that are already connected to T_1 and T_2 in $\mathcal{SC}$. In other terms, all schema augmentations reachable from a schema merge table T_1' in $\mathcal{SC}$ are also connected to the original table T_1 in $\mathcal{SC}$.

6.3. Creating a reduction query

The Compute Schema Augmentation (CSA) API returns a set of schema augmentations for some source table T_0. Assume that a user selects a schema augmentation table T with complement type 'AUG' and relationship path $\mathsf{R}_1(T_0, T_1), \cdots, \mathsf{R}_n(T_{n-1}, T), n >= 1$ as a schema augmentation for T_0. Let Y_n be the set of common attributes of

T_{n-1} and T, and K be the identifier of T. Then, a natural schema complement to T_0 can be obtained by reducing all attributes in $\mathrm{K} - \mathrm{Y}_n$ through one or several reduction queries. Table 10 shows the reduction actions that a user can express on each attribute A of $\mathrm{K} - \mathrm{Y}_n$ and their impact in terms of reduction queries. Without loss of generality, we consider here that each user action reduces a single attribute of $\mathrm{K} - \mathrm{Y}_n$.

Table 10: User actions to create a reduction query

User action	Impact on reduction query
Filter	defines a set $FilP$ of filter predicates $\mathrm{A}_i = v_i, v_i \in dom(\mathrm{A}_i)$; yields a filter reduction;
Pivot	defines a set of attributes $PivA$ that are pivoted as columns; yields a pivot reduction;
Remove	defines a set of attributes $RemA$ which are removed from the result of the reduction query; yields an aggregate reduction;
Aggregate	defines a set $AggA$ of aggregated attributes $F_i(\mathrm{A}_i)$; yields an aggregate reduction

We now introduce the Reduction Query Generation (RQG) API which takes five inputs: (T, $FilP$, $PivA$, $RemA$, $AggA$), and returns two outputs as described below:

Algorithm 4 Reduction Query Generation (RQG)

input:

T:	destination schema augmentation table
$FilP$:	a set of equality predicates of the form $\mathrm{A} = v$ for *filtered* attributes $\mathrm{A} \in \mathrm{K}$
$PivA$:	a set of *pivoted* attributes
$RemA$:	a set of attributes *removed* from K
$AggA$:	a set of *aggregated* attributes of the form $F(\mathrm{A})$ where $\mathrm{A} \in \mathrm{S}$

output:

$\mathcal{Q}$:	reduction query
$AggA'$:	aggregated attributes of the form $F(\mathrm{A})$ in $\mathcal{Q}$

Verify inputs and order reduction operations. The reduction process is separated into three optional reduction steps whose execution depends on the user input. Filter and Pivot are executed, respectively, when $FilP$ and $PivA$ are not empty. Aggregate is executed when $RemA$ or $AggA$ is not empty. The default ordering of reduction operations is: 1) Filter, 2) Pivot, 3) Aggregate.

Step 1: Apply filter reductions. A filter reduction query $\mathcal{Q}_{Fil} = \mathcal{F}ilter_T(FilP)$ on the target schema augmentation table T is created and executed.

Step 2: Apply pivot reductions. A pivot reduction query $\mathcal{Q}_{Piv}$ is generated over the result of query $\mathcal{Q}_{Fil}$. It is of the form $\mathcal{P}ivot_{\mathcal{Q}_{Fil}}(\mathrm{X}_{val} \mid PivA)$, where X_{val} is the set of aggregable attributes in T.

Step 3: Apply aggregate reductions and detect if the result is ambiguous. First, each aggregate $F_i(\mathrm{A}_i)$ in $AggA$ is checked for correctness with respect to the aggregable properties of the attribute A_i and attributes $RemA$. Then, an aggregate query $\mathcal{Q}_{Agg}$ is generated over the result of query $\mathcal{Q}$ generated in the previous Filter and Pivot reduction steps. It is of the form: $\mathcal{A}gg_{\mathcal{Q}}(F_i(\mathrm{A}_i), \ldots \mid \mathrm{X})$, where each $F_i(\mathrm{A}_i)$ belongs to $AggA$ or to the new attributes whose values were pivoted in $\mathcal{Q}$, and $\mathrm{X} = \mathrm{K} - RemA - PivA$.

Generate reduction query. The output reduction query $\mathcal{Q}$ is generated by the previous Filter, Pivot and Aggregate reduction steps. $AggA'$ is the set of aggregable attributes containing attributes from input $AggA$ and attributes whose values were pivoted in the pivot reduction query step (i.e., the X_{val} attributes). This set is needed for the propagation of aggregable properties (see Section 4.1). The identifier K' of the result of $\mathcal{Q}(T)$ is $\mathrm{K}' = \mathrm{K} - FilA - RemA - PivA$ where K is the identifier of T, $RemA$ and $AggA$ are defined in Table 10 and $FilA$ corresponds to the attributes in the filter predicates $FilP$.

Example 35. *Consider* T_1 = SALES *is a schema augmentation to* T_2 = INVENTORY *in the $\mathcal{SC}$ graph of Figure 10. When augmenting the schema of* T_1 *with* T_2*, a reduction query can be expressed as an RQG call:* $RQG(T_1, \emptyset, \emptyset, RemA, \{SUM(AMOUNT)\}$, *where* $RemA = \{MONTH, STATE\}$. *Because $PivA$ and $FilP$ are empty, a single aggregate reduction is executed:* $\mathcal{Q}_{Agg} = \mathcal{A}gg_{T_1}(SUM(AMOUNT) \mid \mathrm{X})$, $\mathrm{X} = \{PROD\_SKU, YEAR, CITY, COUNTRY\}$. *Indeed,* X *is a subset of the identifier of* T_1 *and attributes* MONTH *and* STATE *have been reduced. Thus, the RQG call finally returns a reduction query $\mathcal{Q}_{Agg}$ and $\{SUM(AMOUNT)\}$ as a single aggregated attribute.*

Example 36. *An alternative way to reduce* SALES *when augmenting the schema of* INVENTORY *is to use an RQG call:* $RQG(T_1, \{STATE = \text{'Ohio'}\}, \{MONTH\}, \emptyset, \emptyset)$ *on the $\mathcal{SC}$ graph of Figure 10. Because AggA and RemA are empty, only filter and pivot reductions are applied. A filter reduction is created in step 1 as* $\mathcal{Q}_{Fil} = \mathcal{F}ilter_{T_1}(\{STATE = \text{'Ohio'}\})$. *In step 2, the pivot reduction is applied on the result of $\mathcal{Q}_{Fil}$ as* $\mathcal{Q}_{Piv} = \mathcal{P}ivot_{\mathcal{Q}_{Fil}}(\{AMOUNT\} \mid \{MONTH\})$. *Thus, the RQG call returns a query $\mathcal{Q}_{Piv}$, and an empty set of aggregated attributes.*

6.4. Computing an augmented schema

Suppose that a user selects a table T_{dest} returned by the Compute Schema Augmentation API call as a schema augmentation to a table T_0 with respect to a path of relationships $\mathsf{R}_1(T_0, T_1), \cdots, \mathsf{R}_n(T_{n-1}, T), n >= 1$. We now introduce the Merge Schema Augmentation (MSA) API, which takes five inputs (T_0, T, *path*, *noamb*, *comp*) and returns one merge query $\mathcal{Q}$ as described below. The input table T is defined as follows. When the chosen schema augmentation T_{dest} has not been reduced, the input T is equal to

T_{dest} with a set of new attributes X_{new} and $\mathsf{R}_n(T_{n-1}, T) = \mathsf{R}_n(T_{n-1}, T_{dest})$ maps to an edge in the schema complement graph $\mathcal{SC}$. Otherwise, $T = \mathcal{Q}_{red}(T_{dest})$ represents the reduction query $\mathcal{Q}_{red}$ over T_{dest} obtained as an output of the Reduction Query Generation (RQG) API call, and $\mathsf{R}_n(T_{n-1}, T)$ corresponds to a natural schema complement edge (CT = *'NAT'*). The Boolean parameter *noamb* is *true* when no ambiguous value must appear in a merged result and the Boolean parameter *comp* is true when the merge must be complete.

Algorithm 5 Merge Schema Augmentation (MSA)

Inputs:	
T_0:	starting table
T:	destination table T_{dest} or a reduction query $\mathcal{Q}_{red}$
X_{new}:	attributes in T that will be merged to T_0
path:	$\{\mathsf{R}_1(T_0, T_1), \cdots, \mathsf{R}_n(T_{n-1}, T)\}$
noamb:	*true* when result must be non-ambiguous
comp:	*true* when merge must be complete
Output:	
$\mathcal{Q}$:	final merge query

An *MSA* call is processed in three steps detailed thereafter.

Step 1: Create query $\mathcal{Q}_a$ to update ambiguous tuples in T. When flag *noamb* = *false*, this step does not check for ambiguous results and returns $\mathcal{Q}_a = T$. When flag *noamb* = *true*, we first check if T is non-ambiguous. When T is not ambiguous for each dimension D in T w.r.t. Proposition 8, then we return $\mathcal{Q}_a = T$. Otherwise, by Definition 14, detecting if T contains ambiguous values requires computing for each dimension D used in T, the identifier of the ancestors $\mathrm{X}^*(D)$ of all key attributes in T from dimension D. This identifier is computed over the attribute graph of dimension D restricted to the attributes in $\mathrm{X}^*(D)$ and using Proposition 1. When T is detected as possibly ambiguous with respect to a dimension D, we build a query $\mathcal{Q}_a$ which replaces the measure values of all ambiguous tuples in T by *null* values. A tuple t in T is detected as ambiguous if there exists a dimension D and two tuples $t_1, t_2 \in D$ such that $t_1.\mathrm{X}^*_D \not\equiv t_2.\mathrm{X}^*_D$ and $t_1.\mathrm{X}_D \equiv t_2.\mathrm{X}_D \equiv t.\mathrm{X}_D$. For this, query $\mathcal{Q}_a$ joins the result of T with each dimension table D on their common attributes X_D to obtain all X^*_D attribute values, and *nullifies* (invalidates) all measure attributes of all tuples t where the size of the partition corresponding to t_{X_D} is greater than 1.

Step 2 (optional): Create a query $\mathcal{Q}_c$ to compute a completion table for the merge of T_0 with T. When flag *comp* = *true*, the we apply the steps of Proposition 9 to build a query $\mathcal{Q}_c$ which computes a completion table T^{com}. Let Y be the set of common attributes between T_0 and T (common attributes are obtained by composition of the relationships of *path*). If the LFD $\mathrm{Y} \mapsto \mathrm{S}_0$ holds in $T_0(\mathrm{S}_0)$, then the non-ambiguous merge condition of Proposition 9 is satisfied and a query $\mathcal{Q}_c$ is generated to create the completion table T^{com} that completes the merge of T_0 with $\mathcal{Q}_a$ of Step 1.

Step 3: Create the final query merging T_0 with the results of Steps 1 and 2. First, a query $\mathcal{Q}_{path}$ is created to merge T_0 with the sequence of natural schema complements $T_1, \cdots, T_{n-1}$ represented by $\mathsf{R}_1(T_0, T_1), \cdots, \mathsf{R}_{n-1}(T_{n-2}, T_{n-1})$ in parameter *path*. Let Y_i be the common attributes in relationship R_i. Then, $\mathcal{Q}_{path} = T_0 \bowtie_{\mathrm{Y}_1} T_1 \cdots \bowtie_{\mathrm{Y}_{n-1}} T_{n-1}$. Next, $\mathcal{Q}_{path}$ is merged with $\mathcal{Q}_a$ and we obtain $\mathcal{Q}' = \pi_{\mathrm{S}'_0}(\mathcal{Q}_{path} \bowtie_{\mathrm{Y}_n} \mathcal{Q}_a)$, where S'_0 is the schema of T_0 augmented with the new attributes X_{new} coming from $\mathcal{Q}_a$. Finally, if *comp* = *true*, then $\mathcal{Q}_c$ (obtained from Step 2) will be added to the previous result $\mathcal{Q}'$ and the final result is $\mathcal{Q} = \mathcal{Q}' \cup \mathcal{Q}_c$. Otherwise the final result is $\mathcal{Q} = \mathcal{Q}'$.

Example 37. *Consider a merge schema augmentation $MSA(\mathrm{T}_2, \mathcal{Q}(\mathrm{T}_1), \{SUM(AMOUNT)\}, \{\mathsf{R}_{12}\}, true, true)$ on the $\mathcal{SC}$ graph of Figure 10 (T_1=SALES is a schema augmentation to T_2=INVENTORY), and $\mathcal{Q}(\mathrm{T}_1)$ is the output of the RQG call in Example 35. Since $\mathcal{Q}(\mathrm{T}_1)$ is a reduction query, we add a natural complement edge $R(\mathrm{T}_2, \mathcal{Q}(\mathrm{T}_1)$ (with CT = 'NAT') to the schema complement graph.*

– *Step 1: The destination table $\mathcal{Q}(\mathrm{T}_1)$ is a reduction query and may contain ambiguous values. We must compute all ambiguities over attribute PROD_SKU from dimension PROD, attributes CITY and COUNTRY from dimension STORE and attribute YEAR from dimension TIME. This step generates a query $\mathcal{Q}_a$ which joins $\mathcal{Q}$ with dimension table STORE and sets all aggregated measure attributes of ambiguous tuples $t \in \mathcal{Q}(\mathrm{T}_1)$ to null. For example, suppose that STORE contains two tuples t_1 and t_2 with the same values for attributes CITY and STATE and different REGION values,* e.g. *$(Dublin, Ohio, UnitedStates)$ and $(Dublin, California, UnitedStates)$. Query $\mathcal{Q}$ aggregates these tuples into a single ambiguous tuple t, and $\mathcal{Q}_a$ sets the aggregated* SUM($AMOUNT$) *value in t to null*

– *Step 2: The set of common attributes between T_2 and $\mathcal{Q}(T_1)$ is $\mathrm{Y} = \{PROD\_SKU, YEAR, CITY, COUNTRY\}$. The non-ambiguous merge condition $\mathrm{Y} \mapsto A$ in Proposition 9 holds for all attributes of T_2 and a completion query $\mathcal{Q}_c = T^{com}$, as defined in Proposition 9, is created to generate tuples that complete the merge of T_2 with $\mathcal{Q}(\mathrm{T}_1)$.*

– *Step 3: Table T_2 is connected to $\mathcal{Q}_a$ by a natural schema complement edge $R(\mathrm{T}_2, \mathcal{Q}_a)$ and $\mathcal{Q}_{path} = T_2$. Then $\mathcal{Q} = \pi_{\mathrm{S}'}(T_2 \bowtie_{\mathrm{Y}} \mathcal{Q}_a)))$ is a natural merge, where S' contains the schema of T_2 augmented with attribute SUM(AMOUNT), and the final result is $\mathcal{Q} \cup \mathcal{Q}_c$.*

The following proposition states that the RQG and MSA API compute a correct merge result.

Proposition 10. *Let query $\mathcal{Q}$ be the result of a MSA(T_0, T, path, true, true) call with a start table $T_0(\mathrm{S}_0)$, a target schema augmentation table $T(\mathrm{S})$ and a path of relationships: path $= \mathsf{R}_1(T_0, T_1), \cdots, \mathsf{R}_n(T_{n-1}, \mathcal{Q}_a), n >= 1$. If*

$\mathsf{R}_n(T_{n-1}, \mathcal{Q}_a)$ *maps to an augmentation schema complement edge (CT = 'AUG'), then* $\mathcal{Q}$ *computes a augmented merge of* T_0 *with* $\mathcal{Q}_a$ *(without ambiguous values). Otherwise,* $\mathsf{R}_n(T_{n-1}, \mathcal{Q}_a)$ *maps to a natural schema complement edge (CT = 'NAT') and* $\mathcal{Q}$ *computes a natural merge* T_0 *with* $\mathcal{Q}_a$*.*

7. Related Work

The notion of natural schema complement was first proposed by [8] for web tables (tables extracted from web page) where join relationships between tables are discovered by computing schema matching similarities [20] using a combination of similarity in attribute names, data types, and values (through a variant of Jaccard similarity). In the context of web tables, [9] presents direct and holistic schema matching techniques to discover natural schema complements for a given web table and user-specified attribute names. Similarly, Data civilizer [10] uses schema matching techniques to build a linkage graph between arbitrary datasets for suggesting natural schema complements. Our solution differs firstly by leveraging more "trusted" relationships which are extracted from user defined analytic fact and dimension tables (views). Join relationships are enriched by attribute mapping relationships to express mappings extracted from view definitions relating view attributes and their corresponding source table attributes. However, regardless of these particular features, our framework does not exclude the use of other schema matching techniques, including recent ones like [21], to discover new join relationships between heterogeneous dimensions (e.g., like between *STORE* and *WAREHOUSE* in the examples of this paper). Secondly, we extend previous schema complement models with the notion of reduction-based schema complements which come with new data quality issues. We formally define these issues and show how to check and solve them.

Another notion similar to schema complements is "drilling-across" fact tables related through "conformed dimensions" [1, 2], *i.e.*, through dimensions which are either identical or where one is a subset of the other. Drill-across queries perform left or full outer joins operations between fact tables on their common conformed attributes. Our solution generalizes this notion by handling the more general case of merge queries between dimension tables and between fact and dimension tables. To support "drill-across" queries through two related but non-conformed dimensions, [22] proposes to build the intersection or union of these dimensions provided that they are "compatible", *i.e.*, they can be related by a perfect matching that is coherent, sound, and consistent. Our model is more general because it relaxes each of these constraints on dimensions. Indeed, relaxation of *coherence* is managed by our definition of a merge query which accepts the duplication of common attributes in a merge. *Soundness* and *consistency* constraints are respectively handled by our definition of complete merge, and the detection of ambiguous tuples in the merge results. Thus, our solution can find a schema augmentation over *incompatible* dimensions to a given fact table and compute a merge corresponding to a drill-across query that would not be accepted by [22]. Finally, although the idea of using arbitrary relationships between tables to express general drill-across queries has early been introduced in [23], no well-defined and implemented method to support such queries was published before. To the best of our knowledge, our solution is the first implementation that addresses the use cases presented in [23].

The quality of aggregate queries over fact tables has extensively been studied in the literature through the notion of summarizability defined as the "correct computation of aggregate values with a coarser level of detail from values with a finer level of detail" [13]. Summarizability was introduced in [24] and various conditions to ensure summarizability have been proposed [25, 26, 27, 28]. We adapt these results to the context of schema complements and the use of SQL semantics with nulls to characterize, for example, ambiguous aggregable attributes in the presence of non-strict hierarchies. We also leverage and extend previous work on additive measures [14, 2] to define applicable aggregate functions before a merge, and propose techniques to infer which aggregate functions are applicable after a merge. This enables us to propagate metadata on analytic tables across schema augmentations. Our results go beyond summarizability because, as with the approach suggested in [29, 25], we tolerate inaccurate measure values in a merge result, but clearly identify theses values and control their usage through their aggregable properties or the generation of null values. In addition, our method of computing complete merges is novel and was not considered in previous works on summarizability.

Dimension identifiers are essential for checking the coherence of analytic operations, and especially for verifying the semantic correctness of joins between analytic tables [1, 2, 22]. The work presented in [30] introduced the constraint that the bottom-level dimension attributes determine all other attributes in the dimension hierarchy. A similar kind of user-defined dimension identifiers are *surrogate keys* [2], which correspond to "artificial" bottom-level key attributes like *CUSTOMER_ID* or *TIME_ID*. An analogous approach has been introduced in [28] by enforcing linear dimension hierarchies, where each attribute has at most one parent attribute. On the one hand, the previous constraints and definitions simplify the problem of determining the dimension identifiers and to avoid the definition and analysis of LFD dependencies between dimension attributes. On the other hand, the trade-off of this simplification is the incapacity to infer new dimension identifiers for the result of operations which reduce the number of dimension attributes. This also restricts the set of composable operations to those which preserve these surrogate keys.

Example 38. *Table* SALES *in Figure 8 contains attributes*

MONTH, YEAR from dimension TIME. Suppose that the bottom-level attribute DAY is the surrogate key (dimension identifier) of dimension TIME. Then, without additional information, we cannot determine the new fact identifier of SALES*, as we do in Example 6 by using the corresponding attribute graphs.*

8. Experimental Evaluation

8.1. Attribute Graph and Dimension identifier Computation

All performance experiments for attribute graph generation (ATG) and dimension identifier computation (CDI) are conducted on an SAP HANA instance with 250 GB of main memory and 300 GB of disk space.

8.1.1. Computation of an attribute graph

Our first experiment consists in testing the performance of Algorithm 1 (ATG) for the generation of attribute graphs. We generate several hierarchies (dimension tables) and compare the performance of building attribute graphs according to four parameters as shown in Figures 11 and 12: (a) the size of the input dimension table (rows); (b) the number of hierarchy nodes / dimension attributes (without $\bot$ and $\top$); (c) the number of +-edges in the resulting attribute graph; (d) the total number of edges in the resulting attribute graph.

The first experiments concern the generation of attribute graphs for linear strict hierarchies. The resulting attribute graphs for n level types then have exactly n nodes and $n-1$ **f**-edges (no + and **1**-edges). To achieve this, we generate for each dimension attribute a random number in the interval $[0, 9999]$, and concatenate this value with its parent attribute value. For example, some dimension table T_1 with attributes L1 $\preccurlyeq$ L2 $\preccurlyeq$ L3 $\preccurlyeq$ L4 $\preccurlyeq$ L5 might contain a tuple (L1:'63.789.266.7426.2629', L2:'63.789.266.7426', L3:'63.789.266', L4:'63.789', L5:'63'). The goal of this process is to produce only **f**-edges by ensuring that L1 $\mapsto$ L2 $\mapsto$ L3 $\mapsto$ L4 $\mapsto$ L5. Figures 11a and 11b show that computation time increases linearly with the input table size (5 nodes/attributes) and with the number of nodes / attributes (10k and 50k rows) for these linear strict hierarchies.

The goal of the following experiments is to show the performance evolution w.r.t. to the number of additional + and **1**-edges. These edges are created by ignoring the parent attribute value and allowing null values during the generation of an attribute value (multiple parents). The dimension tables used in the experiments contain $10K$ rows. Figure 12a shows the performance evolution in a hierarchy over 10 attributes for additional +-edges generated by non-strict hierarchies without null values. Figure 12b shows the performance evolution for hierarchies with other additional edges produced by allowing null values during the hierarchy generation process. We can see that the computation time slightly decreases in both cases with the number of additional edges. For hierarchies of less than 10 nodes and samples of less than 50 K rows, which corresponds to a large number of real use cases, the computation of an attribute graph takes less than 20 seconds.

Finally, for a dimension table with a very large sample size of $1.1M$ rows and 5 nodes, the computation of the attribute graph takes 67 seconds, which is still acceptable for an occasional system background call.

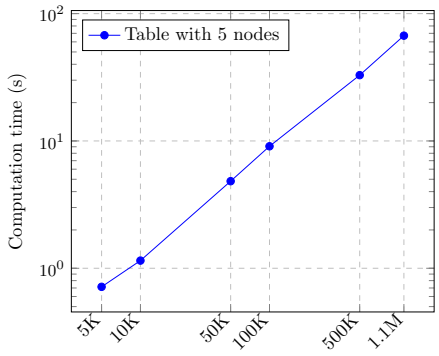
8.1.2. Computation of dimension identifiers

Algorithm 2 (CDI) takes an attribute graph $\mathcal{D}$ and a set of attributes X as input for computing the dimension identifier Y of X, such that Y $\mapsto$ X. If X is the set of all attributes of a dimension table D, Y is the dimension identifier of D. We tested 20 different attribute graphs with 5 to 20 nodes and 4 to 28 edges. All dimension identifiers of these attribute graphs are computed in less than 9 milliseconds with a maximal variation of 2 milliseconds.

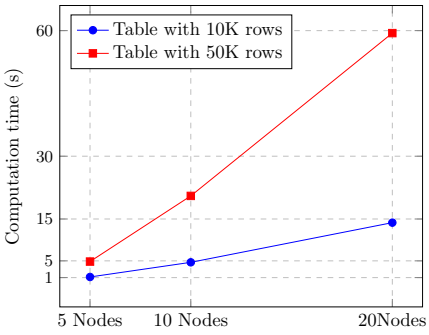
8.1.3. Comparison with related work

Inferring functional dependencies (FD) from relations is an old problem and many efficient algorithms have been proposed in the literature (see [31] for a comparison of 7 representative FD detection algorithms). These algorithms traditionally adopt NULL-NOT-EQ or NULL-EQ [32] semantics for *null* values in the dependent attribute A of a functional dependency X $\rightarrow$ A. A third approach is to apply probabilistic semantics to generate approximate FDs [32]. In our data model, we consider NULL-EQ (literal equality) semantics and we also support *null* values in the determinant attribute set X. Under NULL-EQ semantics, we then could apply an appropriate skolemization function which transforms all *null* values a dimension table into skolems by fully respecting the existing hierarchy. Then, any existing FD detection algorithm could be used to detect all LFD dependencies with NULL-EQ semantics.

However, there exists a second important difference between our setting and the standard relational setting. As it can be seen in the definition of attribute graphs, we only consider functional dependencies X $\rightarrow$ A where X and A appear in the same dimension table and all determinant attributes in X are descendants of the depending attribute A in the corresponding hierarchy type. This reduces the solution space that has to be explored and enables high pruning by leveraging the attribute hierarchy during the detection process. As our experiments show, the cost increases linearly with the number of attributes opposed to exponential increase for general FD algorithms [31, 33]. Thus, our solution is more efficient than general purpose algorithms for the discovery of identifiers in the limited case of dimension tables. Finally, the representation of LFD dependencies in the form of attribute graphs expresses known properties of hierarchies and is more natural to understand by a domain expert.

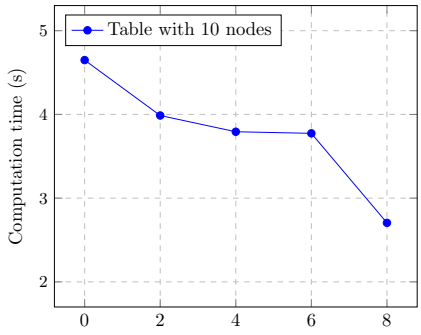


(a) Dimension table size (number of rows)

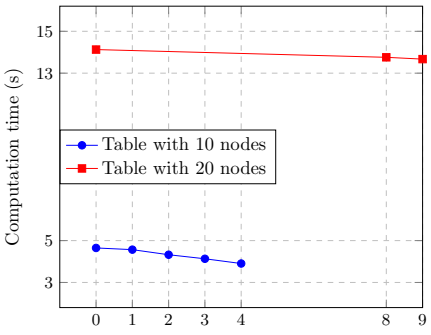


(b) Dimension table size (number of nodes / attributes)

Figure 11: Attribute graph computation time for linear strict hierarchies (**f**-edges)



(a) Number of +-edges



(b) Number of additional edges

Figure 12: Computation time for non-strict hierarchies

8.2. Business intelligence scenario

Our second series of experiments checks the usage of the schema augmentation REST service, including the CSA, RQG, and MSA APIs described in Section 6, in a business intelligence setting. For this, we chose a concrete application of a worldwide clothing company which performs stock analysis in retail stores and customer segmentation. The application contains 281 non-analytic tables used for defining 42 information views representing dimension tables, and 145 carefully optimized information views representing fact tables. We classified th efact tables into two main categories: (a) 22 *first-level views* defined by a star-join between a non-analytic table storing facts (possibly completed with left-outer joins with other tables providing details) and dimension tables, and (b) 72 *join views* defined by a set of joins, aggregations and filters over fact, dimension and non-analytic tables. The remaining fact tables are defined using union and simple aggregations over other fact tables.

8.2.1. Experiments

In the first experiment, our goal is to select a join view, also called *Hand-Crafted View*, denoted by *HCV*, as a target and verify if we can generate an equivalent view, denoted by *GV* for Generated View),by iteratively extending a first-level start view (table) through possibly several schema augmentation steps, each of which consisting of a sequence of CSA - RQG - MSA API calls.

At each iteration, we simulate a user who selects a suggested target schema augmentation, defines optional filter conditions on target table attributes and adds calculated attributes after the merge.

Before running our experiment, we first create the attribute graphs of all dimension tables. Some *HCV* views directly access non-analytic tables. For being conform to our model, which only extends analytic tables with other analytic table, we then build all missing first-level views over the non-analytic tables, which includes the definition of the aggregable properties of all measure attributes. Finally, we crawl all first-level views and their underlying non-analytic tables to extract all direct and derived relationships (including PK-FK relationships) and to compute the dimension and fact identifiers, as explained in Section 5. An extract of the resulting SC graph is shown in Figure 13.

We illustrate our protocol with the example of an existing hand-crafted view *HCV*. The definition of this view starts from a non-analytic table ct_LINEITEM, performs a sequence of left-outer joins and aggregates with five other non-analytic tables storing different facts, followed by a star-join with five dimension tables. The definition of *HCV* is shown in Figure 14 which represents dimension tables by bold rounded rectangles, fact tables by bold square rectan-

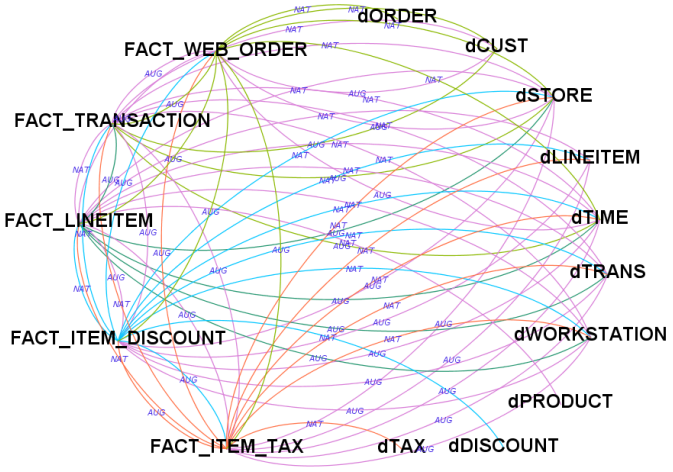


Figure 13: Extract of the SC graph for analytic tables

gles, non-analytic tables by regular square rectangles, and intermediate tables by dashed rectangles. Intermediate tables are created to store the intermediate results during the view constructions (e.g. TEMP_1 represents the result of the join between ct_LINEITEM and ct_TRANSACTION).

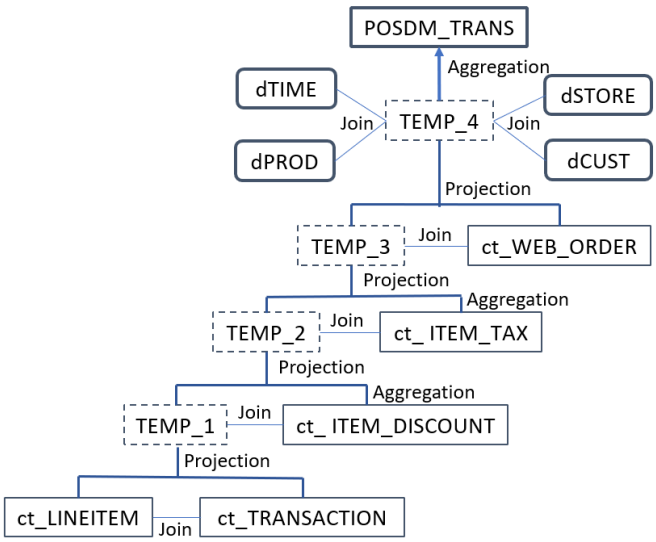


Figure 14: Construction of HCV

Figure 15 gives a partial view of the SC graph for the analytic tables joined in the *HCV* example. The table cardinalities range from $1.4M$ rows (WEB_ORDER) to $23M$ rows (ITEM_TAX and LINEITEM). All tables $D1$ to $D10$ are dimension tables and all other tables are first-level views defining fact tables. We use the notation WEB_ORDER(D5, D8) to state that the fact identifier of table WEB_ORDER only contains dimension attributes of dimension $D5$ and $D8$. For clarity, the figure does not show the edges corresponding to derived relationships. In particular, all five fact tables are pairwise connected by derived SC edges. The primary keys of non-analytic tables are propagated to the corresponding fact tables (views) to account for the dependencies between dimensions, *e.g.* in WEB_ORDER, $\{D5, D8\} \mapsto D7$.

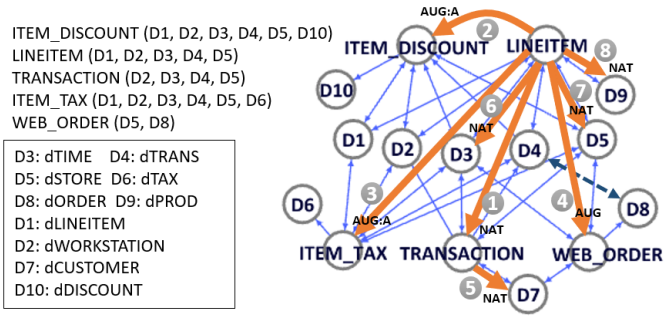


Figure 15: Partial SC graph for analytic tables

Using our REST service, we start from the first level fact table LINEITEM built from table t_LINEITEM and construct a view GV that is equivalent to HCV by following the eight successive schema augmentation steps illustrated in Figure 15. Each target table is connected to the starting table LINEITEM by a path in the SC graph and a bold arc labeled by the number of the augmentation step. In step 1, the merge query adds attributes of TRANSACTION from dimension D7, which did not exist in LINEITEM. We correctly infer that the SC edge from LINEITEM to TRANSACTION has label "NAT" by Definition 17. In step 2, we merge with fact table ITEM_DISCOUNT through an aggregate reduction using aggregate function SUM. In step 4, we merge WEB_ORDER with LINEITEM. These two tables share attributes from dimensions D3 and D5 and there also exists a user-specified relationship (dotted line in Figure 15) between dimension D4 (used in table LINEITEM) and dimension D8 (used in table WEB_ORDER). This allows us to merge WEB_ORDER with LINEITEM through schema augmentation with the common attributes between D4 and D8. All other steps use similar natural and reduction-based schema complements. After the assisted construction of GV, the equivalence both views, GV and HCV, has been verified by comparing the total number of rows and their values.

Our second experimentation goal is to compare the performance of SQL queries using the original view HCV and the equivalent generated view GV. The performance measures are done on a server with 2 TB of main memory and 1 TB of disk space. First, by comparing the total materialization cost and we can observe that computing and reading GV is 1.5 times slower than for HCV. We then apply some aggregate queries with random combinations of dimension attributes and aggregable attributes on HCV and GV. The performance results are quite positive. For 17% of the queries, GV performs from 1.15 to 3.1 faster than HCV, and for 74% of the queries, HCV performs 1.1 to 4.5 faster than GV. For the remaining queries, both views have similar performance. To obtain a better under-

standing of this behaviour, we compared several couples of execution plans generated by the same query on GV and HCV. We observed that the performance variations were caused by the optimization strategy which allocated similar query sub-plans to different execution engines. The performance variations therefore are independent of the view definitions and more related to the current HANA architecture and query optimizer.

Based on our two first experiments, we observed that for hand-crafted join views (which represents half of the fact tables of the application), an equivalent view could be semi-automatically generated using our REST service while achieving comparable execution performance. This is a very positive result because our view generation process is highly accessible to a business users who only can manipulate high-level analytic views and meaningful analytic attributes without any deep expertise about the global analytic schema and table relationships. In contrast to GV, the creation of HCV view requires a strong expertise to manually optimize the sequence of operations in the view, and a precise knowledge of the database schema to express the join conditions, decide when a pre-aggregation is necessary, identify measure attributes and choose which aggregate functions are applicable to them. The preliminary price to pay for our approach is the creation of all the necessary first-level views. However, this is rapidly amortized with the number of join views in the application. In addition, it is possible to capitalize on the SC graph for the definition of future views.

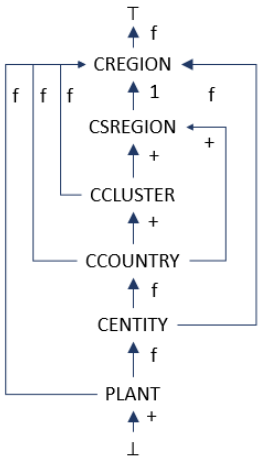


Figure 16: Attribute graph satisfied by dimension dSTORE

As another substantial advantage, our REST service provides formal quality guarantees that are difficult to be verified and confirmed by the developer of a hand-crafted HCV view. To illustrate this point, suppose that a new aggregated view AGG_WEB_ORDER is created as: $\mathcal{A}gg_{\text{WEB\_ORDER}}(SUM(\text{ORDER\_AMOUNT}) \mid \text{X} \cup A\_STORE)$ where $\text{X} = \{\text{DATE}, \text{CUSTOMER\_NO}\}$ and A_STORE is the set of the dimension attributes of dimension $dSTORE$ whose validated attribute graph is depicted on Figure 16. Now suppose that an HCV view needs to perform a join with AGG_WEB_ORDER on attributes $\text{X} \cup JA\_STORE$, where JA_STORE is a subset of A_STORE, to obtain a new measure attribute $SUM(\text{ORDER\_AMOUNT})$. Table 11 shows the cases of ambiguous values for $SUM(\text{ORDER\_AMOUNT})$, depending on the attributes contained in JA_STORE, that would be detected by our RQG API if AGG_WEB_ORDER was selected as a target schema augmentation.

Table 11: Detection of ambiguous values

	Attributes in JA_STORE	Amb	Missing att
Q_1	PLANT, CCOUNTRY, CENTITY	N	-
Q_2	CENTITY, CCLUSTER, CSREGION, CREGION	N	-
Q_3	CCOUNTRY, CSREGION	Y	CCLUSTER
Q_4	CCLUSTER, CREGION	Y	CSREGION

8.2.2. Comparison with related work

We first assess previous works with respect to their capability to generate GV through the steps marked in Figure 15. The input is given by the set of analytic tables, which are essentially tables with additional metadata, so previous solutions formulated for relational databases can be applied.

The *schema complement* approach proposed in [8] relates tables depending on their schema similarity scores using a combination of attribute name and domain similarity measures. In our setting, we can assume that the relationships shown in Figure 15 can be found using the same schema similarity technique.

Next, the merges with natural schema complements applied in steps 1, 5, 6, 7 and 8 of Figure 15 can be suggested and computed by the method described in [8]. However, ITEM_TAX, ITEM_DISCOUNT and WEB_ORDER are not natural schema complements to LINEITEM, so steps 2, 3 and 4 cannot be handled. Operators *Extend attribute* [10] and *Extend* using algorithm *JoinTest* [34] can also discover and merge natural schema complements. Although [10] proposes a technique to discover PK-FK relationships, their operator is limited to the detection of natural schema complements without reduction queries. The *ABA (Augmentation By Attribute name)* operator proposed in [9] augments the schema of a start table with an attribute, whose name is similar to a given name, coming from another related table. The discovery of related tables is as before based on a specific measure of schema similarity. However, due to the limitation that the key of the start table must be composed of a single attribute, the *ABA* operator cannot be used in our example since the start table LINEITEM has an identifier composed of five attributes.

Other related work support high-level operators over OLAP cubes (similar to our analytic tables) that can express some of the merge queries that are automatically generated by our framework. *Drill-across* queries [2] can

merge the schemas of fact tables related through *conformed dimensions*, *i.e.*, dimensions that are either identical or where one dimension is a subset of the other dimension. In our example, fact tables LINEITEM and TRANSACTION have dimensions $D2$, $D3$, $D4$ and $D5$ in common, so the merge of step 1 could be expressed as a drill-across operation. However, the remaining steps raise several issues. First, only natural schema complements can be handled due to the restriction of conformed dimensions, so the merges of steps 2, 3, and 4 cannot be expressed using drill-across operations. Second, only operations between fact tables are considered, so steps 5 to 9 would not be expressible using a drill-across operation.

Drill-across operations have been extended in [22] to fact tables with common *compatible dimensions*. In short, two dimensions are compatible if there exists a matching between the dimensions which preserves the hierarchy levels (*coherency*), level memberships (*soundness*) and roll-up relationships between members of different levels (*consistency*). We can show that dimension compatibility now allows us to execute the steps 2 and 3, since in both steps the common dimensions are compatible. However, step 4 is still not possible, since the common dimensions $D4$ and $D8$ violate the soundness condition [22]: the domain values of their common attributes, TRAN_NO in $D4$ and ORDER_NO in $D8$, are overlapping but not identical.

We now compare our detection of ambiguous values in aggregate queries with related works on *summarizability*. To illustrate, we use the aggregate queries of our experiment defined as: $\mathcal{A}gg_{\text{WEB\_ORDER}}(SUM(\text{ORDER\_AMOUNT}) \mid \text{X} \cup JA\_STORE)$, where $\text{X} = \{\text{DATE}, \text{CUSTOMER\_NO}\}$ and attributes in JA_STORE is specified in Table 11. In our example, we assume that domain values of dimensional attributes in WEB_ORDER are complete. The work on summarizability addresses the following question: "is it *correct* to compute the above aggregate queries by summarizing the result of another aggregate query (i.e.,WEB_ORDER) or should they be computed directly from the non-analytic table ct_WEB_ORDER?"

In their pioneering work, [27] proposed three sufficient conditions to guarantee summarizability of aggregation operations: (a) category (i.e., group-by) attributes must form *disjoint* subsets of facts, (b) these subsets *completely* cover the entire set of facts, and (c) dimension attributes, measures and aggregate functions are *compatible*. Among the four aggregate queries in Table 11, Q_1 and Q_2 satisfy the summarizability conditions: the categorizations of the domain values are *disjoint* and *complete*, and the measure attribute ORDER_AMOUNT is *compatible* with function SUM and dimensions dSTORE, dTIME and dCUSTOMER. However, for query Q_3, the summarizability constraints are violated: there exists a +-edge $\mathsf{R}(\text{CCOUNTRY}, \text{CSREGION})$ in the attribute graph of dSTORE, which signifies that there exists at least one CCOUNTRY value which has multiple CSREGION parent values (violation of the disjointness condition). Thus, Q_3 is not considered to be *correct*. For Q_4, all summarizability constraints are satisfied. Because edge $\mathsf{R}(\text{CCLUSTER}, \text{CREGION})$ has label $\mathbf{f}$, values in CCSLUSTER have a unique value in CREGION, the categorization of their domain values is disjoint. However, Q_4 generates ambiguous results: since edge $\mathsf{R}(\text{CCOUNTRY}, \text{CSREGION})$ is a +-label edge, CSREGION should be included in the group-by clause. Otherwise, tuples with same value of CCOUNTRY and CREGION that belong to different CSREGION will be combined together by this aggregation. Thus, the summarizability conditions of [27] conclude that Q_4 is *correct* while we detect that it is ambiguous.

The work of [35] extended the summarizability conditions of [27] and showed that summarizability holds for aggregate queries with distributive aggregate functions over *strict covering onto* dimensions. In the above aggregate queries, function SUM is distributive and the hierarchy of dSTORE is onto, but not covering and strict. Therefore, the dSTORE hierarchy needs to be modified by adding intermediate artificial values for non-covering mappings and adding concatenated parent values for non-strict mappings. After these data modifications, summarizability is guaranteed for all aggregate queries Q_1, Q_2, Q_3, Q_4. By contrast, our solution does not require any change to the data in the dimension tables; we detect ambiguous queries and can report ambiguous values.

Finally, the work of [28] only considers in their data model linear, ordered and strict hierarchies, which does not capture the cases of dimension tables in our scenario. For example, dSTORE hierarchy is not linear, not ordered nor strict.

8.3. Feature Engineering Scenario

The third series of experiments checks the usage of our REST service in a predictive analytics application of a retail bank. The decision task consists in predicting whether a client without a credit card will obtain a card in the 3 months following a given reference date. The training and prediction process uses tables from a publicly available database [36] related through PK-FK relationships as shown in Figure 17. Each account is described by static characteristics (e.g. date of creation, address of the branch) given in table ACCOUNT and dynamic characteristics (e.g. payments debited or credited, balances) given in table TRANSACTION. The two tables CLIENT and CREDIT_CARD contain information about persons and credit cards respectively. Clients and accounts are linked by a many-to-many relationship stored in the associative table DISPOSITION. Tables GEOCODE and TIME provide detailed geographical and temporal information.

The development of the application includes a *feature engineering* step to build a training dataset. Starting from a table describing a CLIENT entity and from a *reference date*, the developer augments this table through a sequence of database queries with as many new *features* (attributes) as possible. The goal is to obtain more detailed information about the past behaviour of the clients like, for example, the monthly amount and balance of their credit cards during the past 12 months, the delivery of a new

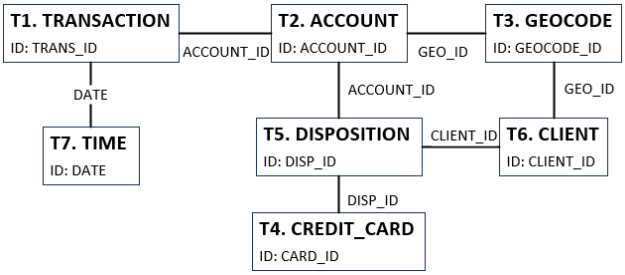


Figure 17: Relationships for the bank retail database

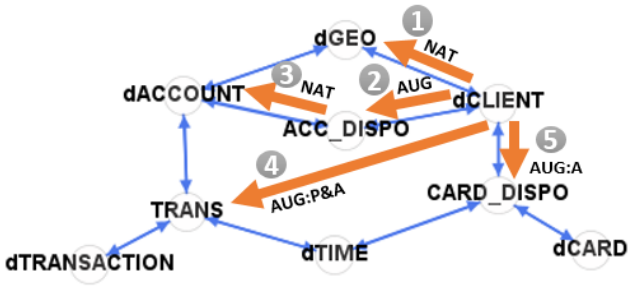


Figure 18: SC graph for the analytic tables

credit card during the past three months, etc. The final *feature dataset* is denoted by *FEAT*. The individual augmentation queries consist of joins, projections, filters, aggregations and pivots and the whole SQL script used to build the training dataset consists of about 1,500 lines of code comprising many sub-queries involving 16 joins and 132 functional expressions to compute measure attribute values.

8.3.1. Experiments

The goal of our experiment is to verify if a data scientist can semi-automatically generate an view GV that is equivalent to *FEAT* by iteratively extending an initial dimension table CLIENT through several schema augmentation steps each of which consists of a sequence of CSA - RQG - MSA API calls.

For running our experiment, we first created 6 dimension tables, denoted by dACCOUNT, dGEO, dCLIENT, dCARD, dTIME and dTRANSACTION, and 3 fact tables, denoted by TRANS, ACC_DISPO, and CARD_DISPO over the bank retail database tables. TRANS is defined over database table TRANSACTION and dimensions dTIME, dACCOUNT and dTRANSACTION and has two measures AMOUNT and BALANCE. ACC_DISPO is defined over database table DISPOSITION and dimensions dCLIENT and dACCOUNT. Finally, CARD_DISPO is defined as a join of tables CREDIT_CARD and DISPOSITION over dimensions dCLIENT, dTIME and dCARD. The attribute graphs are defined for all dimension tables, as well as the aggregable properties for all measure attributes of the fact tables. Finally, all tables were crawled by the HANA Crawlers to extract all direct and derived relationships (including PK-FK relationships) and compute the dimension and fact identifiers. The resulting SC graph among analytic tables is shown in Figure 18.

The view generation process starts from the dimension table dCLIENT(CLIENT_ID, GEO_ID, BIRTH_DATE, SEX, ...), which identifies a client entity, and applies a sequence of schema augmentation steps as indicated by the numbered arrows in the SC graph of Figure 18.

In Step 1, detail attributes from dimension dGEO are added to dCLIENT through a natural schema complement using common attribute GEO_ID. In step 2, dCLIENT is augmented with attribute ACCOUNT_ID from fact table ACC_DISPO through a schema augmentation using the common attribute CLIENT_ID (thus, rows in dCLIENT are multiplied). In step 3, the resulting table is augmented with detail attributes from dimension dACCOUNT through a natural schema complement using common attribute ACCOUNT_ID. In step 4, table dCLIENT is augmented with measures from fact table TRANS. The common attribute between TRANS and dCLIENT is ACCOUNT_ID, that is, a subset of the fact identifier of TRANS which is: {ACCOUNT_ID, DATE, TRANS_ID}. The following user actions are applied to fully reduce the fact identifier. A calculated attribute MONTH(DATE) (i.e., transaction month) is pivoted as a column with values coming from measures AMOUNT and BALANCE, and attribute TRANS_ID is removed. In addition, two user actions are provided: a filter on DATE pre-selects facts within the 12 months preceding a user-given reference date, and attribute TRANS_TYPE is pivoted as a column with values coming from measures AMOUNT and BALANCE. These two operations are injected in the query reduction generation over TRANS and the reduction query is merged with dCLIENT.

We now want to add a measure attribute storing the number of credits cards for each client. In Step 5, attributes from CARD_DISPO are added to dCLIENT through a schema augmentation using common attributes CLIENT_ID. The identifier of CARD_DISPO, consisting of {CLIENT_ID, CARD_ID}, is fully reduced by computing an aggregated CARD_ID using function $COUNT$, and a reduction query is generated over CARD_DISPO. The result is then merged with dCLIENT using the remaining common attribute CLIENT_ID. We get a result table with 156 attributes where 135 of them are measures. As in the BI Scenario (Section 8.2), the correctness of GV with respect to *FEAT* is done by comparing the number of tuples and their values in both tables. We show in Figure 19 the complete definition of GV where each temporary result represents a step in the view generation process.

Our semi-automatic view generation process has several advantages over the manual creation of table *FEAT*. First, the schema augmentations required to build a *FEAT* table are successfully suggested by our CSA API and the order in which they are returned matches pretty well the user needs. Second, simple user actions in steps 4 and 5 yield complex reduction queries involving filter, pivot, and aggregate operations through the RQG API. Isolating

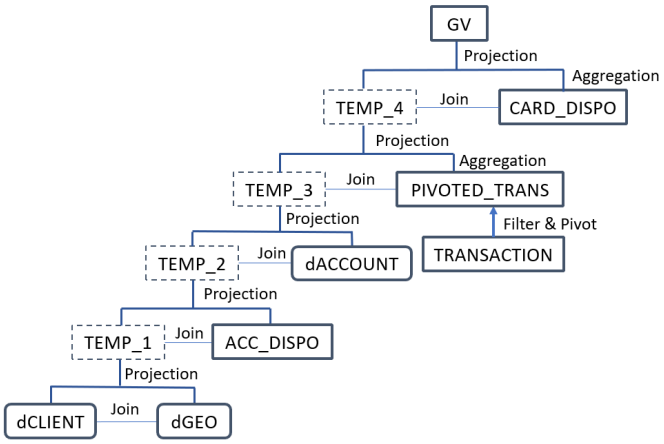


Figure 19: Construction of GV

user reduction actions in the generation process provides a great flexibility. For instance, in Step 4, filter conditions could be changed to select facts within the 2 years preceding the reference date, or pivot operations could changed to WEEK(DATE). Third, the preparatory definition of formal aggregable properties for all measures in fact table TRANS help the expert to choose the correct aggregate functions for measures AMOUNT and BALANCE in Step 4. For instance, function SUM cannot be applied to BALANCE. Finally, our method for propagating aggregable properties controls the dimensions with respect to which measure COUNT(CARD_ID) can be aggregated after Step 5 since the measure only depends on CLIENT_ID.

Note that our REST service could be directly applied to an SC graph consisting only of the retail database tables (i.e., without creating any dimension or fact table) and would still produce table *FEAT* as result. However, working directly with database tables has two main drawbacks. First, the user must understand the operational data model of the database that carries many attributes that are irrelevant for business data analysis. Second, the benefits of metadata such as the separation of dimension and measure attributes and the definition of aggregable properties would be lost.

8.3.2. Comparison with related work

We can assess other related work with respect to their ability to generate table *FEAT*. As we did in Section 8.2.2, we first consider methods that find schema complements. Starting from table dCLIENT, the previous works of [8], [10] and [34] are only able to detect natural schema complements and suggest the merge of step 1 in Figure 18. The other steps require reduction queries which are not supported by these methods. Note that the *ABA* operator of [9] can be applied on dataset dCLIENT since its identifier consists of a single attribute CLIENT_ID. However, this is again only useful for step 1.

Concerning the use of drill-across queries [2], we can show that no step in Figure 18 can be defined by a drill-across operation between two fact tables: steps 1 and 3 merge two dimension tables and steps 2, 4 and 5 merge a dimension with a fact table.

9. Conclusions and Future Work

In this paper, we propose a solution to the problem of discovering and merging schema augmentations for analytic datasets. We formally define efficient algorithms for building reduction-based schema complements which generalize previously defined natural schema complements. We discuss various quality issues and their solution for generating semantically correct schema augmentations and merged tables annotated with proper metadata enabling their correct usage in future analytic queries. We also present the implementation of our solution as a REST service in SAP HANA which, to the best of our knowledge, is the first implementation of that sort. This service assists business users and data scientists to explore a large space of possible schema augmentations and, as illustrated in our experiments, to generate complex views in a controlled way with formally defined quality guarantees concerning attribute ambiguity and data completeness.

We envision several directions for future work. A first direction concerns the selection of schema complements based on user-specified criteria such as keywords and the definition of strategies for ranking the schema complements returned by our algorithm based on user profile and contextual user preferences. Another direction is to refine our relationships to include more sophisticated temporal conditions on time intervals, and the ability to deal with look-up tables. In addition, we want to suggest ways of selecting filter, pivot, and aggregate reduction operations by taking inspiration from data preparation techniques described for instance in [37]. We observed that these capabilities are required in feature engineering scenarios. Finally, use case studies with a real corpus of analytic tables defined in business verticals will be continued to assess and improve the value of our method.

References

[1] C. S. Jensen, T. B. Pedersen, C. Thomsen, Multidimensional Databases and Data Warehousing, Synthesis lectures on data management, Morgan and Claypool Publishers, 2010.

[2] R. Kimball, M. Ross, The Data Warehouse Toolkit, John Wiley & Sons, 2013.

[3] The virtual data model in sap s/4hana.
URL https://help.sap.com/viewer/6b356c79dea443c4bbeeaf0865e04207/1809.000/en-US/8573b810511948c8a99c0672abc159aa.html

[4] A. Pattanayak, High performance analytics with sap hana virtual models, Journal of Computer and Communications 5 (07) (2017) 1–10.

[5] Data Wrangling Tools & Software | Trifacta.
URL https://www.trifacta.com/

[6] Paxata | Self-Service Data Preparation for Data Analytics.
URL https://www.paxata.com/

[7] SAP Agile Data Preparation and Transformation Solution.
URL https://www.sap.com/products/data-preparation.html

[8] A. Das Sarma, L. Fang, N. Gupta, A. Halevy, H. Lee, F. Wu, R. Xin, C. Yu, Finding related tables, in: Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data, ACM, 2012, pp. 817–828.

[9] M. Yakout, K. Ganjam, K. Chakrabarti, S. Chaudhuri, InfoGather: entity augmentation and attribute discovery by holistic matching with web tables, in: Proceedings of the 2012 international conference on Management of Data - SIGMOD '12, ACM Press, 2012, pp. 97–108. doi:10.1145/2213836.2213848.

[10] D. Deng, R. C. Fernandez, Z. Abedjan, S. Wang, M. Stonebraker, A. K. Elmagarmid, I. F. Ilyas, S. Madden, M. Ouzzani, N. Tang, The Data Civilizer System, in: CIDR, 2017.

[11] J. D. Ullman, H. García-Molina, J. Widom, Database System: The Complete Book, 2nd Edition, Prentice Hall, 2009.

[12] A. Badia, D. Lemire, Functional dependencies with null markers, The Computer Journal 58 (5) (2014) 1160–1168.

[13] J.-N. Mazón, J. Lechtenbörger, J. Trujillo, A survey on summarizability issues in multidimensional modeling, Data & Knowledge Engineering 68 (12) (2009) 1452–1469.

[14] J. Horner, I.-Y. Song, P. P. Chen, An analysis of additivity in OLAP systems, in: Proceedings of the 7th ACM international workshop on Data warehousing and OLAP, ACM, 2004, pp. 83–91.

[15] J. Lee, M. Muehle, N. May, F. Faerber, V. Sikka, H. Plattner, J. Krueger, M. Grund, High-performance transaction processing in SAP HANA, IEEE Data Eng. Bull 36 (2) (2013) 28–33.

[16] SAP HANA Modeling Guide, SAP, AG, 2019.
URL https://help.sap.com/doc/227fc55c4fc44a43b43752d6b127bdf3/2.0.04

[17] SAP HANA smart data integration and SAP HANA smart data quality - SAP help portal.
URL https://help.sap.com/viewer/p/HANA_SMART_DATA_INTEGRATION

[18] R. Brunel, J. Finis, G. Franz, N. May, A. Kemper, T. Neumann, F. Faerber, Supporting hierarchical data in SAP HANA, in: Data Engineering (ICDE), 2015 IEEE 31st International Conference on, IEEE, 2015, pp. 1280–1291.

[19] M. Paradies, C. Kinder, J. Bross, T. Fischer, R. Kasperovics, H. Gildhoff, GraphScript: implementing complex graph algorithms in SAP HANA, in: Proceedings of DBPL 2017, ACM Press, 2017, pp. 1–4.

[20] Z. Bellahsene, A. Bonifati, E. Rahm (Eds.), Schema Matching and Mapping, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. doi:10.1007/978-3-642-16518-4.

[21] F. Nargesian, E. Zhu, K. Q. Pu, R. J. Miller, Table union search on open data, Proceedings of the VLDB Endowment 11 (7) (2018) 813–825.

[22] R. Torlone, Two approaches to the integration of heterogeneous data warehouses, Distributed and Parallel Databases 23 (1) (2008) 69–97.

[23] A. Abelló, J. Samos, F. Saltor, On relationships offering new drill-across possibilities, in: Proceedings of the 5th ACM international workshop on Data Warehousing and OLAP, ACM, 2002, pp. 7–13.

[24] M. Rafanelli, A. Shoshani, Storm: A statistical object representation model, in: International Conference on Scientific and Statistical Database Management, Vol. 420, Springer, 1990, pp. 14–29. doi:10.1007/3-540-52342-1_18.

[25] J. Horner, I.-Y. Song, A taxonomy of inaccurate summaries and their management in OLAP systems, in: International Conference on Conceptual Modeling, Vol. 3716, Springer, 2005, pp. 433–448. doi:10.1007/11568322_28.

[26] C. A. Hurtado, C. Gutierrez, A. O. Mendelzon, Capturing summarizability with integrity constraints in OLAP, ACM Transactions on Database Systems 30 (3) (2005) 854–886.

[27] H.-J. Lenz, A. Shoshani, Summarizability in OLAP and statistical data bases, in: Proceedings of the Ninth International Conference on Scientific and Statistical Database Management (SSDBM '97), 1997, pp. 132–143.

[28] T. Niemi, M. Niinimäki, P. Thanisch, J. Nummenmaa, Detecting summarizability in OLAP, Data & Knowledge Engineering 89 (2014) 1–20.

[29] J.-P. Dittrich, D. Kossmann, A. Kreutz, Bridging the gap between OLAP and SQL, Proceedings of the 31st international conference on Very large data bases (2005) 1031–1042.

[30] W. Lehner, J. Albrecht, H. Wedekind, Normal forms for multidimensional databases, in: Scientific and Statistical Database Management, 1998. Proceedings. Tenth International Conference on, IEEE, 1998, pp. 63–72.

[31] T. Papenbrock, J. Ehrlich, J. Marten, T. Neubert, J.-P. Rudolph, M. Schönberg, J. Zwiener, F. Naumann, Functional dependency discovery: An experimental evaluation of seven algorithms, Proceedings of the VLDB Endowment 8 (10) (2015) 1082–1093.

[32] L. Berti-Équille, H. Harmouch, F. Naumann, N. Novelli, S. Thirumuruganathan, Discovery of genuine functional dependencies from relational data with missing values, Proc. VLDB Endow. 11 (8) (2018) 880–892. doi:10.14778/3204028.3204032.
URL https://doi.org/10.14778/3204028.3204032

[33] J. Liu, J. Li, C. Liu, Y. Chen, Discover dependencies from data—a review, IEEE Transactions on Knowledge and Data Engineering 24 (2) (2010) 251–264.

[34] M. J. Cafarella, A. Halevy, N. Khoussainova, Data integration for the relational web, Proceedings of the VLDB Endowment 2 (1) (2009) 1090–1101.

[35] T. B. Pedersen, C. S. Jensen, C. E. Dyreson, Extending practical pre-aggregation in on-line analytical processing (1999) 12.

[36] Home page of pkdd discovery challenge.
URL https://sorry.vse.cz/~berka/challenge/PAST/index.html

[37] A. Zheng, A. Casari, Feature Engineering for Machine Learning, O'Reilly Media, Inc., 2018.

Appendix A. Proofs for propositions

Proposition 1. *Let $\mathcal{D} = (\mathrm{S}, R, \lambda_R, \bot, \top)$ be an attribute graph. Then, the subset $\mathrm{X}_D \subseteq \mathrm{S}$ of all attributes in S with at least one + labeled in-edge and no* **f** *labeled in-edge is a dimension identifier for all valid dimension tables with attributes S.*

Proof. Literal functional dependencies correspond to functional dependencies where *null* values are considered as constants ($t.A \equiv t'.A$ iff $t.A = t'.A$ or $t.A$ and $t'.A$ are both *null*) and, under this interpretation, the Armstrong's axioms for functional dependencies also hold for literal functional dependencies.

Let $\mathrm{W} = \mathrm{S} - \mathrm{X}_D$ be the subset of S such that for all $\mathrm{A} \in \mathrm{W}$, A has no + in-edges or at least one **f** in-edge. We first prove that the literal functional dependency $\mathrm{S} - \mathrm{W} \mapsto \mathrm{S}$ holds for all valid dimension tables of $\mathcal{D}$. For this, it is sufficient to prove that (1) for all attributes $A_i \in W$, there exists a subset of attributes $\mathrm{S}_i \subseteq \mathrm{S} - \{A_i\}$ where $\mathrm{S}_i \mapsto A_i$. Since attribute graphs (and the corresponding attribute dependencies) are acyclic, we can define a partial order $\leq$ over attributes in W such that each attribute $A_i \in \mathrm{W}$ only depends on attributes $A_k \leq A_i$. Then, from $S_k \subseteq S_i$ and the transitivity of $\mapsto$ and if for all attributes $A_i \in W$, $\mathrm{S} - \{\mathrm{A}_i\} \mapsto A_i$ follows $\mathrm{S} - \mathrm{W} \mapsto \mathrm{S}$. For proving (1), we distinguish two cases which separate the attributes in W into two classes. Let $\mathrm{A}_i \in \mathrm{W}$ and $\mathrm{S}_i = \{\mathrm{A}_k \in \mathrm{S} \mid (\mathrm{A}_k, \mathrm{A}_i) \in \mathsf{R}\}$. By the definition of W, for each A_i there exists at least one $\mathrm{A}_k \in \mathrm{S}_i$ where $\mathsf{R}(\mathrm{A}_k, \mathrm{A}_i) = \mathbf{f}$ (case one) or for all $\mathrm{A}_k \in \mathrm{S}_i : \mathsf{R}(\mathrm{A}_k, \mathrm{A}_i) = \mathbf{1}$ (case two).

–*Case one*: Let $\mathrm{A}_i \in \mathrm{W}$ and $\mathrm{A}_k \in \mathrm{S}_i$ such that $\mathsf{R}(\mathrm{A}_k, \mathrm{A}_i) = \mathbf{f}$. By the definition of attribute graph and label **f** edges, $\mathsf{R}(\mathrm{A}_k, \mathrm{A}_i) = \mathbf{f}$ is equivalent with $\mathrm{A}_k \mapsto \mathrm{A}_i$.

–*Case two*: Let $\mathrm{A}_i \in \mathrm{W}$ and for all $\mathrm{A}_k \in \mathrm{S}_i : \mathsf{R}(\mathrm{A}_k, \mathrm{A}_i) = \mathbf{1}$. For any two tuples $t_1, t_2 \in T$, there exists at least one attribute in $A_k \in \mathrm{S}_i$ where $t_1.A_k$ is not null. Then, if $t_1.\mathrm{S}_i \equiv t_2.\mathrm{S}_i$, there exists at least one attribute $\mathrm{A}_k \in \mathrm{S}_i$ such that $t_1.\mathrm{A}_k = t_2.\mathrm{A}_k$ and since $\mathsf{R}(\mathrm{A}_k, \mathrm{A}_i) = \mathbf{1}$, we get $t_1.\mathrm{A}_i \equiv t_2.\mathrm{A}_i$.

We now prove by contradiction that $\mathrm{S} - \mathrm{W}$ is minimal, *i.e.* there exists no subset $\mathrm{Y} \subset \mathrm{S} - W$ where $\mathrm{Y} \mapsto \mathrm{S}$. Assume that $\mathrm{S} - \mathrm{W} \mapsto \mathrm{S}$ but $\mathrm{S} - \mathrm{W}$ is not minimal. Then there exists at least one attribute $\mathrm{A}_p \in \mathrm{S} - \mathrm{W}$, such that $(\mathrm{S} - \mathrm{W} - \mathrm{A}_p) \mapsto \mathrm{S}$. By the definition of W and $\mathrm{A}_p \notin \mathrm{W}, \mathrm{A}_p$ has at least one '+' in-edge and A_p doesn't have 'f' in-edges. Let $\mathrm{A}_k \in \mathrm{S}$ such that $\mathsf{R}(\mathrm{A}_k, \mathrm{A}_p) = +$. Then we can build a valid dimension table (hierarchy) T where there exists a couple of tuples $t_1, t_2 \in T$ (paths in the hierarchy T) such that $t_1.(\mathrm{S} - \mathrm{A}_p) \equiv t_2.(\mathrm{S} - \mathrm{A}_p)$, and $t_1.\mathrm{A}_k \not\equiv t_2.\mathrm{A}_k$, *i.e.* $\mathrm{S} - \mathrm{A}_p \not\mapsto \mathrm{A}_k$. Then, $\mathrm{S} - \mathrm{W} - \mathrm{A}_p \not\mapsto \mathrm{A}_k$, which is in contradiction with $(\mathrm{S} - \mathrm{W} - \mathrm{A}_p) \mapsto \mathrm{S}$. $\mathrm{S} - \mathrm{W}$ is the minimal set such that $\mathrm{S} - \mathrm{W} \mapsto \mathrm{S}$. □

Proposition 2. *Let $T(\mathrm{S})$ be a fact table defined over a set of dimensions $D_1, \cdots, D_n$ and $\mathrm{K}_1, \cdots, \mathrm{K}_n$ be the dimension identifiers of $D_1 \cap \mathrm{S}, \cdots, D_n \cap \mathrm{S}$ respectively. Then $\mathrm{K} = \mathrm{K}_1 \cup \cdots \cup \mathrm{K}_n$ is a fact identifier of T, and K is a minimal identifier if all dimensions in T are mutually independent.*

Proof. Let $\mathrm{S}_D \subset \mathrm{S}$ the set of dimension attributes in T and $\mathrm{S} - \mathrm{S}_D$ be measure attributes. $K_1, \ldots, K_n$ are the dimension identifiers of $D_1 \cap \mathrm{S}, \ldots, D_n \cap \mathrm{S}$ respectively, and we have $K_1 \mapsto D_1 \cap \mathrm{S}, \ldots, K_n \mapsto D_n \cap \mathrm{S}$. By Armstrong union axiom for LFDs, we get $(K_1 \cup \ldots \cup K_n) \mapsto (D_1 \cap S) \cup \ldots \cup (D_n \cap \mathrm{S}) = \mathrm{S}_D$. By the definition of fact tables, all measure measures depend on the dimension attributes, *i.e.* $\mathrm{S}_D \mapsto \mathrm{S}$ and by transitivity of LFD's, we get $(K_1 \cup \ldots \cup K_n) \mapsto \mathrm{S}$. Two dimension identifiers K_i, K_j, $i \neq j$ are independent, if there exists no attribute $A \in K_j$ such that $K_i \mapsto A$. Then, it is easy to show that if all K_i are minimal and mutually independent, $K_1 \cup \ldots \cup K_n$ is a minimal a fact identifier of T. □

Proposition 3. *Let $T'_0(\mathrm{S}'_0)$ be a merge of table $T_0(\mathrm{S}_0)$ with a target schema augmentation $T(\mathrm{S})$. Let K be the identifier of T_0 and $\mathrm{S}_{new} \subseteq \mathrm{S}'_0 - \mathrm{S}_0$ be the set of dimension attributes added to T_0 in the merge. Then, for all minimal subsets $\mathrm{K}_{new} \subseteq \mathrm{S}_{new}$ where $\mathrm{K}_{new} \mapsto \mathrm{S}_{new}$, $\mathrm{K} \cup \mathrm{K}_{new}$ is an identifier of T'_0.*

Proof. Let $K' = K \cup K_{new}$ where $K_{new} \neq \emptyset$. We prove $K' \mapsto \mathrm{S}'_0$ by contradiction. Assume that K' is not the identifier of T'_0. Then there exist two tuples $t_1, t_2 \in T'_0$ such that $t_1.K' \equiv t_2.K'$ but $t_1.\mathrm{B} \not\equiv t_2.\mathrm{B}, \mathrm{B} \in \mathrm{S}'_0$. We distinguish between two cases:

– $\mathrm{B} \in \mathrm{S}_0$: We know that $K \subset K'$ and $K \mapsto \mathrm{S}_0$. Then by $t_1.K' \equiv t_2.K'$ we have $t_1.K \equiv t_2.K$ and $t_1.\mathrm{B} \equiv t_2.\mathrm{B}$ which contradicts our assumption.

– $\mathrm{B} \in \mathrm{S}_{new}$: We know that $K_{new} \mapsto \mathrm{S}_{new}$. Then by $t_1.K' \equiv t_2.K'$ we have $t_1.K_{new} \equiv t_2.K_{new}$ and $t_1.\mathrm{S}_{new} \equiv t_2.\mathrm{S}_{new}$ which contradicts our assumption.

We conclude $K' \mapsto \mathrm{S}'_0$.

□

Proposition 4. *Let $T_0(\mathrm{S}_0)$ and $T(\mathrm{S})$ be two tables such that T is a schema augmentation to T_0 with common attributes Y. Let K be an identifier of T. The result of a reduction query $\mathcal{Q}(T)$ on attributes $\mathrm{K}' = \mathrm{K} \cap \mathrm{Y}$, is a natural schema complement to T_0.*

Proof. We first show that (A) if $\mathrm{K}' \mapsto \mathrm{S}'$, then $T'(\mathrm{S}')$ is a natural schema complement of $T_0(\mathrm{S}_0)$. By definition, $\mathrm{K}' = K \cap \mathrm{Y}$. Then, from $\mathrm{K} \subseteq \mathrm{Y}$ and $\mathrm{K}' \mapsto \mathrm{S}'$ follows $\mathrm{Y} \mapsto \mathrm{S}'$, *i.e.* $\mathrm{K}' \mapsto \mathrm{S}'$ is a sufficient condition for guaranteeing that $T'(\mathrm{S}')$ is a natural schema complement of $T_0(\mathrm{S}_0)$.

We now show that (B) if $Q(T)$ is a reduction query on K', then $\mathrm{K}' \mapsto \mathrm{S}'$. We distinguish the case where $\mathcal{Q}(T)$ only contains one reduction operation and the case where $\mathcal{Q}(T)$ is a sequence of at least two reduction operations. Let $T'(\mathrm{S}')$ be the result of $\mathcal{Q}(T)$.

– *Case 1: $\mathcal{Q}(T)$ only contains one reduction operation.* We prove for each reduction query by contradiction that $\mathrm{K}' \mapsto \mathrm{S}'$ in T'. Suppose that $\mathrm{K}' \not\mapsto \mathrm{S}'$. Then there exist two tuples $t_1, t_2 \in T'$ such that $t_1.K' \equiv t_2.K'$ but $t_1.\mathrm{S}' \not\equiv t_2.\mathrm{S}'$:

- When $\mathcal{Q}(T)$ is a filter reduction on attributes $K-\mathrm{Y}$, by definition, $\mathrm{S}' = \mathrm{S}$ and $t_1.(K-\mathrm{Y}) \equiv t_2.(K-\mathrm{Y})$. Then, since $t_1.K' \equiv t_2.K'$ (assumption), $K = K' \cup (K-\mathrm{Y})$, $\mathrm{K} \mapsto \mathrm{S}$ and $\mathrm{S}' = \mathrm{S}$, we have $t_1.\mathrm{S}' \equiv t_2.\mathrm{S}'$ which contradicts our assumption. Therefore we get $\mathrm{K}' \mapsto \mathrm{S}'$.
- When $\mathcal{Q}(T)$ is an aggregate reduction, we get $\mathrm{S}' = \mathrm{S} - (K-\mathrm{Y})$ and K' is the set of attributes in the group-by clause $\mathcal{Q}(T)$. Then, by definition of aggregation queries, K' is the identifier (key) of T', a which contradicts our assumption $\mathrm{K}' \not\mapsto \mathrm{S}'$.
- When $\mathcal{Q}(T)$ is a pivot reduction, then attributes in $K-\mathrm{Y}$ are pivoted into new columns in T'. Similar to aggregation, by the definition of pivot queries, K' is the identifier of T' and $\mathrm{K}' \mapsto \mathrm{S}'$ (which contradicts our assumption).

Therefore, when $\mathcal{Q}(T)$ only contains one reduction operation, we have $\mathrm{K}' \mapsto \mathrm{S}'$.

–Case 2: $\mathcal{Q}(T)$ contains multiple reduction operations. Let$\mathcal{Q}(T) = \mathcal{Q}_1(\mathcal{Q}_2(\ldots(\mathcal{Q}_n(T))\ldots))$ be a sequence of of $n > 1$ reduction queries on the sets of attributes $\mathrm{Y}_1, \mathrm{Y}_2, \ldots \mathrm{Y}_n$ where $\mathrm{Y}_i \subset \mathrm{Y}_{i+1}$ and $\mathrm{Y}_1 = \mathrm{Y}$. Let $T'_i(S'_i)$ denote the result of reduction query $\mathcal{Q}_i(T'_{i+1})$ on attributes $\mathrm{K}'_i = \mathrm{K}'_{i+1} \cap \mathrm{Y}'_i$. Then, $T'(\mathrm{S}') = T_1(\mathrm{S}_1)$ is the result of reduction query $\mathcal{Q}(T) = \mathcal{Q}_1(T'_2)$ on attributes $\mathrm{K}'_1 = \mathrm{K}'_2 \cap \mathrm{Y}_1 = \mathrm{K}'_2 \cap \mathrm{Y}$. From Case 1, we then know that $\mathrm{K}'_1 \to \mathrm{S}'_1$.

From (A) and (B) follows that $Q(T)$ is a natural schema complement of T_0. □

Proposition 5. Composition of relationships. *Let $\mathsf{R}_1(T_1, T_2)$ and $\mathsf{R}_2(T_2, T_3)$ be two well-formed relationships between tables T_1, T_2 and T_3 with respective common attributes Y_1 and Y_2. If $\mathrm{Y}_3 = \mathrm{Y}_1 \cap \mathrm{Y}_2 \neq \emptyset$, then there exists a well-formed relationship $\mathsf{R}_3(T_1, T_3)$ that is a* composition *of $\mathsf{R}_1(T_1, T_2)$ and $\mathsf{R}_2(T_2, T_3)$ with common attributes Y_3.*

Proof. We prove that there exists a relationship between T_1 and T_3 and the relationship is well-formed. We distinguish four cases by the types of relationships between $\mathsf{R}_1(T_1, T_2)$ and $\mathsf{R}_2(T_2, T_3)$ with common attributes Y_1 and Y_2.

– Case 1: $\mathsf{R}_1(T_1, T_2)$ and $\mathsf{R}_2(T_2, T_3)$ are both join relationships. By definition, for all attributes $\mathrm{A} \in \mathrm{Y}_3 = \mathrm{Y}_1 \cap \mathrm{Y}_3$, $T_1.\mathrm{A} = T_2.\mathrm{A}$ and $T_2.\mathrm{A} = T_3.\mathrm{A}$. Then by transitivity of equality, $T_1.\mathrm{A} = T_3.\mathrm{A}$, *i.e.*, there $\mathsf{R}_3(T_1, T_3)$ is a join relationship with common attributes Y_3.

– Case 2: $\mathsf{R}_1(T_1, T_2)$ is a join relationship and $\mathsf{R}_2(T_2, T_3)$ is an attribute mapping relationship. We first consider the case where all attributes $\mathrm{A} \in \mathrm{Y}_3 = \mathrm{Y}_1 \cap \mathrm{Y}_3$, $T_1.\mathrm{A} = T_2.\mathrm{A}$ and $T_2.\mathrm{A} \twoheadrightarrow T_3.\mathrm{A}$. By definition, $T_2.\mathrm{A} \twoheadrightarrow T_3.\mathrm{A}$ indicates that there exists a query $\mathcal{Q}_{23}$ and a set of tables such that $T_3 = \mathcal{Q}_{23}(T_2, T'_1, \cdots, T'_n)$ and $\forall y \in dom(\mathrm{A})$:

$$\begin{aligned} &\sigma_{\mathrm{A}=y}(\mathcal{Q}_{23}(T_2, T'_1, \cdots, T'_n)) \\ &= \sigma_{\mathrm{A}=y}(\mathcal{Q}_{23}(\sigma_{\mathrm{A}=y}(T_2), T'_1, \cdots, T'_n)) \end{aligned} \tag{A.1}$$

We prove that $T_1.\mathrm{A} \twoheadrightarrow T_3.\mathrm{A}$, *i.e.*, there exists a query $\mathcal{Q}_{13}$ and a set of tables such that $T_3 = \mathcal{Q}_{13}(T_1, T''_1, \cdots, T''_m))$ and $\forall y \in dom(\mathrm{A})$:

$$\begin{aligned} &\sigma_{\mathrm{A}=y}(\mathcal{Q}_{13}(T_1, T''_1, \cdots, T''_m)) \\ &= \sigma_{\mathrm{A}=y}(\mathcal{Q}_{13}(\sigma_{\mathrm{A}=x}(T_1), T''_1, \cdots, T''_m)) \end{aligned} \tag{A.2}$$

Since $\mathsf{R}_1(T_1, T_2)$ is a join relationship, there exists a table T' such that $T_2 = \pi_{\mathrm{S}_2}(T_1 \bowtie T')$. Then, by replacing T_2 in Equation A.1 and by $T_1.\mathrm{A} = T_2.\mathrm{A}$, we get:

$$\begin{aligned} &\sigma_{\mathrm{A}=y}(\mathcal{Q}_{23}(\pi_{\mathrm{S}_2}(T_1 \bowtie T'), T'_1, \cdots, T'_n)) \\ &= \sigma_{\mathrm{A}=y}(\mathcal{Q}_{23}(\sigma_{\mathrm{A}=y}(\pi_{\mathrm{S}_2}(T_1 \bowtie T')), T'_1, \cdots, T'_n)) \end{aligned} \tag{A.3}$$

Assuming $\mathcal{Q}_{13} = \mathcal{Q}_{23}(\pi_{\mathrm{S}_2}(T_1 \bowtie T'), T'_1, \cdots, T'_n)$, by replacement in Equation A.3, we get:

$$\begin{aligned} &\sigma_{\mathrm{A}=y}(\mathcal{Q}_{13}(T_1, T', T'_1, \cdots, T'_n)) \\ &= \sigma_{\mathrm{A}=y}(\mathcal{Q}_{23}(\sigma_{\mathrm{A}=y}(\pi_{\mathrm{S}_2}(T_1 \bowtie T')), T'_1, \cdots, T'_n)) \end{aligned} \tag{A.4}$$

Finally, by pushing all selections into the joins of the right side of Equation A.3, we obtain:

$$\begin{aligned} &\sigma_{\mathrm{A}=y}(\mathcal{Q}_{23}(\pi_{\mathrm{S}_2}(\sigma_{\mathrm{A}=y}(T_1) \bowtie T'), T'_1, \cdots, T'_n)) \\ &= \sigma_{\mathrm{A}=y}(\mathcal{Q}_{13}(\sigma_{\mathrm{A}=y}(T_1), T', T'_1, \cdots, T'_n)) \end{aligned} \tag{A.5}$$

which corresponds to $T_1.\mathrm{A} \twoheadrightarrow T_3.\mathrm{A}$.

We follow the same reasoning for proving that $T_1.\mathrm{A} = T_2.\mathrm{A}$ implies $T_3.\mathrm{A} \twoheadrightarrow T_2.\mathrm{A}$, $T_3.\mathrm{A} \twoheadrightarrow T_1.\mathrm{A}$.

– Case 3: $\mathsf{R}_1(T_1, T_2)$ is an attribute mapping relationship and $\mathsf{R}_2(T_2, T_3)$ is a join relationship. We can follow a similar rewriting process as in Case 2 to prove for all attributes $\mathrm{A} \in \mathrm{Y}_3 = \mathrm{Y}_1 \cap \mathrm{Y}_3$ that if $T_2.\mathrm{A} = T_3.\mathrm{A}$, then $T_1.\mathrm{A} \twoheadrightarrow T_2.\mathrm{A}$ implies $T_1.\mathrm{A} \twoheadrightarrow T_3.\mathrm{A}$ and $T_2.\mathrm{A} \twoheadrightarrow T_1.\mathrm{A}$ implies $T_3.\mathrm{A} \twoheadrightarrow T_1.\mathrm{A}$.

– Case 4. $\mathsf{R}_1(T_1, T_2)$, $\mathsf{R}_2(T_2, T_3)$ are both attribute mapping relationships. We consider the case where for all attributes $\mathrm{A} \in \mathrm{Y}_3 = \mathrm{Y}_1 \cap \mathrm{Y}_2$, $T_1.\mathrm{A} \twoheadrightarrow T_2.\mathrm{A}$ and $T_2.\mathrm{A} \twoheadrightarrow T_3.\mathrm{A}$. Then there exist two queries $\mathcal{Q}_{12}$ and $\mathcal{Q}_{23}$ over a set of tables such that $T_2 = \mathcal{Q}_{12}(T_2, \cdots, T'_m)$, $T_3 = \mathcal{Q}_{23}(T_2, T'_1, \cdots, T'_n)$ and $\forall y \in dom(\mathrm{A})$:

$$\begin{aligned} &\sigma_{\mathrm{A}=y}(\mathcal{Q}_{12}(T_1, \cdots, T'_n)) \\ &= \sigma_{\mathrm{A}=y}(\mathcal{Q}_{12}(\sigma_{\mathrm{A}=y}(T_1), \cdots, T'_n)) \end{aligned} \tag{A.6}$$

and

$$\begin{aligned} &\sigma_{\mathrm{A}=y}(\mathcal{Q}_{23}(T_2, \cdots, T'_m)) \\ &= \sigma_{\mathrm{A}=y}(\mathcal{Q}_{23}(\sigma_{A=y}(T_2), \cdots, T'_m)) \end{aligned} \tag{A.7}$$

We prove that $T_1.\mathrm{A} \twoheadrightarrow T_3.\mathrm{A}$, *i.e.*, there exists a query $\mathcal{Q}_{13}$ and a set of tables $T_3 = \mathcal{Q}_{13}(T_1, \cdots, T'_k)$ such that $\forall y \in dom(\mathrm{A})$,

$$\begin{aligned} &\sigma_{\mathrm{A}=y}(\mathcal{Q}_{13}(T_1, \cdots, T'_k)) \\ &= \sigma_{\mathrm{A}=y}(\mathcal{Q}_{13}(\sigma_{\mathrm{A}=y}(T_1), \cdots, T'_k)) \end{aligned} \tag{A.8}$$

By replacing T_2 by $\mathcal{Q}_{12}$ in Equation A.7, we obtain:

$$\begin{aligned} &\sigma_{\mathrm{A}=y}(\mathcal{Q}_{23}(T_2, \cdots, T'_m)) \\ &= \sigma_{\mathrm{A}=y}(\mathcal{Q}_{23}(\mathcal{Q}_{12}(T_1, \cdots, T'_n), \cdots, T'_m)) \end{aligned} \tag{A.9}$$

By pushing the selection $\sigma_{\mathrm{A}=y}$, we obtain:

$$\begin{aligned}&\sigma_{\mathrm{A}=y}(\mathcal{Q}_{23}(\mathcal{Q}_{12}(T_1,\cdots,T'_n),\cdots,T'_m))\\&=\mathcal{Q}_{23}(\sigma_{\mathrm{A}=y}(\mathcal{Q}_{12}(T_1,\cdots,T'_n),\cdots,T'_m))\end{aligned}\tag{A.10}$$

Then, by applying Equation A.6, we obtain:

$$\begin{aligned}&\sigma_{\mathrm{A}=y}(\mathcal{Q}_{23}(\mathcal{Q}_{12}(T_1,\cdots,T'_n),\cdots,T'_m))\\&=\mathcal{Q}_{23}(\sigma_{\mathrm{A}=y}(\mathcal{Q}_{12}(\sigma_{\mathrm{A}=y}(T_1),\cdots,T'_n)),\cdots,T'_m)\end{aligned}\tag{A.11}$$

By pulling the first selection $\sigma_{\mathrm{A}=y}$ out from $\mathcal{Q}_{23}$, we obtain:

$$\begin{aligned}&\sigma_{\mathrm{A}=y}(\mathcal{Q}_{23}(\mathcal{Q}_{12}(T_1,\cdots,T'_n),\cdots,T'_m))\\&=\sigma_{\mathrm{A}=y}(\mathcal{Q}_{23}(\mathcal{Q}_{12}(\sigma_{\mathrm{A}=y}(T_1),\cdots,T'_n)),\cdots,T'_m)\end{aligned}\tag{A.12}$$

Now let $\mathcal{Q}_{13}(T_1,\cdots,T'_n,\cdots,T'_m) = \mathcal{Q}_{23}(\mathcal{Q}_{12}(T_1,\cdots,T'_n)),\cdots,T'_m)$. Then, we obtain from Equation A.12:

$$\begin{aligned}&\sigma_{\mathrm{A}=y}(\mathcal{Q}_{13}(T_1,\cdots,T'_n,\cdots,T'_m))\\&=\sigma_{\mathrm{A}=y}(\mathcal{Q}_{13}(\sigma_{\mathrm{A}=y}(T_1),\cdots,T'_n,\cdots,T'_m))\end{aligned}\tag{A.13}$$

Therefore, we have for all attributes in $\mathrm{Y}_3 = \mathrm{Y}_1 \cap \mathrm{Y}_2$, $T_1.\mathrm{A} \twoheadrightarrow T_3.\mathrm{A}$, *i.e.* $\mathsf{R}-3(T_1,T_3)$ is an attribute mapping relationship with common attributes Y_3.

□

Proposition 6. Fusion of relationships. *Let* $\mathsf{R}_1(T_1,T_2)$ *and* $\mathsf{R}_2(T_1,T_2)$ *be two well-formed relationships between two tables* T_1 *and* T_2 *with respective common attributes* Y_1 *and* Y_2. *If* $\forall A \in \mathrm{Y}_1 \cap \mathrm{Y}_2$, $\mu_{\mathsf{R}_1}(A) = \mu_{\mathsf{R}_2}(A)$ *then there exists a well-formed relationship* $\mathsf{R}_3(T_1,T_2)$ *that is a* fusion *of* $\mathsf{R}_1(T_1,T_2)$ *and* $\mathsf{R}_2(T_1,T_2)$ *with common attributes* $\mathrm{Y}_3 = \mathrm{Y}_1 \cup \mathrm{Y}_2$.

Proof. We can apply a similar case study as in the proof of Proposition 5. Assume that $\mathsf{R}_1(T_1,T_2)$ and $R_2(T_1,T_2)$ are two natural mapping relationships with common attributes Y_1 and Y_2, *i.e.* $\forall \mathrm{A} \in \mathrm{Y}_1 \cap \mathrm{Y}_2, \mathrm{A} = \nu_{\mathsf{R}_1}(\mathrm{A}) = \nu_{\mathsf{R}_2}(\mathrm{A})$. We prove that there exists a well-formed relationship R_3 between T_1 and T_2 with common attributes $\mathrm{Y}_3 = \mathrm{Y}_1 \cup \mathrm{Y}_2$. We distinguish three cases by the types of both relationships $\mathsf{R}_1(T_1,T_2)$ and $\mathsf{R}_2(T_1,T_2)$.

– *Case 1:* $\mathsf{R}_1(T_1,T_2)$ *and* $\mathsf{R}_2(T_1,T_2)$ *are both join relationships.* By definition, for all attributes $\mathrm{A} \in \mathrm{Y}_3 = \mathrm{Y}_1 \cup \mathrm{Y}_2$, $T_1.\mathrm{A} = T_2.\mathrm{A}$, *i.e.* there exists a join relationship $\mathsf{R}_3(T_1,T_3)$ with common attributes Y_3.

– *Case 2.* $\mathsf{R}_1(T_1,T_2)$, $\mathsf{R}_2(T_1,T_2)$ *are both attribute mapping relationships.* By definition, for all attributes $\mathrm{A} \in \mathrm{Y}_3 = \mathrm{Y}_1 \cup \mathrm{Y}_2$, $T_1.\mathrm{A} \twoheadrightarrow T_2.\mathrm{A}$, *i.e.* there exists an attribute mapping relationship $\mathsf{R}_3(T_1,T_3)$ with common attributes Y_3.

– *Case 3:* $\mathsf{R}_1(T_1,T_2)$ *is a join relationship and* $\mathsf{R}_2(T_1,T_2)$ *is an attribute mapping relationship.* Then, by Case 2, for showing that $\mathsf{R}_3(T_1,T_2)$ is an attribute mapping relationship it is sufficient to show that $\mathsf{R}_1(T_1,T_2)$ is an attribute mapping relationship. We show more generally that any join relationship is also an attribute mapping relationship. By definition, $R_1(T_1,T_2)$ is a join relationship with common attributes Y_1, *i.e.*, for all $\mathrm{A} \in \mathrm{Y}_1$, $T_1.\mathrm{A} = T_2.\mathrm{A}$ and $T_2 = \mathcal{Q}_{12}(T_1,T_2) = \pi_{S_2}(T_1 \bowtie_Y T_2)$ where S_2 is the schema of T_2 and $\mathrm{Y} \subseteq \mathrm{S}_2$. Then, $\forall y \in dom(\mathrm{A})$:

$$\begin{aligned}&\sigma_{\mathrm{A}=y}(\mathcal{Q}_{12}(T_1,T_2))\\&=\sigma_{\mathrm{A}=y}(\pi_{S_2}(T_1 \bowtie_{\mathrm{Y}_1} T_2))\\&=\sigma_{\mathrm{A}=y}(\pi_{S_2}(\sigma_{\mathrm{A}=y}(T_1) \bowtie_{\mathrm{Y}_1} T_2))\\&=\sigma_{\mathrm{A}=y}(\mathcal{Q}_{12}(\sigma_{\mathrm{A}=y}(T_1),T_2))\end{aligned}\tag{A.14}$$

Therefore, R is an attribute mapping relationship. □

Proposition 7. *Let* $T'_0(\mathrm{S}'_0)$ *be a merge of table* $T_0(\mathrm{S}_0)$ *with a target schema augmentation* $T(\mathrm{S})$. *Then the following aggregable properties hold for all aggregable attributes* $A \in \mathrm{S}'_0$:

1. *If* $\mathbf{agg}_A(\mathrm{F},\mathrm{V}_0)$ *holds in* T_0 *and* $A \in \mathrm{S}'_0 \cap \mathrm{S}_0$ *then* $\mathbf{agg}_A(\mathrm{F},\mathrm{V}_0)$ *holds in* T'_0.
2. *If* $\mathbf{agg}_A(\mathrm{F},\mathrm{V})$ *holds in* T *and* $A \in \mathrm{S}'_0 - \mathrm{S}_0$ *, then* $\mathbf{agg}_A(\mathrm{F},\mathrm{V} \cap \mathrm{S}'_0)$ *holds in* T'_0.

Proof. Let $agg_{\mathrm{A}}(F,V'_0)$ be the aggregable property of A in T'_0, *i.e.* $V'_0 \subseteq S'_0$ is the maximal set of dimension attributes such that A can be aggregated along V'_0 with function F in T'_0.

1. Let $\mathrm{U}_0 \subseteq S_0$ be a minimal subset of attributes where $\mathrm{U}_0 \mapsto A$ in T_0 and $\mathrm{V}_0 \subseteq \mathrm{U}_0$. Since $\mathrm{V}_0 \subseteq \mathrm{S}_0 \subseteq \mathrm{S}'_0$ and $\mathrm{U}_0 \subseteq \mathrm{S}'_0$ we conclude that $\mathrm{U}_0 \mapsto \mathrm{A}$ in T'_0 and $\mathrm{V}_0 \subseteq \mathrm{V}'_0$. Since A is not an attribute in T, aggregation along $\mathrm{S}-\mathrm{V}$ is not meaningful and we obtain $\mathrm{V} = \mathrm{V}'_0$
2. Since T is a natural schema complement of T_0, $\mathrm{S}_0 \cap \mathrm{S} \mapsto \mathrm{S}$. Let $\mathrm{U} \subseteq \mathrm{S}_0 \cap \mathrm{S}$ be a minimal subset of attributes where $\mathrm{U} \mapsto \mathrm{A}$ in T and $\mathrm{V} \subseteq \mathrm{U}$. Since $\mathrm{U} \mapsto \mathrm{A}$ and $\mathrm{V} \cap \mathrm{S}'_0 \subseteq \mathrm{V} \subseteq \mathrm{U}$, we conclude that $\mathrm{V} \cap \mathrm{S}'_0 \subseteq \mathrm{V}'_0$. Since A is not an attribute in T_0, aggregation along $S_0 - V$ is not meaningful and we obtain $\mathrm{V} \cap \mathrm{S}'_0 = \mathrm{V}'_0$.

□

Proposition 8. *Let* $T(\mathrm{S})$ *be a non-ambiguous analytic table w.r.t. a dimension* D *of schema* S_D. *Let* $T'(\mathrm{S}') = \mathcal{Q}(T)$ *be a reduction of* T. *Let* $\mathrm{X} = \mathrm{S} \cap \mathrm{S}_D$ *and* $\mathrm{X}' = \mathrm{S}' \cap \mathrm{S}_D$. *If* $\mathrm{X}' \mapsto \mathrm{X}$ *in* D, *then* T' *is* non-ambiguous *w.r.t.* D.

Proof. Let $\mathrm{X}' = \mathrm{S}' \cap \mathrm{S}_D$ and $\mathrm{X}'^* = \{\mathrm{A}_j \in \mathrm{S}_D \mid \exists \mathrm{A}_i \in \mathrm{X}', \mathrm{A}_i \preccurlyeq^* \mathrm{A}_j\}$. To prove that T' is not ambiguous w.r.t. D, we must show that $\mathrm{X}' \mapsto \mathrm{X}'^*$ holds in $T' \bowtie_{\mathrm{X}'} D$ (Definition 14).

We first show that $\mathrm{X}' \mapsto \mathrm{X}'^*$ holds in $T \bowtie_{\mathrm{X}} D$. Let $\mathrm{X}_D = \mathrm{S} \cap \mathrm{S}_D$ and $\mathrm{X}^* = \{\mathrm{A}_j \in \mathrm{S}_D \mid \exists \mathrm{A}_i \in \mathrm{X}, \mathrm{A}_i \preccurlyeq^* \mathrm{A}_j\}$. Since T is non-ambiguous, we know that (a) $\mathrm{X} \mapsto \mathrm{X}^*$ holds in $T \bowtie_{\mathrm{X}} D$. Since $\mathrm{X}' \mapsto \mathrm{X}$ holds in D, $\mathrm{X}' \subseteq \mathrm{X}$ and $\pi_{\mathrm{X}}(T \bowtie_{\mathrm{X}} D) \subseteq \pi_{\mathrm{X}}(D)$, we also obtain (b) $\mathrm{X}' \mapsto \mathrm{X}$ holds in $T \bowtie_{\mathrm{X}} D$. By transitivity of $\mapsto$ and (a) and (b), we obtain (c) $\mathrm{X}' \mapsto \mathrm{X}^*$

holds in $T \bowtie_{\mathrm{X}} D$, and since $\mathrm{X}'^* \subseteq \mathrm{X}^*$, (d) $\mathrm{X}' \mapsto \mathrm{X}'^*$ holds in $T \bowtie_{\mathrm{X}} D$.

We now show that $\pi_{\mathrm{S}_D}(T' \bowtie_{\mathrm{X}'} D) \subseteq \pi_{\mathrm{S}_D}(T \bowtie_{\mathrm{X}} D)$. Since $\mathrm{X}' \mapsto \mathrm{X}$ holds in D and $\mathrm{X}' \subseteq \mathrm{X} \subseteq \mathrm{S}_D$, we know that $\pi_{\mathrm{S}_D}(\pi_{X'}(T) \bowtie_{\mathrm{X}'} D) = \pi_{\mathrm{S}_D}(T \bowtie_{\mathrm{X}} D)$ and since $\pi_{X'}(T') \subseteq \pi_{X'}(T)$, we obtain (e) $\pi_{\mathrm{S}_D}(T' \bowtie_{\mathrm{X}'} D) \subseteq \pi_{\mathrm{S}_D}(T \bowtie_{\mathrm{X}} D)$.

Then, from (d) and (e), we can conclude that $\mathrm{X}' \mapsto \mathrm{X}'^*$ holds in $T' \bowtie_{\mathrm{X}'} D$ □

Proposition 9. *Let $T_0(\mathrm{S}_0)$ and $T(\mathrm{S})$ be two analytic tables with a relationship* R *and common dimension attributes* Y*. Let $T'_0(\mathrm{S}'_0)$ be the merge of T_0 and T and $T^{miss} = T^{cand} - T^{ct}$ be the set of all tuples in T lost in the merge with T_0. We can define a completion table $T^{com}(S'_0)$ for all dimensions D_i of schema S_{D_i} in T_0, $0 \leq i \leq n$, where $(\mathrm{S}_{D_i} \cap \mathrm{S}) \subset (\mathrm{S}_{D_i} \cap \mathrm{S}_0)$:*

$$T^{com} = \Pi_{\mathrm{S}'_0}(T^{miss} \ltimes_{\mathrm{S}_{D_0} \cap \mathrm{Y}} D_0 \ltimes \cdots \ltimes_{\mathrm{S}_{D_n} \cap \mathrm{Y}} D_n) \quad (1)$$

If $\mathrm{Y} \mapsto \mathrm{S}_0$ *(*non-ambiguous merge condition*), then*

$$\mathcal{Q}_m(T_0, T) = T'_0 \cup T^{com} \quad (2)$$

is a complete merge *of T_0 and T with respect to T.*

Proof. Let T^{ct} and T^{cand} be the completion table and candidate completion table of T and T'_0 as defined in Definition 15. We use Definition 16 to prove that $T'_0 = \mathcal{Q}_m(T_0, T)$ is a complete merge w.r.t. to T:

$$T^{ct} = T \ltimes_{\mathrm{Y}} \mathcal{Q}_m(T_0, T) = T \ltimes_{\mathrm{Y}^{top}} \mathcal{Q}_m(T_0, T) = T^{cand}$$

We can replace $\mathcal{Q}_m(T_0, T)$ with the right-hand side of Equation (2) and then replace T^{com} with the right-hand side of Equation (2) to obtain:

$$\begin{aligned} T^{ct} &= T \ltimes_{\mathrm{Y}} (T'_0 \cup T^{com}) \\ &= T^{ct} \cup (T \ltimes_{\mathrm{Y}} T^{com}) \end{aligned} \quad (A.15)$$

Equation A.15 uses T^{com} which extends T_{miss} with all dimension attributes A in S'_0 that don't exist in the augmentation schema S, *i.e.*, $\mathrm{A} \in \mathrm{S}'_0 - \mathrm{S}$. Then, since all these attributes are not common attributes of T_0 and T, *i.e.*, $\mathrm{Y} \cap (\mathrm{S}'_0 - \mathrm{S}) = \emptyset$, their values do not change the result of the natural semi-join $T \ltimes_{\mathrm{Y}} T^{com}$. Therefore, since $\mathrm{Y} \subseteq \mathrm{S}'_0$ we can safely remove the projection $\pi_{S'_0}$ and replace T^{com} by T^{miss} in Equation A.15:

$$T^{ct} = T^{ct} \cup (T \ltimes_{\mathrm{Y}} T^{miss}) \quad (A.16)$$

When we replace T^{miss} with $T^{miss} = T^{cand} - T^{ct}$, we obtain:

$$\begin{aligned} T^{ct} &= T^{ct} \cup (T \ltimes_{\mathrm{Y}} (T^{cand} - T^{ct})) \\ &= T^{ct} \cup ((T \ltimes_{\mathrm{Y}} T^{cand}) - (T \ltimes_{\mathrm{Y}} T^{ct})) \\ &= (T^{ct} \cup (T \ltimes_{\mathrm{Y}} T^{cand})) - (T^{ct} \cup (T \ltimes_{\mathrm{Y}} T^{ct})) \end{aligned} \quad (A.17)$$

By Definition 15, it is easy to show that $T^{ct} \subseteq T^{cand} \subseteq T$. We then obtain $T \ltimes_{\mathrm{Y}} T^{ct} = T^{ct}$ and $T \ltimes_{\mathrm{Y}} T^{cand} = T^{cand}$ and continue with Equation A.17:

$$\begin{aligned} T^{ct} &= (T^{ct} \cup T^{cand}) \cup (T^{ct} - T^{ct}) \\ &= T^{cand} \cup \emptyset = T^{cand} \end{aligned} \quad (A.18)$$

By $T^{ct} = T^{cand}$, we can conclude that $\mathcal{Q}_m$ is a complete merge with respect to T. □

Lemma 1 (Composition of natural schema complements)**.** *Let $T(S)$ be a natural schema complement to $T_0(S_0)$ with respect to relationship $\mathsf{R}_0(T_0, T)$ on attributes Y_0, and $T_1(S_1)$ be a natural schema complement to T with respect to relationship $\mathsf{R}_1(T, T_1)$ on attributes Y_1. Then T_1 is also a* natural schema complement *to the natural merge of T_0 and T,* i.e.*, $T_0 \ltimes_{\mathrm{Y}_0} T$.*

Proof. Let $T'_0(S'_0)$ be the natural merge of T_0 and T, $T'_0 = T_0 \ltimes_{\mathrm{Y}_0} T$.

There exists an attribute mapping relationship $\mathsf{R}'(T'_0, T)$ on common attributes $\mathrm{Y}' = \mathrm{S}'_0 \cap \mathrm{S}$, and $\mathrm{Y}_1 \subseteq \mathrm{Y}'$. And there exists a well-formed relationship $\mathsf{R}'_1(T'_0, T_1)$ on attributes $\mathrm{Y}_1 \cap \mathrm{S}'_0$ by the composition of relationships $\mathsf{R}'(T'_0, T)$ and $\mathsf{R}_1(T, T_1)$. Because T_1 is a natural schema complement to T, we have $\mathrm{Y}_1 \mapsto \mathrm{S}_1$ and $\mathrm{Y}' \mapsto \mathrm{S}_1$. Therefore, T_1 is a natural schema complement to T'_0. □

Lemma 2 (Composition of schema complement and schema augmentation)**.** *Let $T(\mathrm{S})$ be a natural schema complement to $T_0(\mathrm{S}_0)$ with respect to relationship $\mathsf{R}_0(T_0, T)$ on attributes Y_0, and $T_1(\mathrm{S}_1)$ be a schema augmentation to T with respect to relationship $\mathsf{R}_1(T, T_1)$ on attributes Y_1. Then T_1 is a* schema augmentation *to the natural merge of T_0 and T,* i.e.*, $T_0 \ltimes_{\mathrm{Y}_0} T$.*

Proof. Let $T'_0(S'_0)$ be the natural merge of T_0 and T, $T'_0 = T_0 \ltimes_{\mathrm{Y}_0} T$.

There exists an attribute mapping relationship $\mathsf{R}'(T'_0, T)$ on common attributes $\mathrm{Y}' = \mathrm{S}'_0 \cap \mathrm{S}$, and $\mathrm{Y}_1 \subseteq \mathrm{Y}'$. By the composition of relationships $\mathsf{R}'(T'_0, T)$ and $\mathsf{R}_1(T, T_1)$, there exists a well-formed relationship $\mathsf{R}'_1(T'_0, T_1)$ on attributes $\mathrm{Y}_1 \cap \mathrm{S}'_0$. Therefore, T_1 is a schema augmentation to T'_0. □

Proposition 10. *Let query $\mathcal{Q}$ be the result of a MSA(T_0, T, path, true, true) call with a start table $T_0(\mathrm{S}_0)$, a target schema augmentation table $T(\mathrm{S})$ and a path of relationships: path $= \mathsf{R}_1(T_0, T_1), \cdots, \mathsf{R}_n(T_{n-1}, \mathcal{Q}_a), n >= 1$. If $\mathsf{R}_n(T_{n-1}, \mathcal{Q}_a)$ maps to an augmentation schema complement edge (CT = 'AUG'), then $\mathcal{Q}$ computes a augmented merge of T_0 with $\mathcal{Q}_a$ (without ambiguous values). Otherwise, $\mathsf{R}_n(T_{n-1}, \mathcal{Q}_a)$ maps to a natural schema complement edge (CT = 'NAT') and $\mathcal{Q}$ computes a natural merge T_0 with $\mathcal{Q}_a$.*

Proof. We can ignore steps 1 and 2 of MSA. Step 3 returns merge query $\mathcal{Q} = \pi_{\mathrm{S}'_0}(\mathcal{Q}_{path} \ltimes_{\mathrm{Y}_n} \mathcal{Q}_a)$, where S'_0 is augmented S with attributes from $\mathcal{Q}_a$, $\mathcal{Q}_{path} = T_0 \ltimes_{\mathrm{Y}_1} T_1$

$\cdots \rtimes\!\!\ltimes_{\mathrm{Y}_{n-1}} T_{n-1}$ and Y_i is the common attributes in relationship R_i. By Algorithm 3, every relationship in $\mathsf{R}_1(T_0, T_1), \cdots, \mathsf{R}_{n-1}(T_{n-2}, T_{n-1})$ maps a $\mathcal{SC}$ edge with $CT =$ *'NAT'* in $\mathcal{SC}$. By Lemma 1, $\mathcal{Q}_{path}$ is a natural merge of T_0 with its natural schema complements $T_1, \cdots, T_{n-1}$. Also by Lemma 1, if $\mathsf{R}_n(T_{n-1}, T)$ maps a $\mathcal{SC}$ edge with $CT =$ *'NAT'* in $\mathcal{SC}$, then $\mathcal{Q}_a$ is a natural schema complement to the result of $\mathcal{Q}_{path}$ and $\mathcal{Q}$ computes the natural merge of $\mathcal{Q}_{path}$ and T. Finally, by the same lemma, since $\mathcal{Q}_{path}$ is a natural merge of T_0, $\mathcal{Q}$ is a natural merge of T_0 and $\mathcal{Q}_a$. By Lemma 2, if $\mathsf{R}_a(T_{a-1}, T)$ maps to an augmentation $\mathcal{SC}$ edge with $CT =$ *'AUG'*, then $\mathcal{Q}_a$ is a schema augmentation to the result of $\mathcal{Q}_{path}$ and $\mathcal{Q}$ computes the augmented merge of $\mathcal{Q}_{path}$ and $\mathcal{Q}_a$ (and an augmented merge of T_0 and $\mathcal{Q}_a$). □