Semantic Services for Assisting Users to Augment Data in the Context of Analytic Data Sources

Rutian Liu

École Doctoral Informatique, Télécommunications et Électronique – EDITE (ED130)

DOCTORAL THESIS

Discipline: Informatique

Semantic Services for Assisting Users to Augment Data in the Context of Analytic Data Sources

Rutian Liu

Angela Bonifati	Professeur	Université Lyon 1	*Reviewer*
Sofian Maabout	Maître de Conférences (HDR)	Université de Bordeaux	*Reviewer*
Jérôme Darmont	Professeur	Université Lyon 2	*Examiner*
Marie-Jeanne Lesot	Maître de Conférences (HDR)	Sorbonne Université	*Examiner*
Bernd Amann	Professeur	Sorbonne Université	*Examiner*
Stéphane Gançarski	Maître de Conférences (HDR)	Sorbonne Université	*Examiner*

June 24, 2020

Rutian Liu
Semantic Services for Assisting Users to Augment Data in the Context of Analytic Data Sources, June 24, 2020
Reviewers: Angela Bonifati and Sofian Maabout
Supervisors: Bernd Amann (Sorbonne Université), Stéphane Gançarski (Sorbonne Université) and Eric Simon (SAP France)

Sorbonne Université
CNRS – LIP6 UMR 7606
4, place Jussieu
75252 Paris

Contents

1

Introduction

Contents

In this chapter, we introduce the notion of analytic datasets, their role and evolution within the digital evolution of companies and organizations (Section 1.1). Then, in Section 1.2, we show four main challenges that arise when business users want to customize analytic datasets to their needs. Finally, we list the major research contributions of this thesis in Section 1.3.

1.1 The Role and Evolution of Analytics

Business Intelligence (BI) comprises the technologies to produce, process and analyse business information in enterprises. BI analysts heavily rely on trusted and well documented datasets, called *analytic datasets* (or sometimes analytics), which comprise multidimensional *facts* that hold *measures* and refer to one or more hierarchical *dimensions* [1]. BI platform vendors provide sophisticated tools that use analytic datasets for managed data reporting, interactive analysis, KPI monitoring, prediction and visualization. More recently, self-service BI tools emerged to enable business users and data scientists to create customized analytics and powerful visualizations (see e.g., [2]–[5]).

Traditionally, analytic datasets are created by the IT department in the form of data warehouses and data marts [6], or by enterprise application software vendors in the form of predefined and customizable analytic models. Analytic datasets are built from the operational data hold by the transactional databases of an enterprise or organization. More than a decade ago, most of the analytic data was physically stored apart from operational datasets into decision support systems. Complex data extraction, transformation, and load (ETL) processes were used to periodically read operational data and update the corresponding analytic data in the decision support systems. In the recent years, there has been a trend to create analytic data, next to the operational data on which they depend, in the form of views with the goal of providing real-time analytic capabilities. 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) [7], [8]. 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.

Creating analytic datasets is a complex, tedious and time intensive activity which involves design and implementation tasks at multiple levels of the architecture of an information system. First, analytic data models must be designed according to the needs of business users and analysts in different application domains. There, the proper definition of dimensions (aka master data) and measures is essential. Then analytic data must be created from operational data, which involves the definition of possibly complex data integration processes. A key aspect is the governance of the quality of the analytic data because they must form a trusted foundation on which decision-making processes can rely. This involves the creation of data cleaning processes (e.g., duplicate record elimination, enforcement of business rules). Finally, the definition of analytic datasets may be layered, where one layer adds specific business logic to the previous one, to obtain very customized analytics.

The slow process of creation of analytics has been confronted to a profound evolution of enterprises towards a digital transformation, whereby the ability to perform fine-grain analysis of business data and take a prompt action according to perceived changes, is becoming a key to business success. This digital transformation impacts how analytic data are created. Firstly, business users need to combine analytic data obtained from operational systems with data coming from external sources, including Internet of Things (IoT) data or market signal data. Business users also require access to detailed data to refine their analysis. Secondly, business users need to continuously adjust the definition of the analytics they use to monitor their business

so that they can rapidly adapt to changes. This need is aggravated by the increasing number of users who need to customize the analytics they are working with. For instance, operational BI gives every "operational worker" (e.g., clerc, maintenance supervisor, etc) insights needed to make better operational decisions (including access to detailed data on-demand). In addition, data scientists are empowered to conduct data analysis projects which require the preparation of datasets that match their analysis needs.

IT organizations cannot sustain the pace of the growing needs of business users, analysts, and data scientists in this digital transformation of enterprises and organizations. Recently, agile data preparation and integration tools [9]–[11] have emerged to empower business users and data scientists to easily create their own high-quality analytic datasets from transactional data and existing analytic datasets. These data preparation tools can extract structured information from unstructured data, partially automate data cleaning and transformation operations and perform data integration tasks like record linkage and duplicate elimination. However, another typical requirement of business users or data scientists is to augment the schema of an existing analytic dataset with new attributes from one or more semantically related datasets. These attributes may represent additional details on dimensions or new measures. This kind of schema augmentation is a critical need in many data integration scenarios like data mashup generation and feature engineering. Despite the importance of this task, existing data preparation tools provide a limited support for schema augmentation of analytic dataset. This lack of assistance compels business users to re-define multiple times similar analytic datasets in a possibly inconsistent manner or to depend on their IT department to create customized datasets. This creates an important bottleneck in the IT organization and significantly slows down the production of customized analytic datasets.

Addressing the problem of agile and trusted analytic schema augmentation 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 use of agile data preparation tools considerably increases the number of available analytic datasets with respect to those provided by enterprise software application vendors or those created by the IT organization. Second, many semantic relationships between analytic datasets, which are essential to support schema augmentation, can be accurately and automatically extracted from dataset definitions. Indeed, as mentioned before, 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 for example possible to discover the dimensions shared by different analytic datasets. Third, complex data analysis scenarios of analytic datasets are generally supported by rich and carefully designed metadata, such as the correspondence between the table attributes and the hierarchical dimension levels or the units and currencies of measure attributes. These metadata that can also be exploited during an assisted schema augmentation process. Finally, IT organizations and business users invest time to create analytic datasets containing high quality for business process optimization and decision making. Reusing these clean 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.

1.2 Main Challenges

The manual augmentation of an analytic dataset is often a cumbersome and error-prone process, raising multiple challenges. Imagine a simple use case with two analytic datasets represented by two fact tables, SALES and DEM(ographics), and three dimension tables $SALESORG$, $TIME$ and $REGION$.

An example analytic dataset is shown in Table 1.1. Dimension table names are in italic font to distinguish them from fact tables. Underlined attributes are tuple identifiers (primary key attributes) and dimension tables represent hierarchical dimensions. The fact table SALES contains three attributes STORE_ID, CITY and COUNTRY from dimension $SALESORG$ and attribute YEAR from dimension $TIME$. The fact table DEM contains three attributes CITY, STATE, COUNTRY from dimension $REGION$ and attribute YEAR from dimension $TIME$. Notice that attribute STORE_ID is unique in SALES, attributes CITY, STATE, COUNTRY and YEAR are unique in DEM.

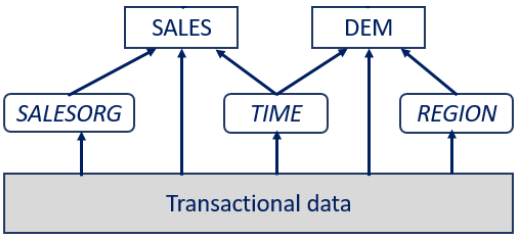


Figure 1.1: View definition of tables SALES and DEM

In our example all tables are defined as views over transactional data as shown on Figure 1.1. Dimension tables are represented by rounded rectangles and fact

Table 1.1: Tables SALES, DEM, *SALESORG*, *REGION*, *TIME*

(a) SALES

	STORE_ID	CITY	COUNTRY	YEAR	AMOUNT(M)
s_1	Oh_01	Dublin	USA	2018	3.2
s_2	Ca_01	Dublin	USA	2018	5.3
s_3	Ir_01	Dublin	Ireland	2018	45.1

(b) DEM (Demographics)

	CITY	STATE	COUNTRY	YEAR	POP(K)	UNEMP(%)
d_1	Dublin	Ohio	USA	2018	61	2.5
d_2	Dublin	California	USA	2018	42	3.1
d_3	Dublin	-	Ireland	2018	527	5.7
d_4	San Jose	California	USA	2018	1,035	2.3

(c) *SALESORG*

STORE_ID	CITY	STATE	COUNTRY
Oh_01	Dublin	Ohio	USA
Ca_01	Dublin	California	USA
Ir_01	Dublin	-	Ireland

(d) *REGION*

CITY	STATE	COUNTRY	CONTINENT
Dublin	Ohio	USA	North America
Dublin	California	USA	North America
Dublin	-	Ireland	Europe
Paris	-	France	Europe
Berlin	-	Germany	Europe

(e) *TIME*

DATE	WEEK	MONTH	YEAR
1/1/2018	1	1	2018
2/1/2018	1	1	2018
3/1/2018	1	1	2018
...	...	...	...

tables by square rectangles. A data analyst now might want to complement the information about stores in table SALES by adding their states. This can be achieved by augmenting the schema of SALES with STATE of dimension $SALESORG$ which yields a new fact table *view* SALES_SALESORG.

This view can be materialized by a left-outer join with SALESORG using the following query (the result is shown in Table 1.2):

Listing 1.1: Query $Q_{SALES\_SALESORG}$

```
SELECT STORE_ID, CITY, STATE, COUNTRY, YEAR, AMOUNT
FROM SALES
LEFT OUTER JOIN SALESORG
ON SALES.STORE_ID = SALESORG.STORE_ID
    AND SALES.CITY = SALESORG.CITY
    AND SALES.COUNTRY = SALESORG.COUNTRY
```

Table 1.2: SALES_SALESORG

	STORE_ID	CITY	STATE	COUNTRY	YEAR	AMOUNT(M)
s_1	Oh_01	Dublin	Ohio	USA	2018	3.2
s_2	Ca_01	Dublin	California	USA	2018	5.3
s_3	Ir_01	Dublin	-	Ireland	2018	45.1

The goal of this thesis is to propose solutions for assisting users in creating such augmented views. In particular, we will show that by exploiting available schema metadata (view definitions, constraints) extended by other user-defined metadata, it is possible to automatically generate for a given table, a set of useful and correct schema augmentations. These assisted generation process must solve various challenges we will describe below.

1.2.1 Relationship extraction

A first challenge for controlled schema augmentation is to discover the relationships between analytic datasets that can be used for identifying relevant schema augmentation paths. Many useful relationships can be extracted from the definitions of analytic dataset and the FK-PK constraints in transactional data. It is for instance possible to analyze the existing view definitions as illustrated in Figure 1.1 to extract all shared dimension attributes which define different relationships between fact tables and the dimension tables. In our example scenario, tables SALES and DEM are for example naturally related through their common attribute YEAR which comes

from the same dimension $TIME$. Similarly, attributes CITY, STATE and COUNTRY in dimension $SALESORG$ and dimension $REGION$ are semantically equivalent, *i.e.* have the same meaning in both tables.

Assume a data analyst who wants to build a new analytic table SALES_DEM by augmenting the schema of dataset SALES with the measure attributes *POP*(ulation) and *UNEMP*(loyment rate) of dataset DEM.

SALES_DEM (STORE_ID, CITY, COUNTRY, YEAR, *AMOUNT*, *POP*, *UNEMP*)

The extracted relationships can be used to define the left-outer join predicates for the materialization query of view SALES_DEM:

Listing 1.2: Query Q_{SALES_DEM}

```
SELECT STORE_ID, CITY, COUNTRY, YEAR, AMOUNT, POP, UNEMP
FROM SALES
LEFT OUTER JOIN DEM
ON SALES.YEAR = DEM.YEAR
    AND SALES.CITY = DEM.CITY
    AND SALES.COUNTRY = DEM.COUNTRY
```

1.2.2 Avoid row multiplication

The second challenge we address in our thesis is to detect and possible avoid row multiplication when merging two tables. Row multiplication can be detected by comparing the identifiers of analytic datasets.

In our example, assume that fact tables SALES and DEM contain the tuples shown in Tables 1.1a and 1.1b. Then, the left outer join query Q_{SALES_DEM} returns the augmented table SALES_DEM as shown in Table 1.3. The left outer-join operation results in multiplying some rows in SALES (STORE_ID is not an identifier in the merged table). For example, the two tuples s_1 and s_2 in SALES are multiplied into four tuple t_1, t_2, t_3, t_4 in SALES_DEM. This is because the join attributes YEAR, CITY and COUNTRY do not constitute a unique identifier in dataset DEM.

This row multiplication of SALES, however, is undesirable for many application scenarios, like feature engineering, data enrichment or data analysis. For these applications, the goal then is to monitor the schema augmentation process in order to keep the number of rows in Sales constant. This controlled schema augmentation

Table 1.3: SALES_DEM

	STORE_ID	CITY	COUNTRY	YEAR	AMOUNT(M)	POP(K)	UNEMP(%)
t_1	Oh_01	Dublin	USA	2018	3.2	61	2.5
t_2	Oh_01	Dublin	USA	2018	3.2	42	3.1
t_3	Ca_01	Dublin	USA	2018	5.3	61	2.5
t_4	Ca_01	Dublin	USA	2018	5.3	42	3.1
t_5	Ir_01	Dublin	Ireland	2018	45.1	527	5.7

avoiding row multiplication leads to the notion of *schema complement* introduced in [12].

1.2.3 Avoid incorrect and ambiguous reduction

A possible solution to avoid row multiplication is to reduce the attributes in the identifier of the target table by applying queries like aggregation, pivot and filter. We call these queries producing tables with less identifier (key) attributes reduction operations. For example, a possible reduction operation aggregates the measures of table DEM grouped by attributes YEAR, CITY, COUNTRY before performing the left outer join. Then, the result table identifier is defined by the attributes YEAR, CITY and COUNTRY (without attribute STATE) and the following merge will generate exactly one tuple for each tuple in table SALES.

However, when looking at the result produced by an aggregation reduction in more detail, we can identify a third challenge concerning the correctness of aggregated attribute vales. We consider mainly two sub-problems to assure that and aggregation query computes a correct result. First, it is not always possible to apply any aggregation function to a given measures. For example, the population *POP* in table DEM can be summed and averaged which is not possible for the unemployment rate *UNEMP* which can only be aggregated by MAX and MIN. Furthermore, after applying function AVG on the population *POP*, we also need to determine which aggregation functions are applicable to the new averaged *POP* value in future operations.

Secondly, we want to detect *ambiguous* aggregated values automatically. Formally, a value is ambiguous if it is not possible to identify the correct entity to which it refers in some dimension. For example, suppose that the previous aggregation (reduction) query on table DEM computes the sum of attribute *POP* and the minimum of attribute *UNEMP* grouped by YEAR, CITY and COUNTRY. The result table AGG_DEM is shown in Table 1.4. We can now show that the cities related to the values of attributes *SUM_POP* and *MIN_UNEMP* in AGG_DEM cannot be identified anymore since attribute STATE has

been removed. For example, r_1.*SUM_POP* in AGG_DEM aggregates the values d_1.*POP* and d_2.*POP* from table DEM with the population of two different cities "Dublin" in "Ohio" and "California" which might lead to an incorrect interpretation by the user. We call r_1.*SUM_POP* in AGG_DEM *ambiguous* with respect to table DEM. Observe that the ambiguity of an aggregated value depends on contents of table DEM and the aggregated values for Dublin in Ireland and San Jose are not ambiguous.

Table 1.4: AGG_DEM

	CITY	COUNTRY	YEAR	SUM_POP(K)	MIN_UNEMP(%)
r_1	Dublin	USA	2018	103	2.5
r_2	Dublin	Ireland	2018	527	5.7
r_3	San Jose	USA	2018	1,035	2.3

Continuing the previous example, a left outer join between fact table SALES and the reduced table AGG_DEM over attributes YEAR, CITY, COUNTRY produces a new fact table SALES_AGG_DEM as shown in Table 1.5. The ambiguous values of SUM_POP and MIN_UNEMP are brought to the new fact table and any further operations on SALES_AGG_DEM might compute an incorrect result. Therefore, ambiguous measure values should be detected and controlled, *e.g.*, by assigning a $null$ value to SUM_POP and MIN_UNEMP for tuples r_1 and r_2 in table AGG_DEM and tuples a_1 and a_2 in table SALES_AGG_DEM or by adding a new Boolean attribute IS_AMBIGUOUS to indicate that these tuples contain ambiguous values.

Table 1.5: SALES_AGG_DEM

	STORE_ID	CITY	COUNTRY	YEAR	AMOUNT(M)	SUM_POP(K)	MIN_UNEMP(%)
a_1	Oh_01	Dublin	USA	2018	3.2	103	2.5
a_2	Ca_01	Dublin	USA	2018	5.3	103	2.5
a_3	Ir_01	Dublin	Ireland	2018	45.1	527	5.7

1.2.4 Avoid incomplete merge

Another solution to avoid row multiplication is to augment the start table by a sequence of joins until the common attributes between the start table and the target table contain the identifier of the target table. For instance, we can add attribute STATE to fact table SALES by applying a left outer join between fact table SALES and dimension table $SALESORG$ to get SALES_SALESORGS (Table 1.6). Then, the common attributes YEAR, CITY, STATE and COUNTRY between table

SALESALES_SALESORGS and dimension DEM form the identifier of DEM and the left outer join will return a new fact table SALES_SALESORG_DEM without row multiplication.

Table 1.6: SALES_SALESORG_DEM

	STORE_ID	CITY	STATE	COUNTRY	YEAR	AMOUNT(M)	POP(K)	UNEMP(%)
b_1	Oh_01	Dublin	Ohio	USA	2018	3.2	61	2.5
b_2	Ca_01	Dublin	California	USA	2018	5.3	42	3.1
b_3	Ir_01	Dublin	-	Ireland	2018	45.1	527	5.7

However, in this case we face our fourth challenge we call *incomplete merge*. For example, suppose we want to compare the values of attribute *AMOUNT* with the population *POP* in each STATE or COUNTRY. For this we sum the population *POP* over STATE and COUNTRY in table SALES_SALESORG_DEM. For example for the state of "California", the total population would be $42K$ in SALES_SALESORG_DEM. However this value is incorrect, since the population of the city of "San Jose" in DEM does not appear in fact table SALES and its augmented table SALES_SALESORG_DEM (the correct total population of "California" with respect to table DEM is $1,077K$). We will call table SALES_SALESORG_DEM *incomplete* with respect to dimension DEM.

Observe that table SALES_SALESORG_DEM could be repaired by adding a tuple b_4 with a null value for attribute STORE_ID as shown in Table 1.7. Because only one tuple is added and null values are allowed in the key attributes, the identifier of Table 1.7 is still STORE_ID. In general case when several tuples are added, the identifier of the repaired SALES_SALESORG_DEM would be the same as if we apply a schema augmentation between SALES and DEM which contains all dimension attributes STORE_ID, CITY, STATE, COUNTRY and YEAR.

Table 1.7: SALES_SALESORG_DEM′

	STORE_ID	CITY	STATE	COUNTRY	YEAR	AMOUNT(M)	POP(K)	UNEMP(%)
b_1	Oh_01	Dublin	Ohio	USA	2018	3.2	61	2.5
b_2	Ca_01	Dublin	California	USA	2018	5.3	42	3.1
b_3	Ir_01	Dublin	-	Ireland	2018	45.1	527	5.7
b_4	-	San Jose	California	USA	2018	-	1,035	2.3

One goal of the thesis will be to automatically detect and possibly repair such incomplete merge results.

1.3 Research Contributions

SAP – as the the market leader in enterprise application software, helping companies to manage business operations and customer relations (over 437 000 clients in 190 countries [1]), has addressed the challenges of allowing business users prepare their own datasets for several years. It's in this context that this thesis is carried as an industrial Ph.D. project which collaborates with the computer science research laboratory LIP6. The objective of this research is to facilitate the operation of extending an initial dataset with columns from other datasets, and to measure and share the new dataset.

This thesis presents a new solution for business users and data scientists who want to augment the schema of analytic datasets with attributes coming from other datasets related through relationships. This is achieved by automatically extracting relationships, discovering related datasets and computing correct, non-ambiguous and complete schema augmentations or schema complements.

More specifically, we make the following technical contributions for solving all challenges presented before.

- 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. (See Section 2.2.2 in Chapter 2).

- We give formal definitions for schema augmentation, schema complement and merge query in the context of analytic tables. We then present several reduction operations that are used to enforce schema complements when schema augmentation yields a row multiplication in the augmented dataset. These operations extend previous contributions on schema augmentation and schema complement (e.g., [12]–[14]) to the case of analytic datasets. (See Sections 3.1 to 3.3 in Chapter 3).

- We define formal quality criteria for schema augmentations, schema complements and merge queries. These criteria are used to define algorithms to control the correctness, non-ambiguity and and completeness of generated schema augmentations. (See Section 3.4 in Chapter 3).

- We describe the implementation of our solution as a REST service within SAP HANA platform and provide a detailed description of our algorithms. We

[1] `https://www.sap.com/france/about/customer-stories.html`

separate the generic part of the algorithms from the specific implementation optimizations done by leveraging the capabilities of SAP HANA database. (See Chapter 4).

- 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. (See Chapter 6).

1.4 Organization of the Manuscript

The remaining of the thesis is structured in following chapters.

Chapter 1 introduces few background foundations that the thesis starts with. In particular, the chapter investigates the four challenges that we are facing to automate the schema augmentations while ensuring the data quality, namely, extract relationships between tables, avoid row multiplications when joins, avoid incorrect and ambiguous reductions, and avoid incomplete merge. These challenges are then formalized and discussed in Chapter 3.

Chapter 2 describes the formalisation of the data models we proposed for our approaches. We use an extended multidimensional data models that contain common terminologies like *Dimensions*, *Facts*, etc., and also new definitions that are adapted to our needs like *Attribute Graph*, *Aggregable Properties*, etc.. In particular, the extracted and derived relationships resolve the first challenge – extract relationships explained in Chapter 1. The presented data models are essential for data quality guarantees and schema augmentation discoveries.

In Chapter 5, we begin our discussion of previous researches done for integrating schemas. There are various operations that perform schema integration, we introduce four fundamental approaches for schema integration as: *Schema Integration*, *Data Integration*, *Schema Complement* and *Drill-across Queries*. Our focus in this chapter is to state whether our approach is still applicable in their contexts.

Chapter 3 presents our solutions that address the rest three challenges introduced in Chapter 1. We propose a general schema integration approach as *Schema Augmentation* that performs a left-outer join between two related tables, and introduce *Reduction Queries* that transform *Schema Augmentation* into *Schema Complement*. We give formal definitions for ambiguity values and incomplete merge, follow by propositions to detect and solve these problems.

Chapter 4 introduces our implementation. We introduce the architecture of the system that implements our approaches and algorithms for the solutions described in Chapter 3. All the algorithms are implemented as REST services on SAP HANA.

Chapter 6 describes several experimental results. We first evaluate the performances of constructing attribute graphs and computing dimension identifiers. We then validate our implementations in the two case studies, it shows how our approach can position and help to improve the user's experience in a real-world scenario.

Finally, we conclude with a brief conclusion and some perspectives in Chapter 7.

?? lists several SQL queries used to construct attribute graphs and construct reduction and merge quires. **??** shows the proofs of a part of the propositions proposed in the thesis, the rest of the proofs are detailed in the context.

2

Data Model

Contents

In this chapter, we first explain the foundations of our data model which extends the relational data model with analytic tables that comprise dimension and fact tables. We define arbitrary value hierarchies that are instances of hierarchy types and describe how they are modeled using dimension tables. We introduce the novel concept of attribute graphs representing dependencies among the attributes of a dimension table, which are used to compute the identifiers of dimension tables. We then formally define fact tables and some constraints over the schema of both fact and dimension tables, called aggregable properties. We then introduce two forms of semantic relationships between tables, which are join and attribute mapping relationships, and define how to derive new relationships using fusion and composition operations.

2.1 Model Overview

In this thesis, we consider analytical databases in which *tables* are separated into *non-analytic tables* and *analytic tables*. Non-analytic tables correspond to standard relational tables storing data created using statements like in Example 2.1. An *analytic table* or *analytic view* is defined by a query over non-analytic and analytic tables. Attributes in an analytic table are categorized into two types: *dimension attributes* and *measures*. Dimension attributes describe subjects, like, CUSTOMER_NAME, ADDRESS, PHONE_NUM, BRAND. Dimension attributes can form a hierarchy. For example, dimension attributes CITY, STATE, COUNTRY can form a hierarchy CITY $\rightarrow$ STATE $\rightarrow$ COUNTRY. Measure attributes store numeric values about the subject behaviors or characteristics, like, ORDER_AMOUNT, POPULATION, QUANTITY.

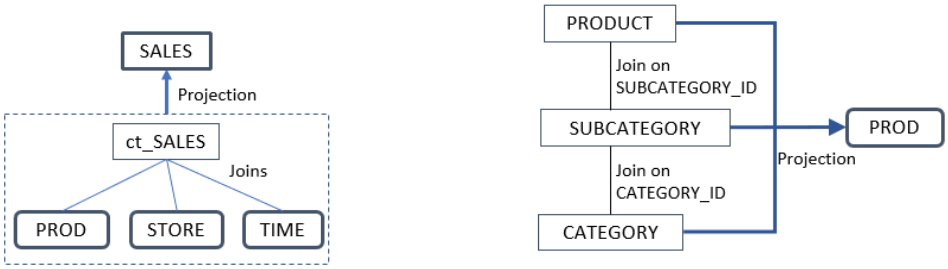
Example 2.1. The SQL statement below shows the definition of a table PRODUCT with five attributes: PRODUCT_SKU, PRODUCT_NAME, BRAND_NAME, WEIGHT and SUBCATEGORY_ID. The attribute PRODUCT_SKU is the primary key of the table and the attribute SUBCATEGORY_ID is a foreign key referring to the primary key of another table SUBCATEGORY.

Listing 2.1: Crate table statement $Q_{PRODUCT}$

```
CREATE TABLE PRODUCT
(
    PRODUCT_SKU INT NOT NULL,
    PRODUCT_NAME VARCHAR(255),
    BRAND_NAME VARCHAR(255),
    WEIGHT DOUBLE(24,3),
    SUBCATEGORY_ID INT NOT NULL,
    PRIMARY KEY (PRODUCT_SKU),
    FOREIGN KEY (SUBCATEGORY_ID) REFERENCES SUBCATEGORY(SUBCATEGORY_ID)
)
```

Following this distinction of attributes, analytic tables are categorized into two types: *dimension tables* and *fact tables*. An analytic table is a *dimension table* if it only contains *dimension attributes* describing the same kind of subjects. For example, table $CUSTOMER$ with attributes CUSTOMER_NAME, PHONE_NUM, ADDRESS, CUSTOMER_ID is a dimension table describing customers. An analytic table is a *fact table* if it contains at least one *measure*. For example, table CUSTOMER_AGE with a measure AVG_AGE and dimension attributes GENDER, COUNTRY, PROFESSION is a fact table.

Example 2.2. Figure 2.1 details the definitions of two analytic views: SALES and $PROD$. Dimension tables are represented by bold rounded rectangles, fact tables by bold square rectangles and non-analytic tables by square rectangles. SALES is a fact table defined by a star join between a non-analytic table ct_SALES and dimensions $TIME$, $PROD$ and $STORE$. $PROD$ is a dimension defined as a projection of the join result between the three non-analytic tables.



(a) Fact table SALES **(b)** Dimension PROD

Figure 2.1: Examples of definitions of analytic tables

Figure 2.2 illustrate the relationships between *analytic tables* and *non-analytic tables*. A dimension table can be defined as a view *built from* other dimensions and non-analytic tables, and a fact table can be defined as a view built from other fact, dimension and non-analytic tables. Dimension attributes in fact tables always *refer to* the dimension they come from according to the view definition. In this thesis, we assume that all analytic tables are defined as non materialized views.

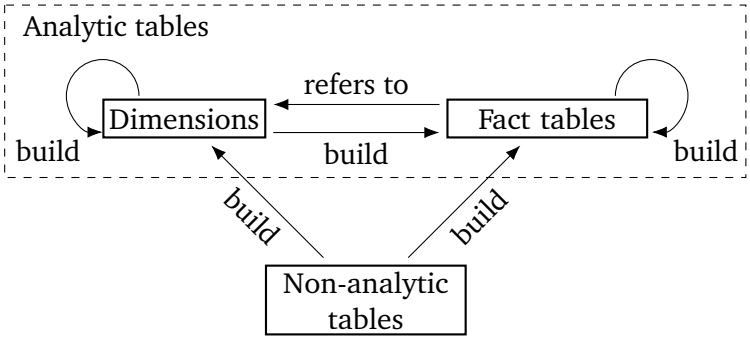


Figure 2.2: Relations between analytic tables and non-analytic tables

2.2 Analytic Tables

In this section, we define more formally the notions of hierarchy, dimension table and fact table.

2.2.1 Preliminaries

We start from the standard relational database definitions where a *relation* or *table* T is defined as a finite multiset of tuples over a set of value domains $\mathrm{S} = \{A_1, \ldots, A_n\}$, called *attributes*, where each value domain may contain a specific null marker. We call the attribute set S the *schema* of T [15]. Given an attribute A in the schema of T and a tuple $t \in T$, we use $t.\mathrm{A}$ to denote the value of A in t. By extension, we use $t.\mathrm{X}$ for a set of attributes X. Then, for two tuples t_1 and t_2, $t_1.\mathrm{A} = t_2.\mathrm{A}$ is true if both $t_1.\mathrm{A}$ and $t_2.\mathrm{A}$ are equal non-null values; false if $t_1.\mathrm{A}$ and $t_2.\mathrm{A}$ are different non-null values; and unknown otherwise (i.e., if either one of $t_1.\mathrm{A}$ or $t_2.\mathrm{A}$, possibly both, are null markers). By extension, for a set of attributes X, $t_1.\mathrm{X} = t_2.\mathrm{X}$ is true if $t_1.\mathrm{A} = t_2.\mathrm{A}$ is true for every $\mathrm{A} \in \mathrm{X}$; false if $t_1.\mathrm{A} = t_2.\mathrm{A}$ is false for some $\mathrm{A} \in \mathrm{X}$, and unknown otherwise. Two tuples are considered duplicates if all non-null attributes are equal and any null marker in one tuple is matched by a null marker in the other tuple; otherwise the tuples are distinct. The two constraints that are frequently used to restrict the data stored in a table are *primary key* constraints where a set of non-null attributes can uniquely identify the tuple within a table, and *foreign key* constraints where a set of attributes in one table refer to the primary keys of another table.

2.2.2 Hierarchical dimension tables

We consider a multidimensional data model in which each dimension consists of hierarchies of values defined by hierarchy types. We first introduce the notions of hierarchy type and hierarchy and then show how hierarchies are represented in dimension tables.

Definition 2.1 (Hierarchy Type)**.** A *hierarchy type* $\mathbf{H} = (\mathsf{L}, \preccurlyeq)$ is a set of level types $\mathsf{L} = \{L_1, \ldots, L_n\}$ 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 2.3. Consider the hierarchy types in Figure 2.3. An arc from A to B means that $A \preccurlyeq B$. Hierarchy type (a) **GEOGRAPHY** has one bottom level type $CITY$ and one top level type $CONTINENT$. Hierarchy type (b) **TIME** has two top level types $WEEK$ and $YEAR$.

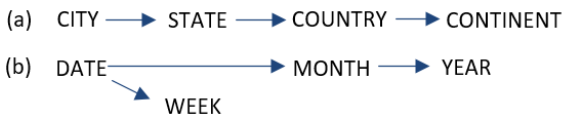


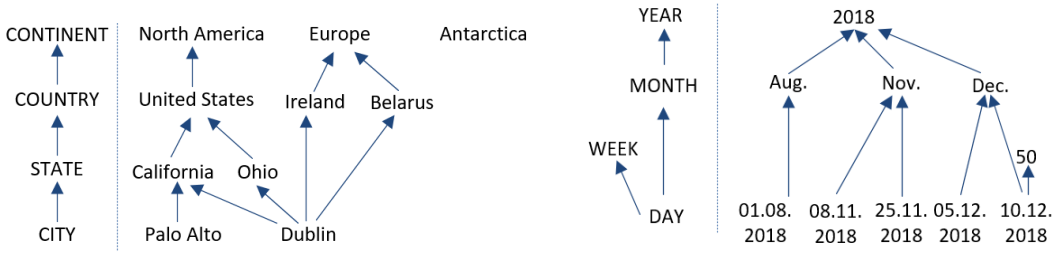
Figure 2.3: Hierarchy types

Hierarchies are common in multidimensional data model. For example, [6] introduced different types of hierarchies and characterized a hierarchy as *a cascaded series of many-to-one relationships* which form a directed graph with only one upper bound. We release this restriction and accept that a hierarchy type can have several lower and upper bounds.

A hierarchy type can have multiple hierarchy instances.

Definition 2.2 (Hierarchy instance)**.** A *hierarchy instance* (*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.*, $v_i \in N_i, v_j \in N_j \Rightarrow L_i \preccurlyeq^* L_j$.

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, the domains of different level types are not necessarily disjoint. 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.



(a) Hierarchy instance of GEOGRAPHY **(b)** Hierarchy instance of TIME

Figure 2.4: Hierarchy instance examples

Example 2.4. A partial view of hierarchy instances of type **GEOGRAPHY** and **TIME** are respectively shown in Figures 2.4a and 2.4b.

Hierarchies can naturally be represented by tables where each level type corresponds to a unique attribute of the table. These tables are called *dimension tables* or, more simply, *dimensions*.

Definition 2.3 (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 $t.\mathrm{A}_j = null$ for all other attributes in X.

The dimension table attributes that have a one-to-one mapping to a hierarchy are called the *dimension attributes* and the remaining attributes are called *detail attributes*. Each detail attribute provides descriptive information of one or more-dimension attributes, like product description, customer age, etc.

Example 2.5. Dimension table $REGION$ in Table 2.1 represents an instance of hierarchy type **GEOGRAPHY**. Each attribute of **GEOGRAPHY** is mapped to the level type with the same name. Each tuple represents a maximal path in the hierarchy instance and may have null markers (denoted with "-") for some attributes. Here, city 'Dublin' is a child value of states 'Ohio', 'California', 'Ontario', and of countries 'Ireland' and 'Belarus'. Table REGION also contains one detail attribute *TIME_ZONE* which describes the time zone of each city identified by (CITY, STATE, COUNTRY).

Table 2.1: Dimension table REGION

CITY	STATE	COUNTRY	CONTINENT	*TIME_ZONE*
Miami	Florida	United States	North America	$UTC-5$
Vancouver	British Columbia	Canada	North America	$UTC-8$
Dublin	Ohio	United States	North America	$UTC-5$
Dublin	California	United States	North America	$UTC-8$
Dublin	-	Ireland	Europe	$UTC0$
Dublin	Ontario	Canada	North America	$UTC-5$
Dublin	-	Belarus	Europe	$UTC+2$
Palo Alto	California	United States	North America	$UTC-8$
Paris	-	France	Europe	$UTC+1$
-	-	-	Antarctica	-

We adopt a *Closed World Assumption* [16] for dimensions, which means that we assume that a dimension table provides a complete information. So, if a value does not occur in the dimension table, it does not exist. For example, if dimension $REGION$ lists cities in the three continents of "North America", "Europe", and "Antarctica" then we assume that dimension table $REGION$ contains *all the cities* that exist in these continents.

2.2.3 Dimension identifiers and attribute graphs

Null markers in dimension attributes represent *non applicable values*. This semantics is different from other interpretations where null values represent missing or unknown values and are considered as placeholders for non-null values. We consider null markers as regular values and apply the same literal equality semantics as in SQL unique constraints (see e.g.,[15]): two attribute values $t_1.\mathrm{A}$ and $t_2.\mathrm{A}$ are *literally equal*, denoted by $t_1.\mathrm{A} \equiv t_2.\mathrm{A}$, iff $t_1.\mathrm{A} = t_2.\mathrm{A}$ or both values are null markers. Observe that $t_1.\mathrm{A} = t_2.\mathrm{A}$ implies $t_1.\mathrm{A} \equiv t_2.\mathrm{A}$ but the opposite is not true. Literal equality naturally extends to sets of attributes and leads to the notion of *Literal Functional Dependencies (LFD)* [17]. Let X and Y be two sets of attributes in a schema S, an LFD $\mathrm{X} \mapsto \mathrm{Y}$ holds for some table T over S iff for any two tuples t_1, t_2 of T, when $t_1.\mathrm{X} \equiv t_2.\mathrm{X}$ then $t_1.\mathrm{Y} \equiv t_2.\mathrm{Y}$. Note that if X does not contain any nullable attribute (which is for instance enforced in SQL for *primary key* or *unique* attributes), the LFD $\mathrm{X} \mapsto \mathrm{Y}$ is equivalent to the *Functional Dependency with Nulls (NFD)* $\mathrm{X} \rightarrow \mathrm{Y}$ [18]. A set of LFDs on a schema S expresses semantic properties constraining the possible "valid" tables over S.

Example 2.6. Given a table $T(\mathrm{A}, \mathrm{B})$ in Table 2.2 with two tuples t_1, t_2 such that a NFD $\mathrm{A} \rightarrow \mathrm{B}$ and a LFD $\mathrm{A} \mapsto \mathrm{B}$ both hold for T. Assuming that tuples t_3, t_4, t_5, t_6, t_7 are inserted to T sequentially, the two types of dependency will respond differently. $t_3.\mathrm{A}$ is a null value which is not allowed for the determinant in NFD, so $\mathrm{A} \rightarrow \mathrm{B}$ is not applicable for of t_3, but the insertion of t_3 is accepted by $\mathrm{A} \mapsto \mathrm{B}$. t_4 is rejected by both $\mathrm{A} \rightarrow \mathrm{B}$ and $\mathrm{A} \mapsto \mathrm{B}$ because there exists already a tuple t_1 such that $t_1.\mathrm{A} = t_4.\mathrm{A}$ but $t_1.\mathrm{B} \neq t_4.\mathrm{B}$; the same argument is applied to reject t_5. For t_6, $\mathrm{A} \rightarrow \mathrm{B}$ is still not applicable and t_6 is rejected by $\mathrm{A} \mapsto \mathrm{B}$, because there exists already t_3 where $t_3.\mathrm{A} \equiv t_6.\mathrm{A}$ and $t_3.\mathrm{B} \not\equiv t_6.\mathrm{B}$. Similar to t_6, $\mathrm{A} \rightarrow \mathrm{B}$ is also not applicable for t_7, and the insertion of t_7 is rejected by $\mathrm{A} \mapsto \mathrm{B}$.

As the previous example shows, the notion of *primary key*, which follows NFD semantics, cannot be used to state that attribute A identifies each tuple in the table. We therefore introduce the notion of *dimension identifier*, which is based on LFDs, and makes it possible declare A as the dimension identifier of the table.

Definition 2.4 (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 $\mathcal{L}$.

Table 2.2: Differences between LFD and NFD

	A	B	$A \rightarrow B$ NFD	$A \mapsto B$ LFD
t_1	a_1	b_1	Y	Y
t_2	a_3	-	Y	Y
t_3	-	b_1	NA	Y
t_4	a_1	b_2	N	N
t_5	a_1	-	N	N
t_6	-	b_2	NA	N
t_7	-	-	NA	N

Y: Accept; N: Reject; NA: Not applicable

Example 2.7. If the only LFD defined on the schema of REGION, is: (CITY, STATE, COUNTRY) $\mapsto$ CONTINENT, then the left-hand side of the LFD is the dimension identifier of REGION.

Most analytic data models assume that dimensions contain a single lowest attribute which is a primary key and therefore also plays the role of a dimension identifier. For example, the Dimensional Normal Form introduced by [19] imposes the constraint that the bottom-level attribute in the hierarchy is always the identifier of the dimension. [20] enforces a linear structured hierarchy (i.e., each attribute in the hierarchy has at most one attribute as the parent level) such that the identifier of a dimension is the bottom-level attribute in the hierarchy. With the goal of keeping the number of attributes in a dimension identifier minimal, [6] suggests the usage of a *surrogate key*, which is a dimension attribute that contains a system-generated identifier for the dimension, like CUSTOMER_ID, TIME_ID.

These constraints simplify the problem of determining the identifier of a dimension but they provide insufficient knowledge to determine the dimension identifier when some dimension attributes are projected out. For example, the bottom-level attribute STORE_ID in dimension $STORE$ is a dimension identifier. Now, assume that STORE_ID is projected out of the dimension table, we cannot determine what is the new identifier in the resulting table. Such a projection occurs when we want to record facts that refer to a higher level of the dimension than STORE_ID, such as the cities in which there are stores. We therefore need to capture the literal functional dependencies (LFD) that exist within a dimension table if we want to re-calculate dimension identifiers when attributes are projected out of the dimension table.

Although LFDs provide a formal system to define a set of logical and structural constraints over dimension tables, their practical use for characterizing a set of valid

dimension tables is limited. The number of LFDs might rapidly increase for non-linear hierarchy types and the rule-based syntax does not exploit the hierarchical type structure to help user in defining validity constraints. We thus introduce the notion of *attribute graph* which is a graph representation for LFDs in dimension tables, and characterizes all its possible "valid" hierarchy instances in a simple and natural way. We show in Section 4.2.1 how attribute graphs can automatically be extracted from dimension tables and, in Section 4.2, how to efficiently compute dimension identifiers from attribute graphs.

Definition 2.5 (Attribute graph)**.** An attribute graph over some attribute hierarchy $\mathcal{A} = (\mathsf{S}, \preccurlyeq)$ is a directed labeled graph $\mathcal{D} = (\mathsf{S}, \mathsf{R}, \lambda_{\mathsf{R}}, \bot, \top)$, 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), $\mathsf{R} \subseteq (\mathsf{S} \cup \{\bot, \top\})^2$ is a set of edges and $\lambda_{\mathsf{R}} : \mathsf{R} \to \{+, \mathbf{1}, \mathbf{f}\}$ is an edge labeling function such that there exists an edge:

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

There might exist also other edges $(\mathrm{A}_i, \mathrm{A}_j) \in \mathsf{R}$ 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. These constraints are formalized in the following definition of *valid* dimension tables.

Definition 2.6 (Valid dimension table)**.** A dimension table T with schema S is *valid* with respect to some attribute graph $\mathcal{D} = (\mathrm{X}, \mathsf{R}, \lambda_{\mathsf{R}}, \bot, \top)$ where $\mathrm{X} \subseteq \mathsf{S}$, if both of the following conditions hold in T:

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

b) If $\mathsf{R}(\mathrm{A}_i, \mathrm{A}_j) = \mathbf{1}$, then $t_1.\mathrm{A}_i = t_2.\mathrm{A}_i$ implies $t_1.\mathrm{A}_j \equiv t_2.\mathrm{A}_j$.

Observe that by Definition 2.6, $\mathsf{R}(\mathrm{A}_i, \mathrm{A}_j) = \mathbf{f}$ 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. an 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 2.8. Figure 2.5 shows an attribute graph for hierarchy type **GEOGRAPHY** that is validated by 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 the arc $\mathsf{R}(\text{STATE}, \text{COUNTRY})$ is labeled by $\mathbf{1}$, for each non-null value of attribute STATE, we can determine a unique value of attribute COUNTRY. Second, $\mathsf{R}(\text{COUNTRY}, \text{CONTINENT}) = \mathbf{f}$ states that the attribute COUNTRY literally determines the attribute CONTINENT. Third, the tuples with the same value of CITY can have different values for STATE ($\mathsf{R}(\text{CITY}, \text{STATE}) = +$) and for COUNTRY ($\mathsf{R}(\text{CITY}, \text{COUNTRY}) = +$) and no value for STATE (arc $\mathsf{R}(\text{CITY}, \text{COUNTRY})$ "skipping" attribute STATE). Finally, there exists a single continent without countries, states and cities, which is represented by the arc $\mathsf{R}(\bot, \text{CONTINENT})$ with label $\mathbf{f}$.

Figure 2.5: Attribute graph of dimension REGION

Example 2.9. Figure 2.6 (a) shows an attribute graph that is validated by dimension table $WAREHOUSE$. The lower and upper bound attributes are respectively WH_ID (connected to node $\bot$) and COUNTRY (connected to node $\top$). Attribute WH_ID literally determines CITY and STATE since both arcs $\mathsf{R}(\text{WH\_ID}, \text{CITY})$ and $\mathsf{R}(\text{WH\_ID}, \text{STATE})$ are labeled by $\mathbf{f}$. Arcs $\mathsf{R}(\text{CITY}, \text{COUNTRY})$ and $\mathsf{R}(\text{CITY}, \text{STATE})$ are labeled by $+$, which signifies that tuples with the same value for CITY can have different values for STATE and COUNTRY. Figure 2.6 (b), Figure 2.6 (c), Figure 2.6 (d) and Figure 2.6 (e) show the attribute graphs for dimensions $STORE$, $PROD$, $TIME$ and TAX respectively.

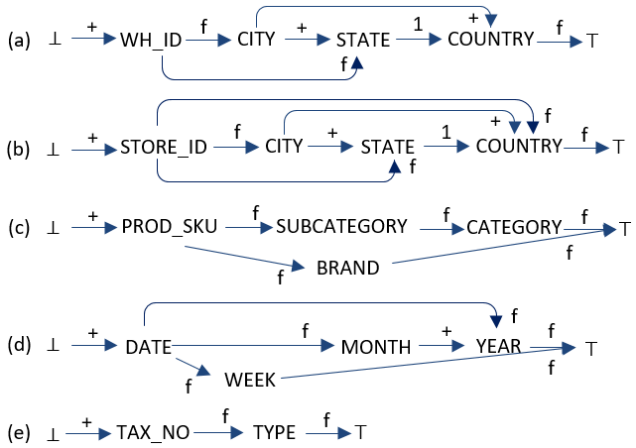


Figure 2.6: Attribute graphs of dimensions WAREHOUSE, STORE, PROD, TIME and TAX

Attribute graphs capture a set of LFDs and can be used to infer dimension identifiers for the valid dimension tables.

Proposition 2.1. Let $\mathcal{D} = (\mathsf{S}, \mathsf{R}, \lambda_{\mathsf{R}}, \bot, \top)$ be an attribute graph, the subset 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 see **??** on **??**.

Example 2.10. In the attribute graph of Figure 2.5, all attributes except attribute CONTINENT have one $+$ in-edge and no **f** in-edge. By Proposition 2.1, the dimension identifier for dimension $REGION$ is therefore {CITY, STATE, COUNTRY}. In the attribute graph of Figure 2.6 (a), attributes CITY and STATE have a label **f** in-edge and the identifier of $WAREHOUSE$ is {WH_ID, COUNTRY}. Similarly, from the attribute graphs in Figure 2.6, dimension $STORE$, $PROD$, $TIME$ and TAX have dimension identifiers STORE_ID, PROD_SKU, DATE and TAX_NO respectively.

2.2.4 Capturing hierarchy properties with attribute graphs

We now show how attribute graphs can capture three well-known semantic hierarchy properties introduced in [21]. We present the definitions of *strict*, *onto* and *covering* hierarchies and show how these definitions can be reformulated and verified using attribute graphs.

Definition 2.7 (Onto hierarchy)**.** Let $H = (N, \leq)$ be a hierarchy of hierarchy type $\mathbf{H} = (\mathsf{L}, \preccurlyeq)$, $N_i, N_j \subset N$ be two domains of values of level types $L_i, L_j \in \mathsf{L}$ respectively such that $L_i \preccurlyeq L_j$, and $L_i \neq \bot$. The value mapping from N_i to N_j is said to be *onto* in H if $\forall v_b \in N_j, \exists v_a \in N_i$ such that $v_a \leq v_b$. If all possible value mappings in H are onto, H is said to be an *onto hierarchy*.

Example 2.11. The hierarchy of Figure 2.4b (Page 19) is onto since all possible value mappings are onto. The hierarchy in Figure 2.4a, (Page 19) is not onto: the mapping from COUNTRY to CONTINENT is not onto since there exists a value "Antarctica" of type CONTINENT that has no child in type COUNTRY. Observe that the existence of this non-onto mapping from COUNTRY to CONTINENT is possible since there exists an edge $\mathsf{R}(\bot, \text{CONTINENT})$ in the attribute graph of Figure 2.5.

The following proposition provides a new definition of onto hierarchies using attribute graphs.

Proposition 2.2. Let $\mathcal{D} = (\mathsf{S}, \mathsf{R}, \lambda_{\mathsf{R}}, \bot, \top)$ be an attribute graph over an attribute hierarchy $\mathcal{A} = (\mathsf{S}, \preccurlyeq)$. Let T of schema S be the dimension that is valid with respect to $\mathcal{D}$. The hierarchy $\mathcal{A}$ is an *onto* hierarchy if $\forall \mathrm{A}_i \in \mathsf{S}$, $\mathsf{R}(\bot, \mathrm{A}_i) \in \mathsf{R}$, then $\nexists \mathrm{A}_j \in \mathsf{S}, \mathsf{R}(\mathrm{A}_j, \mathrm{A}_i) \in \mathsf{R}$.

Proof. We prove by contradiction that the hierarchy $\mathcal{A}$ is not onto when $\forall \mathrm{A}_i \in \mathsf{S}$, $\mathsf{R}(\bot, \mathrm{A}_i) \in \mathsf{R}$, $\nexists \mathrm{A}_j \in \mathsf{S}, \mathsf{R}(\mathrm{A}_j, \mathrm{A}_i) \in \mathsf{R}$.

By the definition of onto hierarchy, $\mathcal{A}$ is not onto, there exists at least one value $v_t \in N_t$ such that there is no value in N_s is the child value of v_t, where N_s, N_t are the domain values of attributes $\mathrm{A}_s, \mathrm{A}_t \in \mathsf{S}$ respectively and $\mathsf{R}(\mathrm{A}_s, \mathrm{A}_t) \in \mathsf{R}, \mathrm{A}_s \neq \bot$. Consequently, v_t of attribute A_t has no descendent value in $\mathcal{A}$, i.e., v_t is a leaf node. In the dimension table T, v_t of attribute A_t is encoded as tuples $t \in T$ where $t.\mathrm{A}_t = v_t$, since v_t has no descendent value, $\forall \mathrm{A}_k \in \mathsf{S}, \mathrm{A}_k \preccurlyeq^* \mathrm{A}_t$, $t.\mathrm{A}_k$ is null. By item 1 in Definition 2.6, there exists an edge $\mathsf{R}(\bot, \mathrm{A}_t)$ in R. Therefore, the existence of $\mathsf{R}(\bot, \mathrm{A}_t)$ and $\mathsf{R}(\mathrm{A}_s, \mathrm{A}_t)$ contradicts the assumption that when $\mathsf{R}(\bot, \mathrm{A}_t) \in \mathsf{R}$, $\mathsf{R}(\mathrm{A}_s, \mathrm{A}_t)$ does not exist.

Therefore, the hierarchy $\mathcal{A}$ is onto if $\forall \mathrm{A}_i \in \mathsf{S}$, $\mathsf{R}(\bot, \mathrm{A}_i) \in \mathsf{R}$, $\nexists \mathrm{A}_j \in \mathsf{S}, \mathsf{R}(\mathrm{A}_j, \mathrm{A}_i) \in \mathsf{R}$. □

Hierarchies without paths of domain values which can "bypass" a level type are called *covering* hierarchies.

Definition 2.8 (Covering hierarchy)**.** Let $H = (N, \leq)$ be a hierarchy of hierarchy type $\mathbf{H} = (\mathsf{L}, \preccurlyeq)$, $N_i, N_{i+1}, \ldots, N_k$ be a sequence of at least three domain values in N such that their corresponding level types are $L_i \preccurlyeq L_{i+1} \preccurlyeq \ldots \preccurlyeq L_k$. If there exists a pair of values $v_i \in N_i, v_k \in N_k$ such that $v_i \leq v_k$, then the sequence $N_{i+1}, \ldots, N_k$ is said to be *non-covering* with respect to N_i. A hierarchy H with no non-covering sequence, is called a *covering hierarchy*.

Example 2.12. The hierarchy instance of Figure 2.4a is non-covering since the sequence (STATE, COUNTRY) is non-covering with respect to CITY: the value 'Dublin' of domain (attribute) CITY is directly connected to a value 'Ireland' of domain COUNTRY ('Dublin' $\leq$ 'Ireland') and domain STATE is "bypassed" and optional. Observe that in the attribute graph of Figure 2.5, there exists a path connecting attributes CITY $\preccurlyeq$ STATE $\preccurlyeq$ COUNTRY and an edge R(CITY, COUNTRY) skipping attribute STATE. The hierarchy instance in Figure 2.4b is covering.

Proposition 2.3. Let $\mathcal{D} = (\mathsf{S}, \mathsf{R}, \lambda_{\mathsf{R}}, \bot, \top)$ be an attribute graph over an attribute hierarchy $\mathcal{A} = (\mathsf{S}, \preccurlyeq)$. Let $T(\mathsf{S})$ be a dimension that is valid with respect to $\mathcal{D}$. The hierarchy $\mathcal{A}$ is a *covering* hierarchy if $\forall \mathrm{A}_i, \mathrm{A}_j \in \mathsf{S}$, when $\mathsf{R}(\mathrm{A}_i, \mathrm{A}_j) \in \mathsf{R}$, then $\nexists \mathrm{A}_k \in \mathsf{S}, \mathsf{R}(\mathrm{A}_i, \mathrm{A}_k) \in \mathsf{R}$.

Proof. By the definition of covering hierarchy, a hierarchy is covering if it does not contain non-covering sequence, or "bypassed" level, this is equivalent with every attribute in the hierarchy only has one parent attribute. Therefore, if $\mathsf{R}(\mathrm{A}_i, \mathrm{A}_j) \in \mathsf{R}$, A_i will not have other parent attribute, and $\mathsf{R}(\mathrm{A}_i, \mathrm{A}_k)$ can not exist in R. □

Onto and covering hierarchies mainly characterize dimension tables without $null$ markers and are called *complete* hierarchies in [22]. In addition, the last notion of *strict hierarchies* characterizes hierarchies without many-to-many child-parent relationships between domain values.

Definition 2.9 (Strict hierarchy)**.** Let $H = (N, \leq)$ be a hierarchy of hierarchy type $\mathbf{H} = (\mathsf{L}, \preccurlyeq)$, $N_i, N_j \subset N$ be two domains of values of level types $L_i, L_j \in \mathsf{L}$ respectively such that $L_i \preccurlyeq L_j$. The value mapping from N_i to N_j is said to be *strict* in H, if $\forall v_a \in N_i$, there exists only one value $v_b \in N_j$ such that $v_a \leq v_b$. If all possible value mappings from any two domains of values in N of H are *strict*, H is said to be a *strict hierarchy*.

Example 2.13. As shown in Figure 2.4a, the mapping from CITY to STATE is not strict since there is a value 'Dublin' that has more than one parent in STATE, e.g., 'California' and 'Ohio'. And the edge $\mathsf{R}(\text{CITY}, \text{STATE})$ is labeled $+$ in the attribute graph of Figure 2.5. Therefore, the non-covering and non-onto hierarchy in Figure 2.4a is also not strict and the hierarchy instance shown in Figure 2.4b is strict, covering and onto.

Proposition 2.4. Let $\mathcal{D} = (\mathsf{S}, \mathsf{R}, \lambda_{\mathsf{R}}, \bot, \top)$ be an attribute graph over an attribute hierarchy $\mathcal{A} = (\mathsf{S}, \preccurlyeq)$. Let T of schema S be the dimension that is valid with respect to $\mathcal{D}$. The hierarchy $\mathcal{A}$ is a *strict* hierarchy if $\forall \mathrm{A}_i, \mathrm{A}_j \in \mathsf{S}$, when $\mathsf{R}(\mathrm{A}_i, \mathrm{A}_j) \in \mathsf{R}$, then $\mathsf{R}(\mathrm{A}_i, \mathrm{A}_j) = \mathbf{f}$.

Proof. By the definition of strict hierarchy, a hierarchy is strict if every child-parent value mapping is strict. We have $\forall v_i \in N_i$, there exists one and only one value $v_j \in N_j$ such that $v_i \leq v_j$ where N_i, N_j are domain values of attributes $\mathrm{A}_i, \mathrm{A}_j \in \mathsf{S}$ respectively and $\mathrm{A}_i \preccurlyeq \mathrm{A}_j$. Then, in the dimension table T, we get $\forall t_a, t_b \in T, t_a \neq t_b$, if $t_a.\mathrm{A}_i \equiv t_a.\mathrm{A}_i$, then $t_a.\mathrm{A}_j \equiv t_b.\mathrm{A}_j$. By item 2.a in Definition 2.6, $\mathsf{R}(\mathrm{A}_i, \mathrm{A}_j) = \mathbf{f}$. □

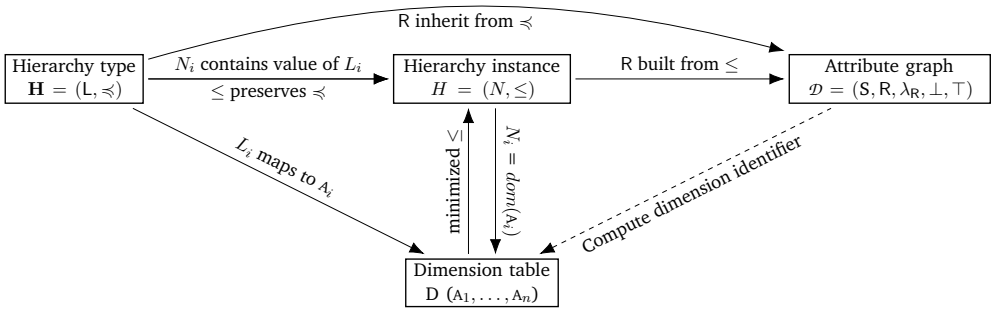


Figure 2.7: Relations between hierarchy, dimension table and attribute graph

By Propositions 2.2 to 2.4 attribute graphs capture all three hierarchy properties *strict*, *onto* and *covering*. More exactly, these propositions provide *sufficient* conditions to decide if a valid dimension table is strict, onto and/or covering.

We summarize the relationships between *hierarchy types*, *hierarchies*, *dimension tables* and *attribute graphs* in Figure 2.7. A hierarchy type can have multiple hierarchy instances where all value mappings preserve the child-parent relationships in the hierarchy type. Each hierarchy instance yields a unique dimension table whose schema contains one dimension attribute for each level type. Attribute graphs reflect the child-parent relationships of hierarchy types and define subsets of *valid* hierarchy instances. They can be used to compute the dimension identifiers of dimension tables (hierarchy instances).

2.2.5 Multidimensional fact tables

We now introduce fact tables that associate measures with dimensions to represent facts.

Definition 2.10 (Fact table)**.** A *fact table* over a set of dimensions $D_1, \cdots, D_n$ is a table $T(\mathrm{S})$ where schema S contains a non-empty subset X_i of dimension attributes from dimension D_i, and a non-empty set of attributes Z representing one or more *measures*. 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 one attribute having the role of *Value* and a possibly empty group of attributes having the role of *Detail*. The *Value* attribute of a measure carries the actual value while the *Detail* attribute provide optional auxiliary information on the measure.

Example 2.14. We give below an example of three fact tables SALES, SALES_SUM, and INVENTORY that are built from two non-analytic tables ct_SALES and ct_INVENTORY. Measure attributes are in italics. SALES is defined as a view over the non-analytic table ct_SALES and dimensions $PROD$, $TIME$ and $STORE$. It describes the daily sales of products in different stores. SALES contains dimension attributes PROD_SKU and BRAND from dimension $PROD$, attribute and YEAR from dimension $TIME$ and attributes CITY, STATE and COUNTRY from dimension $STORE$. Measure *AMOUNT* in table SALES is associated with a detail attribute *CURRENCY* that describes the currency of the *AMOUNT* value, e.g, *USD*, *EUR*.

ct_SALES (PROD_SKU, BRAND, DAY, MONTH, YEAR, STORE_ID, CITY, STATE, COUNTRY, AMOUNT, CURRENCY)
ct_INVENTORY(PROD_SKU, BRAND, MONTH, YEAR, WH_ID, CITY, STATE, COUNTRY, TAX_NO, RATE, TAX_DESC, QTY_ON_HAND, TAX_AMT)
SALES (PROD_SKU, BRAND, YEAR, CITY, STATE, COUNTRY, *AMOUNT*, *CURRENCY*)
SALES_SUM (YEAR, COUNTRY, *SUM_AMOUNT*, *CURRENCY*)
INVENTORY (PROD_SKU, BRAND, MONTH, YEAR, WH_ID, CITY, COUNTRY, TAX_NO, RATE, TAX_DESC, *QTY_ON_HAND*, *UNIT*)

Table SALES_SUM describes total sales for each country and is defined over fact table SALES from where it *inherits* dimension attribute YEAR from dimension $TIME$ and attribute COUNTRY from dimension $STORE$. Its measure attribute *SUM_AMOUNT* also has a detail attribute *CURRENCY* that provides the currency of the *SUM_AMOUNT* value.

Table INVENTORY is defined over non-analytic table ct_INVENTORY and dimensions $WAREHOUSE$, $TIME$, TAX and $PROD$. Its measure attribute *QTY_ON_HAND* has a detail attribute *UNIT* that describes the unit of the *QTY_ON_HAND* value, e.g., *K* or $1,000$, *M* or $1,000,000$.

As shown by the previous example, we make the additional assumption on fact tables that each measure attribute can be associated with a specific *Unit* or *Currency* detail attribute to control the application of aggregation functions.

Similar to dimension tables, we define *fact identifiers* which capture LFDs in a fact table.

Definition 2.11 (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 $\mathcal{L}$.

We suppose that each measure in a fact table is determined by a subset of the dimension attributes of the table. Importantly, and unlike many other multidimensional models [19], [23]–[25], a measure can depend on a subset of the dimensions of a fact table. Then, fact identifiers are determined by LFDs between dimension attributes and attribute graphs can be used to compute fact identifiers of their valid tables as stated in the following proposition.

Proposition 2.5. Let $T(\mathrm{S})$ be a fact table defined over a set of dimensions $D_1, \cdots, D_n$ and $K_1, \ldots K_n$ be the dimension identifiers of $D_1 \cap \mathrm{S}, \ldots, D_n \cap \mathrm{S}$ respectively. Then $K = K_1 \cup \ldots \cup K_n$ is a fact identifier of T, and K is a minimal identifier if all dimensions in T are mutually independent, i.e., for any pair of dimensions $D_i, D_j, 0 \leqslant i, j \leqslant n, i \neq j$ in T, $D_i \not\mapsto D_j$ and $D_j \not\mapsto D_i$.

Proof. Let $\mathrm{S}_D \subset \mathrm{S}$ the set of dimension attributes in T and $\mathrm{S} - \mathrm{S}_D$ be measure attributes. We have $\mathrm{S}_D \mapsto (\mathrm{S} - \mathrm{S}_D)$.

$K_1, \ldots, K_n$ 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 transitivity of LFD's, we get $(K_1 \cup \ldots \cup K_n) \mapsto (\mathrm{S} - \mathrm{S}_D)$, and then by union of LFDs, we have $(K_1 \cup \ldots \cup K_n) \mapsto \mathrm{S}$. Thus, K is a fact identifier.

Two dimension identifiers K_i, K_j, $i \neq j$ are independent, if there exists no attribute $\mathrm{A} \in K_j$ such that $K_i \mapsto \mathrm{A}$. 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 fact identifier of T. □

Example 2.15. Consider fact tables SALES, SALES_SUM and INVENTORY introduced in Example 2.14. In SALES, for the dimension attributes from dimension $PROD$ we have PROD_SKU $\mapsto$ {PROD_SKU, BRAND}, for the dimension attributes from dimension $TIME$ we have {YEAR} $\mapsto$ {YEAR}, and for the dimension attributes from dimension $STORE$ we have {CITY, STATE, COUNTRY} $\mapsto$ {CITY, STATE, COUNTRY}. Therefore, by taking the union of all identifiers, the fact identifier of SALES is {PROD_SKU, YEAR, CITY, STATE, COUNTRY}. Similarly, fact identifiers of SALES_SUM and INVENTORY are defined by the underlined attributes. If all dimensions are mutually independent, these fact identifiers are minimal, *i.e.*, no proper subset is also a fact identifier.

SALES (PROD_SKU, BRAND, YEAR, CITY, STATE, COUNTRY, *AMOUNT*)
SALES_SUM (YEAR, COUNTRY, *SUM_AMOUNT*)
INVENTORY (PROD_SKU, BRAND, MONTH, YEAR, WH_ID, CITY, COUNTRY, TAX_NO,
RATE, TAX_DESC, *QTY_ON_HAND*)

2.2.6 Aggregable attributes in analytic tables

Any attribute of an analytic table is a priori aggregable (using an aggregate query) along some dimension attributes, however it does not necessarily aggregate with all aggregation functions along all dimension attributes.

Example 2.16. In the schema of fact table SALES, measure *AMOUNT* is aggregable using function SUM along all dimension attributes. Dimension attribute PROD_SKU is aggregable using function COUNT or COUNT_DISTINCT along all dimension attributes but no other meaningful function is applicable to that attribute. In fact table INVENTORY, measure *QTY_ON_HAND* is not aggregable along the $TIME$ dimension, *i.e.* summing the quantity of products *QTY_ON_HAND* over all months or all years is meaningless. In other words, an aggregation that sums up *QTY_ON_HAND* must have attributes MONTH and YEAR in the group by clause.

The issue of reasoning on the "semantic aggregability" of attributes has been identified and extensively studied for statistical and OLAP databases [26]. Focused on the aggregate function SUM, the notion of *additivity* was proposed in [6] as: “if a measure in a fact table can be summed along any dimension associated to the fact table then it is additive”. Measures in fact tables are then classified into three categories: *fully-additive* measures can be summed along any dimension associated, *semi-additive* measures can be summed along some, but not all, dimensions, and

non-additive measures cannot be summed along any dimension. Additional special cases of additivity, such as temporal additivity, have also been considered in [27].

We generalize the additivity approach of [6], [27] and [20] by considering any type of aggregate function. Our model enables the designer of an analytic table to declare for each attribute and each applicable aggregation function 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 X, then it is also aggregable along any subsets of X. In the following we denote by $\mathbf{agg}_{\mathrm{A}}(\mathrm{F}, \mathrm{X})$ the *aggregable property* of A and state that property $\mathbf{agg}_{\mathrm{A}}(\mathrm{F}, \mathrm{X})$ holds in T if X is the maximal set of attributes along which A is aggregable using F in T. We give the following formal definition for aggregable properties $\mathbf{agg}_{\mathrm{A}}(\mathrm{F}, \mathrm{X})$.

Definition 2.12 (Aggregable Property)**.** Let S_D 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 aggregation function on A. Aggregable property $\mathbf{agg}_{\mathrm{A}}(\mathrm{F}, \mathrm{X})$, $X \subseteq \mathrm{S}_D$, holds in T if

1. all aggregations along any subsets of X are considered as meaningful by the user;
2. If A is a measure attribute, X contains: (1) the minimal subset of dimension attributes $\mathrm{U} \subseteq \mathrm{S}_D$ such that $\mathrm{U} \mapsto \mathrm{A}$ and (2) all attributes $\mathrm{B} \in \mathrm{S}_D$ such that $\mathrm{U} \mapsto \mathrm{B}$
3. F is applicable to A.

Example 2.17. Considering Example 2.16, attribute *QTY_ON_HAND* depends on all dimensions, so the (supposed) minimal set of attributes U is {PROD_SKU, MONTH, YEAR, TAX_NO}, if dimensions are mutually independent. However, aggregating *QTY_ON_HAND* along the attributes of dimension $TIME$ is considered meaningless to the user, and by item 1 in Definition 2.12 attributes MONTH and YEAR must be removed from U. All other dimension attributes are determined by U. We then state that $\mathbf{agg}_{\mathit{QTY\_ON\_HAND}}(\mathrm{SUM}, Z)$ holds in INVENTORY where Z contains all dimension attributes of INVENTORY except attributes MONTH and YEAR.

Example 2.18. Consider now a fact table PRODUCT_LIST (PROD_SKU, COUNTRY, BRAND, YEAR, QTY) over dimension MKT_PROD and $TIME$, which describes the quantity of products issued every year as shown in Table 2.3. In dimension MKT_PROD, assume that we have the attribute graph displayed in Figure 2.8. Suppose that the measure attribute QTY only depends on the minimal set of attributes $\mathrm{U} = \{\mathrm{PROD\_SKU}, \mathrm{YEAR}\}$. Then, since no other dimension attribute is determined by

U, $\mathbf{agg}_{QTY}(\text{SUM}, U)$ holds in PRODUCT_LIST. Here, the same fact about "cz-tshirt-s" is recorded twice, once for each brand. If we wanted to consider facts to be independent from each other, then we would assume that QTY depends on the minimal set of attributes $\text{U} = \{\text{PROD\_SKU}, \text{BRAND}, \text{YEAR}\}$. That is, each quantity would be recorded separately for each brand and they could be added along dimension MKT_PROD.

Table 2.3: PRODUCT_LIST

PROD_SKU	BRAND	COUNTRY	YEAR	QTY
coca-zara-tshirt-s	COCA COLA	United States	2017	5 000
coca-zara-tshirt-s	ZARA	Spain	2017	5 000
coca-33cl-can	COCA COLA	United States	2017	10 000

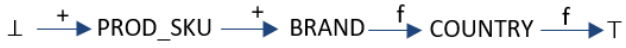


Figure 2.8: Attribute graph of dimension MKT_PROD

To determine applicable aggregation functions for aggregable attributes, different categorizations have been proposed in previous works. One factor that is used to classify attributes is their semantic meaning. For instance, [28] introduced a statistic classification that divides measurement attributes into four types: 1) *Nominal attributes*, which are un-ordered categorical attributes like CLASS_TYPE and can only be aggregated using function COUNT. 2) *Ordinal attributes*, which are ordered categorical attributes, like IQ_SCORE, and aggregable with function MIN, MAX, COUNT, other functions like SUM and AVG are not suggested because the numeric difference between values might not correctly reflect the "semantic" difference, e.g., the difference of IQ scores from 115 to 100 does not have the same meaning as the difference from 100 to 85. 3) *Interval attributes*, which are measure attributes with equal value intervals but don't have an absolute zero, like TEMPERATURE, zero degrees Fahrenheit and zero degrees Celsius are different temperatures, and neither indicates the absence of temperature. An interval attribute can be aggregated by function MAX, MIN, COUNT, AVG. 4) *Ratio attributes*, which are numerical values with an absolute zero, like WEIGHT and SALES, can be aggregated by any aggregation function. Another criteria that is used to classify attributes is their aggregation behaviour. For instance, [21], [29] classified measures into three types: measures that can be (a) summed, (b) averaged, or (c) counted.

The previous categorizations of measures rely on some external knowledge and explicit user effort for classifying every aggregable attribute of an analytic table. To reduce the user effort, we provide a default categorization that can automatically

be extracted from the schema metadata. The two categories **NUM** and **DESC** are inferred from the (SQL) data type of attributes. A third category **STAT** are numerical values that result from the use of some aggregation functions. Table 2.4 describes the six common SQL aggregation functions applicable to each category.

Table 2.4: Categories of aggregable attributes

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

Table 2.5: Category of the domain and the co-domain for common aggregation functions

Functions	Applicable category	Category of the co-domain
SUM, MIN, MAX	**NUM**	**NUM**
COUNT, COUNT_DISTINCT	**NUM**, **DESC**, **STAT**	**NUM**
AVG	**NUM**	**STAT**

Example 2.19. Attribute STORE_ID of data type `string` in dimension table $STORE$, or in fact table SALES, is an aggregable attribute of category **DESC** which can only be counted. Attributes AMOUNT and QTY_ON_HAND in tables SALES and INVENTORY are both of category **NUM**. Thus, by default, both attributes can be summed up unless a user-defined aggregable property is specified, as in Example 2.17 for attribute QTY_ON_HAND.

If A is an attribute of T for which no aggregable property is defined by a user, we use the default category of A to define which aggregation function F is applicable to A and 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). We assume that any knowledge of the categorization of measures is translated into aggregable properties (as done e.g., in [20]). For this purpose, aggregable properties of fact tables can be manually edited by the user.

2.3 Table Relationships

2.3.1 Join and attribute mapping relationships

In this section, we describe two kinds of semantic relationships, *join relationships* and *attribute-mapping relationships*, that can exist between analytic tables. Both kinds of relationships identify pairs of equivalent attributes in two tables. They can be defined manually by the business user or extracted from the view definitions of analytic tables and from the foreign key constraints contained in the analytic schema.

Definition 2.13 (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 $T_1.\mathrm{A} = T_2.\mathrm{B}$. It relates all pairs of tuples in T_1 and T_2 satisfying the predicate $P_1 \wedge \cdots \wedge P_k$, *i.e.*, the set of all tuples joined by the query $T_1 \bowtie_{P_1 \wedge \cdots \wedge P_k} T_2$.

Example 2.20. Fact table SALES of Example 2.14 is a view over dimensions $PROD$, $TIME$ and $STORE$ and a non-analytical table ct_SALES. As shown in Figure 2.1a, the view defines a star join connecting ct_SALES and the dimension tables. The analysis of the view definition yields three join relationships between ct_SALES and each dimension table.

Join relationships might also exist between non-analytic tables (*i.e.*, tables that are neither dimension nor fact tables) through foreign key constraints in the table schema definitions.

Definition 2.14 (Attribute mapping relationship)**.** Let T_1 and T_2 be two tables such that T_1 is derived from T_2 using a query Q and auxiliary tables: $T_1 = Q(T_2, T_i, \cdots, T_n)$ with $n \geq 0$. Query Q defines an *attribute mapping* from $T_2.\mathrm{B}$ to $T_1.\mathrm{A}$, denoted by $T_2.\mathrm{B} \twoheadrightarrow T_1.\mathrm{A}$, if for all possible values $x \in dom(\mathrm{A})$ and all possible instances of $T_2, T_i, \cdots, T_n$:

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

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

Example 2.21. As shown in Figure 2.1b, Page 17, dimension $PROD$ is a view over three non-analytic tables defined as $PROD = \pi_{\mathrm{X}}(\mathrm{PRODUCT} \bowtie_{P_1} \mathrm{SUBCATEGORY} \bowtie_{P_2} \mathrm{CATEGORY})$, where X is the set of attributes in $PROD$, and P_1, P_2 are join predicates. This view defines for each attribute in $\mathrm{A}_i \in \mathrm{X}$, an attribute mapping $PROD.\mathrm{A}_i \twoheadrightarrow T.\mathrm{A}_i$ between $PROD$ and the corresponding non-analytic table $T \in \{\mathrm{PRODUCT}, \mathrm{SUBCATEGORY}, \mathrm{CATEGORY}\}$.

In Example 2.20, the view query for fact table SALES defines an attribute mapping relationship between table SALES and the three dimension tables $PROD$, $TIME$ and $STORE$, as well as the non-analytic table ct_SALES.

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*.

Example 2.22. Consider 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 $\mathsf{R}(\mathrm{ORDER}, TIME)$ mapping the two ORDER attributes to the same attribute $TIME$.DAY, $\mu_{\mathsf{R}}(\mathrm{ORDER.ORDER\_DAY}) = TIME.\mathrm{DAY}$, $\mu_{\mathsf{R}}(\mathrm{ORDER.SHIP\_DAY}) = TIME.\mathrm{DAY}$, is *not a one-to-one mapping* and therefore *not well-formed*. Thus, two distinct relationships $\mathsf{R}_1(\mathrm{ORDER}, TIME)$ and $\mathsf{R}_2(\mathrm{ORDER}, TIME)$ are needed, one for ORDER_DAY and the other for SHIP_DAY.

Given a relationship $\mathsf{R}(T_1, T_2)$ between two tables $T_1(\mathrm{S}_1)$ and $T_2(\mathrm{S}_2)$, when T_1 or T_2 is an analytic table, we refer to *common dimension attributes* as the set of dimension attributes that are both in the schema of T_1 and T_2, noted $\mathrm{Y} = \mathrm{S}_1 \cap_D \mathrm{S}_2$. When T_1 and T_2 are two non-analytic tables, we refer to *common non-analytic attributes* as the set of attributes that are both in the schema of T_1 and T_2, noted $\mathrm{Y} = \mathrm{S}_1 \cap_D \mathrm{S}_2 = \mathrm{S}_1 \cap \mathrm{S}_2$. For the sake of simplicity, and unless specified differently, we shall use in the following the name *common attributes* and expression $\mathrm{Y} = \mathrm{S}_1 \cap_D \mathrm{S}_2$ for common dimension attributes or common non-analytic attributes between two tables, and we assume that any relationship $\mathsf{R}(T_1, T_2)$ defines a *natural* mapping $\mu_{\mathsf{R}}(T_1.\mathrm{A}) = T_2.\mathrm{A}$ for all attributes A in $\mathrm{Y} = \mathrm{S}_1 \cap_D \mathrm{S}_2$.

Example 2.23. In Example 2.20, the common non-analytic attributes for the join relationships $\mathsf{R}(\mathrm{CATEGORY}, \mathrm{SUBCATEGORY})$ and $\mathsf{R}(\mathrm{PRODUCT}, \mathrm{SUBCATEGORY})$ are respectively {CATEGORY_ID}

and $\{$SUBCATEGORY_ID$\}$. Similarly, the attribute mapping relationship R$(PROD, \text{SUBCATEGORY})$ of Example 2.21 between dimension table $PROD$ and non-analytical table SUBCATEGORY has common dimension attributes $\{$PROD_SKU, BRAND$\}$.

Relationship R$(\text{SALES}, \text{ct_SALES})$ described in Example 2.21 is an attribute mapping between attributes $\{$PROD_SKU, BRAND, YEAR, CITY, STATE, COUNTRY, AMOUNT, CURRENCY$\}$. Within those attributes, AMOUNT and CURRENCY are not dimension attributes, so the common dimension attributes between SALES and ct_SALES are $\{$PROD_SKU, BRAND, YEAR, CITY, STATE, COUNTRY$\}$.

Figure 2.9 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 relationships extracted from the analytic scheme while edges with bold dashed lines are user-defined relationships. For instance, a designer has defined a join relationship between dimensions $STORE$ and $WAREHOUSE$ with common attributes CITY, STATE and COUNTRY which did not exist in the analytic schema.

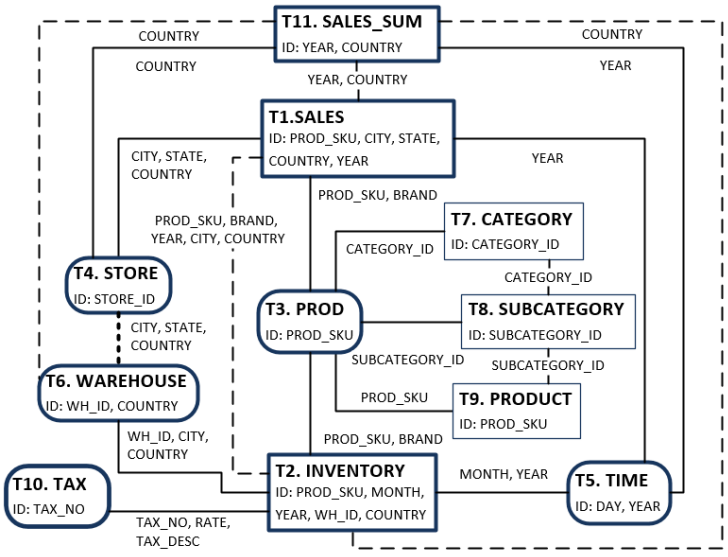


Figure 2.9: Examples of relationships

2.3.2 Derived relationships

It is possible to derive new relationships by the *composition* and the *fusion* of existing relationships. For example, in Figure 2.9, there is no direct relationship (solid-line edge) between fact tables INVENTORY and SALES. However, both tables share dimension tables $PROD$ and $TIME$ which can be composed to derive a new relationship represented by the dashed-line edge.

Proposition 2.6. *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 $\mathrm{Y}_3 = \mathrm{Y}_1 \cap_D \mathrm{Y}_2 \neq \emptyset$, 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 see **??** on **??**.

Example 2.24. In Figure 2.9, relationships R (SALES_SUM, $STORE$) and R ($STORE$, $WAREHOUSE$) are composed to yield R (SALES_SUM, $WAREHOUSE$) with common attribute COUNTRY. Similarly, R (SALES_SUM, $WAREHOUSE$) and R ($WAREHOUSE$, INVENTORY) are composed to generate R (SALES_SUM, INVENTORY) with common attribute COUNTRY.

The fusion of two different relationships between the same pair of tables can produce a new relationship between the same tables.

Proposition 2.7. *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 $\forall \mathrm{A} \in \mathrm{Y}_1 \cap_D \mathrm{Y}_2$, $\mu_{\mathsf{R}_1}(A) = \mu_{\mathsf{R}_2}(A)$ 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 see **??** on **??**.

Given a set of well-formed relationships $\mathsf{R}_0(T_1, T_2), \cdots, \mathsf{R}_n(T_1, T_2), n \geq 0$ between two tables T_1, T_2 with respective common attributes $\mathrm{Y}_0, \cdots, \mathrm{Y}_n$. We refer the union of $\mathrm{Y}_0, \cdots, \mathrm{Y}_n$ as the *common attributes of* T_1 *and* T_2.

Example 2.25. In Figure 2.9, relationships R(SALES, $PROD$) and R($PROD$, INVENTORY) are composed to yield R_1(SALES, INVENTORY) with common attributes {PROD_SKU, BRAND}. Similarly, R(SALES, $TIME$) and R($TIME$, INVENTORY) are composed to generate R_2(SALES, INVENTORY) with common attribute {YEAR}. Finally, R(SALES, $STORE$), R($STORE$, $WAREHOUSE$) and R($WAREHOUSE$, INVENTORY) are composed to produce R_3(SALES, INVENTORY) with common attributes {CITY, COUNTRY}. Then the fusion of R_1(SALES, INVENTORY), R_2(SALES, INVENTORY) and R_3(SALES, INVENTORY) yields R'(SALES, INVENTORY) with common attributes {YEAR, PROD_SKU, BRAND, CITY, COUNTRY}. However, the fusion of the two relationships R_1(ORDER, $TIME$) and R_2(ORDER, $TIME$) in Example 2.23 is not possible because $\mu_{\mathsf{R}_1}(\text{DAY}) \neq \mu_{\mathsf{R}_2}(\text{DAY})$ (both relationships map two different attributes ORDER.ORDER_DAY and ORDER.SHIP_DAY to the same attribute TIME.DAY.

Besides the composition and fusion of relationships, we consider a special case of derived relationships that are defined over *lookup tables*.

Definition 2.15 (Lookup Table Relationships)**.** Let $T_1(\mathrm{S}_1), T_2(\mathrm{S}_2), T(\mathrm{S})$ be three tables such that there exist two well-formed relationships $\mathsf{R}_1(T_1, T)$ and $\mathsf{R}_2(T_2, T)$ with respectively common attributes Y_1 and Y_2. A *relationship* $\mathsf{R}(T_1, T_2)$ over the table $T(\mathrm{S})$ is a non-empty set of attribute mappings from T_1 to T_2 using the mapping function $f(x) : T_1.\mathrm{Y}_1 = \pi_{\mathrm{Y}_1}(\sigma_{\mathrm{Y}_2=x}(T)), x \in dom(T_2.\mathrm{Y}_2)$. Table T is referred to as a *lookup table*.

Proposition 2.8. Let $T_1(\mathrm{S}_1), T_2(\mathrm{S}_2), T(\mathrm{S})$ be three tables such that there exist two well-formed relationships $\mathsf{R}_1(T_1, T)$ and $\mathsf{R}_2(T_2, T)$ with respectively common attributes Y_1 and Y_2. Let $\mathsf{R}(T_1, T_2)$ be the relationship over the *lookup-table* $T(\mathrm{S})$, if $\mathrm{Y}_1 \mapsto \mathrm{Y}_2$ and $\mathrm{Y}_2 \mapsto \mathrm{Y}_1$ hold in T, then $\mathsf{R}(T_1, T_2)$ is a one-to-one mapping relationship.

Proof. We prove by contradiction that assume $\mathsf{R}(T_1, T_2)$ is not a one-to-one mapping relationship. Let $f(x) = T_1.\mathrm{Y}_1, x \in dom(T_2.\mathrm{Y}_2)$ be the mapping function of $\mathsf{R}(T_1, T_2)$. Let t be a tuple of T_1, then there exist two tuples t_a, t_b in T_2 such that $t.\mathrm{Y}_1 = f(t_a.\mathrm{Y}_2) = f(t_b.\mathrm{Y}_2)$ and $t_a \not\equiv t_b$. By $\mathsf{R}_1(T_1, T)$ and $\mathsf{R}_2(T_2, T)$, we now that there exist two tuples t'_a, t'_b in T such that $t'_a.\mathrm{Y}_1 \equiv t'_b.\mathrm{Y}_1 \equiv t.\mathrm{Y}_1$, $t'_a.\mathrm{Y}_2 \equiv t_a.\mathrm{Y}_2$ and $t'_b.\mathrm{Y}_2 \equiv t_b.\mathrm{Y}_2$. Because $\mathrm{Y}_1 \mapsto \mathrm{Y}_2$ holds in T, we obtain $t'_a.\mathrm{Y}_2 \equiv t'_b.\mathrm{Y}_2$ which contradicts the assumption that $t_a \not\equiv t_b$. Therefore, when $\mathrm{Y}_1 \mapsto \mathrm{Y}_2$ and $\mathrm{Y}_2 \mapsto \mathrm{Y}_1$ hold in T, the relationship $\mathsf{R}(T_1, T_2)$ is a one-to-one mapping relationship. □

Example 2.26. Consider the two dimension tables $STORE$ and $WAREHOUSE$ in Figure 2.9. Let COUNTRY_CODE (COUNTRY_NAME, 2_CHAR, 3_CHAR, UN_CODE) be a table storing country names of different formats including the full name, 2-char code, 3-char code and a three-digit numeric code (ISO 3166-1 [1]). Every attribute in COUNTRY_CODE forms a key (identifier). Suppose we have $\mathsf{R}(\text{COUNTRY\_CODE}, STORE)$ and $\mathsf{R}(\text{COUNTRY\_CODE}, WAREHOUSE)$ be two well-formed relationships respectively of the form COUNTRY_CODE.COUNTRY_NAME $\twoheadrightarrow WAREHOUSE$.COUNTRY, COUNTRY_CODE.2_CHAR $\twoheadrightarrow STORE$.COUNTRY. Then there exists a relationship $\mathsf{R}(STORE, WAREHOUSE)$ over the table COUNTRY_CODE where attribute mappings is $STORE.\text{COUNTRY} = f(WAREHOUSE.\text{COUNTRY})$ and the mapping function f is defined as $f(x) = \pi_{\text{2\_CHAR}}(\sigma_{\text{COUNTRY\_NAME}=x}(\text{COUNTRY\_CODE}))$, $x \in dom(WAREHOUSE.\text{COUNTRY})$. Because COUNTRY_NAME $\mapsto$ 2_CHAR and 2_CHAR $\mapsto$ COUNTRY_NAME hold in COUNTRY_CODE, the relationship $\mathsf{R}(STORE, WAREHOUSE)$ is a one-to-one mapping relationship.

2.3.3 Relationships in drill-across OLAP queries

In the context of "drill-across" queries, [6], [30] focus on relationships between fact tables through *conforming dimensions* and [31] studies *Fact-to-Dimension* (FD) relationships, *Fact-to-Fact* (FF) relationships and *Dimension-to-Dimension* (DD) relationships. We now explain how these three kinds of relationships are represented using our definitions.

Fact-to-Fact (FF) relationships. When a fact table T is defined over another fact table T_0, or T and T_0 are defined over a common dimension D, there exists, an FF *attribute-mapping* relationship between table T and T_0. The dimension D is called *conformed dimension* [6] between T and T_0. For example, fact table SALES_SUM is defined as a view that computes the SUM of *AMOUNT* group by COUNTRY, MONTH from SALES. An attribute-mapping (FF) relationship between SALES and SALES_SUM is extracted from the view definition with common attributes COUNTRY, MONTH. Since SALES and INVENTORY have dimensions $PROD$ and $TIME$ in common, there also exists an attribute-mapping (FF) relationship between SALES and INVENTORY through conformed dimensions $PROD$ and $TIME$.

[1] https://en.wikipedia.org/wiki/Country_code

Dimension-to-Dimension (DD) relationships. Dimensions can also be defined over other dimension tables, and thus DD attribute-mapping relationships can be extracted from the view definition of dimensions. For example, dimension table ALL_STORES is a view defined as a union of two dimensions that have the same dimension attributes:

$$ALL\_STORES = CHAIN\_STORES \cup INDEP\_STORES$$

Here, an attribute mapping (DD) relationship between ALL_STORES and each operand dimension of the union is extracted from the view definition. DD Join relationship can also be explicitly defined by the user. As shown in Figure 2.9, the user has defined three natural join predicates on attributes CITY, STATE and COUNTRY, which are shared between dimensions $STORE$ and $WAREHOUSE$. These dimensions are not “conformed”, but their relationship defines a so-called "integrated" dimension in [30].

Since we also take non-analytic tables into consideration, there are other types of relationships involving non-analytic tables that were not discussed in [30] and [1].

Analytic-to-Non-analytic (AN) relationships. The view definitions of dimension and fact tables also refer to non-analytic tables and define AN attribute-mapping relationships between dimension or fact tables and non-analytic tables. For example, dimension $PROD$, introduced in Example 2.21, defined an attribute mapping relationship between $PROD$ and three non-analytic tables.

Non-analytic to Non-analytic (NN) relationships. NN Join relationship can be inferred from foreign key (PK-FK) constraints between non-analytic tables. For example, in the definition of non-analytic dataset ct_SALES, PROD_SKU is the primary key of the non-analytic dataset PRODUCT and a NN join relationship between ct_SALES and PRODUCT is extracted with common attribute PROD_SKU.

2.4 Conclusions

To conclude this chapter, we summarize the data and related metadata introduced in this chapter. Dimensions and measures are two standard notions which frequently appear in analytic data models. In our data model, we take the general definitions

of dimension tables, fact tables and hierarchy as introduced in [6]. We do not consider data evolution or *time-dependent dimensions* that track historical data with the help of historical tables or timestamp attributes, as it was done for instance in [21]. We also assume that all data is valid and accessible until it is deleted. We release certain constraints on the hierarchy structure and table identifiers and enable non-linear hierarchies with multiple bottom and top level attributes. We also release the restriction that the bottom level attribute in a hierarchy is the dimension identifier. We extend the previous models and make them more flexible through three new notions: (1) *attribute graphs* which represent LFDs over dimension attributes; (2) *dimension / fact identifiers* for analytic tables with null-values; (3) *aggregable properties* which describe and check the validity of aggregation operations on fact tables.

Table 2.6 lists the main metadata we need 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 4.2.1 that attribute graphs can also be automatically computed from complete or sampled dimension tables. This substantially reduces the human-effort for enabling schema augmentations and complements to defining analytic schemas and aggregable properties.

Table 2.6: 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
Relationships	User-defined or computed

3

Schema Augmentations and Quality Guarantees

Contents

In this chapter, we present our methods that deal with the challenges introduced in Section 1.2. We first introduce the notion of *schema augmentation* and *merge query* in Section 3.1, then in Section 3.2 we present *natural schema complements* which are a subclass of schema augmentations without *row multiplication*. Row multiplication can be avoided by applying *reduction queries* (Section 3.3) which are a new feature with respect to existing work in schema complements (see Related Work Chapter 5). In Section 3.4, we introduce our solutions for tackling the quality problems of *incorrect and ambiguous reduction* and *incomplete merge*. We introduce a formal framework for ensuring the quality of the merge results and show how these quality guarantees relate to the notion of *summarizability*.

3.1 Schema Augmentations

We first introduce a general case which is to augment a table with new attributes from other tables, this augmentation is called *schema augmentation*.

Definition 3.1 (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 $\mathrm{Y} = \mathrm{S}_0 \cap_D \mathrm{S}$. Then table T is a *schema augmentation* to *source table* T_0 with respect to R.

Schema augmentations are easy to find using the well-formed relationships. By definition, two tables are mutual schema augmentation to each other when they are related through a well-formed relationship and their common attributes Y is not empty.

Example 3.1. Consider the tables and the relationships shown in Figure 2.9. Table INVENTORY is related to table SALES through a relationship with common attributes {PROD_SKU, BRAND, YEAR, CITY, COUNTRY}. Therefore, SALES and INVENTORY are mutual schema augmentation to each other.

Schema augmentations are used to extend one of the two related tables with new attribute values from the other table. This extension is computed through merge queries defined as follows:

Definition 3.2 (Merge query)**.** Let $T(\mathrm{S})$ be a schema augmentation to an analytic table $T_0(\mathrm{S}_0)$ with respect a relationship R and a set of common attributes Y. Then the *merge* of T_0 and T is a *left-outer join query* $\mathcal{Q} = \Pi_{\mathrm{X}}(T_0 \rtimes_{P_1 \wedge \cdots \wedge P_k} T)$, where $k = |Y|$ is the number of common attributes of R, Π is a duplicate elimination projection and the following conditions hold:

1. For each $\mathrm{A}_i \in \mathrm{Y}$, $\exists P_i$ such that $P_i = (T_0.\mathrm{A}_i = T.\mathrm{A}_i) \vee (T_0.\mathrm{A}_i = null \wedge T.\mathrm{A}_i = null)$ (marked $null$ is literal value).
2. If all common attributes are dimension attributes, and there exists a pair of common attributes $\mathrm{A}_1, \mathrm{A}_2 \in \mathrm{Y}$ in T_0 such that $T_0.\mathrm{A}_1 \preccurlyeq T_0.\mathrm{A}_2 \wedge T.\mathrm{A}_1 \not\preccurlyeq T.\mathrm{A}_2$, then $\mathrm{X} = \mathrm{S}_0 \uplus \mathrm{S}$ else $\mathrm{X} = \mathrm{S}_0 \cup \mathrm{S}$ ($\uplus$ denotes disjoint union).

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

Item 1 manages the join predicates in the merge query when there is presence of nulls, because equality in standard SQL semantics is undefined for null values. Item 2 checks if the structure of the hierarchy in T_0 is preserved after merging common attributes. When the dimensions of common attributes are not compatible (do not mutually preserve the hierarchical relationships), the merge query keeps the common attributes separately for each table and applies disjoint union.

In the following, we will refer to the result of a merge query through a schema augmentation as an *augmented merge table.*

Example 3.2. Considering tables in Figure 2.9 on Page 37, table INVENTORY is a schema augmentation to SALES with respect to common attributes: Y= {PROD_SKU, BRAND, YEAR, CITY, COUNTRY}. After comparing the schema of these two tables, the merge query can add some or all new attributes from table INVENTORY (MONTH, WH_ID, TAX_NO, RATE, TAX_DESC and *QTY_ON_HAND*) to table SALES and vice-versa (*AMOUNT*).

SALES(PROD_SKU, BRAND, YEAR, CITY, STATE, COUNTRY, *AMOUNT*)
INVENTORY(PROD_SKU, BRAND, MONTH, YEAR, WH_ID, CITY,
COUNTRY, TAX_NO, RATE, TAX_DESC, *QTY_ON_HAND*)

Formally, a merge query between SALES and INVENTORY through schema augmentation is expressed as $\text{SALES} \rtimes\!\!\ltimes_{\text{Y}} \text{INVENTORY}$ where Y is the set of common attributes, and the augmented merge table does not duplicate the common attributes. The augmented merge table preserves all rows in SALES (with possible row duplication) and contains all attributes in INVENTORY and SALES (the merge query does not apply a projection). The symmetric merge query $\text{INVENTORY} \rtimes\!\!\ltimes_{\text{Y}} \text{SALES}$ preserves all tuples in table INVENTORY and has the same schema.

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

Proposition 3.1. Let $T'_0(\text{S}'_0)$ be a merge of table $T_0(\text{S}_0)$ with a target schema augmentation $T(\text{S})$. Let K be the identifier of T_0 and $\text{S}_{new} \subseteq \text{S}'_0 - \text{S}_0$ be the set of dimension attributes added to T_0 in the merge. Then, for all minimal subsets $\text{K}_{new} \subseteq \text{S}_{new}$ where $\text{K}_{new} \mapsto \text{S}_{new}$, $\text{K} \cup \text{K}_{new}$ is an identifier of T'_0.

Proof. Let $\text{K}' = \text{K} \cup \text{K}_{new}$. We prove $\text{K}' \mapsto \text{S}'_0$ by contradiction. Assume that K' is not the identifier of T'_0, then there exist two tuples $t_1, t_2 \in T'_0$ and an attribute $\text{B} \in \text{S}_0$ such that $t_1.\text{K}' \equiv t_2.\text{K}'$ but $t_1.\text{B} \not\equiv t_2.\text{B}$. We distinguish between two cases:

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

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

Therefore, we conclude $\text{K}' \mapsto \text{S}'_0$.

□

Example 3.3. Consider the merged result SALES' in Example 3.2. Then, the identifier of SALES is K = {PROD_SKU, MONTH, YEAR, CITY, STATE, COUNTRY} and S_{new} = {MONTH, WH_ID, TAX_NO, RATE, TAX_DESC}. We have $\mathrm{K}_{new} = \{\text{MONTH}, \text{WH\_ID}, \text{TAX\_NO}\}$ such that $\mathrm{K}_{new} \mapsto \mathrm{S}_{new}$. Thus, the identifier of SALES' is $\mathrm{K} \cup \mathrm{K}_{new}$ ={PROD_SKU, MONTH, YEAR, CITY, STATE, COUNTRY, WH_ID, TAX_NO}.

3.2 Natural Schema Complement

Merge queries might join one tuple in the source table with several tuples in the schema augmentation table. This tuple duplication might lead to unexpected results. For example, in the merge results of Example 3.2, tuples in SALES will be duplicated because of the newly added dimension attributes MONTH, WH_ID. As we explained in Section 1.2, row multiplication may produce unexpected or erroneous results. For example, an aggregation query that computes the sum of quantity-on-hand over all warehouses will produce a wrong result when it is applied on table SALES' after the augmentation of SALES with table $\mathrm{INVENTORY}$. 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 a source table.

The notion of *schema complement* that merges two tables and brings new attributes without row multiplications, which solves our second challenge "row multiplication" in Section 1.2, was initially proposed in [12]. In this section, we explain our extensions of the original definition of schema complement by considering LFDs and well-formed relationships between tables.

Definition 3.3 (Natural schema complement)**.** Let $T(\mathrm{S})$ be a schema augmentation of $T_0(\mathrm{S}_0)$ with a set of common attributes $\mathrm{Y} = \mathrm{S}_0 \cap_D \mathrm{S}$. Table T is a *natural schema complement* to a table T_0 with respect to R if $\mathrm{Y} \mapsto \mathrm{S}$.

The merge of T_0 and T through natural schema complement follows Definition 3.2 for merging schema augmentations. We refer to the result of a merge query through a natural schema complement by *natural merge table*.

Proposition 3.2. Let table $T(\mathrm{S})$ be a *natural schema complement* of a table $T_0(\mathrm{S}_0)$ with respect to a set of common attributes $\mathrm{Y} = \mathrm{S} \cap_D \mathrm{S}_0$, let $T'_0(\mathrm{S}'_0)$ be the natural merge table of T and T_0, then for each tuple t in T_0 there exists exactly one tuple t' in T'_0 such that $t.\mathrm{S}_0 \equiv t'.\mathrm{S}_0$.

Proof. The proof for this proposition is straightforward.

We first show that if K is an identifier in T_0, then it is also an identifier in T_0'. Let $S_{new} = S_0' - S_0$ be the set of attributes added to T_0 in the merge, then by Proposition 3.1, the identifier of T_0' is $K \cup K_{new}$ where $K_{new} \mapsto S_{new}$. By Definition 3.3, $Y \mapsto S$ and $K \mapsto Y$, by transitivity we get $K \mapsto S$. Because $S_{new} \subseteq S$, we get $K \mapsto S_{new}$. Finally, $K \mapsto S_0 \cup S_{new} = S_0'$, K is the identifier of T_0'.

Second, we show that no tuples in T_0 are lost in T_0'. This is guaranteed by the definition of merge query which uses left-outer join for augmenting table T_0. □

By Propositions 3.1 and 3.2, we now know that when T is a *natural schema complement* of a table T_0, the identifier of T_0 is also the identifier of the natural merge table of T_0 and T.

Example 3.4. Consider the tables and relationships shown in Figure 2.9 (Page 37), the non-analytical table PRODUCT is a schema augmentation to dimension table $PROD$ with respect to two common attributes {PROD_SKU, BRAND}. It is also a natural schema complement since the common attribute PROD_SKU is a primary key in PRODUCT. By comparing their schemas, PRODUCT can bring new details to $PROD$ like WEIGHT and PRODUCT_NAME.

PRODUCT{PROD_SKU, PRODUCT_NAME, BRAND, WEIGHT, SUBCATEGORY_ID}
$PROD$ {PROD_SKU, BRAND, SUBCATEGORY, CATEGORY}

The natural merge $PROD \rtimes_{\text{PROD\_SKU}}$ PRODUCT produces exactly one tuple for each tuple in dimension $PROD$

Considering tables INVENTORY and SALES of Example 3.1, their common attributes do not contain an identifier of SALES. Therefore SALES is not a natural schema complement to INVENTORY.

3.3 Reduction Queries

When T is a schema augmentation but not a natural schema complement of T_0 (the common attributes do not literally determine all attributes of T), it is still possible to transform T into a natural schema complement by applying some query. In this section, we introduce *reduction operations* which "reduce" the attributes in the identifier of T.

Definition 3.4 (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 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*.

Since $\mathrm{K}' \subset \mathrm{K}$, every partitioning (group-by) of T by K' contains one or more tuples. The effect of a reduction query $Q(T)$ on K' is to reduce each partition into a single tuple.

Example 3.5. Consider the fact table SALES with identifier $\mathrm{K} = \{$PROD_SKU, YEAR, CITY, STATE, COUNTRY$\}$. A query $\mathrm{SALES}' = Q(\mathrm{SALES})$ that filters COUNTRY = "USA" is a reduction of SALES on attributes $\mathrm{K}' = \mathrm{K} - \{$COUNTRY$\}$. K' is the identifier of SALES' and COUNTRY is the reduced attribute.

There exists a great variety of reduction queries. In our work, we focus on three common types of reduction queries: aggregate, filter and pivot.

Definition 3.5 (Aggregate reduction)**.** Let $T(\mathrm{S})$ be an analytic table with dimension attributes $\mathrm{S}_D \subseteq \mathrm{S}$ and identifier $\mathrm{K} \subseteq \mathrm{S}_D$, A be an aggregable attribute in S, and F be an aggregation function such that aggregable property $\mathbf{agg}_{\mathrm{A}}(\mathrm{F}, \mathrm{Z})$ holds in T (A can be aggregated along any subset of Z). We denote by $Q(T) = \mathit{Agg}_T(\mathrm{F}(\mathrm{A}) \mid \mathrm{X})$ where $\mathrm{S}_D - \mathrm{Z} \subseteq \mathrm{X} \subseteq \mathrm{S}_D$, an aggregate query on table T that aggregates A using aggregation function F with group-by attributes X. We call a query $Q(T) = \mathit{Agg}_T(\mathrm{F}(\mathrm{A}) \mid \mathrm{K}')$ an *aggregate reduction* of T on attributes K' when $\mathrm{K}' \subset \mathrm{K}$, and call $\mathrm{K} - \mathrm{K}'$ the reduced attributes.

Note that the SQL group-by operator implements literal equality semantics for $null$ values ($null$ values are not distinct).

Example 3.6. Consider a table $T(\mathrm{S})$ in Table 3.1a with dimensional attributes $\mathrm{A}_1, \mathrm{A}_2$ from D_1 and A_3 from dimension D_2, and $\{\mathrm{A}_1, \mathrm{A}_3\}$ forming an identifier K of T. Suppose that the aggregable property $\mathbf{agg}_M(\mathrm{SUM}, \{\mathrm{A}_1, \mathrm{A}_2, \mathrm{A}_3\})$ holds in T. The result of $Q_1 = \mathit{Agg}_T(\mathrm{SUM}(M) \mid \{\mathrm{A}_3\})$ is shown in Table 3.1b. Q_1 reduces the identifier of T by dropping A_1 from the group-by attributes. Thus, Q_1 is an aggregate reduction of T on $\{\mathrm{A}_3\}$, the identifier is $K' = \{\mathrm{A}_3\}$ in the result table, and attribute A_1 is the reduced attribute. The result of another aggregate query $Q_2 = \mathit{Agg}_T(\mathrm{SUM}(M) \mid \{\mathrm{A}_1, \mathrm{A}_3\})$ is shown in Table 3.1c. However, since Q_2 does not reduce the identifier of T, Q_2 is not an aggregate reduction.

Table 3.1: Examples of aggregate queries

(a) Input table T

T	$\underline{\text{A}_1}$	A_2	$\underline{\text{A}_3}$	M
	a_1	b_1	c_1	x_1
	a_1	b_1	c_2	x_2
	a_2	b_1	c_1	x_3
	a_2	b_2	c_2	x_4

(b) Aggregate $\mathcal{Q}_1$

$\mathcal{Q}_1$	$\underline{\text{A}_3}$	$\text{SUM}(M)$
	c_1	$x_1 + x_3$
	c_2	$x_2 + x_4$

(c) Aggregate $\mathcal{Q}_2$

$\mathcal{Q}_2$	$\underline{\text{A}_1}$	$\underline{\text{A}_3}$	$\text{SUM}(M)$
	a_1	c_1	x_1
	a_1	c_2	x_2
	a_2	c_1	x_3
	a_2	c_2	x_4

Definition 3.6 (Filter reduction)**.** Let $T(\text{S})$ be a table with identifier $\text{K} \subseteq \text{S}$. We denote by $\mathcal{Q}(T) = \mathit{Filter}_T(P \mid \text{X})$, $P = \{P_1 \wedge \cdots \wedge P_i\}$, a filter query that filters T by a set of predicates P on attributes X of T. We call query $\mathcal{Q}(T) = \mathit{Filter}_T(P \mid \text{K} - \text{K}')$ a *filter reduction* of T on attributes K', when $\text{K}' \subset \text{K}$ and for each attribute $\text{A} \in \text{K} - \text{K}'$, there exists a predicate $P_i \in P$ of the form $\text{A} = v_i, v_i \in dom(\text{A})$, and $\text{K} - \text{K}'$ are called the reduced attributes.

Example 3.7. Reconsider the table $T(\text{S})$ in Table 3.1a with key $K = \{A_1, A_3\}$. The result of the filter query $\mathcal{Q}_3 = \mathit{Filter}_T(\{\text{A}_1 = \text{`}a_1\text{'}\} \mid \{\text{A}_1\})$ is shown in Table 3.2a. $\mathcal{Q}_3$ reduces the identifier of T by applying a filter on A_1. $\mathcal{Q}_3$ is a filter reduction of T on $\{\text{A}_3\}$. The identifier of the result table is $\{\text{A}_3\}$ and attribute A_1 is the reduced attribute. The result of another filter query $\mathcal{Q}_4 = \mathit{Filter}_T(\{\text{A}_2 = \text{`}b_1\text{'}\} \mid \{\text{A}_1\})$ is shown in Table 3.2b. A_2 is not in the identifier of T and the identifier remains the same after applying $\mathcal{Q}_4$. $\mathcal{Q}_4$ is not a filter reduction.

Table 3.2: Examples of filter queries

(a) Filter $\mathcal{Q}_3$

$\mathcal{Q}_3$	A_1	A_2	$\underline{\text{A}_3}$	M
	a_1	b_1	c_1	x_1
	a_1	b_1	c_2	x_2

(b) Filter $\mathcal{Q}_4$

$\mathcal{Q}_4$	$\underline{\text{A}_1}$	A_2	$\underline{\text{A}_3}$	M
	a_1	b_1	c_1	x_1
	a_1	b_1	c_2	x_2
	a_2	b_1	c_1	x_3

Definition 3.7 (Pivot reduction)**.** Let $T(\text{S})$ be a table with identifier $\text{K} \subseteq \text{S}$ and A be an attribute in S. We denote by $\mathcal{Q}(T) = \mathit{Pivot}_T(\text{A} \mid \text{X})$, where $\text{X} \subset \text{S} - \{\text{A}\}$, a pivot query which pivots attribute A over X. The result is a table T', with identifier $\text{K} - \text{X}$, whose schema contains every attribute of $\text{S} - \{\text{X}, \text{A}\}$ and an attribute $\text{A\_}v_x$ for each value v_x in the domain of $T.\text{X}$. The value $t.\text{A}$ of each tuple $t \in T$ such that $t.\text{X} = v_x$ is a value in the attribute $\text{A\_}v_x$ of the unique tuple t' in T' such that $t.(\text{S} - \{\text{X}, \text{A}\}) = t'.(\text{S} - \{\text{X}, \text{A}\})$. We call query $\mathit{Pivot}_T(\text{A} \mid \text{K} - \text{K}')$ a *pivot reduction* of T on attributes K' when $\text{K}' \subset \text{K}$, and $\text{K} - \text{K}'$ are called the reduced attributes.

Example 3.8. Reconsider the table $T(\mathrm{S})$ in Table 3.1a. The result of pivot query $Q_5 = \mathit{Pivot}_T(M \mid \mathrm{A}_1)$ that pivots attribute M over A_1 is shown in Table 3.3a. The schema of the resulting table T', with identifier $\mathrm{K} - \mathrm{A}_1$, has every attribute of $\mathrm{S} - \mathrm{A}_1$ and one attribute M_v for each value v of $T.\mathrm{A}_1$, that is, two new attributes M_a_1, M_a_2. The value $t.M$ of each tuple $t \in T$ such that $t.\mathrm{A}_1 = v$ is a value in the attribute M_v of the unique tuple t' in T' such that $t.(\{\mathrm{A}_2, \mathrm{A}_3\}) = t'.(\{\mathrm{A}_2, \mathrm{A}_3\})$. Since $\mathrm{A}_1 = \mathrm{K} - \{\mathrm{A}_3\}$, Q_5 is a pivot reduction of T on attribute $\{\mathrm{A}_3\}$, the identifier of the result is $\{\mathrm{A}_3\}$ and attribute A_1 is the reduced attribute. The result of another pivot query $Q_6 = \mathit{Pivot}_T(M \mid \mathrm{A}_2)$ is shown in Table 3.2b. Since $\mathrm{A}_2 \notin \mathrm{K}$, Q_6 is not a pivot reduction.

Table 3.3: Examples of pivot queries

(a) Pivot Q_5

Q_5	A_2	$\underline{\mathrm{A}_3}$	M_a_1	M_a_2
	b_1	c_1	x_1	x_3
	b_1	c_2	x_2	x_4

(b) Pivot Q_6

Q_6	$\underline{\mathrm{A}_1}$	$\underline{\mathrm{A}_3}$	M_b_1	M_b_2
	a_1	c_1	x_1	-
	a_1	c_2	x_2	-
	a_2	c_1	x_3	-
	a_2	c_2	-	x_4

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

We now establish the condition under which a query over T, consisting of a sequence of nested reduction operations, forms a reduction query that computes a schema complement to a given table.

Proposition 3.3. 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 and K is an identifier of T. Let $Q(T) = Q_n(Q_{n-1}(\ldots(Q_1(T))\ldots)), n > 0$ be a sequence of nested queries applied on T such that query Q_i is a reduction query on attributes Y_i. When $\mathrm{Y}_n \subseteq \mathrm{Y}$, then $Q(T)$ is a reduction query of T on Y_n and $Q(T)$ is a natural schema complement to T_0.

Proof. We prove by induction.

$n = 1$: Let $T_1(\mathrm{S}_1)$ be the table produced by $Q(T) = Q_1(T)$, we prove that T_1 is a natural schema complement to T_0. Because $Q_1(T)$ is a reduction query on Y_1, by Definition 3.4, we have Y_1 is the identifier of T_1 and $\mathrm{Y}_1 \subset \mathrm{K}$. When $\mathrm{Y}_1 \subseteq \mathrm{Y}$, since $\mathrm{Y}_1 \mapsto \mathrm{S}_1$ we also have $\mathrm{Y} \mapsto \mathrm{S}_1$, so $T_1(\mathrm{S}_1)$ is a natural schema complement of $T_0(\mathrm{S}_0)$.

Induction step: Let $T_n(\mathrm{S}_n)$ be the table produced by $\mathcal{Q}(T) = \mathcal{Q}_n(\mathcal{Q}_{n-1}(\ldots(\mathcal{Q}_1(T))\ldots))$, assume that T_n is a natural schema complement to T_0, that is, $\mathrm{Y} \mapsto \mathrm{S}_n$.

Let $T_{n+1}(\mathrm{S}_{n+1})$ be the table produced by $\mathcal{Q}_{n+1}(T_n)$, we prove that T_{n+1} is a natural schema complement to T_0 following the same reasoning as before. Because $\mathcal{Q}_{n+1}$ is a reduction query on Y_{n+1}, we have: Y_{n+1} is the identifier of T_{n+1} and $\mathrm{Y}_{n+1} \subset \mathrm{Y}_n$ since Y_n is the identifier of T_n. When $\mathrm{Y}_{n+1} \subseteq \mathrm{Y}$, since $\mathrm{Y}_{n+1} \mapsto \mathrm{S}_{n+1}$ we also have $\mathrm{Y} \mapsto \mathrm{S}_{n+1}$, so $T_{n+1}(\mathrm{S}_{n+1})$ is a natural schema complement of $T_0(\mathrm{S}_0)$.

Q.E.D. $\mathcal{Q}(T)$ is a natural schema complement to T_0. □

Example 3.9. Fact tables SALES and INVENTORY in the relationship graph of Figure 2.9 have common attributes $\mathrm{Y} = \{$PROD_SKU, BRAND, YEAR, CITY, COUNTRY$\}$. INVENTORY is a schema augmentation (formally, all schema complements are also schema augmentations by definition) but not a natural schema complement to SALES, since Y is not a fact identifier of INVENTORY. We can then define several reduction queries that transform INVENTORY into a natural schema complement of $SALES$. The key of INVENTORY is $\mathrm{K} = \{$PROD_SKU, MONTH, YEAR, WH_ID, COUNTRY, TAX_NO$\}$ and each of the following reduction queries on attributes $\mathrm{K}' = \mathrm{Y} \cap \mathrm{K} = \{$PROD_SKU, YEAR, COUNTRY$\}$ reduces attributes $\mathrm{K} - \mathrm{K}' = \{$MONTH, WH_ID, TAX_NO$\}$:

- Aggregate reduction on measure attribute *QTY_ON_HAND*:

$$\mathcal{Agg}_{\mathrm{INVENTORY}}(\mathrm{AVG}(\text{QTY\_ON\_HAND}) \mid \{\text{PROD\_SKU, YEAR, COUNTRY}\})$$

- Filter reduction on MONTH, WH_ID and TAX_NO:

$$\mathit{Filter}_{\mathrm{INVENTORY}}(\text{MONTH} = \text{“Jan”} \wedge \text{WH\_ID} = \text{“1234”} \wedge \text{TAX\_NO} = \text{“abcd”})$$

- Pivot reduction on measure attribute QTY_ON_HAND:

$$\mathcal{Pivot}_{\mathrm{INVENTORY}}(\text{QTY\_ON\_HAND} \mid \{\text{MONTH, WH\_ID, TAX\_NO}\})$$

Finally, it is also possible to reduce table SALES into a natural schema complement of table INVENTORY using nested reduction queries. The identifier of SALES is $\mathrm{K} = \{$PROD_SKU, YEAR, CITY, STATE, COUNTRY$\}$ and we must reduce attributes $\mathrm{K} - \mathrm{Y} = \{$STATE$\}$. This can be done by a reduction query $\mathcal{Q}_1(\mathrm{SALES}) = \mathit{Filter}_{\mathrm{SALES}}(\{\text{STATE} =$ “Ohio”$\}$ that applies a filter reduction on SALES, which reduces attribute STATE. Or

a pivot reduction query $Q_2 = \mathcal{P}ivot_{Q_1(\text{SALES})}(\{\textit{AMOUNT}\} \mid \{\text{STATE}\})$ pivoting measure *AMOUNT* over attribute STATE, which reduces STATE.

3.4 Quality Criteria of Schema Augmentations

We explained in Section 1.2 that inaccuracies can occur when applying reduction queries and merging schema augmentations. In the following section, we introduce formal quality criteria for schema augmentations and show how these criteria are guaranteed by our system. We present different issues related to *summarizability and propagation of aggregable properties*, *ambiguous reduction* and *incomplete merge* and illustrate how these issues can be solved. Section 3.4.1 introduces the notion of summarizability for the correct propagation of aggregated attribute values. Sections 3.4.2 and 3.4.3 deal with the generation of *ambiguous* and *incomplete* attribute values during the construction of schema augmentations (and schema complements).

3.4.1 Propagation of aggregable properties

A first quality issue concerns determination of the aggregable properties of new attributes A added to the schema of a table T_0 after a merge with a table T. In particular, it is critical to infer the aggregable property of A that holds in the augmented table to avoid incorrect aggregations on that augmented table. This raises two problems:

- The first problem is to define the *aggregable properties* of new attributes which are computed by some reduction queries. 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.

Propagation of aggregable properties through reduction queries

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

1. If $\mathrm{A}' = \mathrm{A}$ is an attribute of T and F is applicable to A in T, F is also applicable to A' in T'.
2. If A' holds pivoted values of an attribute A of T and F is applicable to A in T, then F is also applicable to A' in T'.
3. If $\mathrm{A}' = \mathrm{F}(\mathrm{A})$ is the result of applying some aggregation function F over an attribute A in T, then the aggregate functions G that are applicable to A' are determined by the category of the co-domain of function F using Table 2.5.

Filter and pivot reduction operations do not change the type of the aggregable attribute, so an aggregable property that holds in T still holds in T' when T' is the result of a filter or pivot reduction of T. However, an aggregate reduction may change the category of the aggregable attribute. For example, while an attribute of category **NUM** in T is still of category **NUM** in T' when $\mathrm{F} = \mathrm{SUM}$, it becomes of category **STAT** when $\mathrm{F} = \mathrm{AVG}$. This change is detected using the classification in Table 2.5 (on Page 34).

The following example illustrates the previous case analysis.

Example 3.10. Suppose that an attribute $\mathrm{A}' = \mathrm{AVG}(\mathrm{A})$ contains values that are aggregated from attribute A using the average function AVG (case 2). According to Table 2.5, statistical functions like AVG and STDEV have the domain category **NUM**, the co-domain category **STAT** and the aggregation functions that can be applied on A' are COUNT, MIN and MAX. Assume now that attribute $\mathrm{A}' = \mathrm{COUNT}(\mathrm{A})$ contains values that are aggregated from attribute A using the function COUNT. Function COUNT has the domain category **NUM**, **DESC** or **STAT** and co-domain category **NUM**. By default, functions that can be applied on A' are SUM, AVG, COUNT, MIN and MAX.

The identification of all applicable aggregation functions F is not sufficient for defining the aggregable properties of some attribute A' in the result T' of a reduction query. To define the aggregable property $\mathbf{agg}_{\mathrm{A}'}(\mathrm{F}, \mathrm{X}')$ that holds for A' and function F, we must also determine the maximal set of attributes X' along which aggregation is correct. If A' is an attribute in the result of a filter of pivot reduction query, we define the following propagation rules for determining X'

Definition 3.8 (Propagation of aggregable properties with filter and pivot)**.** Let $T'(\mathrm{S}') = \mathcal{Q}(T)$ be the result of a filter or pivot reduction query $\mathcal{Q}(T)$ over an analytic table $T(\mathrm{S})$ and S_D be the set of dimension attributes in T. Then the aggregable properties of the attributes in T' are obtained as follows:

1. If $\mathcal{Q}$ is a filter reduction then $\mathrm{S}' = \mathrm{S}$, and for every aggregable attribute A if $\mathbf{agg}_{\mathrm{A}}(\mathrm{F}, \mathrm{X}), \mathrm{X} \subseteq \mathrm{S}_D$, holds in T then $\mathbf{agg}_{\mathrm{A}}(\mathrm{F}, \mathrm{X})$ also holds in T'.

2. If $\mathcal{Q}$ is a pivot reduction of the form $\mathcal{P}ivot_T(\mathrm{A} \mid \mathrm{Z}), \mathrm{Z} \subset \mathrm{S} - \{\mathrm{A}\}$, such that $\mathbf{agg}_{\mathrm{A}}(\mathrm{F}, \mathrm{X}), \mathrm{X} \subseteq \mathrm{S}_D$, holds in T then if A' is a new attribute that holds pivoted values of A, then $\mathbf{agg}_{\mathrm{A}'}(\mathrm{F}, \mathrm{X}')$ holds in T' where $\mathrm{X}' = \mathrm{X} - \mathrm{Z}$. Otherwise if A' is an attribute of $\mathrm{S}' \cap \mathrm{S}$ such that $\mathbf{agg}_{\mathrm{A}'}(\mathrm{F}, \mathrm{X}), \mathrm{X} \subseteq \mathrm{S}_D$, holds in T then $\mathbf{agg}_{\mathrm{A}'}(\mathrm{F}, \mathrm{X})$ also holds in T'.

Example 3.11. Consider fact table PRODUCT_LIST in Table 3.4. Attribute QTY has **NUM** values and can be summed along all dimension attributes except YEAR, *i.e.* $\mathbf{agg}_{\mathrm{QTY}}(\mathrm{SUM} \mid \mathrm{X})$ where $\mathrm{X} = \{\mathrm{PROD\_SKU}, \mathrm{BRAND}, \mathrm{COUNTRY}\}$ holds for attribute QTY.

Table 3.4: Table PRODUCT_LIST

PROD_SKU	BRAND	COUNTRY	YEAR	QTY
cz-tshirt-s	COCA COLA	United States	2017	5 000
cz-tshirt-s	COCA COLA	United States	2018	7 000
cz-tshirt-s	ZARA	Spain	2017	5 000
cz-tshirt-s	ZARA	Spain	2018	7 000
coca-can-33cl	COCA COLA	United States	2017	10 000

Let $T' = \mathcal{F}ilter_{\mathrm{PRODUCT\_LIST}}(\{\mathrm{YEAR} = \text{'2017'}\})$. Table T' has the same schema as table PRODUCT_LIST and, by rule 1 above, $\mathbf{agg}_{\mathrm{QTY}}(\mathrm{SUM} \mid \mathrm{X})$ still holds in T'.

Let $T' = \mathcal{P}ivot_{\mathrm{PRODUCT\_LIST}}(\mathrm{QTY} \mid \mathrm{BRAND})$ be a query producing two attributes QTY_COCACOLA and QTY_ZARA with values from attribute QTY. Then, by rule 2 above, $\mathbf{agg}_{\mathrm{QTY\_COCACOLA}}(\mathrm{SUM} \mid \mathrm{X}')$ and $\mathbf{agg}_{\mathrm{QTY\_ZARA}}(\mathrm{SUM} \mid \mathrm{X}')$ hold in T_2 where $\mathrm{X}' = \mathrm{X} - \{\mathrm{BRAND}\} = \{\mathrm{PROD\_SKU}, \mathrm{COUNTRY}\}$.

For attributes that are produced by an aggregate reduction query, we define the following propagation rule.

Definition 3.9 (Propagation of aggregable properties with aggregation)**.** Let $T(\mathrm{S})$ be an analytic table with dimension attributes $\mathrm{S}_D \subseteq \mathrm{S}$, and $\mathbf{agg}_{\mathrm{A}}(\mathrm{F}, \mathrm{X})$ be an aggregable property that holds in T. Let $T' = \mathcal{A}gg_T(\mathrm{F}(\mathrm{A}) \mid \mathrm{Z})$ where $\mathrm{S}_D - \mathrm{X} \subseteq \mathrm{Z}$,

and let G be a default applicable aggregate function defined for the co-domain of function F (Table 2.4). Then the aggregable properties of the attributes in T' are obtained as follows:

1. $\mathbf{agg}_{\mathrm{F(A)}}(\mathrm{G}, \mathrm{X}')$ holds for attribute $\mathrm{F}(A)$ in T' with $\mathrm{X}' = \mathrm{X} \cap \mathrm{Z}$.
2. for every attribute $A' \in Z$, if $\mathbf{agg}_{\mathrm{A}'}(\mathrm{F}, \mathrm{X})$ holds in T then $\mathbf{agg}_{\mathrm{A}'}(\mathrm{F}, \mathrm{X}')$ holds in T' with $\mathrm{X}' = \mathrm{X} \cap \mathrm{Z}$.

Example 3.12. Consider table PRODUCT_LIST in Example 3.11. Attribute PROD_SKU is of category **DESC** and can be aggregated with functions COUNT and COUNT_DISTINCT along all other dimension attributes. This is formalized by the aggregable properties $\mathbf{agg}_{\text{PROD\_SKU}}(\text{COUNT} \mid \mathrm{X})$ and $\mathbf{agg}_{\text{PROD\_SKU}}(\text{COUNT\_DISTINCT} \mid \mathrm{X})$ where $\mathrm{X} = \{\text{PROD\_SKU}, \text{BRAND}, \text{COUNTRY}, \text{YEAR}\}$. Now, let $\text{PRODUCT\_LIST\_COUNT} = \mathcal{Agg}_{PRODUCT\_LIST}(\text{COUNT}(\text{PROD\_SKU}) \mid \mathrm{Z})$ where $\mathrm{Z} = \{\text{BRAND}, \text{COUNTRY}, \text{YEAR}\}$. The result is shown in Table 3.5 (the aggregated attribute has been renamed into NB_PRODUCTS).

Table 3.5: Table PRODUCT_LIST_COUNT

NB_PRODUCTS	BRAND	COUNTRY	YEAR
1	COCA COLA	United States	2017
1	COCA COLA	United States	2018
1	ZARA	Spain	2017
1	ZARA	Spain	2018
1	COCA COLA	United States	2017

Suppose now that one wants to compute the total number of products per brand by summing NB_PRODUCTS by brand. Since COUNT does not eliminate duplicates, we obtain the same result as if we counted all products for each brand directly from table PRODUCT_LIST. Now suppose that we applied function COUNT_DISTINCT instead of COUNT to compute table PRODUCT_LIST_COUNT. Then it is easy to see that this time the sum over NB_PRODUCTS is different from counting the number of distinct products per brand directly from table PRODUCT_LIST. We study this issue in the following section.

Enforcing summarizable attributes in aggregable properties

Summarizability is a property that characterizes the equivalence between the aggregation results computed from an intermediate aggregated result and the results

directly obtained from the original table. We introduce the notion of summarizable attributes and show how new propagation rules for aggregable properties can be used to enforce the summarizability of aggregable attributes.

Definition 3.10 (Summarizable attribute)**.** Let $T(\mathrm{S})$ be an analytic table and A be an aggregable attribute such that $\mathbf{agg}_{\mathrm{A}}(\mathrm{F}, \mathrm{X})$ holds in T for an aggregate function F, where X is a set of dimension attributes in T. Let $T_1 = \mathcal{A}gg_T(\mathrm{F}(\mathrm{A}) \mid \mathrm{Z}_1), \mathrm{S} - \mathrm{X} \subseteq \mathrm{Z}_1$. If for any subset $\mathrm{Z}_2 \subset \mathrm{Z}_1$, there exists an applicable aggregate function G such that the equation

$$\mathcal{A}gg_{T_1}(\mathrm{G}(\mathrm{F}(\mathrm{A})) \mid \mathrm{Z}_2) = \mathcal{A}gg_T(\mathrm{F}(\mathrm{A}) \mid \mathrm{Z}_2)$$

holds, then A is said to be *summarizable* with respect to Z_1 and function F using function G.

Attribute summarizability is strongly related to the notion of distributivity of aggregation functions.

Definition 3.11 (Distributive aggregation function)**.** Let F be an aggregation function applicable to a set of domain values V. If for any partitioning $\mathrm{V}_1, \ldots, \mathrm{V}_n$, $n \geq 1$ of V, there exists an aggregate function G such that $\mathrm{F}(\mathrm{V}_1 \cup \ldots \cup \mathrm{V}_n) = \mathrm{G}(\mathrm{F}(\mathrm{V}_1) \cup \ldots \cup \mathrm{F}(\mathrm{V}_n))$ then F is said to be *distributive* using function G over (any partitioning of) V.

If F is *distributive* using function G over any subset of its domain, we say that F is *distributive* using function G. If $\mathrm{F} = \mathrm{G}$, we simply say that F is distributive. It is easy to show that functions $\mathrm{SUM}, \mathrm{MIN}$ and MAX are distributive and function COUNT is distributive using function SUM whereas function $\mathrm{COUNT\_DISTINCT}$ is not distributive using function SUM. Function AVG is distributive over sets containing only two elements or where all elements are identical.

Finally, we say that F is distributive using function G on attribute $T.\mathrm{A}$ with partitioning attributes Z if F is distributive using function G over any partitioning of $T.\mathrm{A}$ *defined by* Z *or any subset of* Z. The following proposition relates the definition of distributive functions to the notion of summarizable attributes.

Proposition 3.4 (Function distributivity and attribute summarizability)**.** Let $T(\mathrm{S})$ be an analytic table with dimension attributes $\mathrm{S}_D \subseteq \mathrm{S}$ and an aggregable attribute A such that $\mathbf{agg}_{\mathrm{A}}(\mathrm{F}, \mathrm{X})$ holds in T. If F is distributive using function G on attribute $T.\mathrm{A}$ with partitioning attributes $\mathrm{Z} \supseteq \mathrm{S}_D - \mathrm{X}$ then A is summarizable with respect to Z and function F using function G.

Proof. Suppose that $\mathbf{agg}_{\mathrm{A}}(\mathrm{F}, \mathrm{X})$ holds in T and $T_1 = \mathcal{A}gg_T(\mathrm{F}(\mathrm{A}) \mid \mathrm{Z}_1)$. To prove that A is summarizable with respect to Z_1 and F using function G, we prove that for any subset $\mathrm{Z}_2 \subset \mathrm{Z}_1$, the following equation holds:

$$\mathcal{A}gg_T(\mathrm{F}(\mathrm{A}) \mid \mathrm{Z}_2) = \mathcal{A}gg_{T_1}(\mathrm{G}(\mathrm{F}(\mathrm{A})) \mid \mathrm{Z}_2) \tag{3.1}$$

First, it is obvious that both tables T and T_1 contain the same Z_2 values and therefore, the result tables in Eq. (3.1) contain the same tuples with distinct Z_2 values. We now show that for each pair of tuples $t \in \mathcal{A}gg_T(\mathrm{F}(\mathrm{A}) \mid \mathrm{Z}_2)$ and $t' \in \mathcal{A}gg_{T_1}(\mathrm{G}(\mathrm{F}(\mathrm{A})) \mid \mathrm{Z}_2)$ where $t.\mathrm{Z}_2 = t'.\mathrm{Z}_2$, we also have $t.\mathrm{G}(\mathrm{F}(\mathrm{A})) = t'.\mathrm{F}(\mathrm{A})$. Let $x = t.\mathrm{Z}_2$ and $\mathrm{V}(x) = \sigma_{\mathrm{Z}_2=x}(T)$ be the partition of T on attributes Z_2 corresponding to the partition identifier x. Then, for tuple $t \in \mathcal{A}gg_T(\mathrm{F}(\mathrm{A}) \mid \mathrm{Z}_2)$ we obtain $t.\mathrm{F}(\mathrm{A}) = \mathrm{F}(\mathrm{V}.\mathrm{A})$. There also exists a set of tuples $t'_i \in T_1, i \geq 1$ where $t'_i.\mathrm{Z}_2 = x$. For each tuple t'_i, there exists a partition $\mathrm{V}_i = \sigma_{\mathrm{Z}_1=x}(T)$ of T on Z_1 attributes, such that $\mathrm{V} = \mathrm{V}_1 \cup \ldots \cup \mathrm{V}_n$ and $t'_1.\mathrm{F}(\mathrm{A}) = \mathrm{F}(\mathrm{V}_i.\mathrm{A})$ All these tuples t'_i have the same Z_2 value (by assumption) and will be aggregated to form a single tuple t' in $\mathcal{A}gg_{T_1}(\mathrm{F}(\mathrm{F}(\mathrm{A})) \mid \mathrm{Z}_2)$ whose value for attribute $\mathrm{G}(\mathrm{F}(\mathrm{A})) = \mathrm{G}(\mathrm{F}(\mathrm{V}_1.\mathrm{A}) \cup \ldots \cup \mathrm{F}(\mathrm{V}_n.\mathrm{A}))$. Since F is distributive using function G, this expression is equal to $\mathrm{F}(\mathrm{V}_1.\mathrm{A} \cup \ldots \cup \mathrm{V}_n.\mathrm{A}) = \mathrm{F}(\mathrm{V}.\mathrm{A})$. Thus, tuples t and t' are equal. □

Example 3.13. Recall the fact table PRODUCT_LIST in Table 3.4 of Example 3.13. Function COUNT is distributive using function SUM on attribute PROD_SKU with partitioning attributes $Z = \{$BRAND, COUNTRY, YEAR$\}$, therefore PROD_SKU is summarizable with respect to Z and COUNT using function SUM. Indeed, if $\mathrm{Z}_2 = \{$COUNTRY, YEAR$\}$ and $T_1 = \mathcal{A}gg_{PRODUCT\_LIST}(\mathrm{COUNT}(\text{PROD\_SKU}) \mid \mathrm{Z})$, the equation $\mathcal{A}gg_{PRODUCT\_LIST}(\mathrm{COUNT}(\text{PROD\_SKU}) \mid \mathrm{Z}_2) = \mathcal{A}gg_{T_1}(\mathrm{SUM}(\mathrm{COUNT}(\text{PROD\_SKU})) \mid \mathrm{Z}_2)$ holds. However, as shown in Example 3.11, COUNT_DISTINCT is not a distributive function using function SUM and we cannot apply the same propagation rule as for function COUNT.

The following proposition presents a sufficient condition for COUNT_DISTINCT to be distributive using function SUM.

Proposition 3.5 (Summarizability with COUNT_DISTINCT and SUM)**.** Let $T(\mathrm{S})$ be an analytic table with a set of dimension attributes S_D and an aggregable attribute A. Let $T_1 = \mathcal{A}gg_T(\mathrm{COUNT\_DISTINCT}(\mathrm{A}) \mid \mathrm{Z}_1)$ where $\mathrm{Z}_1 \subseteq \mathrm{S}_D$. If $\mathrm{Z}_2 \subset \mathrm{Z}_1$ and the

literal functional dependency $\mathrm{Z}_2 \cup \{\mathrm{A}\} \mapsto \mathrm{Z}_1$ holds in T, the following equation is true:

$$\mathcal{A}gg_{T_1}(\mathrm{SUM}(\mathrm{COUNT\_DISTINCT}(\mathrm{A})) \mid \mathrm{Z}_2) = \mathcal{A}gg_T(\mathrm{COUNT\_DISTINCT}(\mathrm{A}) \mid \mathrm{Z}_2) \tag{3.2}$$

We say that attribute A (in T) is summarizable with respect to Z_1 and $\mathrm{COUNT\_DISTINCT}$ using function SUM with partitioning attributes Z_2. If Eq. (3.2) holds for *any* subset $\mathrm{Z}_2 \subset \mathrm{Z}_1$, we say that attribute A is summarizable with respect to Z_1 and $\mathrm{COUNT\_DISTINCT}$ using function SUM.

Proof. The previous proposition mainly states that A is summarizable with respect to Z_1 and $\mathrm{COUNT\_DISTINCT}$ using function SUM with partitioning attributes Z_2 if all tuples in some partition $\mathrm{V} \subseteq T$ generated by attributes $\mathrm{Z}_2 \subseteq \mathrm{Z}_1$ which have the same value for attribute A are assigned to the same sub-partition $\mathrm{V}_i \subseteq \mathrm{V}$ generated by attributes Z_1. This avoids double counting of distinct A values when taking the SUM of $\mathrm{COUNT\_DISTINCT}$ over the partitions generated by attributes Z_1. We first show by contradiction that when $\mathrm{Z}_2 \cup \{\mathrm{A}\} \mapsto \mathrm{Z}_1 - \mathrm{Z}_2$ holds in T, all tuples in some partition W generated by attributes Z_2 with the same value for attribute A are assigned to the same sub-partition $\mathrm{V}_i \subseteq \mathrm{V}$ generated by attributes Z_1. Let $\mathrm{V}(x)$ be a partition of T which contains all tuples t such that $t.\mathrm{Z}_2 = x$. Then there exists a partitioning $\mathrm{V}_0 \ldots, \mathrm{V}_n$, $n \geq 0$ of $\mathrm{W}(x)$ defined by attributes Z_1. Suppose that there exist two tuples $t \in \mathrm{V}_i$ and $t' \in \mathrm{V}_j$ where $i \neq j$ and $t.\mathrm{A} = t'.\mathrm{A}$. Then, since $i \neq j$, we have $t.\mathrm{Z}_2 = t'.\mathrm{Z}_2 = x$, $t.\mathrm{A} = t'.\mathrm{A}$ and $t.\mathrm{Z}_1 \neq t'.\mathrm{Z}_1$ which is in contradiction with $\mathrm{Z}_2 \cup \{\mathrm{A}\} \mapsto \mathrm{Z}_1 - \mathrm{Z}_2$. Then, if d_i is the number of distinct A values in some partition $\mathrm{V}_i \subseteq T$, we can easily show that $\sum_{i=0}^{n} d_i$ is the number of distinct A values in partition $W(x)$. □

Example 3.14. Consider the fact table $\mathrm{PRODUCT\_LIST}$ in Table 3.4 and a query $T_1 = \mathcal{A}gg_{\mathrm{PRODUCT\_LIST}}(\mathrm{COUNT\_DISTINCT}(\mathrm{PROD\_SKU}) \mid \mathrm{Z}_1)$ where $\mathrm{Z}_1 = \{\mathrm{BRAND}, \mathrm{COUNTRY}\}$. . For attribute PROD_SKU, we have $\{\mathrm{PROD\_SKU}, \mathrm{BRAND}\} \mapsto \mathrm{COUNTRY}$. Therefore, PROD_SKU is summarizable with respect to Z_1 and $\mathrm{COUNT\_DISTINCT}$ using function SUM with partitioning attribute $\mathrm{Z}_2 = \{\mathrm{BRAND}\}$. Also, since we have $\{\mathrm{PROD\_SKU}, \mathrm{QTY}\} \mapsto \mathrm{YEAR}$, we can say that QTY is summarizable with respect to $\mathrm{Z}_1 = \{\mathrm{PROD\_SKU}, \mathrm{YEAR}\}$ and $\mathrm{COUNT\_DISTINCT}$ using function SUM with partitioning attributes $\mathrm{Z}_2 = \{\mathrm{PROD\_SKU}\}$. The same is true for QTY and $\mathrm{Z}_1 = \{\mathrm{BRAND}, \mathrm{COUNTRY}\}$ with $\mathrm{Z}_2 = \{\mathrm{BRAND}\}$ or $\mathrm{Z}_2 = \{\mathrm{COUNTRY}\}$.

Given a table $T' = \mathcal{A}gg_T(\mathrm{F}(\mathrm{A}) \mid \mathrm{Z})$ which is the result of an aggregate reduction query, and a function G that is applicable to $\mathrm{F}(\mathrm{A})$, we now introduce a new definition

to determine the subset of dimension attributes X such that $\mathbf{agg}_{\mathrm{F(A)}}(\mathrm{G}, \mathrm{X})$ holds in T'. This definition refines the propagation rule 1 in previous Definition 3.9 (case of attribute $\mathrm{F(A)}$) by exploiting the distributivity property of F.

Definition 3.12 (Propagation of aggregable properties with aggregation *preserving summarizability*)**.** Let $T(\mathrm{S})$ be an analytic table with dimension attributes $\mathrm{S}_D \subseteq \mathrm{S}$ and aggregable property $\mathbf{agg}_{\mathrm{A}}(\mathrm{F}, \mathrm{X})$. Let $T' = \mathcal{A}gg_T(\mathrm{F(A)} \mid \mathrm{Z})$ be the result of an aggregate reduction query where $\mathrm{S}_D - \mathrm{X} \subseteq \mathrm{Z}$. The aggregable property $\mathbf{agg}_{\mathrm{F(A)}}(\mathrm{G}, \mathrm{X}')$ holds for attribute $\mathrm{F}(A)$ in T' and all default applicable aggregate functions G defined for the co-domain of function F (Table 2.4) where X' is defined as follows:

1. if $\mathrm{F} = \mathrm{COUNT\_DISTINCT}$ and $\mathrm{G} = \mathrm{SUM}$, then X' is a *maximal* subset of $\mathrm{X} \cap \mathrm{Z}_1$ such that $(\mathrm{S}_D - \mathrm{X}') \cup \{A\} \mapsto \mathrm{Z}$.
2. if F is distributive using G then $\mathrm{X}' = \mathrm{X} \cap \mathrm{Z}$.
3. otherwise $\mathrm{X}' = \emptyset$.

Example 3.15. Consider table $\mathrm{PRODUCT\_LIST}$ in Example 3.11 with aggregable property $\mathbf{agg}_{\mathrm{PROD\_SKU}}(\mathrm{COUNT} \mid \mathrm{X})$ where $\mathrm{X} = \{\mathrm{PROD\_SKU}, \mathrm{BRAND}, \mathrm{COUNTRY}, \mathrm{YEAR}\}$. Let $\mathrm{PRODUCT\_LIST\_COUNT} = \mathcal{A}gg_{\mathrm{PRODUCT\_LIST}}(\mathrm{COUNT}(\mathrm{PROD\_SKU}) \mid \mathrm{Z})$ where $\mathrm{Z} = \{\mathrm{BRAND}, \mathrm{COUNTRY}, \mathrm{YEAR}\}$ (Table 3.5 in Example 3.11). By Item 2 in Definition 3.12, the aggregable property $\mathbf{agg}_{\mathrm{COUNT(PROD\_SKU)}}(\mathrm{SUM} \mid \mathrm{X}')$ holds for $\mathrm{X}' = \mathrm{X} \cap \mathrm{Z} = \{\mathrm{BRAND}, \mathrm{COUNTRY}, \mathrm{YEAR}\}$.

Example 3.16. Assume table $T' = \mathcal{A}gg_{\mathrm{PRODUCT\_LIST}}(\mathrm{COUNT\_DISTINCT}(\mathrm{PROD\_SKU}) \mid \mathrm{Z})$ where $\mathrm{Z} = \{\mathrm{BRAND}, \mathrm{COUNTRY}, \mathrm{YEAR}\}$ (T' is equal to $\mathrm{PRODUCT\_LIST\_COUNT}$ in Example 3.11). To infer the aggregable properties of attribute $\mathrm{COUNT\_DISTINCT(PROD\_SKU)}$ using rule 1, we must compute all maximal subsets $\mathrm{X}' \subseteq \mathrm{X} \cap \mathrm{Z}$ where $(\mathrm{S}_D - \mathrm{X}') \cup \{A\} \mapsto \mathrm{Z}$. First, it is easy to show that YEAR cannot be determined by $\mathrm{S}_D - \mathrm{YEAR} \cup \{\mathrm{PROD\_SKU}\}$, *i.e.* $\mathrm{YEAR} \notin \mathrm{X}'$. For $\mathrm{X}' = \{\mathrm{BRAND}, \mathrm{COUNTRY}\}$, BRAND and COUNTRY cannot be determined by {YEAR, PROD_SKU}. Finally, for $\mathrm{X}'_1 = \{\mathrm{BRAND}\}$ and $\mathrm{X}'_2 = \{\mathrm{COUNTRY}\}$, we can show that the following two LFD are true in table $\mathrm{PRODUCT\_LIST}$:

$$\{\mathrm{PROD\_SKU}, \mathrm{COUNTRY}, \mathrm{YEAR}\} \mapsto \{\mathrm{BRAND}, \mathrm{COUNTRY}, \mathrm{YEAR}\}$$
$$\{\mathrm{PROD\_SKU}, \mathrm{BRAND}, \mathrm{YEAR}\} \mapsto \{\mathrm{BRAND}, \mathrm{COUNTRY}, \mathrm{YEAR}\}$$

Thus, X'_1 and X'_2 are the two possible maximal subsets of $\mathrm{X} \cap \mathrm{Z}$. And by Item 1 in Definition 3.12, we obtain two aggregable

properties: $\mathbf{agg}_{\text{COUNT\_DISTINCT(PROD\_SKU)}}(\text{SUM} \mid \{\text{BRAND}\})$ and $\mathbf{agg}_{\text{COUNT\_DISTINCT(PROD\_SKU)}}(\text{SUM} \mid \{\text{COUNTRY}\})$.

Example 3.17. Assume table $T' = \mathcal{A}gg_{\text{PRODUCT\_LIST}}(\text{COUNT\_DISTINCT}(\text{PROD\_SKU}) \mid \text{Z})$ where $\text{Z} = \{\text{COUNTRY}, \text{YEAR}\}$. We can show that there exists no subset of $\text{X}' \subseteq \text{Z}$ where $(\text{S}_D - \text{X}') \cup \{A\} \mapsto \text{Z}$. First, as before, it is easy to show that YEAR cannot be determined by $\text{S}_D - \text{YEAR} \cup \{\text{PROD\_SKU}\}$, *i.e.* $\text{YEAR} \notin \text{X}'$. For $\text{X}' = \{\text{COUNTRY}\}$, LFD $\{\text{PROD\_SKU}, \text{YEAR}\} \mapsto \{\text{COUNTRY}, \text{YEAR}\}$ does not hold in PRODUCT_LIST. Then, by Rule 1 above $\mathbf{agg}_{\text{COUNT\_DISTINCT(PROD\_SKU)}}(\text{SUM} \mid \emptyset)$, *i.e.* attribute $\text{COUNT\_DISTINCT}(\text{PROD\_SKU})$ is not aggregable. The same is true for $\text{Z} = \{\text{BRAND}, \text{YEAR}\}$.

The following proposition declares that the propagation rules in Definition 3.12 are correct with respect to attribute summarizability.

Proposition 3.6 (aggregable properties and summarizability)**.** Let $T(\text{S})$ be an analytic table with dimension attributes $\text{S}_D \subseteq \text{S}$ and $T_1 = \mathcal{A}gg_T(\text{F}(\text{A}) \mid \text{Z}_1)$ be the result of an aggregate query. Let G be a default applicable aggregate function defined for the co-domain of function F such that aggregable property $\mathbf{agg}_{\text{F(A)}}(\text{G}, \text{X}')$ holds in T_1 with $X' \neq \emptyset$. Then the following equation holds for all Z_2 such that $\text{S}_D - \text{X}' \subseteq \text{Z}_2 \subset \text{Z}_1$:

$$\mathcal{A}gg_T(\text{F}(\text{A}) \mid \text{Z}_2) = \mathcal{A}gg_{T_1}(\text{G}(\text{F}(\text{A}) \mid \text{Z}_2) \tag{3.3}$$

Proof. We examine all the cases provided by Definition 3.12 to define $\mathbf{agg}_{\text{F(A)}}(\text{G}, \text{X}')$. When $\text{F} = \text{COUNT\_DISTINCT}$ and $\text{G} = \text{SUM}$, we first show that $\text{Z}_2 \cup \{\text{A}\} \mapsto \text{Z}_1$ for all Z_2 where $\text{S}_D - \text{X}' \subseteq \text{Z}_2 \subset \text{X} \cap \text{Z}_1$. By Item 1 of Definition 3.12, X' is the maximal subset of $\text{X} \cap \text{Z}_1$ such that $(\text{S}_D - \text{X}') \cup \{\text{A}\} \mapsto \text{Z}_1$. Then, when Z_2 is such that $\text{S}_D - \text{X}' \subseteq \text{Z}_2$ we have $\text{Z}_2 \cup \{\text{A}\} \mapsto \text{Z}_1$. Then, by Proposition 3.5, Equation (3.3) holds for all Z_2 where $\text{S}_D - \text{X}' \subseteq \text{Z}_2 \subset \text{X} \cap \text{Z}_1$.

When F is distributive using function G, it is distributive over any partitioning of $T.\text{A}$ defined by Z_1 or any subset Z_2 of Z_1. By Proposition 3.4, $T.\text{A}$ is then summarizable with respect to Z_1 and F using function G, and by Definition 3.10, Equation (3.3) holds for any subset $\text{Z}_2 \subset \text{Z}_1$.

When none of the previous cases holds, $X' = \emptyset$ and we obtain $\text{S}_D \subseteq \text{Z}_2 \subset \text{Z}_1$ which is impossible, *i.e.* Z_2 does not exist and $F(A)$ is not aggregable. □

Example 3.18. Consider table $\mathrm{PRODUCT\_LIST}$ in Example 3.11 with aggregable property $\mathbf{agg}_{\mathrm{COUNT(PROD\_SKU)}}(\mathrm{SUM} \mid \mathrm{X}')$ with $\mathrm{X}' = \{\textsc{brand}, \textsc{country}, \textsc{year}\}$. Then, by Proposition 3.6, the following equation holds for any valid aggregation $T_1 = \mathcal{A}gg_{\mathrm{PRODUCT\_LIST}}(\mathrm{COUNT}(\textsc{prod\_sku}) \mid \mathrm{Z}_1)$ and any subset of attributes $\mathrm{Z}_2 \subset \mathrm{Z}_1$:

$$\mathcal{A}gg_T(\mathrm{COUNT}(\textsc{a}) \mid \mathrm{Z}_2) = \mathcal{A}gg_{T_1}(\mathrm{SUM}(\mathrm{COUNT}(\textsc{a}) \mid \mathrm{Z}_2) \tag{3.4}$$

Now, assume that table T_1 is defined as in Example 3.16 with two aggregable properties $\mathbf{agg}_{\mathrm{COUNT\_DISTINCT(PROD\_SKU)}}(\mathrm{SUM} \mid \{\textsc{brand}\})$ and $\mathbf{agg}_{\mathrm{COUNT\_DISTINCT(PROD\_SKU)}}(\mathrm{SUM} \mid \{\textsc{country}\})$ for attribute $\mathrm{COUNT\_DISTINCT}(\textsc{prod\_sku})$. We can conclude that the following equation holds for $\mathrm{Z}_2 = \{\textsc{brand}, \textsc{year}\}$ and $\mathrm{Z}_2 = \{\textsc{country}, \textsc{year}\}$:

$$\mathcal{A}gg_{T_1}(\mathrm{COUNT\_DISTINCT}(\textsc{a}) \mid \mathrm{Z}_2) = \mathcal{A}gg_{T_1}(\mathrm{SUM}(\mathrm{COUNT\_DISTINCT}(\textsc{a}) \mid \mathrm{Z}_2) \tag{3.5}$$

Propagation in merge query results

The above Definition 3.12 and Definition 3.8 introduced the propagation rules of the aggregable properties after applying a reduction query. We now consider the problem of determining the aggregable properties of the attributes after a merge. The following proposition states which aggregable properties hold for attribute $\textsc{a}$ in the augmented table T_0', knowing the aggregable properties of $\textsc{a}$ that hold in the used schema augmentation T.

Proposition 3.7. 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 $\textsc{a} \in \mathrm{S}_0'$:

1. If $\mathbf{agg}_{\mathrm{A}}(\mathrm{F}, \mathrm{V})$ holds in T_0 and $\textsc{a} \in \mathrm{S}_0' \cap \mathrm{S}_0$ is an attribute in table T_0, then $\mathbf{agg}_{\mathrm{A}}(\mathrm{F}, \mathrm{V})$ holds in T_0' when T is a natural schema complement to T_0. Otherwise, $\textsc{a}$ is not aggregable anymore in T_0' ($\mathbf{agg}_{\mathrm{A}}(\mathrm{F}, \emptyset)$ holds T_0').
2. If $\mathbf{agg}_{\mathrm{A}}(\mathrm{F}, \mathrm{V})$ holds in T and $\textsc{a} \in \mathrm{S}_0' - \mathrm{S}_0$ is an attribute in table T, then $\mathbf{agg}_{\mathrm{A}}(\mathrm{F}, \mathrm{V} \cap \mathrm{S}_0')$ holds in T_0' when T is a natural schema complement to T_0. Otherwise, $\textsc{a}$ is not aggregable anymore in T_0' (i.e., $\mathbf{agg}_{\mathrm{A}}(\mathrm{F}, \emptyset)$ holds in T_0').

Proof. Let $agg_{\mathrm{A}}(F, \mathrm{X})$ be the aggregable property of $\textsc{a}$ in T_0', *i.e.* $\mathrm{X} \subseteq \mathrm{S}_0'$ is the maximal set of dimension attributes such that $\textsc{a}$ can be aggregated along X with function F in T_0'.

Case one: $\mathrm{A} \in \mathbf{S}'_0 \cap \mathbf{S}_0$**.** A is an attribute in T_0. Let $\mathrm{U}_0 \subseteq S_0$ be a minimal subset of attributes where $\mathrm{U}_0 \mapsto \mathrm{A}$ in T_0, by Definition 2.12, we obtain $\mathrm{V} \subseteq \mathrm{U}_0$.

1. When T is a natural schema complement to T_0, the identifier of S_0 is still the identifier of S'_0. Therefore, we still have $\mathrm{U}_0 \mapsto \mathrm{A}$ in T'_0 and $\mathrm{V} \subseteq \mathrm{S}'_0$. Because A is not an attribute in T, aggregation along attributes in $\mathrm{S}'_0 - \mathrm{S}_0$ is not meaningful, we obtain that V is the maximal set of dimension attributes such that A can be aggregated along with function F in T'_0, $\mathrm{X} = \mathrm{V}$. The aggregable property of A in T_0 is still $\mathbf{agg}_{\mathrm{A}}(\mathrm{F}, \mathrm{V})$.

2. When T is not a natural schema complement to T_0. By Proposition 3.1, the identifier of S_0 is no more the identifier of S'_0, that tuples in T_0 will be duplicated because of the joins with T, then it's not longer meaningful to aggregate A in a duplicated T_0. Therefore, $\mathbf{agg}_{\mathrm{A}}(\mathrm{F}, \emptyset)$ holds in T'_0.

Case two: $\mathrm{A} \in \mathbf{S}'_0 - \mathbf{S}_0$**.** A is an attribute in T. Let $\mathrm{U} \subseteq \mathrm{S}_0 \cap_D \mathrm{S}$ be a minimal subset of attributes where $\mathrm{U} \mapsto \mathrm{A}$ in T and $\mathrm{V} \subseteq \mathrm{U}$.

1. When T is a natural schema complement to T_0. Because $\mathrm{V} \cap \mathrm{S}'_0 \subseteq \mathrm{V} \subseteq \mathrm{U}$, we conclude that $\mathrm{V} \cap \mathrm{S}'_0 \subseteq \mathrm{X}$. Because A is not an attribute in T_0, aggregation along attributes in $\mathrm{S}_0 - \mathrm{V}$ is not meaningful and we obtain $\mathrm{V} \cap \mathrm{S}'_0$ is the maximal set of dimension attributes such that A can be aggregated along with function F in T'_0, $\mathrm{X} = \mathrm{V} \cap \mathrm{S}'_0$. The aggregable property of A in T_0 is $\mathbf{agg}_{\mathrm{A}}(\mathrm{F}, \mathrm{V} \cap \mathrm{S}'_0)$.

2. When When T is not a natural schema complement to T_0. Follow the same reasoning of Case one, it's not meaningful to aggregate A in T'_0. Therefore, $\mathbf{agg}_{\mathrm{A}}(\mathrm{F}, \emptyset)$ holds in T'_0.

□

Example 3.19. Suppose that SALES' is the natural merge of SALES with table $\mathrm{INVENTORY}$ in Figure 2.9 (on Page 37) reduced by the aggregate query $\mathcal{A}gg_{\mathrm{INVENTORY}}(\mathrm{AVG}(\mathrm{A}) \mid \mathrm{X})$ where $\mathrm{A} = \text{QTY\_ON\_HAND}$ and $\mathrm{X} = \{\text{PROD\_SKU}, \text{YEAR}, \text{COUNTRY}\}$. Function AVG is applied on A that transforms A from category **NUM** to attribute $\mathrm{AVG}(\mathrm{A})$ of category **STAT**. Then, only $\mathrm{COUNT}, \mathrm{MIN}$ and MAX are applicable on $\mathrm{AVG}(\mathrm{A})$ by default. Because $\mathbf{agg}_{\mathrm{A}}(\mathrm{AVG}, \mathrm{V})$ holds in $\mathrm{INVENTORY}$ for $\mathrm{V} = \{\text{PROD\_SKU}, \text{YEAR}, \text{WH\_ID}, \text{CITY}, \text{COUNTRY}, \text{TAX\_NO}\}$, then by item 2 in Proposition 3.7, $\mathbf{agg}_{\mathrm{AVG(A)}}(\mathrm{COUNT}, \mathrm{V}')$ holds in SALES' where $\mathrm{V}' = \{\text{YEAR}, \text{COUNTRY}\}$ (V' only contains dimension attributes which are necessary to determine attribute $\mathrm{AVG}(\mathrm{A})$).

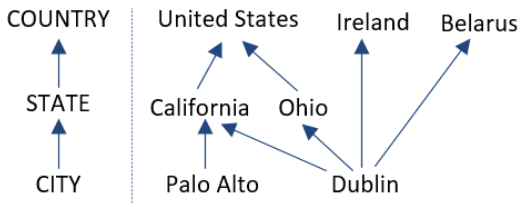


Figure 3.1: Partial hierarchy in dimension $STORE$

3.4.2 Non-ambiguous aggregable attributes

As illustrated in Section 1.2, an attribute value can be ambiguous with respect to a non-strict dimension hierarchy. Remind that ambiguity occurs when it is not possible to identify a unique path (entity) in the dimension for the attribute value.

The next definition states that a schema augmentation $T(\mathrm{S})$ *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 3.13 (Ambiguous analytic table)**.** Let $T(\mathrm{S})$ be an analytic table over a dimension $D(\mathrm{S}_{DT})$. Let $\mathrm{X} = \mathrm{S} \cap \mathrm{S}_{DT}$ be the set of attributes in S from dimension D and $\mathrm{X}^* = \{\mathrm{A}_j \in \mathrm{S}_{DT} \mid \exists \mathrm{A}_i \in \mathrm{X}, \mathrm{A}_i \preccurlyeq^* \mathrm{A}_j\}$ be the ancestors of all attributes in X in the hierarchy type of D. Table T is said to be *non-ambiguous* with respect to D if the literal functional dependency $\mathrm{X} \mapsto \mathrm{X}^*$ holds in $T \bowtie_{\mathrm{X}} D$ and *ambiguous* otherwise.

In the following, X^* is called the closure of X in D. When $\mathrm{X} \mapsto \mathrm{X}^*$, for each tuple t in T, the value set $t.\mathrm{X}$ can identify a unique path in the hierarchy instance graph, and therefore, its value is not ambiguous. We also say that a table T is *ambiguous* if it is ambiguous with respect to at least one dimension and *non-ambiguous* otherwise.

Example 3.20. Fact table SALES contains attributes $\mathrm{X} = \{\textsc{city}, \textsc{state}, \textsc{country}\}$ from dimension $STORE$ (Figure 2.9), the partial hierarchy instance is shown in Figure 3.1. Because $\mathrm{X}^* = \mathrm{X}$, then $\mathrm{X} \mapsto \mathrm{X}^*$ holds in table $\mathrm{SALES} \bowtie_{\mathrm{X}} STORE$. Therefore, SALES is not ambiguous with respect to $STORE$. Now suppose that SALES_2 only contains two attributes $\mathrm{X}_1 = \{\textsc{city}, \textsc{country}\}$ from dimension $STORE$. Then $\mathrm{X}_1^* = \{\textsc{city}, \textsc{state}, \textsc{country}\}$ and $\mathrm{X}_1 \mapsto \mathrm{X}_1^*$ does not hold in $\mathrm{SALES\_2} \bowtie_{\mathrm{X}} STORE$, e.g., given a value pair ("Dublin", "United States") of attributes (CITY, COUNTRY), we can not identify a unique path from CITY to COUNTRY in the partial hierarchy shown in Figure 3.1. In this case, SALES_2 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 with respect to a common dimension D with common attributes $\mathrm{X}_1, \mathrm{X}_2$ respectively, *i.e.* both literal functional dependencies $\mathrm{X}_1 \mapsto \mathrm{X}_1^*$ and $\mathrm{X}_2 \mapsto \mathrm{X}_2^*$ hold in D. Then, by composition of literal functional dependencies, $\mathrm{X}_1 \cup \mathrm{X}_2 \mapsto \mathrm{X}_1^* \cup \mathrm{X}_2^*$ holds in $T_1 \bowtie\!\!\!\!\bowtie_P T_2$.

When reduction queries are applied, it may happen that a reduction query generates an ambiguous schema complement from a non-ambiguous input table (because the reduction query reduces some dimension attributes). The next proposition defines a *sufficient* condition for reduction queries to produce non-ambiguous schema augmentations.

Proposition 3.8. Let $T(\mathrm{S})$ be a non-ambiguous analytic table with respect to a dimension D of schema S_{DT}. Let $T'(\mathrm{S}') = Q(T)$ be a reduction of T. Let $\mathrm{X} = \mathrm{S} \cap \mathrm{S}_{DT}$ and $\mathrm{X}' = \mathrm{S}' \cap \mathrm{S}_{DT}$. If $\mathrm{X}' \mapsto \mathrm{X}$ holds in D, then T' is *non-ambiguous* with respect to D.

Proof. Let $\mathrm{X}' = \mathrm{S}' \cap \mathrm{S}_{DT}$ and $\mathrm{X}'^* = \{\mathrm{A}_j \in \mathrm{S}_{DT} \mid \exists \mathrm{A}_i \in \mathrm{X}', \mathrm{A}_i \preccurlyeq^* \mathrm{A}_j\}$. 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 3.13).

We first show that $\mathrm{X}' \mapsto \mathrm{X}'^*$ holds in $T \bowtie_{\mathrm{X}} D$. Let $\mathrm{X}^* = \{\mathrm{A}_j \in \mathrm{S}_{DT} \mid \exists \mathrm{A}_i \in \mathrm{X}, \mathrm{A}_i \preccurlyeq^* \mathrm{A}_j\}$. 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 $T' \bowtie_{\mathrm{X}'} D$ □

From Proposition 3.8 we can directly conclude that any filter reduction over a non-ambiguous tables generates a non-ambiguous table, because filter reductions do not modify the schema of the table, we have $\mathrm{X}' = \mathrm{X}$, $\mathrm{S}' = \mathrm{S}$ and thus $\mathrm{X}' \mapsto \mathrm{X}$ holds in all dimensions D. However, for pivot and aggregate reduction queries we still must check if $\mathrm{X}' \mapsto \mathrm{X}'^*$ holds in $T' \bowtie_{\mathrm{X}'} D$ (Definition 3.13) to guarantee non-ambiguity ($\mathrm{X}' \mapsto \mathrm{X}$ holds in D is a sufficient but not a necessary condition for non-ambiguity).

Example 3.21. Fact table INVENTORY contains attributes $\mathrm{X} = \{\text{WH\_ID}, \text{CITY}, \text{COUNTRY}\}$ from dimension $WAREHOUSE$. $\mathrm{X}^* = \{\text{WH\_ID}, \text{CITY}, \text{STATE}, \text{COUNTRY}\}$ is the schema of $WAREHOUSE$ and $\mathrm{X} \mapsto \mathrm{X}^*$ holds in $\text{INVENTORY} \bowtie_{\mathrm{X}} WAREHOUSE$. Therefore, table INVENTORY is not ambiguous with respect to $WAREHOUSE$. Now, let T be the result of an aggregate reduction as $\mathcal{A}gg_{\text{INVENTORY}}(\text{AVG}(\text{QTY\_ON\_HAND}) \mid \mathrm{X}')$ where $\mathrm{X}' = \{\text{CITY}, \text{COUNTRY}\}$. Using Proposition 3.8, $\mathrm{X}' \mapsto \mathrm{X}$ 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 $\mathrm{X}'^* = \{\text{CITY}, \text{STATE}, \text{COUNTRY}\}$. Based on the attribute graph in Figure 2.6 (on Page 25), $\mathrm{X}' \mapsto \mathrm{X}'^*$ does not hold, and T is ambiguous with respect to $WAREHOUSE$

3.4.3 Complete merge results

The third quality problem we presented in Section 1.2.4, Page 9, concerns merge queries which might produce results which are incomplete with respect to the original schema augmentation table. A merge is incomplete, if the result of an aggregation of some attribute in the merged table is different from the result of the same aggregation on the original target schema augmentation. We start with an example and then provide a formal definition.

Example 3.22. Consider the two analytic tables T_0 and T below in Table 3.6 related by a well-formed relationship R with common attributes $\mathrm{A}_1, \mathrm{A}_2$ from dimension D_1, and $\mathrm{B}_1, \mathrm{B}_2$ from dimension D_2. The two attribute graphs validated by D_1 and D_2 are shown in Figure 3.2. Identifiers of T_0 and T are $\mathrm{K}_0 = \{\mathrm{A}_1, \mathrm{A}_2, \mathrm{B}_1, \mathrm{B}_2\}$ and $\mathrm{K} = \{\mathrm{A}_1, \mathrm{A}_2, \mathrm{B}_1, \mathrm{B}_2\}$ respectively. Clearly, T is a natural schema complement 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). The results are shown in Table 3.6. By Proposition 3.7, if aggregable property $\mathbf{agg}_M(\mathrm{F}, \{\mathrm{A}_1, \mathrm{A}_2, \mathrm{A}_3, \mathrm{B}_1, \mathrm{B}_2\})$ holds in T, it also holds in T_0'. Then, a valid aggregation query $\mathcal{A}gg(\mathrm{F}(M) \mid \{\mathrm{A}_2, \mathrm{B}_2\})$ 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.

Definition 3.14 (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 $T^{cand} \subseteq T$:

Table 3.6: Example of incomplete merge

T_0	A_1	A_2	B_1	B_2
t_5	a_1	b_1	d_1	e_1
t_6	a_2	b_3	d_3	e_3

T	A_1	A_2	A_3	B_1	B_2	M
t_1	a_1	b_1	c_1	d_2	e_1	y_1
t_2	a_1	b_2	c_2	d_2	e_1	y_2
t_3	a_2	b_1	c_1	d_2	e_3	y_3
t_4	a_1	b_1	c_1	d_3	e_1	y_4

T'_0	A_1	A_2	A_3	B_1	B_2	M
t'_5	a_1	b_1	-	d_1	e_1	-
t'_6	a_2	b_3	-	d_3	e_3	-

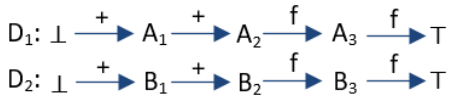


Figure 3.2: Attribute graphs of dimensions D_1, D_2

- the natural semi-join $T^{ct} = T \ltimes_{\mathrm{Y}} T'_0$, where $\mathrm{Y} = \wedge_i (T.\mathrm{A}_i = T'_0.\mathrm{A}_i)$ for each $\mathrm{A}_i \in \mathrm{Y}$, 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 $\mathrm{Y}^{top} = \wedge_i (T.\mathrm{A}_i = T'_0.\mathrm{A}_i)$ for each $\mathrm{A}_i \in \mathrm{Y}^{top}$, is called the *candidate completion table* of T with respect to T'_0.

Example 3.23. In Example 3.22, we have $\mathrm{Y} = \{\mathrm{A}_1, \mathrm{A}_2, \mathrm{B}_1, \mathrm{B}_2\}$ and $\mathrm{Y}^{top} = \{\mathrm{A}_2, \mathrm{B}_2\}$. 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 any result of an augmented merge of T with T_0. The completeness of merge queries can now be defined by comparing the candidate completion table with the completion table.

Definition 3.15 (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 $T^{ct} = T^{cand}$.

Proposition 3.9. Let $T'_0(\mathrm{S}'_0)$ be the natural merge of $T_0(\mathrm{S}_0)$ and $T(\mathrm{S})$ with respect to a set of common attributes Y. Let A be an aggregable attribute in S such that $\mathbf{agg}_{\mathrm{A}}(\mathrm{F}, \mathrm{Y})$ holds in T. Let $\mathrm{Y}^{top} \subseteq \mathrm{Y}$ be the set of highest attributes of Y. Let $Q(T) = \mathcal{A}gg_T(\mathrm{F}(\mathrm{A}) \mid \mathrm{X})$ and $Q(T'_0) = \mathcal{A}gg_{T'_0}(\mathrm{F}(\mathrm{A}) \mid \mathrm{X})$, $\mathrm{Y}^{top} \subseteq \mathrm{X}$, be two valid aggregation queries. If T'_0 is a complete merge with respect to T, then for any two tuples $t_1 \in Q(T), t_2 \in Q(T'_0)$, if $t_1.\mathrm{X} \equiv t_2.\mathrm{X}$, we have $t_1.\mathrm{F}(\mathrm{A}) \equiv t_2.\mathrm{F}(\mathrm{A})$.

Proof. By item 2 in Proposition 3.7, the aggregable property $\mathbf{agg}_{\mathrm{A}}(\mathrm{F}, \mathrm{Y})$ of A still holds in T'_0. Let $\mathrm{V}_0, \mathrm{V}'_0, \mathrm{V}$ be the set of all dimension attributes in T_0, T'_0 and T

respectively. Because $\mathcal{Q}(T_0')$ and $\mathcal{Q}(T)$ are valid aggregation, by Definition 2.12, we have $\mathrm{X} \supseteq \mathrm{V}_0' - \mathrm{Y}$, $\mathrm{X} \supseteq \mathrm{V} - \mathrm{Y}$ and $\mathrm{V}_0' = \mathrm{V}_0 \cup \mathrm{V}$. Therefore, we have $\mathrm{V}_0 \subseteq \mathrm{V}$ which means T contains all dimension attributes of T_0.

We proceed by contradiction. There exist two tuples $t_1 \in \mathcal{Q}(T), t_2 \in \mathcal{Q}(T_0')$ such that $t_1.\mathrm{X} \equiv t_2.\mathrm{X}$, but $t_1.\mathrm{F}(\mathrm{A}) \not\equiv t_2.\mathrm{F}(\mathrm{A})$. This suggests that we cannot get the same value set of A in the partition of tuples $t_1.\mathrm{X}$ in T and the partition of tuples $t_2.\mathrm{X}$ in T_0'.

We distinguish two possible cases: (case 1) the partition of tuples $t_2.\mathrm{X}$ in T_0' contains more tuples than the partition of tuples $t_1.\mathrm{X}$ in T, and (case 2) partition of tuples $t_1.\mathrm{X}$ in T contains more tuples than the partition of tuples $t_2.\mathrm{X}$ in T_0'.

Case 1. Let $t_a \in T_0'$ be a tuple in the partition of $t_2.\mathrm{X}$ in T_0' and $t_a.\mathrm{A}$ is not null, that there does not exist a tuple $t_b \in T$ such that $t_a.\mathrm{Y} \equiv t_b.\mathrm{Y}$ and $t_a.\mathrm{A} \equiv t_b.\mathrm{A}$.

Because A is an attribute in T and $\mathrm{V}_0 \subseteq \mathrm{V}$, we have $\mathrm{V}_0 \cap \mathrm{V} = \mathrm{Y} = \mathrm{V}_0$. When $t_a.\mathrm{A}$ is not null then by definition of a merge there must exist one and only one tuple in T that matches t_a on Y. Therefore, we have a contradiction.

Case 2. Let $t_a \in T$ be a tuple in the partition of $t_1.\mathrm{X}$ in T, that there does not exist a tuple $t_b \in T_0'$ such that $t_a.\mathrm{Y} \equiv t_b.\mathrm{Y}$ and $t_a.\mathrm{A} \equiv t_b.\mathrm{A}$. This means that for any tuple $t_c \in T$, if $t_c.\mathrm{X} \equiv t_a.\mathrm{X}$, we have $t_c.\mathrm{Y} \not\equiv t_a.\mathrm{Y}$.

Because $\mathrm{Y}^{top} \subseteq \mathrm{X}$, we obtain $t_c.\mathrm{Y}^{top} \equiv t_a.\mathrm{Y}^{top}$. By Definition 3.14 that candidate completion table joins T with T_0' on Y^{top} attributes, we get $t_c \in T^{cand}$. By Definition 3.14 that completion table joins T with T_0' on Y attributes and $t_c.\mathrm{Y} \not\equiv t_a.\mathrm{Y}$, we have $t_c \notin T^{ct}$. Thus, $T^{ct} \neq T^{cand}$ which contradicts the assumption that the merge of T_0' is complete.

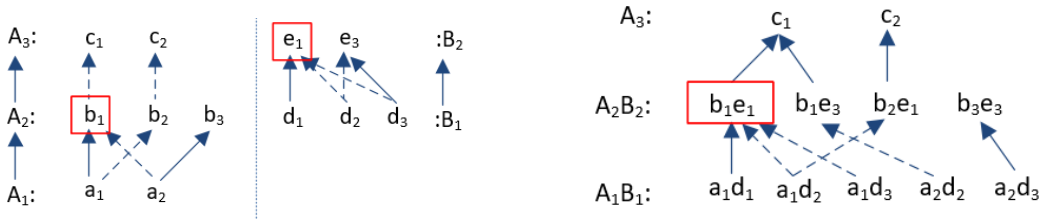
□

By above proposition and proof, we show that when a natural merge is complete, we get the same result for an aggregate query over an aggregable attribute of T grouped by some set of attributes containing the highest common dimension attributes, when it is applied on T_0' or on T. The augmented merge is not considered in this proposition, since the augmented merge will duplicate tuples in T_0, it's obviously we can not guarantee that aggregation queries on T_0' and T return the same results.

However, an implicit assumption of the conditions of Proposition 3.9 is that T_0 has no more dimension attributes than T. Otherwise, the same *valid* aggregate query cannot be expressed on T_0' and T.

Example 3.24. Continuing with Example 3.23, since $T^{cand} \neq T^{ct}$, we can conclude that T'_0 is not a complete merge with respect to T. Figure 3.3a shows the partial hierarchy instances of dimensions D_1 and D_2 that appear in T and T_0: bold arcs are the value pairs that exist in T_0 and dashed arcs are the value pairs that do not exist in T_0 but exist in T. The two value node that surrounded by a red rectangle: b_1 in A_2, e_1 in B_2 are the domain values from Y^{top} that T and T_0 have in common.

For better illustration, we create an artificial hierarchy instance, shown in Figure 3.3b, which combines attributes $\mathrm{A}_2, \mathrm{B}_2$ and $\mathrm{A}_1, \mathrm{B}_1$ together. Bold arcs are the value pairs that exist in T'_0 and dashed arcs are the value pairs that do not exist in T'_0 but exist in T. We can see that T'_0 is not complete with respect to T because the child values of $\mathrm{A}_1\mathrm{B}_1$ below value node "b_1e_1" in $\mathrm{A}_2\mathrm{B}_2$ are not completely included in T'_0. This explains why an aggregation query $\mathcal{Agg}(\mathrm{F}(M) \mid \{\mathrm{A}_2, \mathrm{B}_2\})$ will not produce the same result on each table.



(a) Hierarchy instance of D_1, D_2

(b) Combined hierarchy instance

Figure 3.3: Partial hierarchy instances

When a merge is not complete, it is sometimes possible to complete it with a set of missing tuples, *i.e.* those tuples that are in T^{cand} but not in T^{ct}. The following proposition states how it is possible to complete the result of an incomplete merge query.

Proposition 3.10. Let T be a schema augmentation to T_0 with respect to a relationship R having a set of common 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 that lost in the merge with T_0. For all dimensions $D_i(\mathrm{S}_{D_i})$ in Y, $0 \leq i \leq n$, such that $\mathrm{S}_{D_i} \cap (\mathrm{S}_0 - \mathrm{S}) \neq \emptyset$, when the LFD $(\mathrm{S}_{D_i} \cap \mathrm{S}) \mapsto (\mathrm{S}_{D_i} \cap \mathrm{S}_0)$ holds, we define a completion table $T^{com}(\mathrm{S}'_0)$ as:

$$T^{com} = \Pi_{\mathrm{S}'_0}(T^{miss} \rtimes_{\mathrm{S}_{D_0} \cap \mathrm{S}} D_0 \rtimes \cdots \rtimes_{\mathrm{S}_{D_n} \cap \mathrm{S}} D_n) \tag{3.6}$$

Then

$$Q_m(T_0, T) = T'_0 \cup T^{com} \tag{3.7}$$

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

Proof. We first prove that $\Pi_{\mathrm{S}}(T^{com}) = \Pi_{\mathrm{S}}(T^{miss})$. We propagate the projections in Equation (3.6) and get:

$$T^{com} = T^{miss} \rtimes_{\mathrm{S}_{D_0} \cap \mathrm{S}} (\Pi_{\mathrm{S}'_0}(D_0)) \rtimes \cdots \rtimes_{\mathrm{S}_{D_n} \cap \mathrm{S}} (\Pi_{\mathrm{S}'_0}(D_n)) \tag{3.8}$$

Let $D'_i(\mathrm{S}'_{D_i}) = \Pi_{\mathrm{S}'_0}(D_i)$ be the projection on D_i, then we have $\mathrm{S}'_{D_i} = \mathrm{S}'_0 \cap \mathrm{S}_{D_i} = (\mathrm{S}_{D_i} \cap \mathrm{S}_0) \cup (\mathrm{S}_{D_i} \cap \mathrm{S})$, therefore we have $\mathrm{S}'_{D_i} \cap \mathrm{S} = \mathrm{S}_{D_i} \cap \mathrm{S}$. By the condition that $(\mathrm{S}_{D_i} \cap \mathrm{S}) \mapsto (\mathrm{S}_{D_i} \cap \mathrm{S}_0)$, we obtain $(\mathrm{S}_{D_i} \cap \mathrm{S}) \mapsto (\mathrm{S}_{D_i} \cap \mathrm{S}_0) \cup (\mathrm{S}_{D_i} \cap \mathrm{S})$ which means D'_i is a natural schema complement to T^{miss}. Equation (3.6) is a natural merge of T^{miss} and dimension tables $D'_0, \ldots, D'_n$. By the definition of natural schema complement, the natural merge of T^{miss} and D'_i does not duplicate tuples in T^{miss}, the number of tuples in T^{miss} is the same with T^{com}. Therefore, we have $\Pi_{\mathrm{S}}(T^{com}) = \Pi_{\mathrm{S}}(T^{miss})$.

We now prove that $T_{\mathcal{Q}_m}$ is a complete merge. Let T^{ct} and T^{cand} are the completion table and candidate completion table of T and T'_0 as defined in Definition 3.14. We use Definition 3.15 to prove that $T_{\mathcal{Q}_m} = \mathcal{Q}_m(T_0, T)$ is a complete merge w.r.t. to T:

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

We can replace $\mathcal{Q}_m(T_0, T)$ with the right-hand side of Equation (3.7) and then replace T^{com} with the right-hand side of Equation (3.7) to obtain:

$$\begin{aligned} T^{ct}_{\mathcal{Q}_m} &= T \ltimes_{\mathrm{Y}} (T'_0 \cup T^{com}) \\ &= T^{ct} \cup (T \ltimes_{\mathrm{Y}} T^{com}) \end{aligned} \tag{3.9}$$

Because $\Pi_{\mathrm{S}}(T^{com}) = \Pi_{\mathrm{S}}(T^{miss})$, we can safely replace T^{com} by T^{miss} in Equation (3.9):

$$T^{ct}_{\mathcal{Q}_m} = T^{ct} \cup (T \ltimes_{\mathrm{Y}} T^{miss}) \tag{3.10}$$

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

$$\begin{aligned} T^{ct}_{\mathcal{Q}_m} &= T^{ct} \cup (T \ltimes_{\mathrm{Y}} (T^{cand} - T^{ct})) \\ &= T^{ct} \cup ((T \ltimes_{\mathrm{Y}} T^{cand}) - (T \ltimes_{\mathrm{Y}} T^{ct})) \end{aligned} \tag{3.11}$$

By Definition 3.14, 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 (3.11):

$$\begin{aligned} T^{ct}_{\mathcal{Q}_m} &= T^{ct} \cup (T^{cand} - T^{ct}) \\ &= T^{cand} \end{aligned} \tag{3.12}$$

We apply the same simplification on $T^{cand}_{\mathcal{Q}_m}$ as

$$\begin{aligned} T^{cand}_{\mathcal{Q}_m} &= T \ltimes_{\mathrm{Y}^{top}} (T'_0 \cup T^{com}) \\ &= T^{cand} \cup (T \ltimes_{\mathrm{Y}^{top}} T^{com}) \\ &= T^{cand} \cup (T \ltimes_{\mathrm{Y}^{top}} T^{miss}) \\ &= T^{cand} \cup (T \ltimes_{\mathrm{Y}^{top}} (T^{cand} - T^{ct})) \\ &= T^{cand} \cup ((T \ltimes_{\mathrm{Y}^{top}} T^{cand}) - (T \ltimes_{\mathrm{Y}^{top}} T^{ct})) \end{aligned} \tag{3.13}$$

Similarly, we obtain $T \ltimes_{\mathrm{Y}^{top}} T^{ct} = T^{ct}$ and $T \ltimes_{\mathrm{Y}^{top}} T^{cand} = T^{cand}$ and continue with Equation (3.13):

$$\begin{aligned} T^{cand}_{\mathcal{Q}_m} &= T^{cand} \cup (T^{cand} - T^{ct}) \\ &= T^{cand} \\ &= T^{ct}_{\mathcal{Q}_m} \end{aligned} \tag{3.14}$$

By $T^{ct}_{\mathcal{Q}_m} = T^{cand}_{\mathcal{Q}_m}$, we can conclude that $\mathcal{Q}_m$ is a complete merge 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. For each dimension that exists both in T_0 and T, Equation (3.6) complements the tuples of T^{miss} with values of dimension attributes that exist in T_0 but not in T. The computation of T^{com} avoids generating $null$ values for these attributes and can be skipped if Y already contains all the dimension attributes in T_0. Before computing T^{com}, the condition $(\mathrm{S}_{D_i} \cap \mathrm{S}) \mapsto (\mathrm{S}_{D_i} \cap \mathrm{S}_0)$ makes sure that tuples in T^{miss} do not get duplicated in the computation.

Example 3.25. Consider again table SALES and INVENTORY from Figure 2.9 (Page 37). INVENTORY is a schema augmentation to SALES, assuming an aggregate reduction is applied on INVENTORY that reduces attribute WH_ID as $\mathcal{Agg}_{\mathrm{INVENTORY}}(\mathrm{SUM}(\text{QTY\_ON\_HAND}) \mid \mathrm{X})$, $\mathrm{X} = \{\text{PROD\_SKU}, \text{BRAND}, \text{MONTH}, \text{YEAR}, \text{CITY}, \text{COUNTRY}\}$. The result is shown in Table 3.7b. An natural merge of SALES with reduced INVENTORY using tuples shown in

Table 3.7a will yield SALES′ that contains tuple t_1 augmented by new $null$ valued attributes WH_ID and QTY_ON_HAND (there's not matching tuple in INVENTORY). If we compute T^{cand} and T^{ct} as defined in Definition 3.14, we have $T^{ct} = \emptyset$ and $T^{cand} = \{t_2\}$, by Definition 3.15, the merge result SALES′ is not complete. For dimension $STORE$ in Table 3.7c, we have $S_{STORE} \cap S_{SALES} = \{$CITY, STATE, COUNTRY$\}$, $S_{STORE} \cap S_{INVENTORY} = \{$WH_ID, CITY, COUNTRY$\}$, the condition proposed in Proposition 3.10 that $(S_{STORE} \cap S_{INVENTORY}) \mapsto (S_{STORE} \cap S_{SALES})$ does not hold. Similarly, the condition does not hold for dimension $TIME$. Therefore, we can not compute the completion table, otherwise, we introduce new STATE values that neither exist in SALES nor in INVENTORY.

Table 3.7: Examples of table SALES, INVENTORY and $STORE$

(a) Table SALES

SALES(T_0)	PROD_SKU	BRAND	YEAR	CITY	STATE	COUNTRY	AMOUNT
t_1	i7-32g-black	Apple	2017	San Jose	California	United States	660,000

(b) Table INVENTORY′

INVENTORY′ (T)	PROD_SKU	BRAND	MONTH	YEAR	CITY	COUNTRY	QTY_ON_HAND
t_2	i7-32g-black	Apple	Jan	2017	Dublin	United States	12,000

(c) Table STORE

$STORE$	STORE_ID	CITY	STATE	COUNTRY
	st_01	Dublin	Ohio	United States
	st_02	Dublin	California	United States

Example 3.26. Resuming Example 3.22, we obtain $T^{miss} = T^{cand} - T^{ct} = \{t_1, t_4\}$. Besides, for dimensions D_1, D_2 we have $S_{D_1} \cap (S_0 - S) = S_{D_2} \cap (S_0 - S) = \emptyset$, the condition in Proposition 3.10 to compute the completion table checks. Thus, we have $T^{com} = \{t_1, t_4\}$ 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 of T_0 and T with respect to T.

Table 3.8: Complete merge of T_0 and T

Q_m	A_1	A_2	A_3	B_1	B_2	M
t'_5	a_1	b_1	-	d_1	e_1	-
t'_6	a_2	b_3	-	d_3	e_3	-
t_1	a_1	b_1	c_1	d_2	e_1	y_1
t_4	a_1	b_1	c_1	d_3	e_1	y_4

Based on the previous definitions, we can extend item 2 in Proposition 3.7 by considering the complete merge result as follows.

Proposition 3.11. Let $T_0'(\mathrm{S}_0')$ be a natural merge of table $T_0(\mathrm{S}_0)$ with a target schema augmentation $T(\mathrm{S})$. Let $\mathrm{Y}_0, \mathrm{Y}_1$ be the set of all dimension attributes in T_0 and T respectively. Let A be an aggregable attribute in T such that $\mathbf{agg}_{\mathrm{A}}(\mathrm{F}, \mathrm{V})$ holds in T. Then the aggregable property hold for A in S_0':

1. When T_0' is a complete merge and $\mathrm{Y}_0 \subseteq \mathrm{Y}_1$, then $\mathbf{agg}_{\mathrm{A}}(\mathrm{F}, \mathrm{V} \cap \mathrm{S}_0')$ holds in T_0'.
2. Otherwise, $\mathbf{agg}_{\mathrm{A}}(\mathrm{F}, \emptyset)$ holds.

Proof. By item 2 in Proposition 3.7, when T_0' is a natural merge of T_0 and T, we have $\mathbf{agg}_{\mathrm{A}}(\mathrm{F}, \mathrm{V} \cap \mathrm{S}_0')$ holds in T_0'. Therefore, the first item in the proposition is obviously true. We now discuss the second item. We separate two cases:

Case 1. When T_0' is a complete merge and $\mathrm{Y}_0 \not\subseteq \mathrm{Y}_1$. By the proof of Proposition 3.9, we know that when T contains all dimension attributes of T_0 ($\mathrm{Y}_0 \subseteq \mathrm{Y}_1$), we can get the same aggregation of A values on T and T_0' if we aggregate group by the same set of dimension attributes. Therefore, when $\mathrm{Y}_0 \not\subseteq \mathrm{Y}_1$, aggregations of A on T_0' can not be guaranteed to obtain the same result as aggregation on T, thus, aggregations on T_0' should be avoided in case of producing inconsistent values with respect to T and we have $\mathbf{agg}_{\mathrm{A}}(\mathrm{F}, \emptyset)$ holds in T_0'

Case 2. When T_0' is not a complete merge. It is not ensured that aggregations on attribute A applied on T_0' are correct. Therefore, $\mathbf{agg}_{\mathrm{A}}(\mathrm{F}, \emptyset)$ holds in T_0'. □

More precisely, Proposition 3.11 only can be applied for computing the aggregable properties of attributes when a completion table is computed to complete the merge of T_0 and T.

Example 3.27. Continuing with Example 3.25, the aggregable property of attribute QTY_ON_HAND in $\mathrm{INVENTORY}'$ is $\mathbf{agg}_{\text{QTY\_ON\_HAND}}(\mathrm{SUM}, \mathrm{X})$, $\mathrm{X} = \{\text{PROD\_SKU}, \text{BRAND}, \text{CITY}, \text{COUNTRY}\}$. Because the merge of SALES and $\mathrm{INVENTORY}'$ is not complete, aggregation in SALES' would miss QTY_ON_HAND values. For example, a summation of inventory by brand and year of 'United States' in SALES' will miss the inventory of warehouse 'oh_01', and thus computes a wrong result. Therefore, aggregations on QTY_ON_HAND in SALES' are not meaningful and we conclude that the aggregable property of QTY_ON_HAND in SALES' is $\mathbf{agg}_{\text{QTY\_ON\_HAND}}(\mathrm{SUM}, \emptyset)$.

3.4.4 Summarizability revisited

Summarizability [20], [23], [25], [32] is the ability of "correctly computing aggregate values with a coarser level of detail from values with a finer level of detail". In this section, we first explain our definition of summarizability and then show how we can ensure summarizability for reduction queries and merge queries. In Chapter 5, we compare our definition and conditions with previous work on summarizability.

Example 3.28. Consider table T from Table 3.6 (copied below for clarity) and assume that $\mathbf{agg}_M(\text{SUM} \mid \{\text{A}_1, \text{A}_2, \text{A}_3, \text{B}_1, \text{B}_2\})$ holds in T. We apply two nested aggregation queries $T_{agg1} = \mathcal{A}gg_T(\text{SUM}(M) \mid \{\text{A}_2, \text{A}_3, \text{B}_2\})$ and $T_{agg2} = \mathcal{A}gg_{T_{agg1}}(\text{SUM}(M) \mid \{\text{A}_3\})$. The final result is shown in Table 3.10. We can show that query T_{agg2} is equivalent to query $T_{agg3} = \mathcal{A}gg_{T_{agg1}}(\text{SUM}(M) \mid \{\text{A}_3\})$. The property that any repeated application of an aggregate query $T_{i+1} = \mathcal{A}gg_{T_i}(F(\text{A}) \mid \text{X}_i)$, $\text{X}_i \subset X_{i+1}, 0 \leq i \leq n$ is equivalent to a single application of $\mathcal{A}gg_{T_0}(F(\text{A}) \mid \text{X}_n)$ is called *summarizability* (by Definition 3.10 attribute A is summarizable with respect to X_n and function SUM using function SUM).

T	A_1	A_2	A_3	B_1	B_2	M
	a_1	b_1	c_1	d_2	e_1	y_1
	a_1	b_2	c_2	d_2	e_1	y_2
	a_2	b_1	c_1	d_2	e_3	y_3
	a_1	b_1	c_1	d_3	e_1	y_4

Table 3.10: Result of aggregation queries on T

(a) Table T_{agg1}

A_2	A_3	B_2	SUM(M)
b_1	c_1	e_1	$y_1 + y_4$
b_2	c_2	e_1	y_2
b_1	c_1	e_3	y_3

(b) Table T_{agg2}

A_3	SUM(M)
c_1	$y_1 + y_3 + y_4$
c_2	y_2

Example 3.29. Continuing Example 3.28 where $\text{Z}_1 = \{\text{A}_2, \text{A}_3, \text{B}_2\}$ and F= SUM. We can see that for any subset Z_2 of Z_1, $\text{Z}_2 \subset \text{Y}_1, \text{Z}_2^{top} \subseteq \text{Z}_1^{top}$, $\mathcal{A}gg_{T_{agg1}}(\text{F}(M) \mid \text{Z}_2) = \mathcal{A}gg_T(\text{F}(M) \mid \text{Z}_2)$ and M is summarizable with respect to Z_1 and function SUM. But M is not summarizable with respect to Z_1 and function COUNT.

We now extend the notion of summarizability as the ability of "correctly computing aggregate values from a merged result with respect to the table before the merge". Definition 3.10 and Proposition 3.4 define the summarizability and the propagation

of aggregable properties for new attributes computed by aggregate queries *before* merging the result with some other table. The following definition considers the summarizability of these attributes *after* a complete merge.

Definition 3.16 (Summarizable attribute in a merge table)**.** Let $T_0'(\mathrm{S}_0')$ be the merge of $T_0(\mathrm{S}_0)$ and $T(\mathrm{S})$ with respect to a relationship R with a set of common attributes Y. Let A be an aggregable attribute in T such that $\mathbf{agg}_{\mathrm{A}}(\mathrm{F}, \mathrm{X}), \mathrm{X} \subseteq \mathrm{Y}$ holds in T. Let $Q_1 = \mathcal{A}gg_T(\mathrm{F}(\mathrm{A}) \mid \mathrm{Y}_1), \mathrm{S}-\mathrm{X} \subseteq \mathrm{Y}_1$ be an aggregate query over T and $T'_{Q_1} = T_0 \rtimes\!\bowtie Q_1$ be a complete merge of T_0 and Q_1. If for any Y_1 such that $\mathrm{Y}_1^{top} = \mathrm{Y}^{top}$, where Y^{top} and Y_1^{top} are the set of the highest level attributes of Y, Y_1 respectively, we have

$$\pi_{\mathrm{S}}(\mathcal{A}gg_{T_0'}(\mathrm{F}(\mathrm{A}) \mid \mathrm{Y}_1)) = \pi_{\mathrm{S}}(T_0 \rtimes\!\bowtie \mathcal{A}gg_T(\mathrm{F}(\mathrm{A}) \mid \mathrm{Y}_1))$$

then A is said to be *summarizable* in the merge of T_0 and T with respect to X and function F.

In the complete merge result T_0', all tuples of T_0 that cannot be joined over Y with tuples in T T will have $null$ values for any aggregable attribute A that comes from T. Therefore, we only consider table T for computing aggregations along Y for attribute A. The condition $\mathrm{Y}_1^{top} = \mathrm{Y}^{top}$ guarantees that the aggregations are computed from a lower level to a higher level and they only aggregate values within the same partial hierarchy instance below $T.\mathrm{Y}^{top}$.

The following proposition states that summarizability can be guaranteed for complete merge table and distributive aggregation functions.

Proposition 3.12. Let $T_0'(\mathrm{S}_0')$ be the merge of $T_0(\mathrm{S}_0)$ and $T(\mathrm{S})$ with respect to a relationship R with a set of common attributes Y. Let A be an aggregable attribute in T such that $\mathbf{agg}_{\mathrm{A}}(\mathrm{F}, \mathrm{X}), \mathrm{X} \subseteq \mathrm{Y}$ holds in T. If T_0' is a complete merge, then A is summarizable in the merge of T_0 and T with respect to X and function F.

Proof. By Definition 3.16, we prove that for an arbitrary $\mathrm{Y}_1 \supseteq \mathrm{S} - \mathrm{X}$, $Q_1 = \mathcal{A}gg_T(\mathrm{F}(\mathrm{A}) \mid \mathrm{Y}_1)$, $T'_{Q_1} = T_0 \rtimes\!\bowtie Q_1$ is a complete merge of $T_0(\mathrm{S}_0)$ and Q_1 with respect to common attributes Y_1. We have

$$\pi_{\mathrm{S}}(\mathcal{A}gg_{T_0'}(\mathrm{F}(\mathrm{A}) \mid \mathrm{Y}_1)) = \pi_{\mathrm{S}}(T_0 \rtimes\!\bowtie Q_1) \tag{3.15}$$

Because attribute A is aggregable in $T(\mathrm{S})$ and all tuples in $T'_0 - (T'_0 \ltimes_{\mathrm{Y}} T)$ will have null values for A, the left expression in Equation (3.15) can be replaced by:

$$\begin{aligned}\pi_{\mathrm{S}}(\mathcal{A}gg_{T'_0}(\mathrm{F}(\mathrm{A}) \mid \mathrm{Y}_1)) &= \pi_{\mathrm{S}}(\mathcal{A}gg_{T'_0 \ltimes_{\mathrm{Y}} T}(\mathrm{F}(\mathrm{A}) \mid \mathrm{Y}_1)) \\ &= \mathcal{A}gg_{\pi_{\mathrm{S}}(T'_0 \ltimes_{\mathrm{Y}} T)}(\mathrm{F}(\mathrm{A}) \mid \mathrm{Y}_1)\end{aligned} \tag{3.16}$$

Let T^{ct}, T^{cand} be respectively the completion table and candidate completion table of T and T'_0 as defined in Definition 3.14. We have $\pi_{\mathrm{S}}(T'_0 \ltimes_{\mathrm{Y}} T) = T \ltimes_{\mathrm{Y}} T'_0 = T^{ct}$. Because T'_0 is a complete merge, we have $T^{ct} = T^{cand} = T \ltimes_{\mathrm{Y}^{top}} T'_0$. This allows us to continue with the previous equation:

$$\begin{aligned}\pi_{\mathrm{S}}(\mathcal{A}gg_{T'_0}(\mathrm{F}(\mathrm{A}) \mid \mathrm{Y}_1)) &= \mathcal{A}gg_{T^{cand}}(\mathrm{F}(\mathrm{A}) \mid \mathrm{Y}_1) \\ &= \mathcal{A}gg_{T \ltimes_{\mathrm{Y}^{top}} T'_0}(\mathrm{F}(\mathrm{A}) \mid \mathrm{Y}_1) \\ &= \mathcal{A}gg_{T}(\mathrm{F}(\mathrm{A}) \mid \mathrm{Y}_1) \ltimes_{\mathrm{Y}^{top}} T'_0 \\ &= \mathcal{Q}_1 \ltimes_{\mathrm{Y}^{top}} T'_0\end{aligned} \tag{3.17}$$

Let $T^{ct}_{\mathcal{Q}_1}$ and $T^{cand}_{\mathcal{Q}_1}$ be respectively the completion table and candidate completion table of $\mathcal{Q}_1$ and $T'_{\mathcal{Q}_1} = T_0 \rtimes \mathcal{Q}_1$ as defined in Definition 3.14. Because $T'_{\mathcal{Q}_1}$ is a complete merge of T_0 and $\mathcal{Q}_1$, we have $T^{ct}_{\mathcal{Q}_1} = T^{cand}_{\mathcal{Q}_1}$, and

$$\begin{aligned}\pi_{\mathrm{S}}(T'_{\mathcal{Q}_1}) &= T^{ct}_{\mathcal{Q}_1} = T^{cand}_{\mathcal{Q}_1} \\ &= \mathcal{Q}_1 \ltimes_{\mathrm{Y}^{top}_1} T'_{\mathcal{Q}_1}\end{aligned} \tag{3.18}$$

Because $\mathrm{Y}^{top}_1 = \mathrm{Y}^{top}$, to prove Equation (3.15) is now the same as proving:

$$\mathcal{Q}_1 \ltimes_{\mathrm{Y}^{top}_1} T'_0 = \mathcal{Q}_1 \ltimes_{\mathrm{Y}^{top}_1} T'_{\mathcal{Q}_1} \tag{3.19}$$

which is essentially proving that $\pi_{\mathrm{Y}^{top}_1}(T'_0) = \pi_{\mathrm{Y}^{top}_1}(T'_{\mathcal{Q}_1})$.

By the nature of left-outer join no tuples in T_0 will be removed and the complete merge does not add new domain values for $\mathrm{Y}^{top}(\mathrm{Y}^{top}_1)$ (the complete merge only brings new domain values for $\mathrm{Y}^{top} - \mathrm{Y}$). Therefore, we have $T'_0.\mathrm{Y}^{top}_1 = T_0.\mathrm{Y}^{top}_1$ and $T'_{\mathcal{Q}_1}.\mathrm{Y}^{top}_1 = T_0.\mathrm{Y}^{top}_1$. Thus, we can conclude $\pi_{\mathrm{Y}^{top}_1}(T'_0) = \pi_{\mathrm{Y}^{top}_1}(T'_{\mathcal{Q}_1}) == \pi_{\mathrm{Y}^{top}_1}(T_0 \rtimes \mathcal{Q}_1)$, *i.e.* A is summarizable in the merge of T_0 and T with respect to X and function F, when T'_0 is a complete merge. □

4

Architecture and Algorithms

Contents

The contributions presented in this paper have been implemented within the SAP HANA platform [33], a main-memory relational database system, as a REST application service. In this chapter, we give a brief introduction to the architecture and explain several main components in Section 4.1. Then in Sections 4.2 to 4.5, we explain in details the REST API of our services, their inputs and outputs and the processes.

4.1 SAP HANA Architecture

Analytic tables in SAP HANA are defined as non-recursive *information views* [34] over non-analytic tables and/or previously defined information views, using a DAG of operators (e.g., union, join, projection, aggregate). As shown in Figure 2.1 on

Page 17, dimension and fact tables can be either defined as arbitrary views over non-analytic tables or as project-union views over other analytic views, and hierarchy types are part of the definition of these views. The definition of these information views also includes to verify the role played by dimension attributes or measure attributes as defined in Sections 2.2.2 and 2.2.5. We extended the information view framework by first enabling the definition of new metadata which are the declaration of aggregable properties of measures in fact tables, and attribute graphs for dimensions.

The extended HANA architecture is depicted in Figure 4.1. White squared boxes are new components that implement the algorithms described in Sections 4.2 to 4.5, white rounded boxes are components that store all the metadata described in Chapter 2, whereas grey boxes are existing HANA components that have been extended.

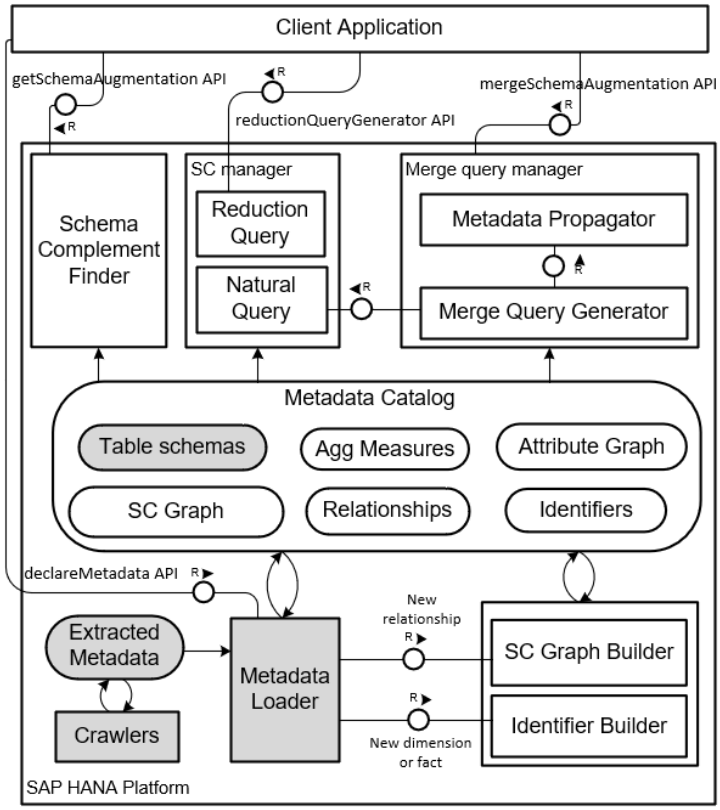


Figure 4.1: Architecture overview

SAP HANA platform also provides schedulable metadata crawlers that asynchronously extract metadata from HANA tables and views, or remote datasets using the wrapper framework to connect to remote data sources. Extracted metadata are translated into a standard representation based on a rich Entity-Relationship

metadata model. Crawlers extract semantic concepts beyond the basic metadata returned by wrappers, such as dimensions and measures in the case of analytical data (e.g., remote data warehouses or HANA calculation views), as well as relationships such as fact-dimension relationships extracted from cube definitions, or attribute mapping and join relationships extracted from the analysis of analytic queries or views. When aggregable properties of attributes are not provided, default values are used (see Section 2.2.6), when attribute graphs are not given, we compute attribute graphs based on the Definition 2.5 (see Section 4.2.1).

Extracted metadata are periodically checked and asynchronously stored by the *Metadata Loader* into the *Metadata Catalog* and asynchronous data profiling processes – not portrayed in the architecture diagram – are used to post-analyze the table instances discovered by the crawlers, and enrich the Metadata Catalog accordingly (e.g., add join relationships for common types).

External data are access by the extractors using the definition of external datasets to extract their semantic properties such as multidimensional concepts in the case of data source systems storing analytic data (e.g., data warehouses) as well as semantic relationships between datasets. Thus, both external data and HANA data are mapped to a common data model.

We now focus our descriptions on the main components and APIs that discover schema complements between datasets and compute merge queries.

All the metadata we crawled or computed from the dataset are stored in **Metadata Catalog**, those metadata are grouped as following.

- *Table Schemas* stores the table representations of local and remote tables or views. Tables information and attributes information are stored separately in two tables.
  - Each table or view is represented by a tuple (t_id, loc, t_type), where t_id is the unique identifier of the table, loc is the physical location of the table, and t_type is the table type (i.e., *fact*, *dimension*, or *non-analytic*), a dimension table or a non-analytic table. For example, fact table SALES is described by the tuple: ($T1$, "SYSTEM"."SALES", *fact*).
  - Each attribute is described by a tuple $(a\_id, a\_name, a\_role, dataType, t\_id, a\_ref)$, where a_id is the unique identifier of the attribute, a_name is the attribute name, a_role is the role it plays as a dimension attribute or a measure (i.e., *id* or *detail* for dimension attribute, *value* or *detail* for measure attribute),

$dataType$ is its data type, and the table t_id the attribute belongs to. When the attribute is of role *detail*, a_ref stores the attribute identifier of which this attribute is associated with. For example, measure attribute AMOUNT in table SALES is of role *value* and is described by the tuple: ("$T1$"."AMOUNT", AMOUNT, $value$, $decimal$, $T1$, -). Measure attribute CURRENCY in SALES is of role detail for attribute AMOUNT, CURRENCY is describe by the tuple: ("$T1$"."CURRENCY", CURRENCY, $detail$, $varchar$, $T1$, "$T1$"."AMOUNT").

- Aggregable attributes are also described in *Agg Measures* with detail information of their aggregable properties.

  - Each aggregable attribute is represented by a tuple ($a\_id, func, a\_source, t\_id$), where a_id is its unique identifier, $func$ is the last aggregate function applied on the attribute, a_source is the attribute identifier of which it's computed from, and t_id is the table id of a_source (the source attribute should in a non-analytic table). For example, attribute *AMOUNT* in table SALES is described by the tuple: ("$T1$"."AMOUNT", SUM, "$T12$"."AMOUNT", $T12$), ($T12$ is table $\mathrm{ct\_SALES}$).

  - An aggregable attribute might have several different aggregable properties for different aggregable functions. The set of aggregable properties of one attribute a_id are encoded as a set of tuples of the form ($a\_id, func, a\_ids$), where $func$ is the aggregate function, a_ids is the set of dimension attributes such that a_id can aggregate along with using $func$. For example, the aggregable property of AMOUNT for function SUM in SALES is $\mathbf{agg}_{\mathrm{AMOUNT}}(\mathrm{SUM}, Z)$, $Z = \{\mathrm{PROD\_SKU}, \mathrm{BRAND}, \mathrm{MONTH}, \mathrm{YEAR}, \mathrm{CITY}, \mathrm{STATE}, \mathrm{COUNTRY}\}$. This aggregable property is encoded as: ("$T1$"."AMOUNT", SUM, {"$T1$"."PROD_SKU", "$T1$"."BRAND", "$T1$"."YEAR", "$T1$"."CITY", "$T1$"."STATE", "$T1$"."COUNTRY"}).

- *Attribute Graph* stores dimension hierarchies that are defined in analytic tables and the corresponding satisfied attribute graphs.

  - The metadata of each hierarchy in dimensions is encoded by a tuple (h_id, t_id, loc), where h_id is the hierarchy identifier, t_id is the related dimension table id, loc is the physical localtion of the hierarchy table (see Section 4.2.1). For example, the hierarchy in dimension

$WAREHOUSE$ is described by the tuple: (1, $T6$, "SYSTEM"."WAREHOUSE.hier.warehouse").

  - The attribute graph that the dimension table satisfies is stored in two tables, one for the nodes in attribute graph and another for the edges. For a detailed example of the two tables see Section 4.2.1.

- *Identifiers* stores the primary keys and fact / dimension identifiers of the tables in *Table schemas*.

  - As illustrated in Section 2.2.5, a table can have more than one identifier. Each identifier is encoded by a tuple ($k\_id, t\_id, type, min$) where k_id is the unique id of the identifier, t_id is the table id, $type$ is the type of the identifier (i.e., computed identifier, extracted primary key, or inferred primary key), and min states whether it is the minimum identifier in the table. For example, the primary key of table $\mathrm{PRODUCT}$ is describe by the tuple: ($\mathrm{K9}$, $T9$, *extracted*, *true*). We show in Section 4.2 that besides extracting primary keys and computing dimension / fact identifiers, we can also obtain an identifier by inferring primary keys.

  - The composition of each identifier is stored in another table separately. Each identifier is represented by a set of tuples of the form ($a\_id, k\_id, length, min\_id$) where a_id stores the attribute id in the identifier with id k_id, $length$ stores the size of the identifier, min_id refers to the minimal identifier in the table ($null$ if itself is the minimal identifier). For example, the primary key of $\mathrm{PRODUCT}$ consists of one attribute PROD_SKU, then there is a tuple as: ("$T9$"."PROD_SKU", $\mathrm{K9}$, 1, -).

- *Relationships* stores all relationships between tables in *Table schemas*. These relationships are either extracted from the definition of the information views or non-analytic tables, or derived relationships computed using composition and fusion of relationships as Propositions 2.6 and 2.7, or user-defined relationships.

  - As stated in Section 2.3, there could exist multiple relationships between two tables. Each relationship is described by a tuple ($r\_id, t\_origin, t\_des, type$), where r_id is the unique id of the relationship, t_origin and t_des are respectively the table id of the source table and the target table of the relationship, $type$ is the relationship type (i.e., join relationship, attribute mapping relationship, computed relationship, or user defined). For example, the relationship between SALES and $WAREHOUSE$ is represented by the tuple: ($\mathrm{R16}$, $T1$, $T6$, *extracted*).

- A relationship contains one or multiple matchings between attributes of the source and the target table. Each relationship is encoded by a set of tuples of the form $(r\_id, a\_origin, a\_des, length)$, where r_id is the relationship id, a_origin, a_des are respectively the attribute id in the n source and the target table, and $length$ is the number of the matchings exist in the relationship. For example, the relationship between SALES and $WAREHOUSE$ contains three attribute matchings, one matching SALES.COUNTRY $= WAREHOUSE$.COUNTRY is encoded by the tuple: (R 16, "$T1$"."COUNTRY", "$T6$"."COUNTRY", 3).

- *Schema complement graph ($\mathcal{SC}$ Graph)* is a directed graph that connects tables in *Table schemas* by their complement types (natural schema complement or schema augmentation). The graph is computed based on relationships we have (Detail see Definition 4.1). We store the tables and edges of SC Graph separately in two tables.

  - Nodes in $\mathcal{SC}$ Graph represent tables (analytic tables or non-analytic tables), each node in represented by a tuple $(n\_id, type, t\_id, k\_id, -)$ where n_id is the unique id of the node, t_id is the table id that the node represents, k_id is the minimal identifier of the table, and $type$ stores the type of the tuple describes (i.e., *table* or *edge*). Because edges in $\mathcal{SC}$ Graph contain a label indicates the complement type and associated relationship. Edges in $\mathcal{SC}$ Graph are represented by a tuple $(e\_id, type, -, -, r\_id)$ where e_id is the unique id of the edge, and r_id is the relationship id the edge associates with. For example, table SALES is a node in the $\mathcal{SC}$ Graph and is represented by the tuple: (1, *table*, $T1$, K1, -). Table $T1$ (SALES) is a schema augmentation to $T2$ (INVENTORY) with respect to relationship R_{12}, there exist an edge $\mathcal{E}(T1, T2)$ in $\mathcal{SC}$ Graph which is encoded in the tuple: (2, *edge*, -, -, R 12).

  - The connections between nodes and edges in $\mathcal{SC}$ are also encoded into a table. Each tuple is of the form $(origin\_id, des\_id, side, type)$, where $origin\_id, des\_id$ are respectively the source and the target id, $side$ describes the side (i.e., "O" for *original side* and "D" for *Destination side*), and $type$ stores the complement type. Each edge in $\mathcal{SC}$ Graph is encoded by two tuples which connects the start table to the edge and the edge to the destination table. For example, the edge $\mathcal{E}(T1, T2)$ describe above is encoded by the two tuples: (1, 2, *O*, *AUG*), (2, 3, *D*, -), *3* is the node id for table $\mathrm{INVENTORY}$.

Metadata Catalog stores every information we need for computing identifiers, finding schema complements and computing merge queries.

Metadata Loader manages the insertions and updates of the metadata of tables. When a new dimension table or a new fact table is crawled, *Metadata Loader* is notified with new *Extracted Metadata*, and *Table schemas*, *Attribute Graph*, *Identifiers*, *Agg Measures* and *Relationships* in *Metadata Catalog* are updated with new metadata. *Identifier Builder* is then called to compute a new identifier, if the attribute graph is not given, it computes the attribute graph and finally stores the attribute graph and computed identifiers into *Attribute Graph* and *Identifiers* respectively. If the attribute graph of a dimension table is updated and a new dimension identifier is computed, then the fact identifier of every fact table that contains dimension attributes of this dimension will be recomputed. When the newly crawled table is defined as a view over other tables which are not extracted into *Metadata Catalog* or partially extracted, those missing tables will be crawled by *Crawlers* first, and *Metadata Loader* will load and process metadata of the new extracted table until all the missing tables are stored in *Metadata Catalog*. When a new non-analytic table is crawled, the PK declaration is extracted directly from its definitions and used to update *Identifiers*, *Identifier Builder* is also called to verify whether this PK could be inferred to analytic tables. When relationships are added or updated in *Relationships*, *SC Graph Builder* applies Propositions 2.6 and 2.7 that runs recursively to compute if there are relationships which could be derived or merged and passed the results to *Relationships*. Modified relationships are then used to update *SC Graph*. Besides, *Metadata Loader* processes *Declare Metadata* API calls from client, which 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*, *Agg Measures* 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.

Metadata Loader maintains the correctness and consistency of the data stored in *Metadata Catalog*, we now introduce other components that collaborate with APIs and response to user's requests.

The **Schema Complement Finder** component processes a *Get Schema Augmentation* API call. It takes a start table name T_0 as input and returns a list of schema complements and augmentations T with their relationships from T_0 to T and a set of attributes in T that could be augmented to T_0. Those schema complements and augmentations are found by traversing the *SC Graph* and comparing the *Table schemas* (see Section 4.3).

SC manager takes the *Reduction Query Generator* API call which contains a target schema augmentation table, and a set of user actions to reduce the identifier of the target table as inputs. Using the *Reduction Query* sub-manager, it returns a reduction query and a description of aggregated attributes in the query. The algorithm to generate such reduction query is described in Section 4.4. The *SC Manager* uses a *Natural Query* sub-manager to compute a query of a path from the starting table to the target schema complement table.

Finally, the **Merge query manager** takes the *Merge Schema Augmentation* API call which has a start table, a target schema augmentation table, and several preferences of the merge query as inputs, it returns as output a final merge query with a list of computed metadata properties (see Section 4.5). The *Merge Query Generator* sub-manager cooperates with *Natural Query* to detect ambiguous values in the target schema augmentation and create a complement query if needed. Using *Metadata Propagator* sub-manager, the metadata of the merged result is propagated correctly, such that the result could be added to *Metadata Catalog* using *Declare Metadata* API.

4.2 Dimension and Fact Identifier Computation

In this section we describe the implementation details of constructing an attribute graph from dimension tables and computing dimension and fact identifier from an attribute graph. Attribute graphs and identifiers are the keys for finding schema complement and detecting ambiguous values, they play a central role in our work.

4.2.1 Computation of attribute graphs

As explained before, attribute graphs are metadata that are provided as a form of constraints over dimension tables and are fundamental information to compute identifiers and ensure non-ambiguity.

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 4.1. Table 4.1 shows tuples that encode the attribute graph of dimension $WAREHOUSE$ in Figure 2.6 (a) (Page 25).

Table 4.1: 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**

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 given dimension table using the indexing scheme of SAP HANA, called *Hierarchy Table* (HT) [35], which encodes the nodes of a hierarchy instance represented in a dimension or a fact table.

Hierarchy table

In SAP HANA, the nodes of a hierarchy instance represented in a dimension, or a fact table, are encoded using an indexing scheme into a corresponding *Hierarchy Table* (HT, for short)[4]. HT encodes the structure of hierarchy from the definition of hierarchy type and distinguishes nodes in the hierarchy instance by the unique

node values which is the complete path to the top-level node value, and contains information about its parent node, its level (or attribute) name, whether it is a leaf node, its level number in the hierarchy, etc. Given a dimension with unknown hierarchy, a hierarchy table can be used to build the attribute graph that satisfies the dimension using SQL queries.

Example 4.2. Node 'Dublin' of 'Ohio' in the hierarchy of GEOGRAPHY in Figure 2.4a (Page 19) is encoded as one tuple in HT (attribute names are on the left) shown in Table 4.2. 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.

Table 4.2: A tuple from hierarchy table

Attribute name	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

Use this hierarchy table, we can build attribute graph satisfies the related dimension table using Algorithm 1 (detail SQL queries see Appendix **??**).

The algorithm follows the Definitions 2.5 and 2.6 (Page 23) computing an attribute graph using queries that are linear with respect to the size of the dimension table. A user can then inspect the result and relax some constraints (e.g., switch a label **1** or **f** edge to label $+$, or remove an edge with label **f**) to reflect cases not captured by the dimension table.

Example 4.3. We illustrate the algorithm using Table 2.1 (Page 20) which is a hierarchy instance of type **GEOGRAPHY**. With an API call $ATG(HT, 2)$, the attribute graph satisfies dimension $REGION$ (DT_ID $= 2$) is constructed in following steps.

1. In step 1, ATGN is initialized with 6 nodes: $\perp$, CITY, STATE, COUNTRY, CONTINENT and $\top$ with LEVEL_NUM $= 0, 1, 2, 3, 4, 5$ respectively and OPTIONAL $= 0$ by default. ATGE is initialized with 6 edges: ($\perp$, CITY), (CITY, STATE), (STATE, COUNTRY), (CITY, COUNTRY), (COUNTRY, CONTINENT) and (CONTINENT, $\top$), edges ($\perp$, CITY) and (CONTINENT, $\top$) are labeled by **f** and $+$ respectively.

Algorithm 1 Attribute Graph Generation (ATG)

input: HT: hierarchy table of the hierarchy
id: unique id of the hierarchy

1. *Initialization*. All attributes in ATGN are initialized with OPTIONAL $= 0$; edges having PARENT_ATT_NAME $= \top$ are initialized with LABEL $= \mathbf{f}$; edges having ATT_NAME $= \perp$ are initialized with LABEL $= +$.
2. Find all attributes (except bottom level attributes $\perp$) in HT that have 'IS_LEAF $= 0$' and add an additional edge from $\perp$.
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 $\mathbf{f}$.
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 $\mathbf{1}$.

---

2. In step 2, value 'Antarctica' of attribute CONTINENT has IS_LEAF = 1 in HT, and CONTINENT is not a bottom level node. Thus, an edge ($\perp$, CONTINENT) is added to ATGE, because in CONTINENT, there is only one node value is leaf node, the edge is labeled by $\mathbf{f}$.

3. In step 3, attributes CITY, STATE and COUNTRY have null values in their NODE_VALUE, they are updated to OPTIONAL $= 1$ in ATGN.

4. In step 4, child-parent value mappings of all the edges are checked, edge (CITY, COUNTRY) is updated to label $+$ in ATGE, the other edges are updated to label $\mathbf{f}$ in ATGE.

5. Step 5 checks the mappings of edges (CITY, STATE) and (STATE, COUNTRY) which are labeled $+$ edges with ATT_NAME being optional, and (STATE, COUNTRY) is updated to label $\mathbf{1}$ in ATGE.

We show the final results of ATGN and ATGE on Table 4.3.

Table 4.3: Attribute graph for dimension REGION

(a) ATGN

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

(b) ATGE

DT_ID	ATT_NAME	PARENT_ATT_NAME	LABEL
2	__bot	CITY	+
2	CITY	STATE	+
2	STATE	COUNTRY	**1**
2	CITY	COUNTRY	+
2	COUNTRY	CONTINENT	**f**
2	CONTINENT	__top	**f**
2	__bot	CONTINENT	**f**

4.2.2 Dimension and fact identifiers

Given an attribute graph over some attribute hierarchies, a dimension identifier for all valid tables with respect to that attribute graph could be computed using the CDI algorithm as Algorithm 2.

The worst-case time complexity of the CDI algorithm is linear in the size (number of nodes and edges) of attribute graph $\mathcal{D}$.

Example 4.4. We apply function $CDI(\mathcal{D}, \mathrm{X})$ on dimension $REGION$ of Table 2.1 where $\mathcal{D}$ is the attribute graph computed in Example 4.3, X is all attributes of $REGION$. $dimId$ is firstly initialized with all attributes of $REGION$, and only CONTINENT is removed, because it has a label **f** in-edge. In Line 10, we get the result dimension identifier $dimId$ is {CITY, STATE, COUNTRY}.

By Proposition 2.5 (Page 30), the fact identifier of a fact table $T(\mathrm{S})$ defined over a set of dimensions $D_1(\mathrm{S}_{D_1}), \ldots, D_n(\mathrm{S}_{D_n})$ is computed by the union of the identifiers of each group of dimension attributes occurring in the fact table. The identifiers of each group of dimension attribute are also computed using CDI algorithm with the attribute graph of the dimension $D_i, i \in [1, n]$ and the set of dimension attributes in the fact table $\mathrm{S} \cap \mathrm{S}_{D_i}$ as inputs.

Algorithm 2 Compute Dimension Identifier (CDI)

input:
$\mathcal{D}$: Attribute graph of dimension table with schema S
X: A subset of S
output:
$dimId$: identifier of X

```
1: begin
2:     Initialize: dimId ← X
3:     for Attribute A in dimId do
4:         if (A has one label f in-edge from X in D) or
5:             (A only has label 1 in-edges from X in D)
6:          then
7:             dimId ← dimId − {A}
8:         end if
9:     end for
10:    return dimId
11: end
```

Example 4.5. We compute the fact identifier for fact table INVENTORY which is defined over dimensions $WAREHOUSE$, $TIME$, TAX and $PROD$ with attribute graphs shown in Figure 2.6 (Page 25). We first compute the identifier of dimension attributes from $WAREHOUSE$ by $CDI(\mathcal{D}, \mathrm{X})$ where $\mathcal{D}$ is the attribute graph of $WAREHOUSE$, $\mathrm{X} = \{$WH_ID, CITY, COUNTRY$\}$ are the set of dimension attributes from $WAREHOUSE$ occur in INVENTORY. $dimId$ is first initialized with attributes of X, CITY is removed because it has a label **f** in-edge starting from WH_ID, and the result $dimId$ is $\mathrm{K}_1 = \{$WH_ID, COUNTRY$\}$. Similarly, we compute the identifiers of dimension attributes from $TIME$, TAX and $PROD$. They are $\mathrm{K}_2 = \{$YEAR$\}$, $\mathrm{K}_3 = \{$TAX_NO$\}$ and $\mathrm{K}_4 = \{$PROD_SKU$\}$ respectively. Therefore, the fact identifier of INVENTORY is the union of $\mathrm{K}_1, \mathrm{K}_2, \mathrm{K}_3$ and K_4, $\mathrm{K} = \{$PROD_SKU, YEAR, WH_ID, COUNTRY, TAX_NO$\}$.

However, the fact identifier computed by Proposition 2.5 (Page 30) might not be minimal. When the fact table is defined as a view from a non-analytic table for which we know the primary 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 for the fact table, we refer this subset as *inferred key*.

Example 4.6. As explained in Example 2.14 (Page 29) that fact table INVENTORY is defined as a view over non-analytic table ct_INVENTORY, assuming that the primary key of ct_INVENTORY is $\mathrm{K}_{ct} = \{$PROD_SKU, YEAR, WH_ID, COUNTRY$\}$.

ct_INVENTORY(PROD_SKU, BRAND, YEAR, WH_ID, CITY, STATE, COUNTRY,

TAX_NO, RATE, TAX_DESC, QTY_ON_HAND, TAX_AMT)

Each attribute of the primary key has a corresponding attribute mapping with one attribute of the initial fact identifier K ={PROD_SKU, YEAR, WH_ID, COUNTRY, TAX_NO } computed by algorithm CDI as given in Example 4.5. K_{ct} is strictly included in K, so K_{ct} can be inferred as an *inferred key*, and INVENTORY now have two identifiers K, K_{ct} where K_{ct} is the minimal key. It implicitly follows that dimension TAX depends on the other dimensions of that fact table.

4.2.3 Maintaining dimension identifiers

An update of a dimension table may violate the attribute graph constraints defined for that table, and the dimension identifier generated from that attribute graph may also be affected. Thus, when a non-analytic table is updated, and a dimension table T is defined over it, we check the consistency of the attribute graph and the validity of the dimension identifier. If the test fails, the update must be rejected and the attribute graph of T can be inspected for further investigation. Since dimension tables are not frequently updated, the test entails a small overhead in the transaction processing workload. In addition, the test is efficiently done using Hierarchy Tables. (Detail query see Appendix **??**)

A strong assumption is that dimension table is viewed as an append-only table, we only detect and apply changes to the attribute graph that are caused by insertions, *i.e.*, new optional nodes, new edges and edges whose labels are changed into either **1** or $+$.

Table 4.4 shows the impacts on dimension identifiers caused by different changes on the in-edges of an attribute A of the attribute graph. Notice that the only operations allowed on the table is insertion, so the new in-edge added to A is either an edge start from $\perp$ or an edge start from its descendent. For a new label **f** in-edge of A, its impact on dimension identifier can not be identified immediately that when it's an edge starting from $\perp$, then A cannot be removed from identifier because it's not an in-edge from attributes in the hierarchy. When it's an edge starting from its descendent, then A can be removed from identifier. When a label **f** in-edge of A is changed to label **1** or $+$, the effects on dimension identifier need to recompute, whether A should be added to dimension identifier depends on if A still has label **f** in-edge.

Table 4.4: Effects on dimension identifiers by attribute graph

	A has **f** in-edge	A only has **1** in-edges	Other cases
f in-edge changes to **1**	Recompute	No effect	-
f in-edge changes to +	Recompute	-	-
1 in-edge changes to +	No effect	Add A	No effect
new **f** in-edge	No effect	No effect	Recompute
new **1** in-edge	No effect	No effect	No effect
new + in-edge	No effect	Add A	No effect

-: Not exist

4.3 Schema Complement Computation

4.3.1 Schema complement graph

As explained in Section 4.1, the Metadata Catalog contains the *native* relationships that are extracted from the definition of tables or explicitly declared by a user, and the *derived* relationships obtained using composition and fusion. From these relationships, schema augmentations could be discovered by exploring a Schema Complement graph defined as follows.

Definition 4.1 (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 (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).

Example 4.7. Figure 4.2 shows a partial $\mathcal{SC}$ graph for the relationships of Figure 2.9 (Page 37). Every relationship results in two edges of the $\mathcal{SC}$ graph, each edge is labelled with a reference to a relationship and its CT label. Nodes represent tables with their identifiers (ID).

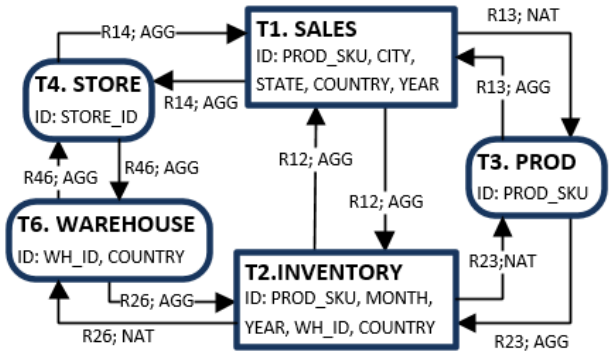


Figure 4.2: $\mathcal{SC}$ graph example for Figure 2.9

We use the term *connected* for tables which are connected by any path in a $\mathcal{SC}$ graph. A table T is said to be *reachable* from T_0, if they are connected through a sequence of natural '*NAT*' edges followed by at most one augmentation '*AUG*' edge. For example, T_1 and T_6 are connected, but T_6 is not reachable from T_1 because the path from T_1 to T_6 contains two '*AUG*' edges. T_1 and T_2 are connected and T_2 is reachable from T_1.

4.3.2 Finding schema augmentations

We now introduce the Get Schema Augmentation API which accepts a source table T_0 that user wants to augment as input, explores $\mathcal{SC}$ graph and returns an array of schema augmentation tables for T_0 as $\{(T_i, Type, Attrs, Paths), \ldots\}$. Each schema augmentation table T_i in the result is reachable from T_0, and the returned value contains the schema name T_i, $Type$: the complement type of T_i (*schema augmentation* or *natural schema complement*), $Attrs$: new attributes with respect to T_0 and $Paths$: all the possible paths start from T_0 to T_i that contain at most one 'AUG' edge. We use an example for illustration.

Example 4.8. In Figure 4.2, INVENTORY (T_2) is a schema augmentation to SALES (T_1). In a Get Schema Augmentation call with input table T_1, T_2 will be returned as one of the schema augmentation tables for T_1. Table 4.5 shows the information of T_2 returned by the API. There are two paths provided in the result that connect SALES to INVENTORY, note that there exists another path SALES $\rightarrow STORE \rightarrow WAREHOUSE \rightarrow$ INVENTORY in Figure 4.2. It is not returned in the result because this path contains two 'AUG' edges.

Table 4.5: T_2 in the result

PARAMETER NAME	VALUE
Name	INVENTORY
Type	schema augmentation
Attrs	{MONTH, WH_ID, TAX_NO, RATE, TAX_DESC, *QTY_ON_HAND*}
Paths	SALES $\rightarrow$ INVENTORY (default) SALES $\rightarrow STORE \rightarrow WAREHOUSE \rightarrow$ INVENTORY ($path_1$)

Get Schema Augmentation API is implemented by function Compute Schema Augmentation (*CSA*). *CSA* is a recursive function that takes two inputs: (T_0, $\mathcal{SC}$), and returns two outputs as described below:

Algorithm 3 Compute Schema Augmentation (CSA)

input:

T_0: start table

$\mathcal{SC}$: schema complement graph contains all nodes that connect to T_0

output:

$result$: $[(T_i, CT, Attrs, rid), \ldots]$

– T_i: schema augmentation table to T_0

– CT: complement type

– $Attrs$: new attributes

– rid: a relationship with T_i being the destination table

$altEdges$: $[(T_i, CT, rid), \ldots]$: list of optional relationships

Traversal step: Traversal all edges in $\mathcal{SC}$ that start from T_0. For each edge $\mathcal{E}(T_0, T_i)$ in $\mathcal{SC}$ that starts from T_0, verify if T_i is visited before.

- If T_i is visited, add $(T_i, \mathcal{E}(T_0, T_i).CT, \mathsf{R}(T_0, T_i))$ to $altEdges$.
  - $\mathcal{E}(T_0, T_i).CT$ is the *complement type* of $\mathcal{E}(T_0, T_i)$;
  - $\mathsf{R}(T_0, T_i)$ is the relationship related to $\mathcal{E}(T_0, T_i)$.
- If T_i is not visited, then mark it as visited, add $(T_i, \mathcal{E}(T_0, T_i).CT, Attrs, \mathsf{R}(T_0, T_i))$to $result$ and continue with *Recursive step*.
  - $\mathcal{E}(T_0, T_i).CT$ is the *complement type* of $\mathcal{E}(T_0, T_i)$;

- $Attrs$ is the set of new attributes in T_i with respect to T_0. If T_0 has never been augmented by T before, any attribute in T that is not in common with T_0 is considered to be new. Otherwise, attributes in T_0 that come from previous schema augmentations with T will not considered as new attributes.

- R (T_0, T_i) is the relationship related to $\mathcal{E}(T_0, T_i)$.

Recursive step: Recall $CSA(T_i, \mathcal{SC})$. When the *complement type* of $\mathcal{E}(T_0, T_i)$ is 'NAT', call $CSA(T_i, \mathcal{SC})$ and append the results to $result$, $altEdges$.

The *Traversal step* records all possible paths to a schema augmentation table T_i while avoids infinite loops during traversal by the annotation of visited. When multiple paths exist from T_0 to T_i, the relationship added to $result$ will be then set to the default path. The *Recursive step* continues the exploration only for 'NAT' edges and extend $result$ and $altEdges$.

Our implementation of the CSA algorithm leverages the SAP HANA graph engine [36] which supports various graph algorithms over graph data stored in a columnar table storage (using a table of nodes and a table of edges). We use the GET_NEIGHBORHOOD algorithm of the graph engine instead of calling the *Recursive step* to compute the set of all tables that are *reachable* from T_0. The GET_NEIGHBORHOOD performs a breadth-first search, the returned *reachable* tables are ordered according to their depth with respect to T_0, we store those tables and their complement types in $Reachable$. Then, we select all edges whose start node is T_0 or a table in $Reachable$ with complement type 'NAT' and destination node is in: $T_0 \cup Reachable$. Edges are ordered by their start tables according to the sequence of $Reachable$, when edges are sharing the same start table, they are ordered by their destination tables. Ordered edges are stored in $Edges$. Finally, we perform *Traversal step* for edges in $Edges$. Given the edges ordering 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 the final result, schema augmentations are returned sorted according to the depth-first search exploration of the $\mathcal{SC}$ graph.

Example 4.9. We illustrate our *iterative implementation* of algorithm CSA with input table T_1 on Figure 4.2 and the $\mathcal{SC}$ of Figure 4.2.

1. First, GET_NEIGHBORHOOD 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 $Reachable = \{(T_3, NAT), (T_4, NAT), (T_2, AUG), (T_6, AUG)\}$.

2. The next step retrieves the list of edges connecting nodes in $T_1 \cup Reachable$, ordered by the sequence of $Reachable$: $Edges = [\mathcal{E}(T_1, T_3), \mathcal{E}(T_1, T_4), \mathcal{E}(T_1, T_2), \mathcal{E}(T_3, T_2), \mathcal{E}(T_4, T_6)]$. Notice that edge $\mathcal{E}(T_4, T_1)$ is not selected because it starts from T_4 whose complement type is 'AUG'.

3. The algorithm then processes each edge in $Edges$ sequentially as follows.

   - For edges $\mathcal{E}(T_1, T_3), \mathcal{E}(T_1, T_4), \mathcal{E}(T_1, T_2)$, their destination tables T_3, T_4, T_2 have not been visited yet, the set of new attributes $Attrs_3, Attrs_4, Attrs_2$ are respectively computed for T_3, T_4, T_2 with respect to T_1, along with their complement type and related relationships are added to $Result$.

   - For edge $\mathcal{E}(T_3, T_2)$, the destination table T_2 is already visited, its complement type and related relationship R_{23} is added to $altEdges$.

   - For the last edge $\mathcal{E}(T_4, T_6)$, the destination table T_6 is not visited, the set of new attributes $Attrs_6$ is computed for T_6 with respect to T_4 using related relationship R_{46}, then added to $Result$.

Finally, we get $Result = \{(T_3, NAT, Attrs_3, \mathsf{R}_{13}), (T_4, NAT, Attrs_4, \mathsf{R}_{14}), (T_2, AUG, Attrs_2, \mathsf{R}_{12}), (T_6, AUG, Attrs_6, \mathsf{R}_{46})\}$ and $altEdges = (T_2, AUG, \mathsf{R}_{23})$.

Using the output of CSA, for each schema augmentation table, we construct a default path to the source table T_0 using the relationships in $Result$, and alternative paths are constructed using the relationships in $altEdges$.

Example 4.10. Continuing Example 4.9, default paths are constructed using relationships in $Result$ as: $T_1 \rightarrow T_3$ for $(T_3, NAT, Attrs_3, \mathsf{R}_{13})$, $T_1 \rightarrow T_4$ for $(T_4, NAT, Attrs_4, \mathsf{R}_{14})$, $T_1 \rightarrow T_2$ for $(T_2, AUG, Attrs_2, \mathsf{R}_{12})$, and $T_1 \rightarrow T_4 \rightarrow T_6$ for $(T_6, AUG, Attrs_6, \mathsf{R}_{46})$. The path from T_1 to T_6 is constructed by appending R_{46} after the path from T_1 to T_4 computed before. There is an additional relationship for table T_2 in $altEdges$ which means there is another path from T_1 to T_2. This additional relationship is between T_2 and T_3, by appending it to the default path from T_1 to T_3, we obtain the alternative path for T_2 as: $T_1 \rightarrow T_3 \rightarrow T_2$. By default, to augment T_1 with attributes from T_2, the default path which uses only R_{12} is selected when generating the merge query as $T_1 \rtimes T_2$, but user can still manually

select the alternative path which consists of R_{13}, R_{23} and obtain the merge query as $T_1 \rtimes T_3 \rtimes T_2$.

4.3.3 Unit conversions

Incompatibility between a start table T_0 and a target schema augmentation table T occurs when the two tables contain different measure units, statics scales, measurement details, etc., a major part of these are produced during data production which are out of our control. We mainly discuss here the incompatible measure units.

Before any further operations, i.e., applying reduction query, merging two tables, units and currency of T_0 and T should be unified. We use built-in SQLScript functions in HANA for units and currency conversion (function CONVERT_CURRENCY and CONVERT_UNIT) [37]. Conversions are computed in simple queries with attribute to be converted, source unit, target unit as mandatory parameters. When it's a currency conversion, the reference date is also mandatory.

Example 4.11. Consider the fact table SALES with aggregation attribute *AMOUNT*, as explained in Section 2.2.5, an attribute CURRENCY with role *detail* is associated with *AMOUNT* to describe the currency of the amount value. A CONVERT_CURRENCY call can convert *AMOUNT* values into *EUR* using the convert rate of '2013-09-23' as follows:

Listing 4.1: Query to convert currency

```
SELECT CONVERT_CURRENCY(
    AMOUNT => AMOUNT,
    "SOURCE_UNIT" => CURRENCY,
    "SCHEMA" => 'SYSTEM',
    "TARGET_UNIT" => 'EUR',
    "REFERENCE_DATE" => '2013-09-23')
AS CONVERTED_AMOUNT
FROM SALES;
```

Example 4.12. Consider the non-analytic table PRODUCT as introduced in Example 5.2 in Chapter 5, Page 119. It contains an attribute WEIGHT which records the products' weight using unit *gram(g)*, a CONVERT_UNIT call can convert values of WEIGHT into *pounds(lbs)* as follows:

Listing 4.2: Query to convert unit

```
SELECT CONVERT_UNIT(
    "QUANTITY" => WEIGHT,
    "SOURCE_UNIT" => 'G',
    "SCHEMA" => 'SYSTEM',
    "TARGET_UNIT" => 'LB')
AS CONVERTED_WEIGHT
FROM PRODUCT;
```

Besides the build-in functions, the conversion could also be done by referring conversion factors in [38], [39]. When the unit is not provided in the table, we assume that measures in two tables have the same unit.

4.4 Reduction Query Generation

The Compute Schema Augmentation (CSA) API returns a set of schema augmentations for a source table T_0. Assume that a user selects a schema augmentation table T with complement type 'AUG' and relationship path $\mathrm{R}_1(T_0, T_1), \cdots, \mathrm{R}_n(T_{n-1}, T), n >= 1$ 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, by Proposition 3.3 a natural schema complement to T_0 can be obtained by reducing all attributes in $\mathrm{K} - \mathrm{Y}_n$ through one or a sequence of nested reduction queries. Table 4.6 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 4.6: User actions to create a reduction query

User action	Impact on reduction query
Filter	defines a set $FilP$ of filter predicates $A_i = v_i, v_i \in dom(A_i)$; 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 $F_i(A_i)$; 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 $A = v$ for *filtered* attributes $A \in K$
$PivA$:	a set of *pivoted* attributes
$RemA$:	a set of attributes *removed* from K
$AggA$:	a set of *aggregated* attributes of the form $F(A)$ where $A \in$ S

output:

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

Preparation: 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 $\mathcal{Q}_{Fil} = \mathcal{F}\mathit{ilter}_T(FilP)$ is created on the target schema augmentation table T 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 $\mathcal{P}\mathit{ivot}_{\mathcal{Q}_{Fil}}(\mathrm{X}_{val} \mid PivA)$, 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: $\mathcal{A}\mathit{gg}_{\mathcal{Q}}(F_i(\mathrm{A}_i), \ldots \mid \mathrm{X})$, where each $F_i(\mathrm{A}_i)$ belongs to $AggA$ or to the new attributes whose values were pivoted in $\mathcal{Q}$, and $\mathrm{X} = \mathrm{K} - RemA - PivA$.

Step 4: Generate final reduction query. The output reduction query 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 2.2.6 Page 31). The identifier K′ of the result of $Q(T)$ is $K' = K - FilA - RemA - PivA$ where K is the identifier of T, $RemA$ and $AggA$ are defined in Table 4.6 and $FilA$ corresponds to the attributes in the filter predicates $FilP$.

Example 4.13. Continue with Example 4.9, T_2 = INVENTORY is returned as a schema augmentation to T_1 = SALES. When augmenting the schema of T_1 with T_2, a reduction query can be expressed as an RQG call: $RQG(T_2, \emptyset, \emptyset, RemA, \{\text{SUM}(\textit{QTY\_ON\_HAND})\})$, where $RemA = \{\text{MONTH}, \text{WH\_ID}\}$. Because $PivA$ and $FilP$ are empty, a single aggregate reduction is executed: $Q_{Agg} = \mathcal{Agg}_{T_2}(\text{SUM}(\textit{QTY\_ON\_HAND}) \mid \text{X})$, $\text{X} = \{\text{PROD\_SKU}, \text{YEAR}, \text{CITY}, \text{COUNTRY}\}$. Indeed, X is a subset of the identifier of T_2 and attributes MONTH and WH_ID have been reduced. Thus, the RQG call finally returns a reduction query Q_{Agg} and $\{\text{SUM}(\textit{QTY\_ON\_HAND})\}$ as a single aggregated attribute.

Example 4.14. An alternative way to reduce INVENTORY when augmenting the schema of SALES is to use an RQG call: $RQG(T_2, \{\text{WH\_ID} = \text{“Oh\_01”}\}, \{\text{MONTH}\}, \emptyset, \emptyset)$ on the $\mathcal{SC}$ graph of Figure 4.2. Because $AggA$ and $RemA$ are empty, only filter and pivot reductions are applied. A filter reduction is first created in step 1 as $Q_{Fil} = \mathcal{Filter}_{T_1}(\{\text{WH\_ID} = \text{“Oh\_01”}\})$. In step 2, the pivot reduction is applied on the result of Q_{Fil} as $Q_{Piv} = \mathcal{Pivot}_{Q_{Fil}}(\{\textit{QTY\_ON\_HAND}\} \mid \{\text{MONTH}\})$. Thus, the RQG call returns a query Q_{Piv}, and an empty set of aggregated attributes.

4.5 Merge Query Manager

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 $\text{R}_1(T_0, T_1), \cdots, \text{R}_n(T_{n-1}, T), n >= 1$. We now introduce the Merge Schema Augmentation (MSA) API, which takes five inputs (T_0, T, $path$, $noamb$, $comp$) and returns one merge query 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 $R_n(T_{n-1}, T) = R_n(T_{n-1}, T_{dest})$ maps to an edge in the schema complement graph $\mathcal{SC}$. Otherwise, $T = Q_{red}(T_{dest})$ represents the reduction query Q_{red} over T_{dest} obtained as an output of the Reduction Query Generation (RQG) API call, and $R_n(T_{n-1}, T)$ 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 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:

Q:	final merge query

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

Step 1: Create query Q_a to update ambiguous tuples in T.

- When flag $noamb = false$, this step does not check for ambiguous results and returns $Q_a = T$.
- When flag $noamb = true$, we first check if T is non-ambiguous. By Definition 3.13, detecting if T contains ambiguous values requires computing for each dimension $D(\mathrm{S}_D)$ used in T, the identifier of the ancestors $\mathrm{X}^*(D)$ of the dimension attributes $\mathrm{X}_D = \mathrm{S} \cap \mathrm{S}_D$. This identifier is computed over the attribute graph of dimension D restricted to the attributes in $\mathrm{X}^*(D)$ and using Proposition 2.1.
  - When T is not ambiguous for each dimension D in T w.r.t. Proposition 3.8, then we return $Q_a = T$.
  - Otherwise, when T is detected as possibly ambiguous with respect to a dimension D, we build a query 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 $comp = false$, this step does not compute completion tuples and returns $\mathcal{Q}_c = \emptyset$.
- When flag $comp = true$. By Proposition 3.10, we first check condition for $T_0(\mathrm{S}_0)$ and $T(\mathrm{S})$ that whether the LFD $(\mathrm{S}_{D_i} \cap \mathrm{S}) \mapsto (\mathrm{S}_{D_i} \cap \mathrm{S}_0)$ holds for all dimension D_i with schema S_{D_i} in $\mathrm{S}_0 \cap_D \mathrm{S}$.
  - When the condition does not hold, we return $\mathcal{Q}_c = \emptyset$.
  - When the condition holds, we compute the completion table $\mathcal{Q}_c = T^{com}$ as defined in Proposition 3.10 for T_0 and $\mathcal{Q}_a$ of Step 1.

Step 3: Create the final query $\mathcal{Q}$ merging T_0 with the results of Steps 1 and 2.

- Create a query $\mathcal{Q}_{path}$ 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, $\mathcal{Q}_{path} = T_0 \rtimes_{\mathrm{Y}_1} T_1 \cdots \rtimes_{\mathrm{Y}_{n-1}} T_{n-1}$.
- Merge $\mathcal{Q}_{path}$ with $\mathcal{Q}_a$ as $\mathcal{Q}' = \pi_{\mathrm{S}_0'}(\mathcal{Q}_{path} \rtimes_{\mathrm{Y}_n} \mathcal{Q}_a)$, where S_0' is the schema of T_0 augmented with the new attributes X_{new} coming from $\mathcal{Q}_a$.
- Finally, add $\mathcal{Q}_c$ (obtained from Step 2) to the previous result $\mathcal{Q}'$ and the final result is $\mathcal{Q} = \mathcal{Q}' \cup \mathcal{Q}_c$.

Example 4.15. Consider a merge schema augmentation call $MSA(\mathrm{T}_1, \mathcal{Q}(\mathrm{T}_2), \{\text{SUM(QTY\_ON\_HAND)}\}, \{\mathsf{R}_{12}\}, true, true)$ on the $\mathcal{SC}$ graph of Figure 4.2 (T_2 = INVENTORY is a schema augmentation to T_1 = SALES), and $\mathcal{Q}(\mathrm{T}_2)$ is the output of the RQG call in Example 4.13. Since $\mathcal{Q}(\mathrm{T}_2)$ is a reduction query, we add a natural complement edge $\mathsf{R}(\mathrm{T}_1, \mathcal{Q}(\mathrm{T}_2))$ with $CT =$ *'NAT'* to the $\mathcal{SC}$ graph.

Generate non-ambiguous result $\mathcal{Q}_a$. The input $noamb = true$, we build query $\mathcal{Q}_a$ to update ambiguous tuples in $\mathcal{Q}(\mathrm{T}_2)$. By Proposition 3.8, we verify the ambiguities for each dimension in $\mathcal{Q}(\mathrm{T}_2)$.

- For dimension $PROD$, $\mathrm{X}_D = \{\text{PROD\_SKU}\}$, $\mathrm{X}^*_D = \{\text{PROD\_SKU, SUBCATEGORY, CATEGORY}\}$, we have $\mathrm{X}_D \mapsto \mathrm{X}^*_D$, $\mathcal{Q}(\mathrm{T}_2)$ is not ambiguous with respect to dimension $PROD$.

- For dimension $TIME$, $\mathrm{X}_D = \{\text{YEAR}\}$, $\mathrm{X}^*_D = \{\text{YEAR}\}$, we have $\mathrm{X}_D \mapsto \mathrm{X}^*_D$, $\mathcal{Q}(\mathrm{T}_2)$ is not ambiguous with respect to dimension $TIME$.

- For dimension $WAREHOUSE$, $\mathrm{X}_D = \{\text{CITY, COUNTRY}\}$, $\mathrm{X}^*_D = \{\text{CITY, STATE, COUNTRY}\}$, we have $\mathrm{X}_D \not\mapsto \mathrm{X}^*_D$, $\mathcal{Q}(\mathrm{T}_2)$ is ambiguous with respect to dimension $WAREHOUSE$.

We generate $\mathcal{Q}_a$ that joins $\mathcal{Q}(\mathrm{T}_2)$ with dimension table $WAREHOUSE$ and sets aggregated measure attribute $t.$SUM(QTY_ON_HAND) to $null$ for all ambiguous tuples $t \in \mathcal{Q}(\mathrm{T}_2)$. A tuple t is ambiguous when its value pair of CITY, COUNTRY contains more than one STATE value in $WAREHOUSE$. For example, suppose that $WAREHOUSE$ contains two tuples t_1 and t_2 with the same values for attributes CITY, COUNTRY and different STATE values, *(Dublin, Ohio, United States)* and *(Dublin, California, United States)*. Query $\mathcal{Q}(\mathrm{T}_2)$ contains one tuple t with CITY, COUNTRY value being *(Dublin, United States)*, then t is ambiguous, $\mathcal{Q}_a$ will set the aggregated SUM(QTY_ON_HAND) value in t to $null$.

Generate the completion table $\mathcal{Q}_c$. The input $comp = true$, we build query $\mathcal{Q}_c$ to compute the completion table for T_1 and $\mathcal{Q}_a$. We first check the LFD condition. For dimension $PROD$, $TIME$, the common attributes between each dimension and $\mathcal{Q}_a$ can determine the common attributes between each dimension and T_1. But this condition does not hold for dimension $STORE$, as explained in Example 3.25 on Page 70. Therefore, we do not compute the completion table T^{com}, and return an empty table $\mathcal{Q}_c$.

Generate final merge query $\mathcal{Q}$. The path used to merge T_1 and $\mathcal{Q}_a$ contains only one relationship $R(\mathrm{T}_1, \mathcal{Q}_a)$ with complement type 'NAT', so we get $\mathcal{Q}_{path} = T_1$. The query for natural merge T_1 and $\mathcal{Q}_a$ is $\mathcal{Q}' = \pi_{\mathrm{S}'}(T_1 \rtimes_{\mathrm{Y}} \mathcal{Q}_a)))$, where S' contains the schema of T_1 augmented with attribute SUM(QTY_ON_HAND). Finally, we obtain $\mathcal{Q} = \mathcal{Q}' \cup \mathcal{Q}_c$.

Query Q is returned by this MSA call, and it is a non-ambiguous, complete merge of T_1 and $Q(T_2)$.

The complete workflow of augmenting the source table T_1 with attributes from T_2 is shown in Figure 4.3. Given a source table T_1, Get Schema Augmentation API first returns a list of schema augmentation tables where each schema augmentation table contains its complement type, new attributes it can bring, and paths connect to the source table. For a target table T_2 that is schema augmentation to T_1, user can use Reduction Query Generation API to generate a reduction query $Q(T_2)$ that transforms T_2 into a natural schema complement to T_1. Finally, using Merge Schema Augmentation API, a final merge result T_1' is generated which naturally merges T_1 with $Q(T_2)$, T_1' is non-ambiguous and complete.

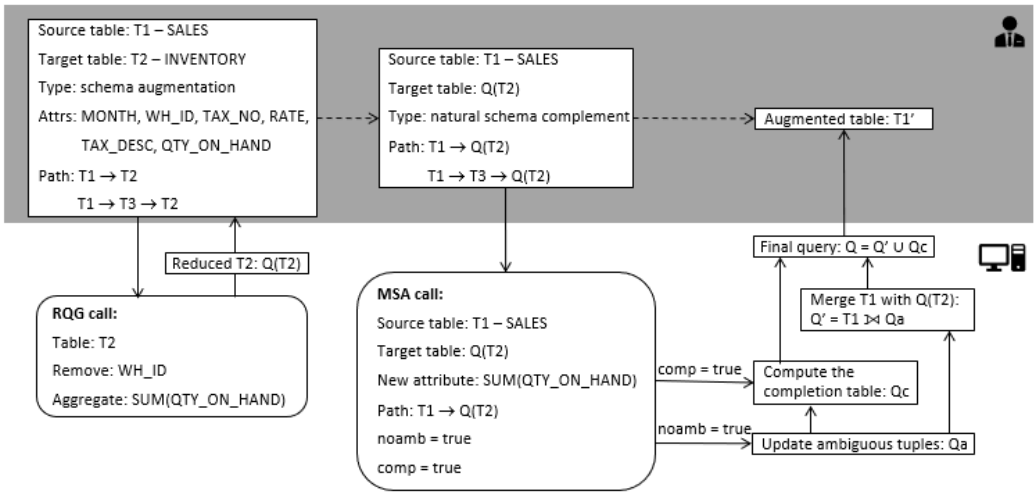


Figure 4.3: The complete workflow of merging T_1 and T_2

The following lemmas and proposition state that the RQG and MSA API compute a correctly merge result.

Lemma 4.1 (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.*, $T_0 \bowtie_{\mathrm{Y}_0} T$.

Proof. Let $T_0'(S_0')$ be the natural merge of T_0 and T, $T_0' = T_0 \bowtie_{\mathrm{Y}_0} T$.

There exists an attribute mapping relationship $\mathsf{R}'(T_0', T)$ on common attributes $\mathrm{Y}' = \mathrm{S}_0' \cap_D \mathrm{S}$, and $\mathrm{Y}_1 \subseteq \mathrm{Y}'$. And there exists a well-formed relationship $\mathsf{R}_1'(T_0', T_1)$

on attributes $\mathrm{Y}_1 \cap_D \mathrm{S}'_0$ 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 4.2 (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.*, $T_0 \rtimes\!\ltimes_{\mathrm{Y}_0} T$.

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

There exists an attribute mapping relationship $\mathsf{R}'(T'_0, T)$ on common attributes $\mathrm{Y}' = \mathrm{S}'_0 \cap_D \mathrm{S}$, 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 $\mathrm{Y}_1 \cap_D \mathrm{S}'_0$. Therefore, T_1 is a schema augmentation to T'_0. □

Proposition 4.1. 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 ignore steps 1 and 2 in a MSA call. Step 3 returns merge query $\mathcal{Q} = \pi_{\mathrm{S}'_0}(\mathcal{Q}_{path} \rtimes\!\ltimes_{\mathrm{Y}_n} \mathcal{Q}_a)$, where S'_0 is augmented S with attributes from $\mathcal{Q}_a$, $\mathcal{Q}_{path} = T_0 \rtimes\!\ltimes_{\mathrm{Y}_1} T_1 \cdots \rtimes\!\ltimes_{\mathrm{Y}_{n-1}} T_{n-1}$ 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 4.1, $\mathcal{Q}_{path}$ is a natural merge of T_0 with its natural schema complements $T_1, \cdots, T_{n-1}$. Also by Lemma 4.1, 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 4.2, 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 $\mathcal{Q}_a$). □

4.6 Extension to Heterogeneous Data Sources

There exist different types of data sources except the analytic views in SAP HANA, we now explain how our implementations could be generalized to heterogeneous data sources outside of SAP HANA. SAP HANA implemented a solution called *SAP HANA Smart Data Integration and SAP HANA Smart Data Quality (SAP HANA SDI)* to access heterogeneous data sources, to provision, replicate, and transform these data into table-based dataset in SAP HANA [40].

There are various data sources supported by SAP HANA SDI, includes SAP systems like SAP ABAP, SAP ASE database, SAP ECC and SAP HANA; databases like Apache Impala, Hadoop, IBM DB2, Microsoft SQL Server, Oracle and PostgreSQL; data preparation tools like Teradata; flat files like SharePoint, Microsoft Excel and PST files; even social media web site like Twitter and Facebook. To connect to source systems that no adapter is provided, SAP HANA also allows users to write their own adapters in JAVA using the Adapter SDK (Documentations of the Adapter SDK see [1]).

Each data source works with its specific adapter inside SDI, these adapters function like a bridge that provides a connection between the source system and SAP HANA as shown in Figure 4.4. User can preserve the connection to a source system as a *remote source*, objects inside the source system are then converted into HANA datatypes as *virtual tables* in the *remote source*. User are allowed to access objects in the *remote source* using two ways:

- A real-time data access. SAP HANA stores the schemas of virtual tables, any SQL queries that operate on virtual tables will then be translated and executed into an equivalent statement in the source system side.
- A snapshot data access. SAP HANA replicates the schemas of virtual tables and stores locally a snapshot of the data, SQL queries can be directly operate inside SAP HANA.

4.7 Conclusions

To conclude this chapter, we summarize the algorithms introduced in this chapter. As described in Chapters 2 and 3, dimension / fact identifier and attribute graph play an important role when defining natural schema complement, detecting ambiguous

[1] `https://help.sap.com/viewer/e974128d984d4bfeaa0c0b582ce24d79/2.0_SPS04/en-US`

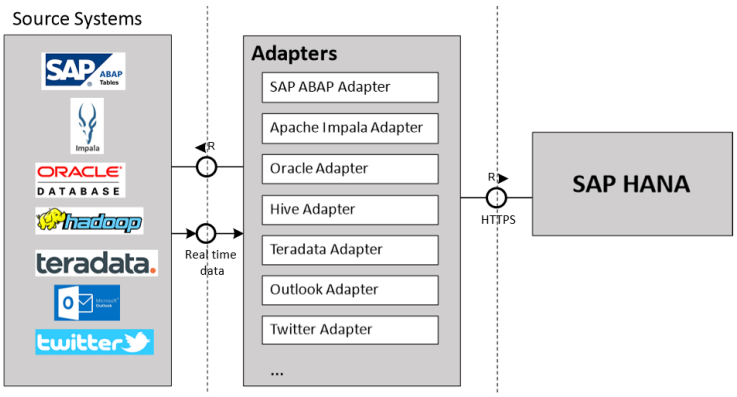


Figure 4.4: An overview of different adapters for SAP HANA

and guaranteeing complete merge. We first present in Section 4.2.1, algorithm *Attribute Graph Generation (ATG)* that computes an attribute graph for a given dimension, followed by algorithm *Compute Dimension Identifier (CDI)* in Section 4.2 that computes the dimension identifier from an attribute graph. With the knowledge of identifiers and the relationships (see Section 2.3), we introduce the notion of *Schema Complement Graph* which captures the types of connections between tables (i.e., schema augmentation or natural schema complement), and algorithm *Compute Schema Augmentation(CSA)* in Section 4.3 that explores schema complement graph and finds schema augmentations for a given source table. To merge the source table with a selected schema augmentation, we present algorithm *Reduction Query Generation (RQG)* in Section 4.4 which generates reduction queries to convert a schema augmentation into a natural schema complement and algorithm *Merge Schema Augmentation (MSA)* in Section 4.5 that merges source table and target table in a non-ambiguous, complete manner. All these algorithms are implemented in SAP HANA as REST APIs, and we show in Section 4.6 that using SAP HANA SDI, our implementations can be easily generalized to different input data sources.

5

State of the art

Contents

5.1 Introduction

The following chapter positions the contributions of this thesis with respect to the related work on different approaches of schema integration. The first part from Section 5.2 to Section 5.5 summarizes the historical evolution from early schema integration approaches, which required an important expertise and user effort to solve data heterogeneity issues, to recent "as-you-go" schema augmentation solutions for rapidly identifying and assembling semantically related datasets into customized user views. The second part, starting from Section 5.6, presents the relevant related work on data quality issues encountered by the assembly of multi-dimensional datasets.

5.1.1 Schema and data integration

The problem of discovering and merging related datasets has been studied for a long time in a variety of contexts. The first studies occurred in the area of database *schema integration* [41], which consists of integrating a given set of database schemas into a unified representation, usually called an *integrated* or *global schema*. The integration of different user views in a proposed database is also called *view integration* [41]. Two distinct tasks are involved in schema integration. The first task consists in selecting and analyzing the local database schemas according to specific information requirements. The second task is to compare the selected schemas.

The relationship of schema integration with the problem studied in this thesis is the following. Given a set of analytic schemas, a global schema then integrates all information contained in the analytic schemas and the schema designer defines all schema mapping queries to populate the global schema. Finally, a user can query the global schema and obtain answers that contain related data coming from one or more tables. We can see that with this approach the design and implementation effort rapidly become inefficient and complex for relating data coming from a large number tables. The number of mapping queries might rapidly increase and need a good knowledge of all available source schemas. This makes the schema integration approach inappropriate for the use cases considered in this thesis. Nevertheless, our approach shares some tasks with the the main schema steps integration including *schema comparison, conforming, merging* and *restructuring* that we describe in Section 5.2.

The *data integration* approach [42], [43] tries to solve some limitations of the schema integration approach, and in particular the issue of generating semantic

mapping queries. Compared to schema integration, data integration is not driven by a given set of data sources but defined according to the requirements of a database application. The data integration workflow starts by the creation of a *mediated schema independently* of some existing *source schemas*. Similar to schema integration, data integration methods also rely on a preliminary *schema matching* [44], [45] (see Section 5.3) phase to detect the relationships between the source schema elements (tables, attributes) and the mediated schema elements and the inter-source data dependencies. These schema matchings are used to iteratively suggest *schema mappings* for instantiating the mediated schema from the source data. Data integration systems are capable to interpret these schema mappings to either transform queries over the mediated schema into queries over the source data (view-based data integration) or to materialize the schemas into a physical database on which can be directly queried (data warehouse integration approach [46].

The problems studied in this thesis are closer to the data integration problem than to the schema integration problem (see Section 5.4 for examples). Assume a set of analytic tables, called *source tables*, for which some inter-table dependencies are known. Then a user can specify a mediated schema, called *target schema*, and express the schema matchings that exist between source schema elements and the target schema elements. The main specificity of our approach is that the user explores and specifies table relationships (schema matchings) which are partially defined in analytic schemas and metadata like foreign key constraints, queries and view definitions. The chosen relationships are used to generate a mediated target schema by augmenting a start source table with other source tables and also serve to generate merge queries which correspond to schema mappings in the general data integration model. The final result is a target view over the source tables that can be queried by the user.

More recently, new approaches have been proposed to support data integration capabilities "as you go", using the notion of dynamic *schema complement* [12] (Section 5.5). These approaches are user-driven and assist users in the construction of new datasets by assembling existing datasets. Starting from a given source dataset, the system suggests to the user other datasets whose schemas augment the schema of the source dataset with new attributes and whose data items are related to the data items of the source dataset. Not surprisingly, like in data integration, these methods require some knowledge about the inter schema dependencies that exist between the source dataset and the candidate datasets for schema augmentation. The approach developed in this thesis fall into this category and we shall position it with respect to the data integration approaches.

5.1.2 Drill-across and summarizability

Finally, in the context of OLAP systems, a lot of work have been done to determine under which conditions the data contained in different OLAP cubes could be combined using so-called *drill-across queries*[6]. Given a set of fact tables, Drill-across operation contains possibly sub-queries that aggregate fact tables to get a unified cardinalities and dimensionality. We shall first see in Section 5.6 how these conditions compare to the data quality guarantees that our proposed method provides when schema augmentation is applied to analytic datasets. We shall then position our work with respect to the problem of summarizability for aggregate tables, which provides conditions to guarantee that a correct computation of coarse-level aggregates can be obtained from fine-grain aggregates.

5.1.3 Schema augmentation

In this thesis, we focus on the approach called *schema augmentation* which improves the schema complement approach in a multidimensional context. This approach explores the relationships between dataset which are extracted automatically from the metadata of the analytic schemas like foreign key constraints, queries and view definitions. Starting from a given source dataset, the system relies on the relationships and discovers target datasets that are schema complements to the source dataset or could be transformed into schema complements to the source datasets. The query that is used to tranform a schema complement is called *reduction query*. A user chooses one target dataset to generate the merge query which is a left-outer join between the source dataset and the target dataset (or reduced target dataset). The schema mappings between the source dataset and the target dataset are automatically inferred from the relationships. We see in Section 5.5 the steps of the approach with respect to data integration, in Chapter 3 the formal definition of the schema augmentation approach and reduction query. The final result is a new dataset that consists of the source dataset and the new attributes from the target dataset.

5.2 Schema Integration

5.2.1 Approach

Schema integration [47] is the process of generating one or more integrated schemas from existing schemas. These schemas represent the semantics of the databases begin integrated and are used as input to the integration process. [41] provides a comparative review of schema integration issues using the Entity-Relationship (ER). In this seminal work, schema integration includes the notions of *view integration* and *database integration*. View integration is a design process generating a global conceptual description of a proposed database from multiple user-views whereas database integration produces a global schema of a collection of heterogeneous databases (with different data models and structures).

Our approach is more related to the notion of view integration which consists of four separate steps.

Step 1: Pre-integration : select and analyze the local schemas to be integrated according to specific information requirements.

Step 2: Schema comparison : compare the selected schemas and detect possible semantic and syntactic conflicts (schema matching)

Step 3: Conforming schemas : resolve the detected conflicts to enable schema merging. The schema integration process involves the user to discover and deal with the different naming and strucutural conflicts [41].

Step 4: Merging and restructuring : create and, if necessary, restructure the final integrated schema. Two strategies might be applied for the integration of multiple schemas. *Binary strategies* merge two schemas in each step whereas *N-ary strategies* can merge three or more schemas at a time.

5.2.2 Examples

Example 5.1. We now illustrate the use of a binary view integration strategy by an example.

Step 1: Pre-integration : Consider the schemas of the two non-analytic tables ct_STORE and ct_WAREHOUSE which are used to define the dimension tables $STORE$ and $WAREHOUSE$ as shown in in Figure 2.9, Page 37. Both tables defined an attribute ID as their primary key:

ct_STORE (ID, STORE_ID, CITY, STATE, COUNTRY, STORE_NAME, WEB_SITE, PHONE)
ct_WAREHOUSE(ID, WH_ID, CITY, STATE, COUNTRY, WH_NAME, SQ_FT, PHONE)

The two schemas ct_STORE and ct_WAREHOUSE can be integrated as follows into a new schema ct_STORE_WH.

Step 2: Schema comparison : The schema integration process involves the user to discover and deal with the different naming and structural conflicts [41]. For example, the user has to compare the attribute names and their domain values to verify whether two attributes are comparable in both schemas. Functional dependencies can also be used to identify possible structure conflicts generated by different key attributes. Name conflicts mainly appear for equivalent attributes with different names (synonym attributes) or non-equivalent attributes with the same name (homonym attributes). The attributes ID, PHONE, CITY, STATE and COUNTRY in both schemas are homonyms. The values of attribute ID in the two schemas are not comparable, since ID identifies a stores in table ct_STORE and warehouses in table ct_WHAREHOUSE. The user decides that attribute PHONE is also not comparable and all other homonym attributes describe the same concept. This results in the following schema matchings identified by the user:

ct_STORE.CITY $\cong$ ct_WAREHOUSE.CITY
ct_STORE.STATE $\cong$ ct_WAREHOUSE.STATE
ct_STORE.COUNTRY $\cong$ ct_WAREHOUSE.COUNTRY

Structure conflicts might, for example, appear through the existence of conflicting functional dependencies between equivalent attributes in both schemas. The functional dependencies in our two schemas are unknown and it is impossible to solve structure conflict.

Step 3: Conforming schemas : This step consists in resolving the conflicts identified in the previous step. The name conflict for attribute ID in ct_STORE and ct_WAREHOUSE can be resolved by adding a prefix to the attribute name in both schemas: CT_WAREHOUSE_ID for warehouse id and CT_STORE_ID for store id. The same prefix is added by the user to attribute PHONE to distinguish between warehouse

phone numbers and store phone numbers. The user continues to conform the two schemas

by (1) moving the matching attributes CITY, STATE and COUNTRY in ct_STORE and ct_WAREHOUSE from their original schemas into a new shared entity named LOCATION and (2) by creating a relationship through an artificial location identifier LOCATION_ID. The intermediate schema is shown in Figure 5.1. We use our graphical notations in the examples instead of the ER graphical notations originally used in [41]. Observe that all entities now have distinct attributes except LOCATION_ID which defines the relationships between the three tables.

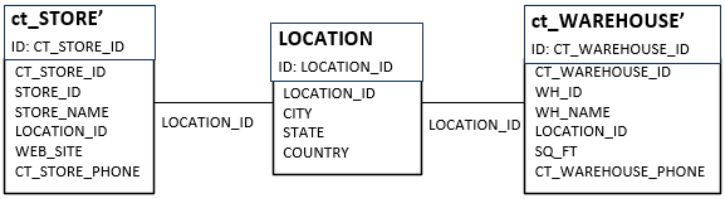


Figure 5.1: Transformed schemas and their relationships

Step 4: Merging and restructuring : The result of the design phase are the two transformed source schemas which are semantically related through a named relationship LOCATION. This can be understood in two different ways. One interpretation is to consider each table instance of the global schema as a view over the source schemas. The other interpretation is the existence of a join relationship between the two source schemas on attributes CITY, STATE and COUNTRY. The user adopts the second interpretation and merges ct_STORE and ct_WAREHOUSE through their common entity LOCATION to produces the following final global schema ct_STORES_WH:

ct_STORES_WH(CT_STORES_ID, STORE_ID, CITY, STATE, COUNTRY, STORE_NAME, WEB_SITE, CT_STORES_PHONE, CT_WAREHOUSE_ID, WH_ID, WH_NAME, SQ_FT, CT_WAREHOUSE_PHONE)

Observe that all previous steps require human decisions and SQL programming efforts to generate the final table.

[41] proposes the three quality metrics to evaluate the resulting integrated schema. The first quality criteria is *completeness and correctness* which validates if the integrated schema is able to represent correctly all information described by the initial schemas. In our example, the completeness and correctness criteria are ensured since every attribute in ct_STORE and ct_WAREHOUSE is kept in ct_STORES_WH and

there exist no functional dependencies that restrict the information that can be stored in the merged schema. The second criteria is *minimality*, which means that there are no redundant attributes in the integrated schema. This is ensured through the creation of the intermediate table LOCATION which factorizes the location attributes of both source tables. The last criteria is *understandability*, which reflects the facility of applying the different transformations to produce the integrated schema. Evaluating the understandability of the resulting global schema also requires human's judgement because there exists no quantitative and objective understandability score for global schemas. In our example, since the only schema conforming operation was to create table LOCATION, we can assume that if schemas ct_STORE and ct_WAREHOUSE were understandable, then ct_STORES_WH should also be understandable.

5.3 Schema Matching Discovery

As shown in the previous section, schema integration is a complex and time consuming task, mostly because it requires important user interaction for detecting and resolving semantic and syntactic conflicts between the source schemas. The goal of schema matching is to assist the users in the detection of semantic relationships the elements of different schemas. An abundant literature [44]–[46], [48] has proposed techniques that automatically discover schema matchings which reduce the user efforts required for schema and data integration. For instance, schema matching techniques can automatically discover join relationships using a combination of attribute name matching, data type matching and data instance matching. In this section, we mention a few techniques that could also be leveraged in our approach.

Schema matchings are classified as one-to-one matchings between two attributes (e.g., TIME.YEAR ≅ ALL_SALES.YEAR), and many-to-many matchings between multiple attributes (e.g., PERSON.NAME ≅ STUDENT.FIRST_NAME + STUDENT.LAST_NAME). In this thesis, we focus on one-to-one schema matchings.

We can mainly distinguish between two schema matching approaches [46] discovering either *heuristic* schema matchings through similarity computations or *reliable* schema matchings by extracting information from schema metadata.

5.3.1 Heuristic schema matching discovery

A common approach to estimate if two attribute from different schemas match (are compatible) is to apply some similarity measure. Similarities can be computed by comparing the attribute names, schema names, domain values, etc. We follow the categorization of [46] to discuss several similarity-based schema matching techniques in more detail. Providing an exhaustive description of the many techniques that have been published is out of scope of this chapter.

Name-based matching: The most intuitive way to match two table attributes by comparing their names. To measure the similarities between attribute names, string-matching algorithms like edit distance, Soundex or Jaccard measure [49] can be used. For example, consider attribute PHONE in table ct_STORE of schema Example 5.1 and Jaccard measure $J(x, y) = |B_x \cap B_y|/|B_x \cup B_y|$ between the bigram sets B_x and B_y of x and y respectively. Suppose we have attributes PHONE, ID and WH_ID. Then we can extract the following sets of bigrams for each attribute:

$$B_{PHONE} = \{\$P, PH, HO, ON, NE, E\$\}$$
$$B_{ID} = \{\$I, ID, D\$\}$$
$$B_{WH\_ID} = \{\$W, WH, H\_, \_I, ID, D\$\}$$

The Jaccard-measure between ct_STORE.PHONE and each attribute in ct_WAREHOUSE then are $J(\text{PHONE}, \text{ID}) == 0$, $J(\text{PHONE}, \text{WH\_ID}) = 0$ and $J(\text{PHONE}, \text{PHONE}) = 1$. Besides string-matching algorithms, there are various other ways to compute similarities between attribute names [46]. A more detailed discussion is out of the scope of this thesis.

Instance-based matching: Attributes also can be matched by comparing their value domains. A simple way is again to apply *Jaccard measure* [49] which estimates the overlap between two sets. For example, consider attribute CITY in table $SALESORG$ shown in Table 1.1 on Page 5 and Jacccard measure $J(\text{A}, \text{B}) = |N(\text{A}) \cap N(\text{B})|/|N(\text{A}) \cup N(\text{B})|$ between the domain value sets of attributes A and B. Suppose we have table

$REGION$ shown in Table 1.1. We extract the domain value sets for each attribute in $REGION$:

$$N(CITY) = \{Dublin, Paris, Berlin\}$$
$$N(STATE) = \{Ohio, California\}$$
$$N(COUNTY) = \{USA, Ireland, France, Germany\}$$
$$N(CONTINENT) = \{NorthAmerica, Europe\}$$

The Jarccard-measure between $SALESORG$.CITY and each attribute in $REGION$ are: $J(SALESORG.\text{CITY}, REGION.\text{CITY}) = 1/3$, $J(SALESORG.\text{CITY}, REGION.\text{STATE}) = 0$, $J(SALESORG.\text{CITY}, REGION.\text{COUNTY}) = 0$, $J(SALESORG.\text{CITY}, REGION.\text{CONTINENT}) = 0$. A threshold t should be defined such that when the similarity of two attributes value domains is higher than t, we might conclude that these two attributes match based on their instance. For example, when $t = 0.2$, we could infer a schema matching as: ct_STORE.CITY $\cong$ ct_WAREHOUSE.CITY.

Combined matching: Most of schema matching approaches adopt combined matching to compute similarities. Two attributes might have the same name but disjoint value domains. For example the domain of attribute PHONE in table $WAREHOUSE$ might be disjoint from the domain of attribute PHONE in table $STORE$, or, inversely, two attributes might contain the same domain values but have different names. The general architecture of combined matching systems [46] includes several name-based and instance based *matchers* producing different similarity matrices between schema elements. These matrices are processed by a *combiner* which produces a single matrix using some aggregation function (avg, min, max or weighted-sum). The produced matrix is transformed by a *constraint enforcer* which exploits external domain knowledge to improve the reliability. For example, product names more likely match with item names than city names. Finally, the enforced matrix is filtered by a *match selector* which selects a subset of matches. A simple selection strategy might select all matches above a certain threshold. Other strategies formulate the selection as an optimization problem like searching the subset of matchings which maximize the total similarity score and matching each attribute at most once.

5.3.2 Reliable schema matching discovery

Heuristic matching strategies can apply a rich set of similarity-based matchers for different data types. However, they represent at least two drawbacks. A first issue is their efficiency. Instance-based matching algorithm need to compare the domain values for several pairs of attributes in two schemas, which rapidly becomes very costly. The second issue concerns the quality of the obtained matchings. Similarity computations only provide approximate schema matchings with limited reliability guarantees for the produced matchings.

Other approaches exploit user-defined table constraints like foreign key constraints to infer schema matchings. The produced schema matchings are created based on user-defined logical schema constraints and more reliable than approximate similarity-based schema matchings. In this thesis, we only consider reliable schema matchings. Schema matchings are inferred by existing implementation of metadata extractor which analysis user-defined schema metadata and queries. (See Section 4.1 in Chapter 4).

Table constraints. PK-FK constraints defined in relational data models can be inferred as schema matchings. For example, consider the source schema in Example 5.2. The PK-FK constraints defined between tables PRODUCT and SUBCATEGORY on attribute SUBCATEGORY_ID and between table SUBCATEGORY and CATEGORY on attribute CATEGORY_ID directly define the following reliable constraints:

$$\begin{aligned} \text{PRODUCT.SUBCATEGORY\_ID} &\cong \text{SUBCATEGORY.SUBCATEGORY\_ID} \\ \text{CATEGORY.CATEGORY\_ID} &\cong \text{SUBCATEGORY.CATEGORY\_ID} \end{aligned}$$

View definitions. Relational and analytic schemas also might contain views which are defined by queries over other tables and views. As shown in Section 2.1, queries define attribute mappings between the view table and the query tables. For example, consider the source schema in Example 5.3, fact table (view) SALES is defined by a query over dimension tables $TIME$ and $SALESORG$. From this query, we can derive one-to-one attribute mappings between SALES and each dimension table:

$$\begin{aligned} \text{SALES.STORE\_ID} &\cong SALESORG.\text{STORE\_ID} \\ \text{SALES.CITY} &\cong SALESORG.\text{CITY} \\ \text{SALES.COUNTRY} &\cong SALESORG.\text{COUNTRY} \\ \text{SALES.YEAR} &\cong TIME.\text{YEAR} \end{aligned}$$

Conformed dimensions [6] and compatible dimensions [30] between fact tables also introduce schema matchings. In Example 5.8, conformed dimension $PROD$ between fact tables SALES and INVENTORY generates the following schema matchings:

$$\begin{aligned}
\text{SALES.PROD_SKU} &\cong \text{INVENTORY.PROD_SKU} \\
\text{SALES.BRAND} &\cong \text{INVENTORY.BRAND} \\
\text{SALES.PROD_SKU} &\cong \text{PROD.PROD_SKU} \\
\text{SALES.BRAND} &\cong \text{PROD.BRAND} \\
\text{INVENTORY.PROD_SKU} &\cong \text{PROD.PROD_SKU} \\
\text{INVENTORY.BRAND} &\cong \text{PROD.BRAND}
\end{aligned}$$

5.4 Mediation-based Data Integration

5.4.1 Approach

As explained in Section 5.1.1, data integration methods are not driven by a given set of data sources but defined according to the information requirements of a database application. A data integration workflow proceeds in three steps:

Step 1: Define a mediated schema: specify the schema of the mediated table for some specific application needs.

Step 2: Specify schema matchings: discover schema matchings between the elements (attributes, tables) of the mediated schema and the source schemas.

Step 3: Infer schema mappings: infer schema mappings from the specified schema matchings.

These steps can be interleaved until a complete schema mapping is obtained for the mediated schema. The notion of completeness here means that an instance of the mediated schema that fulfills all its integrity constraints can be computed using the schema mapping. For more details, we refer the reader to [43], [46].

The schema augmentation problem can be considered from a data integration perspective. Based on the data model introduced in Chapter 2 we can define the data integration steps as follows:

Step 1: Define mediated schema: In our model, the mediated schema corresponds to a source schema extended by some new attributes. The data integration process starts by choosing one *start* table, *e.g.* PRODUCT and a set of

additional information items/attributes, *e.g.* STOCK, which should be added to the start table.

Step 2: Specify schema matchings: In our schema augmentation approach, we use reliable schema matchings (Section 5.3) which are formally captured by join relationship and attribute mapping relationship as described in Section 2.3. Thus, schema matching specification corresponds to extracting and inferring join and attribute mapping relationships which connect the start table with other source schemas that can provide the required additional attributes. These other tables are called target table candidates.

Step 3: Infer schema mappings: This step consists in selecting a subset of candidate target tables where each table can provide one or more new attributes. Thus, schema mappings correspond to merge queries which join the start table with the selected target tables to instantiate the mediated schema. Formally, this step corresponds to find and merge schema augmentations using the exploration of a *schema complement* graph as described in Chapter 3.

We provide a few examples to facilitate the comparison of data integration solutions with the approach proposed in this thesis.

5.4.2 Examples

Example 5.2. Suppose a user wants to generate a simple dataset of one table with information about products like their brand, category, and the amount sold per year.

Step 1: Define mediated schema: The user defines a mediated schema consisting of a single table SALES_PRODUCTS:

SALES_PRODUCTS(PROD_SKU, BRAND, YEAR, CATEGORY_NAME, SUBCATEGORY_NAME, *AMOUNT*)

Step 2: Specify schema matchings: The source schema contains the following tables SALES, PRODUCT, CATEGORY and SUBCATEGORY from Figure 2.9 on Page 37:

SALES (PROD_SKU, BRAND, YEAR, CITY, STATE, COUNTRY, *AMOUNT*)
PRODUCT (PRODUCT_SKU, PRODUCT_NAME, BRAND_NAME, WEIGHT, SUBCATEGORY_ID)
CATEGORY (CATEGORY_ID, CATEGORY_NAME)
SUBCATEGORY (CATEGORY_ID, SUBCATEGORY_ID, SUBCATEGORY_NAME)

The underlined attributes represent the primary key in each table.

Using the schema complement approach, the user will first choose a "start" table that contains all key attributes of the mediated schema (table). In our case, the only possible start table is SALES, which contains both key attributes PRODUCT_SKU and YEAR of table SALES_PRODUCTS. Figure 5.2 shows the start table and the existing join relationships (solid lines) which are extracted from foreign key constraints and view definitions of the source schema. Schema matchings between the mediated schema and the source schemas that would be used to infer schema mappings are indicated by dashed arrows.

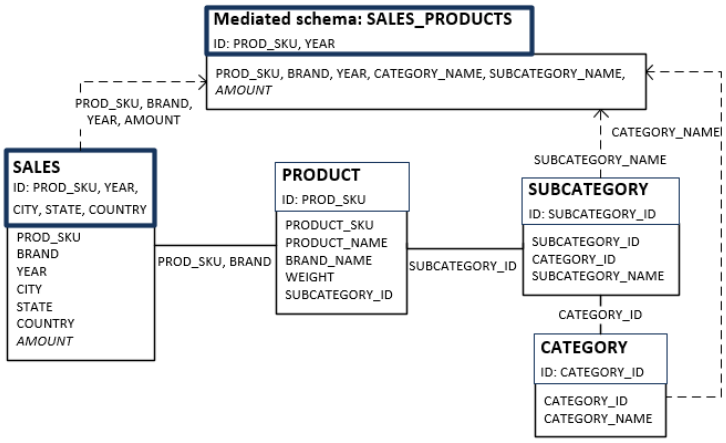


Figure 5.2: Join relationships and schema matchings

Table 5.1 shows the schema matchings specified between the source schemas and target (mediated) schema using the notation of [44]. Attributes of table SALES_PRODUCT are matched with attributes of SALES, SUBCATEGORY and CATEGORY, we call this set of schemas a *cover* of the mediated schema [46].

Table 5.1: Schema matchings from Figure 5.2

SALES.PROD_SKU	$\cong$	SALES_PRODUCT.PROD_SKU
SALES.BRAND	$\cong$	SALES_PRODUCT.BRAND
SALES.YEAR	$\cong$	SALES_PRODUCT.YEAR
SALES.*AMOUNT*	$\cong$	SALES_PRODUCT.*AMOUNT*
SUBCATEGORY.SUBCATEGORY_NAME	$\cong$	SALES_PRODUCT.SUBCATEGORY_NAME
CATEGORY.CATEGORY_NAME	$\cong$	SALES_PRODUCT.CATEGORY_NAME

We can observe, for each attribute of SALES_PRODUCT, there exists one and only one matching between SALES, SUBCATEGORY and CATEGORY, we call them a *minimal cover* of the mediated schema. Therefore, the schema matchings PRODUCT.PRODUCT_SKU $\cong$ SALES_PRODUCT.PROD_SKU, PRODUCT.BRAND

$\cong$ SALES_PRODUCT.BRAND are not shown in Figure 5.2, since attributes SALES_PRODUCT.PROD_SKU and SALES_PRODUCT.BRAND already match with attributes PRODUCT_SKU and BRAND of table SALES.

There is a difference between our schema augmentation approach and the more general data integration. In schema augmentation, the user selects a minimal cover of the mediated schema, whereas data integration uses several covers of the mediated schema [46]. This restriction reduces the complexity of the schema mapping queries (see below) generated by schema augmentation and avoids complex data data fusion operations [50].

Step 3: Infer schema mappings: Given a set of selected schema matchings, schema mapping inference consists in generating a merge query for populating the mediated schema. In our example, all attributes can be provided by tables SALES, CATEGORY and SUBCATEGORY. However, as we can see in Figure 5.2, table SALES is not directly related to the category dimension tables and we also need table PRODUCT to find all categories and sub-categories of a product. A suggested schema mapping then consists of performing a left outer-join between tables SALES, PRODUCT, CATEGORY and SUBCATEGORY, followed by a projection on the attributes of SALES_PRODUCTS. A left outer-join is used to keep all products in table SALES (the primary key attribute PROD_SKU of the target table is matched with attribute PROD_SKU in table SALES). The outer join might generate null values for non-key attributes (due to missing tuples in the other tables). If there exists a constraint indicating that all attributes of the mediated schema must be non-null, an inner join can be applied instead, but this would result in missing certain products in the mediated schema.

Example 5.3. As a second data integration scenario, we use the example in Section 1.2 on Page 4.

Step 1: Define mediated schema: The user defines a mediated schema consisting of a single table SALES_DEM which extends an existing table SALES:

SALES_DEM (STORE_ID, CITY, COUNTRY, YEAR, *AMOUNT*, *POP*, *MIN_UNEMP*, *MAX_UNEMP*)

Step 2: Specify schema matchings: The source schema consists of the following five tables SALES, DEM, $SALESORG$, $TIME$ and $REGION$.

SALES (STORE_ID, CITY, COUNTRY, YEAR, *AMOUNT*)
DEM (CITY, STATE, COUNTRY, YEAR, *POP*, *UNEMP*)
$SALESORG$ (STORE_ID, CITY, STATE, COUNTRY)
$TIME$ (DATE, WEEK, MONTH, YEAR)
$REGION$ (CITY, STATE, COUNTRY, CONTINENT)

The join relationships between these source tables are as depicted in Figure 5.3 by solid lines. Schema matchings between the elements (tables, attributes) of the mediated schema and the elements of source schema that would be used are indicated by dashed arrows.

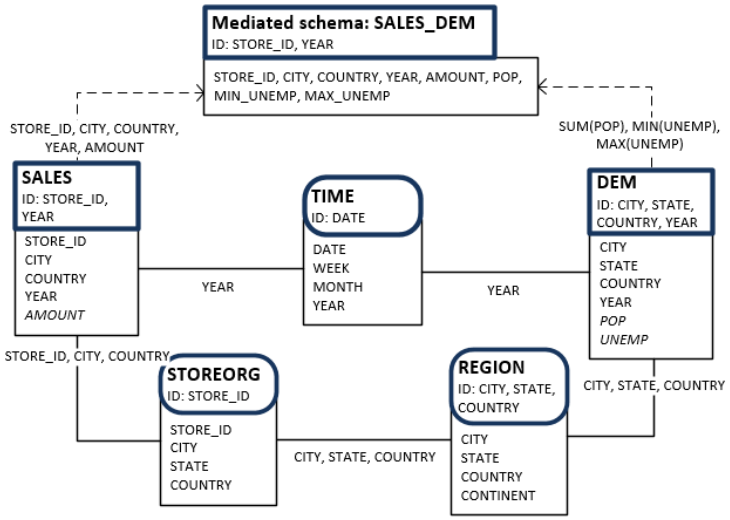


Figure 5.3: Join relationships and schema matchings

Table SALES_DEM does not contain a matching to the attribute STATE of table DEM which is the identifier of DEM. Since measure attributes POP and UNEMP depend on this attribute, the schema matching "reduces" table DEM by aggregating the two measures POP and UNEMP along attribute STATE. For this, we extend our notations of schema matchings with aggregate functions (see Table 5.2).

The schema matchings between SALES_DEM and SALES, DEM shown in Table 5.2 is a minimal cover of the mediated schema, there exist no other schema matchings between attributes STORE_ID, CITY, COUNTRY of table $SALESORG$ and SALES_DEM, or attributes CITY, COUNTRY of table $REGION$ and SALES_DEM. When specifying the schema matchings with aggregate functions, a first challenge for the user is to choose the correct aggregation functions that can be applied to the attributes

Table 5.2: Schema matching extracted from Figure 5.3

SALES.STORE_ID	$\cong$	SALES_DEM.STORE_ID
SALES.CITY	$\cong$	SALES_DEM.CITY
SALES.COUNTRY	$\cong$	SALES_DEM.COUNTRY
SALES.YEAR	$\cong$	SALES_DEM.YEAR
SALES.*AMOUNT*	$\cong$	SALES_DEM.*AMOUNT*
SUM(DEM.*POP*)	$\cong$	SALES_DEM.*POP*
MIN(DEM.*UNEMP*)	$\cong$	SALES_DEM.*MIN_UNEMP*
MAX(DEM.*UNEMP*)	$\cong$	SALES_DEM.*MAX_UNEMP*

of the source tables. Specifically, the user must know that *POP* can be summed and averaged, but *UNEMP* can only compute a maximal or minimal value because it describes an unemployment ratio. The capability of aggregation is called *aggregable properties* introduced in Section 2.2.6 on Page 31 which can assist user during the "reducing" process.

Step 3: Infer schema mappings: Assuming that the above matchings are correctly specified, then $\{\text{SALES}, \text{DEM}\}$ is a cover of the mediated schema selected by the user (table $TIME$ is not necessary since join attribute YEAR appears in both tables). The corresponding *schema mapping* is a nested query which first joins SALES with DEM on their common attributes YEAR, CITY, COUNTRY followed by an aggregation of attributes *POP* and *UNEMP* grouped by all other attributes of SALES_DEM, *i.e.* STORE_ID, CITY, COUNTRY, YEAR, AMOUNT.

Listing 5.1: Data integration approach of joining SALES with DEM

```
SELECT SALES.STORE_ID, SALES.CITY, SALES.COUNTRY, SALES.YEAR, SALES.AMOUNT,
       AVG(DEM.POP) AS POP,
       MIN(DEM.UNEMP) AS MIN_UNEMP,
       MAX(DEM.UNEMP) AS MAX_UNEMP
FROM SALES LEFT OUTER JOIN DEM
       ON SALES.CITY = DEM.CITY
       AND SALES.COUNTRY = DEM.COUNTRY
       AND SALES.YEAR = DEM.YEAR
GROUP BY STORE_ID, CITY, COUNTRY, YEAR, AMOUNT
```

Because attribute STORE_ID of SALES does not exist in table DEM and STORE_ID is the key attribute of SALES, the left-outer join between SALES and DEM duplicates the values of POP and UNEMP in the left-outer joins. Therefore, the schema mappings generates incorrect values for attributes *POP* in SALES_DEM (*MIN_UNEMP* and *MAX_UNEMP* are computed by function MINand MAX, they are not effected by the duplication). Existing data integration methods are neither detecting nor resolving

this problem. This is yet another difference between our schema augmentation approach and the general data integration approaches, we generate the corresponding schema mapping by first reducing DEM using an aggregation of attributes *POP* and *UNEMP*, then joins SALES with the reduced DEM. The corresponding schema mapping generated in our schema augmentation approach is as follows:

Listing 5.2: Schema augmentation approach of joining SALES with DEM

```
SELECT SALES.STORE_ID, SALES.CITY, SALES.COUNTRY, SALES.YEAR,
       SALES.AMOUNT, AGG_DEM.POP, AGG_DEM.MIN_UNEMP, AGG_DEM.MAX_UNEMP
FROM SALES LEFT OUTER JOIN
       ( SELECT CITY, COUNTRY, YEAR,
         AVG(DEM.POP) AS POP,
         MIN(DEM.UNEMP) AS MIN_UNEMP,
         MAX(DEM.UNEMP) AS MAX_UNEMP
         FROM DEM
         GROUP BY CITY, COUNTRY, YEAR
       ) AGG_DEM
  ON SALES.CITY = AGG_DEM.CITY
  AND SALES.COUNTRY = AGG_DEM.COUNTRY
  AND SALES.YEAR = AGG_DEM.YEAR
```

An alternative way to generate the mediated schema of Listing 5.2 using data integration approach is to define an intermediate table: AGG_DEM, the intermediate table stores the result of an aggregation on DEM that computes AVG(POP), MIN(UNEMP) and MAX(UNEMP) groups by CITY, COUNTRY, YEAR. Then schema matchings between the mediated schema SALES_DEM and AGG_DEM can be specified between attributes POP, MIN_UNEMP and MAX_UNEMP. Finally, the schema mappings is inferred as a left-outer join between SALES and AGG_DEM.

The previous examples show that data integration techniques facilitate the generation of schema mappings mainly by requiring a user to select schema matchings or providing a minimal cover of the mediated schema. Data integration approach also deal with the cases when several minimal covers are provided, a schema mapping would be generated for each minimal cover then combined into one large query using union. A similar approach called *Entity Complement* which also takes a combination of schema mappings will be introduced in Section 5.5.2.

Obviously, the methods presented before for the discovery of schema matching and the generation of schema mappings can reduce the human effort. More sophisticated methods have been proposed to refine the suggested schema mappings such as the exploitation of examples, as described in [43]. However, with respect to our

requirements explained in Chapter 1, the work required for the user is still beyond what is expected.

5.5 Schema Augmentation and Entity Complement

5.5.1 Schema augmentation approaches for web tables

Schema augmentation is a special approach for data integration which does not require the creation of a mediated schema. Thus all tables are part of a source database schema. A schema augmentation workflow proceeds in three steps:

Step 1: Select start table: the user selects a start table whose schema will be augmented using the schema of other source tables.

Step 2: Find candidate tables: the user explores the suggested schema matchings and finds a set of matching candidate tables, called target tables, that can provide new attributes to the schema of the start table.

Step 3: Generate augmented table: the schema mapping queries between the start table and a selected candidate target table are generated. These mapping queries connect both tables through a "path" of matching tables. The merge mainly consists of performing a left outer-join between the start table, the matching tables and the target table.

More generally, schema augmentation can be applied to any kind of database table, materialized or virtual (defined by a view). We now introduce several specific schema augmentation approaches.

Schema complements for web tables

Originally, the idea of *schema complement* was developed in the context of web tables [12] with the idea of “adding as many properties as possible to the entities of a given table while preserving the consistency of its schema”. Schema complement is a special case of schema augmentation where the augmented source table has the same number of tuples as the start table. The method proposed in [12] is limited to what we call *natural schema complement* in Section 3.2 on Page 46. The key to determine if a target table is a schema complement to some start table is that the attributes in the target table which match with the start table, called *common attributes*, are a key in the target table. Then, by applying left outer joins, the merge

of the start table with a schema complement table preserves all tuples in the start table without any duplication.

Example 5.4. Assuming that we have a schema contains the two tables ct_STORE and ct_WAREHOUSE presented before in Example 5.1 on Page 111, and assume that their primary keys are respectively, (STORE_ID) and (WH_ID, COUNTRY).

Step 1: Select start table: User selects ct_STORE as the start table.

Step 2: Find candidate tables: The techniques presented in [12] are able to discover that the domains of the attributes CITY, STATE, COUNTRY in ct_STORE and ct_WAREHOUSE are largely overlapping and therefore are comparable (match). Schema matchings are defined for these attributes as follows:

$$\begin{aligned} \text{ct\_WAREHOUSE.CITY} &\cong \text{ct\_STORE.CITY} \\ \text{ct\_WAREHOUSE.STATE} &\cong \text{ct\_STORE.STATE} \\ \text{ct\_WAREHOUSE.COUNTRY} &\cong \text{ct\_STORE.COUNTRY} \end{aligned}$$

The common attributes between ct_STORE and ct_WAREHOUSE are CITY, STATE, COUNTRY, but these attributes do not functionally determine all attributes in ct_WAREHOUSE. Therefore, ct_WAREHOUSE is not a schema complement to ct_STORE and since there is no other candidate table, we can not complement table ct_STORE.

Example 5.5. In our second scenario, suppose that we have the same source tables as in Example 5.2, the identifier of each table being indicated by underlined attributes.

Step 1: Select start table: User selects SALES as the start table.

Step 2: Find candidate tables: The schema matchings are already known as specified in Example 5.2 as shown in Figure 5.2. Table PRODUCT is a candidate schema complement to SALES, and since their common attributes PROD_SKU and BRAND contains the primary key PROD_SKU of PRODUCT (functionally determine all attributes of PRODUCT), table PRODUCT is a schema complement to start table SALES.

Step 3: Generate augmented table: The corresponding schema mapping is a merge query that takes a left outer-join of table SALES with PRODUCT on their common attributes PROD_SKU and BRAND. The result of the merge is an augmented table SALES_CAT with the new attributes SUBCATEGORY_ID and SUBCATEGORY_NAME.

Notice that the other candidate table SUBCATEGORY is not a schema complement to SALES. However it becomes a schema complement to the augmented table SALES_CAT since there exists a schema matching between SALES_CAT and SUBCATEGORY on attribute SUBCATEGORY_ID which is the primary key of SUBCATEGORY. Similarly, after merging SALES with SUBCATEGORY and adding the new attributes SUBCATEGORY_NAME and CATEGORY_ID, table CATEGORY also becomes a schema complement and can be merged to add attribute CATEGORY_NAME.

After three iterations, the final augmented SALES_PRODUCTS corresponds to the mediated schema of Example 5.2 with the guarantee that there exists exactly one tuple in the augmented table for each tuple in SALES (no tuple duplication).

Compared to schema integration (Section 5.2) which integrates an arbitrary number of source schemas with a pre-defined mediated schema, schema complement builds the new integrated schema incrementally. Instead of defining a mediated schema containing all attributes (Step 1), it starts from a start table and uses schema matchings (Step 2) to identify the target tables that can provide new attributes. By incrementally extending the start table with new attributes from the target tables, the final result is a complete mediated schema which can be expressed by a single schema mapping query that contains a set of selected source tables. This schema complement process always makes sure that the common attributes between the start table and the target table are a key of the target table. This avoids that the left-outer join generates duplicates for the tuples in the start table. However, this restriction also reduces the number of candidate target tables which can be merged with the start table. For example, it cannot address the use case of Example 5.3 on Page 121, because it does not support merge queries with aggregation.

The second goal of building schema complements is to bring a maximum number of new attributes and values. In particular, the approach proposed in [12] is able to populate new attributes describing the same concept from different tables. For example, if ct_STORE is the start table, it is possible to map new attributes CITY_NAME and CITY_NAME_ENG with the same domain values to a single attribute CITY through schema complements and to populate this attribute with the values from both domains. On the other hand, two attributes with the same name but describe

different concepts (name conflict) are mapped to two different attributes by a adding a prefix to the attribute names.

In conclusion, the schema complement approach proposed in [12] can be considered as an efficient solution to build new tables integrating data from different other source tables with minimal user interaction. Moreover, if the key attributes of schemas and relationships between schemas are known, data integration using schema complement becomes a completely automatic task. However, because of the restriction that common attributes should contain the identifiers of target tables, schema complement is not able to deal with target tables that are need to be reduced, like applying an aggregation.

Keyword-based schema complement discovery

The *OCTOPUS* system [51] implements a series of integration-related operations like search, extraction, data cleaning and integration within web tables. The integration operation *EXTEND* enables users to "find related tables that can be joined to add new attributes to a table". Compared to schema complement[12], which automatically detects and adds a maximum number of attributes to the start table, *EXTEND* requires two user-defined parameters to control the augmentation process. Given a start table T, an attribute name A and a keyword k, a web search engine returns a set of candidate tables extracted from the web pages containing keyword k. The candidate tables are ordered by the probability of co-occurring with keyword k and by the Jaccard distance between their attributes and attribute name A. Finally, the highest scored schema matchings between the candidate attributes and attribute name A are selected to build schema mappings.

Example 5.6. Consider the source schema in Example 5.2, Page 119. Given the start table PRODUCT, the attribute SUBCATEGORY_ID and a keyword "name", an augmented schema generated by *EXTEND* is PRODUCT′ with a new attribute whose name is similar to "name" and is related to SUBCATEGORY_ID. Firstly, search operation returns two candidate tables SUBCATEGORY and CATEGORY that respectively contain attributes with the keyword "name": SUBCATEGORY_NAME and CATEGORY_NAME. By computing the compatibility between the values in SUBCATEGORY_ID of table PRODUCT and each attribute in tables SUBCATEGORY and CATEGORY, attribute SUBCATEGORY_ID in table SUBCATEGORY is more related because the FK-PK constraints, all the values of PRODUCT.SUBCATEGORY_ID are contained in SUBCATEGORY_ID.SUBCATEGORY. Therefore, PRODUCT′ can be generated by joining PRODUCT with SUBCATEGORY on SUBCATEGORY_ID.

The *InfoGather* system introduced in [13] proposed an operation *ABA (Augmentation By Attribute name)* which corresponds to algorithm *JoinTest* used by the previous *EXTEND* operator [51]. Given a start table T, an ABA operation performs a merge query that augments the start table with the specified new attribute A. This query is noted as $Q(\mathrm{K}, \mathrm{A})$ where K is the key attribute of start table T, A is the new attribute name. Schema matchings between the start table and the target table are detected when the target table contains K as the key attribute and the domain values of the two key attributes overlap. Besides, tables containing attribute A are also considered as target tables. Target tables are then ordered by their matching scores with the start table (i.e., percentage of matched values of K). The final schema mapping query applies left outer join of the start table and the target table with the highest score on the key attribute K. Although ABA operation restricts the key of the start table (and the target table) to consist only of one attribute, this could be generalized to a set of attributes.

Other systems like *Data Civilizer* [14] propose a similar operation called *extend attribute* which describes queries that extend the start table T with *new* attributes that do not exist in T. *Data Civilizer* constructs a schema matching graph called *linkage graph* and stores schema matchings that are computed using similarity computations (schema level and entity level), PK-FD constraints, inclusion dependencies, etc.

5.5.2 Entity complement approaches

Entity complements for web tables

Schema augmentation generates schema mapping queries using left outer joins for adding *new attributes* to the tuples of some existing source table. This kind of "*vertical*" schema augmentation can be combined with "*horizontal*" entity/data completion approaches which consist in adding *new tuples* by union.

Entity complement was first introduced in the context of Web tables in [12] with the idea of "providing complementary sets of entities to the source schema by union". For taking the union of two tables, their schemas must be union compatible, *i.e.* contain the same number of attributes with compatible types. The entity complement workflow is similar to the schema complement workflow:

Step 1: Select start table: the user selects a source table to be complemented.

Step 2: Find candidate tables: the user explores the schema matchings and selects the candidate tables that are related to the start table and contain new

tuples which are related to the tuples in the start table. If necessary, the candidate tables are made union compatible with the start table (essentially by removing irrelevant attributes through projection).

Step 3: Generate complemented table: schema mappings are merge queries that take the union of the start table and the target tables.

Example 5.7. Consider the source schema with the following tables:

SALES_SUM (COUNTRY, YEAR, *AMOUNT*)
INTERNET_SALES (COUNTRY, YEAR, *AMOUNT*)
SALESORG (STORE_ID, CITY, STATE, COUNTRY)
TIME (DATE, WEEK, MONTH, YEAR)
REGION (CITY, STATE, COUNTRY, CONTINENT)

Step 1: Select start table: The user selects SALES_SUM as the start table to be complemented.

Step 2: Find candidate table: Figure 5.4 depicts the join relationships between the source tables in solid lines. As we can see, table INTERNET_SALES contains all the attributes of table SALES_SUM. Therefore, table INTERNET_SALES can complete SALES_SUM with internet sales.

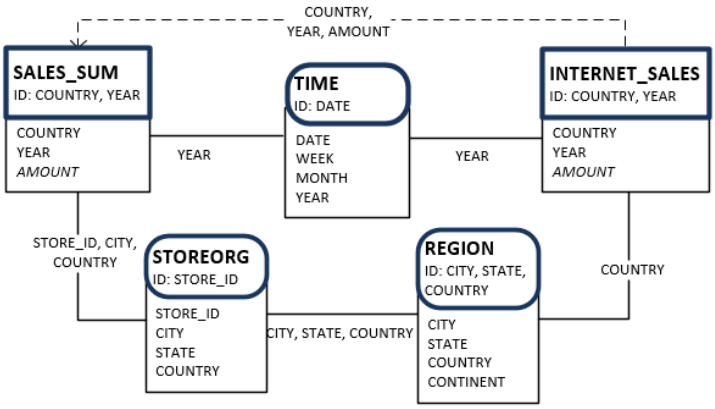


Figure 5.4: Schema matchings between source schemas

We then can extract the following to schema matchings for SALES_SUM and INTERNET_SALES (dashed arrow in Figure 5.4):

Table 5.3: Schema matching specified in Figure 5.4

SALES_SUM.COUNTRY	$\cong$	INTERNET_SALES.COUNTRY
SALES_SUM.YEAR	$\cong$	INTERNET_SALES.YEAR
SALES_SUM.*AMOUNT*	$\cong$	INTERNET_SALES.*AMOUNT*

Step 3: Generate complemented table: The final schema mapping is a simple query that performs a union of SALES_SUM with table INTERNET_SALES.

Listing 5.3: Union of SALES_SUM and INTERNET_SALES

```
SELECT COUNTRY, YEAR, AMOUNT
FROM SALES_SUM
UNION
SELECT COUNTRY, YEAR, AMOUNT
FROM INTERNET_SALES
```

The entity complement process enriches the entities in the start schema without modifying the schema. The previous example shows that when schema matchings and mappings are known, an entity complement can be easily applied by some user without some specific expertise. We can compute the above result using data integration approach by specify the schema of SALES_SUM as the mediated schema. However, INTERNET_SALES might have overlapping values for attribute COUNTRY and YEAR with SALES_SUM. Therefore, the simple union generates conflicting amount values for the same country and year. This is a well known problem which can be solved using *data fusion* [50] instead of simple set union. In this thesis, we to solve the data fusion problem using schema augmentations. The schema mapping generated is a left-outer join query that joins table SALES_SUM with INTERNET_SALES on common attributes COUNTRY and YEAR. The result table contains a new attribute INTERNET_AMOUNT which stores internet sales amount from INTERNET_SALES.

This alternative method can be generalized for all the entity complement applied between fact tables. It replaces entity complement by a schema augmentation that joins on the common dimension attributes and adds the measures as new attributes to the source table.

We do not discuss entity complement in the context of dimension tables, also called *master* tables in data warehouses, because merging their tuples usually entails a governance process that includes processing steps like duplicate elimination and data fusion. This process also requires human expert interaction.

5.6 Drill-across Queries in Multi-dimensional Databases

5.6.1 Drill-across queries using conformed dimensions

Another similar notion which has been studied in multidimensional databases is the *drill-across* operation (or *multipass query*). Drill-across operations are widely supported by many BI products and platforms for OLAP cubes (fact tables). Initially described in [6], drilling across corresponds to the operation which "combines performance measurements from different business processes in a single report". Drill-accross operations join two or more sub-queries sharing the same identifiers (keys). These identifiers correspond to *conformed-dimensions* [6] identifying the entities on which two fact tables can be joined. Kimball et al. [6] detailed different types of conformed dimensions. Dimensions with the same dimension keys, attribute column names, attribute definitions and attribute values are called *identical conformed dimensions* (it is implicitly assumed that parent-child mappings are preserved also). Usually, identical conformed dimensions are built from the same non-analytic table or obtained by duplication. When one dimension only contains a subset of the attributes or rows of some base dimension, the dimension with less attributes or rows, is called *shrunken conformed dimension*. Identical conformed dimensions can be joined directly, while shrunken conformed dimensions are obtained by some aggregation (rollup) or filtering with respect to their base dimensions.

Drill-across queries are generated by the following steps.

Step 1: Select start and target tables: the user selects a source table and some target fact tables.

Step 2: Infer conformed dimensions: the user identifies the schema matchings between the dimensions in the start table and the target tables. The matching dimensions are called *conformed dimensions*.

Step 3: Generate schema mapping: infer the schema mapping which is a merge query that joins the start table and the target tables using the *conformed dimensions*. The join query might be applied to the result of sub-queries that aggregate or filter some tables according to their shrunken conformed dimensions.

The formulation of drill across queries requires the identification of conformed dimensions.

Example 5.8. To illustrate the concept of conformed dimensions, consider the two fact tables SALES and INVENTORY in Figure 2.9 on Page 37.

Step 1: Select source and target tables. The user selects table SALES as the start table and INVENTORY as the target table.

Step 2: Infer conformed dimensions. Table INVENTORY has four dimensions TAX, $PROD$, $WAREHOUSE$ and $TIME$ and table SALES has three dimensions $STORE$, $PROD$ and $TIME$. As shown in Figure 5.5, both tables share dimension attributes PROD_SKU and BRAND from the same dimension table $PROD$ and therefore have the same domain. Therefore, $PROD$ is identical conformed dimension which can be directly used to formulate drill-across queries. Both tables also share dimension attribute YEAR from dimension table $TIME$. However, table INVENTORY also contains attribute MONTH from dimension table $TIME$. Therefore attribute YEAR is a shrunken conformed dimension in table INVENTORY which must be aggregated along attribute MONTH before joining with table SALES. Consider now dimen-

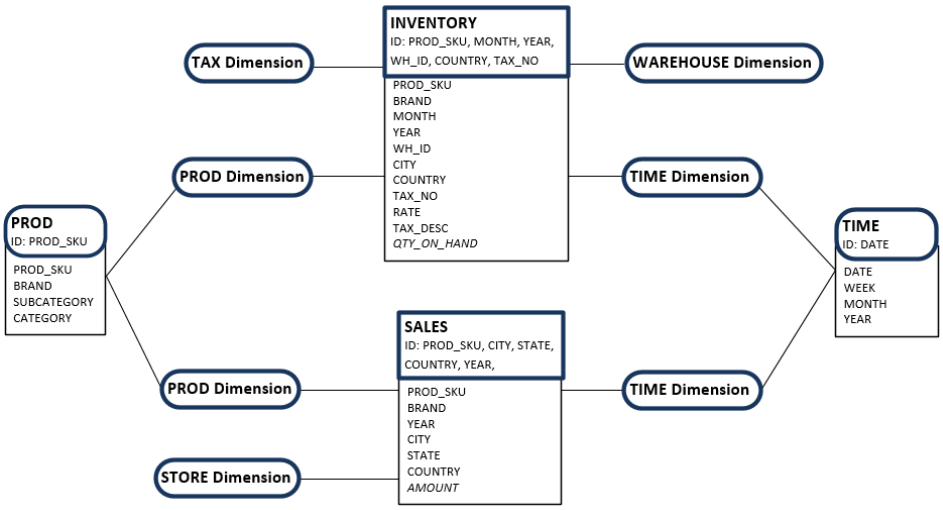


Figure 5.5: Fact tables SALES and SALES_SUM

sion $WAREHOUSE$ in table INVENTORY and dimension $STORE$ in table SALES. Both dimensions share attributes CITY, STATE and COUNTRY describing the same concepts in both dimensions. However, deciding if these two dimensions are conformed dimensions also depends on their domain values. If $WAREHOUSE$ only contains warehouses situated in North America while $STORE$ contains stores in all English speaking countries, the domain values of CITY, STATE and COUNTRY in the two dimensions have a non-empty intersection

but are not identical. Thus, $WAREHOUSE$ and $STORE$ are not considered to be conformed and they cannot be used in the drill-across query.

Step 3: Generate schema mapping. The previous step identifies the two dimensions $TIME$ and $PROD$ as conformed dimensions, but not $WAREHOUSE$ and $STORE$. Therefore, is it not possible to directly apply a merge between the two tables and we first need to reduce the dimensions in tables SALES and INVENTORY.

We can relate the two notions of drill-across queries and schema augmentation. Suppose that we want to augment the schema of SALES with the measure attributes of INVENTORY to compare a product's sales amount with its inventory level. By definition, schema augmentation mainly consists in generating the following outer-join (merge) query between the source table and its augmentation table:

Listing 5.4: Schema augmentation approach to join SALES with INVENTORY

```
SELECT SALES.PROD_SKU, SALES.BRAND, SALES.YEAR,
       SALES.CITY, SALES.STATE, SALES.COUNTRY, SALES.AMOUNT,
       INVENT.QTY_ON_HAND
FROM SALES LEFT OUTER JOIN
    (   SELECT PROD_SKU, BRAND, YEAR, CITY, COUNTRY,
        AVG(QTY_ON_HAND) AS QTY_ON_HAND
        FROM INVENTORY
        GROUP BY PROD_SKU, BRAND, YEAR, CITY, COUNTRY
    ) INVENT
ON SALES.PROD_SKU = INVENT.PROD_SKU
AND SALES.BRAND = INVENT.BRAND
AND SALES.YEAR = INVENT.YEAR
AND SALES.CITY = INVENT.CITY
AND SALES.COUNTRY = INVENT.COUNTRY
```

Suppose that we want to achieve schema augmentation using drill across queries. As defined before, drill-across operations are only possible through conformed dimensions. This has two consequences in our augmentation scenario. First, non-common dimensions must be removed from each fact table (which entails to "rolling-up" the facts of each table). Second, the two fact tables must have the same granularity on their common dimensions $PROD$ and $TIME$, which is not the case since SALES records total sales for every year and INVENTORY records inventory levels for every month. To align the granularity of the two fact tables on $TIME$, the facts of INVENTORY must be rolled up to the YEAR level. Thus, two separate pre-processing queries $Q(\text{SALES})$ and $Q(\text{INVENTORY})$ must be defined and executed on each fact table to enable a drill-across operation. In this

case, Q(SALES) and Q(INVENTORY) are aggregation query that computes the sum of the sales amount and average of inventory quantity on hand respectively grouped by PROD_SKU, BRAND, YEAR (the result is shown in Figure 5.6). Then a new fact table SALES_INVENTORY is created using an outer-join over Q(SALES) and Q(INVENTORY) in Figure 5.6 through dimensions $PROD$ and $TIME$.

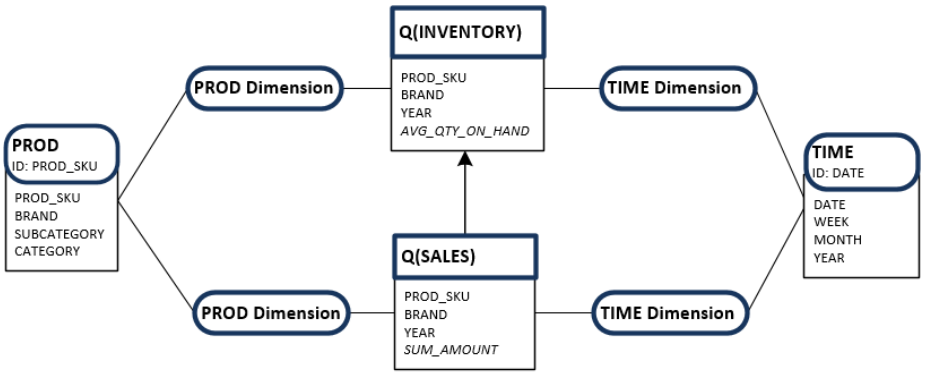


Figure 5.6: The integration of SALES and INVENTORY

Listing 5.5: Drill-across query over SALES and INVENTORY

```
SELECT PROD_SKU, BRAND, YEAR, SUM_AMOUNT, AVG_QTY_ON_HAND
FROM Q(SALES) AS Q_SALES FULL OUTER JOIN Q(INVENTORY) AS Q_INVENTORY
ON Q_SALES.PROD_SKU = Q_INVENTORY.PROD_SKU
    AND Q_SALES.BRAND = Q_INVENTORY.BRAND
    AND Q_SALES.YEAR = Q_INVENTORY.YEAR
```

SALES_INVENTORY (PROD_SKU, BRAND, YEAR, SUM_AMOUNT, AVG_QTY_ON_HAND)

In the drill-accross query scenario, the two queries $Q(SALES)$ and $Q(INVENTORY)$ in the FROM clause have to be defined by the user who wants to extend the schema of SALES with attributes from INVENTORY. In particular, the aggregation function to apply on the measure attributes must be properly selected. Furthermore, in comparison with the approach proposed in this thesis, drill-across queries are more restrictive since they are limited to conformed dimensions.

In the schema augmentation framework, drill-across queries can be simulated as a special case with some restrictions. The first restriction is that the start and candidate tables are necessarily fact tables. Schema matchings in drill-across queries are limited to identical and shrunken conformed dimensions. Thus, other heuristic (similarity- and value-based) and reliable (FK-PK constraints) schema matchings which can be used in schema augmentation but do not lead to conformed dimensions are also excluded in the drill-across query scenario. Both restrictions reduce

many opportunities to combine fact and dimension tables with respect to schema augmentation operations.

5.6.2 Drill-across queries using compatible dimensions

Paper [30] describes a framework for applying drill-across queries in the context of heterogeneous data sources. Drill-across queries are only allowed through *compatible dimensions*, which reduces the limitation imposed by conformed dimensions. Two dimensions are *compatible* if they match "perfectly" by satisfying three properties:

1. *coherency*: the matching preserves the hierarchy levels;
2. *soundness*: the matching preserves the domain values of each level;
3. *consistency*: the matching preserves the child-parent value mappings between hierarchy levels.

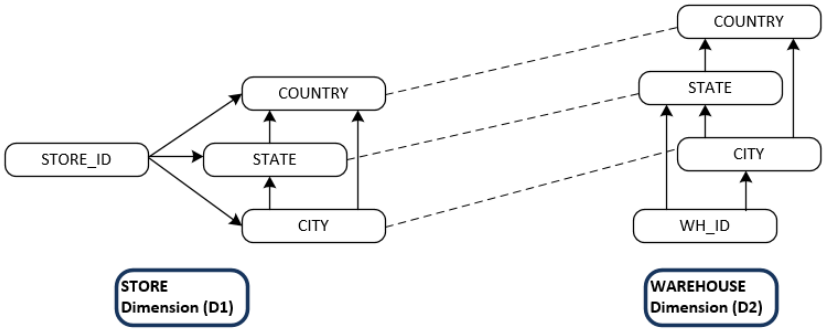
We illustrate the notion of compatible dimensions in the following example.

Example 5.9. Consider the drill-across query scenario in Example 5.8 merging tables SALES and INVENTORY of Figure 2.9 on Page 37. Table SALES_INVENTORY does not contain dimension $STORE$ (it is reduced by aggregation) because this dimension does not *conform* to dimension $WAREHOUSE$. Dimension $STORE$ is not *compatible* with dimension $WAREHOUSE$ since the domain values of the shared attributes are not identical (soundness condition).

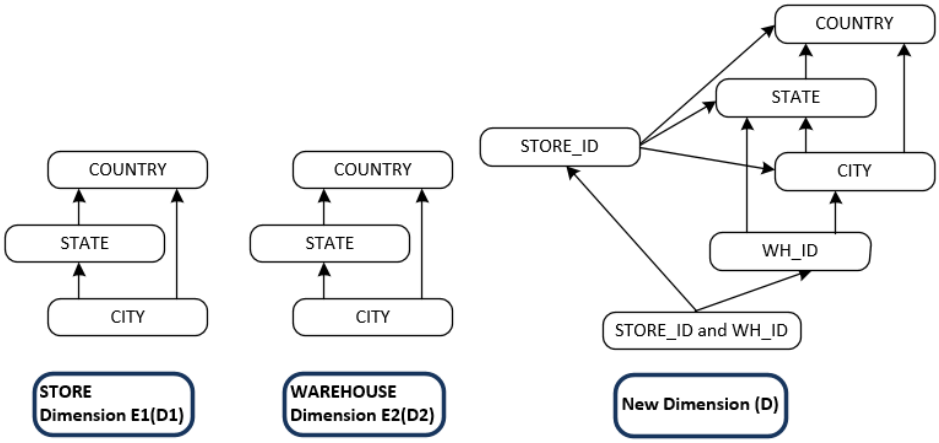
We now illustrate the two ways for building compatible dimensions between $WAREHOUSE$ and $STORE$ presented in [30]. The source table and target table are the same as in Example 5.8, so Step 1 is identical. We then continue with step 2, inferring schema matchings between SALES and INVENTORY. Because $PRODUCT$ and $TIME$ are conformed dimensions, they are also compatible dimensions. The main goal is then to transform dimensions $WAREHOUSE$ and $STORE$ into compatible dimensions. The schema matchings between the two dimensions are shown in Figure 5.7a.

Step 2: Infer conformed dimensions

Loosely coupled approach: The *loosely coupled approach* generates query expressions to propagate common tuples between $WAREHOUSE$ and $STORE$ for creating a perfect match. Observe that the matching is consistent and coherent. We therefore must create a query that makes the matching sound. This can be



(a) Two matched dimensions



(b) Loosely coupled approach **(c)** Tightly coupled approach

Figure 5.7: Two approaches to formulate compatible dimensions

obtained by the following two semi-join query expression applied on $STORE$ (D1) and $WAREHOUSE$ (D2):

$$E1(D1) = \pi_{\text{CITY,STATE,COUNTRY}}(D1 \ltimes_{CITY} D2)$$
$$E2(D2) = \pi_{\text{CITY,STATE,COUNTRY}}(D2 \ltimes_{CITY} D1)$$

Queries $E1(D1)$ and $E2(D2)$ project both tables on the common attributes CITY, STATE, COUNTRY. Because the matching between $WAREHOUSE$ and $STORE$ is consistent, it is sufficient to match both dimension tables on the lowest level attribute CITY within the three matching attributes and we also can conclude that E1($D1$)=E2($D2$). The two result dimensions shown in Figure 5.7b are now compatible.

Tightly coupled approach: The second approach to build compatible dimensions is called *tightly coupled approach* which is obtained by deriving the union of $WAREHOUSE$ and $STORE$ as a new dimension. This can be ob-

tained by applying a outer join which generates a new bottom level attribute STORE_ID_WH_ID, the new bottom level attribute is a concatenation of values in STORE_ID and WH_ID.

Listing 5.6: Query Q_D

```
SELECT CONCAT(STORE_ID, WH_ID) AS STORE_ID_WH_ID,
    STORE_ID, WH_ID,
    CITY, STATE, COUNTRY
FROM STORE JOIN WAREHOUSE
ON STORE.CITY = WAREHOUSE.CITY
AND STORE.STATE = WAREHOUSE.STATE
AND STORE.COUNTRY = WAREHOUSE.COUNTRY
```

The new dimension is shown in Figure 5.7c, and a perfect match of dimensions $WAREHOUSE$ and $STORE$.

Step 3: Generate schema mapping. The schema mapping is built by joining $Q2(\text{SALES})$ and $Q2(\text{INVENTORY})$ using conformed dimensions $PROD$ and $TIME$, and compatible dimensions $E1(D1), E2(D2)$ as shown in Figure 5.8a (or D as shown in Figure 5.8b). $Q2$ aggregates group-by attributes PROD_SKU, BRAND, YEAR, CITY and COUNTRY.

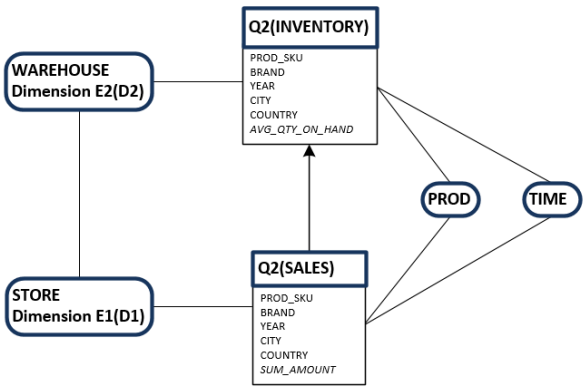
Listing 5.7: Query to generate SALES_INVENTORY_2

```
SELECT PROD_SKU, BRAND, YEAR, CITY, COUNTRY, SUM_AMOUNT, AVG_QTY_ON_HAND
FROM Q2(SALES) JOIN Q2(INVENTORY)
    ON Q2(SALES).PROD_SKU = Q2(INVENTORY).PROD_SKU
   AND Q2(SALES).BRAND = Q2(INVENTORY).BRAND
   AND Q2(SALES).YEAR = Q2(INVENTORY).YEAR
   AND Q2(SALES).CITY = Q2(INVENTORY).CITY
   AND Q2(SALES).COUNTRY = Q2(INVENTORY).COUNTRY
```

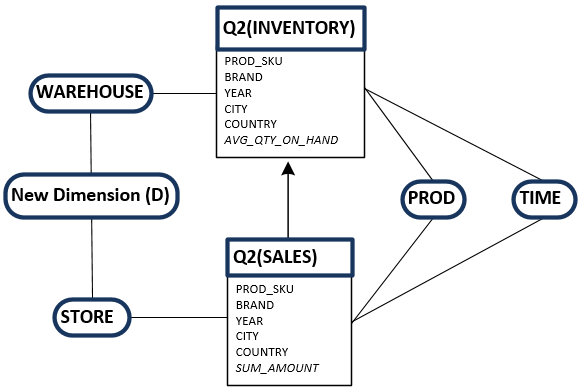
The result table SALES_INVENTORY_2 has the following schema:

SALES_INVENTORY_2 (PROD_SKU, BRAND, YEAR, CITY, COUNTRY, SUM_AMOUNT, AVG_QTY_ON_HAND)

Compatible dimensions using the loosely or tightly coupled approaches relax the restriction for conformed dimensions that domain values and attribute names must be identical. New schema matchings can be found using the two approaches for creating compatible dimensions by generating new hierarchies. In the loosely coupled approach, the dimension hierarchies are reduced to the set of common attributes and all other attributes are removed (for example, attribute WH_ID in $WAREHOUSE$



(a) Integration using dimensions E1(D1) and E2(D2)



(b) Integration using dimension D

Figure 5.8: The integration of SALES and INVENTORY using compatible dimensions

and attribute STORE_ID in $STORE$ are removed). Removing attributes is a significant modification of the existing dimensions that also has a string impact on the related fact tables which must be examined by the data experts. In the tightly coupled approach, all attributes of all dimensions are preserved which might result in a complex hierarchy (for example D). As we will discuss in Section 5.7, complex hierarchy structures also introduce new issues for correctly aggregating data over these structures. In addition, drill-across queries using compatible dimensions are still restricted to fact tables.

5.7 Summarizable Analytic Tables

The notion of summarizability was introduced in the context of analytic tables with two different but related perspectives, which are the statistical and multidimensional (OLAP) data modeling perspectives. A common problem is to define correctness conditions for the computation of an aggregate query over an analytic table. In this

section, we are mainly interested in conditions that are expressed over a database schema or an aggregate query. We review existing approaches, compare each proposition with our work, and emphasize the original contributions of our work. To facilitate comparisons, we use the notations for analytic tables introduced in this thesis to describe the propositions made in previous works.

5.7.1 Summarizability in statistical data models

The notion of *summarizability* was initially defined by Rafanelli and Shoshani [52] for statistical databases and later refined in [23]. In this context, *base data*, also referred to as "micro-data", describe all the details about the objects or individuals over which a summarization operation can be applied to produce a so-called *statistical object*, also referred to as "macro-data". Using the terminology defined in this thesis, base data can be modelled as either a non-analytic table or a fact table that has the finest possible granularity, summarization operations are aggregate queries with a semantics that ignores group-by attributes having null values, and a statistical object can be modeled as a fact table that results from a summarization operation over the base data. However, specific assumptions are made on a fact table. Firstly, all facts in a fact table have the same granularity, which means that all measure attributes depend on all the dimensions of a fact table, and not a subset of those dimensions. Secondly, all dimensions in a fact table are independent (that is, no dimension functionally depends on other dimensions). Thirdly, a measure attribute A is associated with the list of summary functions (i.e., aggregate functions) that can be applied to A. Dimensions, in the statistical data model of [23], are restricted to strict hierarchy types, that is, each attribute of a dimension has at most one parent attribute in the hierarchy type. Hierarchy types have a single bottom and top level attribute.

Summarizability is defined as the property of a statistical data model which “guarantees *correct* results of summary operations over statistical objects”. More precisely, suppose we have a statistical object modeled as a fact table $T(S)$ and a summarization query[1] Q over T, $Q = \mathcal{A}gg^*_T(F(\mathrm{A})|\mathrm{X})$, where A is a summary attribute in S, F is a function "applicable" to A (this notion will be explained below), X is a set of dimension attributes in S, and $\mathcal{A}gg^*$ denotes a summarization operation that does not consider tuples with null values in X attributes. We shall say that attribute A is *summarized along* the dimension attributes that are not in X. We also assume that T is itself built using a summarization query from a base table T_0.

[1] We use the expression "summarization query" to distinguish it from aggregation query, which uses the SQL semantics for null values.

Then, the summarizability property consists of defining conditions under which the summarization query $\mathcal{Q}$ is *correct*, which means that there exists a function G such that: $\mathcal{Agg}^*_{T_0}(F(\mathrm{A})|\mathrm{X}) = \mathcal{Agg}^*_{T}(G(\mathrm{A})|\mathrm{X})$. Thus, summarizability is formulated at the level of a given summarization query.

Three necessary conditions are proposed by Lenz and Shoshani in [23] to check whether a summarization query $\mathcal{Q} = \mathcal{Agg}^*_{T}(F(\mathrm{A})|\mathrm{X})$, as defined above, is correct. Let S_D be the set of dimension attributes for dimension D in S, and $S_D^\top$ be the bottom level attribute of S_D. The conditions defined in [23] are:

1. *Disjointness*. For each dimension D along which summarization is done in Q, the child-parent value mappings between the dimension attributes of D in T_0 should be many to one. This could be expressed by requiring that the corresponding edges in the attribute graph of dimension D are labelled as $\mathbf{f}$.

2. *Completeness with respect to* $F(\mathrm{A})$. For each dimension D of T the domain of values of each attribute in S_D in T, must be *complete*, that is: (1) the attribute values of $S_D^\top$ contains all possible values with respect to T_0, as required by $F(\mathrm{A})$, and (2) every value of a dimension attribute that is a child level of $S_D^\top$ must map to a parent value in $S_D^\top$ within T_0.

3. *Applicable summary function*. Summary function $F(\mathrm{A})$ should be "applicable" with respect to all dimensions along which summarization is done in $\mathcal{Q}$.

We comment each condition. The first condition imposes that each hierarchy along which summarization is done is strict, which actually implies that the lowest level attribute of each dimension is the identifier of the dimension. In the original formulation of [23], the disjointness condition is expressed over the attributes of D in T_0 that are sub-levels of the attributes of $S_D - X$ but the authors also impose, by definition of a statistical object, that the hierarchy of attributes in S_D is strict. We then grouped all conditions in our formulation of the first condition. In our work, we address the problems raised by non-strict hierarchies by propagating the aggregable properties of an aggregable attribute after an aggregation is done (see Proposition 3.6). Note that a related problem, is the detection of ambiguous analytic tables (see Definition 3.13) and ambiguous queries. We however do not reject ambiguous queries as [23] do by imposing strict hierarchies in T, but instead nullify summary attribute values for those tuples that are ambiguous in the result of an aggregate query. This will be illustrated later.

The second condition on completeness has multiple facets. One is the interpretation of expression "all possible values" in item (1) of the condition. Take for instance table $\mathrm{STORE\_SALES}$ shown in Table 5.7a, how do we know that all possible values

are listed in the domain of attribute STORE_ID? One interpretation is that we have an external knowledge of whether the list of all values listed in the base table T_0 from which the statistical object T is built is complete. The other interpretation is that the only possible values are exclusively those listed in the base table T_0. We used this later interpretation. In our work, we assume the existence of a dimension table, like $SALESORG$, that contains all stores. We therefore adopt a *Closed World Assumption* [16] on the dimension table: the only dimension values that exist are those listed in the dimension table.

Another facet of the completeness condition is that item (1) is asserted with respect to a summarized attribute $F(\text{A})$. This means that the user who is formulating query Q must decide whether completeness is needed or whether the values listed in T are sufficient to compute a summary attribute. In our work, we follow a similar approach by providing the option to the user to decide if completeness must be enforced during a merge (see algorithms in Section 4.5). However, our completeness requirement is weaker as we shall see in Example 5.11 below.

Finally, the third facet of the completeness condition is captured in the condition of item (2). It implies that the bottom level dimension attributes in T are *mandatory*. We do not have such a restriction in our work due to our interpretation of nulls and our use of (SQL) aggregate operations that handle null values as regular values.

The third condition in the definition focuses on testing the compatibility between the type of dimensions and the type of measures used in T. The following types of measures are distinguished: *stock* (i.e., a simple value at a particular point in time), *flow* (i.e., cumulative values over a period) and *value-per-unit* (i.e, determined value for a fixed time). Dimensions can be of type temporal or non-temporal. When a measure attribute A is aggregated using a given function over some dimension D, the types of A and D should be compatible with respect to that function. For instance, let us consider the two common aggregate functions: SUM and AVG. Measure attribute ANNUAL_INCOME is a measure of type *flow* so it can be summed and averaged over both temporal and non-temporal dimensions. Attribute BALANCE of credit cards is a measure of type *stock*, so it cannot be summed up over temporal dimensions like $TIME$, but it can be averaged over non temporal dimensions. Finally, attribute ITEM_PRICE is of type *value-per-unit*, so it cannot be summed over any dimension but it can be averaged. In our work, the third condition of [23] is captured by our more general notion of "aggregable property" but we leave open the method by which aggregable properties are obtained.

We now illustrate the possible correctness issues that arise when a summarization query is expressed over a fact table and draw a comparison with our work.

Table 5.4: Table PRODUCT_LIST

PROD_SKU	BRAND	COUNTRY	YEAR	QTY
cz-tshirt-s	COCA COLA	United States	2017	5 000
cz-tshirt-s	COCA COLA	United States	2018	7 000
cz-tshirt-s	ZARA	Spain	2017	5 000
cz-tshirt-s	ZARA	Spain	2018	7 000
coca-can-33cl	COCA COLA	United States	2017	10 000

Example 5.10. Consider the base table PRODUCT_LIST (PROD_SKU, COUNTRY, BRAND, YEAR, QTY) defined over dimension MKT_PROD and $TIME$ whose instance is displayed on Table 5.4. We assume that all products are listed in PRODUCT_LIST. Suppose that we define a statistical object PRODUCT_SUM (BRAND, YEAR, TOTAL) built using a summarization query: $\mathcal{A}gg^{*}_{PRODUCT\_LIST}(\text{COUNT\_DISTINCT}(\text{PROD\_SKU})|\text{BRAND}, \text{YEAR})$, whose result is displayed on Table 5.5.

Table 5.5: Table PRODUCT_SUM

BRAND	YEAR	TOTAL
COCA COLA	2017	2
COCA COLA	2018	1
ZARA	2017	1
ZARA	2018	1

Now, consider a summarization query:

$$Q_1 = \mathcal{A}gg^{*}_{PRODUCT\_SUM}(\text{SUM}(\text{TOTAL})|\text{BRAND})$$

that aggregates TOTAL along dimension YEAR. The disjointness condition of [23] is obviously satisfied by dimension $TIME$. However, aggregable attribute TOTAL is of type *stock*, which means that, by the third condition of [23], it cannot be summed up along a temporal dimension like $TIME$. Indeed, the number of distinct products by brand reported by query Q_1 (e.g., value 2 for brand ZARA) would be different from the number directly computed from PRODUCT_LIST (e.g., 1 for brand ZARA). Hence, query Q_1 is considered to be incorrect.

Now, consider another query:

$$Q_2 = \mathcal{A}gg^{*}_{PRODUCT\_SUM}(\text{SUM}(\text{TOTAL})|\text{YEAR})$$

Here, the disjointness condition is not satisfied since for dimension MKT_PROD the product "cz-tshirt-s" maps to different BRAND values. Again, the number of distinct products by year reported by query Q_2 (e.g., value 2 for year 2018) would be different from the number directly computed from PRODUCT_LIST (e.g., 1 for year 2018). Hence, query Q_2 is also considered to be incorrect. Thus, no further summarization of PRODUCT_SUM is possible.

By comparison with our work, the same result is obtained by observing that the answer to the following question is negative: "In PRODUCT_LIST, is PROD_SKU summarizable with respect to COUNT_DISTINCT and $X = \{\text{BRAND}, \text{YEAR}\}$"? Per Proposition 3.5, the answer would be positive if: (1) $\mathbf{agg}_{\text{PROD\_SKU}}(\text{COUNT\_DISTINCT}, \{\text{PROD\_SKU}, \text{COUNTRY}\})$ holds in PRODUCT_LIST, and (2) the literal functional dependency $\text{S}_D - \text{X} \mapsto \text{X}$, where S_D is the set of dimension attributes, holds in PRODUCT_LIST. Condition (1) is verified since PROD_SKU is aggregable using COUNT_DISTINCT along any dimension attribute. However, $\text{S}_D - \text{X}$, which is $\{\text{PROD\_SKU}, \text{COUNTRY}\}$, does not determine X, and therefore condition (2) is not satisfied. Hence, we conclude that PRODUCT_SUM enables *incorrect* summarization operations (by comparison to PRODUCT_LIST).

Table 5.6: Table PRODUCT_LIST

PROD_SKU	BRAND	COUNTRY	YEAR	QTY
cz-tshirt-s-17	COCA COLA	United States	2017	5 000
cz-tshirt-s-18	COCA COLA	United States	2018	7 000
cz-tshirt-s-17	ZARA	Spain	2017	5 000
cz-tshirt-s-18	ZARA	Spain	2018	7 000
coca-can-33cl	COCA COLA	United States	2017	10 000

We can also detect which aggregation query is correct over PRODUCT_SUM by computing the aggregable properties on attribute TOTAL using propagation rules (see Proposition 3.6). For instance, take query Q_1. As explained before, $\mathbf{agg}_{\text{PROD\_SKU}}(\text{COUNT\_DISTINCT}, X')$ holds in PRODUCT_LIST, where X' contains all dimension attributes of PRODUCT_LIST. So if S is the schema of PRODUCT_LIST, $X' \cap \text{S} = \{\text{BRAND}, \text{YEAR}\}$. The dimension attributes of PRODUCT_SUM that are not determined by the dimension attributes of PRODUCT_LIST not in PRODUCT_SUM is: $K = \{\text{BRAND}, \text{YEAR}\}$, so $X = X' \cap \text{S} - K = \emptyset$, and $\mathbf{agg}_{\text{TOTAL}}(\text{COUNT\_DISTINCT}, \emptyset)$ holds in PRODUCT_SUM. However, suppose that table PRODUCT_LIST is such that a given product gets a different PROD_SKU every year (so the table could look like Table 5.6), then the LFD PROD_SKU $\mapsto$ YEAR would hold in PRODUCT_LIST and we would have: $K =$

$\{$BRAND$\}$, hence $\mathbf{agg}_{\text{TOTAL}}(\text{COUNT\_DISTINCT}, \text{YEAR})$ holds in PRODUCT_SUM and query Q_1 is now correct. We currently do not check LFD across dimensions and require that such an aggregable property is explicitly provided by the user to enable query Q_1 (the default empty set is currently automatically generated).

Example 5.11. Consider another statistical object modeled by a fact table STORE_SALES shown in Table 5.7a where STORE_ID, CITY, STATE and COUNTRY are attributes of dimension $SALESORG$, attribute YEAR is from dimension $TIME$ and attribute AMOUNT represents the sales amount of each store. This table is computed using a summarization query from table SALES (PROD_SKU, BRAND, MONTH, YEAR, STORE_ID, CITY, STATE, COUNTRY, AMOUNT). We assume that AMOUNT is a measure of type *flow* which can be summed over any dimension. We also suppose also that all possible values for the dimension are listed in a table $SALESORG$ with STORE_ID as a dimension identifier.

Table 5.7: Fact and dimension table

(a) STORE_SALES

STORE_ID	CITY	STATE	COUNTRY	YEAR	AMOUNT
Oh_01	Dublin	Ohio	USA	2017	3.2
Ca_01	Dublin	California	USA	2017	5.3
Oh_01	Dublin	Ohio	USA	2018	8.2
Ca_01	Dublin	California	USA	2018	6.3
Pa_01	Paris	-	France	2017	45.1

(b) $SALESORG$

STORE_ID	CITY	STATE	COUNTRY
Oh_01	Dublin	Ohio	USA
Ca_01	Dublin	California	USA
Ca_02	Palo Alto	California	USA
Pa_01	Paris	-	France

Consider a summarization query:

$$Q_1 = \mathcal{A}gg^*_{STORE\_SALES}(\text{SUM}(AMOUNT)|\text{CITY})$$

that adds all amounts by city (so, aggregate along dimension attributes of $SALESORG$ and $TIME$). The result is displayed in Table 5.8a. The disjointness condition of [23] is not satisfied since for dimension $SALESORG$ city "Dublin" maps to different STATE values.

By comparison with our work, we would propagate the aggregable property of AMOUNT from SALES to STORE_SALES using rule Proposition 3.6 case 1, which

yields $\mathbf{agg}_{\text{AMOUNT}}(\text{SUM}, \{\text{AMOUNT}, X\})$, where X is the set of dimension attributes in STORE_SALES since function SUMis distributive using SUM. Hence, we consider AMOUNT to be summarizable with respect to SUMand X. However, we would say that the summary value for the tuple with city "Dublin" is *ambiguous* (because it would add up the amounts of stores "Oh_01" and "Ca_01"). Thus, we would return a null value for $\text{SUM}(AMOUNT)$ while the tuple for "Paris" that is not ambiguous will be returned with aggregated value 45.1 (See Table 5.9a).

Table 5.8: Results of summarization queries

(a) $\mathcal{Q}_1$(STORE_SALES)

CITY	SUM(AMOUNT)
Dublin	23
Paris	45.1

(b) $\mathcal{Q}_2$(STORE_SALES)

STATE	SUM(AMOUNT)
Ohio	11.4
California	11.6

To illustrate the completeness issue, consider now the following query:

$$\mathcal{Q}_2 = \mathcal{A}gg^*_{STORE\_SALES}(\text{SUM}(AMOUNT)|\text{STATE})$$

that adds all amounts by state (see Table 5.8b. Assuming that SUM(AMOUNT) requires a complete answer, the second condition of [23] is not satisfied by item (1) since attribute STORE_ID does not contain all stores for STATE 'California'. Indeed, $\mathcal{Q}_2$ is considered to return an incorrect SUM(AMOUNT) value for 'California' because STORE_SALES "misses" the sales amount of store Ca_02. However, if STORE_SALES is supposed to contain all sales for the current month (i.e., Ca_02 did not sell anything) we could consider that item (1) of the completeness condition is not violated for SUM(AMOUNT) in query $\mathcal{Q}_2$.

To compare with our work, if STORE_SALES was only containing sales for state 'Ohio', we would consider query $\mathcal{Q}_2$ to be complete because no city is missing from state 'Ohio' in $SALESORG$. The completeness condition of [23] would still not be satisfied.

Table 5.9: Results of aggregation queries

(a) $\mathcal{Q}_1$(STORE_SALES)

CITY	SUM(AMOUNT)
Dublin	-
Paris	45.1

(b) $\mathcal{Q}_2$(STORE_SALES)

STATE	SUM(AMOUNT)
Ohio	11.4
California	11.6
-	45.1

Item (2) of the second condition is also not satisfied because city 'Paris' does not map to a parent value in attribute STATE. Indeed, if we add all amounts in the result of Q_2 we get a value of 23 which is obviously not the correct total amount. In our context, the result of Q_2 would contain a tuple with an amount value of 45.1 for value of state null (See Table 5.9b). Hence, completeness would not be compromised.

5.7.2 Summarizability in multidimensional data models

In a multidimensional database context, summarizability refers to the property that "the correct computation of higher-level aggregates over a fact table can be obtained from previously available lower-level aggregates over the same table". There, the definition of summarizability is equivalent to the definition we introduced in Definition 3.10 on Page 56. Given a summarization query $\mathcal{Q} = \mathcal{A}gg^*_T(F(\mathrm{A})|\mathrm{X})$ over a fact table T, we want to know if A is *summarizable* with respect to F and X, that is, higher-level aggregates $F(\mathrm{A})$ can be computed correctly from the result of $\mathcal{Q}$ using any query $\mathcal{Q}' = \mathcal{A}gg^*_{\mathcal{Q}}(F(\mathrm{A})|\mathrm{X}')$, where $X' \subset X$ and A is the name of the attribute computed by $F(\mathrm{A})$ in $\mathcal{Q}$. While [23] is interested to characterize which query $\mathcal{Q}'$ is correct, the summarizability conditions for an attribute with respect to a function and a set of dimension attributes is interested to capture properties that will make any query Q' correct. We shall see that it is however possible to characterize summarizability at the level of a query when the general property does not hold.

In this context, we review different solutions that have been proposed to either detect if an aggregation over a fact table is summarizable, or process an aggregation query in such a way that future incorrect aggregations are prevented.

Other solutions propose an alternative modeling of analytic data, or methods to modify the representation of dimension data, to enforce summarizability of aggregation queries. We do not consider these solutions in the following sections and a survey can be found in [26].

Conditions over hierarchies and functions

In [21], [29], Pedersen et al. use a more general data model for analytic data than the model of [23]. As we did before, we describe the data model of [21] using our notations and we ignore very specific concepts that were introduced to deal with time-varying dimensions and precision.

In [21], dimensions can include multiple hierarchies, that is, the hierarchy type associated with a dimension can be non-strict. Fact tables are also generalized as follows. There is no distinction of measure attributes in a fact table. Any attribute is aggregable, or can be a summary attribute using the terminology of [23]. This specificity has however no impact on the summarizability conditions since we are interested in characterizing queries over fact tables in which aggregated attributes are indicated. Thus, we shall ignore that specificity in the rest. Facts in a fact table are defined over all dimensions but some values of dimension attributes may not be characterized and hence have a $null$ value. Dimensions in a fact table can have mutual dependencies, that is, the value of one dimension can depend on the values of other dimensions. Finally, each attribute is associated with the type of aggregate function that can be applied to it. More specifically, [21] distinguishes between three types of aggregation functions:

1. Σ functions, which are applicable to data that can be added together,
2. Φ functions, which are applicable to data that can be used for average calculations, and
3. c functions, which are applicable to data that can only be counted.

Considering the standard aggregation functions, we have $\Sigma = \{\mathrm{SUM}, \mathrm{COUNT}, \mathrm{AVG}, \mathrm{MIN},$
$\mathrm{MAX}\}$, $\Phi = \{\mathrm{COUNT}, \mathrm{AVG}, \mathrm{MIN}, \mathrm{MAX}\}$ and $c = \{\mathrm{COUNT}\}$. The aggregation types are ordered in increasing order as: c, Φ, Σ.

Given a summarization query $\mathcal{Q} = \mathcal{A}gg^*_T(F(\mathrm{A})|\mathrm{X})$ over a fact table T, we want to know if A is *summarizable* with respect to F and X, that is, higher-level aggregates $F(\mathrm{A})$ can be computed correctly from the result of $\mathcal{Q}$ using some query $\mathcal{Q}' = \mathcal{A}gg^*_{\mathcal{Q}}(F(\mathrm{A})|\mathrm{X}')$, where $X' \subset X$ and A is the name of the attribute computed by $F(\mathrm{A})$ in $\mathcal{Q}$. Note that this requires that both $\mathcal{Q}$ and $\mathcal{Q}'$ are correct queries in the sense of [23]. However, the characterization of summarizability is expressed with respect to the summarization operation done in $\mathcal{Q}$.

Given a summarization query $\mathcal{Q} = \mathcal{A}gg^*_T(F(\mathrm{A})|\mathrm{X})$ over a fact table T, the conditions proposed by Pedersen et.al. in [21] to determine if A is *summarizable* with respect to F and X are the following. We denote X_D as the set of dimension attributes for dimension D in X.

1. Function F must be *applicable* to A, that is, F must belong to the aggregation type of A.
2. Function F must be *distributive* over the domain of values in A.

3. For every dimension D, all the paths from the bottom level attribute up to the attributes X_D must be *strict* in D (Definition 2.9 on Page 27).

4. For every dimension D, all the hierarchies up to the attributes X_D must be *onto* (Definition 2.7 on Page 26) and *covering* (Definition 2.8 on Page 27).

We now comment each condition and draw comparisons with the conditions of Lenz and Shoshani [23] and our work. The first condition of [21] assumes, like in [23], that we know the functions that are applicable to an aggregable attribute. However, the notion of aggregation type goes further since it also partially captures the notion of applicable function as defined in [23]. This is partially done because, unlike [23] which considers the temporality of dimensions, an aggregation type is not defined with respect to the type of dimensions along which a summarization can be performed. For the same reason, the notion of aggregation type is weaker than our notion of aggregable property (Definition 2.12) that specifies the maximum number of dimensions along which an aggregation function can be applied.

The second condition uses a definition of distributive function that is more restrictive than our Definition 3.11 because it requires that F is such that for any two sets, V_1 and V_2, $F(V_1, \cup V_2) = F(F(V_1) \cup F(V_2))$. Thus, functions like $\mathrm{COUNT}$are discarded. Otherwise, the second condition is similar to our Proposition 3.4. It also relates to the third condition of [23], as shown previously in [20], which compares the categorization of measures of [23] with the notions of additive and semi-additive measures.

The third condition given by the *strictness* requirement is similar to the first condition of [23]. The same comparison we made with our work before also applies to this condition.

In the fourth condition, the *onto* and *covering* requirements are related to a semantics of aggregation that ignores null values in group-by attributes. These requirements are equivalent to item (2) of the second condition (completeness) of Lenz and Shoshani [23] but does not cover item (1) which is ignored in [21]. Indeed, suppose that store 'Pa_01' is removed from both $\mathrm{STORE\_SALES}$ and $SALESORG$ in Example 5.11. Then, according to the definition of [21], AMOUNT is summarizable with respect to SUM and grouping attribute STATE. Therefore query Q_2 is considered to be correct while it does not meet item (1) of the completeness condition of [23]. In our work, we assume a standard SQL semantics of aggregation which treats a null value as a regular value, so the previous anomalies targeted by the *onto* and *covering* requirements do not exist.

Multidimensional normal forms

Several works proposed multidimensional normal forms for analytic tables that provide guarantees for the correctness of summarization queries [19], [24]. These normal forms can be used to design dimension and fact tables over which correct summarization queries can be easily detected and evaluated.

We first describe the multidimensional data model considered by these previous works using our terminology and notations. Dimensions are instances of hierarchy types that satisfy some constraints: the levels of a hierarchy type can be either *mandatory* or *optional* and there exist a single bottom level type and an implicit top level type, called *ALL*. In the attribute graph defined over the attribute hierarchy of a dimension, an optional level corresponds to an attribute that can be bypassed and whose value in a dimension table can be *null*. The hierarchy in each dimension is strict (in the sense of NFD), that is, each arc in the attribute graph of the dimension between two attributes A_i and A_j, such that $A_i \preccurlyeq A_j$, is labelled with a $\mathbf{1}$. A dimension is also associated with a (possibly empty) set of *context dependencies*: let A_i and A_j be two dimension attributes of a dimension D such that A_i is optional, $A_j \neq ALL$, and $A_i \preccurlyeq A_j$. If $c \in dom(A_j)$ and $c \neq null$, then (A_i, A_j, c) is a context dependency for D stating that for every tuple t of the dimension table, $t.A_j = c \Leftrightarrow t.A_i \neq null$. Intuitively, the interpretation of a context dependency is that A_j plays the role of a discriminating attribute in the hierarchy and value c is the discriminating value to indicate when the optional attribute A_i has a non-null value. Note that the use of an equivalence ($\Leftrightarrow$), in the above formula, is quite strong since it forces the existence of a single discriminating value.

A fact table is defined over a set of dimensions at the finest level of detail, that is, all dimension attributes of a dimension are included in the schema of the fact table, and each measure in the fact table is determined (in the sense of NFD dependencies) by the set of bottom level dimension attributes in each dimension. We assume for simplicity here that, unlike [24], measures do not depend on each other, which does not limit the generality of the conditions for summarizability discussed later in this section. In addition, *summarizability constraints* in [24] express which measure in a fact table may be aggregated along what dimension hierarchy according to what aggregation function whereas [19] uses the same categorization of measure attributes as [23]. However, unlike the aggregable properties proposed in this thesis, no formal treatment of summarizability constraints is provided in [24].

By comparison with [23], fact tables here model "micro data" because they represent facts at their lowest level of granularity. Note that although fact tables in [21] can

Table 5.10: PROD_SALES

PROD_SKU	SUBCATEGORY	CATEGORY	BRAND	AMOUNT
p_01	Video projector	video	Epson	42
p_02	TV	video	Philips	58
p_05	TV	video	Samsung	90
p_03	Radio	audio	Philips	45
p_04	CD-player	audio	Samsung	5

represent facts at a coarser granularity, their summarizability conditions impose that the bottom level dimension attributes of each dimension determine (in NFD sense) measure attributes. Next, as in [21] and unlike [23], non-strict hierarchy types are allowed as well as optional dimension attributes. In [21], fact tables allowed dimensions that are mutually dependent but, as said before, this has no impact on summarizability. The novelty of [19], [24] in comparison with [21] is the addition of context dependencies to the data model.

In [19], the following multidimensional normal form, called MNF, is proposed for a fact table (we actually combined in our definition below the conditions originally expressed as Dimensional Normal Form and Multidimensional Normal Form in [19]).

Let T be a fact table defined over a set of dimensions with a measure (summary) attribute A. Table T is in Multidimensional Normal Form (MNF) if the following conditions are satisfied:

1. For each dimension of T, (1) dimension attributes are all mandatory attributes, and (2) the values of the bottom level dimension attribute in the dimension are complete.

2. All dimensions are mutually independent, i.e., there is no NFD between any two dimension attributes of two distinct dimensions.

3. The set of (unique) bottom level attributes of all dimensions functionally determines (FD) attribute A.

Example 5.12. Fact table STORE_SALES of Table 5.7a on Page 145 is not in MNF since condition 1 is violated (e.g., STATE is optional). The fact table PROD_SALES displayed in Table 5.10 (using dimension $PROD$(PROD_SKU, SUBCATEGORY, CATEGORY, BRAND) whose attribute graph is depicted in Figure 2.6 on Page 25) is in MNF if we assume that all products are listed in the table.

Let T be a fact table in MNF, and $\mathcal{Q} = \mathcal{A}gg^*_T(F(\mathrm{A})|\mathrm{X})$ be a summarization query over T. Then, using the same definition of summarizability as before (on Page 148), A is *summarizable* with respect to F and X if F is applicable to A with respect to any subset of X in the result of $\mathcal{Q}$. Here, the notion of applicability of F to A can be expressed using either the categorization of attributes [19] or the summarizability constraints on A [24], the later being more ambiguous as we stated earlier.

We now comment each condition in the definition of MNF. In the first condition, item (1) is equivalent to the conditions on hierarchies expressed by [21]. Indeed, if all dimension attributes are mandatory they cannot have null values and all NFD become FD between attributes, which implies that the hierarchy is strict. Since there are no null values, all hierarchies are also covering. The third condition of the definition implies that all the hierarchies are also onto since bottom level attributes cannot have null values by definition of FD. Item (2) of the first condition is analogous to the completeness condition of [23]. Here again, the means to test this requirement are left unspecified and seem to require some external knowledge. Note that in [24], this point on completeness is simply dismissed. Finally, the second condition is not really needed for summarizability; it has been added to assure that dimensions do not share dimension attributes. Note that in [24], this condition is expressed by requiring that for every measure A of T, the bottom level dimension attributes in the schema of T form a primary key in T and there is no other primary key for A.

Since the conditions provided by MNF for summarizability are similar to the previous work of [23] and [21], the same remarks apply for the comparison with our work.

The previous definition of MNF is quite restrictive since the data model allows fact tables to have optional dimension attributes which are forbidden by MNF. The next definition extends MNF to a so-called *Generalized Multidimensional Normal Form (GMNF)* as follows. We describe here the slightly more general definition of [24].

Let T be a fact table defined over a set of dimensions with a measure (summary) attribute A, T is in Generalized Multidimensional Normal Form (GMNF) if the following conditions are satisfied:

1. For each dimension D of T: (1) for every optional dimension attribute A_i of D, there exists a context dependency $(\mathrm{A}_i, \mathrm{A}_j, c)$ in D, (2) the values of the bottom level dimension attribute in D are complete.

2. All dimensions are mutually independent, i.e., there exists no NFD between any two dimension attributes of two distinct dimensions.

Table 5.11: PROD_NEW_SALES

PROD_SKU	SUBCATEGORY	CATEGORY	VIDEO_RES	BRAND	AMOUNT
p_01	Video projector	video	1920x1080	Epson	42
p_02	TV	video	3840x2160	Philips	58
p_05	TV	video	3840x2160	Samsung	90
p_03	Radio	audio	-	Philips	45
p_04	CD-player	audio	-	Samsung	5

3. The set of (unique) bottom level attributes of all dimensions functionally determines (FD) attribute A.

We comment these conditions. Item (2) of the first condition, as well as the second and third conditions are inherited from MNF. The true difference is brought by item (1) of the first condition. It constrains the semantics of every optional dimension attribute A_i so that there exists at an upper level an attribute A_j that plays the role of discriminator for A_i. Note that A_j can itself be an optional attribute, in which case there will again be a context dependency (A_j, A_k, c') in D. Eventually, the discriminator attribute will be a mandatory attribute since by definition of context dependency, the upper level attribute cannot be ALL.

Example 5.13. Consider an extended version of the product dimension $PROD\_NEW$, whose attribute graph is depicted in Figure 5.9, in which a new attribute VIDEO_RES is added such that VIDEO_RES $\preccurlyeq$ CATEGORY and PROD_SKU $\preccurlyeq$ VIDEO_RES. Consider the fact table PROD_NEW_SALES over $PROD\_NEW$ whose instance is displayed in Table 5.11.

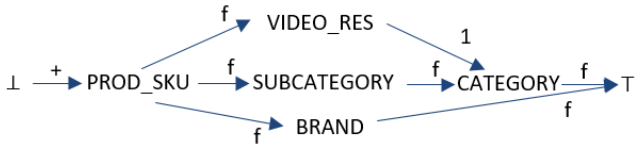


Figure 5.9: Attribute graph of $PROD\_NEW$

Now, assume that we have a context dependency associated with $PROD\_NEW$: (VIDEO_RES, CATEGORY, 'video'). Then, table PROD_NEW_SALES is in GMNF if we assume that all products are listed in the table. It is easy to see that conditions 2 and 3 are satisfied on the attribute graph. Condition 1 is also satisfied because the only optional attribute VIDEO_RES has an upper level discriminating attribute with value 'video'.

Suppose that we add another optional attribute LED_TECH to $PROD\_NEW$ such that: LED_TECH $\preccurlyeq$ VIDEO_RES, LED_TECH $\preccurlyeq$ SUBCATEGORY and PROD_SKU $\preccurlyeq$

Table 5.12: PROD_NEW_SALES2

PROD_SKU	SUBCATEGORY	CATEGORY	VIDEO_RES	LED_TECH	BRAND	AMOUNT
p_01	Video projector	Video	1920x1080	-	Epson	42
p_02	TV	Video	3840x2160	LED UHD 4K	Philips	58
p_05	TV	Video	3840x2160	OLED UHD 4K	Samsung	90
p_03	Radio	Audio	-	-	Philips	45
p_04	CD-player	Audio	-	-	Samsung	5

LED_TECH. This leads to a new dimension $PROD\_NEW2$ whose attribute graph is depicted in Figure 5.10. If we add a context dependency associated with $PROD\_NEW2$: (LED_TECH, SUBCATEGORY, 'TV'), then table PROD_NEW_SALES2, defined over $PROD\_NEW2$, with an instance displayed in Table 5.12, is still in GMNF.

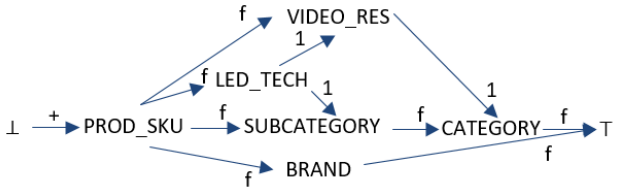


Figure 5.10: Attribute graph of $PROD\_NEW$

As a final example, fact table STORE_SALES of Table 5.7a on Page 145 is not in GMNF since condition 1 is violated. The optional STATE has a null value for different countries and it is not possible to create a single context dependency for attribute STATE using either attribute COUNTRY or CONTINENT.

Similarly to MNF, we can now define summarizability for fact tables in GMNF as follows. Let T be a fact table in GMNF, and $Q = \mathcal{A}gg^*_T(F(\mathrm{A})|\mathrm{X})$ be a summarization query over T. Then, A is *summarizable* with respect to F and X if :

1. F is applicable to A with respect to any subset of X in the result of Q.
2. One of the two conditions hold:
   a) X does not contain any optional dimension attribute, or
   b) if X contains an optional attribute A_i then let $(\mathrm{A}_i, \mathrm{A}_j, c)$ be the associated context dependency, a filter condition: $\mathrm{A}_j = c$ must be applied on T before the summarization query is applied

In other words, summarizability holds provided that a subset of the fact table is considered, and this subset is given by the context dependencies of the dimensions over which the fact table is defined.

Example 5.14. Consider a summarization query Q_1 that adds AMOUNT in table PROD_NEW_SALES2 grouped by CATEGORY and BRAND. Then AMOUNT is summarizable with respect to SUM and X $= \{$CATEGORY, BRAND$\}$ because SUM is still applicable to the resulting AMOUNT attribute and X only contains mandatory attributes. Consider another summarization query Q_2 that adds AMOUNT in table PROD_NEW_SALES2 grouped by VIDEO_RES and BRAND. Then, provided that PROD_NEW_SALES2 is first filtered with a filter: CATEGORY = 'Video' before applying Q_2, then AMOUNT is summarizable with respect to SUM and $X' = \{$VIDEO_RES, BRAND$\}$.

We now compare GMNF with our work in the same context. Going back to Example 5.14, attribute AMOUNT is determined (in LFD sense) by the minimal subset of dimension attributes {PROD_SKU}, which in turns determines all other dimension attributes of PROD_NEW_SALES2. Since SUM is applicable to AMOUNT, by Definition 2.12 on Page 32, $\mathbf{agg}_{AMOUNT}(\text{SUM}, Z)$ holds in PROD_NEW_SALES2, where Z is the set of all dimension attributes of PROD_NEW_SALES2. Consider first query Q_1 of Example 5.14. By Proposition 3.4 on Page 56, attribute AMOUNT is summarizable with respect to SUM and $X = \{$CATEGORY, BRAND$\}$ since $X \subset Z$ and SUM is distributive. Consider now query Q_2, since $X' = \{$VIDEO_RES, BRAND$\}$ is a subset of Z, attribute AMOUNT is also summarizable with respect to SUM and X', without requiring any pre-filtering of PROD_NEW_SALES2. The reason why this happens is our usage of SQL aggregation operations that considers null values as regular values. Indeed, the result of Q_2 is displayed in Table 5.13, and it is easy to see that the summarizability condition is satisfied.

Table 5.13: Query result of Q_2

VIDEO_RES	BRAND	AMOUNT
1920x1080	Epson	42
3840x2160	Philips	58
3840x2160	Samsung	90
-	Philips	45
-	Samsung	5

Reasoning over constraints on dimensions

In [25], the conditions for summarizability use constraints that are expressed over dimensions and generalize the idea of context dependencies introduced in [19], [24].

As before, we first describe the multidimensional data model of [25] using our notations. Dimension hierarchies always have one top level attribute called ALL and possibly multiple bottom level attributes. As in [19], [21], [24], a dimension attribute can have multiple parent dimension attributes in the hierarchy (dimensions are called "heterogeneous"), and there can be both, mandatory and optional dimension attributes. Every child-parent attribute mapping should be functional (i.e., every value only maps to one parent value), which means that an NFD dependency holds between any pair of attributes A_i, A_j where $A_i \preccurlyeq A_j$. As in [19], [24], fact tables are defined over dimensions at the finest level of detail, that is, the schema of the fact table includes the bottom level attributes of the dimensions. Measure attributes are determined by all the dimensions and can only be aggregated using distributive functions (defined as in our Definition 3.11 on Page 56). Dimensions are also supposed to be mutually independent in a fact table. These later requirements are more restrictive than in our data model since we allow measures that only depend on a subset of dimensions.

In [25], Hurtado et.al. define summarizability as a property of dimensions. The rationale is that if we can characterize summarizability for dimensions, then any fact table built over summarizable dimensions will have summarizable measures. Let D be a dimension, X a subset of dimension attributes, and B a dimension attribute in D such that $A_i \preccurlyeq$ B for some attribute A_i of X. The notion of *summarizability* is formalized by the equivalence of two summarization queries. Attribute B is summarizable from X in D if and only if for every fact table T defined over D and distributive aggregate function F using G we have:

$$\mathcal{A}gg^*_T(F(M)|\mathrm{B}) = \mathcal{A}gg^*_{T'}(G(M)|\mathrm{B}) \tag{5.1}$$

where M is a measure attribute in T, and $T' = \mathcal{A}gg^*_T(F(M)|\mathrm{X}' \cup \mathrm{B}), \mathrm{X}' \subseteq \mathrm{X}$.

The summarizability condition can then be expressed independently of the fact tables which refer to these dimensions, as follows. Attribute B is summarizable from X in D if and only if for every bottom level attribute $A_\perp$ of D, we have:

Table 5.14: $PROD\_NEW$

PROD_SKU	SUBCATEGORY	CATEGORY	VIDEO_RES	BRAND
p_01	Video projector	Video	1920x1080	Epson
p_02	TV	Video	3840x2160	Philips
p_05	TV	Video	3840x2160	Samsung
p_03	Radio	Audio	-	Philips
p_04	CD-player	Audio	-	Samsung

$\Pi_{A_\perp,B}(D) = \bigcup_{A_i \in X}(\Pi_{A_\perp,A_i}(D) \bowtie_{A_i} \Pi_{A_i,B}(D))$. Here, Π denotes a duplicate elimination projection and $\bowtie$ denotes a null-eliminating join. We illustrate this condition below.

Example 5.15. Consider the product dimension $PROD\_NEW$ in Table 5.14. Then attribute CATEGORY is summarizable from $X = \{\text{SUBCATEGORY}\}$ because $\Pi_{\text{PROD\_SKU,CATEGORY}}(PROD\_NEW) = \Pi_{\text{PROD\_SKU,SUBCATEGORY}}(PROD\_NEW) \bigcup \Pi_{\text{SUBCATEGORY,CATEGORY}}(PROD\_NEW)$. However, attribute CATEGORY is not summarizable from $X = \{\text{VIDEO\_RES}\}$ because the join between $\Pi_{\text{PROD\_SKU,VIDEO\_RES}}(PROD\_NEW)$ and $\Pi_{\text{VIDEO\_RES,CATEGORY}}(PROD\_NEW)$ eliminates products 'p_03' and 'p_04'.

The above definition of summarizability relates to our definition 3.10 as follows. An attribute B is summarizable from X in D if and only if for any fact table T defined over D such that a measure attribute M depends on the bottom-level attributes of D and F is a distributive function using G over the partitions of A, then M is summarizable with respect to $\text{X} \cup B$ and F for a subset the subset of attributes $\{B\}$. Thus, instead of enforcing equation 5.1 for every subset of $\text{X} \cup B$ as in our definition (where it is denoted Z_2), it is only enforced for $\{B\}$.

Consequently, the summarizability definition of Hurtado et al. could be used, as in [23], to characterize correct summarization queries. Let T be a fact table resulting from a summarization query over a table T_0 and having a measure attribute M. Let $Q = \mathcal{A}gg^*_T(F(M)|\text{X})$ be a summarization query over a T, then Q is *correct* if every attribute B of a dimension D in X is summarizable from $X_D - B$ in D.

To check summarizability, Hurtado et al. propose to specify constraints on dimensions and then transform the summarizability problem into the problem of verifying the satisfaction of a set of dimension constraints by some dimension schema. Let A_i be dimension attributes of a dimension D, the following types of constraints are introduced.

1. $D \models \langle \text{A}_i, \text{A}_{i+1}, ..., \text{A}_j \rangle$ means that for every attribute value v of A_i, there exists a path in D from v to a value of A_j going through a value of A_{i+1}. We shall say that A_i *rolls up to* A_j

2. $D \models \langle \text{A}_i, ..., \text{A}_j = k \rangle$ means that for every attribute value v of A_i, there exists a path in D from v to an attribute value v' of A_j if and only if $v' = k$

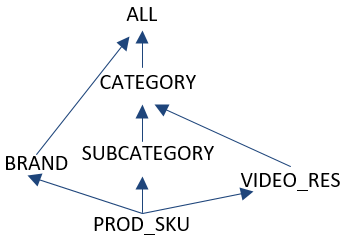
Constraints can then be composed using the usual Boolean logical connectives. Now, assume that a set of constraints have specified on the schema of D. To determine if attribute B is summarizable from X in D, one must determine if, for each bottom level attribute $A_\perp$ of D, the following constraint is satisfied :

$$D \models \langle A_\perp, ..., B \rangle \implies (\langle A_\perp, ..., A_1, ..., B \rangle \oplus ... \oplus (\langle A_\perp, ..., A_n, ..., B \rangle)$$

where $X = \{A_1, ..., A_n\}$ and $\oplus$ denotes an exclusive disjunction.

We use the following examples to illustrate the use of constraints to determine summarizability.

Example 5.16. Consider dimension $PROD\_NEW$ whose hierarchy type is displayed in Figure 5.11. All the child-parent mappings in $PROD\_NEW$ are functional except for attribute VIDEO_RES. Thus, the two following constraints (a) and (b) are expressed on $PROD$. Note that constraint (a) is equivalent to the context dependency of [19], [24].



(a) Hierarchy type of $PROD\_NEW$

(a) $\langle$PROD_SKU, VIDEO_RES, CATEGORY $=' Video' \rangle$

(b) $\langle \text{A}_i, \text{A}_j \rangle$, for all edges (A_i, A_j) in the hierarchy type

(b) Constraints on $PROD\_NEW$

Figure 5.11: The dimension schema of $PROD\_NEW$

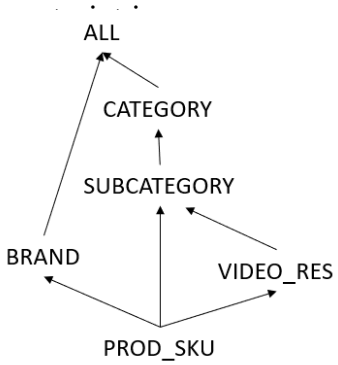
Constraint	Meaning
(a)	A value of PROD_SKU rolls up to VIDEO_RES and CATEGORY only for the 'Video' value of CATEGORY
(b)	All other attributes directly roll up to their parent attribute

To determine if CATEGORY is summarizable from X = {SUBCATEGORY} we must determine if the following constraint can be satisfied:

$$PROD \models \langle \text{PROD\_SKU}, \ldots, \text{CATEGORY} \rangle \Rightarrow \langle \text{PROD\_SKU}, \text{SUBCATEGORY}, \text{CATEGORY} \rangle$$

Using the constraints we have ⟨PROD_SKU, SUBCATEGORY⟩ and ⟨SUBCATEGORY, CATEGORY⟩, which can be composed. Therefore, the condition holds and CATEGORY is summarizable from X.

Example 5.17. we now consider a variation of the previous example in which VIDEO_RES has now SUBCATEGORY for parent in the hierarchy type. A disjunction



l for constraint (a), which cannot be expressed in [24].
ʏ is not summarizable from VIDEO_RES.

(a) ⟨PROD_SKU, VIDEO_RES, SUBCATEGORY = 'TV' ⟩ ⊕
⟨PROD_SKU, VIDEO_RES, SUBCATEGORY = 'Video projector' ⟩

(b) $\langle A_i, A_j \rangle$, for all other edges (A_i, A_j) in the hierarchy type

(a) Hierarchy type of $PROD\_NEW$

(b) Constraints on $PROD\_NEW$

Figure 5.12: The dimension schema of $PROD\_NEW$

It is clear that the data model and constraints proposed by [25] subsume the data model with context dependencies of [19], [24]. We can then generalize the summarizability property for queries as we did before for tables which are in GMNF.

To conclude the comparison with our work, it must be noted that the method Hurtado et al. has several limitations. First, it does not accept non-strict hierarchies: dimension attributes can be optional but when they have a non-null value, their value maps to a single parent value in the hierarchy. Second, it does not accept measures that do not depend on all dimensions. Third, the method does not take into account the notion of completeness captured by item (1) in the completeness condition of [23]. We already explained how this notion is captured in our work. Last, the expression of comprehensive dimension constraints can be labor-intensive.

5.7.3 Conclusion on summarizability

The above works on summarizability were mostly focused on the conditions that the value mappings between attributes should be one-to-one and the domain values of each attribute should be complete [23]. Other works also concentrate on type compatibility between measures and dimensions. [6] captured the ability of summing up measures and classified measures as *additive*, *semi-additive* and *non-additive*. Aggregations using function SUM can be applied on additive measures safely, and cannot be applied on non-additive measures. Semi-additive measures cannot be aggregated along all dimension attributes. For example, in fact table SALES_SUM, measure SALES_SUM is additive whereas attribute QTY_ON_HAND in table INVENTORY is semi-additive because it can not be summed up over temporal attributes like YEAR. [27] provides a detailed case analysis for possible factors impacting additivity.

[20] combined the classifications of [23] and [27] and introduced a new classification of measures defining five measure types : *tally*, *semi-tally*, *reckoning*, *snapshot* and *conversion factor*. For example, attribute SUM_SALES in SALES_SUM is *tally* since it it is summarizable over any dimension and attribute QTY_ON_HAND in INVENTORY is *reckoning* measure since it is only summarizable over non-temporal dimensions. Attribute STOCK_PRICE is a *snapshot* measure and TEMPERATURE is *conversion factor* measure (both are not summarizable).

[32] separated summarization problems into schema level problems (e.g., non-strict hierarchy), data level problems (e.g., imprecision on measure values) and computation level problems (e.g., type compatibility[23]). [26] makes a survey on summarizability issues and classifies summarizability solutions into two axes: 1) solutions which give guidelines for designing complex multidimensional structures and 2) solutions for summarizability problems over existing complex multidimensional structures.

We introduce our solutions in Sections 3.4.1 to 3.4.3 and propose a new way to enforce summarizability even when the hierarchy is non-strict, non-covering and non-onto. The summarizability conditions proposed in [29] require that hierarchies should be strict, covering and onto to avoid double counting during aggregation. In our model, this corresponds to the problem of ambiguity where the aggregation groups cannot identify a unique path in the hierarchy. Aggregations that are ambiguous can be detected and ambiguous values can then be annotated (see Section 3.4.2). Whether the aggregate functions are distributive is expressed using aggregable properties providing additional information on the aggregable dimensions (see Section 3.4.1). Furthermore, we consider the problem of incomplete

merge and propose a solution to complete the domain values by adding missing tuples (see Section 3.4.3). This issue has not been discussed in previous works on summarizability.

5.8 Summary

Table 5.15 gives a global summary for the four approaches mentioned above and our approach. For each approach, Table 5.15 lists its input metadata, the required user actions, and the final result. Optional input items and user actions are in italic. = Besides, the possibilities of automate the process are also discussed, there are three steps considered: discovering schema matchings between schemas, inferring schema mappings using known schema matchings, and generating the final query to create the desired schema.

Schema integration: Schema integration approach takes a set of source schemas as input and generates a unified, global schema that represent all the source schemas. The integration is mainly a manual process which heavily relies on the human understanding of the source schemas to identify and solve semantic and structural conflicts. The solutions to solve conflicts are various and human decisions are almost needed in every schema integration step. For example, to solve schema conflict of two schemas containing different identifiers, the identifier of the global schema could either be the union or the intersection of the two identifiers. The generation of such a global schema therefore needs actions of users with a deep knowledge of the respective source schemas.

Data integration: Data integration approach reduces human effort by automating the detection of schema matchings and the definition of schema mapping queries. The approach takes a set of source schema as input, user is then required to specify the schema of the mediated table. Schema matchings between source schema and the mediated table can be either given as input or detected automatically. Within a set of schema matchings, the system can suggest schema mappings which can be selected and adapted by the user.

Schema complement: Schema complement approach completely automates the schema matching, schema mapping and query generation steps by exploiting semantic metadata like functional dependencies. The approach reduces user interaction to

the selection of a source table and a desired target table with new attributes. The outcome is to create a merge query that joins the source and target tables.

Drill-across queries: Drill-across approach also automates the schema matching, schema mapping and query generation steps by analyzing source schemas and building schema matchings through conformed dimensions. Users must express a drill-across query that takes at least two fact tables as input and returns a new target table which is the merge of the input fact tables.

Schema Augmentation: Schema augmentation approach is an extension of the schema complement approach. The schema matchings can be automatically obtained from different kinds of metadata (foreign-key dependencies, view definitions and user queries) in form of join and attribute mapping relationships. The schema mapping and merge query generation steps are completely automated. The user just selects a start table and a desired target table with new attributes.

Table 5.15: Comparisons of four schema and data integration approaches

Approaches	Input Metadata	Required User Actions	Automatisation opportunities			Result
			Schema matchings	Schema mappings	Merge query	
Schema Integration[41]	– A set of source schemas	– Analysis source schemas – Solve conflicts between schemas – Merge (or transform) schemas	No	No	No	A unified, global schema which could represent all the source schemas
Mediation-based Data Integration[42], [43]	– A set of source schemas – Schema matchings between source schema and mediated table	– Define the schema of the mediated table – *Infer schema mappings*	No	Yes	Yes	A query that merges the source schemas and generates the mediated table
Schema Complement[12]	– A source table – A set of source schema	– Select target table	Yes	Yes	Yes	An augmented source table with new attributes from target table
Drill Across[6]	– A set of source schemas – A set of dimensions used in the source source schemas	– Select at least two source fact tables	Yes	Yes	Yes	A new fact table contains all the measures of source fact tables
Schema Augmentations[53]	– A source table – A set of source schema and metadata of these schemas – *Hierarchy defined in the schemas* – *Aggregable properties of attributes*	– Select target table – Choose reduction operations – *Select new attributes* – *Select the path to merge*	Yes	Yes	Yes	An augmented source table

6

Applications and Experiments

Contents

In this chapter, we describe the experiments we carried out using the implementation of our framework discussed in Chapter 4. The experiments covered both the efficiency of the implementation and the validation of our approach.

6.1 Performance Tests

The performance experiments evaluate the performance of the implementation of the attribute graph generation algorithm ATG (Algorithm 1, page 87 and the dimension identifier computation CDI (Algorithm 2, page 89). The experiments are conducted on a HANA instance with 250 GB of main memory and 300 GB of disk space.

6.1.1 Attribute graph computation

The ATG algorithm takes a hierarchy table (HT) as input and computes the attribute graph by applying a sequence of SQL queries (the steps are described in in Section 4.2.1 and the SQL queries are described in **??**). The hierarchy table HT is

the only input and the performance of ATG mainly depends on the size, the number of null values and the number of attributes in HT.

Each hierarchy table HT is an encoding of a dimension table and its size depends on the size of the dimension table, the size of the attribute active domains (distinct values for each attribute), the number of distinct child-parent value mappings in the hierarchy instance etc. More formally, we can identify the following four parameters that have an effect on the size of HT:

1. The size of the input dimension table (number of rows);
2. The number of hierarchy nodes / dimension attributes (without $\bot$ and $\top$);
3. The number of $+$-edges in the resulting attribute graph;
4. The total number of edges in the resulting attribute graph.

To evaluate the performance impact of each parameter, we generated several sets of hierarchies (dimension tables), where for each set we vary one parameter.

Experiment 6.1: Varying the dimension table size
The size of the hierarchy table HT increases along with the size of the dimension table and we expect that the ATG computation time should increase proportionally.

We generate six dimension tables for this experiment. To measure only the influence of the table size, we enforce that the dimension table does not contain any null values and the resulting attribute graphs all have a strict linear structure. Then, the attribute graph of a hierarchy with n nodes (attributes) has exactly $n - 1$ edges and all edges are labeled by $\mathbf{f}$. The tuple generation algorithm for each attribute in the hierarchy a new unique value, which is the concatenation of its parent-attribute value with a randomly generated numeric value in the interval $[0, 9999]$. This randomized process produces linear hierarchies with no null values where each attribute literally determines its parent attribute (the corresponding attribute graph only contains $\mathbf{f}$-edges).

Example 6.1. The generated tuples of a dimension table T with attributes L1 $\preccurlyeq$ L2 $\preccurlyeq$ L3 $\preccurlyeq$ L4 $\preccurlyeq$ L5 are shown in Table 6.1. Each value is computed by its parent value concatenating a random numeric value in the interval $[0, 9999]$.

Table 6.1: A dimension table with a strict linear structured hierarchy

	L1	L2	L3	L4	L5
t_1	63.789.266.7426.2629	63.789.266.7426	63.789.266	63.789	63
t_2	99.729.575.186.7874	99.729.575.186	99.729.575	99.729	99
	...				

The computation time of the six linear dimension tables with 5 attributes is shown in Figure 6.1. The result shows that the computation time increases linearly with the input dimension table size. As detailed in **??**, SQL query used to compute the attribute graph is a one-pass procedure using simple selections and joins, so the computation time of SQL query increases together with the size of the table that the query applied.

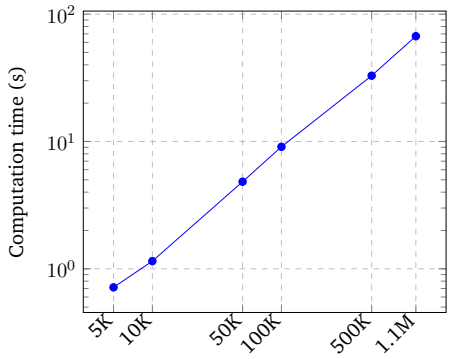


Figure 6.1: Computation time for different dimension table size (number of rows)

Experiment 6.2: Varying the number of dimension attributes

In this experiment, we control the number of dimension attributes in the dimension table. When the number of dimension attributes grows, the total amount of distinct attribute domain values in the dimension table also grows and will increase the size of the HT which causes the augmentation of the attribute graph's computation time.

We use the same tuple generation process as in Experiment 1 for producing dimension tables with a strict linear hierarchy. Each attribute then has an active domain of the same size which corresponds to the number of tuples in the table. We reuse the two dimension tables with $10K$ and $50K$ rows and five attributes (nodes) generated for Experiment 1 and generate four other dimension tables by augmenting the number of attributes. We obtain two sets of tables with $10K$ and $50K$ rows

respectively, where each sets contains three dimension tables with respectively 5, 10 and 20 attributes.

The computation time for each table is shown in Figure 6.2. The result shows that the computation time increases linearly with the number of attributes in the dimension table. HT describes for each node in the complete paths from the bottom level attribute value to the top level attribute value. Then, for a strict linear hierarchy, each new attribute increases the size of HT by the number of attribute values and, since the number of the complete paths remains the same ($10K$ rows or $50K$ rows), HT grows linearly. Since the performance of $TGTG$ increases linearly with the size of HT, in our experiments it also grows linearly with the number of attributes. We can conclude that for linear strict hierarchies, the performance of ATG mainly depends on the size of HT and is independent of the number of attributes.

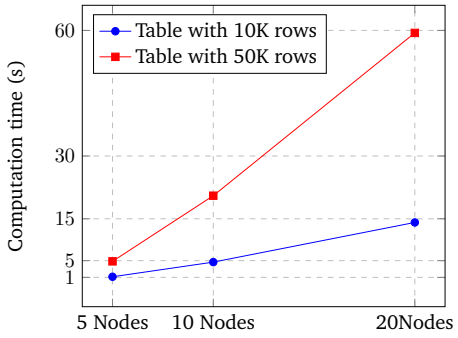


Figure 6.2: Computation time for different number of nodes

Experiment 6.3: Varying the number of $+$-edges

In the third set of experiments, we relax the condition of strictness (each value has a single parent) and allow attribute values to have several parent values. This introduces $+$-edges in the corresponding attribute graphs and increases the size of the HT table (the number of non-strict child-parent value mappings in the hierarchy leads to the augmentation of the complete paths to the top level attribute value for certain domain values). To achieve non-strictness, we modify the generation of the dimension table with $10K$ rows over 10 attributes and generate five dimension tables with an increasing number of $+$-edges in the result attribute graphs. These $+$-edges are simply created by removing the parent prefix of the values for certain attributes.

Example 6.2. Tuples of a dimension table T with attributes L1 $\preccurlyeq$ L2 $\preccurlyeq$ L3 $\preccurlyeq$ L4 $\preccurlyeq$ L5 are shown in Table 6.2. The attribute values of L3 are generated without

concatenating its parent values. Then, for example, t_3 and t_4 share the same value of L3 but have different values for attribute L4. Consequently, the edge (L3, L4) is a $+$-edge.

Table 6.2: A dimension table with one $+$-edge

	L1	L2	L3	L4	L5
t_3	63.789.266.7426.2629	63.789.266.7426	266	63.789	63
t_4	10.124.266.1924.9275	10.124.266.1924	266	10.124	10
	...				

We generate 5 dimension tables which contain contain $10K$ rows over 10 attributes with respectively 0, 2, 4, 6, 8 $+$-edges. Figure 6.3 shows the performance evolution for additional $+$-edges. We can see that, contrary to the previous experiments, where the computation time increased with the size of the HT table, in this experiment the time for computing the attribute graph decreases with the number of $+$-edges. This result can be explained by the fact that the number of distinct attribute values of certain attributes decreases. Based on the SQL query shown in **??** Step 4 (**??**), when the number of distinct attribute values of certain attributes decreases, the number of partitions by attribute values decreases and the intermediate tables becomes smaller (e.g., time to compute PLUS_EDGE_WITH_NULL), this decreases the computation time for Step 4 in Algorithm 1 (Page 87). Since there are no nullable attributes, Step 5 in Algorithm 1 is ignored.

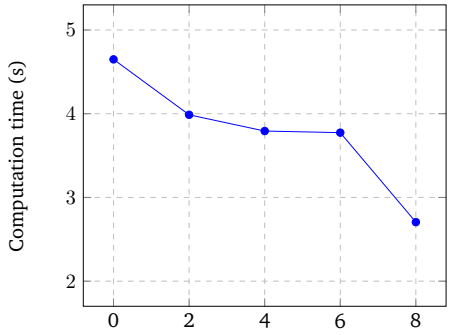


Figure 6.3: Computation time for different number of $+$-edges

Experiment 6.4: Varying the numbers of attribute graph edges

Finally, we relax the constraint of linearity and generate dimension tables producing attribute graphs with a varying numbers of optional edges.

To avoid the impact of $+$-edges, we enforce the resulting attribute graphs to only contain **f**-edges and **1**-edges. We generate two sets of dimension tables with 10 and 20 attributes respectively, and augment the number of edges for each set. The number of edges is modified by introducing null values for some attribute which produces additional optional attribute graph edges that jump over this attribute. The table generation still follows the same rules as in Experiment 1 (page 166) where attribute values are a concatenation of their parent values and a random numeric value between $[0, 9999]$. The only change consists in introducing random $null$ values. To ensure that the additional edge created by some null values is a **f**-edge, non-null attribute values are a concatenation of their first non-null ancestor value concatenated with the symbol '-' and the randomly generated numeric values.

Example 6.3. Some tuples of a dimension table T with attributes L1 $\preccurlyeq$ L2 $\preccurlyeq$ L3 $\preccurlyeq$ L4 $\preccurlyeq$ L5 are shown Table 6.3. The first tuple generates an f-edge from attribute L2 to attribute L3 and the second tuple and additional edge from attribute L2 to L4 because of the null value in attribute L3. Tuple t_6 contains a null value for L3 and t_6.L2 uses '10.124.-' to represent its parent value where '-' represents the null value in L3 and '10.124' represents the value of L4 concatenated with the value of L5. Each value of L2 and each non-null value of L3 contain their unique concatenated parent values. The edge (L3, L4) has label 1 and both edges (L2, L3) and (L2, L4) have label f. There is no label $+$-edge in the attribute graph generated by T.

Table 6.3: A dimension table with one additional edge

	L1	L2	L3	L4	L5
t_5	63.789.266.7426.2629	63.789.266.7426	63.789.266	63.789	63
t_6	10.124.-.1924.9275	10.124.-.1924	-	10.124	10
	...				

We generate 8 dimension tables containing $10K$ rows over 10 and 20 attributes respectively. Figure 6.4 shows the performance evolution for hierarchies with different additional edges. The number of additional edges is the difference between its actual edge number and $n - 1$ (n is the total attribute number). The size of the HT will augment proportionally the number of edges in the result attribute graph, and we might expect that the computation time of generation attribute graph will increase in consequence. However, as the results show, the computation time slightly decreases with the number of additional edges. Similar to Experiment 3, this result can be explained by the fact that the number of attribute values of certain attributes decreases, which decreases the computation time for Step 4 in Algorithm 1. Because

there are nullable attributes, Step 5 in Algorithm 1 can not be ignored, the decrease of time here is less significant than Experiment 3.

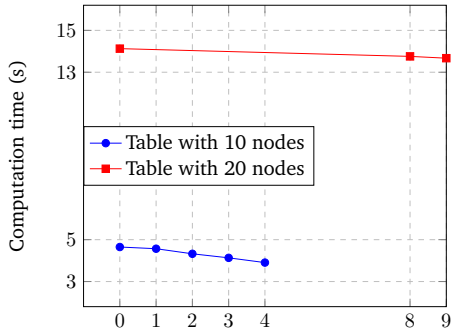


Figure 6.4: Computation time for different number of additional edges

Finally, for hierarchies of less than 10 nodes and less than $50K$ rows, which corresponds to a large number of real use cases, the computation of an attribute graph takes less than 20 seconds. 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 a system background call.

6.1.2 Dimension identifier computation

Algorithm 2 (page 89) computes the dimension identifier Y for a set of attributes X such that $Y \mapsto X$ (CDI) using the attribute graph $\mathcal{D}$ of a dimension table. It is obvious that the performance of CDI only depends in the attribute graph $\mathcal{D}$ and is independent of the size of the dimension table.

We tested the performance of CDI with 20 different attribute graphs containing 5 to 20 nodes and 4 up to to 28 edges. With X equal to the set of all attributes (nodes), all dimension identifiers are computed in less than 9 milliseconds with a variation of 2 milliseconds.

6.2 Validation with Real Datasets

6.2.1 Business use case

Our second series of experiments evaluates the practical usage of our REST services implementing Algorithm 3 for Computing Schema Augmentations (CSA),

Algorithm 4 for Reduction Query Generation (RQG), and Algorithm 5 for Merging Schema Augmentations(MSA) described in Chapter 4. The experiments are done on an SAP server with 2 TB of main memory and 1 TB of disk space.

Business Dataset

We use the dataset extracted from a real-world business intelligence application running for a worldwide clothing company which performs retail store stock analysis, customer analysis and customer segmentation. The application contains 42 information views representing dimension tables, and 145 carefully optimized information views representing fact tables.

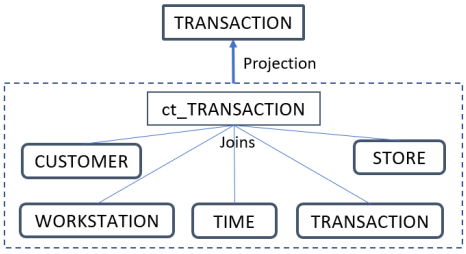
We categorize the fact table views into the following four types:

- *first-level views* (22 views) are the views defined as star joins between a non-analytic table storing facts (possibly completed by using left-outer joins with other tables providing details) and dimension tables.
- *join views* (72 views) are the views defined using a sequence of joins (mostly left-outer joins) or star joins among analytic and non-analytic tables, possible contain some aggregations. They are also called *hand-craft views* and denoted by *HCV*.
- *union views* (36 views) are the views defined by the union of two or more analytic tables.
- *aggregation views* (15 views) are the views defined by aggregation and projection queries on other fact tables.

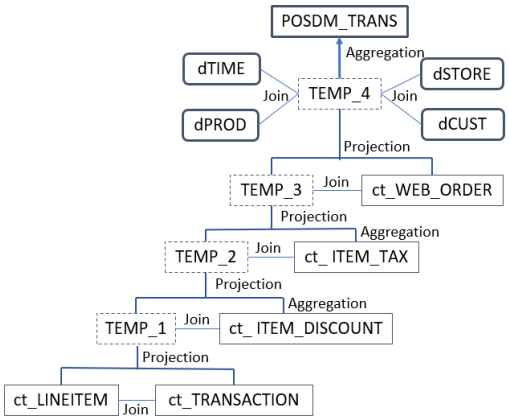
Example 6.4. Figure 6.5 shows four different types of views defined in the business application. Dimension tables are represented by bold rounded rectangles, fact tables by bold square rectangles, database tables by regular square rectangles, and intermediate tables by dashed rectangles. Intermediate tables store temporary results during the view constructions (e.g. $\mathrm{TEMP\_1}$ stores the temporary result of the join between $\mathrm{ct\_LINEITEM}$ and $\mathrm{ct\_TRANSACTION}$).

Fact table $\mathrm{TRANSACTION}$ shown in Figure 6.6a is a first-level view, it is defined as a star join between the non analytic table $\mathrm{ct\_TRANSACTION}$ and five dimension tables $CUSTOMER$, $WORKSTATION$, $TIME$, $TRANSACTION$ and $STORE$. Fact table $\mathrm{POSDM\_TRANS}$ shown in Figure 6.6b is a join view, it contains a sequence of joins between tables $\mathrm{ct\_LINEITEM}$, $\mathrm{ct\_TRANSACTION}$,

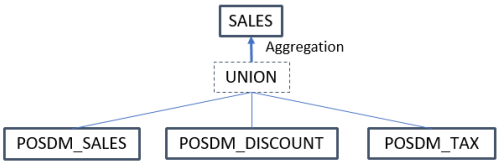
Figure 6.5: Constructions of views



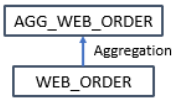
(a) *First-level view*: TRANSACTION



(b) *Join view*: POSDM_TRANS



(c) *Union view*: SALES



(d) *Aggregation view*: AGG_WEB_ORDER

ct_ITEM_DISCOUNT, ct_ITEM_TAX and ct_WEB_ORDER, followed by a star join with dimension tables dPROD, dTIME, dCUST and dSTORE, there are aggregations applied on table ct_ITEM_DISCOUNT and ct_ITEM_TAX. Fact table sales shown in Figure 6.6c is a union view, it is defined by a union between three fact tables POSDM_SALES, POSDM_DISCOUNT and POSDM_TAX. Fact table AGG_WEB_ORDER shown in Figure 6.6d is an aggregation view, it is defined as an aggregation on the fact table WEB_ORDER.

Data preparation

In our first experiment, we select a user-defined 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, and possibly defines a filter condition on some attributes of the target table or adds some calculated attributes after the merge.

We illustrate our protocol with the example of an *Hand-Crafted View* (*HCV*) – POSDM_TRANS defined as Figure 6.6b. The definition of the *HCV* starts from a non-analytic table ct_LINEITEM, performs a sequence of left-outer joins with four non-analytic tables storing different facts: ct_TRANSACTION, ct_ITEM_DISCOUNT, ct_ITEM_TAX and ct_WEB_ORDER, followed by a star-join with dimension tables $TIME$, $PRODUCT$, $STORE$ and $CUSTOMER$.

Before running our experiment, we first extracted the attribute graphs of all dimension tables (dTIME, dLINEITEM, dWORKSTATION, dTRANS, dSTORE, dPROD, dTAX, dORDER, dCUSTOMER and dDISCOUNT) and of five *first-level views* corresponding to the non-analytic tables used in *HCV*:

- LINEITEM is defined over the non-analytic table ct_LINEITEM and dimensions dTIME, dLINEITEM, dWORKSTATION, dTRANS, dSTORE and dPROD, with two measures *QUANTITY* and *SALES_AMOUNT*.
- TRANSACTION is defined over the non-analytic table ct_TRANSACTION and dimensions dWORKSTATION, dTIME, dTRANS, dSTORE and dCUSTOMER, with measure *TOTAL_COST*.
- ITEM_DISCOUNT is defined over the non-analytic table ct_ITEM_DISCOUNT and dimensions dTIME, dLINEITEM, dWORKSTATION, dTRANS, dSTORE and dDISCOUNT, with measure *DISCOUNT_AMOUNT*.
- ITEM_TAX is defined over the non-analytic table ct_ITEM_TAX and dimensions dTIME, dLINEITEM, dWORKSTATION, dTRANS, dSTORE and dTAX, with measure *TAX_AMOUNT*.
- WEB_ORDER is defined over the non-analytic table ct_WEB_ORDER and dimensions dTIME, dSTORE, dORDER and dCUSTOMER, with two measures *ORDER_AMOUNT* and *SHIPPING_COST*.

We also define the aggregable properties of all measure attributes (*SALES_AMOUNT*, *QUANTITY*, *DISCOUNT_AMOUNT*, etc.) using the rules defined in Section 2.2.6.

The first-level views; the dimension tables and the underlying non-analytic tables are crawled to extract all direct and derived relationships (including PK-FK relationships) and to compute the dimension and fact identifiers, as explained in Section 4.2. An extract of the resulting SC graph among analytic tables is shown in Figure 6.7.

Figure 6.8 gives a partial view of the SC graph for the analytic tables joined in the *HCV*. The table cardinalities range from $1.4M$ rows (WEB_ORDER) to $23M$

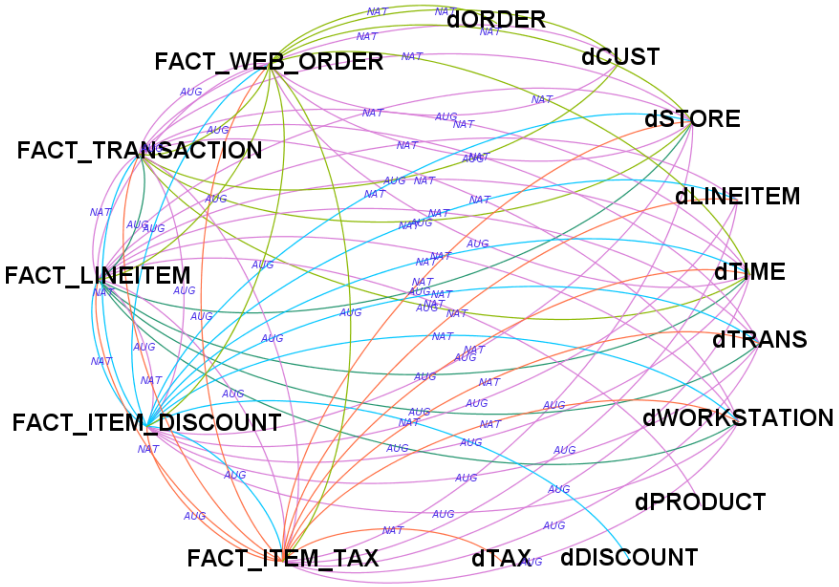


Figure 6.7: Complete SC graph for analytic tables

rows (ITEM_TAX and LINEITEM). We use the notation WEB_ORDER(D5, D8) to state that the fact identifier of table WEB_ORDER is the union of dimension identifiers 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 as shown in Figure 6.7. 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, the dependency as $\{D5, D8\} \mapsto D7$ is propagated from the primary key of ct_WEB_ORDER.

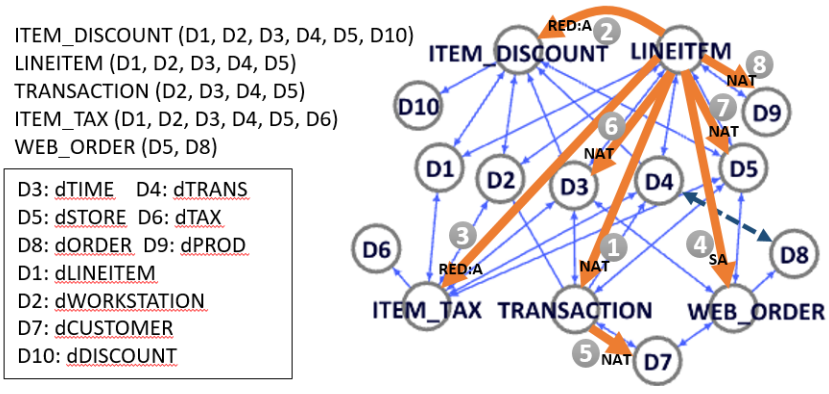


Figure 6.8: Partial SC graph for analytic tables

Experiment 6.5: Controlled Generation of Hand-Crafted View *HCV*

The goal of this experiment is to show that an expert can use our schema complement workflow to generate a view (GV) which is equivalent to the pre-selected hand-crafted view (HCV).

We apply our schema augmentation REST service workflow to generate the view *GV* starting from the first level fact table LINEITEM. This table is merged with other tables in eight successive schema augmentation steps illustrated by numbered arcs in Figure 6.8. The final result *GV* should produce the same table (schema and contents) as *HCV*.

GV is generated in 8 steps where each steps merges the result of the previous step with a new dataset:

1. Step 1 adds some attributes from dimension dCUSTOMER (D7) by a "natural" merge (the SC edge in Figure 6.7 from LINEITEM to TRANSACTION has label “NAT”).

2. In steps 2 and 3, LINEITEM is complemented with measure attribute *DISCOUNT_AMOUNT* from table ITEM_DISCOUNT and measure attribute *TAX_AMOUNT* from table ITEM_TAX. Both SC edge connecting LINEITEM to these two tables in Figure 6.7 have label “AUG”. Table ITEM_DISCOUNT is first transformed by an aggregate reduction over attributes from dimension D10 : dDISCOUNT into a natural schema complement of the result of step 1 before being merged. The same "reduce and merge" step is applied to table ITEM_TAX which is first reduced over attributes from dimension $D6$ into a natural schema complement before being merged with the result of step 2.

3. Step 4 adds attributes from dimension D8 : dORDER. The SC edge from LINEITEM to WEB_ORDER has label “AUG” with common attributes from dimensions D3 : dTIME and D5 : dSTORE. We define a relationship between dimensions D4 : dTRANS and D8 : dORDER (dotted line in Figure 6.8) and compute the augmented merge with table WEB_ORDER.

4. Steps 5 to 8 add the attributes from dimensions D7 : dCUSTOMER, D3 : dTIME, D5 : dSTORE and D9 : dPROD by a sequence of natural merge operations.

Finally, we compare the number of rows and the values of the hand-crafted view *HCV* and the generated view *GV* generated by the previous steps over the real-world dataset. Both view definitions compute the same result and we conclude that we were able to rebuild the hand-crafted view in a structured and controlled way using

our REST services. In particular, we can also formally state that GV satisfies the quality criteria introduced in Section 3.4 which were not guaranteed by HCV.

Experiment 6.6: Materalization Cost : *GV* versus *HV*

Our second use-case experiment compares the computation performance of the original view *HCV* and the generated view *GV* using SQL queries. The performance measures were done on a server with 2 TB of main memory and 1 TB of disk space. We might expect that the materialization of *GV* takes more time than the materialization of *HCV* since the *HCV* SQL query directly accesses the non-analytic tables and might benefit from the existing indexes on these tables.

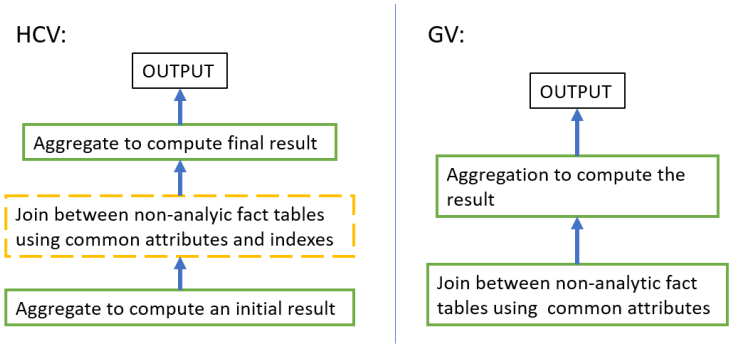
We first compared the time necessary for a complete materialization of both views and can observed that the computation of *GV* is 1.5 times slower than the computation *HCV* which confirms our initial assumption. Secondly, we applies a series of aggregation queries with random combinations of dimension attributes and aggregable attributes on *HCV* and *GV*. The performance results are more open to discussion. In 17% of the cases, *GV* performs 1.15 to 3.1 faster than *HCV*, and in 74% of the cases, *HCV* performs 1.1 to 4.5 faster than GV. In the rest of cases, both have similar performance.

In order to better understand the factors causing these performance variations we compared the execution plans of the queries generated for GV and HCV. In Table 6.4 shows the execution times of two simple aggregation queries in *HCV* and *GV* and we can see that both queries have an opposited behavior when executed on GV and on HCV. Their execution plans are shown in Figures 6.9a and 6.9b. The query plans

Table 6.4: Performance of Q_1, Q_2 in GV and HCV

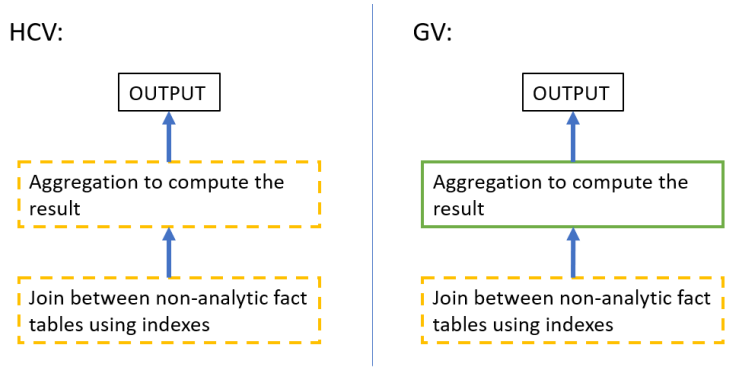
	Q_1	Q_2
HCV (in s)	10.33	2.61
GV (in s)	3.53	10.144

are deployed and executed on two different HANA query engines, an analytic query engine (solid green box) and a non-analytic query engine (dashed orange box). By comparing the execution plans with the execution time in Table 6.4, we observe that query plans using both engines take more time than queries executed in a single engine: query plan $Q_1(HCV)$ uses both engines and costs 8 seconds more than query plan $Q_1(GV)$ which is executed only on one engine. This difference is probably related to the additional overhead for switching between these to engines. We can



$$Q_1 = Agg_{GV/HCV}(SUM(DISCOUNT)|\{D_1, D_2, D_3, D_4, D_5\})$$

(a) Execution plans of Q_1



$$Q_2 = Agg_{GV/HCV}(SUM(TAX)|\{D_1, D_2, D_3, D_4, D_5\})$$

(b) Execution plans of Q_2

Figure 6.9: Query execution plans comparisons

conclude that the performance variations are mainly caused by the optimization strategy and the switching overhead for plans using both query engines.

Observations

Experiments 5 and 6 illustrate that our REST-based schema augmentation workflow can be used to regenerate for a randomly chosen join-view *HCV* (which represents half of the fact tables of the considered application) an equivalent view (*GV*) with comparable execution performance. This is a very positive result since it illustrates that our schema augmentation workflow can assist business users to build complex views by manipulating analytic views with meaningful attributes. By contrast with our solution, the creation of an *HCV* requires a strong programming (SQL) and data modeling expertise to manually build these views and a precise knowledge of the database schema to express the join conditions, to decide when a pre-aggregation is necessary, to identify useful measure attributes and to choose which aggregation

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 overhead 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.

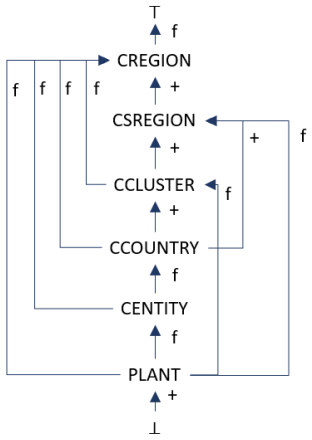


Figure 6.10: Attribute graph in dimension STORE

Illustration of ambiguous value detection

As another substantial advantage, our REST service provides quality guarantees that are difficult to fulfill by the developer of an *HCV* view. To illustrate this point, suppose that the aggregated view AGG_WEB_ORDER shown in Figure 6.6d is created as: $\mathcal{A}gg_{\text{WEB\_ORDER}}(\text{SUM}(\textit{ORDER\_AMOUNT}) \mid \text{X} \cup \text{A\_STORE})$ 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 6.10. Now suppose that an expert wants to extend an *HCV* with a new measure attribute $\text{SUM}(\textit{ORDER\_AMOUNT})$ by joining *HCV* with AGG_WEB_ORDER on attributes $\text{X} \cup$ JA_STORE, where JA_STORE is a subset of A_STORE. The choice of the attributes in JA_STORE strongly influences the correctness of the obtained result, and in particular the existence of ambiguous values. Table 6.5 shows the cases of ambiguous values for $\text{SUM}(\textit{ORDER\_AMOUNT})$, 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 6.5: Detection of ambiguous values

Attributes in JA_STORE	Is ambiguous	Missing attributes
PLANT, CCOUNTRY, CENTITY	N	-
CENTITY, CCLUSTER, CSREGION, CREGION	N	-
CCOUNTRY, CSREGION	Y	CCLUSTER
CCLUSTER, CREGION	Y	CSREGION

6.2.2 Feature engineering use case

In the following experiment, we validate again the usage of our REST service in a *feature engineering* application.

Feature Engineering Dataset

We perform the experiment in a predictive analysis application of a retail bank where data analysts want to predict whether clients without a credit card will obtain a card within 3 months following a given reference date.

The application uses tables from a publicly available database *PKDD* [54]. The tables are related through PK-FK relationships as shown in Figure 6.11. Each account has both static characteristics (e.g. date of creation, branch address) stored in table ACCOUNT and dynamic characteristics (e.g. debited and credited payments, account balance) stored in table TRANSACTION. Table CLIENT describes characteristics of persons who can *use* and *own* accounts. Clients and accounts are related by table DISPOSITION: one client can use several accounts and one account can by used by several clients, but each account can only be *owned* by one client. Table CREDIT_CARD describes the credit cards that are issued for an account. Tables GEOCODE and TIME provide detailed geographical and time information.

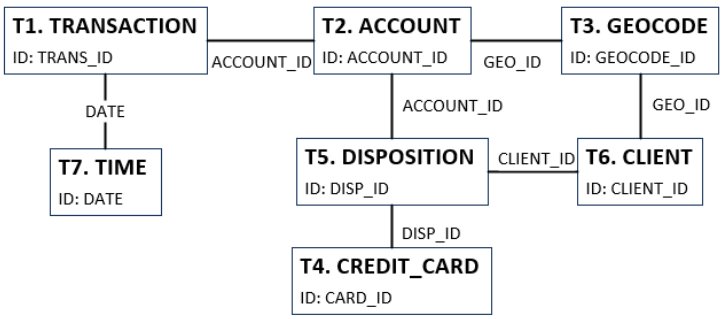


Figure 6.11: Relationships for the bank retail database

The application development includes a *feature engineering* step to build a training dataset. This training dataset is reused as input information for further operations during the prediction analysis. The training dataset is built by starting from a table describing a CLIENT entity and a *reference date*. The developer augments this table with as many new attributes (denoted by *features*) as possible through a sequence of database queries. The goal is to better describe the past behaviours of their client entities, for example, the amount and balance of their credit card during the past 12 months, whether they got a new card in the past three months, etc). The final *feature dataset* is denoted by *FEAT*. The augmentation queries include joins, calculated projections, filters, aggregations and pivot operations.

Data Preparation and Experiments

The goal of our experiment is to verify if a data scientist can semi-automatically generate a view *GV* that is equivalent to *FEAT* by iteratively extending the dimension table representing a client entity through several schema augmentation steps.

In our experiment, the script of SQL queries used to build the training dataset (*FEAT*) consists of around $1,500$ lines of code comprising many sub-queries involving 16 joins and 132 expressions to compute measure attribute values.

Before running our experiment, we first created 6 dimension tables, denoted by $\mathrm{dACCOUNT}$, dGEO, $\mathrm{dCLIENT}$, dCARD, dTIME and $\mathrm{dTRANSACTION}$, and 3 fact tables over the bank retail database tables as:

- TRANS is defined over database table $\mathrm{TRANSACTION}$ and dimensions dTIME, $\mathrm{dACCOUNT}$ and $\mathrm{dTRANSACTION}$ and has two measures *AMOUNT* and *BALANCE*.
- $\mathrm{ACC\_DISPO}$ is defined over database table $\mathrm{DISPOSITION}$ and dimensions $\mathrm{dCLIENT}$ and $\mathrm{dACCOUNT}$.
- $\mathrm{CARD\_DISPO}$ is defined over a join of tables $\mathrm{CREDIT\_CARD}$ and $\mathrm{DISPOSITION}$, and over dimensions $\mathrm{dCLIENT}$, dTIME and dCARD.

The attribute graphs of the dimension tables and the aggregable properties of measures for the fact tables are defined manually. Finally, all tables are crawled by the HANA Crawlers to extract all direct and derived relationships (including PK-FK relationships) and to compute the dimension and fact identifiers. The resulting SC graph is shown in Figure 6.12.

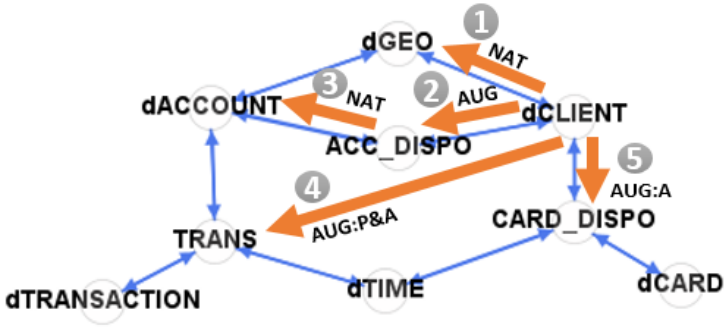


Figure 6.12: SC graph for the analytic tables

Experiment 6.7: Feature Table View Generation
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 complement / augmentation steps as indicated by the numbered arrows in the SC graph of Figure 6.12.

1. In Step 1 performs a natural merge of dCLIENT and dGEO using common attribute GEO_ID ("NAT" edge in Figure 6.12).

2. In step 2 applies an augmented merge of the previous table with fact table ACC_DISPO using the common attribute CLIENT_ID to add attribute ACCOUNT_ID ("NAT" edge in Figure 6.12). This step multiplies the rows in dCLIENT.

3. In step 3, the resulting table is augmented with detail attributes from dimension dACCOUNT through a "NAT" edge using the common attribute ACCOUNT_ID.

4. In step 4, measures from fact table TRANS are added. The SC edge from dCLIENT to TRANS is labeled "AUG" with common attribute ACCOUNT_ID. To apply a natural merge of augmented dCLIENT with TRANS, a reduction query that reduces attributes DATE, TRANS_ID is executed. The following user-defined actions have also been executed:

   a) **User actions:** a filter on attribute DATE pre-selects the facts within the 12 months preceding a user-given reference date; the attribute TRANS_TYPE is pivoted as a column with values coming from measures *AMOUNT* and *BALANCE*.

b) **Reduction operations:** A calculated attribute MONTH(DATE) (transaction month) is pivoted as a column with values from measures *AMOUNT* and *BALANCE*; the attribute TRANS_ID is removed by an aggregation reduction.

The two operations are injected in the reduction query generation over table TRANS. The final query is used to be merged with data set dCLIENT.

5. In Step 5, a new measure attribute that gives the number of credits cards for each client is added from table CARD_DISPO. The SC edge from dCLIENT to CARD_DISPO has label "AUG" and common attribute CLIENT_ID. A reduction query is applied on CARD_DISPO that removes DATE and computes an aggregated CARD_ID using aggregation function $COUNT$. Then a natural merge is computed between the previously augmented dataset dCLIENT and the reduced fact table CARD_DISPO.

The final result table *GV* has 156 attributes of which 135 are measures. Both, the generated table *GV* and the user defined table *FEAT*, are identical. We show in Figure 6.13 the complete definition of *GV* where each temprorary result represents a step in the view generation process.

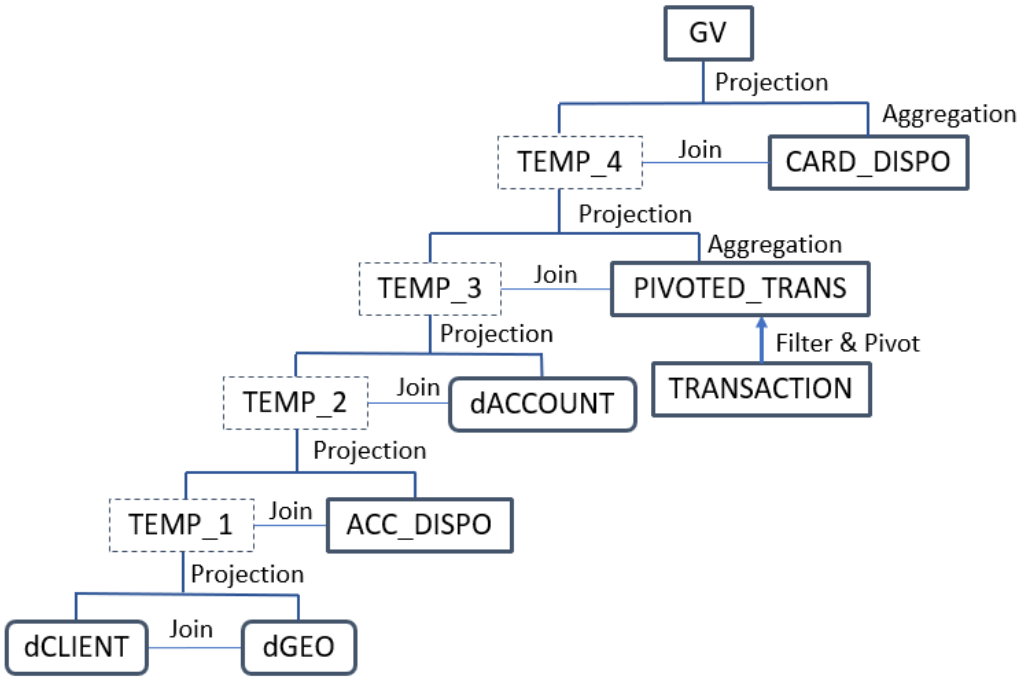


Figure 6.13: Construction of GV

Observations

Our semi-automatic view generation process has several advantages versus the manual creation of table *FEAT*. First, the schema augmentations required to build table *FEAT* are successfully suggested by our CSA API and the order in which they are returned matches pretty well the user needs. Second, the simple user actions in steps 4 and 5 yield complex reduction queries involving filter, pivot, and aggregate operations through the RQG API. Isolating the reduction actions in the generation process provides a great flexibility. For instance, in Step 4, the filter conditions could be changed to select facts within the 2 years preceding the reference date, or the pivot operations could be changed to WEEK(DATE). Third, the formal aggregable properties on fact table TRANS enables a fine control of which aggregate functions can be applied to 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.

7

Summary and Perspectives

Contents

7.1 Summary

In this thesis, we present a complete solution for discovering and merging schema augmentations for analytic and non-analytic tables. We introduce *attribute graphs* to describe the logical structure of hierarchical dimensions and *aggregable properties* to express the capability of aggregation for each attribute. Attribute graphs capture the functional (FD) and literal functional (LFD) dependency constraints in analytic tables and define an efficient data structure for computing dimension and fact identifiers. We introduce *schema augmentation graphs* which facilitate the process of discovering candidate augmentation tables (schemas) for extending some given start tables. Our model includes the definition of three reduction operations to transform schema augmentation tables into natural schema complement tables. We also introduce *formal quality conditions* to check the correctness of the target augmentation table by detecting *ambiguity* and *incompleteness* issues with respect to the underlying dimensions and start table. We also present a number of *algorithms for validating and repairing schema augmentation tables* before merging them with the start table. The theoretical model and the is completely implemented in *SAP HANA* and accessible through an API interface. We also validated the performance and usability of our model through a series of *experiments* and different application scenarios.

Our schema augmentation approach generalizes the existing state of the art on schema complements and drill-across queries. It only requires the source schema definitions and some automatically generated and user-defined attribute metadata (aggregable properties) to detect schema augmentations and produce correctly aggregated schema complements. It reduces the assumptions enforced by previous approaches on the input datasets and accepts non-strict, non-covering and non-onto dimension hierarchies with multiple top-level attributes.

7.2 Future Work Directions

The results of this thesis opens several directions for future work.

7.2.1 Schema matching discovery

Our work only considers reliable one-to-one schema matchings defined by attribute mapping relationships, join relationships and derived relationships. The discovery of the relationships are embedded into the metadata loader as explained in Chapter 4. One perspective of our work is to extend our approach to other types of schema matchings that can be discovered automatically through heuristic methods [12], [14], [46], [51]. Whereas heuristic schema matching algorithms introduce uncertain table relationships, they can produce new interesting schema augmentation candidates which are not detected through reliable schema matchings. Many-to-many relationships also are a challenge for our approach, since they require the generation of more sophisticated schema mapping queries including data fusion operators.

Example 7.1. For example, consider the dimension $STORE$ from Figure 6.10 on Page 179 and a non-analytic table COUNTRY_CODE with tuples as shown in Table 7.1:

Using the instance-based matching described in Section 5.3 on Page 114, we could compute the Jaccard similarities between attribute domains of $STORE$ and COUNTRY_CODE to discover possible schema matchings. Then, a potential schema matching is $STORE$.CCOUNTRY $\cong$ COUNTRY_CODE.COUNTRY_NAME with score $J(STORE.\text{CCOUNTRY}, \text{COUNTRY\_CODE.COUNTRY\_NAME}) = 0.75$.

Table 7.1: Table STORE and COUNTRY_CODE

(a) *STORE*

PLANT	CENTITY	CCOUNTRY	CCLUSTER	CREGION
9044	XAGNER	CHINA	CHINA	ASIA
1159	ROZAS	SPAIN	SPAIN & PORTUGAL	EMEIA
1029	RENNES	FRANCE	FRANCE & BELGIUM	EMEIA

(b) COUNTRY_CODE

COUNTRY_NAME	2_CHAR	3_CHAR	UN_CODE
SPAIN	ES	ESP	724
FRANCE	FR	FRA	250
CHINA	CN	CHN	156
PORTUGAL	PT	PRT	620

7.2.2 User-specified augmentation and reduction operation suggestion

In our current implementation, the user selects the start table and a candidate target table for generating a merge query. When necessary, for transforming a target table into a schema complement, the user specifies the reduction operations for each attribute that needs to be reduced. The manual choice of the target table and the reduction operations introduces more flexibility and control, but requires from the user some basic knowledge about the schema of the target table.

On future goal of our schema augmentation service is to assist users in their choice by generating ranked lists of candidate target tables and recommended reduction operations. The ranking criteria could be various and based on user-defined keywords or data-specific criteria like the data coverage of the common attributes or the distance of the target table from the start table in the schema complement graph. Reduction operations could be proposed based on the user preferences, on the user action history or on the desired data types of the reduced attributes.

Example 7.2. Consider table INVENTORY of Figure 2.9 on Page 37 as the start table. All other tables in Figure 2.9 are schema augmentation candidates for table INVENTORY. We could rank these tables as follows:

- Dimension tables $TIME$, $PROD$, $WAREHOUSE$, TAX, $STORE$ get higher scores when the user wants to augment INVENTORY with new dimension attributes.

- Fact tables SALES, SALES_SUM get higher scores when the user wants to augment INVENTORY with new measures.
- $WAREHOUSE$, $PROD$, $TIME$, TAX, SALES, SALEs_SUM get higher scores when the user prefers the "closest" target tables.
- SALES gets the highest score when the user prefers the "most similar" target tables (SALES has five attributes common with INVENTORY).

Assuming $STORE$ is selected as the target table, attribute STORE_ID must be reduced. The system might suggest ranking among the three reduction operations based on the following observations. An aggregate reduction could count the number of stores grouped by CITY, STATE, COUNTRY. A second possible reduction could be to pivot table $STORE$ by attribute STORE_ID. However, since STORE_ID is the dimension identifier, a pivot reduction would produce a new column for each store, and each tuple would have only one new column with a non-null value. Finally, filter reduction would produce a result table with a single tuple (store). The final ranking would put pivoting at the end of the list (two many columns with many null values) and probably prefer aggregation to filtering except if the user is interested into a particular store.

Bibliography

[1] C. S. Jensen, T. B. Pedersen, and C. Thomsen, *Multidimensional Databases and Data Warehousing*. Jan. 1, 2010.

[2] "Business Intelligence and Analytics Software." (), [Online]. Available: https://www.tableau.com/ (visited on 01/10/2019).

[3] "Power BI | Interactive Data Visualization BI Tools." (), [Online]. Available: https://powerbi.microsoft.com/en-us/ (visited on 01/10/2019).

[4] "Data Analytics for Modern Business Intelligence | Qlik." (), [Online]. Available: https://www.qlik.com/us (visited on 01/10/2019).

[5] "Sap Analytics Cloud." (), [Online]. Available: https://www.sapanalytics.cloud/ (visited on 03/09/2020).

[6] R. Kimball and M. Ross, *The Data Warehouse Toolkit*. 3rd: John Wiley & Sons, 2013, ISBN: 978-1-118-53080-1.

[7] SAP. "The virtual data model in sap s/4hana." (), [Online]. Available: https://help.sap.com/viewer/6b356c79dea443c4bbeeaf0865e04207/1809.000/en-US/8573b810511948c8a99c0672abc159aa.html (visited on 08/26/2019).

[8] A. Pattanayak, "High performance analytics with sap hana virtual models," *Journal of Computer and Communications*, vol. 5, no. 07, pp. 1–10, 2017.

[9] Trifacta. "Data Wrangling Tools & Software | Trifacta." (), [Online]. Available: https://www.trifacta.com/ (visited on 01/10/2019).

[10] "Paxata | Self-Service Data Preparation for Data Analytics." (), [Online]. Available: https://www.paxata.com/ (visited on 01/10/2019).

[11] "SAP Agile Data Preparation and Transformation Solution." (), [Online]. Available: https://www.sap.com/products/data-preparation.html (visited on 01/10/2019).

[12] A. Das Sarma, L. Fang, N. Gupta, *et al.*, "Finding related tables," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, ACM, 2012, pp. 817–828.

[13] M. Yakout, K. Ganjam, K. Chakrabarti, and S. Chaudhuri, "InfoGather: Entity augmentation and attribute discovery by holistic matching with web tables," en, in *Proceedings of the 2012 international conference on Management of Data - SIGMOD '12*, ACM Press, 2012, pp. 97–108, ISBN: 978-1-4503-1247-9. DOI: 10.1145/2213836.2213848.

[14] D. Deng, R. C. Fernandez, Z. Abedjan, *et al.*, "The Data Civilizer System," in *CIDR*, 2017.

[15] J. D. Ullman, H. García-Molina, and J. Widom, *Database System: The Complete Book*, 2nd ed. Prentice Hall, 2008.

[16] "Closed World Assumption." (), [Online]. Available: https://en.wikipedia.org/wiki/Closed-world_assumption (visited on 03/01/2019).

[17] A. Badia and D. Lemire, "Functional dependencies with null markers," *The Computer Journal*, vol. 58, no. 5, pp. 1160–1168, 2014.

[18] P. Atzeni and N. M. Morfuni, "Functional dependencies in relations with null values," *Inf. Process. Lett.*, vol. 18, no. 4, pp. 233–238, 1984, ISSN: 0020-0190. DOI: 10.1016/0020-0190(84)90117-0.

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

[20] T. Niemi, M. Niinimäki, P. Thanisch, and J. Nummenmaa, "Detecting summarizability in OLAP," *Data & Knowledge Engineering*, vol. 89, pp. 1–20, Jan. 2014, ISSN: 0169023X.

[21] T. B. Pedersen, C. S. Jensen, and C. E. Dyreson, "A foundation for capturing and querying complex multidimensional data," *Information Systems*, vol. 26, no. 5, pp. 383–423, 2001, ISSN: 03064379. DOI: 10.1016/S0306-4379(01)00023-0.

[22] A. Shoshani, "OLAP and statistical databases: Similarities and differences," in *Proceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, ACM, 1997, pp. 185–196.

[23] H.-J. Lenz and 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.

[24] J. Lechtenbörger and G. Vossen, "Multidimensional normal forms for data warehouse design," *Information Systems*, vol. 28, no. 5, pp. 415–434, 2003.

[25] C. A. Hurtado, C. Gutierrez, and A. O. Mendelzon, "Capturing summarizability with integrity constraints in OLAP," *ACM Transactions on Database Systems*, vol. 30, no. 3, pp. 854–886, 2005, ISSN: 03625915.

[26] J.-N. Mazón, J. Lechtenbörger, and J. Trujillo, "A survey on summarizability issues in multidimensional modeling," *Data & Knowledge Engineering*, vol. 68, no. 12, pp. 1452–1469, 2009, ISSN: 0169023X.

[27] J. Horner, I.-Y. Song, and 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.

[28] S. S. Stevens, "On the theory of scales of measurement," *Science*, vol. 103, no. 2684, pp. 677–680, Jun. 1946.

[29] T. B. Pedersen, C. S. Jensen, and C. E. Dyreson, "Extending practical pre-aggregation in on-line analytical processing," in *Proceedings of the 25th International Conference on Very Large Data Bases*, ser. VLDB '99, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999, pp. 663–674, ISBN: 1558606157.

[30] R. Torlone, "Two approaches to the integration of heterogeneous data warehouses," *Distributed and Parallel Databases*, vol. 23, no. 1, pp. 69–97, 2008, ISSN: 0926-8782, 1573-7578.

[31] A. Abelló, J. Samos, and 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.

[32] J. Horner and I.-Y. Song, "A taxonomy of inaccurate summaries and their management in OLAP systems," in *International Conference on Conceptual Modeling*, vol. 3716, Berlin, Heidelberg: Springer, 2005, pp. 433–448.

[33] J. Lee, M. Muehle, N. May, *et al.*, "High-performance transaction processing in SAP HANA," *IEEE Data Eng. Bull*, vol. 36, no. 2, pp. 28–33, 2013.

[34] *SAP HANA Modeling Guide*. SAP, 2019. [Online]. Available: `https://help.sap.com/doc/227fc55c4fc44a43b43752d6b127bdf3/2.0.04`.

[35] R. Brunel, J. Finis, G. Franz, *et al.*, "Supporting hierarchical data in SAP HANA," in *Data Engineering (ICDE), 2015 IEEE 31st International Conference on*, IEEE, 2015, pp. 1280–1291.

[36] M. Paradies, C. Kinder, J. Bross, T. Fischer, R. Kasperovics, and H. Gildhoff, "GraphScript: Implementing complex graph algorithms in SAP HANA," in *Proceedings of DBPL 2017*, Munich, Germany: ACM Press, 2017, pp. 1–4.

[37] In *SAP HANA SQL and System Views Reference*, SAP, 2016, pp. 88–99.

[38] "Unit converter." (), [Online]. Available: `https://www.unitconverters.net/`.

[39] E. Roschke, "Units and conversion factors," 2001.

[40] "SAP HANA smart data integration and SAP HANA smart data quality - SAP help portal." (), [Online]. Available: `https://help.sap.com/viewer/p/HANA_SMART_DATA_INTEGRATION` (visited on 01/10/2019).

[41] C. Batini, M. Lenzerini, and S. B. Navathe, "A comparative analysis of methodologies for database schema integration," *ACM computing surveys (CSUR)*, vol. 18, no. 4, pp. 323–364, 1986.

[42] L. L. Yan, R. J. Miller, L. M. Haas, and R. Fagin, "Data-driven understanding and refinement of schema mappings," in *Proceedings of the 2001 ACM SIGMOD international conference on Management of data*, 2001, pp. 485–496.

[43] R. J. Miller, L. M. Haas, and M. A. Hernández, "Schema mapping as query discovery," in *Proceedings of the 26th International Conference on Very Large Data Bases*, ser. VLDB '00, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2000, pp. 77–88, ISBN: 1558607153.

[44] E. Rahm and P. A. Bernstein, "A survey of approaches to automatic schema matching," *the VLDB Journal*, vol. 10, no. 4, pp. 334–350, 2001.

[45] Z. Bellahsene, A. Bonifati, and E. Rahm, Eds., *Schema Matching and Mapping*, en. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011. DOI: 10.1007/978-3-642-16518-4.

[46] A. Doan, A. Halevy, and Z. Ives, *Principles of Data Integration*, 1st. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2012, ISBN: 0124160441.

[47] S. Ram and V. Ramesh, "Schema integration: Past, present, and future," *Management of Heterogeneous and Autonomous Database Systems*, pp. 119–155, 1999.

[48] F. Nargesian, E. Zhu, K. Q. Pu, and R. J. Miller, "Table union search on open data," en, *Proceedings of the VLDB Endowment*, vol. 11, no. 7, pp. 813–825, Mar. 2018, ISSN: 21508097.

[49] P. Jaccard, "The Distribution of the Flora in the Alpine Zone.1," en, *New Phytologist*, vol. 11, no. 2, pp. 37–50, 1912, ISSN: 1469-8137. (visited on 11/28/2019).

[50] J. Bleiholder and F. Naumann, "Data fusion," *ACM computing surveys (CSUR)*, vol. 41, no. 1, pp. 1–41, 2009.

[51] M. J. Cafarella, A. Halevy, and N. Khoussainova, "Data integration for the relational web," *Proceedings of the VLDB Endowment*, vol. 2, no. 1, pp. 1090–1101, 2009.

[52] M. Rafanelli and A. Shoshani, "Storm: A statistical object representation model," in *International Conference on Scientific and Statistical Database Management*, vol. 420, Berlin, Heidelberg: Springer, 1990, pp. 14–29. DOI: 10.1007/3-540-52342-1_18.

[53] R. Liu, E. Simon, B. Amann, and S. Gançarski, "Discovering and merging related analytic datasets," *Information Systems*, vol. 91, p. 101 495, 2020.

[54] "Home page of pkdd discovery challenge." (), [Online]. Available: https://sorry.vse.cz/~berka/challenge/PAST/index.html.

List of Figures

List of Tables