Tabular Data Integration for Multidimensional Data Warehouse

Yuzhao Yang

▶ **To cite this version:**

Yuzhao Yang. Tabular Data Integration for Multidimensional Data Warehouse. Computer Science [cs]. Université Toulouse 1 Capitole (UT1 Capitole); Université de Toulouse; IRIT - Institut de Recherche en Informatique de Toulouse, 2022. English. NNT : . tel-03903570

HAL Id: tel-03903570
https://hal.science/tel-03903570

Submitted on 16 Dec 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par : *l'Université Toulouse 1 Capitole (UT1 Capitole)*

Présentée et soutenue le *15 Décembre 2022* **par :**

Yuzhao YANG

Tabular Data Integration for Multidimensional Data Warehouse

JURY

Agnès FRONT	Professeure, Université Grenoble Alpes	Rapportrice
Maguelonne TEISSEIRE	Professeure, INRAE	Rapportrice
Ladjel BELLATRECHE	Professeur, ENSMA	Examinateur
Olivier TESTE	Professeur, Université Toulouse 2 Jean Jaurès	Examinateur
Jérôme DARMONT	Professeur, Université Lumière Lyon 2	Co-directeur
Franck RAVAT	Professeur, Université Toulouse 1 Capitole	Directeur

---

École doctorale et spécialité :
EDMITT - Ecole Doctorale Mathématiques, Informatique et Télécommunications de Toulouse : Informatique et Télécommunications

Unité de Recherche :
IRIT: Institut de Recherche en Informatique de Toulouse (UMR 5505)

Directeur(s) de Thèse :
Franck RAVAT et *Jérôme DARMONT*

Rapportrices :
Agnès FRONT et *Maguelonne TEISSEIRE*

Acknowledgements

I would like to express my sincere gratitude to my supervisor **Professor Franck RAVAT** for the quality of his supervision with his profound knowledge, rigorous attitude and strong work enthusiasm. He sacrificed his time to give me pertinent advice and revise my work with high standards which helped me to look for perfection. He kept me motivated when I was slack, encouraged me when I was down. He also generously offered his praises to me when I got advancement, which render me a deeper passion for my research. Moreover, he actively introduced me to French culture and cuisine, so that I enjoyed a wonderful French life.

I would like to gratefully acknowledge my co-supervisor **Professor Jérôme DARMONT** for the quality of his supervision, his valuable advice, constant support, and remarkable patience during my PhD study. His elaborate guidance has allowed me to make continuous progress on my research, his careful revision of my work has lead my achievements to a higher quality. I would also like to thank him for giving me the opportunity to be a part of the BI4people project, with this interesting research subject and for the well-prepared project meetings that he organised where I largely benefited.

I would like to express my deep appreciation to all the members of the defense committee for making my defense be an enjoyable moment. I would like to give my warm thanks to my thesis reviewers **Professor Agnès FRONT** and **Professor Maguelonne TEISSEIRE** for evaluating and approving my work. Their attentive reviews, valuable comments, and enlightening questions have contributed greatly to polishing my work.

I am deeply grateful to my thesis examiners, **Professor Ladjel BELLATRECHE** and **Professor Olivier TESTE**, for their brilliant comments, helpful ideas and inspiring suggestions, enabling me to optimize my work.

Special thanks to **Professor Olivier TESTE**, head of SIG research team, for hosting me in the warm home of *Institut de Recherche en Informatique de Toulouse (IRIT)*. I would also like to thank him for his insightful guidance and constructive advice on my research. I am also thankful to **Doctor Fatma ABDELHEDI** for her valuable remarks and helpful suggestions to my work.

My sincere thanks also goes to **all my fellow colleagues** in *Univertité Toulouse I Capitole*, *IRIT*, for the company, exchange and joy they brought.

I would like to thank to **my family** for their love, support as well as their tremendous understanding and encouragement. My love and gratitude for them can hardly be expressed in words.

Finally, I would like to acknowledge the **French National Research Agency (ANR)** [1] for the funding of the BI4PEOPLE project to which my PhD thesis is attached.

[1] `https://anr.fr/Project-ANR-19-CE23-0005`

Abstract

Business Intelligence (BI) plays an important role in companies to support decision making processes. Nowadays, small companies, organizations or even individuals can exploit numerous data. However, the lack of experts prevents them from carrying BI projects out. It is thus necessary to automate the BI design process to make BI accessible for everyone. In BI architectures, data are integrated into Data Warehouses (DWs) usually modeled in a multidimensional way. Yet, tabular data widely exist in small enterprises, organizations and in the open data world. As a result, we intend to automate the DW design from tabular data.

Automatic DW design from tabular data requires the detection of different multidimensional components (facts, dimensions, hierarchies...). In case of multiple sources, several DWs may be generated. If they share common information, it is necessary to merge them as one integrated DW. During DW merging, missing data imputation should be carried out to achieve a better data analysis. Therefore, we propose a solution composed of three parts: (i) automatic DW design, (ii) automatic DW merging and (iii) dimensional data imputation.

Automatic DW design from tabular data is composed of measure detection and dimension detection for constructing facts and dimensions, respectively. For measure detection, we propose a machine learning-based approach that extracts three categories of features from numerical columns. Dimension detection includes functional dependency-based hierarchy detection and the distinction of parameters and weak attributes based on syntactic and semantic rules. We carry out experiments to validate that our approach is able to detect measures and different dimension elements with high effectiveness and efficiency.

For automatically merging DWs, we propose a process at both the schema and instance levels, consisting of level merging, hierarchy merging, dimension merging and star schema merging. Our approach takes the different DW structure elements into account. Moreover, our approach considers different cases and may generate star or constellation schemas. We conduct experiments to validate that our DW merging solution can correctly merge DWs at both schema and instance levels.

Finally, to address dimensional missing data, we propose a hybrid imputation approach named Hie-OLAPKNN that combines a hierarchical imputation (Hie) and a K-nearest neighbors-based imputation (OLAPKNN). Hierarchical imputation is based on functional dependencies between hierarchy levels and is launched first. The remaining missing data can then be completed by OLAPKNN, which applies a specific dimension instance distance and considers hierarchy dependency constraints. Our experiments show that Hie-OLAPKNN outperforms other approaches in terms of effectiveness, efficiency and respect of hierarchy strictness.

Résumé

La Business Intelligence (BI) joue un rôle important dans les entreprises pour soutenir les processus de prise de décision. Aujourd'hui, les petites entreprises, les organisations ou même les particuliers peuvent exploiter de nombreuses données. Cependant, le manque d'experts les empêche de mener à bien des projets de BI. Il est donc nécessaire d'automatiser le processus de conception et d'implémentation de systèmes de BI afin de le rendre accessible à tous. Dans les architectures BI, les données sont intégrées dans des entrepôts de données (EDs) généralement modélisés de manière multidimensionnelle. De plus, les données tabulaires sont largement répandues dans les petites entreprises, les organisations et dans le monde des données ouvertes. Par conséquent, nous avons l'intention d'automatiser la conception d'EDs multidimensionnels à partir de données tabulaires sans connaissance à priori des schémas.

La conception automatique d'EDs à partir de données tabulaires nécessite la détection de différents composants multidimensionnels (faits, dimensions, hiérarchies...). En cas de sources multiples, plusieurs EDs peuvent être générés. S'ils partagent des informations communes, il est nécessaire de les fusionner en un seul ED intégré. Pendant la fusion d'EDs, l'imputation de données manquantes doit être effectuée pour permettre une analyse de données de meilleure qualité. Par conséquent, nous proposons une solution composée de trois parties : (i) la conception automatique d'EDs, (ii) la fusion automatique d'EDs et (iii) l'imputation de données multidimensionnelles.

La conception automatique d'EDs à partir de données tabulaires comprend la détection de mesure et la détection de dimension pour définir respectivement le fait et les dimensions. Pour la détection de mesures, nous proposons une approche basée sur l'apprentissage automatique qui extrait trois catégories de caractéristiques. La détection de dimensions comprend la détection de hiérarchies (basée sur des dépendances fonctionnelles) et la distinction des paramètres et des attributs faibles (basée sur des règles syntaxiques et sémantiques). Nous avons réalisé des expérimentations pour valider que notre approche est capable de détecter les mesures et les différents éléments de dimension avec une efficacité et une efficience élevées.

Concernant la fusion automatique d'EDs, nous proposons un processus basé sur les schémas et les instances, composé de la fusion de niveaux, la fusion de hiérarchies, la fusion de dimensions et la fusion de schémas en étoile. Les expérimentations ont permis de valider notre solution de fusion d'EDs.

Enfin, pour traiter les données manquantes multidimensionnelles, nous proposons une approche d'imputation hybride appelée Hie-OLAPKNN qui combine une imputation hiérarchique (Hie) et une imputation basée sur les K-voisins les plus proches (OLAPKNN). L'imputation hiérarchique est basée sur les dépendances fonctionnelles entre les niveaux hiérarchiques. OLAPKNN applique une distance d'instances de dimension et tient compte des contraintes de dépendance hiérarchique. Nos expérimentations montrent que Hie-OLAPKNN

surpasse les autres approches en termes d'efficacité, d'efficience et de respect des contraintes hiérarchiques.

Publication List

National Conference

Yang, Y., Darmont, J., Ravat, F., & Teste, O. (2020). Automatic Integration Issues of Tabular Data for On-Line Analysis Processing. In 16e journées EDA Business Intelligence & Big Data (EDA), vol. RNTI-B-16, pp.5-18

International Conference

Yang, Y., Darmont, J., Ravat, F., & Teste, O. (2021). An Automatic Schema-Instance Approach for Merging Multidimensional Data Warehouses. In 25th International Database Engineering & Applications Symposium (IDEAS) (pp. 232-241).

Yang, Y., Abdelhedi, F., Darmont, J., Ravat, F., & Teste, O. (2021). Internal Data Imputation in Data Warehouse Dimensions. In International Conference on Database and Expert Systems Applications (DEXA) (pp. 237-244). Springer, Cham.

Yang, Y., Abdelhédi, F., Darmont, J., Ravat, F., & Teste, O. (2022). Automatic Machine Learning-Based OLAP Measure Detection for Tabular Data. In International Conference on Big Data Analytics and Knowledge Discovery (DaWak) (pp. 173-188). Springer, Cham.

Yang, Y., Darmont, J., Ravat, F., & Teste, O. (2022). Dimensional Data KNN-Based Imputation. In European Conference on Advances in Databases and Information Systems (ADBIS) (pp. 315-329). Springer, Cham.

Contents

List of Figures

List of Tables

Chapter I

Introduction

Contents

1 Research Context

Business intelligence (BI) systems are widely used in the industry, especially in large companies (Llave, 2017), combining operational data with analytical tools to present information in a structured and effective way to support decision making for planners and decision makers (Negash and Gray, 2008; Nelson, 2010). Chugh and Grandhi (2013) summarize the advantages of the application of BI systems in companies including (1) allowing companies to analyse data from multiple sources in multiple dimensions; (2) creating intelligence for decision making by seeking out patterns and meanings in data; (3) improving management strategies by rapidly rendering accurate reporting; (4) supporting in identifying the causes of operational problems to reduce inventory costs; and (5) helping to make accurate predictions to find future opportunities.

With the current digitization trend, small companies, organizations or even individuals can exploit a large number of data every day (Grabova et al., 2010; Raj et al., 2016) and the rise of open data makes various data even more accessible (Braunschweig et al., 2012). To be competitive and obtain valuable information from such data, these small entities are also interested in BI systems (Grabova et al., 2010).

Nevertheless, the design and implementation of a BI system need to be realized by experts who have the professional knowledge and deep skills in BI technologies, such as data warehousing and data visualization (Romero and Abelló, 2010). However, there is a general lack of such technical expertise in small entities (Raj et al., 2016). Moreover, commercial BI tools are expensive and are not affordable for them. Despite the existence of open source BI platforms (Lapa et al., 2014; Tutunea and Rus, 2012), they are still technically out of the reach of our target users (Abelló et al., 2013). As a result, the project BI4people [1] aims at bringing the power of BI systems to the largest possible audience, by automating the BI design and implementation process from data integration to On-Line Analytical Processing (OLAP) analysis and data visualization.

In current BI systems, data are integrated into Data Warehouses (DWs) in a multidimensional way (Chaudhuri et al., 2011). Data warehousing is the most challenging aspect of BI, requiring about 80% of the time and effort and generating more than 50% of the unexpected project costs (Watson and Wixom, 2007). Thus, automating the DW design and implementation process is an indispensable task in the BI4people project.

There exist various forms of data, but most of the data in small enterprises and organizations, as well as most of open data, are in tabular form (Roman et al., 2016; Borisov et al., 2021). There are different automatic DW design approaches Romero and Abelló (2009). Most of these methods focus on data sources with schema: relational data with Entity-Relationship (ER) schema, XML data with Document Type Definitions (DTDs), etc. Automatic DW design from tabular data without schema arises little attention and is not well addressed in the literature.

[1] `https://anr.fr/Project-ANR-19-CE23-0005`

Therefore, as a part of the BI4people project, in this manuscript, we intend to automate the DW design and implementation process from tabular data to allow small enterprises, organizations and even individuals without deep technical expertise to easily analyse data with BI systems.

2 Problem Definition

As we discussed in Section 1, we focus on tabular data, which are usually without schema. The lack of schema makes it hard to discover the relationships between attributes to design DW multidimensional schemas. Tabular data bear simple or complex structures (Adelfio and Samet, 2013). It is thus important to analyse the characteristics of different tabular data structures and customize different automatic DW design solutions. A DW is usually modelled as a multidimensional schema, which is composed of analysis subjects (facts) containing indicators (measures). These subjects are analysed according to different axes of analysis (dimensions) that are composed of attributes modeled through different views (hierarchies) (Ravat et al., 2008a). **Therefore, we have to identify attributes in tabular data as different elements such as measures or dimension attributes and detect the relationships between the attributes to create hierarchies.**

Users may have data coming from multiple sources and a DW may be constructed for each one of them. If there are DWs having common information, users may need to merge the DWs for analysing the data in a consolidated way. However, merging multidimensional DWs is challenging because it is not only necessary to merge them at the schema level, but also to merge the values of different attributes. Complex DW structure also requires to the consider different multidimensional components when merging DWs. **Therefore, we have to automatically merge these DWs into one integrated DW at both schema and instance levels by considering the multidimensional structure.** Moreover, a DW may be modelled as a star or constellation schema according to the number of facts and their association to the dimensions. We must take this into account.

During the merging process, there may be missing values in attributes of the merged DW. Missing data make aggregated data incomplete and thus have an impact on OLAP analyses. These missing data produce dashboards containing erroneous values and may thus lead to decision-making that can negatively impact the company. **Therefore, it is indispensable to carry out data imputation to replace missing data** for the sake of a more complete and accurate data analysis. Missing data imputation requires taking the DW structure and dependency constraints among hierarchy levels into account.

3 Manuscript Outline

Facing the various problems discussed in Section 2, this PhD thesis aims to automate the DW design and implementation to enable non-expert users take advantage of BI

by integrating data into DWs for further OLAP analyses and data visualisation. To do so, we propose a complete solution covering not only the automatic DW design and implementation from tabular data, but also the follow-up tasks in case of multiple sources including automatic DW merging and data imputation. Furthermore, we implement our solution and develop an application that allows users to implement the designed DW and carry out the merging and imputation processes.

The manuscript is organized as follows.

- In Chapter II, we propose a solution for automatic DW design and implementation from tabular data. The solution is composed of measure detection for the construction of facts and dimension detection for the construction of dimensions. Regarding measure detection, we consider numerical columns as candidate measures and propose a machine learning-based approach by defining general, statistical and inter-column features extracted from numerical attributes. Regarding dimension detection, we first propose an algorithm to create hierarchies by detecting functional dependencies. We then propose some syntactic and semantic rules to identify dimension attributes as parameters or weak attributes. We carry out experiments to validate our solution. Measure detection is validated by comparing the effectiveness of different machine learning algorithms with baseline approaches and by analysing the feature category effectiveness, model generality and feature importance. Dimension detection is validated by the efficiency and the effectiveness for the detected dimensions at dimension aspect, dimension attribute aspect and relationship aspects.

- In Chapter III, we propose a process for merging two DWs modelled as star schemas at both schema and instance levels. Our process is composed of level merging, hierarchy merging, dimension merging and star merging. The process considers different multidimensional components and generates a merged DW modelled as a star or constellation schema in different cases. We carry out experiments with the TPC-H benchmark's data to validate the process in both star and constellation schema generation cases. We verify the merged schema and instance results to validate the correct merging.

- In Chapter IV, we propose an approach named Hie-OLAPKNN for DW dimensional data imputation. The approach is hybrid and combines a hierarchical imputation (Hie) and a k-nearest neighbors-based imputation (OLAPKNN). Hierarchical imputation is carried out first. It is a reliable approach based on actual functional dependencies among intra- and inter-dimensional hierarchy levels. OLAPKNN is then carried out to replace the remaining missing data. Since OLAPKNN replaces missing data by nearest neighbors, we define a specific distance metric for dimension instances by considering dimensions' structure. Moreover, the OLAPKNN algorithm takes hierarchy dependency constraints into account. We conduct experiments to compare Hie-OLAPKNN with other approaches from the literature by verifying the

effectiveness, efficiency and respect of hierarchy strictness.

- In Chapter V, we implement a complete solution by integrating the approaches of automatic DW design and implementation, automatic DW merging and data imputation. We first present the functional and technical architecture of the application. We then explain the different functionalities with the presentation of the front-end and back-end.

Chapter II

Automatic Data Warehousing

Contents

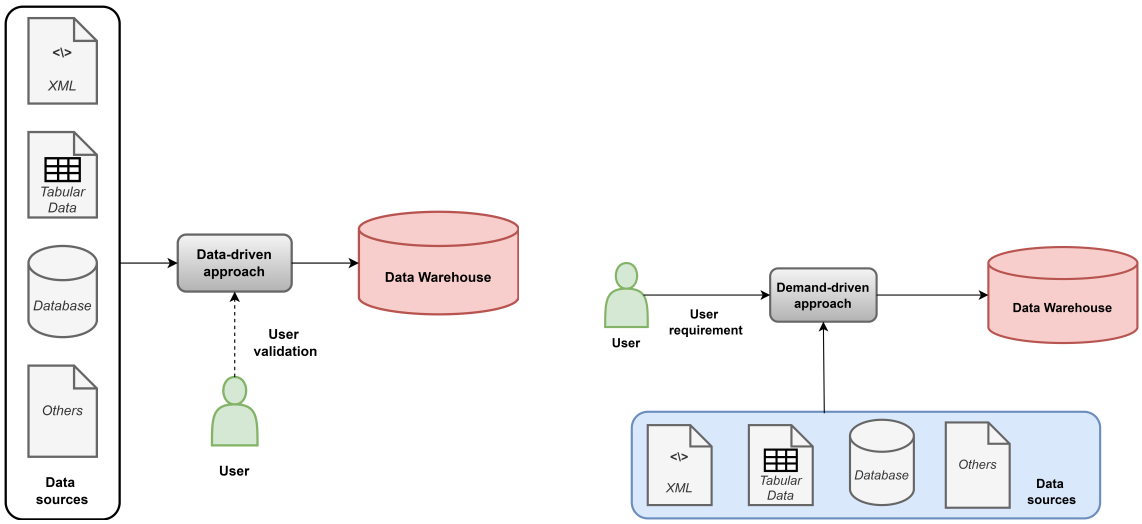
1 Introduction

1.1 Context

Data Warehouse is the core of the BI system which models the data by a multidimensional way allowing decision makers to analyse data by On-Line Analytical Processing (OLAP) (Golfarelli and Rizzi, 2009). With the development of information systems and the availability of numerous open datasets, various data become much more accessible to small enterprises, organizations and even individuals, who have data analysis needs by BI tools to help them take decisions. However, the DW design is normally carried out manually and requires experts with BI experience (Romero and Abelló, 2010). So the DW design process is typically costly and time-consuming. However, these users do not have enough budget or BI experts. Thus, it is difficult for them to take advantage of BI. Moreover, they may not necessarily know or anticipate precise requirements. They may also have some requirements but do not know how to express them in a proper way which help for the DW design. Therefore, it is necessary to automate the DW design process to make the non-expert users to carry out analysis with warehoused data.

DW design is an important part of information system design (Céret et al., 2013). There are different approaches of DW design (Romero and Abelló, 2009), which can be classified into data-driven approaches and demand-driven approaches as shown in Fig. II.1. In the data-driven approaches (Fig. II.1a), the DW schema is generated from the data sources by analysing the data and schema. The user may also get involved in the processes by validating the results. The data-driven DW design processes are mostly automatic or semi-automatic solutions. Meanwhile there are demand-driven approaches (Fig. II.1b) which start from user requirements and map the data sources to generate the schema satisfying these requirements manually or automatically. Moreover, there are hybrid approaches taking both user requirements and the data source into account. Since there are various DW design difficulties for our target user as we analysed, our work focuses on the data-driven approaches by proposing automatically DW schema and ask the user's participation for the validation.

Most of the data-driven approaches focus on data sources with an explicit schema (Romero and Abelló, 2009), e.g. relational data with Entity-Relationship (ER) schema, XML data with Document Type Definitions (DTDs), etc. Nevertheless, tabular data such as spreadsheet data and Comma Separated Value (CSV) files are very common in enterprises, and even more in the open data world. We thus focus on tabular data whose schemas are not available. Thus we have to detect the different multidimensional components based on the data instances which may arise several challenges. A DW is composed of fact(s) and dimensions which contain particular multidimensional elements. In the fact(s), there are measures; in the dimensions there are hierarchies and different types of attributes including parameters and weak attributes. Thus we have to detect these different multidimensional components.



(a) Data-driven (b) Demand-driven

Figure II.1: Two types of DW design processes

In addition, tabular data may bear quite simple or very complex structures (Adelfio and Samet, 2013). Simple structures consist of one header row followed by rows containing data values. Headers label the data rows below, while data rows contain tuples akin to relational database tuples. Most CSV files bear a simple structure, while spreadsheet files and HTML tables can be more complex, e.g., cross tables (Lautert et al., 2013). Such tables contain two or several dimensions, and may also contain several dimension levels. Moreover, there also exists other complex structures such as concise tables, nested tables, multivalued tables and split tables (Lautert et al., 2013). For tabular data of complex structure, the most important task is to identify the table structure to extract DW elements or transform them into simple structure. These tasks can be solved by some existing algorithms (Chen and Cafarella, 2013; Du et al., 2021; Koci et al., 2016; Wang et al., 2021). Thus, in the following, we focus on the automatic DW design for tabular data of simple structure.

We then discuss the challenges for the detection of the different multidimensional components from tabular data of simple structure.

1.2 Challenges of Measure Detection

In simple-structured tabular data without schema or metadata, DW elements cannot be directly extracted as the data do not bear a particular layout. Measures are usually numerical data, but numerical columns are not necessarily measures, since there also exists descriptive numerical attributes. Moreover, a column with the same semantic may be treated differently in different contexts. For example, the population of a country may be a measure if the analysis subject is the country information. But if the country is a hierarchical level in a geographical dimension, population is just a descriptive, so-called weak attribute, and not a measure. Thence, it is also difficult to detect measures based

on the semantics of the column

1.3 Challenges of Dimension Detection

To detect dimensions, we should identify the hierarchical relationships between attributes to create dimension hierarchies. Moreover, we have also to decide which attributes are parameters and which ones are weak attributes.

For tabular data of simple structure, there is no layout particularity. There is no schema where we can get the cardinalities neither. We thus have to derive the hierarchical relationships by discovering the functional dependencies among the attributes. For the distinction of parameters and weak attributes, a parameter can be regarded as the identifier of its level. Thus the weak attributes are functionally determined by their parameters. However, in the functional dependency relationships, we can not simply tell whether an attribute determined by another attribute is a parameter of a level or a weak attribute of its determinant attribute. Furthermore, sometimes several attributes of a same level may all be candidates of parameter, we have to choose the most appropriate one.

1.4 Our Process Overview

Facing to these challenges, we propose a process to resolve them. The overview of our process is shown in Fig. II.3.

For tabular data of complex structure, existing algorithms (Chen and Cafarella, 2013; Du et al., 2021; Koci et al., 2016; Wang et al., 2021) can be used for the identification of table structure. For cross tables, measures can be extracted from data region. Headers can be viewed as DW dimensions, and the different levels of hierarchical headers form hierarchies. The other types of complex structures can be converted into simple structures.

Since the DW design for complex structure tabular data can be solved by existing approaches, we focus on that of simple structure. We propose an automatic DW design process for tabular data of simple structure as shown in the red-framed part. To solve the challenges of measure detection, we propose a machine learning-based measure detection approach. Then to solve the challenges of dimension detection, we propose a functional dependency-based hierarchy detection and a rule-based approach for distinction of parameters and weak attributes.

1.5 Outline

The remainder of this chapter is organized as follows. In Section 3, we review and compare the related works about data-driven automatic DW design. In Section 4, we detail and discuss the measure detection process and the machine learning features we propose. In Section 5, we explain how to build hierarchies from functional dependency trees and how

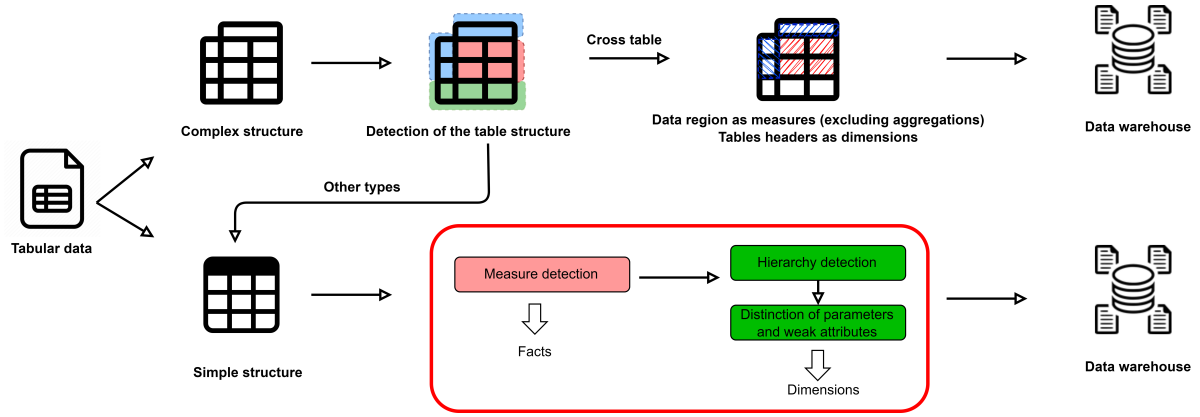


Figure II.2: Process overview

to decide whether an attribute is a parameter or a weak attribute. In Section 6 and Section 7, we present and interpret our experimental results respectively for measure detection and dimension detection. Finally, in Section 8, we conclude this chapter.

2 Preliminary

We introduce in this section, basic concepts of a DW (Ravat et al., 2008a) that we use throughout this manuscript.

Definition 2.1 (Data warehouse). *A **data warehouse**, denoted by DW, is defined as $(N^{DW}, F^{DW}, D^{DW}, Star^{DW})$, where*

- *N^{DW} is the data warehouse's name,*
- *$F^{DW} = \{F_1, ..., F_m\}$ is a set of facts,*
- *$D^{DW} = \{D_1, ..., D_n\}$ is a non-empty set of dimensions,*
- *$Star^{DW} : F^{DW} \rightarrow 2^{D^{DW}}$ is a mapping associating each fact to its linked dimensions. The notation 2^X denotes the powerset of the set X.*

A DW can be modelled by a star or a constellation schema. In a star schema, there is a single fact connected with different dimensions, i.e. $|F^{DW}| > 1$. A constellation schema consists of more than one fact which share one or several common dimensions, i.e. $|F^{DW}| = 1$.

A dimension models an analysis axis and is composed of attributes.

Definition 2.2 (Dimension). *A **dimension**, denoted by $D_c \in D^{DW}$, is defined as $(N^{D_c}, A^{D_c}, H^{D_c}, I^{D_c})$, where*

- *N^{D_c} is the dimension's name,*

- $A^{D_c} = \{a_1^{D_c}, ..., a_u^{D_c}\} \cup \{id^{D_c}\}$ *is a non-empty set of attributes, where* id^{D_c} *represents the dimension's identifier, which is also the parameter of the lowest level and called the root parameter.*
- $H^{D_c} = \{H_1^{D_c}, ..., H_v^{D_c}\}$ *is a non-empty set of hierarchies,*
- $I^{D_c} = \{i_1^{D_c}, ..., i_q^{D_c}\}$ *is a set of dimension instances. The value of an attribute* $a_u^{D_c}$ *of the instance* $i_q^{D_c}$ *is denoted as* $i_q^{D_c}.a_u^{D_c}$.

A hierarchy represents a particular vision (perspective) of a dimension. Each attribute represents one data granularity according to which measures could be analysed.

Definition 2.3 (Hierarchy)**.** *A* ***hierarchy*** *of a dimension* D_c*, denoted by* $H_e \in H^{D_c}$*, is defined as* $(N^{H_e}, Param^{H_e}, Weak^{H_e})$*, where*

- N^{H_e} *is the hierarchy's name,*
- $Param^{H_e} =< id^D, p_2^{H_e}, ..., p_v^{H_e} >$ *is a non-empty ordered set of dimension attributes, called* ***parameters****, which set granularity levels along the dimensions:* $\forall k \in [1...v], p_k^{H_e} \in A^{D_c}$*. The roll up relationship between two parameters can be denoted by* $p_1^{H_e} \preceq_{H_e} p_2^{H_e}$ *for the case where* $p_1^{H_e}$ *roll up to* $p_2^{H_e}$ *in* H_e*. For* $Param^{H_e}$*, we have* $id^D \preceq_{H_e} p_1^{H_e}, p_1^{H_e} \preceq_{H_e} p_2^{H_e}, ..., p_{v-1}^{H_e} \preceq_{H_e} p_v^{H_e}$.
- $Weak^{H_e} = Param^{H_e} \rightarrow 2^{(A^{D_c} - Param^{H_e})}$ *is a mapping possibly associating each parameter with one or several* ***weak attributes****, which are also dimension attributes providing additional information.* $Weak^{H_e}[p_x^{H_e}] = \{w_1^{p_x^{H_e}} ..., w_y^{p_x^{H_e}}\}$ *is the weak attribute set for parameter* $p_x^{H_e}$*. All parameters and weak attributes of* H_e *constitute the hierarchy attributes of* H_e*, denoted by* $A^{H_e} = Param^{H_e} \cup (\bigcup_{p_v^{H_e} \in Param^{H_e}} Weak^{H_e}[p_v^{H_e}])$.

There exists different types of hierarchy, but the most basic and common one is the strict hierarchy (Malinowski and Zimányi, 2004) where a value at a hierarchy's lower-granularity level belongs to only one higher-granularity level value (Trujillo et al., 2001). Thus in this manuscript, we only consider the case of the strict hierarchy.

A fact reflects information that has to be analysed according to dimensions and is modelled through one or several indicators called measures.

Definition 2.4 (Fact)**.** *A* ***fact****, denoted by* $F_g \in F^{DW}$*, is defined as* $(N^{F_g}, M^{F_g}, I^{F_g}, IStar^{F_g})$*, where*

- N^{F_g} *is the fact's name,*
- $M^{F_g} = \{m_1^{F_g}, ..., m_w^{F_g}\}$ *is a set of measures.*
- $I^{F_g} = \{i_1^{F_g}, ..., i_q^{F_g}\}$ *is a set of fact instances. The value of a measure* $m_w^{F_g}$ *of the instance* $i_q^{F_g}$ *is denoted as* $i_q^{F_g}.m_w^{F_g}$.

- *$IStar^{F_g} : I^{F_g} \rightarrow \mathcal{D}^{F_g}$ is a function associating each fact instances to their linked dimension instances, where $\mathcal{D}^{F_g}$ is the cartesian product over sets of dimension instances, which is defined as $\mathcal{D}^{F_g} = \prod_{D_k \in Star^{DW}(F_g)} I^{D_k}$.*

3 Related Work

In this section, we present the different data-driven automatic DW creation approaches in chronological order. We also analyse these approaches by comparing them in different aspects including the input source, the pre-processing, the detection of different DW elements, the DW implementation and user intervention.

3.1 Approaches

3.1.1 Boehnlein and Ulbrich-vom Ende (1999)

The authors propose an approach to derive a multidimensional DW schema from a Structured Entity Relationship Model (SERM) that is an extension of ER which allows designing extensive data models, visualizing the dependency order between data objects and avoiding inconsistencies and unnecessary relationships. This approach consists of three stages as follows.

1. **Identification of Business Measures** Measures are determined by business goals. This stage requires business knowledge about the company's services. Then by analysing how services can be evaluated for the business goals, adequate measures can be defined.

2. **Identification of Dimensions and Hierarchies** To identify potential dimensions and hierarchies, the authors propose to enclose the data objects by the dependencies in the SERM. The starting point is the data objects assigned to the chosen measures. Data objects are then connected to form different dimensions. Data objects with one-to-many cardinalities form different hierarchies.

3. **Identification of Integrity Constraints Along Dimension Hierarchies** In this stage, the authors transform the identified multidimensional structure into a star schema. They include primary keys of the dimensions in the fact tables. They also propose the alternative to create a snowflake schema by the normalization of dimension tables.

3.1.2 Moody and Kortink (2000)

This paper depicts an approach to create a multidimensional schema from an **Entity Relationship (ER) schema**. The approach includes four following steps.

1. **Entity Classification** In this step, the authors propose to classify entities in the ER schema into three categories:

(a) *Transaction Entities* contain business events such as orders, payments and bookings. This category of entities contain measures that are used to construct fact tables.

(b) *Component Entities* define the components and details of a business transaction. A component entity is directly connected to a transaction entity via a one-to-many relationship. They are entities that help for the construction of dimension tables.

(c) *Classification Entities* are connected to component entities via one-to-many relationships. A classification entity is functionally dependent on a component entity and is useful for constructing dimension tables, especially dimension hierarchies.

To remove ambiguities in case an entity can be classified into multiple classes, the authors also define a precedence rule. The transaction entity has the highest precedence and the component entity has the lowest.

2. **Hierarchy Identification** Hierarchies are identified by the sequence of entities joining one-to-many relationships. The authors propose to create maximal hierarchies that cannot be extended upwards with other entities.

3. **Dimension Model Production** Knowing the identified entity categories and hierarchies, the authors propose various dimensional models including flat, terraced, star, snowflake and star cluster schemas. The generation of a star schema starts with fact table for each transaction entity whose keys are linked to component entities. A dimension table is created for each component entity. The related classification entities are also included in the dimension to form hierarchies.

4. **Evaluation and Refinement** The authors argue that DW modelling is an iterative process. Thus, other operations may be needed after the generation of the first schema. These operations include

   (a) combining fact tables with the same primary keys;

   (b) combining related dimension tables into a single dimension to avoid a large number of dimension tables;

   (c) dealing with many-to-many relationships to avoid breaks in the hierarchical chain;

   (d) converting sub/supertype relationships into dimension hierarchies.

3.1.3 Golfarelli et al. (2001)

The authors propose an approach for building DW conceptual schema starting from an Extensible Markup Language (XML) source with a Document Type Definition (DTD).

They focus on the DTD, modelling relationships by sub-elements. The output is a star schema. The approach is composed of four following steps.

1. **DTD Simplification** This step simplifies some details in the DTD, such as transforming a nested definition into a flat representation, grouping sub-elements with same name and transforming many unary operators into single unary operators, e.g. transforming all "+" operators into "*" operators.

2. **DTD Graph Creation** In this step, a DTD graph representing the DTD structure is created by methods from the literature such as the CPI algorithm (Lee and Chu, 2000).

3. **Fact Definition** The user chooses one or many vertices in the DTD graph as measures, so that each one of them becomes the root of a fact schema.

4. **Attribute Tree Creation** Based on the one-to-many relationships between the sub-elements, an attribute tree is created. It can then be transformed into a star schema.

3.1.4 Phipps and Davis. (2002)

In this paper, the authors propose an automatic DW design approach whose input is an ER schema. The output of the approach is a Multidimensional Entity-Relationship Model (MERM). The approach is composed of following steps.

1. **Fact Node Creation** The authors claim that numerical fields are more likely to be measures. Thus the more numerical fields an entity contains, the more likely it is to be a fact. Therefore, in this step, they order the entities with numerical fields in descending order. Then, they create a fact node for each entity and create a MERM for each fact node. We thus get a list of candidate schemas.

2. **Fact Attribute Creation** In this step, the fact node of each candidate MER schema is added to the numerical fields of the original entity as the fact's attributes.

3. **Date Dimensions Creation** The date or time fields in each selected entity help create a date dimension and its levels. The date granularity is decided by the user.

4. **Other Dimension Creation** If there are remaining fields in a selected entity, they are normally text fields. A dimension and a corresponding hierarchy level node are created for each remaining field.

5. **Add Hierarchy Levels** In this step, the authors recursively include the many side of one-to-many relationships to create hierarchies. Each candidate schema is completed after this step.

6. **Candidate Schema Selection and Refinement** The final validation of the schema involves the user. Candidate schemas are evaluated by queries to decide

which schemas best meet users' needs. Selected schemas are also refined according to users' requirements. Refinements includes

(a) verifying whether the identified measures are actual measures;

(b) determining the granularity of date information;

(c) determining whether there are calculated measures;

(d) determining whether there are schemas that can be merged;

(e) verifying whether there are unnecessary fields that can be eliminated;

(f) verifying whether there are required data not existing in the original OLTP database.

3.1.5 Vrdoljak et al. (2003)

This paper describes a semi-automatic process for DW design from XML sources modelled by XML schemas. It follows a similar process as (Golfarelli et al., 2001), but with a different XML model. It includes the following steps.

1. **XML Schema Preprocessing** The XML schema may be sometimes complex and bear redundancy, so this step simplifies the schema as in (Golfarelli et al., 2001).

2. **Schema Graph Creation and Transformation** In this step, a graph is created based on the XML schema. Two transformations are carried out. First, functional dependencies are explicitly expressed by key attributes. Second, vertices not storing any value are eliminated.

3. **Fact Selection** Facts are chosen among the vertices and the arcs representing a many-to-many relationship by the user.

4. **Dependency Graph Creation** For each fact, a dependency graph whose root is the fact is built based on the schema graph. Vertices are inserted into the dependency graph by verifying the one-to-many cardinalities. When cardinalities are not provided, XQueries are performed to look for to-one relationships. Many-to-many relationships may be chosen with respect to users' interest. The dependency graph helps building hierarchies.

5. **Logical Schema Creation** With measures and facts being already chosen, dimensions and hierarchies are derived by the dependency graph.

3.1.6 Jensen et al. (2004)

In this paper, the authors present an approach aiming to discover multidimensional snowflake schemas from relational databases. The approach includes three following steps.

1. **Metadata Collection** A metadata model is firstly proposed where there are metadata about tables, including attribute information, keys, cardinalities, etc. For each attribute, there is also a metadata "role" being "key", "measure" or "descriptive" determined by a Bayesian network taking the collected metadata as inputs.

2. **Database Structure Discovery** In this step, the authors discover candidate keys and foreign keys by detecting functional dependencies and inclusion dependencies, with the help of metadata. These keys are applied for the construction of dimensions in the snowflake schema.

3. **Multidimensional Schema Construction** The fact table is identified in the previous step before the detection of inclusion dependencies. It is a semi-automatic process requiring the user's participation. For the construction of dimensions, inclusion dependencies can form different connected graphs. If there is an inclusion dependency that connects an attribute of the fact table and another attribute in a connected graph, then this connected graph may be a dimension. This attribute on the connected graph is the root parameter of the dimension. For the construction of hierarchies, the authors sort the attributes in the dimension by distinct descending order. Then, the authors verify roll-up relationships via SQL queries to create the hierarchies.

3.1.7 I.-Y.Song et al. (2007)

A semi-automatic method named SAMSTAR is proposed in this paper, which generates star schema from ER schema. SAMSTAR can be summarized by the following steps.

1. **ER schema to binary ER schema Conversion** In this step, the authors propose to split the ER schema into a binary ER schema, by splitting ternary relationships into three binary ones and splitting many-to-many relationships into two one-to-many relationships with a new intersection entity.

2. **Facts CTV Creation** The Connection Topology Value (CTV) is proposed by the authors, which is a composite function of the topology value of direct and indirect many-to-one relationships. The CTV is calculated for each entity. A threshold is set and the entities whose CTV are higher than it are identified as candidate fact tables.

3. **Dimension Creation** Dimensions are created by identifying the entities having direct and indirect many-to-one relationships with a fact entity. Synonyms in the Wordnet and Annotated Dimensional Design Pattern (A_DDP) are also used to extend the dimension list.

4. **Generated Schema Post-processing** Finally, they post-process the generated schema by requiring the users' intervention. The user choose the final dimension entities based on their requirement. The user also checks redundant time dimen-

sions, possibly merge related dimensions and rename tables. The final star schema is then generated.

3.1.8 Romero and Abelló (2007)

The authors propose a semi-automatic multidimensional design approach from OWL ontology representing heterogeneous data sources, and express multidimensional patterns with Description Logic (DL).

1. **Fact Creation** The authors consider that a concept is more likely to be a fact if it is related to many potential dimensions and measures. So, they first discover potential dimensions and measures. Dimensions are discovered by deriving functional dependencies from the ontology and finding many-to-one relationships. Measures are pointed out by finding the numerical concepts related to one-to-one relationships. Facts can be found. The user chooses the facts according to subjects of interest.

2. **Potential Bases Discovery** The authors define a minimal set of levels functionally determining a fact as a base. This step aims to point out sets of concepts that are likely to be bases of each identified fact. So they search for the concepts being able to identify all instances of a fact to be potential bases. The user finally chooses the bases making sense to her/him. The concepts in the bases form the identifiers of the dimensions.

3. **Dimension Hierarchy Creation** In this step, the authors look for the to-one roll-up relationships and create a directed graph following the paths of these relationships to build the hierarchies.

3.1.9 Usman et al. (2010, 2013)

The authors propose an automatic method to generate a star schema from a tabular data. It is based on data mining techniques and contains two layers.

1. **Data Mining Layer** This is a pre-processing layer. The authors use the hierarchical agglomerative clustering to generate clustered data with their hierarchical relationships.

2. **Automatic Schema Generation Layer** In this layer, the authors identify dimensions and facts. Numerical data form the fact table and nominal data form dimensional tables. The hierarchical relationships obtained in the previous layer are employed to build the hierarchies.

3.1.10 Ouaret et al. (2014)

This paper describes a rule-based approach generating a star schema from an XML schema. The idea is to transform the XML schema into a UML diagram and then derive

a star schema. The approach is composed of the following steps:

1. **UML Class Diagram Generation** In this first step, the authors transform the XML schema into a UML class diagram by pre-defined rules.
2. **UML Class Diagram Reduction** They reduce the generated UML diagram by removing some redundant, isolated, trivial classes and merge one-to-one relationships.
3. **Star Schema Creation** Based on the UML schema, they define rules to construct different multidimensional elements including
   - **measures**: numerical no-key attribute are potential measures,
   - **facts**: classes with a large number of numerical attributes are potential facts,
   - **dimensions**: the classes having many-to-one and one-to-one relationships with facts are considered as dimensions.

A tool is developed allowing users to generate an XML multidimensional schema from an XML schema and create an XML DW from the XML data sources.

3.1.11 Sautot et al. (2015)

This paper introduces an automatic hierarchy design method for OLAP schema from ecological database based on data mining techniques. The paper focuses on the context of ecological data, where measures and dimensions are normally clearly identified. Their method for detecting hierarchies can be summarized as follows.

1. **Data and Metadata Collection** The authors collect the data and metadata that are to be used for the creation of hierarchies from the database. Then, the data type of each attribute is identified, which is necessary for the clustering algorithm.
2. **Hierarchical Clustering** They propose to use the hierarchical agglomerative clustering with Gower index as a distance metric to cluster the data.
3. **Dimension Hierarchy Construction** They use the obtained hierarchical relationships to construct dimension hierarchies.

3.1.12 Elamin et al. (2017)

This paper proposes a heuristic-based approach for generating a star schema from an ER model. The authors define several heuristic rules for different parts of the process.

1. **Database Schema Extraction** In this phase, they extract table names, column types, keys, etc.
2. **Schema Reverse Engineering** Several rules are proposed to identify each table

as an entity, a relationship or a weak entity that contains partial keys.

3. **Multidimensional Schema Generation** Then they define rules for the identification of different multidimensional components.
   - Facts can be discovered from relationship tables and weak entity tables.
   - Measures are identified from numerical non-key attributes in fact tables.
   - Dimensions are identified from the tables referred by foreign keys in a fact table. Date and time attributes are also transformed into dimensions.
   - Hierarchies are created by the foreign key references between tables. Parameters are assigned to tables' primary keys. The rest of the attributes are weak attributes.

3.1.13 Sanprasit et al. (2021)

In this paper, an automatic approach to generate a star schema from semi-structured data (CSV files and spreadsheets) is proposed using semantic techniques. The approach contains steps as follows.

1. **Attribute Metadata Extraction and Analysis**
   (a) The authors propose to use an arithmetic data encoding technique to infer column names based on the training dataset. Wordnet is used to handle heterogeneous terminologies.
   (b) Then, they infer the attribute data types by referencing to a data type ontology.
   (c) Measures are identified through constraints from the domain ontology.
2. **Star Schema Construction**
   (a) Attributes that can be semantically classified into a same domain ontology class construct a dimension.
   (b) Hierarchical relationships in the domain ontology help to build up dimension hierarchies.
   (c) The fact table is created based on measures.
   (d) Surrogate keys are created to associate the dimension and fact tables.

3.2 Comparative Analysis

Table II.1 shows a comparison of the related works accounted for in the previous sections. We provide an analysis concerning the input source and schema, pre-processing, fact

generation, dimension generation, DW implementation and user intervention.

3.2.1 Inputs

Approaches' input can be mainly classified into structured data with schema, semi-structured data with schema and semi-structured data without schema.

- **Structured data with schema** We can observe that many approaches (7 out of 12) treat structured data (database data) with schema (Boehnlein and Ulbrich-vom Ende, 1999; Moody and Kortink, 2000; Phipps and Davis., 2002; Jensen et al., 2004; I.-Y.Song et al., 2007; Sautot et al., 2015; Elamin et al., 2017).
- **Semi-structured data with schema** There are 4 approaches taking semi-structured data as inputs. Some of them take semi-structured data with schema such as XML files with DTD (Golfarelli et al., 2001) or XML schema (Vrdoljak et al., 2003; Ouaret et al., 2014) or ontology with OWL (Romero and Abelló, 2007).
- **Semi-structured data without schema** It is a challenge to deal with semi-structured flat data since they do not have explicit schema. This is also the data type on which we focus. However there are only 2 approaches dealing with flat data without schema (Usman et al., 2010, 2013; Sanprasit et al., 2021).

3.2.2 Preprocessing

All approaches addressing structured and semi-structured data sources with schema include preprocessing at the schema level. The approaches whose inputs are semi-structured data without schema conduct preprocessing at the instance level.

- **Schema-level preprocessing** Some approaches (5 out of 12) transform the original schema or create new schemas (Boehnlein and Ulbrich-vom Ende, 1999; I.-Y.Song et al., 2007; Golfarelli et al., 2001; Vrdoljak et al., 2003; Ouaret et al., 2014). Other approaches perform the classification of schema elements (Moody and Kortink, 2000; Elamin et al., 2017) or the collection of schema information (Jensen et al., 2004; Sautot et al., 2015). The other preprocessings include creating candidate star schema from the identified facts (Phipps and Davis., 2002) and describing the multidimensional patterns by DL (Romero and Abelló, 2007).
- **Instance-level Preprocessing** The instance level pre-processing for semi-structured data without schema includes carrying out hierarchical clustering on the data (Usman et al., 2010, 2013) and inferring column name from column data (Sanprasit et al., 2021). Such preprocessing can be regarded as extracting schema elements from data instances.

3.2.3 Fact Generation

Since the measures are key element of a fact, the main task in fact generation is to identify measures. A fact is predefined in Sautot et al. (2015). Some approaches consider the identification of measures and facts as the same process (Boehnlein and Ulbrich-vom Ende, 1999; Moody and Kortink, 2000; Phipps and Davis., 2002; I.-Y.Song et al., 2007; Jensen et al., 2004; Usman et al., 2010, 2013; Sanprasit et al., 2021), while the others distinguish measure and fact detection (Elamin et al., 2017; Golfarelli et al., 2001; Vrdoljak et al., 2003; Ouaret et al., 2014).

Facts are the analysis subjects and are strongly related to user requirements. Moreover, fact measures are normally numerical data. Thus, measure or fact detection is mainly based on user participation and numerical attributes.

- **User Participation-based generation** There are 6 approaches where measures and facts are selected manually by the user (Boehnlein and Ulbrich-vom Ende, 1999; Moody and Kortink, 2000; Golfarelli et al., 2001; Vrdoljak et al., 2003; Jensen et al., 2004; Romero and Abelló, 2007) and 3 approaches need the user's validation (I.-Y.Song et al., 2007; Ouaret et al., 2014; Elamin et al., 2017).

- **Numerical attribute-based generation** There are 5 approaches approaches that identify measures and facts based on numerical data (Phipps and Davis., 2002; Romero and Abelló, 2007; Usman et al., 2010, 2013; Ouaret et al., 2014).

- **Others** The other techniques for detecting together measures and facts include calculating CTV based on many-to-one relationships (I.-Y.Song et al., 2007) and exploiting a domain ontology (Sanprasit et al., 2021). The other technique for detecting facts is considering the number of foreign keys within the primary key (Elamin et al., 2017).

User participation decreases the degree of automation. However it can better satisfy user requirements. Numerical-based methods cannot guarantee that all numerical attributes are appropriate measures. The ontology-based approach is limited because it requires the appropriate domain ontology to get a good result.

Most of the approaches consider the generation of multi-facts, which means that they are able to generate star or constellation schemas. There are 3 approaches (Boehnlein and Ulbrich-vom Ende, 1999; Usman et al., 2010, 2013; Sanprasit et al., 2021) considering only the generation of one fact. Thus they are only able to generate star schemas. In these approaches, several schemas are generated in case of multiple analysis subjects, which increase the workload.

3.2.4 Dimension Generation

Dimension generation is realized by the following techniques:

- **One-to-many relationship-based generation** Dimensions are identified from one-to-many relationships associated with facts in 5 approaches out of 12 (Moody and Kortink, 2000; Golfarelli et al., 2001; I.-Y.Song et al., 2007; Romero and Abelló, 2007; Ouaret et al., 2014).
- **Dependency-based generation** 4 approaches are based on functional or inclusion dependencies to detect dimensions (Boehnlein and Ulbrich-vom Ende, 1999; Jensen et al., 2004; Elamin et al., 2017; Romero and Abelló, 2007).
- **Data Type-based generation** There are 2 approaches that consider textual and date attributes to create dimensions (Phipps and Davis., 2002; Usman et al., 2010, 2013).
- **Others** An ontology (Sanprasit et al., 2021) can also be applied for the creation of dimensions. Queries (Vrdoljak et al., 2003) can be employed for the validation of the created dimensions.

One-to-many relationship-based, dependency-based and query-based dimension detection rely on database constraints and are thus more reliable. Data type and ontology-based approaches do not verify these constraints and may thus detect wrong dimensions. Moreover, the ontology-based approach suffers from the problem of getting an appropriate domain ontology, as we mentioned for the measure and fact detection.

Hierarchy detection is a complex task where we must decide the hierarchical order of attributes. However, it is not considered or not explained in I.-Y.Song et al. (2007), Golfarelli et al. (2001) and Ouaret et al. (2014). In the approaches considering hierarchy detection, many approaches are based on one-to-many relationships. The others use hierarchical clustering or ontology for hierarchy detection.

- **One-to-many relationship-based generation** There are 7 approaches based on one-to-many relationships to construct hierarchies (Boehnlein and Ulbrich-vom Ende, 1999; Moody and Kortink, 2000; Phipps and Davis., 2002; Romero and Abelló, 2007). Some other techniques based on SQL queries (Jensen et al., 2004), foreign keys (Elamin et al., 2017) and dependency graph (Vrdoljak et al., 2003) can essentially be also regarded as variants of applying one-to-many relationships.
- **Hierarchical clustering-based** Dimension attributes can be clustered by hierarchical clustering to form different hierarchies. It is applied in 2 approaches for detecting hierarchies in a dimension (Sautot et al., 2015; Usman et al., 2010, 2013).
- **Ontology-based** Domain ontology can also be helpful (Sanprasit et al., 2021) for hierarchy detection. Dimension attributes can be related to concepts in a domain ontology. Hierarchies can be generated according to hierarchical relationships of the domain ontology.

One-to-many relationships exist between different hierarchy levels. This is why it is

the most applied criterion for hierarchy detection. However, the hierarchy clustering based approaches provide hierarchical relationships based on instance similarity. They can be semantically correct but may not match with the cardinality relationships between hierarchy levels. The ontology-based approach still has the same limit as mentioned above.

The distinction of parameters and weak attributes is only taken into account in Elamin et al. (2017) and Romero and Abelló (2007). In Elamin et al. (2017), the attributes which are originally primary keys in the ER schema are identified as parameters. In (Romero and Abelló, 2007), the distinction is decided manually by the user.

3.2.5 Data Warehouse Implementation

Most of the approaches do not consider DW implementation and focus only on multi-dimensional schema design. Only Ouaret et al. (2014), Usman et al. (2010, 2013) and Sanprasit et al. (2021) mention an implementation part where Ouaret et al. (2014) create a XML database. However, implementation details are not mentioned in the other two approaches.

3.2.6 User intervention

Only Ouaret et al. (2014)'s approach does not need the user's intervention and is claimed to be fully automatic. However, the authors plan to integrate user requirements in future works. All the other approaches are semi-automatic, which demand user intervention.

- **Measure/Fact Selection and Validation** Most approaches (7 out of 12) ask the user for measure/fact selection and validation (Boehnlein and Ulbrich-vom Ende, 1999; Moody and Kortink, 2000; Golfarelli et al., 2001; Vrdoljak et al., 2003; Jensen et al., 2004; Romero and Abelló, 2007; Elamin et al., 2017).
- **Schema or Schema Element Validation** In 4 approaches, the user is asked for validating or selecting the generated schema or schema elements (attributes, dimensions, etc.).
- **Others** Other user intervention operations include threshold definition (I.-Y.Song et al., 2007) and algorithm parameter tuning (Usman et al., 2010, 2013).

User intervention makes the approaches not fully automatic, yet it is important because it makes the identified schema conform to user requirement (Ravat et al., 2009).

Table II.1: Comparison of different automatic DW design approaches

	Input		Pre-processing	Fact			Dimension			DW Implemantation	User Intervention
	Source Type	**Data Source Schema**		**Measure Detection**	**Fact Detection**	**Multi-fact**	**Dimension Detection**	**Hierarchy Detection**	**Parameter/ Weak Attribute Distinction**		
Boehnlein and Ulbrich-vom Ende (1999)	Relational database	ER/SER (Structured Entity Relationship)	Transformation to SER	Manual		-	Association with measures	Cardinality	-	-	Measure/Fact selection
Moody and Kortink (2000)		ER (Entity Relationship)	Entity classification	Manual		✓	Component entities	Classification entities + Cardinality	-	-	Measure/Fact selection +Validation
Phipps and Davis. (2002)			Candidate star schema identification	Numerical attributes		✓	Date + Textual Attributes	Cardinaltiy	-	-	Validation
I.-Y.Song et al. (2007)			Transformation to binary ER	Manual/Connection Topology Value (CTV)		✓	Cardinality	-	-	-	CTV threshold + Validation
Jensen et al. (2004)		Not mentionned	Metadata collection	Manual		✓	Functional + Inclusion dependencies	SQL Queries	-	-	Measure/Fact selection
Elamin et al. (2017)		Logical schema	Identification of entity, relationship and weak entity tables	Numerical non-key attributes	Relationship tables + certain weak entity tables	✓	Fact foreign keys	Dimension foreign keys	Primary keys as parameters	-	Measure/Fact selection + Validation
Sautot et al. (2015)	Data warehouse	Star/ Constellation without hierarchy	Data type identification	Predefined				Hierarchical agglomerative clustering + Gower index	-	-	Hierarchy attribute selection
Golfarelli et al. (2001)	XML file	DTD (Document Type Definition)	XML simplification + DTD graph construction	Manual	Manual	✓	DTD graph cardinality	-	-	-	Measure/Fact selection
Vrdoljak et al. (2003)		XML schema	XML simplication + Schema graph construction	Manual	Manual	✓	Schema graph cardinilaty + Queries	Dependency graph	-	-	Measure/Fact selection + Dimension selection
Ouaret et al. (2014)			UML diagram generation	Numerical non-key attributes	Classes with a large number of numerical attributes	✓	Cardinality	-	-	XML database	-
Romero and Abelló (2007)	Ontology	OWL (Web Ontology Language)	Multidimensional pattern by DL (Description Logic)	Numerical concepts	Manual	✓	Functional dependency + Cardinality	Directed graph of to-one relationships	Manual	-	Measure/Fact selection
Usman et al. (2010),Usman et al. (2013)	Flat dataset	-	Hierarchical agglomerative clustering	Numerical attributes		-	Nominal attributes	Hierarchical agglomerative clustering	-	✓	Clustering parameter
Sanprasit et al. (2021)	CSV, spreadsheets)	-	Column name inference	Domain ontology		-	Same ontology class	Domain ontology	-	✓	Validation

3.3 Summary

There are few approaches addressing the semi-structured data without schema, since the detection of multidimensional elements can be challenging without a schema. The only two existing approaches have several limits. Usman et al. (2010, 2013) only consider data types for the generation of facts and dimensions. Hierarchies are generated by hierarchical clustering. The authors do not consider any database constraint, which may render the result unreliable. (Sanprasit et al., 2021) rely on a domain ontology, the DW element detection result depends on whether appropriate ontologies can be obtained. Parameter and weak attribute distinction as well as DW implementation are not widely discussed neither, which needs to attract more attention.

3.4 Automatic DW Design for Simple-structured Tabular Data

The proposed automatic DW design process from tabular data of simple structure is composed of measure detection and dimension detection where there are hierarchy detection and distinction of parameters and weak attributes. For the measures detection, we intend to find the numerical columns that conform to the characteristics of measures. We hypothesize that there are differences in terms of features between numerical data that are potential measures and those that are not. Therefore, in this chapter, we define specific features for numerical columns and propose a machine learning-based method to automatically detect measures. We then detect dimensions. For the detection of hierarchies, since there are functional dependencies among different levels of a hierarchy, we propose to detect functional dependencies and model them as trees to derive hierarchies. Then we define several syntactic and semantic rules based on characteristics of parameters and weak attributes to identify each attribute as one of them. Finally, the detected multidimensional components are linked to construct a multidimensional schema.

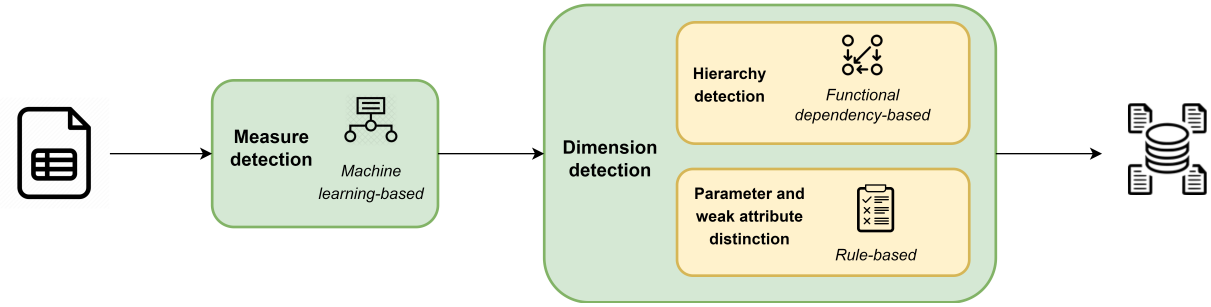


Figure II.3: Automatic DW design process for simple-structured tabular data

4 Measure Detection

In this manuscript, as mentioned that we focus on tabular data of simple structure. As analysed in Section 1, there are various challenges to detect measures from tabular data without schema. Machine learning algorithms can be employed to solve these challenges

since models can be trained according to some features of the numerical columns and capture the characteristics of measures. Therefore, we propose a machine learning-based process for measure detection.

4.1 Overview

Figure II.4 shows an overview of our measure detection process for simple structure tabular data. We first give a precise definition of measures as Definition 4.1 and tabular data of simple structure as Definition 4.2.

Since measures are numerical, we regard all numerical columns as candidates. So first, preprocessing the dataset is necessary for the selection of numerical columns. Second, to distinguish between measure and non-measure numerical columns, we extract features from numerical columns. Third, we use machine learning classifiers to estimate whether they are measures. Finally, users are asked to get involved for the validation of the proposed detected measures.

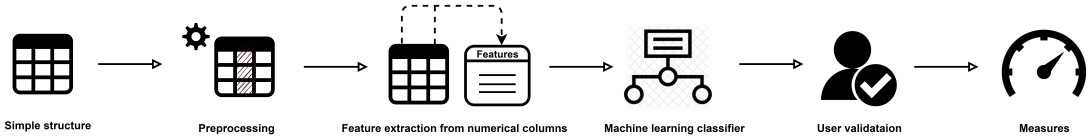


Figure II.4: Measure detection for tabular data

Definition 4.1 (Measure)**.** *We define a* ***measure*** *as a numerical and quantitative attribute of the analysis subject evaluating the activities of an organisation that can be aggregated with respect to dimensions. It can be additive, semi-additive or non-additive (Horner et al., 2004).*

Definition 4.2 (Simple structure)**.** *A tabular dataset of simple structure* $\boldsymbol{T}$ *is defined as* $\{\boldsymbol{C}, \boldsymbol{R}, \boldsymbol{A}, \boldsymbol{V}\}$*, where:*

- $\boldsymbol{C} = \{C_1, C_2, ...C_{n_c}\}$ *is a set of columns, where* n_c *is the number of columns in* $\boldsymbol{TS}$*. For a given column* $C_i \in \boldsymbol{C}$*, index* i *corresponds to the column's position in* $\boldsymbol{T}$*. The number of non-null values in column* C_i *is denoted as* $n_t(C_i)$*. The number of non-null distinct values is denoted as* $n_u(C_i)$*;*
- $\boldsymbol{R} = \{R_1, R_2, ..., R_{n_r}\}$ *is a set of rows (excluding the first, header row), where* n_r *is the number of non-header rows in* $\boldsymbol{TS}$*. For a given row* $R_j \in \boldsymbol{R}$*,* j *represents the index of the row corresponding to its position in* $\boldsymbol{TS}$*;*
- $\boldsymbol{A} = \{A_{C_1}, A_{C_2}, ..., A_{C_{n_c}}\}$ *is a set of attribute headers which is usually the first line. For a given attribute header* $A_{C_i} \in \boldsymbol{A}$*,* C_i *represents the column labeled by* A_{C_i}*;*
- $\boldsymbol{V}$ *is a matrix of cell values whose dimension is* $n_r \times n_c$*. For a given cell value* $V_{R_j,C_i} \in \boldsymbol{V}$*,* R_j *and* C_i *are the row and the column where the cell is located, respectively.*

Example 4.1. *In Fig. II.5, there is a CSV tabular dataset of simple structure. It contains several rows where R_1 is the first row. It contains several columns where C_5 is the fifth column whose header is A_{C_5} (City). The value of the first row and fifth column is V_{R_1,C_5} (Barcelone).*

IdCus	NameCus	Age	Email	City	Region	Country	MemLevel	IdProd	NameProd	Brand	CompSize	IdSubcat	Subcat	IdCat	Cat	Date	Price	Qty
1001	Louis	25	Louis@gm.com	Barcelone	Catalonia	Spain	2	AP233	Iphone13	Apple	Large	PH	Phone	TN	Technology	01/06/2022	819.99 €	1
1001	Louis	25	Louis@gm.com	Barcelone	Catalonia	Spain	2	AP233	Iphone13	Apple	Large	PH	Phone	TN	Technology	05/06/2022	825 €	n/a
1002	Gabriel	57	Gabriel@gm.com	Toulouse	Occitanie	France	1	SS112	Galaxy S10	Samsung	Large	PH	Phone	TN	Technology	03/06/2022	450 €	3
1003	Pierre	34	Pierre@hm.com	Paris	Ile-de-France	France	0	HF008	Computer table	Homefine	Medium	TB	Table	OF	Office	24/05/2022	125 €	3
1003	Pierre	34	Pierre@hm.com	Paris	Ile-de-France	France	0	SS467	Galaxy Book2	Samsung	Large	LP	Laptop	TN	Technology	03/06/2022	1 599.99 €	2
1004	Anna	45	Anna2@gm.com	Barcelone	Catalonia	Spain	1	SS467	Galaxy Book2	Samsung	Large	LP	Laptop	TN	Technology	29/05/2022	1 500 €	1
1005	Louis	22	Gab@hm.com	Castres	Occitanie	France	1	SH002	Ergonomic chair	SIHOO	Small	CH	Chair	OF	Office	01/06/2022	248.50 €	2
1006	Lucas	30	Luca@gm.com	Milan	Lombardy	Italy	2	NK112	Air Max	Nike	Large	SH	Shoe	CL	Clothing	06/06/2022	1 000 €	2

Figure II.5: Example of CSV table

4.2 Preprocessing

As candidate measures are numerical columns, we must firstly identify numerical columns. If all values of a column are numerical, we easily identify numerical columns. However, there are sometimes columns containing numerical values with their unit or columns containing both numerical and textual values used for replacing empty cells. Such mixed values must lead to numerical columns and require preprocessing.

Columns containing values with a unit are identified by verifying whether each cell bear the same structure, e.g., "text + number" or "number + text". We also verify whether the text of each column is the same or if it is categorical by using the algorithm proposed by Alobaid et al. (2019). Then, we extract numerical values via regular expressions and tag the column as numerical. Eventually, numerical columns containing empty values replaced by some text, e.g., "n/a", "null" or "unknown", are treated as numerical, with textual values being removed.

Example 4.2. *For the CSV tabular dataset in Fig. II.5, we preprocess the data. We can find that for the column of Price, the values are composed of "number + text" which are indeed the price with the unity. So it is considered as a numerical column and we remove the textual values of "$" for the feature extraction. Then by verifying the data, we can find in the column of Qty, there is textual value "n/a" representing missing value. It is thus also removed.*

4.3 Feature Extraction

After the preprocessing phase, we extract the numerical columns' features. When defining features, we analyse both general information and some statistical characteristics of numerical columns. Since tabular data of simple structure may exhibit specific column positional habits, we also consider column inter-relationships. Features are thus subdivided into three categories: general features, statistical features and inter-column features. For a given numerical column C_i, we define the following features.

4.3.1 General Features

These features reflect basic information on numerical columns. Such general features may help check whether a numerical column is likely to be quantitative and help evaluate business activities. General features follow.

- **Data type**: $type = \begin{cases} 1 & \text{if } type(C_i) = integer \\ 0 & \text{if } type(C_i) = float \end{cases}$, where $type(C_i)$ is C_i's data type.

Intuitively, float data are more likely to be quantitative and to allow evaluating activities. For example, temperature, salary and sales amount are float data can be considered as measures in most cases.

- **Positive/Negative/Zero value ratio**: $rpos = \frac{n_{pos}(C_i)}{n_t(C_i)}$, $rneg = \frac{n_{neg}(C_i)}{n_t(C_i)}$, $rzero = \frac{n_{zero}(C_i)}{n_t(C_i)}$, where $n_{pos}(C_i)$, $n_{neg}(C_i)$ and $n_{zero}(C_i)$ are the number of positive, negative and zero values in C_i, respectively, and $n_t(C_i)$ is the number of non-null values in C_i.

We get respectively the ratio of the positive, negative and zero values of the column. These features may help identifying both qualitative and quantitative columns. Qualitative data values, e.g., ID or zip code, are rarely negative or equal to zero. Thus, when there are many zero and negative values in a column, it is more likely to be a measure.

- **Unique value ratio**: $runique = \frac{n_u(C_i)}{n_t(C_i)}$.

The unique value ratio can reveal some typological information about a column. For example, in a descriptive dataset, IDs are always unique, so the unique value ratio is always equal to 1. In a dataset containing fact table data, keys and descriptive data may be repetitive, but equal measures should be quite scarce.

Example 4.3. *Given the numerical column* ***IdCus*** *of the CSV table from Fig. II.5, we can get* $runique = \frac{n_u(IdCus)}{n_t(IdCus)} = \frac{6}{8} = 0.75$. *There are 8 values in the columns and 6 distinct values* $\{1001, 1002, 1003, 1004, 1005, 1006\}$. *Given the numerical column* ***Price****, we get* $runique = \frac{n_u(Price)}{n_t(Price)} = \frac{8}{8} = 1$ *i.e. every value in the column is distinct.*

- **Same digital number**:
  $sdn = \begin{cases} 1 & \text{if } \forall i \in [1, n_t(C_i) - 1], nd_{R_j,C_i} = nd_{R_{j+1},C_i} \wedge type(C_i) = integer \\ 0 & \text{if } (\exists i \in [1, n_t(C_i) - 1], nd_{R_j,C_i} \neq nd_{R_{j+1},C_i} \wedge type(C_i) = integer) \\ & \quad \vee (type(C_i) = float) \end{cases}$, where nd_{R_j,C_i} is the number of digits in cell value V_{R_j,C_i}, which is calculated as $nd_{R_j,C_i} = floor(log_{10}^{V_{R_j,C_i}}) + 1$.

This feature tells whether all the values of an integer column have the same number

of digits. If it is the case, the column is likely to be a nominal number (Alobaid et al., 2019) representing the name or identifier of an element that cannot be a measure. For example, the French social security number always contains 15 digits.

Example 4.4. *Given the numerical column* ***Price****, $sdn = 0$ since it is a float column. Given the numerical column* ***IdCus****, $sdn = 1$ since it is a column of integers and each value has the same size of 4 digits.*

4.3.2 Statistical Features

Since candidate columns are numerical, statistical features must be considered, because they reflect the distribution of column values, which may be different for quantitative and qualitative attribute values. Statistical features follow.

- **Average/Minimum/Maximum/Median/Upper quartile/Lower quartile values**: $avg = avg(C_i)$, $min = min(C_i)$, $max = max(C_i)$, $median = median(C_i)$, $upquar = upquar(C_i)$ and $lowquar = lowquar(C_i)$ represent the average, minimum, maximum, median, upper quartile and lower quartile of C_i, respectively.

We consider these basic statistical metrics as features. In some specific columns, their values always vary in a certain range. Using these features can thus be helpful for capturing such statistical behaviours.

- **Coefficient of variation**: $coevar = \begin{cases} standdev(C_i) & \text{if } avg(C_i) = 0 \\ \dfrac{standdev(C_i)}{avg(C_i)} & \text{if } avg(C_i) \neq 0 \end{cases}$, where $standdev(C_i)$ is C_i's standard deviation.

The standard deviation can depict the amount of dispersion of a column values. Measures and descriptive attributes may have different degrees of dispersion, but by using the coefficient of variation, which is the ratio of the standard deviation by the average, we achieve a standardized degree of dispersion. For example, given two attributes "price of phone" and "temperature of city", the average price is much higher than that of temperature. A price variation of 10 is relatively much lower than that of temperature. Since the coefficient of variation is a ratio, when the average is equal to 0, it does not exist. Here, we define that when the average is 0, the feature is equal to the standard deviation of the column.

Example 4.5. *Given the numerical column* ***Price****, we can get $avg(Price) = 821.06$, and the standard deviation $standdev(Price) = 365.44$, we can thus get $coevar = \dfrac{365.44}{821.06} = 0.445$.*

- **Range ratio**: $rrange = \begin{cases} \dfrac{max - min}{n_u(C_i) - 1} & \text{if } n_u(C_i) > 1 \\ 0 & \text{if } n_u(C_i) = 1 \end{cases}$

The range ratio calculates the range of values with respect to the number of distinct values. It is useful to identify some ordinal data, even if they occur repetitively. For example, if we have student numbers ranging from 1000 to 2000 in a tabular dataset, but also courses and grades, a student number may occur many times while the range ratio is always 1 no matter the number of occurrences.

Example 4.6. *Given the numerical column* ***IdCus****, the number of non-null distinct values is* $n_u(IdCus) = 6$*. We can also get* $max(IdCus) = 1006$ *and* $min(IdCus) = 1001$*, so we obtain* $rrange = \frac{1006 - 1001}{6 - 1} = 1$*. Given the numerical column* ***Price****, the number of non-null distinct values is* $n_u(Price) = 8$*. We can also get* $max(Price) = 1599.99$ *and* $min(Price) = 125$*, so we obtain* $rrange = \frac{1599.99 - 125}{8 - 1} = 210.71$*.*

4.3.3 Inter-Column Features

Measures are aggregatable and are normally accompanied with attributes by which they are aggregated, as per the "group by" SQL clause. Typically, attributes linked to aggregations are located before measures in the source file. Therefore, we consider inter-column features that take inter-column relationships into account in the whole dataset.

- **Location ratio**: $rloc = \frac{i - 1}{n_c - 1}$.

In many tables, the identifier and some other basic information usually lie at beginning positions, while measures are usually in the latter positions. Thus, we also take column location into account. However, different datasets have different number of columns, so we must normalize the location feature as a ratio ranging between 0 and 1 by adding minus 1 in the calculation.

Example 4.7. *The numerical column* ***IdCus*** *is the first column of the table, so* $i = 1$*. There are* 19 *columns in the table, so we have* $n_c = 19$*. We thus obtain* $rloc = \frac{1 - 1}{19 - 1} = 0$*. In the same way, for the numerical column* ***MemLevel****, we have* $rloc = \frac{8 - 1}{19 - 1} = 0.39$ *and for* ***Qty****,* $rloc = \frac{19 - 1}{19 - 1} = 1$*. Thus, we can see that the location ratio for the first column is 0 and for the last is 1, and that the location ratio for a column in middle is between 0 and 1.*

- **Numerical column ratio**: $rnum = \frac{n_{num}}{n_c}$, where n_{num} is the number of numerical columns in the whole dataset.

The numerical column ratio is the ratio of numerical column number by total column number. This is a feature at the global level of the table, so the value of the feature is the same for the numerical columns in the same tabular data. We consider this feature

because when there are measures in tabular data, the ratio of the numerical columns may be increased.

- **Multiple functional dependencies**:
  $$several fds = \begin{cases} 1 & \text{if } \exists fd \in fdset, (fd.rhs = A_{C_i}) \wedge (size(fd.lhs) > 1) \\ 0 & \text{otherwise} \end{cases}$$
  where $fdset$ is the set of functional dependencies containing one attribute on the right-hand side, $fd.rhs$ is the right hand side attribute of functional dependency fd and $size(fd.lhs)$ is the number of attributes in the left hand side of fd.

In existing methods that exploit data sources with schemas, many-to-many relationships are usually employed for measure detection. In a DW, we usually analyse a fact with respect to different dimensions and measure values depend on dimensions' identifier. Thus, we consider whether there is a functional dependency with A_{C_i} depending on several attributes as a feature.

Example 4.8. *Given the numerical column* ***Price****, we have the functional dependency* $\{IdCus,\ IdProd, Date\} \rightarrow Price$ *where the right-hand side is* ***Price*** *and there are 3 attributes in the left-hand side. Therefore,* $several fds = 1$.

- **Numerical neighbor**:
  $$numn = \begin{cases} 1 & \text{if } (i = 1 \wedge type(C_{i+1}) \in num) \vee (i = n_c \wedge type(C_{i-1}) \in num) \\ & \quad \vee (i \neq 1 \wedge i \neq n_c \wedge type(C_{i+1}) \in num \wedge type(C_{i-1}) \in num) \\ 0.5 & \text{if } (i \neq 1 \wedge i \neq n_c \wedge type(C_{i+1}) \in num \wedge type(C_{i-1}) \notin num) \\ & \quad \vee (i \neq 1 \wedge i \neq n_c \wedge type(C_{i+1}) \notin num \wedge type(C_{i-1}) \in num) \\ 0 & \text{otherwise} \end{cases}$$
  where $num = \{integer, float\}$.

In a tabular dataset, the columns describing similar information are often clustered together. Measures are also likely to be located close together, meaning that there are numerical columns in neighboring positions. Thus, we define this feature to see whether neighbors of a column are also numerical. If so, the column is likely to be a measure.

Example 4.9. *The numerical column* ***MemLevel*** *has 2 neighbors and none of them is numerical, so* $numn = 0$. ***Price*** *has 2 neighbors and one of them is numerical (****Qty****), so* $numn = 0.5$. ***Qty*** *has one neighbor that is a numerical column, so we have* $numn = 1$.

4.4 Machine Learning Classification

To predict if a numerical column can be measure with the proposed features, we need to have a reliable model. Thus we should collect datasets, then we extract the proposed features and label the classes ("measure" or "not measure") for the numerical columns. The user can collect datasets from open data sites and also use her/his own datasets. Next, the feature values can be fed into machine learning classifiers to train a model. However, if the non-expert user is not able to collect datasets and label the numerical

columns, they can use the model that we obtain in the experiments for measure detection. Having the trained model, for given tabular data, feature values of the numerical columns can be extracted to detect measures.

Example 4.10. *We train a model by numerous datasets by random forest classifier. For the csv table in Fig. II.5, we extract the proposed features for the numerical columns including* ***IdCus****,* ***Age****,* ***MemLevel****,* ***Price*** *and* ***Qty*** *like shown in Fig. II.6. We can then use the trained model to classify each numerical columns and we finally detect the measures* ***Price*** *and* ***Qty****.*

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1	column	type	rpos	rneg	rzero	runique	sdn	avg	min	max	median	upquar	lowquar	coevar	rrange	rloc	rnum	severalfds	numn
2	IdCus	1	1	0	0	0.75	1	1003.125	1001	1006	1003	1001.75	1004.25	0.00180209	1	0	0.263157895	1	0
3	Age	1	1	0	0	0.75	1	34	22	57	32	25	36.75	0.346581355	7	0.111111111	0.263157895	1	0
4	MemLevel	1	0.75	0	0.25	0.375	1	1.125	0	2	1	0.75	2	0.741798187	1	0.388888889	0.263157895	1	0
5	Price(€)	0	1	0	0	1	0	821.06	125	1599.99	822.495	399.625	1125	0.659060412	210.7128571	0.944444444	0.263157895	1	0.5
6	Qty	1	1	0	0	0.375	1	1.875	1	3	2	1	2.25	0.445078912	1	1	0.263157895	1	1

Figure II.6: Example of extracted features

4.5 User Validation

The result of automatic measure detection cannot be 100% accurate. Thus, we have to ask the user to validate the detected measures. The validation includes two checks. First, we ask the user to check whether there are attributes that are detected as measures, but which are actually dimension attributes. Then, we ask the user to check whether there are attributes that are measures for users but that are not detected. After the validation, we can finally obtain all the measures.

5 Dimension Detection

The objective of this section is to detect DW dimensions. First, we discover the functional dependencies to detect the hierarchies. Then, we decide whether an attribute is a parameter or a weak attribute. Finally, we create dimensions based on the detected hierarchies.

5.1 Functional Dependency Detection

In a hierarchy, the values of a higher-granularity level is dependent of the values of its lower-granularity levels. In other words, there is functional dependency relationships (Ullman, 1983) between different levels of a hierarchy. We detect hierarchies with the help of the functional dependencies between non-measures columns of a table. Functional dependency is formally defined in Definition 5.1.

Definition 5.1 (Functional dependency)**.** *Let A be the attribute set of a dataset, $X \subseteq A$ be a set of attributes and $Y \in A$ be an attribute. X is said to functionally determine Y if and only if $\forall t_1, t_2 \in T, t_1[X] = t_2[X] \Rightarrow t_1[Y] = t_2[Y]$. This relationship between X*

and Y is called a ***functional dependency*** *(FD), denoted by $X \rightarrow Y$. X is called the* ***left-hand side*** *and Y is called the* ***right-hand side*** *of the functional dependency.*

However, the FDs of a simple structure table is not obvious. We should use the FD detection algorithm to discover the FDs. We choose to apply HyFD (Papenbrock and Naumann, 2016) because it achieves the best performance at both run time and memory consumption aspects and has the best row and column scalability against the seven most cited and important algorithms that are tested in (Papenbrock et al., 2015b).

For the creation of hierarchies, we do not need all detected FDs. Useful FDs should satisfy the following criteria for the hierarchy detection.

- In a dimension, FDs hold between two parameters or between a parameter and a corresponding weak attribute. Thus, we are only interested in the FDs whose left-hand side has one attribute.
- Let $X, Y, Z \in A$, according to Armstrong's axioms (Armstrong, 1974), if $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$ (transitivity). We must remove the transitivity, i.e., we retain only $X \rightarrow Y$ and $Y \rightarrow Z$ since $X \rightarrow Z$ can be inferred.
- We call $X, Y \in A$ equivalent attributes if $X \rightarrow Y$ and $Y \rightarrow X$, denoted by $X \leftrightarrow Y$. Given a FD containing one of the equivalent attributes, it also holds for the other one, i.e., $\exists Z \in A$, if $X \rightarrow Z$, then$Y \rightarrow Z$. In this case, we consider X and Y as a same attribute and treat them as one attribute X/Y i.e. we remove $Y \rightarrow Z$ and retain only $X/Y \rightarrow Z$.

Example 5.1. *In the CSV table from Fig. II.5, we remove the detected measures Price and Qty. Then, we detect the FDs of the remaining attributes. By launching the HyFD algorithm and filtering the FDs by the above criteria, we first obtain 4 equivalent attribute groups: $IdCus \leftrightarrow Age \leftrightarrow Email$, $IdProd \leftrightarrow NameProd$, $IdSubcat \leftrightarrow Subcat$ and $IdCat \leftrightarrow Cat$. We can then consider attributes of each equivalent attribute groups as one attribute: $IdCus/Age/Email$, $IdProd/NameProd$, $IdSubcat/Subcat$ and $IdCat/Cat$. Then we obtain the following FDs: $IdCus/Age/Email \rightarrow NameCus$, $IdCus/Age/Email \rightarrow MemLevel$, $IdCus/Age/Email \rightarrow City$, $City \rightarrow Region$, $Region \rightarrow Country$, $IdProd/NameProd \rightarrow Brand$, $Brand \rightarrow ComSize$, $IdProd/NameProd \rightarrow IdSubcat/Subcat$, $IdSubcat/Subcat \rightarrow IdCat/Cat$.*

To make sure that the FDs we discover conform to the actual dependency relationship of attributes in the real world, we hypothesize that there is enough data in terms of quantity and variety so as to represent real dependency relationships. Moreover, there should be no error in data, but if this is the case, we can detect approximate FDs (Liu et al., 2012).

5.2 Functional Dependency Tree

Dimension hierarchies can be represented by tree structures (Markl et al., 1999). Therefore, we can build functional dependency trees to construct different hierarchies and dimensions. The advantage of using functional dependency trees is that they have similar tree structure as hierarchies, so that we can easily detect hierarchies by finding the root-to-leaf paths. Functional dependency trees (Definition 5.2) are built by connecting functional dependencies.

Definition 5.2 (Functional dependency tree)**.** *A **functional dependency tree** (FD tree) is a directed tree denoted by $T = \{V_r, V_l, V_b, E\}$, where:*

- *V_r is a singleton set of the root node, with $|V_r| = 1$ and $v_r \in V_r$ is the root node of the tree,*
- *V_l is a set of all leaf nodes,*
- *V_b is a set of all branch nodes,*
- *$V = V_r \cup V_l \cup V_n$ is a set of all tree nodes containing all the attributes of FDs,*
- *E is a set of directed edges. $e_{12} = (v_1, v_2) \in E$ denotes an edge connecting two nodes v_1 and v_2 with from v_1 to v_2. It also means that the FD $v_1 \rightarrow v_2$ holds.*

Example 5.2 (Functional dependency tree)**.** *The detected FDs in Example 5.1 can form two functional dependency trees (Fig. II.7). For instance, $T_1 = \{V_{r1}, V_{l1}, V_{b1}, E_1\}$, where:*

- $V_{r1} = \{IdCus/Email/Age\}$,
- $V_{l1} = \{NameCus, MemLevel, Country\}$,
- $V_{b1} = \{City, Region\}$,
- $E_1 = \{(IdCus/Email/Age, NameCus), (IdCus/Email/Age, MemLevel), (IdCus/Email/Age, City), (City, Region), (Region, Country)\}$.

Then, we should extract hierarchies from the FD trees. The problem of finding a hierarchy is equivalent to the problem of finding a root-to-leaf path of a tree. Root-to-leaf path retrieval is similar to the depth-first search algorithm for graph traversal (Skiena, 2008). So we apply the depth-first search algorithm for root-to-leaf path retrieve. The depth-first search algorithm visits each node from the root to each leaf. We can thus record the nodes in the order of appearance in the path and retrieve the hierarchy.

5.3 Functional Dependency Tree Element Set

FDs of a tabular dataset may form different FD trees and then form different hierarchies and dimensions. To construct each tree, we have to find all the FDs that have links among them and group them together, which is costly. So, instead, we directly obtain a

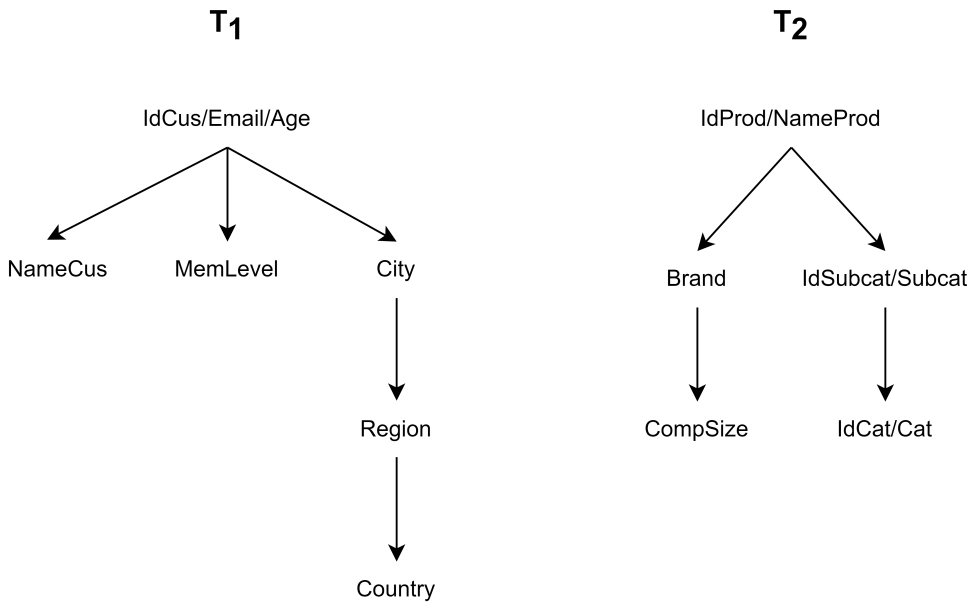


Figure II.7: Examples of FD trees

functional dependency tree element set containing all the elements of the FD trees derived from the detected FDs.

Definition 5.3 (Functional dependency tree element set)**.** *A* ***functional dependency tree element set*** *is a set of elements of a functional dependency tree set (*$TS = \{FDT_1, FDT_2, ...,$
$FDT_n\}$*), denoted by* $TE^{TS} = \{V_r^{TS}, V_l^{TS}, V_b^{TS}, PC^{TS}\}$*, where:*

- V_r^{TS} *is a root node set containing all root nodes of the FD trees,* $V_r^{TS} = \bigcup_{i=1}^{n} V_{ri}$,
- V_l^{TS} *is a leaf node set containing all leaf nodes of the FD trees,* $V_l^{TS} = \bigcup_{i=1}^{n} V_{li}$,
- V_b^{TS} *is a branch node set containing all branch nodes of the FD trees,* $V_b^{TS} = \bigcup_{i=1}^{n} V_{bi}$,
- $PC^{TS} = V_p^{TS} \rightarrow V_c^{TS}$ *is a parent-children map associating each parent node to its child nodes,* $V_p^{TS} = V_r^{TS} \cup V_l^{TS}$ *and* $V_c^{TS} = V_l^{TS} \cup V_l^{TS}$.

Example 5.3 (Functional dependency tree element set)**.** *The FD trees from Fig. II.7 bears a FD tree set* $TS = \{T_1, T_2\}$*. It is obtained by the FDs of Example 5.1. To create the two FD trees, we have to separate the FDs and group the FDs having the same attributes on any side together. However, given these FDs, we can directly get the functional dependency tree element set* $TE^{TS} = \{V_r^{TS}, V_l^{TS}, V_b^{TS}, PC^{TS}\}$*, where:*

- $V_r^{TS} = \{IdCus/Email/Age, IdProd/NameProd\}$,
- $V_l^{TS} = \{NameCus, MemLevel, Country, CompSize, idCat/Cat\}$,

- $V_b^{TS} = \{City, Region, Brand, IdSubcat/Subcat\}$,
- $PC^{TS} = \{IdCus/Email/Age : \{NameCus, MemLevel, City\}, City : \{Region\}, Region : \{Country\}, IdProd/NameProd : \{Brand, IdSubcat/Subcat\}, Brand : \{CompSize\}, IdSubcat/Subcat : \{IdCat/Cat\}\}$.

Algo. 1 describes the creation of the FD tree element set. We first construct an empty FD tree element set TE^{TS} ($line_1$). We also create a list for all left-hand sides of FDs ($line_2$) and a list for all right-hand side of FDs ($line_3$). For each FD in the FD set ($line_4$), its left-hand side is put into the left-hand side list ($line_5$) and its right-hand side is put into the right-hand side list ($line_6$). The right-hand side is added into a map as the left-hand side key value ($line_{7-9}$). When the loop of the FDs ends, we get the complete left-hand and right-hand side lists, as well as the parent-children map. Root node has no any other attribute determining it, which means that it does not act as a right-hand side in any FD. Thus, the left-hand side list is $lhsList = V_r^{TS} \cup V_b^{TS}$. A leaf node has no any other attribute determined by it, which means that it does not act as a left-hand side in any FD. Thus the right-hand side list is $rhsList = V_l^{TS} \cup V_b^{TS}$. Finally, we thus get the branch node set, the root node set and the leaf node set of the FD tree element set ($line_{11-13}$).

Algorithm 1: $getFDTreeElems(FDS)$

Input : Set of detected functional dependencies FDS
Output: Functional dependency tree element set TE^{TS}

```
1  TE^TS ← {∅, ∅, ∅, ∅} ;
2  lhsList ← ∅ ;
3  rhsList ← ∅ ;
4  for FD ∈ FDS do
5      lhsList ← lhsList + FD.lhs ;
6      rhsList ← rhsList + FD.rhs ;
7      if FD.lhs ∉ PC^TS.keys() then
8          PC^TS[FD.lhs] ← FD.rhs
9      else
10         PC^TS[FD.lhs] ← PC^TS[FD.lhs] + FD.rhs
11 V_b^TS ← lhsList ∩ rhsList ;
12 V_r^TS ← lhsList − V_b^TS ;
13 V_l^TS ← rhsList − V_b^TS ;
14 return TE^TS
```

Example 5.4 (Creation of an FD tree element set)**.** *FDs from Example 5.1 are each scanned to get* $lhsList = \{IdCus/Email/Age, IdProd/NamePro, City, Region, Brand, IdSubcat/Subcat\}$, $rhsList = \{City, Region, Brand, IdSubcat/Subcat, NameCus, MemLevel, Country, CompSize, idCat/Cat\}$ *and* PC^{TS}*, as in Example 5.3. We thus*

obtain $V_b^{TS} = lhsList \cap rhsList = \{City, Region, Brand, IdSubcat/Subcat\}$*, and then* $V_r^{TS} = lhsList - V_b^{TS}$ *and* $V_l^{TS} = rhsList - V_b^{TS}$ *as in Example 5.3.*

5.4 Hierarchy Detection

Algo. 2 describes the detection of hierarchies from a FD tree element set by using depth-first search. It is a recursive algorithm. The inputs also include the node, the hierarchy to be detected and the hierarchy set containing all detected hierarchies. Here, since we are not yet in the step of parameter and weak attribute distinction, we simplify the representation of the hierarchies by using only their ordered parameter sets. In the first recursion of the algorithm, the node is the root node, the hierarchy and the hierarchy set are empty sets (Algo. 2, $line_4$). We add the attribute into the hierarchy when we pass a node ($line_1$). If the node is not a leaf node ($line_2$), we continue to recursively pass the next level ($line_{3-4}$). If the node is a leaf node ($line_5$), a root-to-leaf path is found and a hierarchy is completely retrieved. Then, we can add the hierarchy into the hierarchy set ($line_6$).

Example 5.5 (Hierarchy detection)**.** *We take the example of the root node* $IdCus/Email/Age$ *of the functional dependency tree element set* TE^{TS} *from Example 5.3. We call Algo. 2 by* $getHierarchy(TE^{TS}, IdCus/Email/Age, \emptyset, \emptyset)$*. Fig. II.8 illustrates the retrieval of the hierarchies where* ***v*** *represents the current executed node.* ***H*** *represents the current retrieved hierarchy.* ***leaf node*** *denotes whether the current node is a leaf node.* ***HS*** *represents the current hierarchy set. Each recursion loop ends when the leaf node and the retrieved hierarchy are added into the hierarchy set. The result is returned up to the result of the first recursion as shown with the dashed lines and we finally obtain 3 hierarchies:* $< IdCus/Email/Age, NameCus >$*,* $< IdCus/Email/Age, MemLevel >$ *and*
$< IdCus/Email/Age, City, Region, Country >$*.*

Algorithm 2: $getHierarchies(TE^{TS}, v, H, HS)$

Input : Functional dependency tree element set TE^{TS}, node v and hierarchy H, hierarchy set HS
Output: Hierarchy set HS

```
1 H ← H + v;
2 if v ∉ V_l^TS then
3     for v_2 ∈ PC^TS[v] do
4         HS ← HS ∪ getHierarchie(TE^TS, v, H, HS)
5 else
6     HS ← HS + H
7 return HS
```

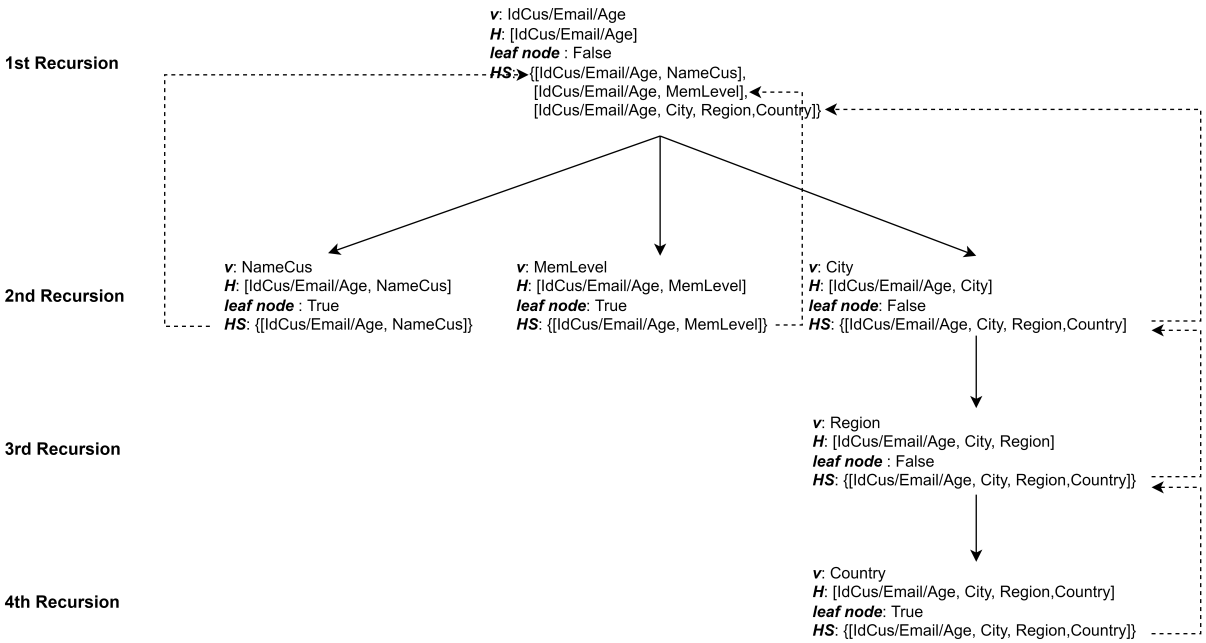


Figure II.8: Example of hierarchy detection

5.5 Distinction between Parameters and Weak Attributes

Our hierarchy detection method considers all attributes as the same, i.e., without distinguishing whether an attribute is a parameter or a weak attribute. Thus, in this section, we define rules for making this distinction.

The value of a weak attribute is determined by its parameter and usually does not determine the value of any other attributes.

- the group of equivalent attributes, or
- the highest-granularity level of hierarchies.

So we must address both cases.

5.5.1 Equivalent Attributes

Equivalent attributes that are not on the highest-granularity level are attributes of the same level. Thus, there must be one parameter and the other attributes are weak attributes. The parameter should be the attribute that can be an identifier. We define the following rules to choose the parameter.

- At the schema level, we look for the attribute whose name contains some strings that indicate that the attribute could be a parameter, such as “code”, “id”, etc.
- If there is no such attribute, we look at the instance level. We look for the attribute whose values can be abbreviations of other attributes by seeing if its strings consist of other attribute values of the same instance.

- If there is no such attribute, we look for the remaining string attributes. We look for the attribute that is of nominal or ordinal numerical types (Alobaid et al., 2019).
- If there is no such attribute, we look for the attribute whose values are composed of both strings and numerical data.
- If there is no such attribute, we look for the attribute that has the shortest string length.
- Date type data are treated as weak attributes.

We search the parameter with respect to the above rules in order. All the remaining attributes are weak attributes.

5.5.2 Highest-granularity Level

We must decide whether each highest-granularity level attribute is a parameter or a weak attribute. To do so, we have to verify whether the attribute really has a semantic hierarchical relationship with the other attributes of the hierarchy, so we define these following rules.

- At the schema level, we verify the hierarchical relationships between the highest-granularity level attribute and the other attributes by checking whether their names or subset of names match the semantic hierarchy relationship in Wordnet (Miller, 1995). If so, the attribute is a parameter.
- If there is no such attribute, we look at the instance level. We also verify their hierarchical relationships with Wordnet, but with the instance values, to decide whether it is a parameter.
- If not, we verify whether the attribute is categorical by setting a threshold on the distinct value ratio (distinct value number divided by total value). If the distinct value ratio is lower than the threshold, then it is treated as a parameter.
- Date type data are treated as weak attributes.

If none of the rules is satisfied, then the attribute is a weak attribute.

Example 5.6. *The detected hierarchies from Example 5.5 have an equivalent attribute group (IdCus, Email, Age) that is not on the highest-granularity level. We find the string "Id" in attribute name "IdCus", so attribute IdCus is a parameter. Conversely, Email and Age are weak attributes. Then, we look at the highest-granularity level attributes of the 3 detected hierarchies. Hierarchy* $< IdCus, City, Region, Country >$ *matches the hierarchical relationship City, Region and Country, so Country is a parameter. Hierarchies* $< IdCus, NameCus >$ *and* $< IdCus, MemLevel >$ *do not bear hierarchical relationships in Wordnet between the highest-granularity level attributes and the other attributes at both schema (attribute name) and instance level. MemLevel is of ordinal numerical type, so it*

is a parameter. NameCus is not of numerical type, so we verify whether it is categorical by setting a threshold of 0.6. The distinct ratio of NameCus is $5/6 = 0.83 > 0.6$*, so it cannot be a parameter. It is thus a weak attribute.*

5.6 Construction of DW

Algo. 3 describes the full process of automatic DW design and implementation by combining the previous steps to construct a DW. It is to be noted that the names of the components are not assigned in the algorithm since they can be assigned automatically or by the user.

Given a tabular data of simple structure T, the measure detection is first carried out ($line_1$). Then FDs are obtained by HyFD algorithm from the non-measure attributes ($line_2$).

A FD tree element set is created based on the detected FDs ($line_4$). Each FD tree can be considered as a dimension where the root node is the dimension identifier. Thus for each root node, we create a dimension ($line_{6-26}$) by detecting hierarchies ($line_{6-10}$) and applying the proposed rules to identify dimension attributes as parameters or weak attributes ($line_{10-20}$).

A fact is created with the detected measures ($line_{29}$). A star schema is then generated by linking the fact to the dimensions ($line_{31}$). The constellation schema contains more than one fact, however, normally a tabular data contains only one fact, which is the case we assume. Thus we only consider the generation of a star schema.

In terms of the implementation, we apply the R-OLAP architecture (Kimball and Ross, 2011), which is the most used OLAP implementation (Pujolle et al., 2011). we first implement the tabular data in the database as a table T ($line_3$). For the dimensions, we create a table for each dimension and extract the dimension instances by projecting the distinct attribute instances from the table T ($line_{24}$). For the fact, we create a fact table and project the measure instances from T ($line_{27}$) and link them with the corresponding dimension instances ($line_{28}$) by adding foreign keys.

Example 5.7. *The measure detection, the detection of certain hierarchies and the identification of certain attributes as parameters/weak attributes of the CSV table from Fig. II.5 are illustrated in the previous examples. A fact can be created based on the detected measures. By carrying out the hierarchy detection and parameter and weak attribute distinction for all nodes of the FD tree element set obtained in Example 5.3, we obtain three dimensions. There is a dimension of date, thus we ask the user to choose the date granularities. We then ask the user to rename the fact, dimension and hierarchy names. We can finally obtain a star schema like shown in Fig. II.9.*

The implementation result is shown in Fig. II.10. Three tables are created respectively for three dimensions. Each table contains the attributes of its corresponding dimension.

Algorithm 3: *autoDW*(T)

Input : Tabular data of simple structure T
Output: A data warehouse DW

Launch measure detection to obtain M^F;
Launch HyFD for non-measure attributes to obtain functional dependency set FDS;
Implement tabular data T in database as table T;
$TE^{TS} \leftarrow getFDTreeElems(FDS)$;
$D^{DW} \leftarrow \emptyset$;
for $V_{ri} \in V_r^{TS}$ **do**
  $id^{D_i} \leftarrow V_{ri}$;
  $A^{D_i} \leftarrow \emptyset$;
  $ParamSet \leftarrow getHierarchies(TE^{TS}, V_{ri}, \emptyset, ParamSet)$;
  for $Param^{H_j} \in ParamSet$ **do**
    $A^{D_i} \leftarrow A^{D_i} \cup Param^{H_j}$;
    $Weak^H H_j \leftarrow \emptyset$;
    for $k = 0$ *to* $|Param^{H_j}| - 1$ **do**
      if $Param^{H_j}[k]$ *is an equivalent attribute combination* **then**
        Apply the rules in Section 5.5.1 to identify a parameter p and a weak attribute set $Weak^{H_j}[p]$
      if $Param^{H_j}[k]$ *is a highest-granularity level parameter* **then**
        Apply the rules in Section 5.5.2;
        if $Param^{H_j}[k]$ *is a weak attribute* **then**
          $Weak^{H_j}[Param^{H_j}[k-1]].add(Param^{H_j}[k])$;
          $Param^{H_j}.delete(Param^{H_j}[k])$;
    $H_j \leftarrow (N^{H_j}, Param^{H_j}, Weak^{H_j})$;
    $H^{D_i}.add(H_j)$;
  $I^{D_i} \leftarrow \Pi_{A^{D_i}} T$;
  $D_i \leftarrow (N^{D_i}, A^{D_i}, H^{D_i}, I^{D_i})$;
  $D^{DW}.add(D_i)$;
$I^F \leftarrow \Pi_{M^F} T$;
$IStar^F \leftarrow \{i_p^F \rightarrow i_q : i_p^F \in I^F \wedge i_q \in \prod_{D_r \in D^{DW}} I^{D_r} \wedge i_p^F, i_q$ are in the same tuple of T$\}$;
$F \leftarrow (N^F, M^F, I^F, IStar^F)$;
$F^{DW} \leftarrow \{F\}$;
$Star^{DW}[F] \leftarrow D^{DW}$;
$DW \leftarrow (N^{DW}, F^{DW}, D^{DW}, Star^{DW})$;
return DW

The dimension identifiers are assigned as primary keys. A fact table is created containing the measures as well as the foreign keys which associate the fact table to the dimensions.

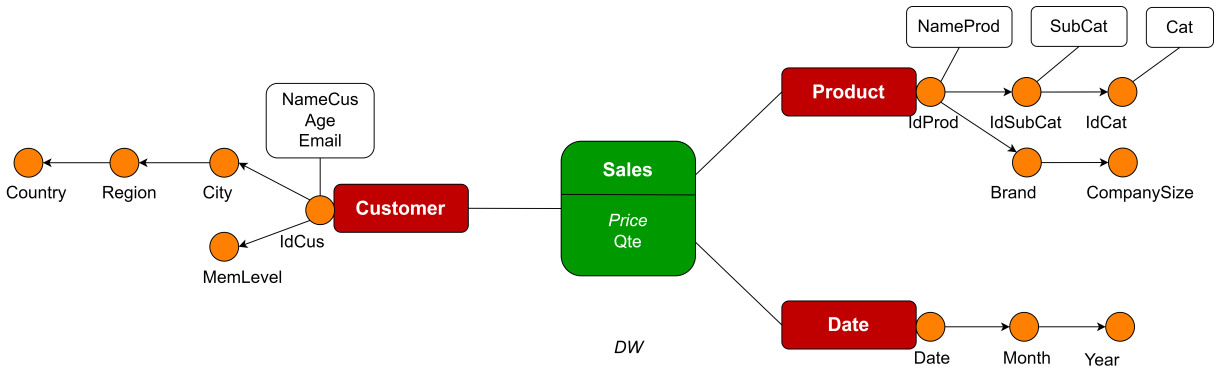
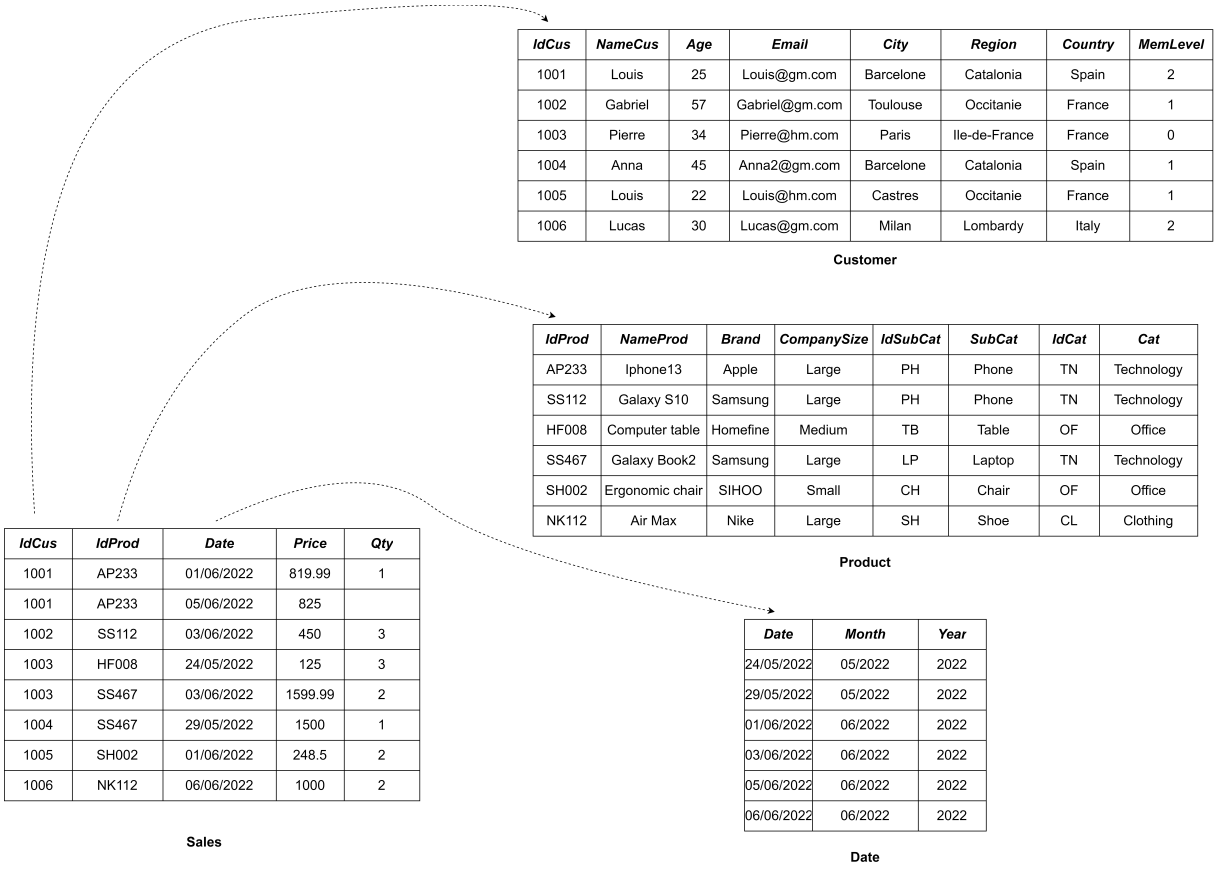


Figure II.9: Final schema



IdCus	NameCus	Age	Email	City	Region	Country	MemLevel
1001	Louis	25	Louis@gm.com	Barcelone	Catalonia	Spain	2
1002	Gabriel	57	Gabriel@gm.com	Toulouse	Occitanie	France	1
1003	Pierre	34	Pierre@hm.com	Paris	Ile-de-France	France	0
1004	Anna	45	Anna2@gm.com	Barcelone	Catalonia	Spain	1
1005	Louis	22	Louis@hm.com	Castres	Occitanie	France	1
1006	Lucas	30	Lucas@gm.com	Milan	Lombardy	Italy	2

Customer

IdProd	NameProd	Brand	CompanySize	IdSubCat	SubCat	IdCat	Cat
AP233	Iphone13	Apple	Large	PH	Phone	TN	Technology
SS112	Galaxy S10	Samsung	Large	PH	Phone	TN	Technology
HF008	Computer table	Homefine	Medium	TB	Table	OF	Office
SS467	Galaxy Book2	Samsung	Large	LP	Laptop	TN	Technology
SH002	Ergonomic chair	SIHOO	Small	CH	Chair	OF	Office
NK112	Air Max	Nike	Large	SH	Shoe	CL	Clothing

Product

IdCus	IdProd	Date	Price	Qty
1001	AP233	01/06/2022	819.99	1
1001	AP233	05/06/2022	825	
1002	SS112	03/06/2022	450	3
1003	HF008	24/05/2022	125	3
1003	SS467	03/06/2022	1599.99	2
1004	SS467	29/05/2022	1500	1
1005	SH002	01/06/2022	248.5	2
1006	NK112	06/06/2022	1000	2

Sales

Date	Month	Year
24/05/2022	05/2022	2022
29/05/2022	05/2022	2022
01/06/2022	06/2022	2022
03/06/2022	06/2022	2022
05/06/2022	06/2022	2022
06/06/2022	06/2022	2022

Date

Figure II.10: Implementation result

6 Experimental Assessment for Measure Detection

The process of our experiments for measure detection is shown in Fig. II.11. We carry out experiments with various datasets and algorithms to validate the proposed measure detection approach. The objectives of the experiments include: (1) validating the effectiveness of the proposed ML-based approach for measure detection, (2) validating the

effectiveness of the different proposed feature categories, (3) validating the generality of the trained mode with the proposed features and (4) analysing the importance of the proposed features in each algorithms.

6.1 Experimental Conditions

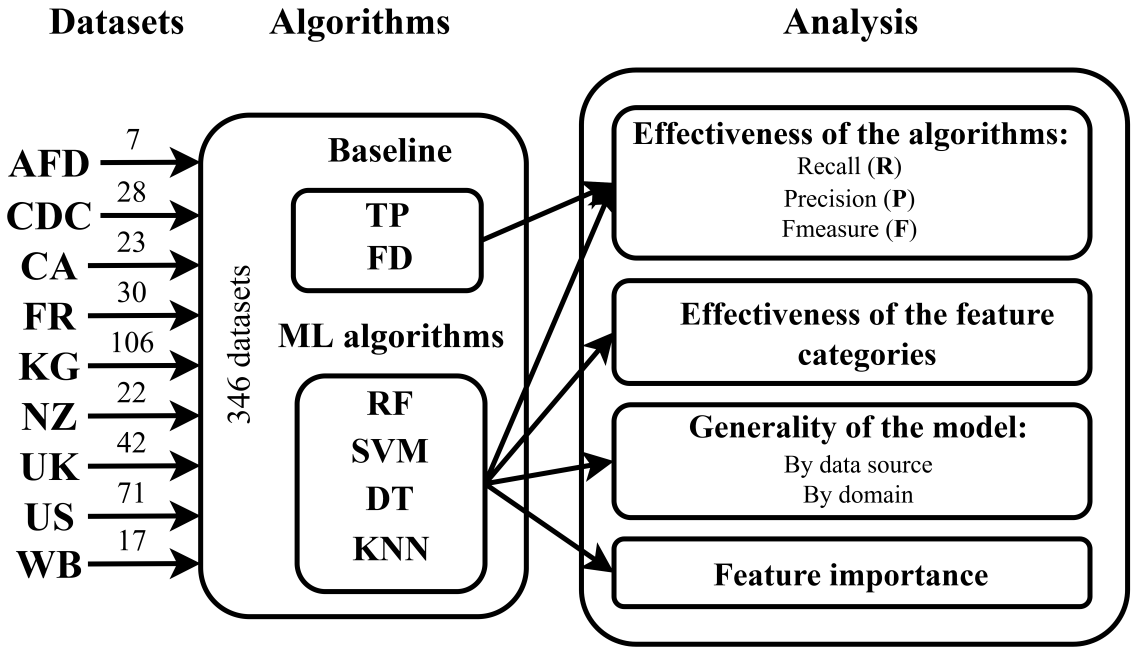


Figure II.11: Experiment overview

6.1.1 Technical Environment

Our experiments are conducted on an Intel(R) Core(TM) i5-10210U 1.60 GHz CPU with 16 GB RAM. The programming language that we apply is Python 3.7. This is also the technical environment for the other experiments in this manuscript.

6.1.2 Datasets

We use 9 datasets in our experiments, The objective of employing data coming from different sources is to guarantee that our datasets cover different domains, topics and languages so that the result is more convincing. The datasets come from sources including the governmental open data sites of France (**FR**)[1], Canada (**CA**)[2], UK (**UK**)[3] and US (**US**)[4], the French Development Agency (**AFD**)[5], the New Zealand's official data agency (**NZ**)[6], the American Center for Disease Control and Prevention (**CDC**)[7], the World

[1] `https://www.data.gouv.fr`
[2] `https://open.canada.ca`
[3] `https://data.gov.uk`
[4] `https://www.data.gov`
[5] `https://opendata.afd.fr`
[6] `https://www.stats.govt.nz`
[7] `https://data.cdc.gov`

Bank (**WB**)[8] and Kaggle (**KG**)[9].

The datasets that we choose contain at least one numerical column. In our corpus, there are files that are used for other specific purpose, e.g., machine learning, which are not suitable to DW creation. We discard them. There are also files with very poor data quality or completely lacking the information to understand the semantic meaning of columns, which makes it difficult to tell whether a column can be a measure. We also discard such files.

Each dataset contains numerous tables with numerical columns on which features are extracted to feed the algorithms. Data are classified into five domains including Economy (**ECO**), Health (**HLT**), Government (**GOV**), Environment (**ENV**) and Society (**SOC**). Each domain includes a different number of files (Table II.2). Eventually, the languages used in data sources differ, i.e., files from **AFD** and **FR** are in French while the others are in English. The number of the CSV files in the datasets is 346 and there are 3524 columns including 1382 numerical columns. There are 900 numerical columns that can be considered as ground truth measures.

Table II.2: Number of files by domains

Domain	**ECO**	**HLT**	**GOV**	**ENV**	**SOC**
File Number	143	57	80	28	38

Table II.3 shows information about each data source and all data sources (**Total**), including the number of files ($\mathbf{N}_f$), the number of numerical columns ($\mathbf{N}_c$), the number of measures ($\mathbf{N}_m$) and the ratio of number of measures by the number of numerical columns ($\mathbf{R}_m$). Figures in brackets are the minima and maxima. The original datasets can be found in our github[10].

For each dataset, we compute all our features for each numerical column, and label them to build the training and test sets. Empty values in columns are ignored.

6.1.3 Baseline Methods

Numerical Typology-Based Method (TP) In a previous work, we propose to select measures with respect to the type of numerical attributes (**YANG, Y.** et al., 2020). Numerical data may be classified into nominal data, ordinal data, intervals and ratios (Alobaid et al., 2019).

Nominal data are labels composed of digits which are used instead of names to identify things. Ordinal data implies an order among a set of elements but with no regard to the difference between the elements. Interval is used to denote the increase or expansion in some way on a scale such as the temperature. Ratio is the scale that we use to measure

[8] `https://data.worldbank.org`
[9] `https://www.kaggle.com`
[10] `https://github.com/Implementation111/measure-detection`

Table II.3: Data source characteristics

Data Source	N_f	N_c	Rg_c	N_{nc}	Rg_{nc}	Rt_{nc}	N_m	Rg_m	Rt_m	L
AFD	7	82	(6, 18)	15	(1, 14)	18.29	8	(0, 3)	53.33	Fr
CDC	28	247	(3, 30)	100	(1, 12)	40.49	70	(1, 6)	70.00	En
CA	23	285	(5, 29)	156	(2, 28)	54.74	113	(0, 28)	72.44	En
FR	30	410	(2, 54)	123	(1, 38)	30.00	39	(0, 7)	31.71	Fr
KG	106	1041	(2, 29)	394	(1, 17)	37.85	271	(0, 10)	68.78	En
NZ	22	162	(3, 15)	62	(1, 13)	38.27	43	(0, 12)	69.35	En
UK	42	390	(2, 39)	137	(1, 9)	35.13	99	(0, 8)	72.26	En
US	71	714	(2, 28)	311	(1, 20)	43.56	194	(0, 18)	62.38	En
WB	17	193	(5, 26)	84	(1, 18)	43.52	63	(0, 13)	75.00	En
Total	346	3524	(2, 54)	1382	(1, 38)	39.22	900	(0, 28)	65.12	En

things and which contain a real zero like the number of students. Among all these kinds of numerical data, the interval and ratio type is most likely to be measures.

Algorithms are proposed to detect the different numerical type (Alobaid et al., 2019). So we implement these algorithms and apply them on each numerical column to get its numerical type. Then we choose the columns of interval and ratio types to be measures.

Functional Dependency-Based Method (FDB) As we already mentioned, in existing methods aimed at data with schemas, measures are selected in tables exhibiting many-to-many relationships; in other words, columns that are functionally dependent on dimension primary keys. With this idea in mind, we detect functional dependencies (FDs) in tabular data and select as measures the numerical columns that are functionally determined by several, other attributes. The FD detection algorithm that we use is HyFD (Papenbrock and Naumann, 2016) as we explained in Section 5.1.

We take advantage of the Metanome toolbox (Papenbrock et al., 2015a), which is developed by the team of the HyFD designers and which integrate different FD detection algorithms including HyFD. The tool is developed by Java and we extract the code concerning HyFD. Then we integrate the Java code in our implementation Python code and use an API to execute them to obtain the FDs. The extracted FDs are also used for generating the values of feature **severalfds**.

6.1.4 Application of ML Algorithms

To validate the proposed machine learning-based solution and the proposed features, we apply the following widely used Machine Learning (ML) classification algorithms (Sen et al., 2020):

- a random forest classifier (**RF**),
- a support vector machine classifier with an Radial Basis Function (RBF) kernel (**SVM**),

- a decision tree classifier based on the CART (Classification and Regression Trees) algorithm (**DT**),
- a k-nearest neighbors classifier (**KNN**)

Deep learning models are not employed because they are more suitable for interpreting images, sounds and texts (LeCun et al., 2015), while we analyse numerical columns.

We define the ground truth by analysing each dataset context according to its website's description, header semantics and metadata. We also uphold the criteria from Definition 4.1. Since we are in the context of data-driven DW design without specific requirement, we consider numerical columns that can be potential measures in all possible cases.

6.2 Experimental Results

6.2.1 Algorithm Effectiveness

We run the two baseline methods from (Section 6.1.3) and train models with our features with four ML algorithms (Section 6.1) on all datasets (Section 6.1). The ML algorithms are run by pycaret [11] Python library where the hyperparameters are tuned automatically. For the model generality and feauture importance experiments, we run ML algorithms from the sklearn[12] Python library.

We use three performance metrics to verify the effectiveness of different algorithms: Recall (**R**), Precision (**P**) and F-score (**F**). Let N_{mm} and N_{mn} be the number of measures predicted as measures and non-measures, respectively; and N_{nm} and N_{nn} the number of non-measure predicted as non-measures and measures, respectively.

Then, $\mathbf{R} = \frac{N_{mm}}{N_{mm} + N_{mn}}$, $\mathbf{P} = \frac{N_{mm}}{N_{mm} + N_{nm}}$ and $\mathbf{F} = \frac{2 \times Precision \times Recall}{Precision + Recall}$.

Table II.4 shows the resulting values of **R**, **P** and **F** where the results of ML algorithms are obtained through a 10-fold cross validation by merging all datasets and randomly split them into 10 folds. The distribution of the cross validation results is depicted in Figure II.12.

Table II.4: Global results

Metric	**TP**	**FDB**	**RF**	**SVM**	**DT**	**KNN**
R(%)	80.05	75.43	**96.64**	94.77	94.08	90.16
P(%)	73.57	77.50	**90.89**	78.44	88.44	87.61
F(%)	76.67	76.45	**93.65**	85.76	91.12	88.78

We observe that **RF** exhibits the best F-score (94.82%) and the result is not more

[11] `https://pycaret.org/`
[12] `https://scikit-learn.org`

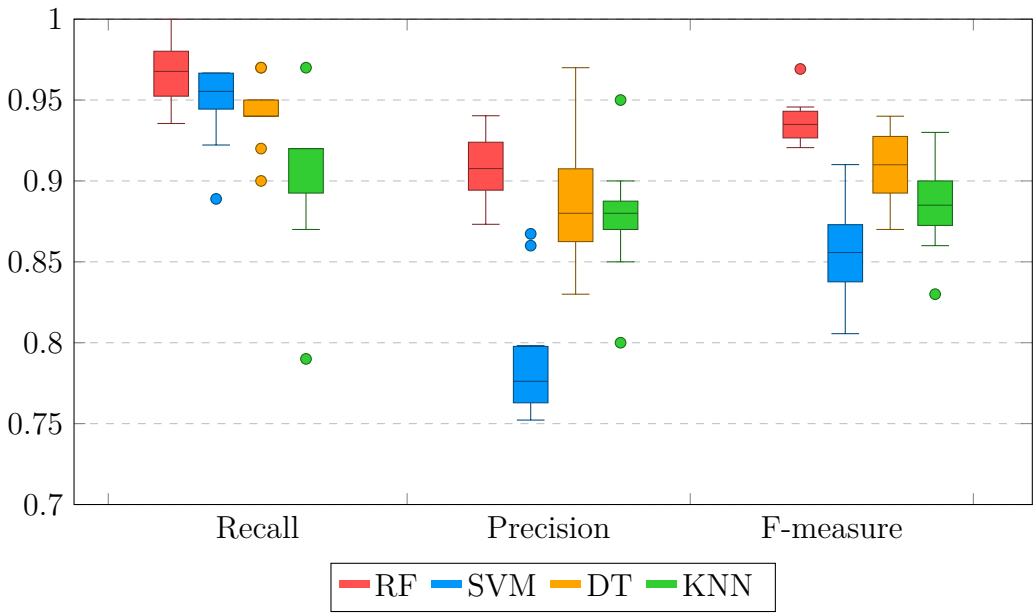


Figure II.12: Cross validation distribution

dispersed than that of the other algorithms. Thus, **RF** shows the best performance on the measure detection problem. We also observe that **TP** and **FDB** do not have a good effectiveness when predicting measures, but **FDB** performs better than **TP**. **TP**'s bad performance is due to

- interval and ratio numerical columns are not all measures, e.g., longitude and latitude;
- numerical typology detection algorithms are not flexible enough to cope with real-world data, because they are based on fixed rules.

Regarding **FDB**, a numerical column that is functionally determined by several other columns may not always be a measure. For example, let us consider a table describing sale facts with respect to customers and products, where sale amount is indeed a measure. The customer ID is the customer dimension's primary key, but the customer's name and email may uniquely identify a customer, and thus may functionally determine the age of the customer, a numerical column that is not a measure.

Our ML-based measure detection method takes different types of features into account and can thus better handle the above exceptions and achieve better results.

6.2.2 Feature Category Effectiveness

To verify the effectiveness of each feature category we propose, we test different combinations of feature categories with our **RF**-based method. We first test single feature categories, combinations of two categories and then we compare the effectiveness of all categories. The result is shown in Table II.5, where **GE** represents general features, **ST** represents statistical features and **IC** represents inter-column features (Section 4.3). **ST**

Table II.5: Performance of feature categories and their combinations

ML Algorithms	Metrics	**GE**	**ST**	**IC**	**GE+ST**	**GE+IC**	**ST+IC**	**ALL**
RF	**R(%)**	88.10	94.27	92.68	95.30	93.67	91.93	**96.64**
	P(%)	83.59	86.28	80.91	88.21	86.13	91.14	**90.89**
	F(%)	85.69	90.01	86.37	91.57	89.67	91.50	**93.65**
SVM	**R(%)**	92.20	93.96	88.89	94.07	92.86	93.70	**94.77**
	P(%)	74.45	76.80	75.47	76.85	76.90	76.71	**78.44**
	F(%)	82.32	84.35	81.63	84.45	84.47	84.23	**85.76**
DT	**R(%)**	89.05	89.16	89.90	89.97	88.47	89.12	**91.20**
	P(%)	78.53	86.24	83.62	**89.22**	88.26	87.15	89.17
	F(%)	83.29	87.59	86.54	89.55	88.28	88.07	**90.12**
KNN	**R(%)**	84.13	91.95	92.07	85.56	**92.57**	92.08	90.16
	P(%)	83.73	82.45	81.48	86.06	84.14	83.65	**87.61**
	F(%)	83.82	86.90	86.42	85.68	88.11	87.59	**88.78**

exhibits the best individual contribution. Yet, we can clearly see that combining feature categories achieves better performance in terms of recall, precision and F-score, than using single feature categories. Ultimately, combining all feature categories yields the best performance. The results of applying other ML algorithms can be found in our github.

6.2.3 Model Generality

To verify that the trained model achieved with our **RF**-based method is generic, we train data by excluding the datasets of one source and test on them. We also carry out the same test by domain, i.e., economy (**ECO**), health (**HLT**), government (**GOV**), environment (**ENV**) and society (**SOC**). The results are shown in Figure II.13, where the charts depict the results by source and domain, respectively. By comparing with former results, the difference of F-score ranges from -5.02% to 4.23% for the test with respect to the source and from -3.17% to 3.36% for the test with respect to the domain. The trained model is thus generic regardless of the source and data domains.

6.2.4 Feature Importance

To analyse our different features, we compute the permutation importance, i.e., the decrease in prediction accuracy when a feature is permuted (Fisher et al., 2019) of each feature for all ML algorithms. Figure II.14 shows that the importance of a feature varies with respect to the algorithm. For example, with **SVM** and **KNN**, the statistical features are more important than others, while with **RF** and **DT**, the features bearing the highest importance values are more equally distributed in each feature category. There are also features that bear negative importance values with some algorithms, e.g., numerical neighbor in algorithm DT, but not every time, while they always have positive importance values with other algorithms. There is no feature that always bears zero or negative importance values with one single algorithm, which means that all our features have a contribution to the ML classifiers. With **RF**, which bears the best performance,

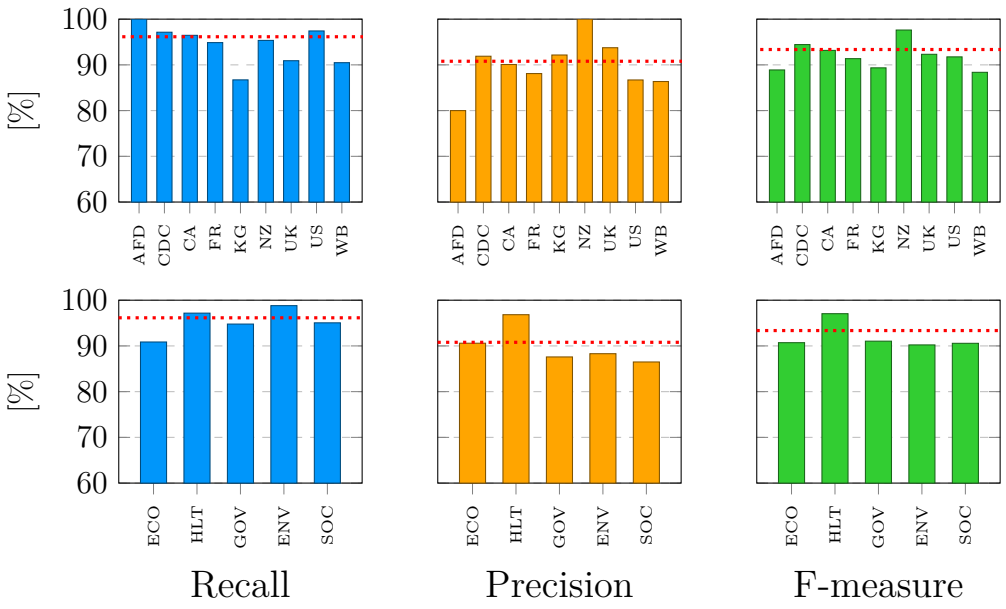


Figure II.13: Performance with respect to source and domain with **RF**

the most important feature is the location ratio. By checking the CSV files, we observe that most of the measures are situated at the last part of the file, while most of the columns in the front part are descriptive, which probably explains the importance of the location ratio.

7 Experimental Assessment for Dimension Detection

In order to validate the effectiveness and efficiency of our dimension detection algorithms, we conduct experiments with different tests by applying various datasets.

7.1 Dataset

In our experiments, we use 8 datasets including 1 synthetic data **Example** containing the same data as the example in this chapter and 5 real-world datasets. Among these real-world datasets, 3 datasets come from Kaggle[13] including **Sales1**, **Sales2** which contain sales data but with different information of different supermarkets and **DevApp** which contains data about some application development projects; 2 datasets come from the site of the World Bank[14] including **Countries** containing various indicator data about different countries and **Covid** containing data about the covid pandemic in different countries.

Table II.6 shows the information of these datasets. The information includes the number of columns after excluding the measures of each dataset (N_c), the number of rows (N_r), the number of the dimensions (N_d) and hierarchies (N_h).

[13] `https://www.kaggle.com/`
[14] `https://data.worldbank.org/`

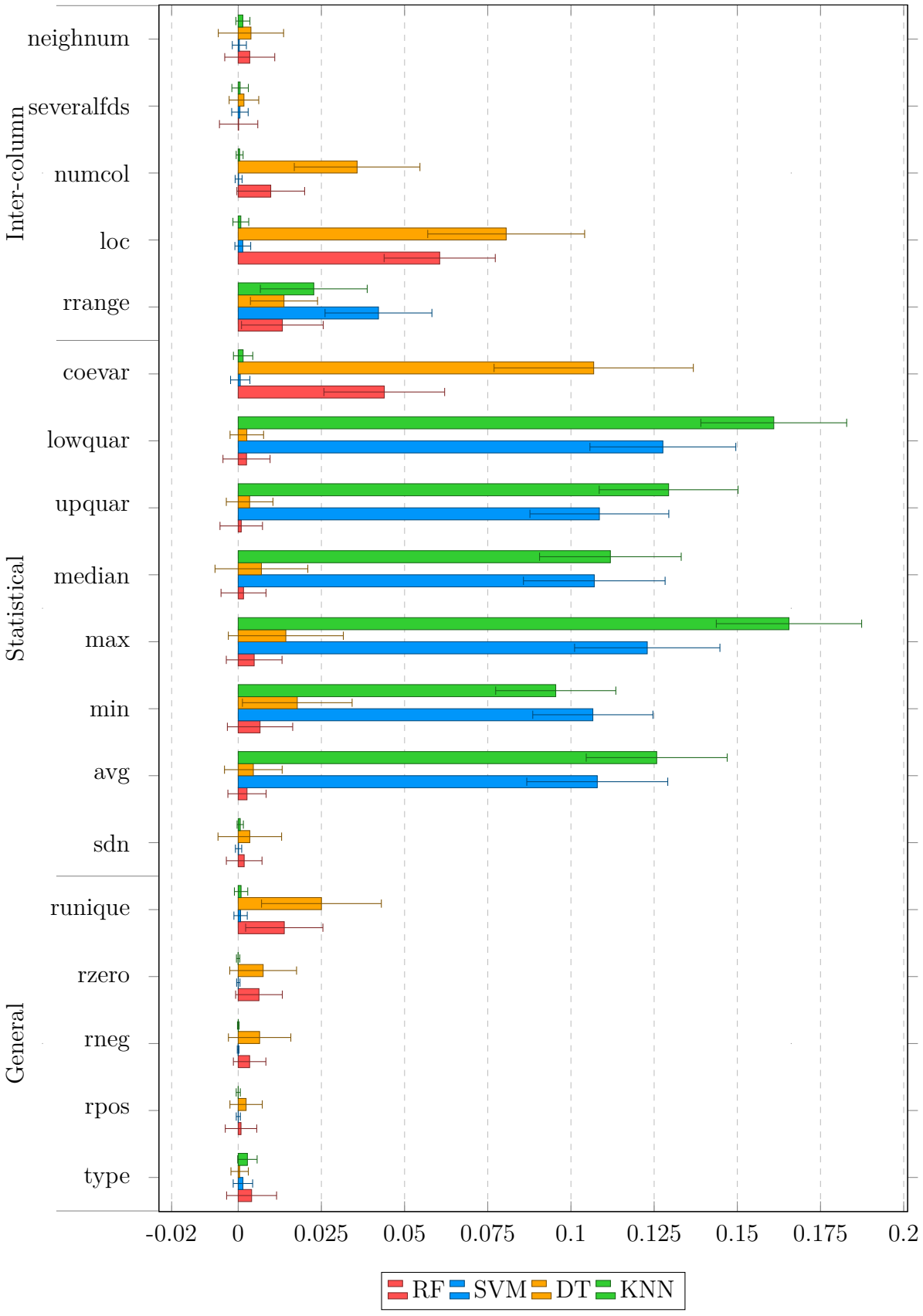


Figure II.14: Feature importance

Table II.6: Dataset information

Dataset	N_c	N_r	N_d	N_h
Example	17	9	3	4
Sales1	16	9918	3	4
Sales2	8	794	2	3
DevApp	11	2752	2	3
Countries	4	84	1	1
Covid	12	128041	2	6

7.2 Metrics

To carry out the experiments, we first remove the measures for each file. We then execute the dimension detection algorithms.

We analyse the data in the files and the descriptions in the sources to manually design multidimensional schemas as the ground truth. To evaluate the effectiveness, we use the metrics recall (**R**), precision (**P**) and F-score (**F**) defined as follow:

$$\mathbf{R} = \frac{\{Detected\} \cap \{True\}}{\{True\}}, \mathbf{P} = \frac{\{Detected\} \cap \{True\}}{\{Detected\}} \text{ and } \mathbf{F} = \frac{2 \times Precision \times Recall}{Precision + Recall},$$

Where $\{Detected\}$ is the set of the detected elements and $\{True\}$ is the set of the true elements of ground truth. Here, the elements may be different, we evaluate the effectiveness at three aspects containing six levels of elements:

- **Dimension aspect**: We verify the effectiveness for the detection of dimensions by checking if we detect the correct dimensions and if they contain the correct attributes.
  - **Dimension ID**: We verify if we detect the correct dimensions by verifying the dimension identifiers. So here, the elements are the dimension identifiers.
  - **Attribute**: We verify for each detected dimension, if it contains the same attributes as the ground truth dimension. So here, the elements are the attributes of a dimension.
- **Dimension attribute aspect**: Then we verify if the attributes in the dimensions are correctly distinguished as parameters and weak attributes.
  - **Parameter**: We verify if we detect the correct parameters. So here, the elements are the parameters.
  - **Weak attributes**: We verify if we detect the correct weak attributes. So here, the elements are the weak attributes.
- **Relationship aspect**: We also verify the relationships in dimensions including the hierarchical relationships between parameters and the same level relationships

between a parameter and the corresponding weak attributes.

- **Hierarchical relationship**: We verify if we detect the correct hierarchical relationships between the levels. So here, the elements are the binary relationships between the levels. More precisely, we look at the binary relationships between the parameters. However, there may be the case where the true parameter of a level is identified as a weak attribute, and its equivalent attribute is identified as the parameter. In this case, we can regard the relationship between this detected parameter and another parameter as the relationship between the true parameter and another parameter since the distinction between parameters and weak attributes is evaluated by dimension attribute aspect metrics.

- **Same level relationship**: We verify if we detect the correct potential same level relationships. So here, the elements are the same level relationships. For the detected binary relationships, we consider the equivalent attributes as having the same level relationships. We also consider an attribute and its neighboring determinant attributes which do not determine any other attribute as having the same level relationships. We consider these potential same level relationships instead of the truly detected same level attributes because the same level attributes are detected based on (1) the same level relationships and (2) the distinction of the parameters and weak attributes. However, the distinction of the parameters and weak attributes is evaluated by dimension attribute aspect metrics.

To evaluate the efficiency, we test the run time of the dimension detection process for each dataset.

7.3 Experimental results and analysis

The ground truth schema and detected schema of each DW are shown in Appendix A.

7.3.1 Dimension aspect effectiveness

Table II.7 shows the results of the dimension aspect effectiveness. We can observe that the precision, the recall and the F-score are all 100% with respect to the effectiveness of the detection of dimension IDs and dimension attributes. Thus, our algorithm is able to correctly detect all dimensions and is able to detect the correct attributes for each dimension.

7.3.2 Dimension attribute aspect effectiveness

Table II.8 shows the results of the dimension attribute aspect effectiveness. For the datasets **Example**, **DevApp** and **Countries**, the effectiveness metrics are all 100%, which means that the parameters and weak attributes are correctly distinguished. For the

Table II.7: Dimension ID aspect results

Dataset	Element	Precision (%)	Recall (%)	F-score (%)
Example	Dimension ID	100.00	100.00	100.00
	Attribute (D1)	100.00	100.00	100.00
	Attribute (D2)	100.00	100.00	100.00
	Attribute (D3)	100.00	100.00	100.00
Sales1	Dimension ID	100.00	100.00	100.00
	Attribute (D1)	100.00	100.00	100.00
Sales2	Dimension ID	100.00	100.00	100.00
	Attribute (D1)	100.00	100.00	100.00
	Attribute (D2)	100.00	100.00	100.00
	Attribute (D3)	100.00	100.00	100.00
	Attribute (D4)	100.00	100.00	100.00
	Attribute (D5)	100.00	100.00	100.00
DevApp	Dimension ID	100.00	100.00	100.00
	Attribute (D1)	100.00	100.00	100.00
Countries	Dimension ID	100.00	100.00	100.00
	Attribute (D1)	100.00	100.00	100.00
Covid	Dimension ID	100.00	100.00	100.00
	Attribute (D1)	100.00	100.00	100.00
	Attribute (D2)	100.00	100.00	100.00

other two datasets whose parameters and weak attributes are not correctly distinguished, we study the original data to find the reasons.

In the dataset **Sales1**, according to the domain knowledge, there may be several postal codes for a city. Therefore, the attribute *Postal Code* should be a weak attribute of the identifier *Customer ID* and *City* should be the neighboring higher parameter of *Customer ID* in the geographical hierarchy. However, in this dataset, there is one postal code value for a city, so the algorithm consider *Postal Code* and *City* as equivalent attributes. Since *Postal Code* contains "Code", it is identified as the neighboring higher parameter of *Customer ID* and *City* is identified as its weak attribute.

In the dataset **Sales2**, there is the same problem of *Postal Code* and *City*. Moreover, the attribute *Product name* is supposed to be the weak attribute of the identifier *Product ID*. However, there exists the functional dependency of $Product\ Name \rightarrow SubCategory$, which makes it become a parameter in the hierarchy of category where it should not be.

In the dataset **DevApp**, the attribute *Suffix* is the type of road for *Address*, which should be a weak attribute of the dimension identifier *APNO*. However, it is detected as the highest level attribute and is a categorical data. It is thus wrongly detected as a parameter.

In the dataset **Covid**, the attribute *indicatordescription* should be a weak attribute of the identifier *indicator*. However, besides the functional dependency of *indicator* $\rightarrow$

indicatordescription, there is also a functional dependency of *indicatordescription* → *indicatortopic*, which make it become a parameter in a hierarchy. In addition, the attribute *unitmeasure* should be a weak attribute of the identifier *indicator* according to the domain knowledge. But it is a categorical data and is thus identified as a parameter.

Table II.8: Dimension attribute aspect results

Dataset	Element	Precision (%)	Recall (%)	F-score (%)
Example	Parameter	100.00	100.00	100.00
	Weak attribute	100.00	100.00	100.00
Sales1	Parameter	83.33	100.00	90.91
	Weak attribute	100.00	50.00	66.67
Sales2	Parameter	92.86	100.00	96.30
	Weak attribute	100.00	66.67	80.00
DevApp	Parameter	85.71	100.00	92.31
	Weak attribute	100.00	80.00	88.89
Countries	Parameter	100.00	100.00	100.00
	Weak attribute	100.00	100.00	100.00
Covid	Parameter	77.78	100.00	87.50
	Weak attribute	100.00	60.00	75.00

7.3.3 Relationship aspect effectiveness

Table II.9 shows the results of the relationship aspect effectiveness. For the datasets **Example** and **Countries**, the effectiveness metrics are all 100%, which means that the hierarchical relationships and the same level relationships are correctly detected. For the other datasets, the wrongly detected relationships are generated due to the same reason as explained for the dimension attribute aspect effectiveness.

Table II.9: Relationship aspect results

Dataset	Element	Precision (%)	Recall (%)	F-score (%)
Example	Hierarchical	100.00	100.00	100.00
	Same level	100.00	100.00	100.00
Sales1	Hierarchical	100.00	100.00	100.00
	Same level	50.00	66.67	57.14
Sales2	Hierarchical	87.50	100.00	93.33
	Same level	66.67	80.00	72.73
DevApp	Hierarchical	83.33	100.00	90.91
	Same level	100.00	83.33	90.91
Countries	Hierarchical	100.00	100.00	100.00
	Same level	100.00	100.00	100.00
Covid	Hierarchical	57.14	80.00	66.67
	Same level	100.00	75.00	85.71

Table II.10: Run time results

Dataset	**Example**	**Sales1**	**Sales2**	**DevApp**	**Countries**	**Covid**
Run Time (s)	1.26	0.93	40.67	102.19	0.05	105.07

7.3.4 Efficiency

Table II.10 shows the run time of the dimension detection for each dataset. We can see that the run time for the applied datasets ranges from 0.05s to 105.07s, which is enough efficient for the users.

8 Conclusion

Tabular data do not have specific schema and particular layout, making it hard to perform data-driven automatic DW design. However, by analysing the literature, we observed that few approaches consider tabular data without schema and their solutions have several limits. Thus in this chapter, we proposed an automatic DW design process from tabular data. For tabular data of complex structure, their structure can be identified or can be transformed to simple structure tabular data by existing algorithms. Therefore, in our process, we mainly focus on tabular data of simple structure. Our process includes the machine learning-based measure detection, functional dependency-based dimension detection and rule-based parameter and weak attribute distinction.

Our solution is able to treat the various challenges that we analysed. To solve the challenges of measure detection, we proposed a machine learning-based method by defining three categories of features for numerical columns. Compared to the other approaches in the literature, which mostly simply select the numerical attributes as measures, the advantage of our solution is that the training of machine learning models using these features allows to capture some characteristics of measures. We carried out experiments with numerous real-world csv tables coming from different sources and divers domains. We applied four classical machine learning classifiers and two baseline approaches. From the results, we observe that the machine learning classifiers applying the proposed features outperforms the baseline approaches. The random forest algorithm performs best among all ML algorithms which reaches a F-score of 93.65% and has an augmentation of F-score of up to 17.2% with respect to the baseline methods, which means that it is able to correctly detect more measure. The model generality with respect to different sources and domains was also verified in our experiments. The results show that the model trained with our proposed features also works well for the data having different source and domain from the training data. Moreover, in our experiments, the feature importance of each feature in each ML algorithm was also analysed. The results help us to understand the features and explain the trained model. The measure detection approach is validated through a paper in the international conference Dawak2022 (**YANG, Y.** et al., 2022a).

To solve the challenges of dimension detection, we proposed a functional dependency-

based approach for building hierarchies. The approach filters the detected FDs and forms FD trees to discover hierarchies. Most approaches from the literature build hierarchies based on one-to-many cardinality relationships for data with schema and based on hierarchical clustering for those without schema. We rely on functional dependencies since the discovery of functional dependencies can help us find one-to-many relationships to derive hierarchies for tabular data without schema. Compared to the hierarchical clustering, the advantage of relying on functional dependencies is that they disclose the indeed hierarchical relationships detected and validated by the instances, while hierarchical clustering can only find semantically correct hierarchical relationships, but which may not be correct at the cardinality level. The distinction of parameters and weak attributes received little attention in the literature and is also considered in our process. The only solution mentioned in the literature is to assign primary keys as parameters, which is only suitable for relational databases. In our context of tabular data, we proposed a rule-based solution dealing with different cases (equivalent attributes and highest-granularity level parameters) and based on the syntax, semantic and data type of data. We conducted experiments by applying one synthetic dataset and five real-world datasets. We validated our approach in terms of effectiveness and efficiency. The effectiveness is evaluated at three aspects containing six levels. The dimension aspect gets 100% for all metrics, while the dimension attribute and relationship aspects have some wrongly detected elements. By analysing the results and datasets, we summarized two main reasons. The first reason is there may be particular cases at the instance level so that the functional dependencies discovered from data may not always conform to the actual hierarchical relationships in the real-world. The second reason is that there are also some particular cases for the distinction of parameter and weak attribute that our rules cannot all cover. The efficiency is validated by the run time and our approach is proved to be efficient. This dimension detection approach is validated through a paper in the national conference EDA2020 (**YANG, Y.** et al., 2020)

It is to be noted that in this chapter, we only discussed the automatic DW design from one single source. However, in real-world cases, a user may have several sources which have some common information. We should thus create a DW for each one of them and then merge them together to carry out consolidated analyses.

Chapter III

Data Warehouse Merging

Contents

1 Introduction

1.1 Context

Data warehouse merging is the process of merging DWs having common information into a unified DW to enable the user to analyse the consolidated data. When we have multiple sources, we can apply the proposed automatic DW design process to generate a DW schema for each source. The implementation of the DWs can be carried out according to the schemas. We can then merged the DWs having common information to analyse data at more complete viewpoints. We do not first merge these different sources and then generate a DW because of two reasons. First, the merged table may contain too many attributes, which may give rise to functional dependencies that do not semantically correct. Second, too many missing values may be generated when merging the sources, which may impact the results of the detected functional dependencies (Papenbrock and Naumann, 2016).

Moreover, the DW merging is also helpful in general case. For example, in a company, various independent DWs containing some common elements and data may be built for different geographical regions or functional departments. There may also exist common elements and data between the DWs of different companies. The ability to accurately merge diverse DWs into one integrated DW is therefore considered as a major issue (Kwakye et al., 2013). Multidimensional DWs merging constitutes a promising solution to have more opportunities of analysing the consistent data coming from these different sources. Automating the DW merging process can facilitate the tasks of the DW designers. It can make the DWs merged more efficiently for decision-makers. Companies can thus gain benefits at both time and cost aspects. As a result, it is necessary to propose an automatic DW merging process.

1.2 Challenges of DW merging

Merging two multidimensional DWs is a complex task which should answer some problems. The first consists in identifying the common basic components (dimension attributes, measures) and defining semantic relationships between these components. The second is to merge schemas which have common components. But merging two multidimensional DWs is difficult because two dimensions can (1) be completely identical in terms of schema but not necessarily of instances, (2) have common hierarchies or have sub-parts of hierarchies in common without necessarily sharing common instances. Likewise, two schemas can deal with the same fact or different facts and even if they deal with the same, they may or may not have measures in common without necessarily having common data.

Moreover, the final merged DW should respect the constraints of the original multidimensional elements especially the hierarchical relationships between attributes. When we merge two dimensions having matched attributes of two DWs, the final DW should preserve all the partial orders of the original hierarchies (i.e. the binary relationships of

aggregation between parameters) of these two dimensions. It's also necessary to integrate all the original instances of the DWs, which may cause the generation of empty values in the final DW. Thus, the merging approach should be able to allow the proper analysis with empty values.

Furthermore, the original DWs may have common or different dimensions. Therefore, merged DW may have a star or constellation schema.

1.3 Our Process Overview

As a result, we define in this chapter an automatic approach to merge two multidimensional DWs especially modelled by star schema (i.e. schema containing one fact) at both schema and instance levels, which (1) generates an integrated DW conforming to the multidimensional structures of the original DWs, (2) integrates the original instances into the integrated DW and is compatible with empty values generated during the merging process, (3) generates a star or constellation schema in different cases.

Merging two DWs implies matching steps and steps dedicated to the merging of dimensions and facts. The matching of parameters and measures are based on syntactic and semantic similarities (Meng et al., 2013)(Elavarasi et al., 2014) for the attribute or measure names. Since the matching is intensively studied in the literature, we focus in this paper only on the merging steps like illustrated in Fig III.1. A DW is composed of dimensions and facts, a dimension contains different hierarchies where there are different levels. So in regard to the merging, first, we define an algorithm for merging a matched level of two hierarchies at the schema level. Second, we propose an algorithm of hierarchy merging by applying the level merging at the schema level. Third, we define an algorithm of dimension merging concerning both instance and schema levels and by applying the hierarchy merging. Finally, we define an algorithm of star schema merging which may is based on dimension merging and which generate a star schema or a constellation schema and which merge the fact instances.

1.4 Outline

The remainder of this chapter is organized as follows. In Section 2, we review the related works about the matching and merging of DW. In Section 3, 4, 5 and 6, we explain our proposed automatic approach to merge different DWs including respectively the merging of hierarchy levels, hierarchies, dimensions and facts by giving algorithms concerning both the schema and instance levels. In Section 7, we present our experiments in order to validate our approach. Finally, in Section 8, we conclude this chapter.

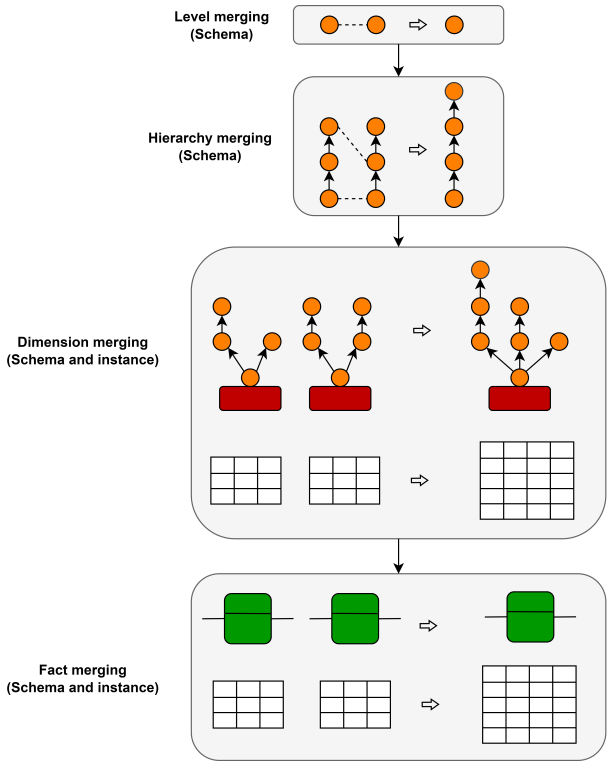


Figure III.1: Overview of the merging process

2 Related Work

The DW merging process concerns matching tasks and merging tasks. The matching task consists in generating correspondences between similar elements (dimension attributes and fact measures) of schemas (Bernstein et al., 2011a) to retrieve links of two DWs. The merging task is more complex. The merging should be carried out at two levels: the schema level and the instance level. Schema merging is the process of integrating several schemas into a common, unified schema (Quix et al., 2007). Thus DW schema merging aims at generating a merged unified multidimensional schema. The instance level merging treats the integration and the handling of the instances. In the following of the chapter, "matching" will be used to designate schema matching without considering instances, while "merging" will be used to refer to the complete merging of schema and the corresponding instances. The general data matching (Rahm and Bernstein, 2001; Bernstein et al., 2011b; Dorneles et al., 2011) and merging (Lin and Mendelzon, 1998; Pottinger and Bernstein, 2003; Bleiholder and Naumann, 2009) techniques are already widely researched in the literature. However, these matching or merging processes are mainly dedicated to the relational database, XML and flat data which are different from DW. They do not have multidimensional structure and constraints, such as the hierarchies in the DW. So we focus on the specific DW matching and merging. The approaches with respect to the DW matching and merging are as follows.

2.1 Multidimensional Schema Matching

(Bergamaschi et al., 2011) propose an approach for matching aggregation levels of DW dimensions. Their technique is based on the fact that the cardinality ratio of two aggregation levels which are in a same hierarchy is nearly always the same no matter in which dimension they are. So they create and manipulate the cardinality matrix for different dimensions to discover the matched attributes. However, this approach only consider the dimension level matching.

A process to automatically match multidimensional schemas is proposed in (Banek et al., 2007). They match two multidimensional schemas by evaluating the semantic similarity of the multidimensional component names. For attributes and measures, they also compare the data type. They use the selection metric of bipartite graph to determine the mapping choice and define rules aiming at preserving the partial orders of hierarchies for the mapping of hierarchies. (Riazati and Thom, 2011) aim to match star schemas by proposing a new representation model of star schema. The model is described by UML and it adds supplementary metadata inferred from the relational schema. Then existing matching systems can be used to match the schemas. In (Elamin et al., 2018), the authors propose an approach for multidimensional schema matching in the context of matching a set of star schemas generated from business requirement and another one generated from the data sources. They use semantic similarity based on Wordnet to find the matched fact and dimension names. The DW designer will intervene to check the set of common facts and manually match the unmatched elements in the other sets.

To summarize, these approaches match multidimensional schemas based on the cardinality ratio, similarity of multidimensional elements or metadata. However, the cardinality based approach is not reliable because (1) to have the similar cardinality ratio between same real-world entities, we should make sure that there are enough categories of the entities and (2) not all pairs of real-world entities have fix cardinality ratio. So the typical similarity-based matching should be used. It is applied in most of the matching systems and is well studied. We thus focus on the merging of multidimensional schemas and instances.

2.2 Multidimensional Schema and Instance Merging

The merging part directs at merging multidimensional DW schemas into one consolidated schema and merging the instances together. The following approaches concern multidimensional schema and instance merging.

2.2.1 Feki et al. (2005)

In this paper, an approach for automatic multidimensional schema merging is proposed. The approach is composed of two phases: (1) transforming multidimensional schemes into UML class diagrams, and (2) merging the UML class diagrams. They define several

rules to explain how to translate each element of a multidimensional schema into an UML class diagram element. Then for the merging of the classes in the UML class diagrams, they propose two linguistic criteria. The first one is based on the class name comparison, they propose four types of relations including equivalence, generalization, composition and variation. The classes' name should be the first three types to be able to be merged. The second criterion is based on the ratio of the common attributes between classes, the classes should have the attribute relationship of equivalent, inclusion or strong intersection to be merged. Finally the classes which satisfy these two types of criteria can be merged into one class.

2.2.2 Torlone (2008)

In (Torlone, 2008), two approaches for merging heterogeneous DWs are proposed. The first one is called "loosely coupled approach" which aims to select shared data between sources. Dimensions having common attributes are merged together and only the intersection of these dimensions are reserved. The other is called "tightly couple approach" which combines the data of different sources by taking the union of the matched dimensions to merge two DWs. The common attributes are merged together, the hierarchies remain the same as the original ones. They do not merge the hierarchies by creating new ones. The instance merging of the two approaches is realized by a d-chase procedure.

2.2.3 Kwakye et al. (2013)

An approach of DW merging at the schema and instance levels is proposed in this paper. They match attributes based on the lexical similarity of schema names and instances and by considering the schema data types and constraints. Having the mapping correspondences, dimensions or facts having the matched attributes are merged together with matched attributes merged together. The instance data are then populated by considering some conflicts. Solutions are also given, such as creating new surrogate keys for the identifier conflicts.

2.2.4 Olaru and Vincini (2014)

In this paper, an approach for merging multidimensional dimensions is proposed. The merging of hierarchies is based on the cardinality ratio between different dimension levels. They suppose that the cardinality ratio between same real-world concepts is approach. Thus they model hierarchies as directed labeled graphs and create a connectivity matrix whose values are cardinality ratios to find equivalent levels as well as drill-down and roll-up relationships of the levels to merge dimensions. The merging at the instance level is realized by clustering the data based on their semantic and syntactic similarities.

2.3 Analysis of Merging Approaches

Table III.1 shows the comparison of the different DW merging approaches. We compare these approaches in three aspects: merging level, schema type and considered multidimensional element.

2.3.1 Merging Level

Regarding the merging level aspect, we analyse whether the approaches consider the merging at both schema and instance levels or at only one level. We can see that (Feki et al., 2005) merge DWs at only schema level while the other approaches counter both schema and instance levels.

2.3.2 Schema Type

Regarding the schema type aspect, we compare the input and output schema type of each approach. We can observe that the input schema of the four approaches are all star schema except that (Olaru and Vincini, 2014) only focus on the dimensions. So the output of (Olaru and Vincini, 2014) is also star schemas with merged dimensions. (Feki et al., 2005) obtain a UML class diagram as output since they transform the star schemas into UML class diagrams for the merging. However, the UML representation is not an universal model of star schema and it should be retransformed into a star schema. The outputs of (Torlone, 2008) and (Kwakye et al., 2013) are respectively a constellation schema and a star schema. However, the merged output schema may not always be one form, it may be a constellation or a star schema according to the link between the original facts and dimensions.

2.3.3 Multidimensional Element

Regarding the multidimensional element aspect, we evaluate if each approach takes into account the merging of all possible multidimensional elements including fact, dimension, hierarchy and weak attribute. Only (Feki et al., 2005) consider all these elements, since they all can be represented in the UML class diagrams. None of the other three approaches consider the merging of weak attributes. (Torlone, 2008) and (Olaru and Vincini, 2014) do not consider the merging of fact tables and only concern about dimensions. The hierarchy merging is a tough task as we mentioned in Section 1.1. The approach of (Kwakye et al., 2013) does not include the hierarchy merging. In (Feki et al., 2005) and (Torlone, 2008), only equivalent levels are merged together, but they do not consider the merging of the other levels by detecting their possible hierarchical relationships. (Olaru and Vincini, 2014) use cardinality ratio for the hierarchy merging which we do not believe reliable as we argued in Section 2.1.

Table III.1: Comparison of different approaches

	Merging Level		Schema Type		Mutidimensional Element			
	Schema	**Instance**	**Input**	**Output**	**Fact**	**Dimension**	**Hierarchy**	**Weak Attribute**
Feki et al. (2005)	✓	-	Star schemas	UML class diagram	✓	✓	✓	✓
Torlone (2008)	✓	✓	Star schemas	Constellation schema	-	✓	✓	-
Kwakye et al. (2013)	✓	✓	Star schemas	Star schema	✓	✓	-	-
Olaru and Vincini (2014)	✓	✓	Star schema dimensions	Star schema dimensions	-	✓	✓	-

2.3.4 Analysis Conclusion

After analysing these approaches, we observe that all the of approaches have the problem of being incomplete in term of the merging level or in terms of the multidimensional elements. None of the approaches proposes an appropriate hierarchy merging technique. Moreover, none of the approaches generates an output schema may be a star or constellation schema.

3 Level Merging

We first discuss about the level merging of two hierarchies at the schema level. When two parameters of two hierarchies are matched, they can be merged into one parameter. They also represent the same granularity level. So we should then merge their weak attributes. Algo. 4 describes the level merging process. We first define an ordered set of map which will save the matched parameters and the merged weak attributes of the merged level ($line_1$). Then we loop through each parameters of the two hierarchies ($line_{2-3}$) and process two steps composed of the record of matched parameters and the merging of weak attributes.

3.1 Record of Matched Parameters

The first step consists in finding the matched parameters of the two hierarchies ($line_4$) and record the matched parameters of H_1 and H_2 into the map ($line_{6-7}$). The matched parameters will later be used for the hierarchy merging.

3.2 Merging of Weak Attributes

For the merged parameters, they may contain different weak attributes. So we have to merge their weak attributes. The merging of the weak attributes is to take their union: each two matched weak attributes are merged together into one weak attribute; the merged weak attributes and the other non-matched weak attributes constitute the merged weak attribute set of the merged parameter ($line_7$). It is then added into the map ($line_8$). We can thus update the ordered set of map M ($line_9$).

Example 3.1. *In Fig. III.10, when merging H_1 and H_3, for the matched parameters $H_1.IdCus$ and $H_3.IdCus$, they can be merged together. For the weak attributes $\{NameCus$,*

Algorithm 4: $mergeLevel(H_1, H_2)$

Input : Two hierarchies to be merged H_1, H_2
Output: An ordered set of map containing of matched parameters and merged weak attributes M

```
M ← ∅;
for p_i^{H1} ∈ Param^{H1} do
    for p_j^{H2} ∈ Param^{H1} do
        if p_i^{H1} ≃ p_j^{H2} then
            mapM['p'_1] = p_i^{H1};
            mapM['p'_2] = p_j^{H2};
            Weak_12 ← Weak^{H1}[p_i^{H1}] ∪ Weak^{H2}[p_j^{H2}];
            mapM['weak'] = Weak_12;
            M ← M + mapM;
return M;
```

Age, Email} *of* H_1 *and* {*NameCus, Phone*} *of* H_3*, we have* $H_1.NameCus \simeq H_3.NameCus$*, they are merged into one weak attribute. So we can get the merged weak attribute set:* {*NameCus, Age, Email, Phone*}*.*

The merging example with weak attributes at the schema and instance level are only shown in Fig. III.8 and in Fig. III.10. As we already know how to merge weak attributes of matched parameters, in the following algorithms and examples, for a hierarchy level, we only keep the parameter for the simplicity.

4 Hierarchy Merging

In this section, we define the schema merging process of two hierarchies coming from two different dimensions. There are two challenges in this process. The first challenge is that we should preserve the partial orders of the parameters. The second one is how to decide the partial orders of the parameters coming from different original hierarchies in the merged hierarchies. These challenges are solved in algorithm 5 which is achieved by 3 steps: generation of sub-hierarchy pairs, merging of sub-hierarchy pairs and generation of final hierarchy set.

A sub-hierarchy is a continuous sub-part of a hierarchy which we call the parent hierarchy of the sub-hierarchy. This concept will be used in our algorithm. A sub-hierarchy has the same elements as a hierarchy, but its root parameter is not considered as an identifier. All parameters of a sub-hierarchy are contained in its parent hierarchy and have the same partial orders than those in the parent hierarchy. By "continuous", we mean that in the parameter set of the parent hierarchy of a sub-hierarchy, between the lowest- and highest-granularity level parameters of the sub-hierarchy, there is no parameter which is in the parent hierarchy but not in the sub-hierarchy. We give the following formal definition for

sub-hierarchy.

Definition 4.1 (Sub-hierarchy)**.** *A sub-hierarchy SH of a hierarchy $H \in H^D$ in a dimension D is defined as $SH = Sub(H, p_1^{SH}, p_v^{SH}) =< p_1^{SH}, ..., p_v^{SH} >$ which is an ordered set of parameters, $\forall k \in [1...v], p_k^{SH} \in Param^H$. According to the relationship between a sub-hierarchy and its parent hierarchy, we have:*

1. $\forall p_1^{SH}, p_2^{SH} \in SH,\ p_1^{SH} \preceq_{SH} p_2^{SH} \Rightarrow p_1^{SH}, p_2^{SH} \in Param^H \wedge p_1^{SH} \preceq_H p_2^{SH}$,
2. $\forall p_1^H, p_2^H, p_3^H \in Param^H,\ p_1^H \preceq_H p_2^H \wedge p_2^H \preceq_H p_3^H \wedge p_1^H, p_3^H \in SH \Rightarrow p_2^H \in SH$.

In Algo. 5, we first call the algorithm of level merging to find the matched parameters ($line_1$). If there is no matched parameter ($line_2$), the merged results will be two hierarchy sets containing respectively the two original hierarchies ($line_{3-5}$). In the case where there are matched parameters, we can carry out the merging of the two hierarchies. We then explain each step of the hierarchy merging.

4.1 Generation of Sub-hierarchy Pairs

The algorithm generates pairs containing 2 sub-hierarchies (SH_1 and SH_2) of the original hierarchies whose lowest and highest level parameters are adjacent in the ordered set M that we obtain ($line_{12-14}$). The last parameters of the two hierarchies are the last parameters of the last sub-hierarchy pair. However, if they are not matched, they are not added into M so that we are not able to create the last sub-hierarchy pair. So in this case where the last parameters of the two hierarchies do not match, they are add into M ($line_{8-11}$).

Example 4.1. *In Figure III.2a, we have two hierarchies H_1 and H_2, and $H_1.IdCus \simeq H_2.IdCus$, $H_1.City \simeq H_2.City$, $H_1.Country \simeq H_2.Country$. So for the first sub-hierarchy pair, the first parameter of SH_1^1 and SH_2^1 is $IdCus$ and their last parameter is $City$, thus we have: $SH_1^1 =< IdCus, City >$, $SH_2^1 =< IdCus, City >$. In the second sub-hierarchy pair, we get the sub-hierarchy of H_1 from $City$ to $Country$: $SH_1^2 =< City, Region, Country >$, and the sub-hierarchy of H_2 from $City$ to $Country$: $SH_2^2 =< City, Country >$. In Figure III.2b, we also have the hierarchy H_1, and we have another hierarchy H_3. The matched parameters are $H_1.IdCus \simeq H_3.IdCus$, $H_1.Region \simeq H_3.Region$ and $H_1.Country \simeq H_3.Country$. So we get the first parameter pair $Sh_1^1 =< IdCus, City, Region >$, $SH_3^1 =< IdCus, Deparment, Region >$ and the second parameter pair $SH_1^2 =< Region, Country >$, $SH_3^2 =< Region, Country >$. The last parameters of H_1 and H_3 do not match, $< Country, Continent >$ is thus added into the matched parameter pair M of the algorithm so that the last sub-hierarchies of H_1 and H_3 are $SH_1^3 =< Country >$ and $SH_3^3 =< Country, Continent >$.*

Algorithm 5: $mergeHierarchies(H_1, H_2)$

Input : Two hierarchies to be merged H_1, H_2
Output: A set of merged hierarchies H_m or two sets of merged hierarchies H_{m1} and H_{m2}

$M \leftarrow mergeLevels(H_1, H_2)$;
if $M = \emptyset$ **then**
  $H_{m1} \leftarrow \{H_1\}$;
  $H_{m2} \leftarrow \{H_2\}$;
  return H_{12}, H_{21}
else
  $H_m \leftarrow \emptyset$;
  $mapM_{last}[p'_1] = Param^{H_1}[|Param^{H_1}| - 1]$;
  $mapM_{last}[p'_2] = Param^{H_2}[|Param^{H_2}| - 1]$;
  if $mapM_{last}[p'_1] \neq mapM_{last}[p'_2]$ **then**
    $M \leftarrow M + mapM_{last}$;
  for $i = 1$ *to* $|M| - 1$ **do**
    $SH_1^i \leftarrow Sub(H_1, [M[i-1][p'_1], M[i][p'_1]])$;
    $SH_2^i \leftarrow Sub(H_2, [M[i-1][p'_2], M[i][p'_2]])$;
    if $SH_1^i \subseteq SH_2^i$ **then**
      $SH_{12}^i \leftarrow \{SH_2^i\}$;
    else if $SH_2^i \subseteq SH_1^i$ **then**
      $SH_{12}^i \leftarrow \{SH_1^i\}$;
    /* $FD(SH_1^i, SH_2^i)$ returns FDs of the parameters of SH_1^i, SH_2^i */
    else if $FD(SH_1^i, SH_2^i) \neq FD(SH_1^i) \cup FD(SH_2^i)$ **then**
      $SH_{12}^i \leftarrow \emptyset$;
      for $SH_a \in getAllHierarchies(FD(SH_1^i, SH_2^i))$ **do**
        $SH_{12}^i \leftarrow SH_{12}^i + SH_a$;
    else
      $SH_{12}^i \leftarrow \{SH_1, SH_2\}$;
    $H_m \leftarrow \{H_a.extend(SH_b) : H_a \in H_m, SH_b \in SH_{12}^i\}$;
  if $id^{D_1} \simeq id^{D_2}$ **then**
    $H_m \leftarrow H_m \cup \{H_1, H_2\}$;
    return H_m
  else
    $SH_1^0 \leftarrow Sub(H_1, p_1^{H_1}, M[0][p'_1])$;
    $SH_2^0 \leftarrow Sub(H_2, p_1^{H_2}, M[0][p'_2])$;
    $H_{m1} \leftarrow \{SH_1^0.extend(H_b) : H_b \in H_m\} \cup \{H_1\}$;
    $H_{m2} \leftarrow \{SH_2^0.extend(H_b) : H_b \in H_m\} \cup \{H_2\}$;
    return H_{m1}, H_{m2}

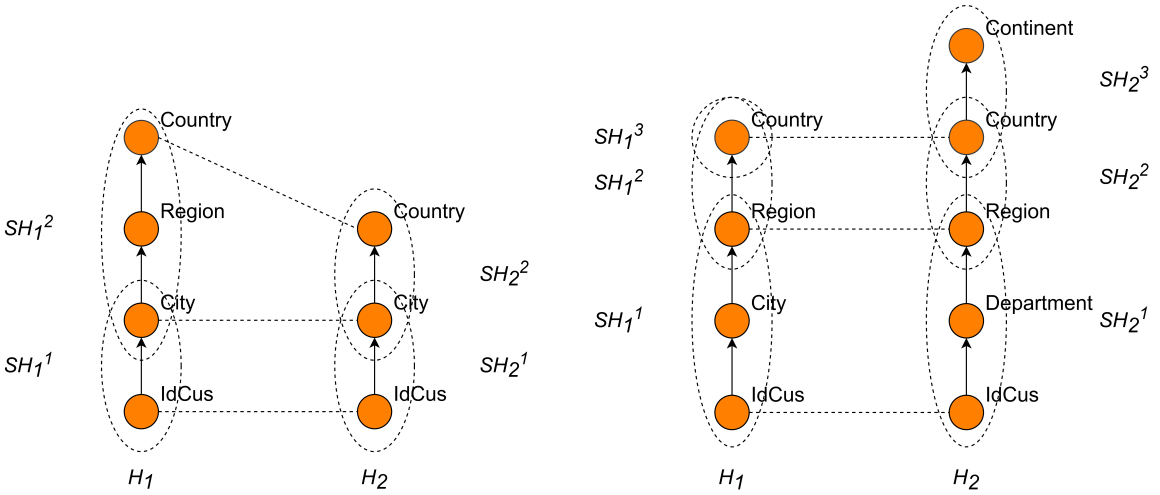


Figure III.2: Example of generation of sub-hierarchy pairs

4.2 Merging of Sub-hierarchies

We then merge each sub-hierarchy pair to get a set of merged sub-hierarchies and combine each of these sub-hierarchy sets to get a set of merged hierarchies ($line_{15-25}$). The matched parameters will be merged into one parameter, so it's the unmatched parameters that we should deal with. We have 2 cases in terms of the unmatched parameters.

First Case In the sub-hierarchy pair, if one of the sub-hierarchies has no unmatched parameter, we obtain a sub-hierarchy set containing one sub-hierarchy whose parameter set is the same as the other sub-hierarchy ($line_{15-18}$).

Example 4.2. *For the first parameter pair SH_1^2 and SH_2^2 of H_1 and H_2 in Fig. III.2a. we see that SH_2^2 does not have any unmatched parameter, so the obtained sub-hierarchy set contains one sub-hierarchy whose parameter set is the same as SH_1^2 which is $SH_{12}^2 =< City, Region, Country >$.*

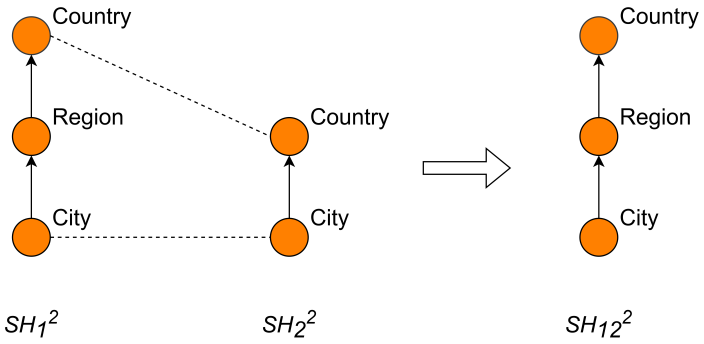


Figure III.3: Example of hierarchy merging

Second Case The second case is that both two sub-hierarchies have unmatched parameters ($line_{19-25}$). We then see if these unmatched parameters can be merged into

one or several hierarchies and discover their partial orders. Our solution is also based on the functional dependencies (FDs) of these parameters. To be able to detect FDs of the parameters of the two sub-hierarchies, we should make sure that there are intersections between the instances of these two sub-hierarchies which means that they should have same values on the root parameter of the sub-hierarchies. If it is able to detect FDs, and we can discover new FDs apart from the original FDs ($line_{19}$), we can apply the Algo. 2 in the Chapter II to get a set of sub-hierarchies ($line_{20-22}$). If it is not possible to discover FDs, the two sub-hierarchies are impossible to be merged and we obtain a sub-hierarchy set containing the two original sub-hierarchies ($line_{23-24}$).

Example 4.3. *For the first sub-hierarchy pair SH_1^1 and SH_2^1 of H_1 and H_2 in Fig. III.2b, we suppose that their instances are like presented in Figure III.4. There is an unmatched parameter City in SH_1^1 and an unmatched parameter Department in SH_3^1. We have to decide their partial order. So we take the intersection of their instance which is the dashed framed part in Fig. III.4. By detecting FDs, we can find the relationship City $\rightarrow$ Departement. So we obtain $SH_{12}^1 =< IdCus, City, Department, Region >$.*

IdCus	City	Region	Country
C1	CT1	R1	CN1
C2	CT2	R1	CN1
C3	CT2	R1	CN1
C4	CT3	R3	CN2
C5	CT5	R2	CN1
C6	CT4	R3	CN2
C7	CT3	R3	CN2

Instance of H_1

IdCus	Department	Region	Country	Continent
C1	D1	R1	CN1	CTN1
C2	D3	R1	CN1	CTN1
C3	D3	R1	CN1	CTN1
C4	D4	R3	CN2	CTN1
C6	D4	R3	CN2	CTN1
C8	D2	R2	CN1	CTN1
C9	D5	R4	CN3	CTN2

Instance of H_3

IdCus	City	Department	Region
C1	CT1	D1	R1
C2	CT2	D3	R1
C3	CT2	D3	R1
C4	CT3	D4	R3
C6	CT4	D4	R3

Instance of intersection of SH_1^1 and SH_3^1

Figure III.4: Example of hierarchy instance

After the merging of each sub-hierarchy pair, we extend the final merged hierarchy set by the new merged result ($line_{25}$).

4.3 Generation of Final Hierarchy Set

The generation of the final hierarchy set is depicted in $line_{26-34}$. The two original hierarchies may have different instances, so there may be empty values in the instances of the merged hierarchies. Some empty values can be replaced, which will be presented in

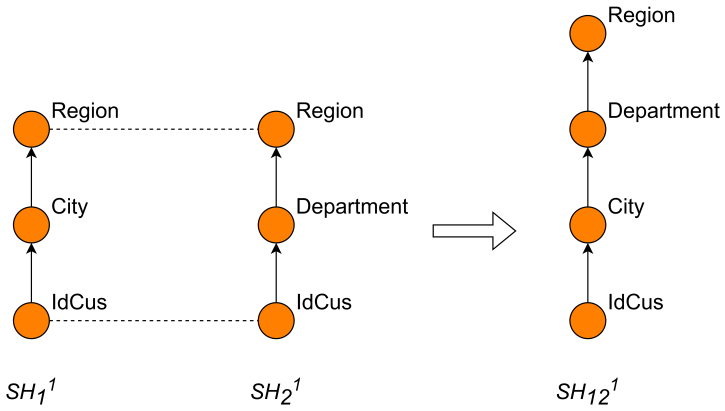


Figure III.5: Example of hierarchy merging

the next chapter. But not all empty values can be replaced. The empty values generate incomplete hierarchies and make the analysis difficult. Inspired by the concept of structural repair (Ariyan and Bertossi, 2011), we also add the two original hierarchies into the final hierarchy set. Then for a parameter which appears in different hierarchies, it can be divided into different parameters in different hierarchies of the hierarchy set so that each hierarchy is complete. Thus, for the multidimensional schema that we obtain, we provide 2 forms: database form and analysis form like shown in Figure III.6. In the database, the data of a parameter is actually stored in one column, so in the database form, one parameter appears only once in the schema. However it can be regarded as different parameters in the schema, so in the analysis form, one parameter can be marked with different numbers if it is in different hierarchies. We use the analysis form to present the merged schema in this paper to clearly present the completeness of the hierarchies.

For the generation of the final hierarchy set, we discuss 2 cases where the 2 hierarchies have the matched root parameters which means their dimensions are the same analysis axis and the opposite case which leads to 2 kinds of output results (one or two sets of merged hierarchies).

First Case If the root parameters of the two original hierarchies match, we simply add the two original hierarchies into the merged hierarchy set obtained in the previous step to get one final merged hierarchy set to guarantee the completeness of the hierarchies. ($line_{26-28}$).

Example 4.4. *For H_1 and H_2 in Fig. III.6, we also suppose that the instances are like shown in Fig. III.4. In Example 4.3, we get $SH_{12}^1 =< IdCus, City, Department, Region >$. By the above defined sub-hierarchy merging rules, we can also get $SH_{12}^2 =< Region, Country >$ and $SH_{12}^3 =< Country, Continent >$. We combine these merged sub-hierarchies to obtain the merged hierarchy $H_{12} =< Code, City, Department, Region, Country, Continent >$. We add H_{12} into the hierarchy set H_m and then also add the original hierarchies H_1 and H_2. Thus H_m is the final merged hierarchy set.*

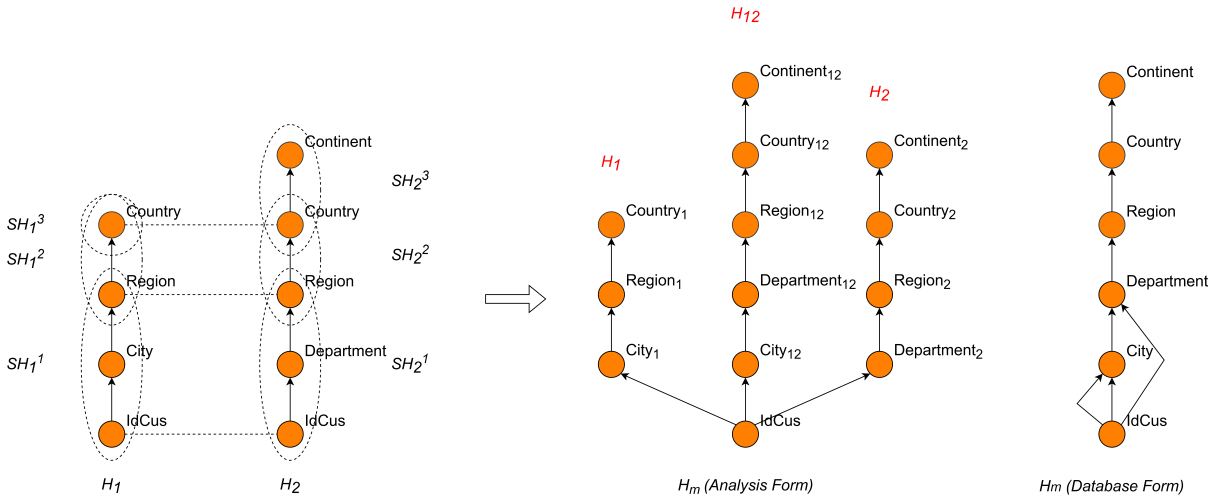


Figure III.6: Hierarchy merging example

Second Case If the root parameters of the two original hierarchies do not match, we get two merged hierarchy sets instead of one. For each original hierarchy, the final merged hierarchy set is the extension of the sub-hierarchy containing all the parameters which are not included in any one of the sub-hierarchies created before with the merged hierarchy set that we obtain, plus this original hierarchy itself ($line_{30-34}$).

Example 4.5. *In Fig. III.7, H_1 is the same as the H_1 in Example 4.4, and H_2 is a hierarchy which has a different root parameter from H_1 and which has a parameter Region matched with $H_1.Region$. So in this example, the root parameters do not match. We thus have $SH_1^1 =< Region, Country >$ and $SH_2^1 =< Region, Continent >$. Then, we can merge each sub-hierarchy pair and combine the results to get the merged sub-hierarchy $SH_{12}^1 =< Region, Country, Continent >$. For H_1, the remaining part $< IdCus, City >$ is associated to it to get the merged hierarchy H_{12}. We then also add the original hierarchy H_1 to get the merged hierarchy set H_{m1} of containing H_1 and H_{12}. We do the same thing for H_2 and get the merged hierarchy set containing H_2 and H_{21}.*

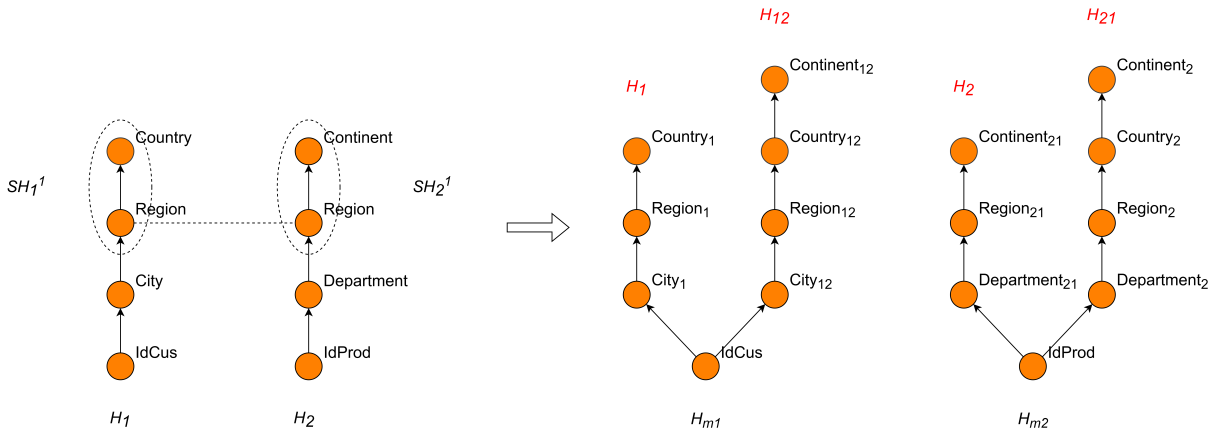


Figure III.7: Hierarchy merging example

5 Dimension Merging

This section concerns the merging of two dimensions having matched parameters which is realized by Algo. 6. We consider both the schema and instance levels for the merging of dimensions. The schema merging is based on the merging of hierarchies. Concerning the instances, we have 2 tasks: merging the instances and replacing the empty values which we will discuss in Chapter IV.

Algorithm 6: $mergeDimensions(D_1, D_2)$

Input : Two dimensions D_1, D_2 to be merged
Output: One merged dimension D_{12} or two merged dimensions D_{12} and D_{21}

```
1  if id^{D1} ≃ id^{D2} then
2      H^{D12} ← ∅;
3      for H_i ∈ H^{D1} do
4          for H_j ∈ H^{D2} do
5              H^{D12} ← H^{D12} ∪ MergeHierarchies(H_i, H_j);
6      A^{D12} ← A^{D1} ∪ A^{D2};
7      I^{D12} ← I^{D1} ∪ I^{D2};
8      return D_12
9  else
10     H^{D12}, H^{D21}, A^{D12}, A^{D21} ← ∅;
11     for H_i ∈ H^{D1} do
12         for H_j ∈ H^{D2} do
13             H_m1, H_m2 ← MergeHierarchies(H_i, H_j);
14             H^{D12} ← H^{D12} ∪ H_m1;
15             H^{D21} ← H^{D21} ∪ H_m2 ;
16     for H_u ∈ H^{D12} do
17         A^{D12} ← A^{D12} ∪ Param^{H_u};
18     for H_v ∈ H^{D21} do
19         A^{D21} ← A^{D21} ∪ Param^{H_v};
20     return D_12, D_21
```

5.1 Schema Merging

The root parameters of two original dimensions may be matched or unmatched. Thus we discuss the schema merging for these two cases.

First Case If the root parameters of the two dimensions match, the algorithm generates a merged dimension ($line_{1-6}$). The hierarchy set of the merged dimension is the union of the hierarchy sets generated by merging every 2 hierarchies of the original dimensions ($line_{1-8}$). The attribute set of the merged dimension is the union of the attribute sets of the original dimensions ($line_6$).

Example 5.1. *Given 2 original dimensions D_1 and D_2 in Figure III.8 and their instances in Figure III.10, we can get the merged dimension schema D_{12} in Figure III.8. In D_{12}, H_1 and H_2 are the original hierarchies of D_1, H_3 and H_4 are those of D_2, H_{13} is a merged hierarchy of H_1 and H_3, and H_{24} is a merged hierarchy of H_2 and H_4. We can thus get $H^{D_{12}} = \{H_1, H_2, H_3, H_4, H_{13}, H_{24}\}$.*

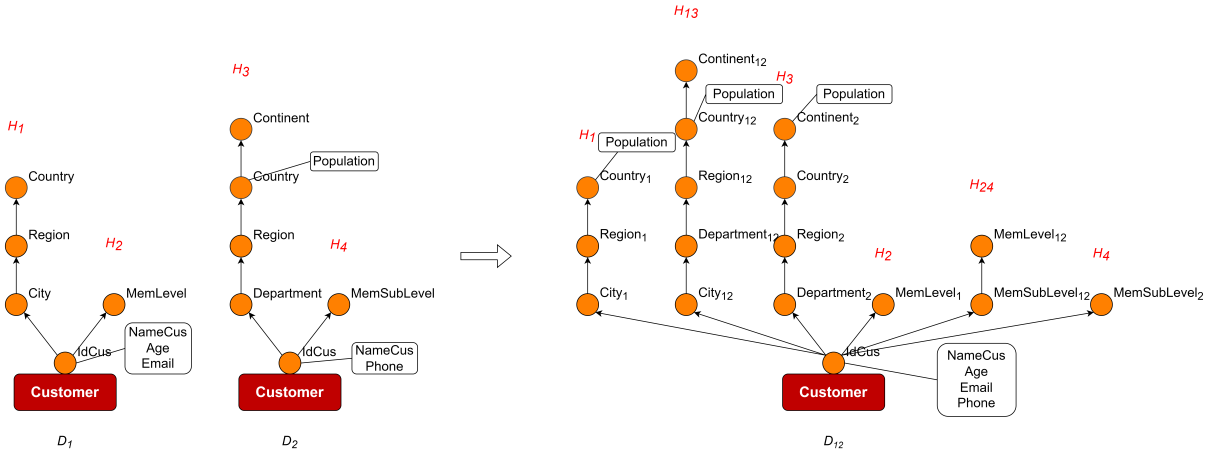


Figure III.8: Dimension merging example (schema)

Second Case When the root parameters of the two dimensions do not match, we get a merged dimension for each original dimension, which is realized by $line_{10-22}$. For each original dimension, the hierarchy set of its corresponding merged dimension is the union of all hierarchy sets generated by merging every 2 hierarchies of the original dimensions ($line_{11-15}$), the attribute set is the union of the attributes of each hierarchy in the merged dimension ($line_{18}, line_{20}$).

Example 5.2. *Given 2 original dimensions D_1 and D_2 in Figure III.9 and their instances in Figure III.11, after the execution of Algo. 6, we can get the merged dimension schema D_{12} and D_{21} in Figure III.9. In D^{12}, H_1 and H_2 are the original hierarchies of D_1, H^1_{13} is the merged hierarchy of H_1 and H_3. In D^{21}, H_3 is the original hierarchy of D_2, H^2_{13} is the merged hierarchy of H_1 and H_3. So for D_1, we have $H^{D^{12}} = \{H_1, H_2, H_{13}\}$. For D_2, we get $H^{D^{21}} = \{H_3, H_{31}\}$.*

5.2 Instance Merging

For instance merging, we also discuss the cases where the root parameters of the original dimensions match or not.

First Case When the root parameters of the two dimensions match, the instances of the merged dimension are obtained by the union of the two original dimension instances which means that we insert the data of the two original dimension tables into the merged dimension table and merge the tuples which have the same root parameter instance ($line_7$).

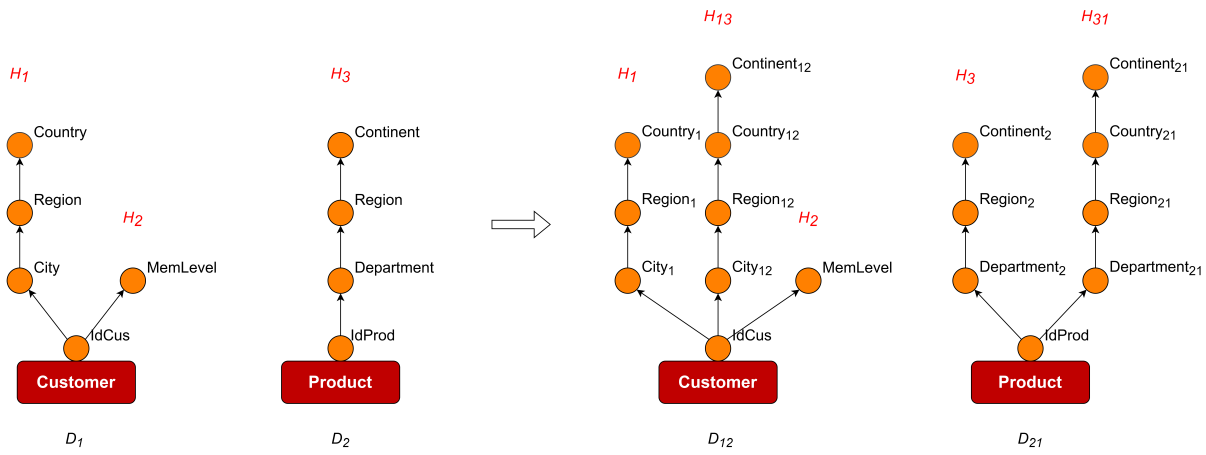
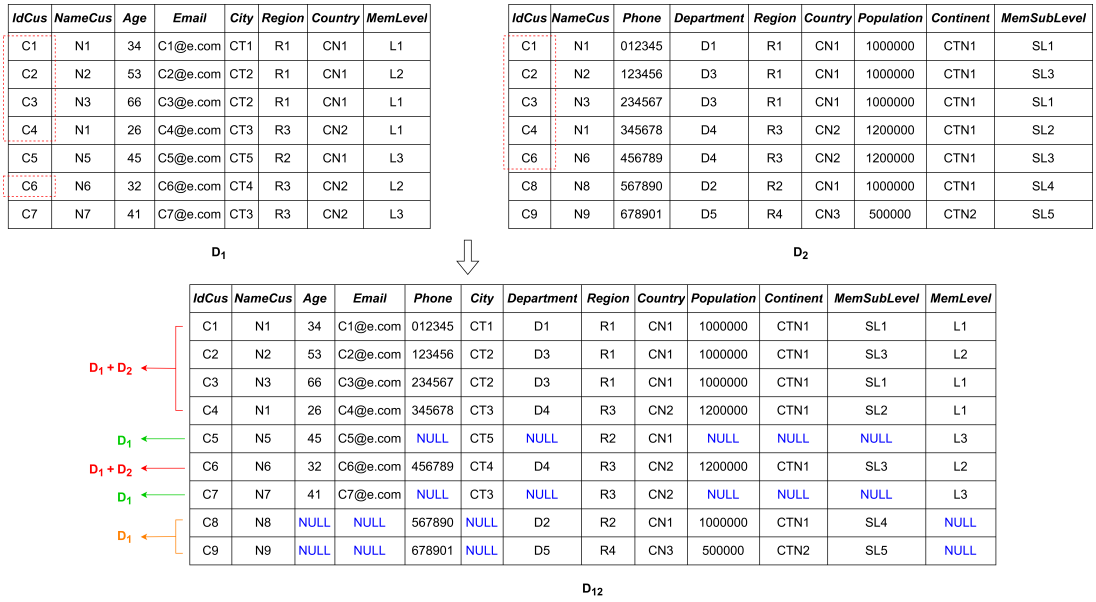


Figure III.9: Dimension merging example (schema)

Example 5.3. *For the dimension schema merging example in Example 5.1, the instance merging result is presented in Figure III.10. All the data in the original dimension tables D_1, D_2 are integrated into the merged dimension table D_{12}. The original tables of the instances are marked on the left of the merged table D_{12} with different colors. There are instances coming from both D_1 and D_2, which means that they have the same root parameter in D_1 and D_2, and are therefore merged together.*



IdCus	NameCus	Age	Email	City	Region	Country	MemLevel
C1	N1	34	C1@e.com	CT1	R1	CN1	L1
C2	N2	53	C2@e.com	CT2	R1	CN1	L2
C3	N3	66	C3@e.com	CT2	R1	CN1	L1
C4	N1	26	C4@e.com	CT3	R3	CN2	L1
C5	N5	45	C5@e.com	CT5	R2	CN1	L3
C6	N6	32	C6@e.com	CT4	R3	CN2	L2
C7	N7	41	C7@e.com	CT3	R3	CN2	L3

D1

IdCus	NameCus	Phone	Department	Region	Country	Population	Continent	MemSubLevel
C1	N1	012345	D1	R1	CN1	1000000	CTN1	SL1
C2	N2	123456	D3	R1	CN1	1000000	CTN1	SL3
C3	N3	234567	D3	R1	CN1	1000000	CTN1	SL1
C4	N1	345678	D4	R3	CN2	1200000	CTN1	SL2
C6	N6	456789	D4	R3	CN2	1200000	CTN1	SL3
C8	N8	567890	D2	R2	CN1	1000000	CTN1	SL4
C9	N9	678901	D5	R4	CN3	500000	CTN2	SL5

D2

	IdCus	NameCus	Age	Email	Phone	City	Department	Region	Country	Population	Continent	MemSubLevel	MemLevel
D1 + D2	C1	N1	34	C1@e.com	012345	CT1	D1	R1	CN1	1000000	CTN1	SL1	L1
	C2	N2	53	C2@e.com	123456	CT2	D3	R1	CN1	1000000	CTN1	SL3	L2
	C3	N3	66	C3@e.com	234567	CT2	D3	R1	CN1	1000000	CTN1	SL1	L1
	C4	N1	26	C4@e.com	345678	CT3	D4	R3	CN2	1200000	CTN1	SL2	L1
D1	C5	N5	45	C5@e.com	NULL	CT5	NULL	R2	CN1	NULL	NULL	NULL	L3
D1 + D2	C6	N6	32	C6@e.com	456789	CT4	D4	R3	CN2	1200000	CTN1	SL3	L2
D1	C7	N7	41	C7@e.com	NULL	CT3	NULL	R3	CN2	NULL	NULL	NULL	L3
D1	C8	N8	NULL	NULL	567890	NULL	D2	R2	CN1	1000000	CTN1	SL4	NULL
	C9	N9	NULL	NULL	678901	NULL	D5	R4	CN3	500000	CTN2	SL5	NULL

D12

Figure III.10: Dimension merging example (instance)

The attribute set of the merged dimension contains all the attributes of two original dimensions, while the original dimensions may contain their unique attributes. So there may be empty values in the merged dimension table on the instances coming from only one of the original dimension tables and we should replace the empty values on the basis of the existing data that we will introduce in the next chapter.

Second Case When the root parameters of the two dimensions do not match, the instance merging is done by $line_{18}$ and $line_{21}$. We keep the original instances of the original dimensions. For the newly added attributes, they are empty and will be completed by the data imputation of the next chapter.

Example 5.4. *The instance merging of Example 5.2 is demonstrated in Figure III.11. For each original dimension D_1 and D_2, we get a merged dimension table D_{12} and D_{21}. In D_{12}, the attribute Continent comes from the dimension table D_2, and the attribute Country comes from D_1.*

IdCus	City	Region	Country	MemLevel
C1	CT1	R1	CN1	L1
C2	CT2	R1	CN1	L2
C3	CT2	R1	CN1	L1
C4	CT3	R3	CN2	L1
C5	CT5	R2	CN1	L3
C6	CT4	R3	CN2	L2
C7	CT3	R3	CN2	L3

D_1

IdProd	Department	Region	Continent
P1	D1	R1	CTN1
P2	D3	R1	CTN1
P3	D3	R1	CTN1
P4	D4	R3	CTN1
P5	D4	R3	CTN1
P6	D2	R2	CTN1
P7	D5	R4	CTN2

D_2

IdCus	City	Region	Country	Continent	MemLevel
C1	CT1	R1	CN1	NULL	L1
C2	CT2	R1	CN1	NULL	L2
C3	CT2	R1	CN1	NULL	L1
C4	CT3	R3	CN2	NULL	L1
C5	CT5	R2	CN1	NULL	L3
C6	CT4	R3	CN2	NULL	L2
C7	CT3	R3	CN2	NULL	L3

D_{12}

IdProd	Department	Region	Country	Continent
P1	D1	R1	NULL	CTN1
P2	D3	R1	NULL	CTN1
P3	D3	R1	NULL	CTN1
P4	D4	R3	NULL	CTN1
P5	D4	R3	NULL	CTN1
P6	D2	R2	NULL	CTN1
P7	D5	R4	NULL	CTN2

D_{21}

Figure III.11: Dimension merging example (instance)

6 Star Schema Merging

In this section, we discuss the merging of two star schemas. Here, for simplicity, we use the notion "star schmea" for denoting a DW modelled by star schema, so the merging is carried not only at the schema level, but also at the instance level. Having two star schemas, we can get a star schema or a constellation schema because the fact table of each schema may be merged into one schema or not. The star schema merging is related to the dimension merging and fact merging. Two star schemas are possible to be merged only if there are dimensions having matched root parameters between them.

For the dimensions of the two star schemas, we have two cases: 1. The two star schemas

Algorithm 7: $mergeAllDimensions(S_1, S_2)$

Input : Two stars to be merged S_1 and S_2
Output: A set of merged dimensions $D^{S_{12}}$

```
1  for D_i ∈ D^{S_1} do
2      for D_j ∈ D^{S_2} do
3          if id^{D_i} ≄ id^{D_j} then
4              D_i^{S_1}, D_j^{S_2} ← MergeDimensions(D_i, D_j);
5  D^{S_12} ← ∅;
6  for D_u ∈ D^{S_1} do
7      for D_v ∈ D^{S_2} do
8          if id^{D_u} ≃ id^{D_v} then
9              D^{S_12} ← D^{S_12} ∪ MergeDimensions(D_u, D_v);
10 for D_k ∈ D^{S_12} do
11     for H_m^{D_k} ∈ H^{D_k} do
12         if ∄ i_r^{D_k} ∈ I^{D_k}(i_r^{D_k} is on H_m^{D_k} ∨ (i_r^{D_k} is only on
             H_m^{D_k} ∧ (H_m^{D_k} ∈ H^{D^{S_1}} ∨ H_m^{D_k} ∈ H^{D^{S_2}}))) then
13             H^{D_k} ← H^{D_k} − H_m^{D_k};
14 return D^{S_12}
```

have the same number of dimensions and for each dimension of one schema, there is a dimension having the matched root parameter in the other schema. 2. There exists at least one dimension among the two star schemas which does not have a dimension having the matched root parameter in the other star schema.

The dimension merging of two star schemas is common for the two cases which is done by Algo. 7. We first merge every two dimensions of the two star schemas which have unmatched root parameters because the merging of such dimensions is able to replenish the original dimensions with complementary attributes ($line_{1-4}$). Then the dimensions having matched root parameters are merged to generate the merged dimensions of the merged multidimensional schema ($line_{5-9}$). After the merging of the instances of the dimension tables, there may be some merged hierarchies to which none of the instances belong. In this case, if there will be no more update of the data, such hierarchies should be deleted. There may also be original hierarchies in the merged dimensions such that there is no instance which belongs to them but does not belong to any merged hierarchy containing all the parameters of this original hierarchy. The instances belonging to this kind of hierarchies belong also to other hierarchies which contains more parameters, so they become useless and should also be deleted ($line_{10-13}$).

Example 6.1. *For the merging of the dimensions of two star schemas S_1 and S_2 in Fig. III.12. The dimension Product of S_1 and the dimension Customer of S_2 are first merged since their root parameters do not match but they have other matched parameters.*

There are then attributes of dimension Customer of S_2 added into dimension Product of S_1. The two dimensions Customer and the two dimensions Product have matched root parameters, so they are merged into the final star schema. After the merging and the imputation of the instance, we verify each hierarchy in the merged dimension tables. If the merged instances of dimension Customer are like shown in Fig. III.10 except that the null values of Continent are all replaced, we can delete the hierarchy $IdCus \rightarrow Department \rightarrow Region \rightarrow Continent$ since all the instances belonging to this hierarchy also belong to the merged hierarchy $City \rightarrow Department \rightarrow Region \rightarrow Country \rightarrow Continent$.

We then discuss the merging of the other elements in the two cases which is processed by Algo 8:

Algorithm 8: $mergeStar(S_1, S_2)$

Input : Two stars to be merged S_1 and S_2

Output: A merged multidimensional schema which may be a star schema S_{12} or a merged constellation schema C_{12}

```
1  if |D^{S1}| = |D^{S2}| and ∀D_i ∈ D^{S1} ∃D_j ∈ D^{S2}(id^{D_i} ≃ id^{D_j}) then
2      D^{S12} ← MergeAllDimensions(S1, S2);
3      M^{F^{S12}} ← M^{F^{S1}} ∪ M^{F^{S2}};
4      I^{F^{S12}} ← I^{F^{S1}} ∪ I^{F^{S2}};
5      IStar^{F^{S12}} ← IStar^{F^{S1}} ∪ IStar^{F^{S2}};
6      return S12
7  else
8      D^{S12} ← MergeAllDimensions(S1, S2);
9      F^{C12} ← {F^{S1}, F^{S2}};
10     return C12
```

First Case For the first case ($line_1$), we merge the two fact tables into one fact table and get a star scheme. The dimension merging is achieved by Algo. 7 ($line_2$). The measure set of the merged star schema is the union of the two original measures ($line_3$). The fact instances are the union of the measure instances of the two input stars ($line_4$). The function associating fact instances to their linked dimension instances of the merged schema is also the union of the functions of the original schemas ($line_5$).

Example 6.2. *For the two original star schemas in Fig. III.13, the dimension merging is discussed above so we mainly focus on the merging of fact table instances in this section. The dimensions Customer, Product of S_1 have respectively matched root parameters in the dimensions Customer, Product of S_2. They also have the same number of dimensions. The original fact tables are merged into one fact table by merging the measures of S_1 and S_2 to get the fact table of S_{12}. At the instance level, in Fig. III.13, we have the instances of the fact tables, for the instances of F^{S_1} and F^{S_2}, the framed parts are the instances having common linked dimension instances, so they are merged into the merged fact table*

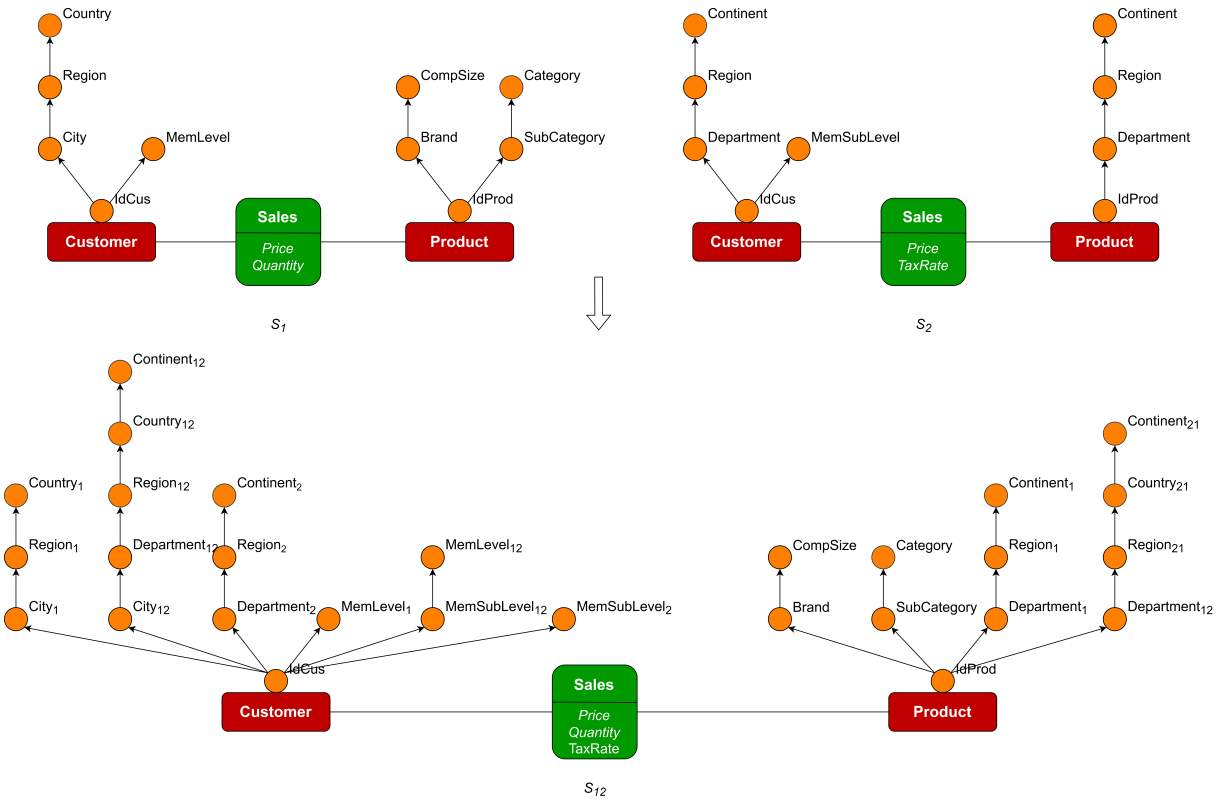
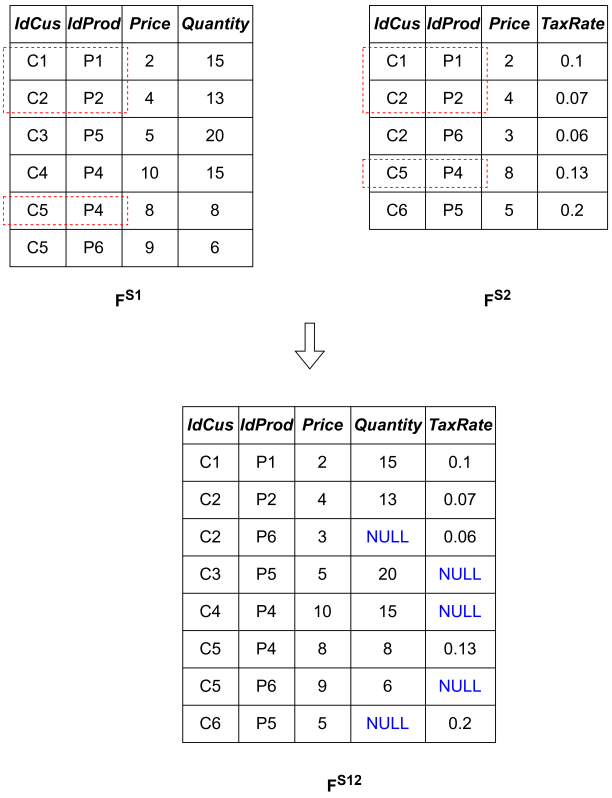


Figure III.12: Star merging example (schema)

$F^{S_{12}}$, the other instances are also integrated in $F^{S_{12}}$ but with empty values in the merged instances which can be treated in the newt chapter.

Second Case For the second case, since there are unmatched dimensions, the two facts have different links with different dimensions and can thus not be merged, the merged schema should be a constellation schema. We merge the dimensions by Algo. 7 ($line_8$). The facts of the original schemas have no change at both schema and instance levels and compose the final constellation schema ($line_9$). They are linked to the merged dimensions instead of the original ones.

Example 6.3. *This example is shown in Fig. III.14. For the original star schemas S_1 and S_2, they have dimensions Customer which have matched root parameters. The original star schemas also have their unique dimensions: Product of S_1 and Date of S_2. So the merged schema is a constellation schema generated by merging the dimensions Customer and by keeping the other dimensions and fact tables. At the instance level, we have a new merged dimension table of Customer, the other dimension tables and the fact tables remain unchanged.*



IdCus	IdProd	Price	Quantity
C1	P1	2	15
C2	P2	4	13
C3	P5	5	20
C4	P4	10	15
C5	P4	8	8
C5	P6	9	6

F^{S1}

IdCus	IdProd	Price	TaxRate
C1	P1	2	0.1
C2	P2	4	0.07
C2	P6	3	0.06
C5	P4	8	0.13
C6	P5	5	0.2

F^{S2}

IdCus	IdProd	Price	Quantity	TaxRate
C1	P1	2	15	0.1
C2	P2	4	13	0.07
C2	P6	3	NULL	0.06
C3	P5	5	20	NULL
C4	P4	10	15	NULL
C5	P4	8	8	0.13
C5	P6	9	6	NULL
C6	P5	5	NULL	0.2

F^{S12}

Figure III.13: Star merging example (instance)

7 Experimental Assessment

We carry out experiments to validate that our approach can correctly merge DWs at both schema and instance levels and can generate star or constellation schemas in different cases.

7.1 Datasets

We apply TPC-H benchmark data in our experiments. Originally, the TPC-H benchmark serves for benchmarking decision support systems by examining the execution of queries on large volumes of data. The TPC-H benchmark provides a pre-defined relational schema[1] with 8 tables and a generator of data. The generated data can be used to create DWs containing various dimensions, facts and hierarchies, which can cover different cases mentioned in our algorithms. We generated 100M of data files. We employe the files *Lineitem*, *Customer*, *Nation*, *Orders*, *Part*, *Region* and *Supplier* where there are respectively 600572, 15000, 25, 150000, 20000, 5, 1000 tuples.

7.2 DW Generation Strategy

In our algorithm, we discussed two cases where a star or a constellation schema is generated. So we carry out experiments for these two cases by creating two DWs for each case.

[1]http://tpc.org/tpc_documents_current_versions/pdf/tpc-h_v2.18.0.pdf

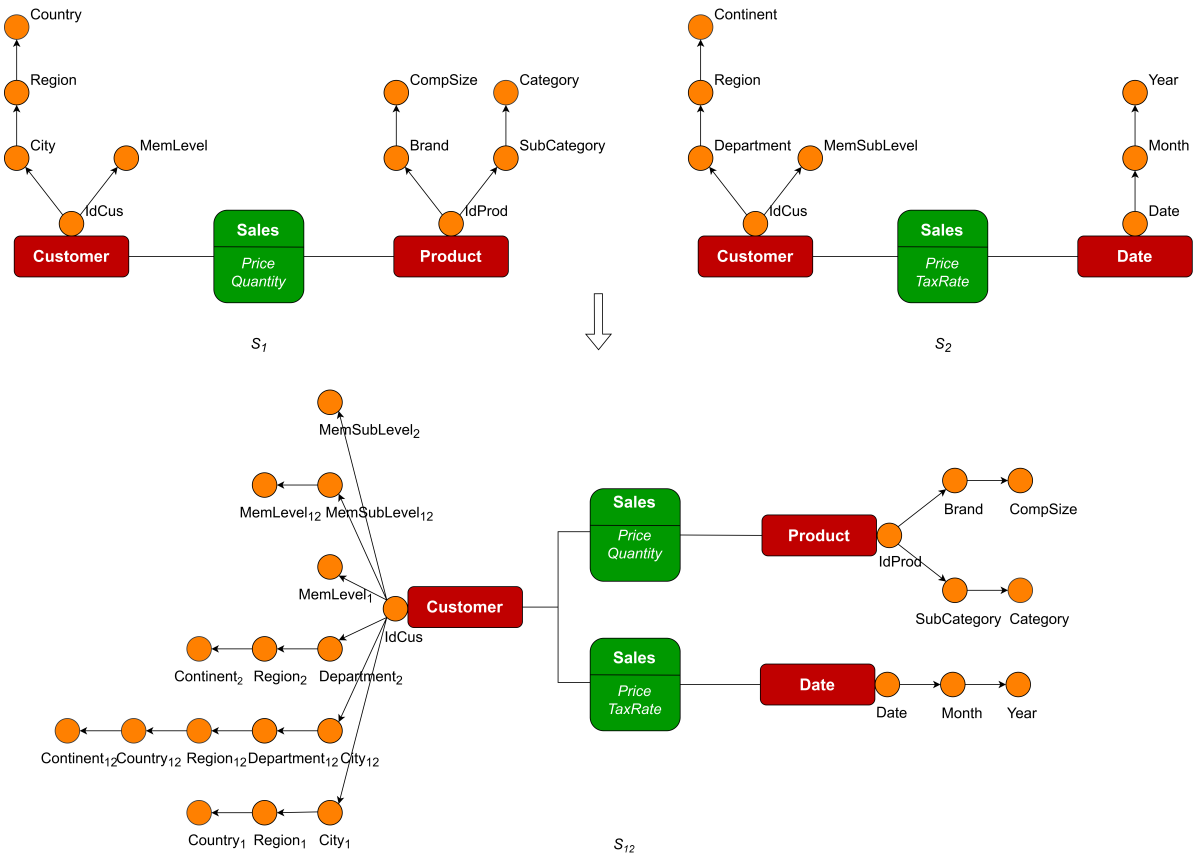


Figure III.14: Star merging example (schema)

First, we use files *Customer*, *Supplier*, *Part*, and attribute *orderdate* of file *Orders* to generate dimensions *Customer*, *Supplier*, *Part*, and *Orderdate*. We use *Lineitem* to generate facts *Lineorder*. Second, to have deeper hierarchies, we include the data of *Nation* and *Region* into *Customer* and *Supplier*. Third, to make sure that there are both common and different instances in different DWs, for each dimension, instead of selecting all the corresponding data, we select randomly 3/4 of them. For the fact table, we select the measures related to these dimension data. The DWs are implemented in R-OLAP format through the Oracle 11g DBMS.

7.3 Star Schema Generation

The objective of this experiment is to verify the correct generation of a star schema at the schema and instance levels by merging two star schemas having same dimensions (*Customer*, *Supplier*, *Part*) with the matched root parameter for each dimension.

7.3.1 Schema Level Merging Result

The run time of this experiment is 5.49s. The original DW schemas and the generated star schema are shown in Fig. III.15 which is consistent with the expectation. The three dimensions *Supplier*, *Part*, *Date* of the original DWs are merged. Between the different dimensions $S_1.Supplier$ and $S_2.Customer$, there is a matched attribute *Nation*, so they

are also merged such that $S_1.Supplier$ provides $S_2.Customer$ with the attribute *Region*. Then the $Customer$ in the merged DW also has the attribute *Region*.

In our algorithm, different dimensions (root parameters do not match) having common parameters are firstly merged. $S_1.Supplier$ and $S_2.Customer$ are different dimensions, but they have common parameter $Nationkey$. So they are firstly merged so that a merged hierarchy $Custkey \rightarrow Nationkey \rightarrow Regionkey$ is created and is added to $S2.Customer$.

Then, dimensions having matched root parameters are merged together. By merging $S_1.Customer$ and $S_2.Customer$, $S_{12}.Customer$ is generated which contains the original hierarchies of $S_1.Customer$ and $S_2.Customer$ as well as the hierarchy obtained by merging $S_1.Supplier$ and $S_2.Customer$. The weak attributes of the root parameter $Custkey$ are also merged together. By merging $S_1.Supplier$ and $S_2.Supplier$, $S_{12}.Supplier$ is generated containing the merged hierarchies and original hierarchies with the merged weak attributes. By merging $S_1.Part$ and $S_2.Part$, $S_{12}.Part$ is generated. It containes the original hierarchies and a hierarchy $Partkey \rightarrow Brand \rightarrow Manufacture$ created by merging hierarchies $Partkey \rightarrow Brand$ of $S_1.Part$ and $Partkey \rightarrow Manufacture$ of $S_2.Part$.

The facts $S_1.Lineorder$ and $S_2.Lineorder$ are both associated to the dimensions having matched root parameters. Therefore, a star schema is finally generated. The measures are obtained by merging the measures in the original fact tables. The merged fact is associated to the merged dimensions to finally form S_{12}.

By analysing the merging process and the obtained merged schema, we confirm that our algorithm correctly generate a merged star schema by correctly merging facts, dimensions, hierarchies and weak attributes at the schema level.

7.3.2 Instance Level Merging Results

The results of the instance level merging for the star schema generation experiment is shown in Table III.2. We verify the correctness of the merging by the number of attributes. For an attribute, N_1 denotes the number of values in S_1, N_2 denotes the number of values in S_2, $N_\cap$ denotes the number of common values between S_1 and S_1 and is only applied for dimension or fact keys, N_m is the number of values in the merged DW.

For a dimension key or a combination of fact keys, it values are unique. So for them, the number of values is the number of tuples in the dimension/fact, we then have the relationship of these numbers for them $N_m = N_1 + N_2 - N_\cap$. By verifying the results in the table, we validate the existence of such relationships for the merged dimensions and fact.

For the other dimension or fact attributes, since there is no missing data in the original DWs, if both DWs have an attribute of a dimension/fact, the N_m of the attribute should be equal to the N_m of the dimension/fact key. If only one of the DWs has an attribute of a dimension/fact, the N_m of the attribute should be equal to the number of the attribute

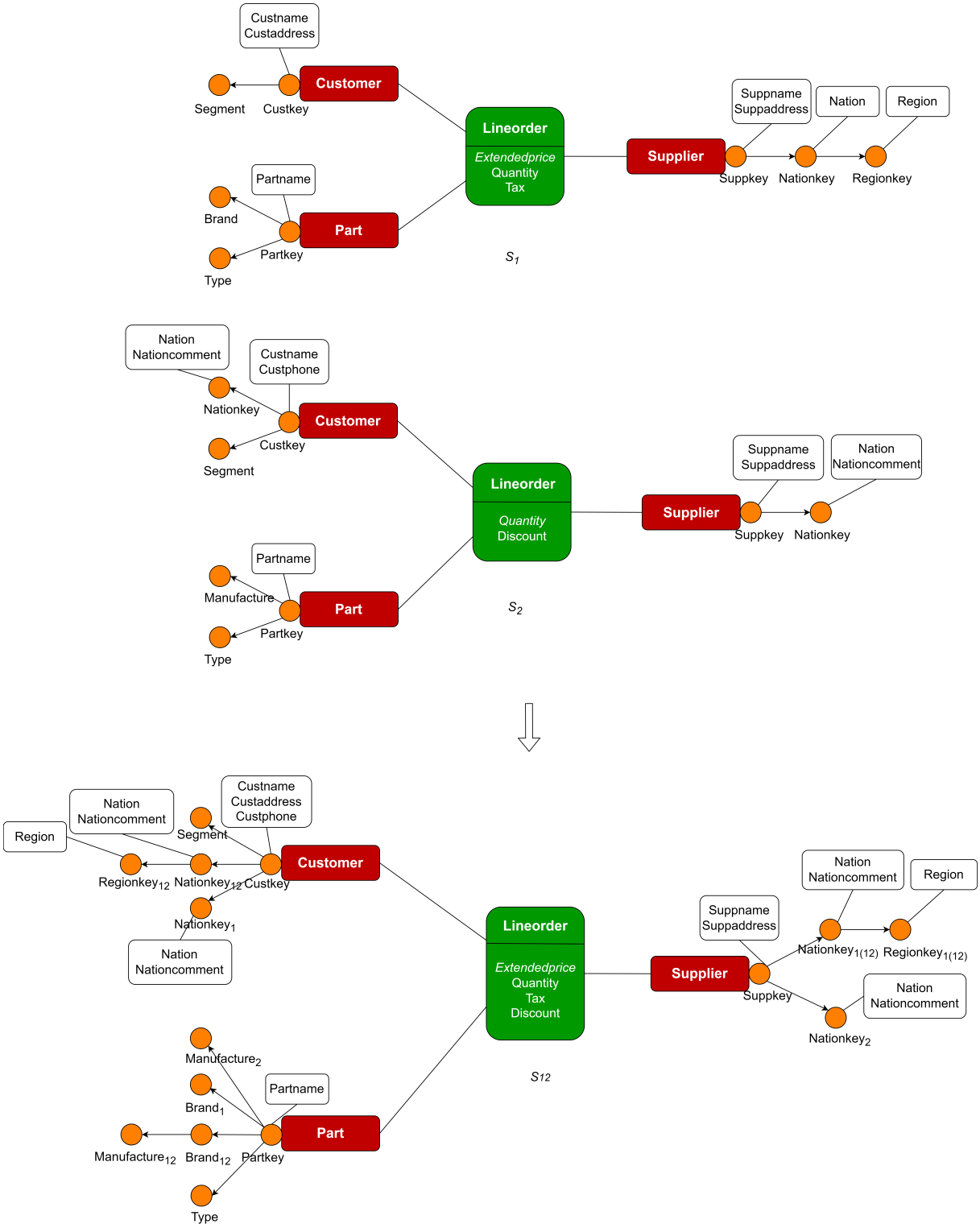


Figure III.15: Star schema generation

Table III.2: Results of star generation

Dimension /Fact	Attribute	N_1	N_2	$N_\cap$	N_m
Customer	**Custkey**	11250	11250	8426	14074
	Custname	11250	11250		14074
	Custaddress	11250	0		11250
	Custphone	0	11250		11250
	Nationkey	0	11250		11250
	Nation	0	11250		11250
	Nationcomment	0	11250		11250
	Segment	11250	11250		14074
Supplier	**Suppkey**	750	750	570	930
	Suppname	750	750		930
	Suppaddress	750	750		930
	Nationkey	750	750		930
	Nation	750	750		930
	Nationcomment	0	750		750
	Regionkey	750	0		750
	Region	750	0		750
Part	**Partykey**	15000	15000	11215	18785
	Partname	15000	15000		18785
	Brand	15000	0		15000
	Manufacture	0	15000		15000
	Type	15000	15000		18785
Lineorder	**Custkey Partkey Suppkey**	253423	253782	107736	399469
	Quantity	253423	253782		399469
	Extendedprice	253423	0		253423
	Tax	253423	0		253423
	Discount	0	253782		253782

values in the DW having this attribute. By verifying the results in the table, we also validate the existence of such relationships.

Finally, we also validate that attribute values are the same in the merged dimensions or fact as in the original ones by SQL queries.

By analysing the merged DW instances and by comparing them to the original ones, we confirm that our algorithm correctly generate a merged star schema DW at the instance level.

7.4 Constellation Schema Generation

The objective of this experiment is to verify the correct generation of a constellation schema at the schema and instance levels by merging two star schemas having same

dimensions (*Customer*, *Supplier*) with the matched root parameter for each dimension, as well as different dimensions ($S_1.Part$ and $S2.Date$).

7.4.1 Schema Level Merging Result

The run time of this experiment is 3.17s. The original DW schemas and the generated constellation schema are shown in Fig. III.16 which is consistent with the expectation.

The dimensions *Customer* and *Supplier* of S_1 and S_2 are the same as those in the star schema generation experiment. Thus we obtain the same merged dimensions *Customer* and *Supplier* in C_{12}. Since there is no dimension in S_2 having the matched root parameter as $S_1.Part$ and there is no dimension S_1 having the matched root parameter as $S_2.date$, these two dimensions remain the same in the merged schema as in the original schemas.

The facts $S_1.Lineorder$ and $S_2.Lineorder$ are associated to $S_1.Customer$ and $S_2.Customer$ whose root parameters are matched and are associated to $S_1.Supplier$ and $S_2.Supplier$ whose root parameters are also matched. Meanwhile, $S_1.Lineorder$ and $S_2.Lineorder$ are respectively associated to $S_1.Part$ and $S_2.Lineorder$, whose root parameter are not matched in the other DWs. Therefore, a constellation schema is generated. The two facts in the constellation schema have the same measures as the original ones.

By analysing the merging process and the obtained merged schema, we confirm that our algorithm correctly generate a merged constellation schema by correctly merging facts, dimensions, hierarchies and weak attributes at the schema level.

7.4.2 Instance Level Merging Results

The results of the instance level merging for the constellation schema generation experiment is shown in Table III.3. For the dimension keys and the combinations of fact keys, the relationships of $N_m = N_1 + N_2 - N_\cap$ from the observation of the results are validated.

For the other dimension or fact attributes, we also validate the relationship as discussed in Section 7.3.2 by verifying the results in the table

Finally, we also validate that attribute values are the same in the merged dimensions or facts as in the original ones by SQL queries.

By analysing the merged DW instance and by comparing them to the original ones, we confirm that our algorithm correctly generate a merged constellation schema DW at the instance level.

We get the results conforming to the expectations in these two experiments, we can thus conclude that our algorithm works well for the different cases of the generation of a star or a constellation schema at both schema and instance levels.

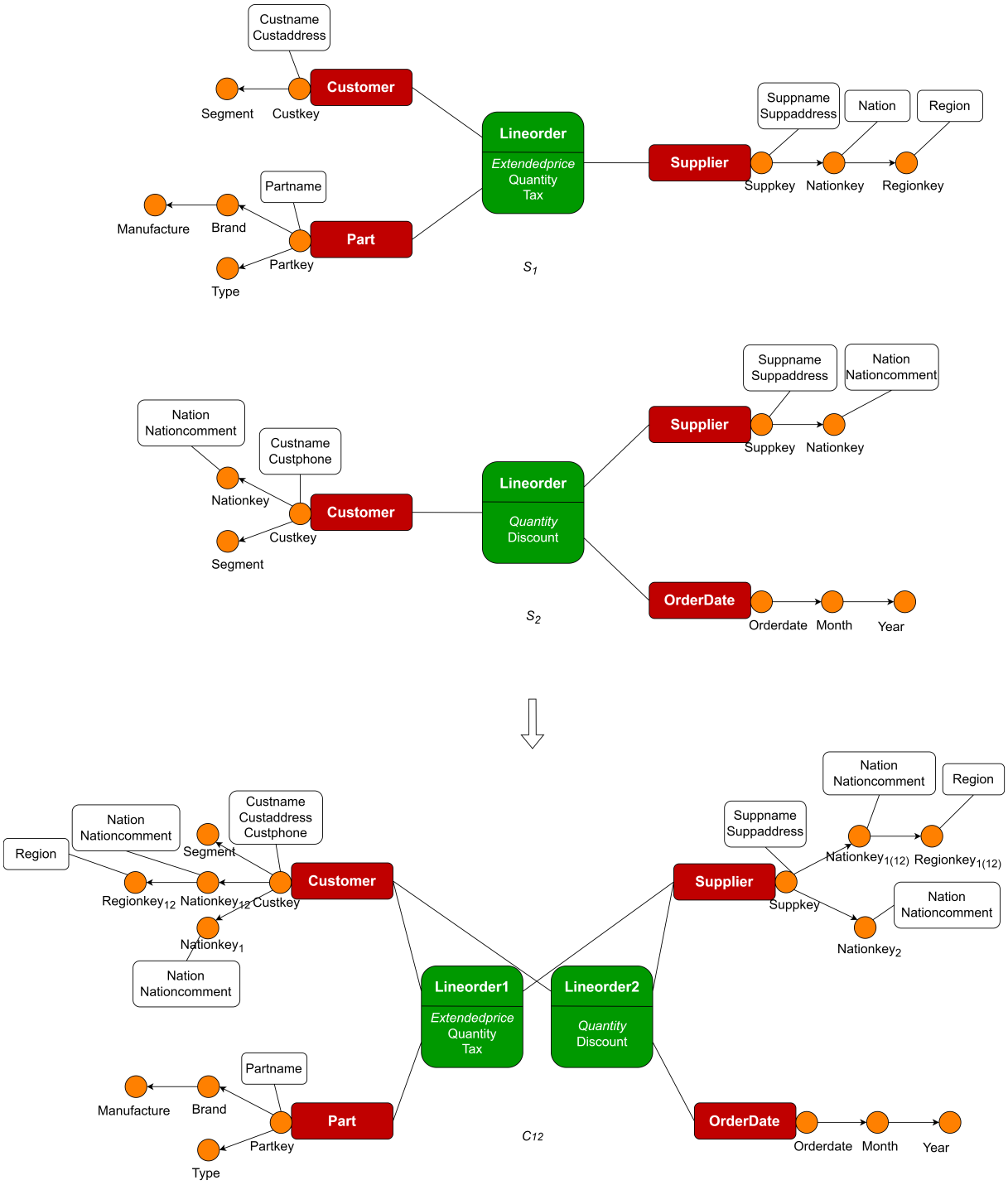


Figure III.16: Constellation schema generation

Table III.3: Results of constellation schema generation

Dimension/Fact	Attribute	N_1	N_2	$N_\cap$	N_m
Customer	**Custkey**	11250	11250	8449	14051
	Custname	11250	11250		14051
	Custaddress	11250	0		11250
	Custphone	0	11250		11250
	Nationkey	0	11250		11250
	Nation	0	11250		11250
	Nationcomment	0	11250		11250
	Segment	11250	11250		14051
Supplier	**Suppkey**	750	750	561	939
	Suppname	750	750		939
	Suppaddress	750	750		939
	Nationkey	750	750		939
	Nation	750	750		939
	Nationcomment	0	750		750
	Regionkey	750	0		750
	Region	750	0		750
Part	**Partykey**	15000	0	0	15000
	Partname	15000	0		15000
	Brand	15000	0		15000
	Manufacture	15000	0		15000
	Type	15000	0		15000
Orderdata	**Orderdate**	0	1805	0	1805
	Month	0	1805		1805
	Year	0	1805		1805
Lineorder1	**Custkey Partkey Suppkey**	253420	0	0	253420
	Quantity	253420	0		253420
	Extendedprice	253420	0		253420
	Tax	253420	0		253420
Lineorder2	**Custkey Suppkey Orderdate**	0	336584	0	336584
	Quantity	0	336584		336584
	Discount	0	336584		336584

8 Conclusion

When we have several DWs generated from multiple sources and which share some same information, they can be merged to allow a consolidated analysis of the data.

The DW merging consists of the matching and merging of multidimensional elements. Since the matching is well studied in the literature, we only focused on the merging process. However, the complex structure of multidimensional DW makes this task difficult. A DW has a specific schema and its instances are stored in different tables. A star or constellation schema may be generated in different above-mentioned cases. To answer these problems by resolving the difficulties, in this chapter, we proposed an automatic approach to merge two different DWs at both schema and instance levels, which generate a star or constellation schema.

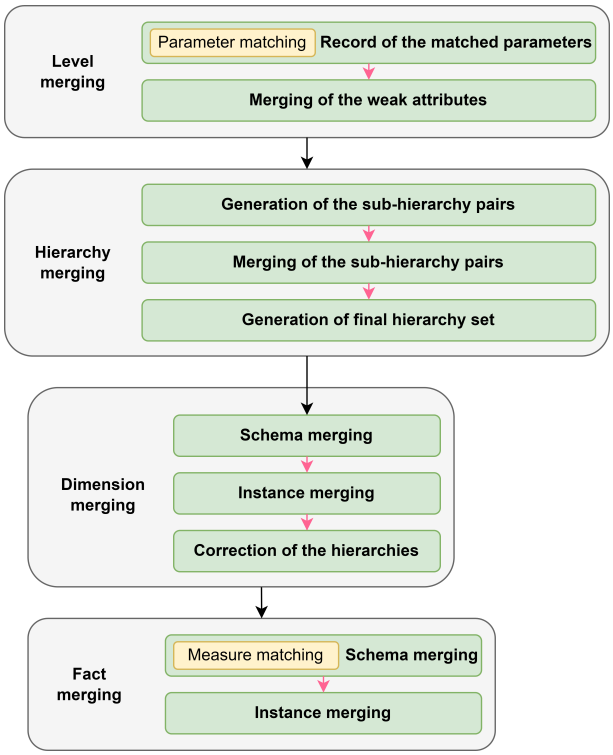


Figure III.17: Summary of the merging process

Fig. III.17 summarizes our proposed merging process. The DW merging is composed of dimension merging and fact merging. The dimension merging is based on the hierarchy merging which is based on level merging. At the level merging stage, we defined algorithms to match and record the matched parameters. Then they are merged by considering the merging of their corresponding weak attributes which is little considered in the literature. At the hierarchy merging stage, according to the level merging results, we defined algorithms to merging hierarchies having matched parameters. We decompose the hierarchies into sub-hierarchy pairs. By proposing merging processes of the sub-hierarchy pairs in different cases, we generate the final merged hierarchy set. Compared to the hierarchy merging approaches in the literature, we not only merge the equivalent levels,

but also create new hierarchy relationships for non-matched levels based on functional dependencies. At the dimension merging stage, we defined algorithms to merge dimension schemas by calling hierarchy merging algorithms. The instance merging is also proposed in the algorithms. For the fact merging, we match measures and carry out the schema and instance merging. Moreover, we also consider different cases where a star or constellation schema may be generated, which is not taken into account in the literature.

We conducted experiments with TPC-H benchmark data to validate that the merging process works correctly. Our process is able to generate different kinds of schemas (star or constellation) in different cases. For both cases, at the schema level, our process can correctly merge different multidimensional components including facts, dimensions, hierarchies and weak attributes, etc. At the instance level, the instances are correctly merged without undesired addition or loss of data. The DW merging approach is validated through a paper in the international conference IDEAS2021 (**YANG, Y.** et al., 2021b).

During the merging of DW instances, missing data may be generated which produce incomplete hierarchies. An analysis form is proposed in this chapter to repair the hierarchies at the schema level by displaying all complete hierarchies. However, the instance level repair is also necessary by replacing missing values with appropriate data, which leads to the data imputation task.

Chapter IV

Data Warehouse Imputation

Contents

1 Introduction

1.1 Context

As we mentioned in the previous chapter, during the DW merging, if there are different attributes in the original DWs, missing data are generated in the merged DW. Missing data may also come from DW sources (operational data sources, missing data of original DWs) if not treated during the Extract-Transform-Load (ETL) process. There are 2 types of missing data in DWs: dimensional missing data which are missing data in the dimensions and factual missing data which are in the facts. These missing data have impact on OLAP analyses. Factual missing data are usually quantitative, making analysis results incomplete and preventing users from getting reliable aggregates. Dimensional missing data are usually qualitative, making aggregated data incomplete and making it hard to analyse them with respect to hierarchy levels. Therefore, it is significant to replace missing data.

1.2 Challenge

Data imputation is the process of replacing the missing values by some plausible values based on information available in the data (Li et al., 2004a). The current DW data imputation research mainly focuses on factual data (Wu and Barbará, 2002; de S. Ribeiro et al., 2011; Amanzougarene et al., 2014; Bimonte et al., 2020). Yet as we mentioned that dimensional missing also have impact on data analysis. Therefore it is essential to propose the imputation of dimension attributes in order to ensure that high-quality analyses are offered to decision makers and even potential new analyses with the addition of missing attributes. However a DW dimension has a complex structure containing different hierarchies with different granularity levels having their dependency relationships. When we replace dimensional missing data, we have to take the DW structure and the dependency constraints into account.

1.3 Our Approach Overview

However, to the best of our knowledge, there is no other specific data imputation method for DW dimensions. Thus, in this chapter, we propose a dimensional data imputation approach named Hie-OLAPKNN based on hierarchical relationships of dimension attributes and k-nearest neighbors (KNN) algorithm. The approach is composed of two parts (Fig. IV.1).

The first part is a hierarchical imputation (Hie). Since there are functional dependencies between different granularity levels on a hierarchy, we can take advantage of these functional dependencies to replace missing values. The hierarchical imputation relies on both inter- and intra-dimensional hierarchical dependency relationships. The hierarchical imputation is convincible because we use accurate data based on real functional dependency relationships. However, this method is limited owing to the sparsity problem (Song

and Sun, 2020) which means that for an instance to be replaced, there may not be an instance sharing the same value on a lower-granularity level of the hierarchy. Thus, in order to replace as many values as possible, we also propose a second part to replace the remaining missing values.

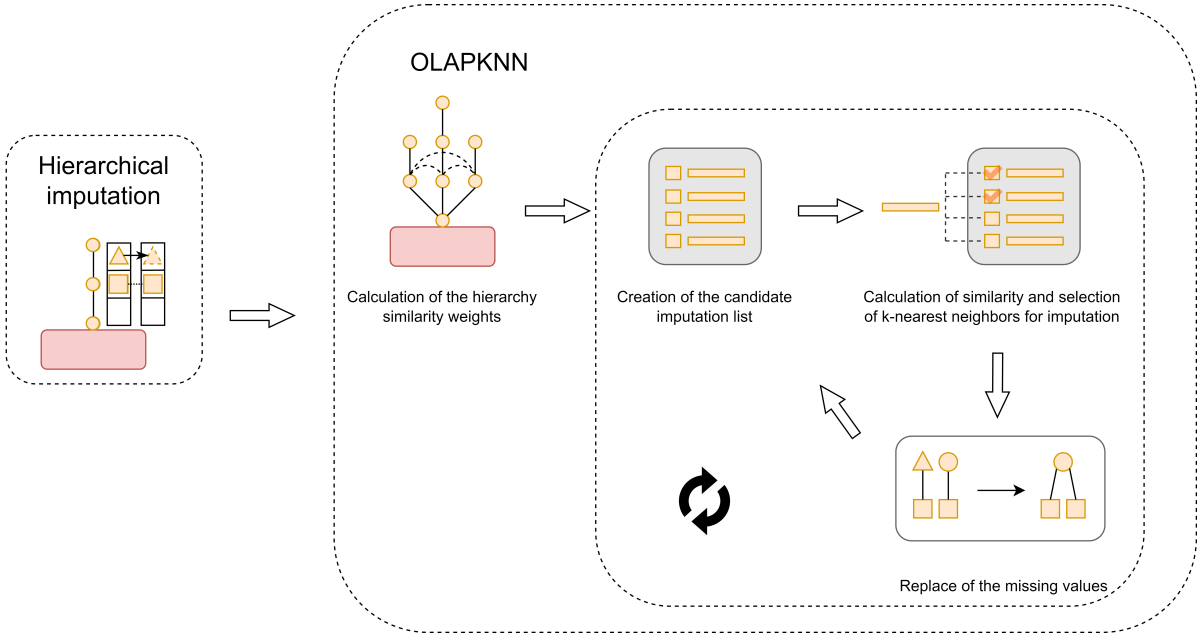


Figure IV.1: Overview of the Hie-OLAPKNN imputation approach

The second part of imputation is based on KNN algorithm (OLAPKNN). KNN imputation finds the k nearest neighbors of an instance with missing data then fill in the missing data based on the mean or mode of the neighbors' value (Troyanskaya et al., 2001). We choose KNN because it is a non-parametric and instance-based algorithm, which is widely applied for data imputation (Beretta and Santaniello, 2016) and has been proved to have relatively high accuracy (Li and Parker, 2014; Troyanskaya et al., 2001). Compared to the basic KNN imputation, OLAPKNN considers the structure complexity and the dependency constraints of the dimension hierarchies. KNN imputation relies on the distance among dimension instances, so we also propose a specific distance metric for dimension instances by considering different dimension elements. As shown in Fig IV.1, in OLAPKNN, we first calculate the hierarchy weight which is dedicated for the calculation of dimension instance distance. Then the imputation is realized by a iterative process. In each loop we replace the missing values of a certain number of continuous attributes. Next we create a candidate list of replaced values by respecting hierarchical relationships. Then by the proposed distance metric, we choose the nearest neighbors of the missing value for the imputation.

1.4 Outline

The remainder of this chapter is organized as follows. In Section 2, we review the related work about data imputation algorithms. In Section 3, we introduce the first part of our

imputation approach, the hierarchical imputation. In Section 4.1, we propose a distance calculation metric for dimension instances which serves to find the nearest neighbors in KNN algorithm. In Section 5, we explain in detail the OLAPKNN dimension imputation algorithm. In Section 6, we validate our proposal by some experiments. In Section 7, we conclude this chapter.

2 Related Work

Data imputation is an important topic that arises a lot of attentions. Most of the data imputation techniques are dedicated for missing data in flat files, while only few works focus specifically on the imputation of missing data in DWs. In this section, we present the general data imputation techniques and analyse them in terms of their target data types as well as their advantages and disadvantages. Then we discuss the data imputation approaches for DWs.

2.1 General Imputation Approaches

There are various data imputation techniques for missing data in flat files (Miao et al., 2018; Lin and Tsai, 2020; Osman et al., 2018). These techniques can be classified into four categories: statistical-based, machine learning-based, rule-based and external source-based.

2.1.1 Statistical-based

Mean/mode imputation (Schafer and Graham, 2002; Twala et al., 2005) The mean/-mode imputation is the most basic and simplest data imputation approach. For missing numerical data, the mean approach is applied where the average of the existing attribute values are used to replace the missing data. For missing textual data, the mode approach is applied where the value appears the most is used to replace the missing data.

EM algorithm The Expectation-maximization (EM) algorithm (McLachlan and Krishnan, 2007) is also a widely used statistical technique for data imputation. The EM algorithm is an iterative algorithm which aims to carry out maximum likelihood estimation for statistical models in the presence of latent variables. It is compose of E-step and M-step: the E-step estimates parameters from the data and existing models and then calculates the likelihood function based on the estimated parameters; the M-step finds the parameters which maximize the likelihood function. The two steps repeat until the algorithm converge. Then with the final obtained estimated parameter, the missing data can be expected and be replaced (Graham et al., 2009; Lin, 2010; Schneider, 2001).

Regression imputation Regression techniques can be applied for missing data imputation. In regression techniques, the non-missing data are trained to estimate the

regression coefficients. Then the missing data can be estimated and replaced by the regression model. The numerical and categorical data can respectively be replaced by the linear regression (Beaumont, 2000; Qin et al., 2007; Baraldi and Enders, 2010) and the logistic regression (Allison, 2005; Sentas and Angelis, 2006).

Hot and cold deck There are also hot and cold deck techniques for missing data imputation. In the hot deck technique (Andridge and Little, 2010; Wu et al., 2012), for a missing value, its closest record containing non-missing value of the attribute in the dataset is found. Then the missing value is replaced by the this non-missing value. Cold deck (Little and Rubin, 2019) is similar to the hot deck except that it searches for the replaced value in a previous data collection or a different dataset.

2.1.2 Machine learning-based

KNN KNN algorithm is the most used machine learning-based data imputation algorithm (Lin and Tsai, 2020). In the KNN algorithm, the k-nearest neighbor tuples of a missing value tuple are found. A distance function is defined, the process of find the nearest neighbors is the process of minimizing the distance between the missing value tuple and the other tuples. Thus the distance function attracts a lot of attentions in the literature. In terms of the distance metric, the Euclidean distance is a typical one (Troyanskaya et al., 2001; García-Laencina et al., 2009; Lee and Styczynski, 2018). The grey theory-based distance is also employed in (Zhang, 2012; Huang and Lee, 2004; Pan et al., 2015). Moreover, the normalized numerical distance and string similarity measures are used in (Song and Sun, 2020).

In the distance function, each attribute may have an equivalent weight (Zhang, 2012; Huang and Lee, 2004; Troyanskaya et al., 2001; Lee and Styczynski, 2018). They may also have specific weights defined by some dependency measurements such as mutual information (García-Laencina et al., 2009; Pan et al., 2015).

For the final replaced value, it may be obtained by the average or mode value (Zhang, 2012; Huang and Lee, 2004), or the distance-based weighted average or mode value of the k-nearest neighbors (Dudani, 1976; García-Laencina et al., 2009; Pan et al., 2015; Troyanskaya et al., 2001; Lee and Styczynski, 2018). Here, average values are applied for the imputation of numerical data and mode values are applied for the imputation of categorical data.

Other Supervised Learning Techniques Other supervised learning algorithms are also employed for data imputation. The tuples without missing data are trained to gain a model. Then the missing values can be predicted with the relevant attribute values based on the trained model. The algorithms include the decision tree algorithms such as CART (Hapfelmeier et al., 2012; Ding and Simonoff, 2010; Burgette and Reiter, 2010) and C4.5(Twala, 2009; Ding and Simonoff, 2010), random forest (Pantanowitz and Marwala,

2009; Xia et al., 2017; Tang and Ishwaran, 2017; Kokla et al., 2019), support vector machine (Wang et al., 2006; Zhang and Liu, 2009), naive bayes (Garcia and Hruschka, 2005; Hruschka et al., 2007), and so on.

Clustering Clustering algorithm is the only unsupervised algorithm for data imputation. In clustering-based imputation, the tuples not containing missing data are classified into different clusters. Then the distance between the tuples containing missing data and the centroid of the clusters are calculated. The missing data can thus be replaced by the kernel function or by cluster values such as the centroid's value or the closest neighbor in the cluster. The clustering algorithms for data imputation include k-means or fuzzy k-means (Zhang et al., 2008; Li et al., 2004b; Liao et al., 2009), fuzzy c-means (Samat and Salleh, 2016; Sefidian and Daneshpour, 2019), Gaussian mixture clustering (Ouyang et al., 2004; Yan et al., 2015), and so on.

Neural Network Neural network algorithms receive many attentions in recent years' research. Several imputation approaches are proposed based on various neural network algorithms such as the probabilistic neural network (Nishanth and Ravi, 2016), generalized regression neural networks (Gheyas and Smith, 2010), artificial neural network (Verpoort et al., 2018), auto-encoder neural network (Choudhury and Pal, 2019). Some neural network-based imputation approaches are also proposed for specific types of data like the imputation of traffic data using convolutional neural network (Zhuang et al., 2019) and for time-variant data using recurrent neural network (Sangeetha and Senthil Kumaran, 2020)

2.1.3 Rule-based

Editing Rule (Fan et al., 2010) propose an approach for fixing errors and replacing missing data based on editing rules and master data. Editing rules are dynamic constraints that tell us which attributes should be changed and how to update them. Editing rules can be extracted by business rules. Then can also be reasoned from master data by the techniques proposed in the paper. The missing data can thus be replaced by the patterns of the editing rules.

Dependency Rule Data dependencies are important integrity constraints in the database theory. They are also used for the imputation of missing data. For a missing value, knowing it is determined by which attributes, it can be replaced by searching for the same values of these attributes. (Wijsen, 2005) propose to repair databases by using functional dependencies with a chase algorithm for data imputation. Some extended dependencies such as conditional functional dependencies and conditional inclusion dependencies (Bohannon et al., 2006; Fan, 2008) (Bohannon et al., 2006), which are dependencies under certain conditions like given the value of certain attributes, are also propose for replacing missing data.

Association Rule Association rule is a popular data mining technique which aims to extract interesting correlations, frequent patterns, associations or casual structures among sets of items in the databases or other data repositories (Zhao and Bhowmick, 2003). It reflects the co-occurrences of data values. Unlike data dependencies which should meet strict satisfactions, the association rules are probabilistic (Agrawal et al., 1994). So the support and the confidence which are respectively statistical significance and meaningfulness measures should be defined for the association rule mining. Several approaches (Wu et al., 2004, 2007; Shen et al., 2007) are proposed for data imputation based on association rule mining.

2.1.4 External Source-based

Crowd-sourcing Crowdsourcing is a type of participative online activity in which an individual, organization, or company with enough means proposes to a group of individuals of varying knowledge, heterogeneity, and number, via a flexible open call, the voluntary undertaking of a task (Estellés-Arolas and González-Ladrón-de Guevara, 2012). The imputation of missing data can be treated as a crowdsourcing task, we can thus get the different possible answers for the imputed data. According to the obtained values, the confidence or possibility of each value can be calculated. The value with the highest confidence or possibility can be applied to fill in the missing value (Ye and Wang, 2014; Ye et al., 2020).

Web Information There exists numerous information on the web. So missing data can be replaced by searching them on the web. Several web-based imputation approaches are proposed (Li et al., 2014; Tang et al., 2017; Liu et al., 2018). In these approaches, data imputation queries are generated and are input into the web to search for the answer. The most important part in the web-based imputation is the query formulation, the result is more accurate if the query is better defined. Different query formulation algorithms are proposed in these articles including greedy iterative scheduling algorithm (Li et al., 2014), dependency-based graph model (Tang et al., 2017), genetic-based algorithm (Liu et al., 2018), etc.

Knowledge Base A knowledge base is a machine-readable collection of knowledge about the real world containing entities and relations between them (Pellissier Tanon et al., 2020). Some approaches (Qi et al., 2017, 2018) are proposed to extract the replaced values for missing data. The imputation by knowledge base is based on the knowledge-based pattern mining. The pattern is described with the subject, predicate and object. The relationships of the attributes can be detected based on the existing values. The semantic type of each attribute is also discovered. The missing values can then be imputed by the queries obtained by the detected relationships. The results are validated by semantic types.

2.1.5 Hybrid Approaches

The approaches on single technique have different limits or are suitable for different specific contexts. Thus hybrid approaches are proposed where different techniques are combined to obtain better imputation results. Hybrid imputation applies techniques of different categories such as combining naive bayes and EM algorithm (Zhang et al., 2011), combining dependencies and web information (Li et al., 2015), combining KNN and regression (Zhang et al., 2019), combining KNN and likelihood maximization (Song and Sun, 2020), combining k-means and association rules (Chhabra et al., 2018), combining fuzzy c-means, support vector regression and genetic algorithm (Aydilek and Arslan, 2013), etc. There are also hybrid approaches using techniques of a same category like combining knn and random forest (Latifi and Koch, 2012) and combining crowdsourcing and knowledge base (Wang et al., 2017), etc.

2.2 Analysis of the Approaches

Table IV.1 shows the comparison of the different categories of imputation approaches. We list the number of approaches that we mention in the literature review in each category (#), the number of the approaches in each category being able to impute numerical data and categorical data as well as their ratio. Based on our analysis and the analysis of these approaches in some surveys (Graham et al., 2009; Miao et al., 2018; Lin and Tsai, 2020; Osman et al., 2018), we also discuss the advantages and disadvantages for each category of imputation approaches in the table. Since we focus on dimensional data which are mostly categorical, we discuss especially the application of these approaches for categorical data.

Regarding the statistical-based imputation, 7 of 13 of the approaches in this category can treat categorical data. The mean/mode can be used for categorical data imputation by the mode imputation. However, if we take this strategy, all the missing values of an attribute will be replaces by a same mode value which may be highly biased. The EM algorithm imputation approaches focus on numerical data and can not be used for categorical data. With respect to the regression imputation, the logistic regression can be used for the imputation of categorical data. However, the regression can not fit a good model when there is a complex correlation between the attributes. Moreover, when we apply the logistic regression, the categorical data are treated as dummy values (Graham et al., 2009). For categorical data of high-cardinality, a large amount dummy values will be generated and may impact the efficiency. For hot or cold deck imputation, they are a large category of similarity-based imputation. The KNN-based imputation can also be considered as hot deck. There are also other hot deck imputation which select randomly replaced value from potential ones, so the disadvantage of these hot deck approaches is that the replaced values depend on the selected auxiliary variables.

Regarding the machine learning-based imputation, 12 of 33 of the approaches in this category can treat categorical data. KNN-based imputation is the most applied approach. It is suitable for different cases in terms of the amount and the type of data. Since it looks

every row to search for the nearest neighbors, it has a relatively high computational cost. The clustering-based imputation is similar to KNN since it is also based on the similarity. However, in these clustering algorithms, the centroids are calculated by the mean of the cluster values, so it is not suitable for categorical data. For the other supervised learning algorithms, 4 of 13 can treat the imputation of categorical data. And the trained model is based on the existing data, so when there are a large number of missing data, the model may not be accurate. Neural network-based imputation can achieve a high accuracy. However only 1 approach can impute categorical data. To train a good neural network model, we need a huge amount of data. Moreover, the neural network training usually takes a long time.

Regarding the rule-based imputation, the approaches are all suitable for both numerical and categorical data. The editing rules and dependency rules are certain and robust rules, and the replaced value based on these rules are also robust. However, editing rules are extracted by some business rules or by master data which are not always available for users. The dependency rules can be extracted from the schema or from the data, however, the dependency relationships containing the replaced value in the determined side do not always exist. So we may only be able to replace a small part of the missing values by dependency-based approaches. The association rules are conditional rules so they can reveal more relationships for the imputation. However, the reasoning of association rules requires the definition of the support and confidence threshold, which may be hard for the user.

Regarding the external source-based imputation, the approaches can all impute both numerical and categorical data since they can retrieve the real information from external sources. However, for crowdsourcing-based imputation, the conduction of crowsourcing is costly and requires a huge budget. So it is not affordable for everyone. In the web information-based approaches, appropriate queries should be built to find reliable information for missing data. The searching of the results by the execution of a large amount of queries is also time-consuming. For the imputation based on knowledge base, the appropriate knowledge bases are required to replace the missing data, which are not always available.

Regarding the hybrid approach, 6 of 8 can replace categorical data. Since they combine several approaches, they also combine the advantages of different approaches. However, the combination of the different approaches may increase the computational time.

Table IV.1: Comparison of impuation approaches

	#	Numerical Data	Numerical Data (%)	Categorical Data	Categorical Data (%)	Advantages	Disadvantages
Statistical-based	13	11	85%	7	54%		
Mode/Mean	2	2	100%	2	100%	Easy to implement	High bias
EM Algorithm	3	3	100%	0	0%	Statistical parameters can be well estimated	Need to suppose an appropriate probability model
Regression	5	3	60%	2	40%	Work well if there is a strong correlation between attributes	Replaced values follow a single regression curve
Hot/Cold Deck	3	3	100%	3	100%	Non-parametric, easy to implement	Replaced values depend on the selected auxiliary variables
Machine learning-based	33	31	94%	12	36%		
KNN	7	7	100%	7	100%	No need of attribute relationship discovery, suitable for different amounts and types of data	Computationally costly
Clustering	7	7	100%	0	0%	No need of attribute relationship discovery	Not suitable for categorical data bacause of the calculation of centroid
Other Supervised Learning	13	12	92%	4	31%	High accuracy when the model is well fitted	limitation of data type and bias in certain contexts
Neural Network	6	5	83%	1	17%	High accuracy	Computationally costly, bad performance when few sample data
Rule-based	8	8	100%	8	100%		
Editing Rule	1	1	100%	1	100%	Replaced values are robust	Replaced data may not exist in the dataset, requires the master data
Dependency Rule	3	3	100%	3	100%	Replaced values are robust	Replaced data may not exist in the dataset
Association Rule	4	4	100%	4	100%	Able to find conditional relationships	Replaced data may not exist in the dataset, defintion of the thresholds
External Source-based	9	9	100%	9	100%		
Crowdsourcing	3	3	100%	3	100%	Able to obtain real information	High budget cost
Web Information	3	3	100%	3	100%	Able to obtain real information	Need appropriate query and time-consuming for searching
Knowledge Base	3	3	100%	3	100%	Able to obtain real information	Need appropriate knowledge base
Hybrid	8	8	100%	6	75%	Combine the advantage of different methods	Computationally costly

100

2.3 Imputation Approaches for DW

The existing work concerning the imputation of the missing data in DWs focus on the imputation of factual missing data. A statistic-based imputation method is proposed in Wu and Barbará (2002) which is able to predict the missing values of measures by combining logistic model and loglinear model. In de S. Ribeiro et al. (2011), a KNN based data imputation for factual missing data is presented. They enrich the fact table by the selection of their attributes in dimension tables. The enrichment provides a better characterization of the fact table tuples. Then KNN algorithm is applied to select the similar tuples to replace the tuples of missing data. In Amanzougarene et al. (2014), the authors propose a hybrid method using constraint programming and KNN for the missing measure imputation. They replace missing measures by a CSP (constraint satisfaction problem) solver using the defined constraints on the measures and the constraints of aggregation functions. The result is adjusted by the reduced domain obtained through KNN. A missing data imputation framework for DWs with multi-granular facts based on linear programming is proposed in Bimonte et al. (2020). They do not propose a specific imputation method, but a framework to optimise the imputation result. They define the total of the difference between the adjusted values and the estimated value obtained by some imputation method as the objective function. The constrains are based on the aggregated facts and they define different constrains for different aggregation functions.

However, to the best of our knowledge, there is no specific imputation approach for dimensions. Therefore, in this manuscript, we propose a hybrid approach for the imputation of missing data in dimensions. Our approach is based on dependencies and KNN. Since DWs have multidimensional schema where there are hierarchies in the dimensions. The different levels of a hierarchy have their functional dependency relationships. So since the functional dependencies can be easily obtained, we can apply the dependency-based imputation approach to replace missing values. However, as we analysed, the disadvantage is that there may be not enough dependency relationships for replacing all missing values. We should thus carry out another imputation. We then propose to replace the remaining missing values by a KNN-based imputation. We choose KNN because as we analysed, it is suitable for different types and amounts of data. The imputation approaches of the other categories suffer from the limit of the data types or may be biased in certain contexts or fit only the large amount of data. The disadvantage of KNN is the high computational cost. But we firstly replace some missing data by dependencies, which can reduce the imputation work for KNN.

Hence we propose Hie-OLAPKNN which combines the hierarchical imputation with a KNN-based imputation method.

3 Hierarchical Dimension Imputation

There exists functional dependency relationships between different levels of a dimension hierarchy. The functional dependency is a robust rule. Knowing the functional dependency relationships, we can know the corresponding higher-granularity level value of a given level value. Thus, we take advantage of such relationships to replace missing values. These relationships exist between the attributes of a hierarchy in a dimension. In addition, if there are identical attributes in different dimensions of a DW, these relationships may also exist between the attributes of different dimensions. Therefore, we have two types of hierarchical dimension imputation which can be classified as intra-dimensional imputation and inter-dimensional imputation, which we explain respectively in this section.

3.1 Intra-dimensional Imputation

Intra-dimension imputation relies on data from the same dimension. There are parameters and weak attributes in a dimension. The functional dependency relationships exist between the parameters of a hierarchies or between a parameter and its weak attributes. So for the imputation of a dimension, we first replace the parameters, then when the parameter imputation is finished, we can carry out the weak attribute imputation. It is important to note that, since the parameter sets of hierarchy are ordered sets, the imputation of parameters is sequential (from the lowest-granularity level to the highest-granularity level parameter). This ensures that imputation is maximal, as the value of a higher-granularity level parameter depends on its lower-granularity level parameters. If we replace the values of a higher-granularity level parameter before the imputation of its lower-granularity level parameters, there may be some missing data of the lower-granularity level parameters which are not yet replaced but which can be used to replace the higher-granularity level parameters so that the imputation is not maximal. Our intra-dimension imputation method is presented in Algo. 9. We first check each parameter in each hierarchy of the DW ($line_{1-2}$). If there exist missing data for this parameter ($line_3$), we search for an instance with the same value in a lower-granularity parameter and whose value exists ($line_{4-5}$). Then, we can then fill in the missing data with this value ($line_6$).

The values of a weak attribute depend on the values of its parameter. Then, for each weak attribute of the parameter we check, if there are missing data ($line_7$), we search for the instance that has the same value of its parameter or a lower-granularity level parameter whose value exists ($line_{8-9}$). The missing weak attribute data can then be supplied by this value ($line_{10}$).

Example 3.1. *Fig. IV.2 shows the hierarchical intra-dimensional imputation. The dimension table D_{12} is the merged dimension in Example 5.3 of Chapter III. For the instance of attribute Department whose IdCus is C_7, there is a null value after the merging. We can find another instance whose IdCus is C_4 which has the same value as it on a lower-granularity parameter City. In the same time, it has a value of the attribute Department,*

Algorithm 9: *IntraImputation*(D)

Input : A dimension D having empty values to be completed

1 **for** $H \in H^D$ **do**
2 **for** *each* $p_v^H \in Param^H$ **do**
3 **for** *each* $i_e^D \in \{i^D \in I^D : i^D.p_v^H \text{ is null}\}$ **do**
4 **while** $p_{v_2}^H \in Param^H \wedge p_{v_2}^H \preceq_H p_v^H$ **do**
5 **if** $\exists i_{e_2}^D \in I^D, i_{e_2}^D.p_{v_2}^H = i_e^D.p_{v_2}^H \wedge i_{e_2}^D.p_v^H$ *is not null* **then**
6 $i_e^D.p_v^H \leftarrow i_{e_2}^D.p_v^H$;
7 **for** *each* $i_{e_3}^D \in \{i^D \in I^D : i^D.w_y^{p_v^H} \text{ is null} \wedge w_y^{p_v^H} \in Weak^H[p_v^H]\}$ **do**
8 **while** $p_{v_3}^H \in Param^H \wedge (p_{v_3}^H \preceq_H p_v^H \vee p_{v_3}^H = p_v^H))$ **do**
9 **if** $\exists i_{e_4}^D \in I^D, i_{e_4}^D.p_{v_3}^H = i_{e_3}^D.p_{v_3}^H \wedge i_{e_4}^D.p_v^H$ *is not null* **then**
10 $i_{e_3}^D.w_y^{p_v^H} \leftarrow i_{e_4}^D.w_y^{p_v^H}$;

IdCus	NameCus	Age	Email	Phone	City	Department	Region	Country	Population	Continent	MemSubLevel	MemLevel
C1	N1	34	C1@e.com	012345	CT1	D1	R1	CN1	1000000	CTN1	SL1	L1
C2	N2	53	C2@e.com	123456	CT2	D3	R1	CN1	1000000	CTN1	SL3	L2
C3	N3	66	C3@e.com	234567	CT2	D3	R1	CN1	1000000	CTN1	SL1	L1
C4	N1	26	C4@e.com	345678	CT3	D4	R3	CN2	1200000	CTN1	SL2	L1
C5	N5	45	C5@e.com	NULL	CT5	NULL	R2	CN1	$NULL \rightarrow 1000000$	$NULL \rightarrow CTN_1$	NULL	L3
C6	N6	32	C6@e.com	456789	CT4	D4	R3	CN2	1200000	CTN1	SL3	L2
C7	N7	41	C7@e.com	NULL	CT3	$NULL \rightarrow D_4$	R3	CN2	$NULL \rightarrow 1200000$	$NULL \rightarrow CTN_1$	NULL	L3
C8	N8	NULL	NULL	567890	NULL	D2	R2	CN1	1000000	CTN1	SL4	NULL
C9	N9	NULL	NULL	678901	NULL	D5	R4	CN3	500000	CTN2	SL5	NULL

D_{12}

Figure IV.2: Hierarchical intra-dimensional imputation

so the value D_4 can be used to replace the empty value as shown in the figure. For the empty value of C_5 and C_7 on the parameter Continent, we can find instances having the same value on parameter Country as them and whose values of Continent (C_3, C_6) are not empty. These values can thus be applied for the imputation. For the empty values of C_5 and C_7 on weak attribute Population, they are replaced in the same way.

3.2 Inter-dimensional Imputation

In a DW, there may be common attributes in different dimensions. Therefore, we can replace missing data with such inter-dimensional common attributes. The main idea of inter-dimension imputation is similar to intra-dimension imputation's, except that instead of searching for parameters in the same hierarchy, we search for common parameters of hierarchies in other dimensions (Algo. 10, $line_{3-4}$ and $line_{9-10}$). When performing the

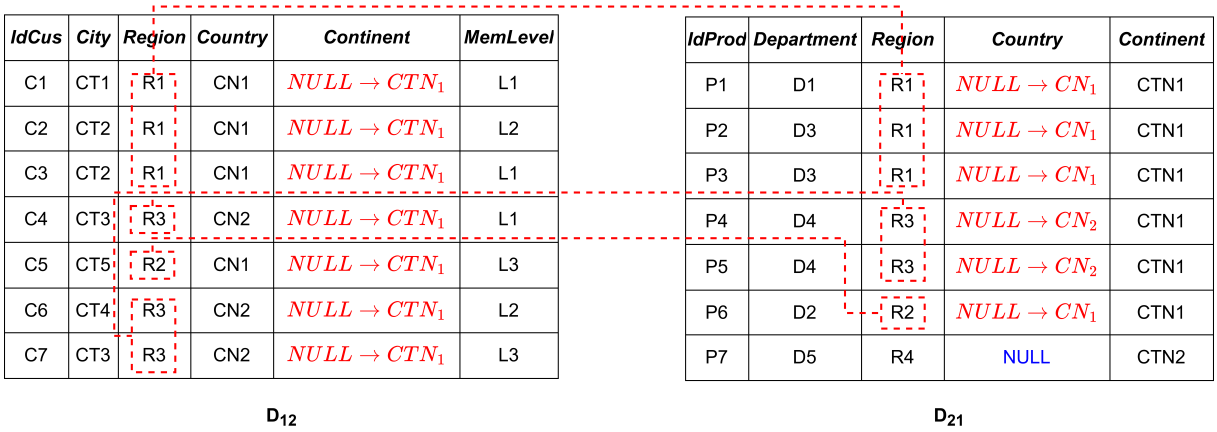
imputation of weak attributes, we must make sure that, in the searched dimension, the searched parameter is semantically identical with the parameter of the weak attribute to be replaced; and that it bears a semantically identical weak attribute ($line_{10-11}$). We say "semantically identical" because in a DW, common attributes may be presented differently in different dimensions. Since in a DW, the designer would normally not use two vocabularies to describe a same entity, but may use the different prefixes or suffixes to distinguish the same entity in different dimensions, we must therefore use string similarity to match attribute names.

Algorithm 10: $InterImputation(D, DW)$

Input : A dimension D having empty values to be completed and the data warehouse to which it belongs DW

1 **while** $p_v^H \in Param^H$, *where* $H \in H^D$ **do**
2 **for** *each* $i_e^D \in \{i^D \in I^D : i^D.p_v^H$ *is null*$\}$ **do**
3 **for** *each* $p_{v_2}^{H_2} \in Param^{H_2}$, *where* $H_2 \in H^{D_2}$, $D_2 \in D^{DW} \wedge D_2 \neq D$ **do**
4 **if** $p_{v_2}^{H_2} \simeq p_v^H$ **then**
5 **while** $p_{v_3}^{H_2} \in Param^{H_2} \wedge p_{v_3}^{H_2} \preceq_{H_2} p_{v_2}^{H_2}$ **do**
6 **if** $\exists i_{e_2}^{D_2} \in I^{D_2} \exists p_{v_4}^H \in Param^H, p_{v_3}^{H_2} \simeq p_{v_4}^H \wedge i_{e_2}^{D_2}.p_{v_3}^{H_2} = i_e^D.p_{v_4}^H \wedge i_{e_2}^{D_2}.p_{v_2}^{H_2}$ *is not null* **then**
7 $i_e^D.p_v^H \leftarrow i_{e_2}^{D_2}.p_{v_2}^{H_2}$;
8 **for** *each* $i_{e_3}^D \in \{i^D \in I^D : \exists w_y^{p_v^H} \in Weak^H[p_v^H],\ i^D.w_y^{p_v^H}$ *is null*$\}$ **do**
9 **for** *each* $p_{v_5}^{H_3} \in H_3$, *where* $H_3 \in H^{D_3}$, $D_3 \in D^{DW} \wedge D_3 \neq D$ **do**
10 **if** $p_{v_5}^{H_3} \simeq p_v^H \wedge \exists w_{y_2}^{p_{v_5}^{H_3}} \in Weak^{H_3}[p_{v_5}^{H_3}], w_{y_2}^{p_{v_5}^{H_3}} \simeq w_y^{p_v^H}$ **then**
11 **while** $p_{v_6}^{H_3} \in Param^{H_3} \wedge (p_{v_6}^{H_3} \preceq_{H_3} p_{v_5}^{H_3} \vee p_{v_6}^{H_3} \simeq p_v^H)$ **do**
12 **if** $\exists i_{e_4}^{D_3} \in I^{D_3} \exists p_{v_7}^H \in Param^H, p_{v_6}^{H_3} \simeq p_{v_7}^H \wedge i_{e_4}^{D_3}.p_{v_6}^{H_3} = i_{e_3}^D.p_{v_7}^H \wedge i_{e_4}^{D_3}.w_{y_2}^{p_{v_6}^{H_3}}$ *is not null* **then**
13 $i_{e_3}^D.w_y^{p_v^H} \leftarrow i_{e_4}^{D_3}.w_{y_2}^{p_{v_6}^{H_3}}$;

Example 3.2. *The inter-dimensional imputation for the merged dimensions D_{12} and D_{21} of Example 5.4 in Chapter III is shown in Fig. IV.3. D_{12} and D_{21} have common parameter Region, which is a lower-granularity parameter of the missing value attribute Continent of D_{12} and of the missing value attribute Country of D_{21}. For the instances of D_{12} having missing values on Continent, we can find the instances of D_{21} having the same values of Region and replace the missing values with the values of Continent of D_{21}. Missing values of D_{21}.Country can also be replaced in the same way.*



IdCus	City	Region	Country	Continent	MemLevel
C1	CT1	R1	CN1	$NULL \rightarrow CTN_1$	L1
C2	CT2	R1	CN1	$NULL \rightarrow CTN_1$	L2
C3	CT2	R1	CN1	$NULL \rightarrow CTN_1$	L1
C4	CT3	R3	CN2	$NULL \rightarrow CTN_1$	L1
C5	CT5	R2	CN1	$NULL \rightarrow CTN_1$	L3
C6	CT4	R3	CN2	$NULL \rightarrow CTN_1$	L2
C7	CT3	R3	CN2	$NULL \rightarrow CTN_1$	L3

D_{12}

IdProd	Department	Region	Country	Continent
P1	D1	R1	$NULL \rightarrow CN_1$	CTN1
P2	D3	R1	$NULL \rightarrow CN_1$	CTN1
P3	D3	R1	$NULL \rightarrow CN_1$	CTN1
P4	D4	R3	$NULL \rightarrow CN_2$	CTN1
P5	D4	R3	$NULL \rightarrow CN_2$	CTN1
P6	D2	R2	$NULL \rightarrow CN_1$	CTN1
P7	D5	R4	NULL	CTN2

D_{21}

Figure IV.3: Hierarchical inter-dimensional imputation

3.3 Hierarchical Imputation Order

When there are common attributes among different dimensions, we can carry out both intra- and inter- dimensional imputation. It is necessary to define the order of these two types of imputation to replace a maximal number of values with the least operations. The intra-dimensional imputation is based on functional dependency relationships of the dimensional values. Thus if we do not add other new values into the dimension, the number of the relationships will not change. The intra-dimensional imputation is indeed a process of spreading the existing FD relationships of the dimension to every instance. However, by launching inter-dimensional imputation, we may add some new FD relationships of other dimensions' values into the dimension whose values are to be replaced. If we carry out intra-dimensional imputation before inter-dimensional imputation, the newly added FD relationships by inter-dimensional imputation may not be applied to every instance. We should thus repeat intra-dimensional imputation to apply them for all the dimension. Therefore, the proper imputation order should be first carrying out inter-dimensional imputation and then intra-dimensional imputation. We can first add new FD relationships and then apply the original FD relationships and the new ones to every instance so that we replace a maximal number of values. We take an example to show why can not launching intra-dimensional imputation before the inter-dimensional imputation.

Example 3.3. *For the original dimensions D_{12} and D_{21} in Fig. IV.4 and in Fig. IV.5, there are common attributes between them so we can carry out inter-dimensional imputation to replace the missing values. These 2 figures show respectively the cases where we first launch the intra-dimensional imputation and first launch the inter-dimensional imputation. For the instance of D_{12} whose IdCus is C_1, its missing value of attribute Continent can be replaced by the instances of D_{21} whose value of Region is the same. However since the imputation of this value is accomplished with the aid of the attribute Region, and the Region of C_3 is missing, the missing value of Continent of C_3 can not be replaced during the inter-dimensional imputation. But it can then be replaced by intra-dimensional imputation with the aid of the values of Country. So if we first launch intra-dimensional imputation, as shown in Fig. IV.4, we should carry out three*

times of imputations (intra-dimensional, inter-dimensional, intra-dimensional) to be able to replace all missing values that can be replaced. If we first launch inter-dimensional imputation, as shown in Fig. IV.5, we only need two times of imputations (inter-dimension, intra-dimensional).

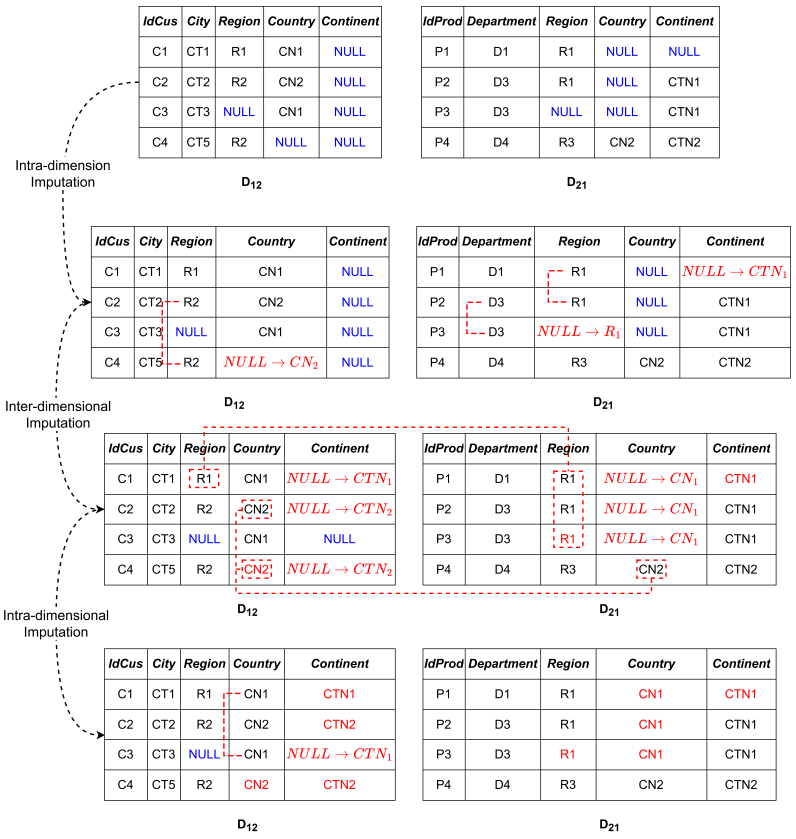


Figure IV.4: Example of first launching intra-dimensional imputation

4 Dimension Instance Distance

As mentioned in Section 1, the hierarchical imputation suffers from the limit of sparsity. Thus, after the hierarchical imputation, we propose to use the KNN-based algorithm named OLAPKNN to replace the remaining missing data. In KNN algorithm, the value or the class of an object is obtained based on its k-nearest neighbors. So in KNN imputation, the replaced missing value is obtained based on the k-nearest neighbors of the missing data instance. We should thus calculate the distance between dimension instances containing missing data to be replaced and other instances in the dimension. In a dimension D, for an instance $i_1 \in I^D$ containing missing data in a hierarchy $H_1 \in H^D$, and another instance $i_2 \in I^D$, we propose to calculate their distance by 4 levels:

- The **dimension instance distance** is the final distance between two instances i_1 and i_2, denoted by $\Delta(i_1, i_2)$. Since the attributes on the same hierarchy have their dependency relationships, we consider the attributes of a hierarchy as an entirety. $\Delta(i_1, i_2)$ is thus calculated by the weighted sum of the **hierarchy instance**

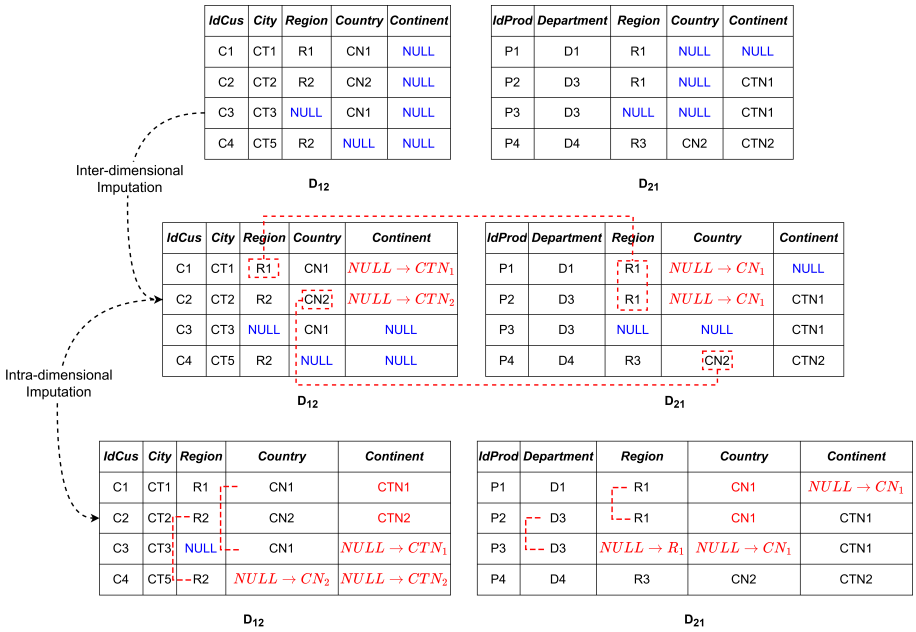


Figure IV.5: Example of first launching inter-dimensional imputation

distances.

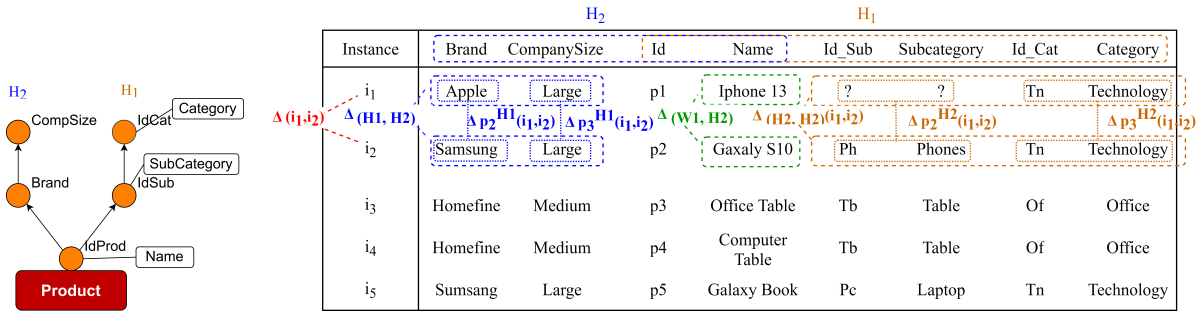
- The **hierarchy instance distance** is the distance of the attributes of a hierarchy $H_2 \in H^D$ i.e. distance between $\{i_1.a_1 \in i_1 : a_1 \in A^{H_2}\}$ and $\{i_2.a_1 \in i_2 : a_1 \in A^{H_2}\}$, denoted by $\Delta_{H_2}(i_1, i_2)$. It is calculated by the weighted sum of the **hierarchy level instance distances**. The lowest-granularity level of each hierarchy is the same i.e. dimension identifier with its weak attributes, so we consider the hierarchy instance distance from the second level of the hierarchy and we regard each weak attribute of *id* as a hierarchy containing only one parameter.

- The **hierarchy level instance distance** is the instance distance between the attributes of a level l on a hierarchy H_2 i.e. distance between $\{i_1.a_2 \in i_1 : a_2 \in p_l^{H_2} \cup Weak^{H_2}[p_l^{H_2}]\}$ and $\{i_2.a_2 \in i_2 : a_2 \in p_l^{H_2} \cup Weak^{H_2}[p_l^{H_2}]\}$, denoted by $\Delta_{p_l^{H_2}}(i_1, i_2)$. It is calculated by the average of the instance distances of the level's parameter and weak attributes (**attribute distances**).

- The **attribute distance** is the instance distance of an individual attribute $a_u \in A^D$ i.e. distance between $i_1.a_u$ and $i_2.a_u$, denoted by $\Delta(i_1.a_u, i_2.a_u)$.

We next explain how to go from calculating the attribute distance to calculating the hierarchy level instance distance then to calculating the hierarchy instance distance and finally obtaining the dimension instance distance between i_1 and i_2 with a series of examples.

Example 4.1. *Given a dimension Product containing two hierarchies H_1 and H_2 whose schema and instances are shown in Fig. IV.6. Instance i_1 contains missing values on H_1, Fig. IV.7 shows the calculation of the distance $\Delta(i_1, i_2)$ between i_1 and another instance*



(a) Schema (b) Instances

Figure IV.6: Schema and instances of dimension *Product*

i_2.

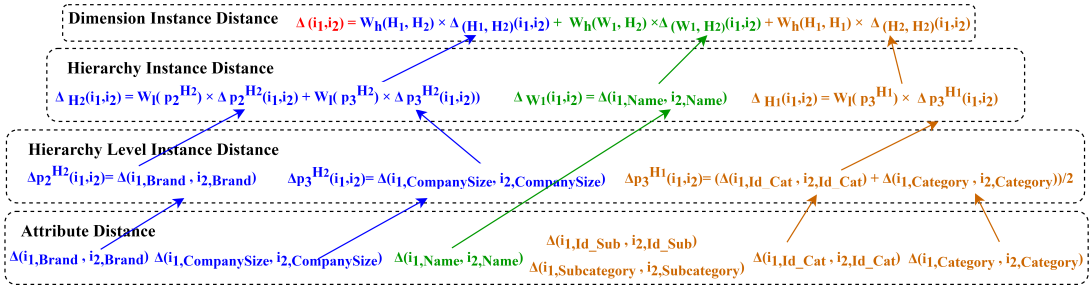


Figure IV.7: Distance between i_1 and i_2

4.1 Attribute Distance

There are different attribute data types which can be mainly classified into numerical and textual. For numerical data, we propose to use normalized distance of numerical data (Han et al., 2012) because it is a distance normalized to the range between 0 and 1.

For textual data, the different distance metrics can be classified into semantic distance and syntactic distance (Wang and Dong, 2020). For the strings having semantically meaningful information, the semantic distance is more accurate than syntactic distance because two strings may describe the similar or identical entity but without being syntactically similar. Therefore, we first apply semantic distance e.g. cosine distance based on word2vec (Jatnika et al., 2019). If the attribute value can not be found in the model, which means that the strings do not having semantically meaningful information, we can then use the syntactic distance e.g. normalized Levenshtein Distance (Yujian and Bo, 2007).

For an attribute a_{u_1}, if $i_1.a_{u_1}$ is missing, then $\Delta(i_1.a_{u_1}, i_2.a_{u_1})$ cannot be calculated and is not taken into count for the distance calculation. For an attribute a_{u_2}, if $i_2.a_{u_2}$ is missing, then $\Delta(i_1.a_{u_2}, i_2.a_{u_2})$ is obtained by the average distance between $i_1.a_{u_2}$ and other instances whose value of a_{u_2} is not missing.

Example 4.2. *Following the calculation rules of the attribute distance, we obtain* $\Delta(i_1.Brand,$

$i_2.Brand) = 0.71$, $\Delta(i_1.CompanySize, i_2.CompanySize) = 0$, $\Delta(i_1.Name, i_2.Name) = 0.8$, $\Delta(i_1.IdCat, i_2.IdCat) = 0$, $\Delta(i_1.Category, i_2.Category) = 0$. *Since* $i_1.IdSub$ *and* $i_1.Subcategory$ *are missing,* $\Delta(i_1.IdSub, i_2.IdSub)$ *and* $\Delta(i_1.Subcategory, i_2.Subcategory)$ *cannot be calculated and are not taken into count for the calculation of* $\Delta(i_1, i_2)$.

4.2 Hierarchy Level Instance Distance

The hierarchy level instance distance $\Delta_{p_l^{H_2}}(i_1, i_2)$ is calculated as (IV.1).

$$\Delta_{p_l^{H_2}}(i_1, i_2) = \frac{\Delta(i_1.p_l^{H_2}, i_2.p_l^{H_2}) + \sum\limits_{w \in Weak[p_l^{H_2}]} \Delta(i_1.w, i_2.w)}{1 + |Weak[p_l^{H_2}]|} \tag{IV.1}$$

As a hierarchy level may contain several attributes including a parameter and some weak attributes, we obtain the average distance of each attribute as the hierarchy level instance distance. As we mentioned that we only consider the levels from the second level of each hierarchy, we do not calculate the distance for the first level of hierarchies.

Example 4.3. *According to (IV.1), for the levels in* H_1, *we have* $\Delta_{p_3}^{H_1}(i_1, i_2) = (0 + 0)/2 = 0$. *As the parameter and weak attribute values of the second level* $i_1.IdSub$ *and* $i_1.Subcategory$ *are missing, the distance of this level is not taken into account. For* H_2, *since the two levels contain only one parameter without weak attribute, their hierarchy level is equal to the attribute distance of the parameter, so we have* $\Delta_{p_2}^{H_2}(i_1, i_2) = 0.71$, $\Delta_{p_3}^{H_2}(i_1, i_2) = 0$.

4.3 Hierarchy Instance Distance

The hierarchy instance distance is calculated as (IV.2), where $W_l(p_l^{H_2})$ is the hierarchy level weight.

$$\Delta_{H_2}(i_1, i_2) = \sum_{p_l^{H_2} \in H_2 \setminus \{id\}} W_l(p_l^{H_2}) \Delta_{p_l^{H_2}}(i_1, i_2) \tag{IV.2}$$

The hierarchy instance distance is calculated by multiplying the distance of each hierarchy level with a hierarchy level weight and by adding them together. The hierarchy level weight is considered because the parameters on the lower-granularity levels have thinner granularity, their weight for measuring the hierarchy instance distance should be higher. For two instances, it is harder for two instances to be similar on a lower-granularity level than on a higher-granularity level. For a weak attribute $w \in Weak^{H_2}[id]$ of the first hierarchy level, $\Delta_w(i_1, i_2) = \Delta(i_1.w, i_2.w)$.

Hierarchy Level Weight The lower granularity-level parameter has higher hierarchy level weight. Thus, we propose two hierarchy level weights: one is based on the cardinal-

ities of the parameters and another is an incremental weight.

- For the cardianlity-based weight, we consider the number of the distinct values of the level as the portion of the weight. Thus for the cardianlity-based hierarchy level weight of the lth level at H_2 is calculated as (IV.3), where $dv(n)$ denotes the number of distinct values of the nth level.

$$W_l^c(p_l^{H_2}) = \frac{dv(l)}{\sum_{j=2}^{|Param^{H_2}|} dv(j)} \tag{IV.3}$$

- However, when the cardinality ratio between certain parameters is very large, the cardinality-based weight may be biased. So we also propose another type of incremental hierarchy level distance weight. For the incremental weight, we consider the weight of the highest-granularity as one portion and it increases by one portion for each neighboring lower-granularity level. The total weight should be equal to 1, thus the incremental hierarchy level weight of the lth level at H_2 is calculated as (IV.4).

$$W_l^i(p_l^{H_2}) = \frac{2(|Param^{H_2}| - l + 1)}{|Param^{H_2}|^2 - |Param^{H_2}|} \tag{IV.4}$$

Example 4.4. *Our example has only 5 instances, so we can use cardinality-based weight to get hierachy level weight. We thus have for* H_1*:* $W_l(p_2^{H_1}) = 3/(3+2) = 0.6$ *and* $W_l(p_3^{H_1}) = 2/(3+2) = 0.4$*. For* H_2*:* $W_l(p_2^{H_2}) = 3/(3+2) = 0.6$ *and* $W_l(p_3^{H_2}) = 2/(3+2) = 0.4$*. We can then calculate the hierarchy instance distances:* $\Delta_{H_1}(i_1, i_2) = 0.4 \times 0 = 0$*,* $\Delta_{H_2}(i_1, i_2) = 0.6 \times 0.71 + 0.4 \times 0 = 0.426$*,* $\Delta_{w_1}(i_1, i_2) = 0.8$*.*

4.4 Dimension Instance Distance

The dimension instance distance $\Delta(i_1, i_2)$ is calculated as (IV.5), where $W_h(H_1, H_2)$ and $W_h(H_1, w)$ are hierarchy weights of H_2 and w with respect to H_1.

$$\Delta(i_1, i_2) = \sum_{H_2 \in H^D} W_h(H_1, H_2)\Delta_{H_2}(i_1, i_2) + \sum_{w \in Weak^{H_2}[id]} W_h(H_1, w)\Delta_w(i_1, i_2) \tag{IV.5}$$

It is calculated by multiplying each hierarchy instance distance with a hierarchy weight and by adding them together. Since the attributes of a hierarchy provide the same category of information and have the dependency relationships among them, we can consider them as an entirety of attributes. We consider the hierarchy weight because the attribute values of a hierarchy have different correlation and dependency with respect to other different hierarchies. We should thus evaluate to how many degrees the values of the other

hierarchies determine the values of the missing value hierarchy to decide the hierarchy weights.

4.5 Using Dependency Degree as Hierarchy Weight

The dependency degree in the rough set theory measures the degree of the dependency between attributes, so it is applied for the hierarchy weight. The rough set theory is an important mathematical approach to deal with vagueness (Pawlak, 1982; Pawlak and Skowron, 2007). In the rough set theory (Pawlak and Skowron, 2007; Pawlak, 1977), a dataset is called an information system defined as a pair $S = (U, A)$, where U and A are non-empty finite sets, U is a set of objects and A is a set of attributes. For each attribute $a \in A$, it determines a function $f_a : U \to V_a$, where V_a is the domain of attribute a containing the set of values of a. Any subset B of A determines an indiscernibility relation:

$$I(B) = \{(x, y) \in U \times U : f_a(x) = f_a(y), \forall a \in B\} \tag{IV.6}$$

In fact, it can be seen that the indiscernibility relation is indeed a binary equivalence relation containing all the object pairs having the same attribute values on B. The family of all equivalence classes of $I(B)$ is denoted by $U/I(B)$, in short U/B. An equivalence class containing an element x is denoted as $I(B)(x)$, in short $B(x)$. The indiscernibility relation is further used to define the approximations on sets $X \subseteq U$:

$$B_*(X) = \{x \in U : B(x) \subseteq X\} \tag{IV.7}$$

$$B^*(X) = \{x \in U : B(x) \cap X \neq \emptyset\} \tag{IV.8}$$

$B_*(X)$ is the lower approximation of the set X with respect to B which can be certainly classified as X using B, while $B_*(X)$ is the upper approximation of the set X with respect to B which can be possibly classified as X using B.

For an information system $S = (U, A)$, we can distinguish A into two classes $C, D \subseteq A$, which are respectively called condition and decision attributes. We then get a decision system $S = (U, C, D)$, the degree k to which D depends on C, denoted $C \Rightarrow_k D$ is defined as:

$$k = \gamma(C, D) = \frac{card(POS_c(D))}{card(U)} \tag{IV.9}$$

where

$$POS_c(D) = \bigcup_{X \in U/D} C_*(X) \tag{IV.10}$$

and $card(X)$ represents the cardinality of an non-empty set X.

Hierarchy Weight When calculating the hierarchy distance weight, we can consider a decision system $S = (I^D, A_n^{H_2}, A_n^{H_1})$, since we do not take the first level of a hierarchy into account, $A_n^{H_1} = A^{H_1} \setminus (\{id\} \cup Weak^{H_1}[id])$, $A_n^{H_2} = A^{H_2} \setminus (\{id\} \cup Weak^{H_2}[id])$. The second hierarchy level parameters $p_2^{H_1}, p_2^{H_2}$ determine all the other hierarchy attributes in $A_n^{H_1}$ and $A_n^{H_2}$, we can reduce the attribute sets of $A_n^{H_1}$ and $A_n^{H_2}$ to the sets containing only the values of the second hierarchy level parameter $p_2^{H_1}, p_2^{H_2}$. According to (IV.9), the degree k to which H_1 depends on H_2, denoted $H_2 \Rightarrow_k H_1$ is thus defined as:

$$k = \gamma(A_n^{H_2}, A_n^{H_1}) = \gamma(p_2^{H_2}, p_2^{H_1}) = \frac{card(POS_{p_2^{H_2}}(p_2^{H_1}))}{card(I^D)} \tag{IV.11}$$

where $POS_{p_2^{H_2}}(p_2^{H_1}) = \bigcup_{X \in I^D/p_2^{H_1}} p_{2*}^{H_2}(X)$ and $card(X)$ is the cardinality of an non-empty set X, the missing second level parameter values are not taken into account. For H_1 itself, we have $\gamma(A_n^{H_1}, A_n^{H_1}) = 1$.

The hierarchy distance weight of H_2 with respect to H_1 is the ratio of their dependency degree with respect to the sum of the dependency degrees of all hierarchies and first level weak attributes in D with respect to H_1 as (IV.12).

$$W_h(H_1, H_2) = \frac{\gamma(A_n^{H_2}, A_n^{H_1})}{\sum\limits_{H_3 \in H^D} \gamma(A_n^{H_3}, A_n^{H_1}) + \sum\limits_{w \in Weak^{H_1}[id]} \gamma(w, A_n^{H_1})} \tag{IV.12}$$

Example 4.5. *In our example, we have* $card(I^D) = 5$, $card(POS_{p_2^{H_2}}(p_2^{H_1})) = 2$, *so* $\gamma(A_n^{H_2}$,
$A_n^{H_1}) = 2/5 = 0.4$. *In the same way, we can get* $\gamma(w_1, A_n^{H_1}) = 2/5 = 0.4$, *we also have* $\gamma(A_n^{H_1}, A_n^{H_1}) = 1$. *We can thus get the hierarchy weights:* $W_h(H_1, H_2) = 0.4/(0.4 + 0.4 + 1) = 0.22$, $W_h(H_1, H_1) = 1/(0.4 + 0.4 + 1) = 0.56$ *and* $W_h(w_1, H_2) = 0.4/(0.4 + 0.4 + 1) = 0.22$. *We can finally obtain the dimension instance distance* $\Delta(i_1, i_2) = 0.22 \times 0.46 + 0.22 \times 0.8 + 0.56 \times 0 = 0.28$

5 OLAPKNN

5.1 OLAPKNN Overview

The OLAPKNN imputation is shown in Algo. 11. Since there are parameters and weak attributes in a dimension, OLAPKNN is composed of two steps including the imputation of parameters and the imputation of weak attributes. The weak attributes' values are determined by their parameters' values, so we impute the parameters before imputing their weak attributes. In order to respect the dependency constraints of the hierarchy levels, we create candidate lists for possible replaced values in the imputation process. We also consider possible conflicts during the imputation and propose solutions to deal with them. The imputation steps can be briefly summarized as follow.

1. For missing data of each hierarchy ($line_1$), we create candidate lists of the instances containing possible replaced values and select the k nearest neighbors in the candidate lists to replace the missing data ($line_2$).

2. There are weak attributes which can be imputed together with their parameter. Finally for the remaining missing weak attribute data, they are imputed in the similar way ($line_3$).

Algorithm 11: $OLAPKNN(D)$

```
1 for H ∈ H^D do
2 |   imputeParam(D, H) ;
3 |   imputeWeak(D, H) ;
```

Next, we explain in detail the OLAPKNN imputation algorithm.

5.2 Imputation for Parameters by OLAPKNN

5.2.1 Parameter Imputation Order

We first introduce the continuous missing parameter group in order to explain the imputation order for parameters.

Definition 5.1. *For an instance $i_r \in I^D$ in the dimension D containing missing values on parameters of a hierarchy H, all these parameters are in a set $Pm_r^H = \{p_v^H \in Param^H : i_{r,p_v^H}$ is empty$\}$. For the parameters in Pm_r^H, they can be divided into one or several continuous missing parameter groups. A **continuous missing parameter group (CG)** contains one or several parameters which are neighbors on H and are maximal neighbors in Pm_r^H. By neighbors on H, we mean that for the parameter p_{lowest} having the lowest-granularity level in the CG on H and the one $p_{highest}$ having the highest-granularity level, if there exists any parameter $p_{middle} \in Param^H$, such that $p_{lowest} \preceq_H p_{middle} \preceq_H p_{highest}$, then $p_{middle} \in Pm_r^H$. By maximal neighbors in Pm_r^H, we mean that if there exists any parameter $p_{low_2} \in Param^H$, such that $p_{low_2} \preceq_H p_{lowest}$, then $p_{low_2} \notin Pm_r^H$; if there exists any parameter $p_{high_2} \in Param^H$, such that $p_{highest} \preceq_H p_{high_2}$, then $p_{high_2} \notin Pm_r^H$. We call all CGs of a hierarchy H containing a same number of parameters a n-CGs of H, where n denotes the number of parameters.*

Algo. 12 shows the imputation of the parameters. For a given hierarchy H on a dimension D, we carry out the imputation for parameters in the n-CGs by the ascending order of n ($line_1$). We can thus make sure that all the $(n-1)$-CGs instances are imputed so that we can carry out the imputation for the n-CGs based on the existing data. Then for each n-CGs, we look at all possible CG combinations ($line_{2-3}$). Next we verify if there are instances containing missing values for each possible CG ($line_{4-9}$). According to $Definition$ 5.1, the instances of a CG on H have missing values on all parameters of the group. If there is a neighboring lower-granularity or higher-granularity parameters of

the group, the instances do not have missing value on them ($line_9$).

Algorithm 12: $imputeParam(D, H)$

```
1  for ncontinuous ← 1 to |Param^H| − 1 do
2      for i ← 1 to |Param^H| − ncontinuous do
3          P_CG ← Param^H[i : i + ncontinuous − 1] ;
4          p_low, p_high ← ∅ ;
5          if i > 1 then
6              p_low ← Param[i − 1] ;
7          if i < |Param^H| − ncontinuous then
8              p_high ← Param[i + ncontinuous] ;
9          I_missing = {i_r ∈ I^D : (∀p_cg ∈ P_CG, i.r, p_cg = null) ∧ (∃p_low ⟹ i_r.p_low ≠
             null) ∧ (∃p_high ⟹ i_r.p_high ≠ null)} ;
10         lowMap ← Map ;
11         for i_m ∈ I_missing do
12             I_candidate ← getCandidateList(D, P_CG, p_high, i_m, 1) ;
13             vWeightMap ← getVWeightMap(D, i_m, I_candidate, k, P_CG) ;
14             lowMap ←
                 replaceNoPlow(D, H, lowMap, vWeightMap, i_m, P_CG, p_low) ;
15         if ∃p_low then
16             replacePlow(lowMap, P_CG, H, D, p_low) ;
```

Algorithm 13: $getCanList(D, P_{CG}, p_{high}, i_m, parameter)$

```
1  if parameter = 1 then
2      if ∃p_high then
3          I_candidate ← {i_r ∈ I^D : (∃p_cg ∈ P_CG, i_r.p_cg ≠ null)∧(i_r.p_high = i_{r_missing}.p_high)}
             ;
4      else
5          I_candidate ← {i_r ∈ I^D : (∃p_cg ∈ P_CG, i_{r,p_cg} ≠ null)} ;
6  else
7      I_candidate ← {i_r ∈ I^D : (i_r.weak ≠ null)} ;
8  return I_candidate
```

5.2.2 Candidate List

Since some missing data are already replaced by the hierarchical imputation, for the remaining missing data, they can no longer be replaced with the aid of their lower-granularity level parameters. For a value of one parent parameter, there may be several possible values on a child parameter of its. So for a missing data instance of a CG, we can find all possible replaced values based on their neighboring higher parameter and create a candidate list (Algo. 12 $line_{11}$). The candidate list contains not only the candidate replaced values of CG attributes but also the values of all other attributes of the dimension because we need all attribute's value for the calculation of the distances.

Algo. 13 shows the candidate list creation for an instance of a CG. If the neighboring higher-granularity level parameter p_{high} of the CG exists, we search for all the instances having the same values on p_{high} as the CG instance, and containing non-missing values on the CG parameters. Then these instances can be added into the candidate list ($line_{1-3}$). If there does not exist a neighboring higher parameter for a CG, we add all the instances of the dimension which contain non-missing values on the CG parameters into the candidate list ($line_{4-5}$).

5.2.3 Creation of Replaced Value Weight Map

For the CG instance, we can get a map for each possible replaced values in the nearest neighbors with their distance-based weight for the selection of the final replaced value as described in Algo. 14. We first create a map of each instance in the candidate list with its distance with respect to the missing instance ($line_{1-3}$). Then we can select the k nearest candidate instance to create a candidate list if the candidate list contains more than k instances, if not, we can keep all candidate instances ($line_{4-5}$). The selected candidate instances may contain same replaced values, so we create a map of each replaced values with their weight ($line_6$). According to (Dudani, 1976), for a instance i_m of a CG, for a selected candidate list containing k instances, the distance weight of the n nearest instance i_{cn} can be calculated as (IV.13), where i_{ck} denotes the kth nearest instance and i_{c1} denotes the nearest instance. It is to be noted that $W_d(i_m, i_c) = 1$ when $\Delta(i_m, i_{ck}) = \Delta(i_m, i_{c1})$.

$$W_d(i_m, i_c) = \frac{\Delta(i_m, i_{ck}) - \Delta(i_m, i_{cn})}{\Delta(i_m, i_{ck}) - \Delta(i_m, i_{c1})} \tag{IV.13}$$

Thus the weight of a candidate of replaced values is the sum of the weight of the instances which contain them ($line_{4-5}$).

Algorithm 14: $getVWeightMap(D, i_m, I_{candidate}, k, P_{CG})$

```
1 iDistanceMap ← Map ;
2 for i_c ∈ I_candidate do
3     iDistanceMap[i_{c,id}] ← Δ(i_m, i_c) ;
4 if |I_candidate| > k then
5     iDistanceMap ← iDistanceMap.top(k);
6 vWeightMap ← Map ;
7 for i_{c,id} ∈ iDistanceMap.keys() do
      /* addMap(Map, key, value): Create the map if it does not exist. Add the
         value to the existing value if the key exists, assign the value to the key
         if not.                                                              */
8     addMap(vWeightMap, {i_c.p_cg : i_c.p_cg ∈ i_c.P_CG}, Wv(i_m, i_c));
9 return vWeightMap
```

5.2.4 Replacement of Values

To fill in the missing values of CG, we have two cases: the first case (Algo. 12 $line_{13}$) is that there is no lower non-id parameter of the missing parameter group, the second case (Algo. 12 $line_{14-15}$) is that there is such parameter. The difference is that for the second case, we have to take the strictness of hierarchy into account by making sure that a lower parameter value of the CG has only one higher-granularity level parameter after the imputation.

The replacement of missing values in the first case is described in Algo. 15. We can take the values having the highest weight in the weight map ($line_1$) to fill in the missing values of the CG ($line_{2-3}$).

The replacement of missing values in the second case is described in Algo. 15 and Algo. 16. We create a map $lowMap$ for each neighboring lower-granularity level parameter value which corresponds to another map of the each possible replaced value and its total weight (Algo. 12 $line_{10}$). For each instance of the CG, we get the replaced values with the highest value weight (Algo. 15 $line_1$). For the value of its neighboring lower-granularity parameter, we update the replaced values and the weight (Algo. 15 $line_{8-10}$). When all the missing instances of a CG are treated, we get a final $lowMap$. For each value of the neighboring lower-granularity level parameter in $lowMap$, we can take the replaced values with the highest weight to fill in the missing values (Algo. 16 $line_{1-5}$).

Algorithm 15: $replaceNoPlow(D, H, lowMap, vWeightMap, i_m, P_{CG}, p_{low})$

```
1  i_replace, P_CG ← vWeightMap.top(1).key() ;
2  if ∄p_low then
3  |  i_m.P_CG ← i_replace.P_CG ;
4  |  for p_cg ∈ P_CG do
5  |  |  for w^{p_cg} ∈ Weak^H[p_cg] do
6  |  |  |  if i_m.w^{p_cg} = ∅ then
7  |  |  |  |  i_m.w^{p_cg} ← {i_r.w^{p_cg} ∈ I^D : i_r.p_cg = i_m.p_cg}.getOne() ;
8  else
9  |  addMap(lowMap[i_m.p_low], i_replace.P_CG, vWeightMap[i_replace.P_CG]) ;
10 |  addMap(lowMap, i_m.p_low, lowMap[i_m.p_low]) ;
11 return lowMap
```

Example 5.1. *In the example of Fig. IV.6. For the imputation of the missing values of the level of "IdSub" for the instance i_1, we first create a candidate list. Its neighboring higher-granularity level parameter is "IdCat", we can find that the instances i_2 and i_5 have the same value on this parameter as i_1, so they are added into the candidate list. We already got the distance between i_1 and i_2 in the previous example $\Delta(i_1, i_2) = 0.28$, by the same way, we get $\Delta(i_1, i_5) = 0.32$. By IV.13, we get $W_d(i_1, i_2) = 1$ and $W_d(i_1, i_5) = 0$. If we choose $k = 1$, we can thus select the instance i_2 which has the highest replaced value*

Algorithm 16: $replacePlow(lowMap, P_{CG}, H, D, p_{low})$

```
1 for i_m.p_low ∈ lowMap.keys() do
2     vWeightMap ← lowMap[i_m.p_low].top(1) ;
3     i_replace.p_low ← vWeightMap.key() ;
4     for i_m ∈ {i_r ∈ I^D : i_r.p_low = i_m.p_low} do
5         i_m.P_CG ← i_replace.P_CG ;
6         for p_cg ∈ P_CG do
7             for w^{p_cg} ∈ Weak^H[p_cg] do
8                 if i_m.w^{p_cg} = ∅ then
9                     i_m.w^{p_cg} ← {i_r.w^{p_cg} ∈ I^D : i_r.p_cg = i_m.p_cg}.getOne() ;
```

weight. We then replace the IdSub of i_1 with "Ph". The weak attribute SubCategory of this level of i_2 is not empty, so we can also replace the missing value of SubCategory of i_1 by "Phone".

5.3 Imputation of Weak Attributes

In this part, we discuss the imputation of weak attributes which is performed during (Algo. 11 $line_4$) and after (Algo. 11 $line_5$) the imputation of parameters.

5.3.1 Weak Attribute Imputation During Parameter Imputation

When replacing a missing value of a parameter, if there are missing values on its weak attributes, they can be replaced by the non-missing corresponding weak attribute value of the replaced parameter since the weak attribute values are determined by the value of the corresponding parameter (Algo. 15 $line_{4-7}$, Algo. 16 $line_{6-9}$).

5.3.2 Weak Attribute Imputation After Parameter Imputation

The imputation for weak attributes after the parameter imputation is described in Algo. 17. For a hierarchy H of dimension D, we search for the missing values for each weak attribute of each parameter ($line_{1-4}$). As the weak attribute value is determined by its parameter, we create a map $pwMap$ for each parameter value corresponds to another map of the each possible replaced weak attribute value and its total weight ($line_5$). Then for each instance containing missing values of the weak attribute, we create a candidate list ($line_{6-7}$) containing instances of the dimension having non-missing values on the weak attribute (Algo. 13 $line_{6-7}$). We then get the replaced value weight map ($line_8$) and take the value having the highest weight as the candidate replaced value ($line_9$). We can thus then update the replaced weak attribute value and its weight for the current parameter value ($line_{10-11}$). When we finish selecting the replaced values and getting their weights for each value of the parameter, we can choose the value having the highest weight as the final replaced value for the weak attribute ($line_{12-13}$).

Algorithm 17: $imputeWeak(D, H)$

```
for p ∈ Param^H do
    if p ∈ Weak^H.keys() then
        for w ∈ Weak^H[p] do
            I_missing = {i_r ∈ I^D : i_{r,w} = null} ;
            pwMap ← Map ;
            for i_m ∈ I_missing do
                I_candidate ← getCandidateList(I_missing, D, [w], p_high, 0) ;
                vWeightMap ← getVWeightMap(D, i_m, I_candidate, k, [w]) ;
                replaceweak(type, vWeightMap, w) ;
                i_{replace,w} ← vWeightMap.top(1).key() ;
                addMap(pwMap, i_m.p, pwMap[i_m.p]) ;
            for i_{m,p} ∈ pwMap.keys() do
                i_{m,w} ← pwMap[i_m.p].top(1) ;
```

6 Experimental Assessments

To validate the effectiveness and efficiency of out proposed Hie-OLAPKNN algorithm, we implement our algorithm and conduct experiments with different datasets.

6.1 Dataset

We apply one benchmark dataset **TPCH** (as in chapter III) and four real-world datasets from the relational dataset repository site[1] including **Adventure** which is a dataset about a fictious, multinational bicycle manufacturer called Adventure Works Cycles; **F1** which has information concerning Formula 1 races, starting from the 1950 season until today; **GoSales** is a dataset from IBM containing information about daily sales, methods, retailers, and products of a fictitious outdoor equipment retail chain "Great Outdoors" (GO); **Organisation** is a geography dataset from University of Göttingen describes information about 185 countries. These dataset are in the relational database form. We create a data warehouse for each dataset according to the schema provided in their source and based on the attribute semantics. The DW schema of each dataset and the number of tuples in each dimension are shown in Appendix B. The DWs are integrated in R-OLAP format with Oracle 11g. The benchmark dataset is used to validate the application of combing intra- and inter- dimensional imputation in case of dimensions having same parameters in hierarchical imputation. The four real-world datasets are used to validate the application of combining hierarchical imputation and OLAPKNN and to compare our proposed algorithm to other approaches from the literature.

[1] `https://https://relational.fit.cvut.cz/`

6.2 Experimental methodology

6.2.1 Experimental Metrics

The objective of the experiments is to validate the effectiveness and the efficiency of our algorithm and the strictness of the hierarchies with the imputed data.

- For the effectiveness, we use the accuracy metrics of recall, precision and F-score defined as follow.
  - **Recall** $= \frac{\{Imputed\} \cap \{True\}}{\{True\}}$;
  - **Precision** $= \frac{\{Imputed\} \cap \{True\}}{\{Imputed\}}$;
  - **F-score** $= \frac{2 \times Precision \times Recall}{Precision + Recall}$,

    where $\{Imputed\}$ is the set of the imputed values by the algorithms and $\{True\}$ is the set of the ground truth imputed values.
- For the efficiency, we use **run time** as the metric.
- For the strictness of the hierarchies, we define a metric of **strictness degree**.

  The strictness degree can be calculated for a parameter p with respect to a single attribute a which may be its higher-granularity level parameter or its corresponding weak attribute. We first count the number of distinct values $N_d(p)$ for the parameter. Then for each distinct value, the corresponding value of a should be single. So this means that there should be $N_d(p)$ strict relationships. We then verify each relationships in the dimension with imputed values and get the number $N_r(p, a)$ of relationships that do not bear conflicts. Finally, the strictness degree of p with respect to a is calculated as (IV.14).

$$SD(p, a) = \frac{N_r(p, a)}{N_d(p)} \tag{IV.14}$$

  The strictness degree can also be calculated for several parameters $P = \{p_1, ...p_n\}$ with respect to several attributes $A = \{a_1, ...a_n\}$. Here, we also have a mapping $M = P \rightarrow A$ indicating for each parameter, the strictness degree is calculated with respect to which attributes. So the number of relationships can be obtained by the addition of multiplying the distinct value of each parameter with the number of its corresponding weak attributes. We also verify these relationships and obtain the number of relationships that do not bear conflicts $N_r(p_i, a_i)$ for each parameter $p_i \in P$ and each of its corresponding weak attribute $a_i \in M[p_i]$. Then we can get the number of all relationships without conflict. Finally, the strictness degree

$SD(P, A, M)$ can be calculated as (IV.15).

$$SD(P, A, M) = \frac{\sum_{p_i \in P} \sum_{a_i \in M[p_i]} N_r(p_i, a_i)}{\sum_{p_i \in P} N_d(p_i)|M[p_i]|} \tag{IV.15}$$

6.2.2 Experimental Strategies

We apply different missing rates (1%, 5%, 10%, 20%, 30%, 40%) for the categorical attributes. To generate a certain percentage of missing data for an attribute, we sort randomly all the instances and remove attribute data of the first certain percentage of instances. For each dataset, we carry out 20 tests and get the average metrics.

We conduct the experiments with two strategies. The calculation of the strictness degree is different in these two strategies.

1. The first one is to apply each missing rate for each single attribute. We then obtain the average metrics for each missing rate of the application of all categorical attributes.

   Then we discuss the calculation of the strictness degree in different cases. Since the values of dimension identifier are all unique, we do not need to calculate the strictness degree of the identifier with respect to other attributes. Thus, we have several cases as follows.

   - If the attribute is a parameter, and it does not have a non-id lower-granularity level parameter, we calculate the strictness degree $SD(P, A, M)$ of the parameter with respect to its higher-granularity level parameter and its weak attributes. Therefore, P contains only the parameter itself, A contains its higher-granularity level parameter and its weak attributes, M is obtained by the dimension schema. If the parameter does not have higher-granularity level parameter or weak attributes, we do not calculate the strictness degree for it.

   - If the attribute is a parameter, and it has a non-id lower-granularity level parameter, we calculate the strictness degree $SD(P, A, M)$ of the parameter with respect to its higher-granularity parameter and its weak attributes. In addition, we should also calculate the strictness degree of its non-id lower-granularity level parameter with respect to itself. Therefore, in this case, P contains the parameter itself and its non-id lower-granularity level parameter, A contains the parameter itself, its higher-granularity level parameter and its weak attributes, M is obtained by the dimension schema.

   - If the attribute is a weak attribute and its corresponding parameter is non-id, we calculate the strictness degree $SD(p, a)$ of its corresponding parameter with respect to itself. Therefore, p is the corresponding parameter of the weak

attribute and a is the weak attribute itself.

2. The second strategy is to apply each missing rate for all categorical attributes and obtain the average metrics of each missing rate.

   We calculate the strictness degree $SD(P, A, M)$ of each non-id parameter with respect to their higher-granularity parameter and weak attributes. Therefore, P contains all non-id parameters of the dimension, A contains the higher-granularity parameter and weak attributes of each non-id parameters of the dimension, M is obtained by the dimension schema.

We carry out these two strategies because the missing data may exist only in one attribute, which means that we only replace the missing values of one attribute based on the instances containing no missing value. Missing data may also exist in several attributes, which means that we may replace missing values based on the instances containing missing values, which may have impacts on the accuracy.

6.2.3 Comparison with Other Approaches

We compare our Hie-OLAPKNN algorithm with some other approaches to verify is it performs better. The approaches include the single algorithm of Hie-OLAPKNN, i.e. hierarchical imputation and OLAPKNN, and approaches from the literature. They are presented as follows.

H: This is the hierarchical imputation of Hie-OLAPKNN algorithm.

OLAPKNN: This is the OLAPKNN algorithm of Hie-OLAPKNN algorithm

KNN (Domeniconi and Yan, 2004): This approach uses the basic KNN algorithm to generate the replaced values for missing data.

NB (Garcia and Hruschka, 2005): This is a machine learning-based imputation approach based on the naive bayes algorithm.

MIBOS (Wu et al., 2012): This is a statistical-based hot deck imputation method.

6.2.4 Parameter choice of the algorithms

For the algorithms applied in the experiments, there are parameters in **KNN** and **(Hie-)OLAPKNN**. For **KNN**, we have to choose the k, and for **(Hie-)OLAPKNN**, we have to choose the k as well as the hierarchy level weight which may be cardinality-based weight (w_c) or incremental weight (w_i). To decide the value of these parameters, we test with different k between 1 and 10 for **KNN** and **(Hie-)OLAPKNN** and with different hierarchy level weight for **(Hie-)OLAPKNN**. We test with the missing rate of 20%, we still carry out 20 times of tests and take the average F-score. Finally, we choose the k and hierarchy weight having the best F-score for the experiments.

Table IV.2: Algorithms' parameters

Algorithm	Parameter	Dataset and dimension						
		TPCH		Adventure	F1	GoSales		Organisation
		Cus	Supp	Prod	Race	Prod	Ret	Organisation
KNN	k			1	2	1	1	3
(Hie-)OLAPKNN	k	2	2	1	2	2	1	1
	Hierarchy level weight	w_c	w_c	w_i	w_c	w_i	w_i	w_c

6.3 Results and analysis for Experiment1

6.3.1 Effectiveness

The effectiveness for single attribute imputation strategy and multiple attribute imputation of experiment1 are respectively shown in Fig. IV.8 and Fig. IV.10.

We can observe that there is no difference with respect to the effectiveness among using only inter- or intra- dimensional hierarchical imputation and using both. There is no difference neither among using these three types of hierarchical imputation in **Hie-OLAPKNN**. By analysing the imputation results, we find that the application of only intra- or inter-dimensional hierarchical imputation is able to replace all possibly replaced missing values (attributes expect for the weak attributes of the first level and the first and second level parameters). Then we analyse the data and find that for the attributes that can be replaced, the distinct value ratio is (number of distinct values divided by number of total values) very low. For example, in the attribute *Nationkey* of dimension *Customer*, there are 1500 values while there are only 25 distinct values, thus the distinct ratio is 0.017, which is very low. The distinct ratio of attributes *Regionkey* in dimension *Customer* is 0.003. The distinct ratios of attribute *Nation* and *Regionkey* in dimension *Supplier* are respectively 0.025, and 0.005. These attributes work as lower-granularity level parameters to which we reference for searching for replaced values. Moreover, for an attribute, each distinct value occurs uniformly. This means that when we generate missing data with missing rates from 1% to 40%, it is hard to have all values of a distinct value get deleted. The two dimensions also have the same distinct values on these attributes. Thus, for the missing values that can be possibly replaced, we can always find an instance having the same value on a lower parameter. The missing values can then always be replaced even by using only intra- or inter-dimensional hierarchical imputation.

As this experiment cannot show the difference between using only intra- or inter-dimensional imputation and using both, we design another strategy for generating missing data. As we discussed, *Nationkey* and *Regionkey* are parameters to which we reference for searching for replaced values. So we first generate missing data by deleting all values of certain distinct values for these attributes and then generate random missing data for these and other attributes. *Regionkey* has 5 distinct values and they uniformly distribute in the dimensions. Therefore, to delete all values of a distinct value for this attribute, we delete about 20% missing values. So we take three missing rates 30%, 50% and 70%

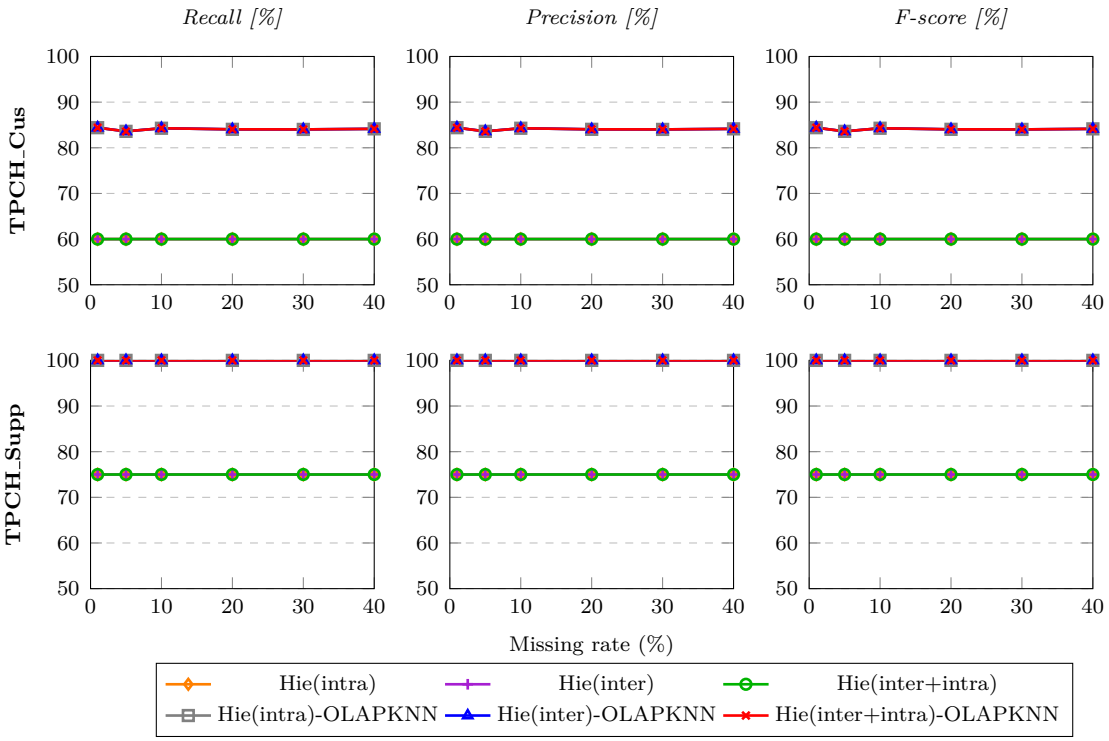


Figure IV.8: Effectiveness results of single attribute imputation of experiment1

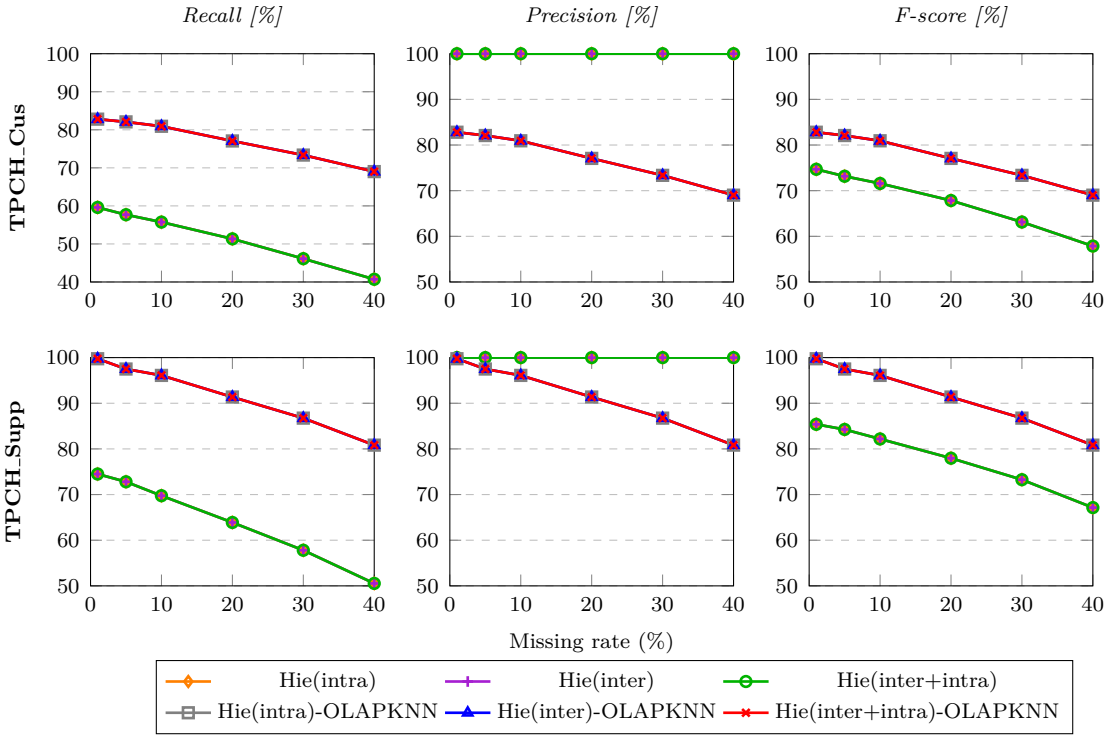


Figure IV.9: Effectiveness results of multiple attribute imputation of experiment1

in order to delete all values of 1, 2 and 3 distinct values and then generate randomly about 10% missing values for *Regionkey*. *Nationkey* has 25 distinct values and they also uniformly distribute in the dimensions. Thus by applying these missing rates, we delete all values of 5, 10 and 15 distinct values and then generate randomly about 10% missing values for *Nationkey* and *Regionkey*. For the other attributes, we apply these missing rates to generate randomly missing data as the previous strategy.

The effectiveness results for this second missing data generation strategy is shown in Fig. IV.8 and Fig. IV.10.

In the three **Hie** imputations, we can observe that they all have 100% precision. The combination of inter- and intra-dimensional imputation performs better than only using inter- and intra-dimensional imputation in terms of recall and F-score. For example, for $TPCH\_Cus$, it has recall and F-score of up to respectively 10.86% and 11.3% higher with respect to the intra-dimensional imputation; and respectively 3.04% and 4.4% higher with respect to the inter-dimensional imputation. It means that by applying both inter- and intra-dimensional imputation, there are more missing values which are replaced. Since we delete randomly distinct values from the two dimensions, after the generation of missing data, there may be (1) deleted distinct values that do not exist in one of the two dimensions and (2) deleted distinct values that do not exist in both dimensions. As the attributes of these deleted distinct values are used as the reference of lower-granularity level parameter, by combining inter- and intra-dimensional imputation, the missing values which should be replaced based on deleted distinct values not existing in both dimensions cannot be replaced. However, by using only inter- or intra-dimensional imputation, the missing values which should be replaced based on these two types of deleted distinct values cannot be replaced. That is why the combination of inter- and intra-dimensional imputation replaces more missing values than the application of only one of them. Between inter- and intra-dimensional imputation, we see that in these two dimensions, inter-dimensional imputation replace more missing values than intra-dimensional imputation. The inter-dimensional imputation has a recall of up to respectively 10.13% and 4.39% higher in $TPCH\_Cus$ and $TPCH\_Supp$. This is because of our missing generation strategy. For a dimension, we delete all values of certain distinct values, so there are more missing data for these distinct values. However, they cannot be replaced by intra-dimensional imputation. But as the distinct values are selected randomly in different dimensions, the deleted distinct values may exist in other dimensions. So by inter-dimensional imputation, more data can be replaced. But this does not mean that the inter-dimensional imputation is always better than intra-dimensional imputation. Because the effectiveness of hierarchical imputation depends on the dimension instances, i.e. whether a missing value can be replaced depends on whether there exists an instance having same lower-granularity level parameter.

For the three Hie-OLAPKNN imputations with different hierarchical imputations, we can observe that Hie-OLAPKNN with intra-dimensional imputation performs worst (e.g.

F-score of up to 12,50% and 12,59% lower with respect to inter-dimensional imputation and the combination of both in $TPCH\_Cus$) since it applies intra-dimensional imputation and then OLAPKNN, which are both based on intra-dimensional data. Hie-OLAPKNN with the combination of inter- and intra-dimensional imputation performs best since it first carry out inter- and intra-dimensional imputation which can replace most missing values among these three hierarchical imputation. But it performs only a litter better than Hie-OLAPKNN with inter-dimensional imputation (F-score of up to 0.08% and 0.09% higher in $TPCH\_Cus$ and $TPCH\_Supp$). This is because the missing values that can be replaced by intra-dimensional imputation but that are not replaced in inter-dimensional imputation are replaced by OLAPKNN with intra-dimensional data. The little difference is due to the high effectiveness of OLAPKNN for this dataset.

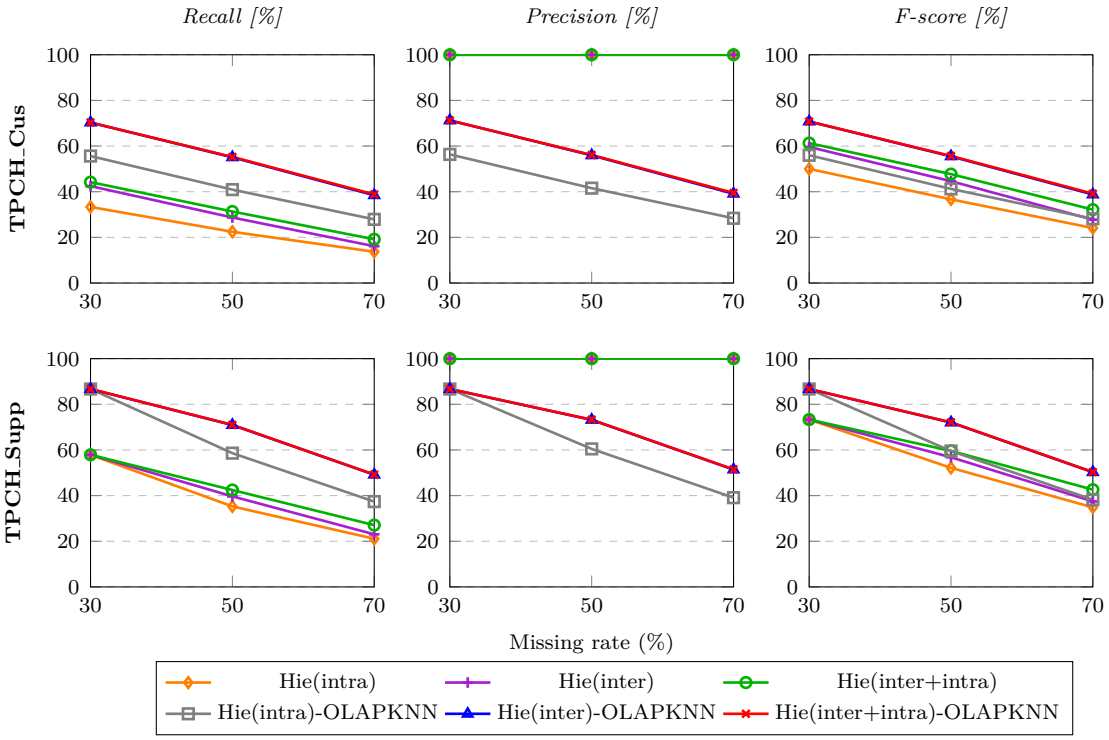


Figure IV.10: Effectiveness results of with second missing data generation strategy

6.3.2 Run time

The run time of the single attribute imputation, multiple attribute imputation and multiple attribute imputation with the second missing data generation strategy are respectively shown in Fig. IV.11, Fig. IV.12 and Fig. IV.13. We can see that Hie-OLAPKNN imputations are slower than hierarchical imputations, which is no doubt since they carry out OLAPKNN after hierarchical imputation.

For the three hierarchical imputations, we can observe that there is nearly no difference in terms of run time in these different tests. But we can still find that in general, the inter-dimensional imputation costs more time (up to 0.43s in $TPCH\_Cus$ and 0.22s in $TPCH\_Supp$) than intra-dimensional imputation because it costs time to search for iden-

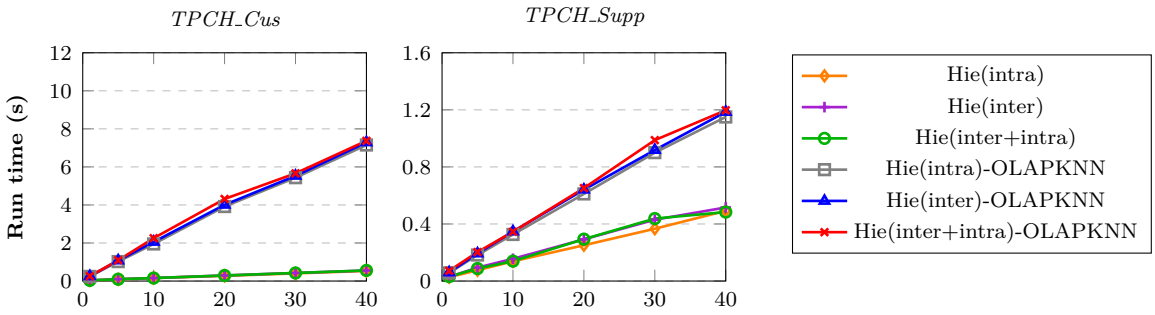


Figure IV.11: Run time results of single attribute imputation of experiment1

tical attributes in other dimensions. Also, the combination of inter- and intra-dimensional imputation costs more time (up to 0.17s in $TPCH\_Cus$ and 0.16s in $TPCH\_Supp$) than inter-dimensional imputation since it launches two types of hierarchical imputation instead of one.

For the three Hie-OLAPKNN imputations, we can see that the results are different in these three tests. For the simple and multiple attribute imputations with the first missing value generation strategy, the run time order of the Hie-OLAPKNN imputations with different hierarchical imputations is the same as the order for the hierarchical imputations since they replace the same number of values and the run time difference only comes from the hierarchical imputations. In the multiple attribute imputation with the second missing value generation strategy, we observe that the Hie-OLAPKNN with intra-dimensional imputation costs most time (up to 7.68s and 9.28s more with respect to inter-dimensional imputation and the combination in $TPCH\_Cus$; 1.23s and 1.71s in $TPCH\_Supp$) because the intra-dimensional hierarchical imputation replace less data so that OLAPKNN takes more time to replace the remaining data. For the same reason, The Hie-OLAPKNN with inter-dimensional imputaion costs more time (up to 3.97s in $TPCH\_Cus$ and 0.49s in $TPCH\_Supp$) than the one with the combination of inter- and intra-dimensional imputation. We can also observe that Hie-OLAPKNN with intra-dimensional runs faster than Hie-OLAPKNN with inter-dimensional in $TPCH\_Cus$ at the missing rate of 10%. It can be explained by the fact that when there are so many missing data, there are even more missing values after the hierarchical imputation with intra-dimensional imputation than other hierarchical imputations. It will be harder to find candidate replaced values and thus costs less time for the search of nearest neighbors.

6.3.3 Strictness

The strictness degrees of these algorithms are always 100% with different experiment strategies since both **Hie** and **OLAPKNN** considers the dependency constraints of hierarchy levels.

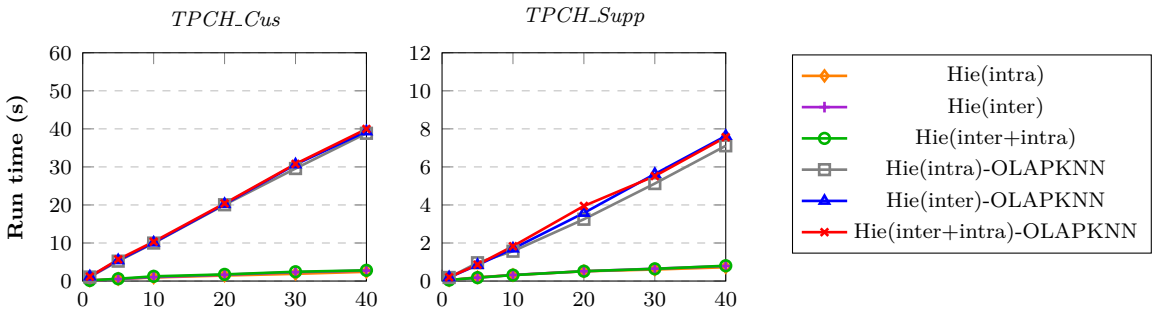


Figure IV.12: Run time results of multiple attribute imputation of experiment1

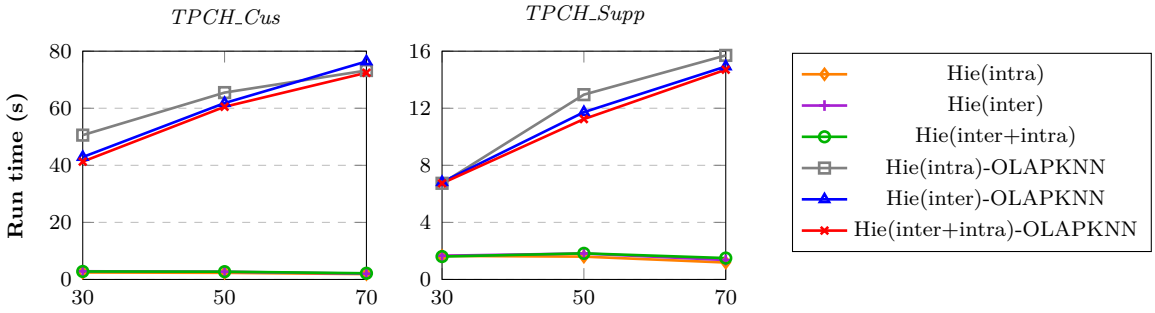


Figure IV.13: Run time results second missing data generation strategy

6.3.4 Results and Analysis for Experiment2

6.3.5 Effectiveness

The effectiveness results for single attribute imputation strategy and multiple attribute imputation are respectively shown in Fig. IV.14 and Fig. IV.15.

In the single attribute imputation, we can see that the effectiveness metrics of the algorithms decrease (e.g. decrease of F-score of 0.94% for **Hie-OLAPKNN** in $F1$) or increase (e.g. increase of F-score of 3.19% for **NB** in *Adventure*) for the missing rates from 1% to 10%. This is because when the missing rate is low, there are not so many missing values. So the denominators of the metrics are small, a small change of the numerator may largely change the results of the metrics, which make the results of the metrics not stable and may decrease or increase. For the missing rate from 10% to 40%, the effectiveness metrics of the algorithms slightly decrease (e.g. F-score has 4.99% and 3.05% decrease for **Hie-OLAPKNN** and **MIBOS** in *Adventure*). They decrease because with the increase of the missing value number, the number of correctly replaced values decreases, which make it harder to find the correct values. The decreases are slight because we have only one attribute where there are missing values and all the other attributes are complete. So if there are correctly replaced values, the algorithms always have nearly the same effectiveness. We can also observe that the algorithms **KNN**, **NB** and **MIBOS** always have the same values on precision, recall and F-score for the same dataset at the same missing rate. This means that they always replace all missing values. They always obtain best F-scores than **Hie** as **Hie** replaces less missing values. However, they always perform

worse (lower F-score) than **OLAPKNN** and **Hie-OLAPKNN** algorithms since they do not consider the DW structure and constraints as we do in **(Hie)-OLAPKNN**.

In the multiple attribute imputation, we can observe that the effectiveness metrics of the algorithms decrease or increase for the missing rates from 1% to 10% due to the same reason as the single attribute imputation. For the missing rate from 10% to 40%, the effectiveness metrics decrease larger (e.g. decrease of F-score of 8.16% and 4.99% in multiple and single attribute imputation for **Hie-OLAPKNN** in *Adventure*; 35.31% and 9.12% for **KNN**) than the single attribute imputation and the metric values are lower (e.g. F-score of 68.13% and 74.19% in multiple and single attribute imputation for **OLAPKNN** in *Organisation* at missing rate of 20%; 42.4% and 64.2% for **NB**). The metrics decrease due to the same reason as the single attribute imputation that there are less correctly replaced values. Moreover, when replacing missing values for one attributes, there are also missing values or wrongly replaced values in the other attributes, which makes the metrics decrease more largely and makes the metrics value lower than the single attribute imputation. Also, the algorithms **KNN**, **NB** and **MIBOS** always replace all missing values and always perform worse than **OLAPKNN** and **Hie-OLAPKNN**.

We then analyse in detail the results for the proposed algorithms **Hie**, **OLAPKNN** and **Hie-OLAPKNN**.

Hierarchical imputation For the hierarchical imputation **Hie**, it always has the worse recall and F-score than the other approaches in single attribute imputation (e.g. F-score of up to 20.37%, 23.72%, 20.17%, 22.28%, 26.78% lower with respect to **KNN**, **NB**, **MIBOS**, **OLAPKNN** and **Hie-OLAPKNN** in *Adventure*). It has the worst recall and F-score in most datasets at most of the missing rates (e.g. F-score of up to 66.35%, 60.02%, 52.02%, 65.35%, 65.85% lower with respect to **KNN**, **NB**, **MIBOS**, **OLAPKNN** and **Hie-OLAPKNN** in *GoSales_Prod*). This is because of the sparsity limit as we introduced in the previous sections. However, in the multiple attribute imputation, it has precision of 100% for datasets **Adventure**, **F1** and **GoSales_Prod** since it is based on the functional dependencies of the hierarchy levels. For dataset **Organisation**, the precision ranges from 98.60% to 100%. By analysing the data, we find that there is a non-strict hierarchy in this dimension because there is a value of the parameter *province* which corresponds to three different values of the parameter *country*. For **Gosales_Ret**, it has precision of 0%. By analysing the schema of this dimension, we find that the two hierarchies of this dimension have only two levels and the highest-granularity levels do not have weak attribute. Thus by employing hierarchical imputation, all missing value parameters have only the dimension identifier as lower-granularity parameter whose values are unique. This is why there is no missing value which can be replaced and the metrics of effectiveness are all 0% for this dataset. In the single attribute imputation, as the weak attributes of the dimension identifier and the parameter of the second level cannot be replaced and we take the average metrics for the experiments with different attribute, we obtain relatively low precision. However, for the attributes whose missing

values are replaced by hierarchical imputation, the precision is always 100% or nearly 100% (for **Organisation** due to the non-strict hierarchy).

OLAPKNN For OLAPKNN imputation (**OLAPKNN**), we can observe that it always has the second best effectiveness metrics in the single (e.g. F-score of up to 1.89%, 4.22%, 4.55% higher with respect to **KNN**, **NB**, **MIBOS** in $F1$) and multiple (e.g. F-score of up to 42.70%, 33.88%, 11.71% higher with respect to **KNN**, **NB**, **MIBOS** in $F1$) attribute imputation experiments. The advantage is more obvious when it comes to the multiple attribute imputation. Few exceptions may happen when the missing rate is between 1% and 10% due to the unstable results of the missing rate of this range as we explained. Other exceptions are the precision results for certain dataset in the multiple attribute imputation since the hierarchical imputation can reach 100% of precision. The better effectiveness of **OLAPKNN** thanks to two reasons. First, we define the specific distance metric which takes into account the dimension structure and characteristics. Second, we create candidate lists by considering hierarchy dependency constraints. **OLAPKNN** replaces all missing values for datasets **Adventure**, **Gosales_Prod** and **Gosales_Ret** in the single and multiple attribute imputation experiments. But it does not replace all missing values for **F1** and **Organisation** because for certain attributes, their neighboring higher-granularity level parameter value is unique or we cannot find the candidate value by the neighboring higher-granularity level parameter.

Hie-OLAPKNN For the hybrid imputation combining hierarchical and OLAPKNN (**Hie-OLAPKNN**), we can observe that it always has the best effectiveness metrics in the single (e.g. F-score of up to 1.89%, 4.22%, 4.55%, 0.38% higher with respect to **KNN**, **NB**, **MIBOS**, **OLAPKNN** in $F1$) and multiple attribute imputation experiments (e.g. F-score of up to 44.84%, 38.01%, 15.84%, 3.51% higher with respect to **KNN**, **NB**, **MIBOS**, **OLAPKNN** in $F1$). Exceptions may happen in the same case and due to the same reasons as **OLAPKNN**. As **Hie-OLAPKNN** combines hierarchical imputation and **OLAPKNN**, it absorbs the advantages of these two approaches. The hierarchical imputation is first launched which have 100% precision for missing values in strict hierarchies. Even if the hierarchical imputation cannot replace missing values for the weak attributes of the dimension identifier and the second level parameter and may replace few missing values in case of high distinction ratio, the remaining missing values can be replaced by **OLAPKNN**. The application of the hierarchical imputation makes **Hie-OLAPKNN** outperforms **OLAPKNN**. As **OLAPKNN** already outperforms other approaches, **Hie-OLAPKNN** thus achieve the best effectiveness with respect to all compared aproaches.

6.3.6 Efficiency

The efficiency results for single attribute imputation strategy and multiple attribute imputation are respectively shown in Fig. IV.17 and Fig. IV.16. We can observe that in the single attribute imputation, the run time increases with the increase of the missing rate for all algorithms. In the multiple attribute imputation, the run time increases with

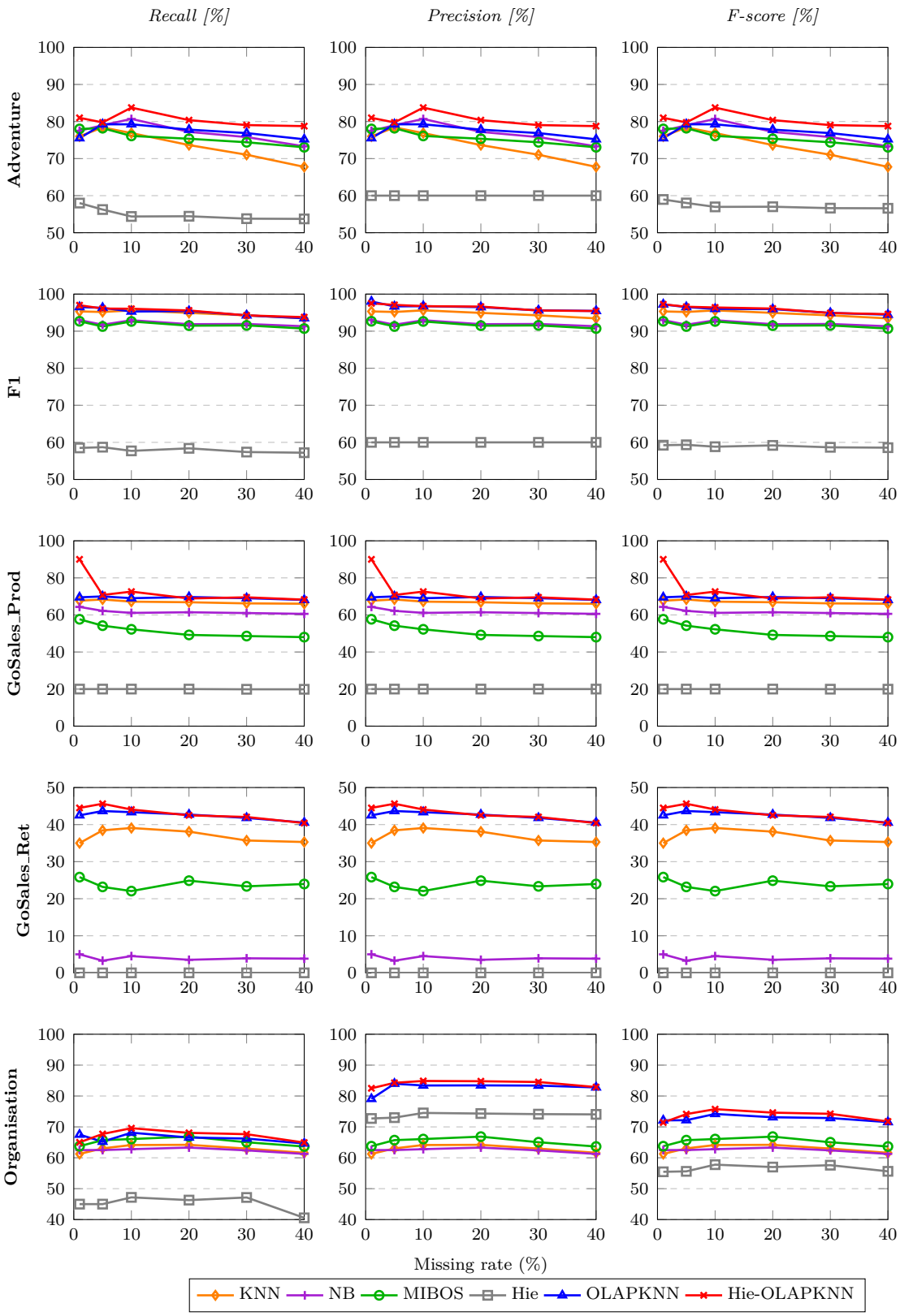


Figure IV.14: Effectiveness results of single attribute imputation

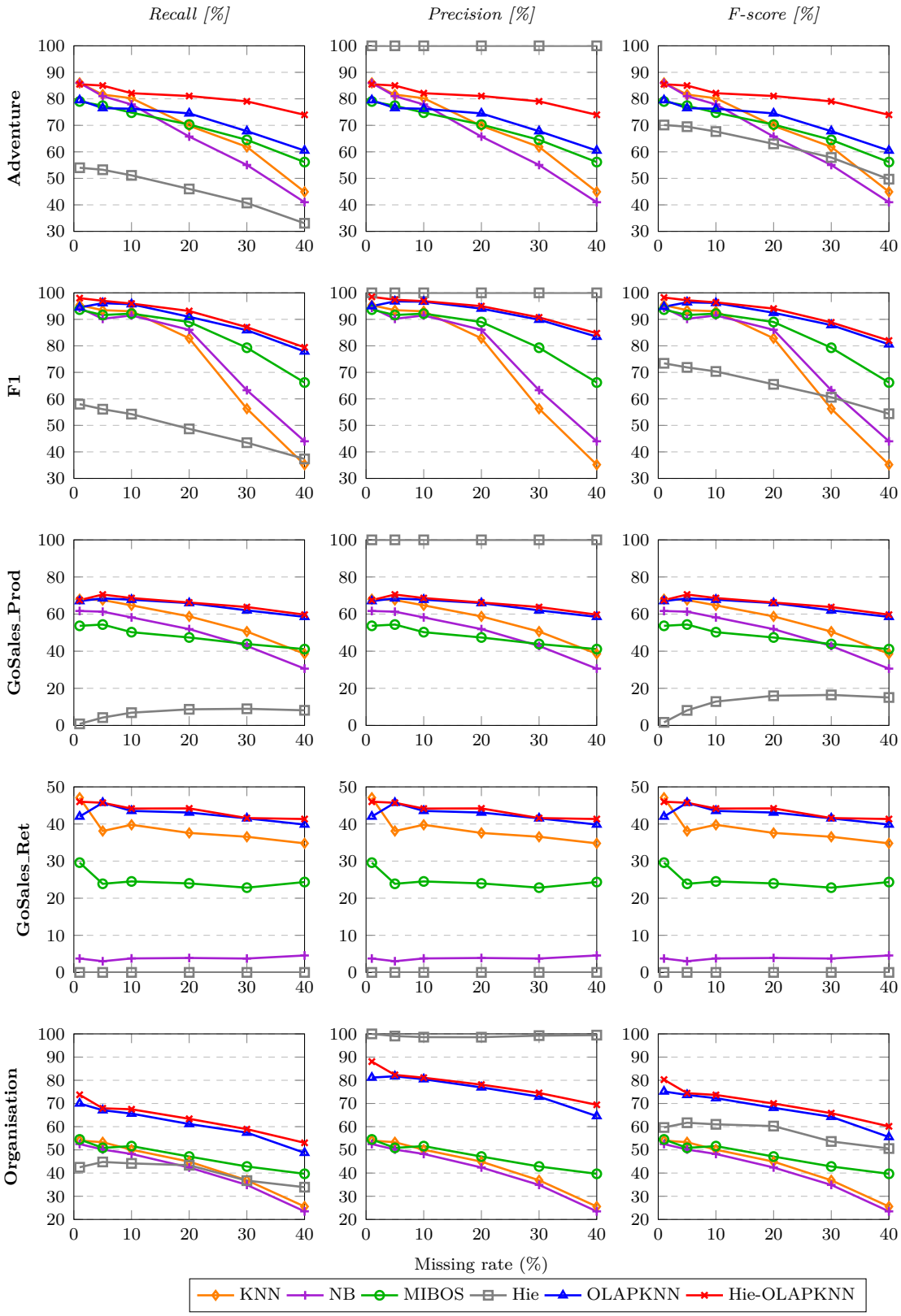


Figure IV.15: Effectiveness results of multiple attribute imputation

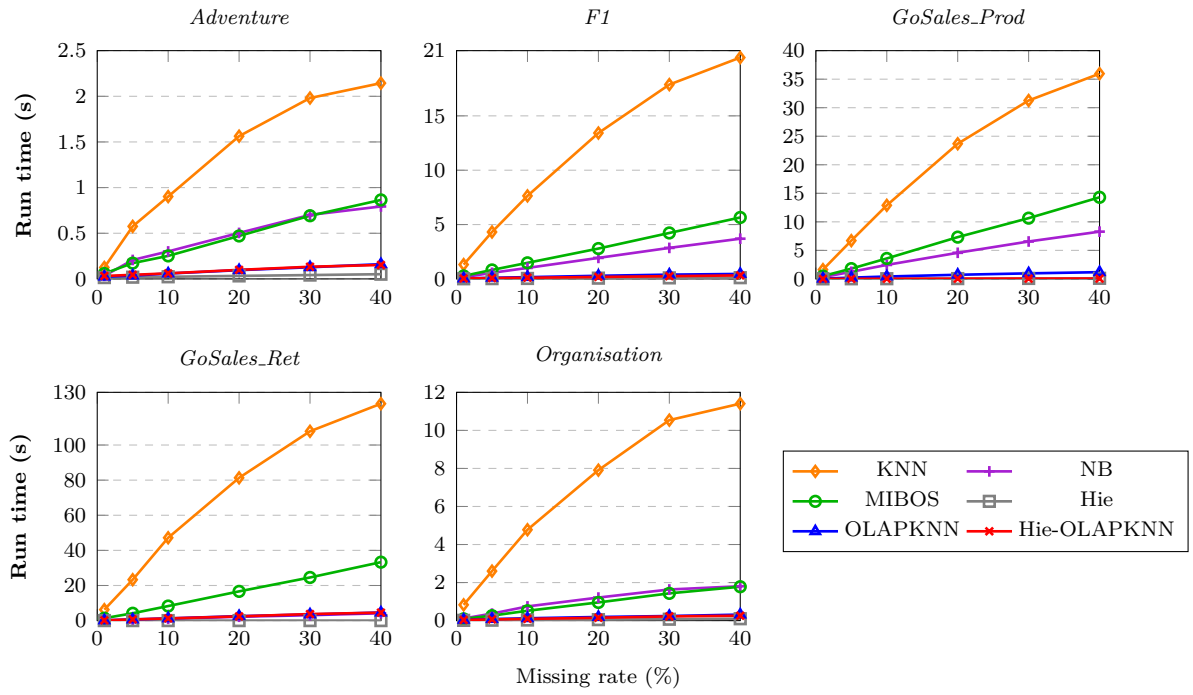


Figure IV.16: Run time results of single attribute imputation

the increase of the missing rate for most of the algorithms. However, for **KNN** and **NB**, the run time first increases and then decreases. This is because when the missing rate is high, there are more instances containing missing values and which cannot be used for imputation in **KNN** and **NB**.

We can also observe that our proposed algorithms **Hie**, **OLAPKNN** and **Hie-OLAPKNN** run faster than other approaches for the applied datasets in both single and multiple attribute imputation. **OLAPKNN** runs faster than **KNN**, **NB** and **MIBOS** (e.g. up to respectively 1.98s, 0.63s, 0.7s faster in *Adventure* of single attribute imputation; 2.51s, 0.51s, 1.39s in *Adventure* of multiple attribute imputation) since it create candidate lists based on the neighboring higher-granularity level parameters and which reduce the searching range of the nearest neighbors. **Hie-OLAPKNN** runs faster than **OLAPKNN** (e.g. up to 0.14s and 1.75s faster in $F1$ of single and multiple attribute imputation) because it applies hierarchical imputation which has the lowest run time before carrying out **OLAPKNN**. **Hie** has the lowest run time among all approaches (e.g. up to 0.21s and 0.93s faster than **Hie-OLAPKNN** in $F1$ of single and multiple attribute imputation). It has the highest efficiency because **Hie** replaces missing values based on functional dependencies between the hierarchy levels and by only searching for instances having the same lower parameter values which can be realised by SQL queries.

6.3.7 Strictness

The strictness results for single attribute imputation strategy and multiple attribute imputation are respectively shown in Fig. IV.18 and Fig. IV.19. Since dataset **GoSales_Prod**

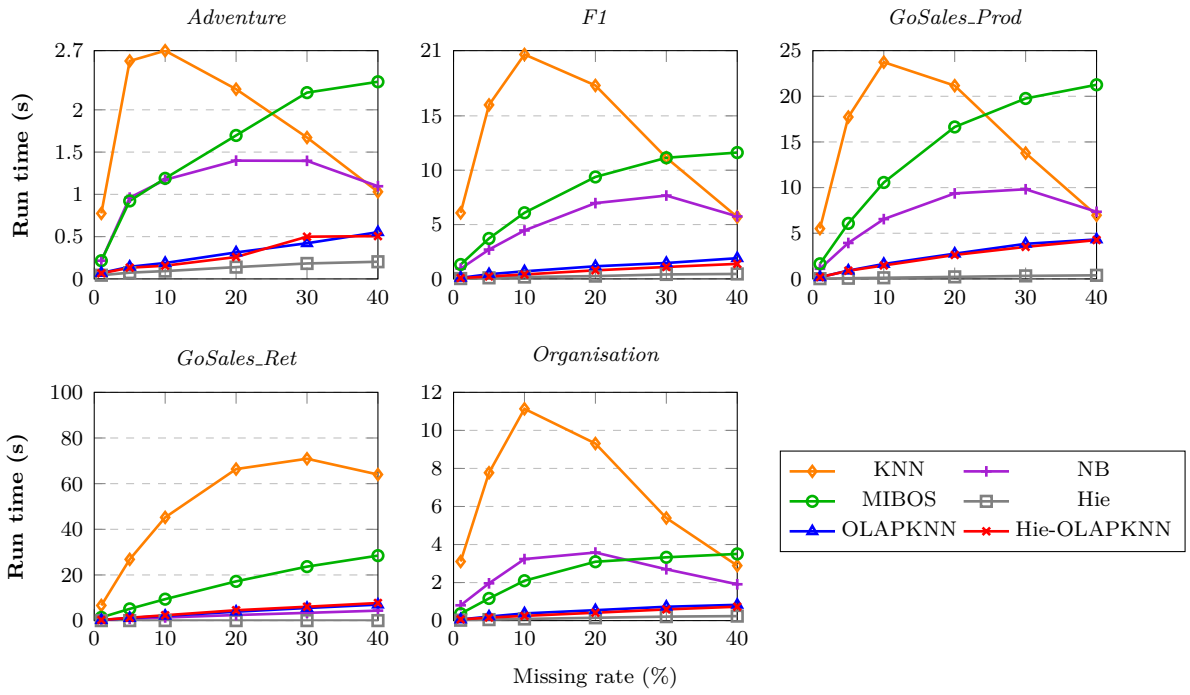


Figure IV.17: Run time results of multiple attribute imputation

has no second level parameter, It is not considered in the strictness experiments.

For the other datasets, we can observe that **KNN**, **NB** and **MIBOS** always have strictness degree less than 100%. Moreover, the strictness degrees of these approaches slightly decrease in the single attribute imputation and largely decrease in the multiple attribute imputation with the increase of missing rate (e.g. strictness degree decrease of 1.66s 27.95s for **KNN** in *Advanture* of single and multiple attribute imputation). The decrease trend of the strictness is similar as the decrease trend of the effectiveness since the wrongly replaced values may make the hierarchies non-strict.

Our **Hie**, **OLAPKNN** and **Hie-OLAPKNN** approaches always have 100% of strictness degree for datasets **Adventure**, **F1** and **GoSales_Prod**. **Hie** is a dependency-based imputation approach, so the replaced values respect the strictness of the hierarchies. Dimensions with replaced values by **OLAPKNN** achieve 100% strictness degree thanks to two reasons. First, it create candidate lists according to the neighboring higher-granularity level parameter, which makes replaced values respect the dependency relationships between the missing value parameters and their higher-granularity level parameters. Second, when there exists lower-granularity non-id parameters of the missing value parameter, we unify the replaced missing parameter values in case of conflicts. This ensures the respect of dependency relationships between the missing value parameters and their lower-granularity level parameters. Since **Hie-OLAPKNN** combines these two approaches, it also replaces missing values by respecting the hierarchy strictness. For dataset **Organisation**, these three approaches do not get 100% strictness degree in both single and multiple attribute imputations due to the non-strict hierarchy in the dimension. Nevertheless, the

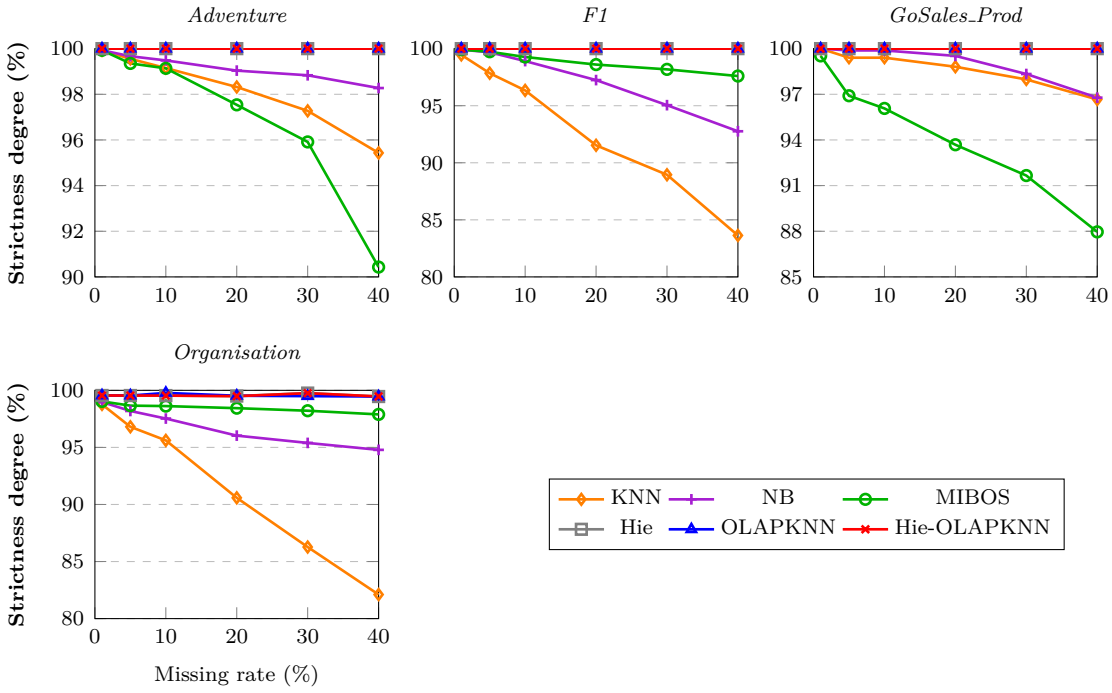


Figure IV.18: Strictness results of single attribute imputation

strictness degree is always more than 99%.

7 Conclusion

In this chapter, we proposed a hybrid approach for dimensional data imputation named Hie-OLAPKNN which combines a rule-based, i.e., hierarchical relationship-based approach and a machine learning-based, i.e., KNN-based approach. To our knowledge, it is the first specific work for the imputation of dimensional data.

Dimensional imputation requires the consideration of the dimension structure complexity and the preservation of hierarchical dependency relationships. Our approach meets the requirements. The hierarchical imputation is based on the existing data found in both intra- and inter-dimensional hierarchical relationships. The OLAPKNN replaces missing values by their nearest neighbors in the candidate list. We proposed a novel distance metric for dimension instances by considering different dimension elements and their relationships. We also proposed the creation of candidate list by taking into account dependency constraints of hierarchies. In our imputation approach, we take in charge the imputation of both parameters and weak attributes. The hierarchical imputation and OLAPKNN imputation are respectively validated through papers in the international conference DEXA2021 (**YANG, Y.** et al., 2021a) and ADBIS2022 (**YANG, Y.** et al., 2022b).

We carry out various experiments which have shown that (1) combining inter- and intra-dimensional imputation in case of identical attributes between different dimensions

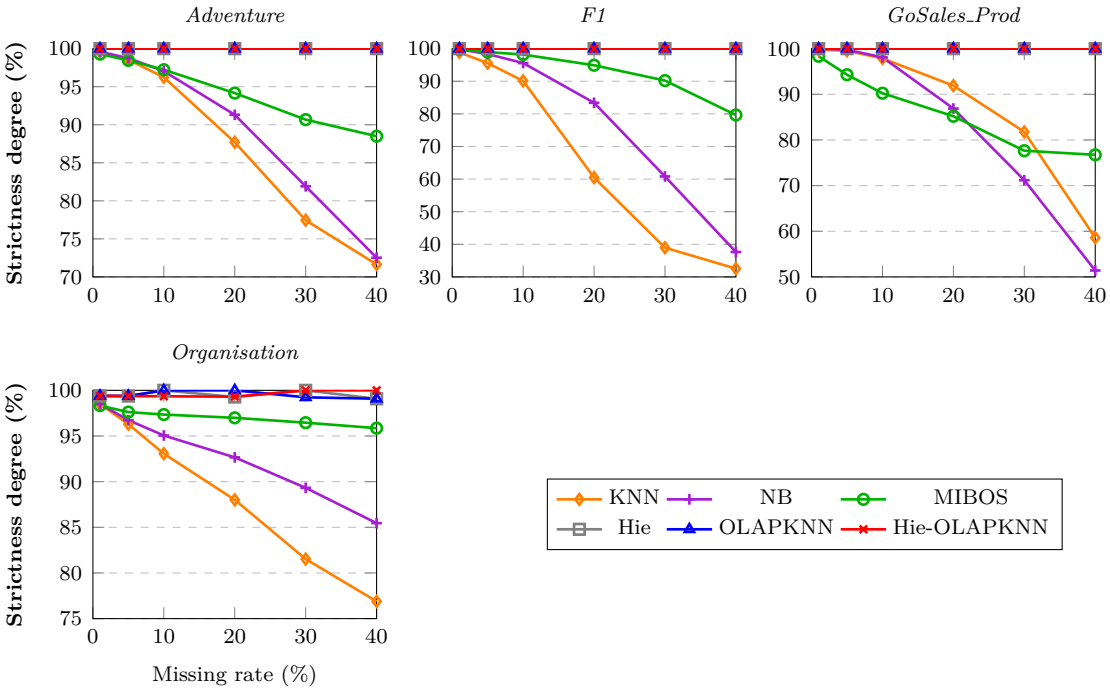


Figure IV.19: Strictness results of multiple attribute imputation

can correctly replace more missing values than applying one of them in hierarchical imputation or Hie-OLAPKNN (e.g. up to 12.59% higher F-score in $TPCH\_Cus$); (2) combining inter- and intra-dimensional imputation in case of identical attributes between different dimensions is faster than applying only one of them in Hie-OLAPKNN (e.g. up to 9.28s less run time in $TPCH\_Cus$); (3) Hie-OLAPKNN can correctly replace more missing values than applying only hierarchical imputation, OLAPKNN or other approaches from the literature (e.g. up to 44.84% higher F-score in $F1$); (4) Hie-OLAPKNN runs faster than applying only OLAPKNN or other approaches (e.g. up to 2.51s less run time in *Adventure*); (5) dimensions with missing values replaced by Hie-OLAPKNN, OLAPKNN and hierarchical imputation respect the hierarchy strictness while the other approaches do not able to respect such strictness.

Chapter V

Implementation

Contents

1 Introduction

In the previous chapters, we proposed a solution to automate the DW design from tabular data, merge different DWs in case of multiple sources and replace missing values during the merging. The different parts of the solution have been validated through various experiments. To put our solution into practice, all these parts should be integrated into one application to offer a complete solution for users. Since the target users of our solution are non-expert users, our application should be user-friendly especially for non-expert users. So the application should use non-technical vocabularies instead of technical ones so that non-expert users can understand DW information and the process. Expert users are not targeted, but they may also get involved in the process to make deeper validation and customisation, we should also consider the requirements of such users. As a result, we implement an application which (1) combines our three proposed functionalities, (2) has a user-friendly interface and (3) provides different versions for non-expert and expert users. We first present the functional and technical architectures of the application.

1.1 Functional Architecture

The functional architecture of our application is shown in Fig V.1. It has three functionalities corresponding to the three parts of our proposal.

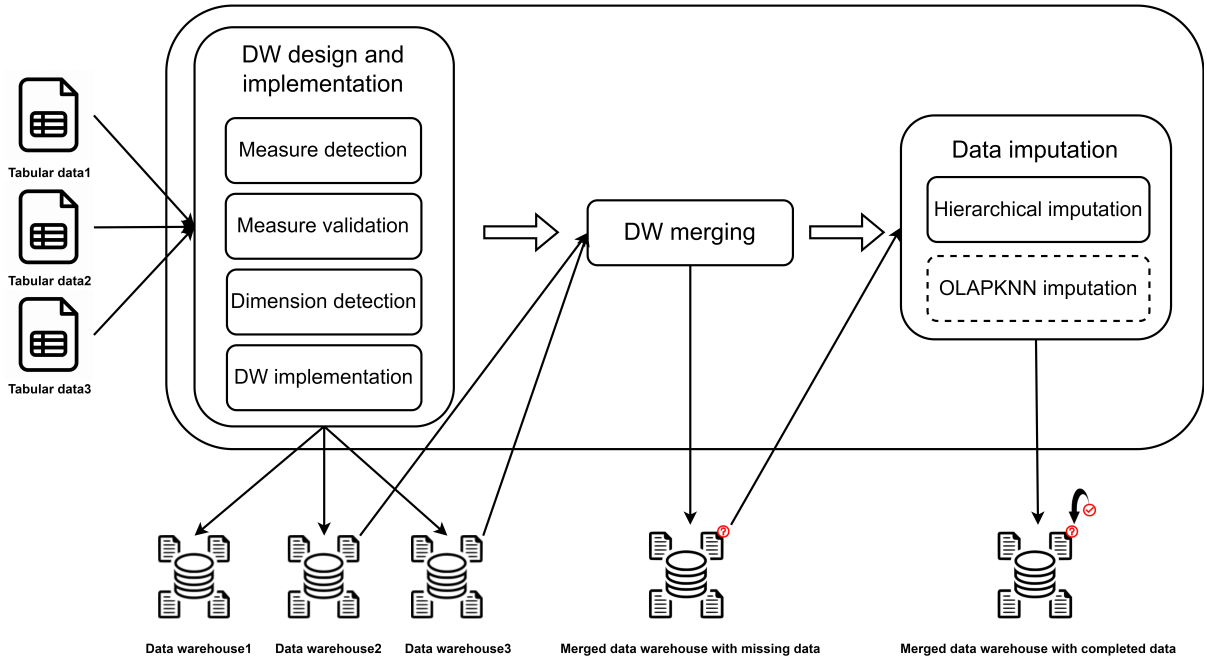


Figure V.1: Technical architecture

- The first functionality is the automatic DW design and implementation. The input tabular data are processed by our proposed automatic DW design approach. First, the measure detection is carried out. Second, the detected measures are proposed to users and the application asks the user to validate the measures. Third, the

dimensions are detected. Finally, the DW is implemented into the database by following the detected schema.

- The second functionality is the automatic DW merging. The user can choose the DWs to be merged. The DW merging is then carried out to generate a merged DW at the schema and instance levels.
- The third functionality is the dimensional data imputation. The merged DW generated by the second functionality may contain missing values, this functionality aims to replace dimensional missing data by our algorithms. Our algorithm Hie-OLAPKNN combines hierarchical imputation and OLAPKNN imputation. The hierarchical imputation replaces missing data with exact values, while OLAPKNN replace missing data with estimated values. Therefore, hierarchical imputation can always be employed, but OLAPKNN should be used according to the user's tolerance of estimated values that may be inexact. We thus give alternatives to the user of applying Hie-OLAPKNN, or only hierarchical imputation. The merged DW with replaced data is then generated.

1.2 Technical Architecture

The technical architecture of our application is shown in Fig. V.2.

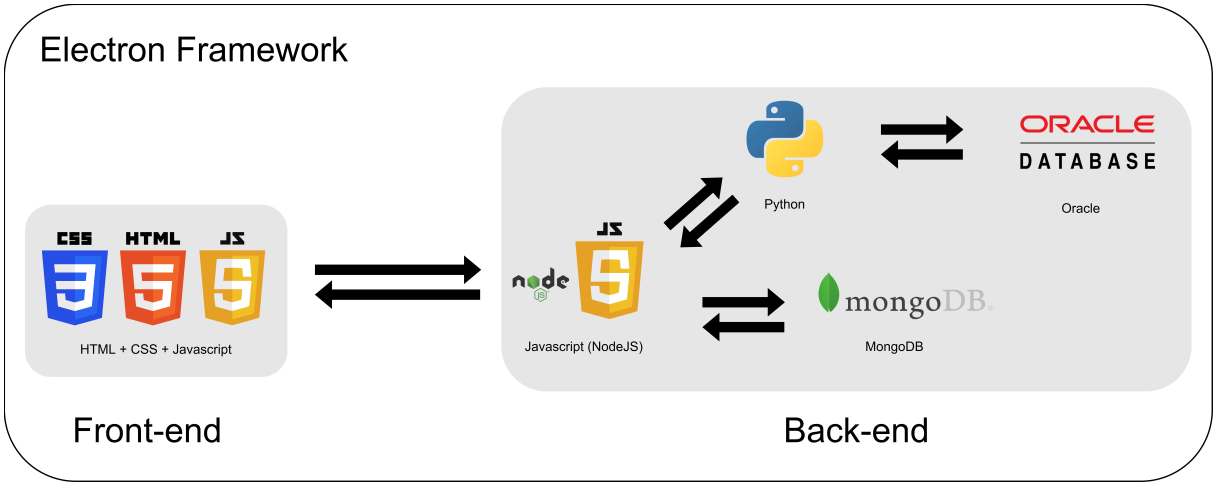


Figure V.2: Technical architecture

1.2.1 Development Framework

We apply the framework Eletron [1] to develop our application. Eletron is a framework to build cross-platform desktop applications using JavaScript, HTML and CSS. It combines the Chromium browser as rendering engine and the Node.js runtime as back-end environment. We apply Eletron because it has the following advantages (Peguero and Cheng, 2021).

[1] https://www.electronjs.org/

- It is an open-source framework maintained by Github and has a large active community of contributors.
- The combination of Chromium and Node.js allows developers to facilitate the user interface creation by using web technologies such as JavaScript, HTML and CSS.
- It allows the development of one application version which is compatible with various operating systems such as Mac, Windows, and Linux.

1.2.2 Front-end and Back-end

Regarding the front-end, we create the user interface by using HTML, CSS and JavaScript, which are typical front-end web development languages.

Regarding the back-end, the development is carried out in the Node.js environment as we use the Electron framework. Node.js is a JavaScript runtime built on Chrome's V8 JS engine. Our research is a part of the BI4people project, other parts of the automatic BI solution are implemented by other groups with different languages such as Java, Python. The advantage of using Node.js is that it provides modules to easily run code of various programming languages. The algorithms of our solution are implemented by Python since it has rich libraries which are helpful for implementing machine learning algorithms (e.g. *sklearn*[2], *pycaret*[3]) and different distance metrics (e.g. *textdistance*[4],*gensim*[5]). The Python code is run in Node.js by the module *child_process*[6]. We use Oracle as the database for the implementation of the DWs. In Python, the library *cx_Oracle*[7] is used to connect to Oracle database for the implementation of DWs and the extraction of DW data. The MongoDB database is used to store DW schema data. We choose MongoDB because it is a document database, which offers various data types such as embedded document, making it convenient to store the complex structure schema of DWs. Moreover, data in MongoDB are stored in BSON (Binary JSON) format, and data represented in JSON can be natively stored in MongoDB. We can thus present scheme data in JSON to users allowing them to easily modify them and update them in MongoDB.

1.3 Outline

The remainder of this chapter is organized as follows. In Section 2, we present the functionality of automatic DW design and implementation. In Section 3, we introduce the functionality of DW merging. In Section 4, we illustrate the functionality of dimensional data imputation. In Section 5, we conclude this chapter.

[2]`https://scikit-learn.org/stable/`
[3]`https://pycaret.org/`
[4]`https://pypi.org/project/textdistance/`
[5]`https://radimrehurek.com/gensim/`
[6]`https://nodejs.org/api/child_process.html`
[7]`https://oracle.github.io/python-cx_Oracle/`

2 Automatic DW Design and Implementation

We illustrate our application with TPCH benchmark data by generating two CSV files containing different attributes and whose DW schemas are the same as those in Section 7.4.

2.1 Front-end

As shown in Fig. V.3, the front-end contains a user interface where a user can select one or several tabular files from the file system by clicking the button "Choose files". In the left of the interface, there are our proposed three functionalities, the user can choose to carry out one of them independently. In our application, we provide two versions including a non-expert version and an expert version that the user can choose in the left. The non-expert version is the default version which shows DW information to the user with non-technical vocabularies and supplementary explications to help non-expert users understand DW information. The expert version use technical vocabularies to describe DW information and allows users to carry out more customised operations.

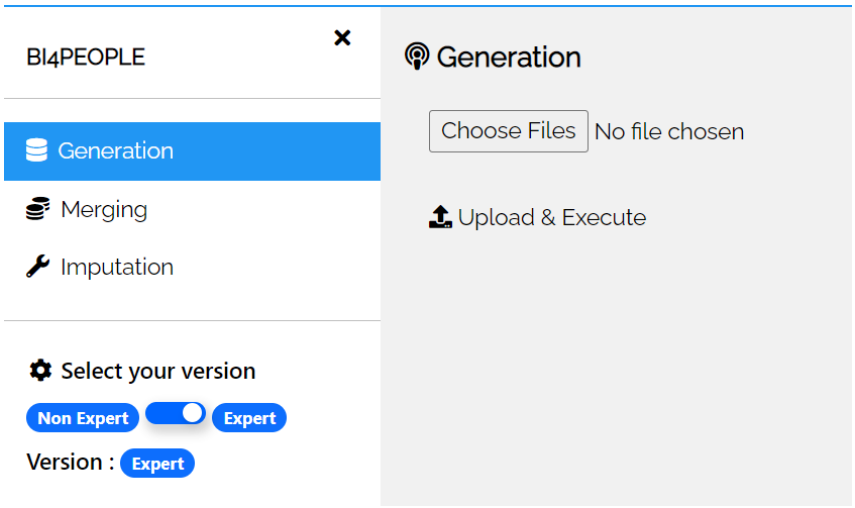


Figure V.3: Upload files

As shown in Fig. V.4, when the user choose the files and click "Upload & Execute", the measure detection will run for each file. The name of each file will be shown in the interface and we can continue other steps for each file independently by clicking on the file name.

The measure detection results of non-expert and expert versions are respectively shown in Fig. V.5 and Fig. V.6. In the non-expert version, the vocabulary "indocator" is used instead of "measure". In the interface, we show the user the proposed measures that are detected by the machine learning algorithm. We also show the other numerical columns by asking the user if they can also be measures. The user can validate the measures by checking the check boxes. Then by clicking "Next", the dimension detection will be carried out.

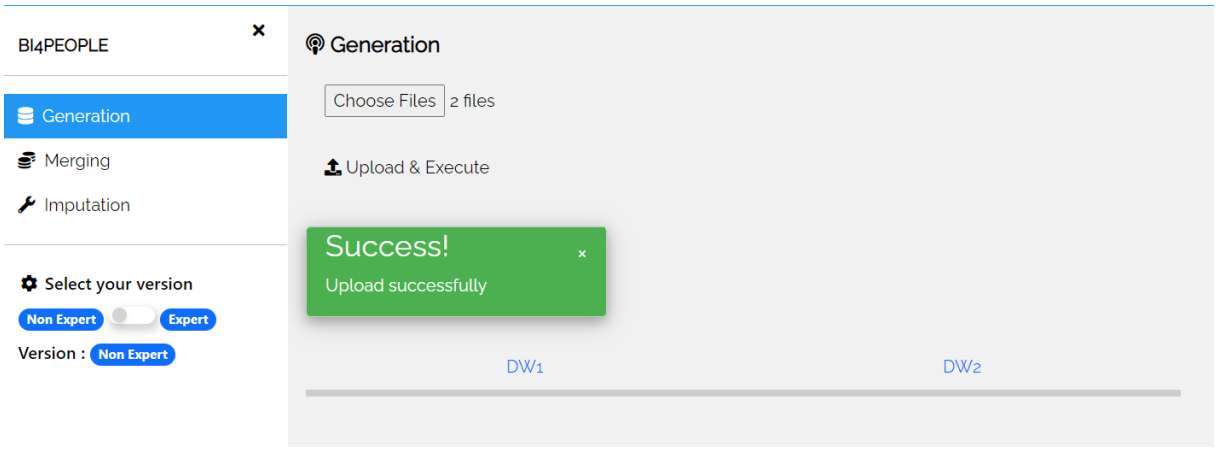


Figure V.4: Files uploaded successfully

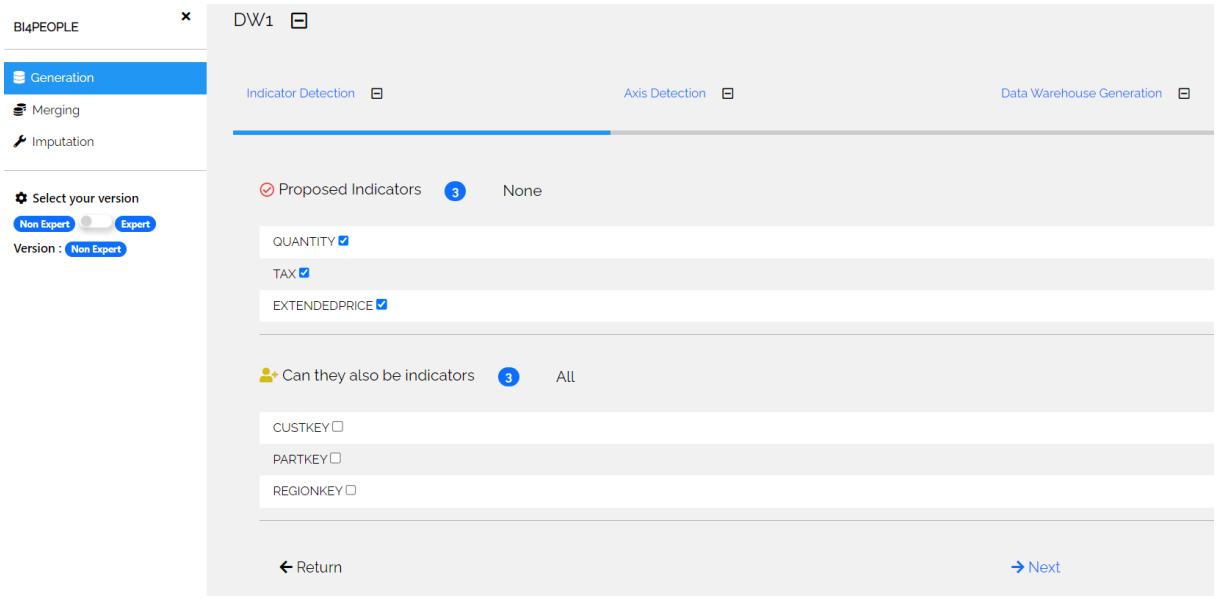


Figure V.5: Measure detection in non-expert version

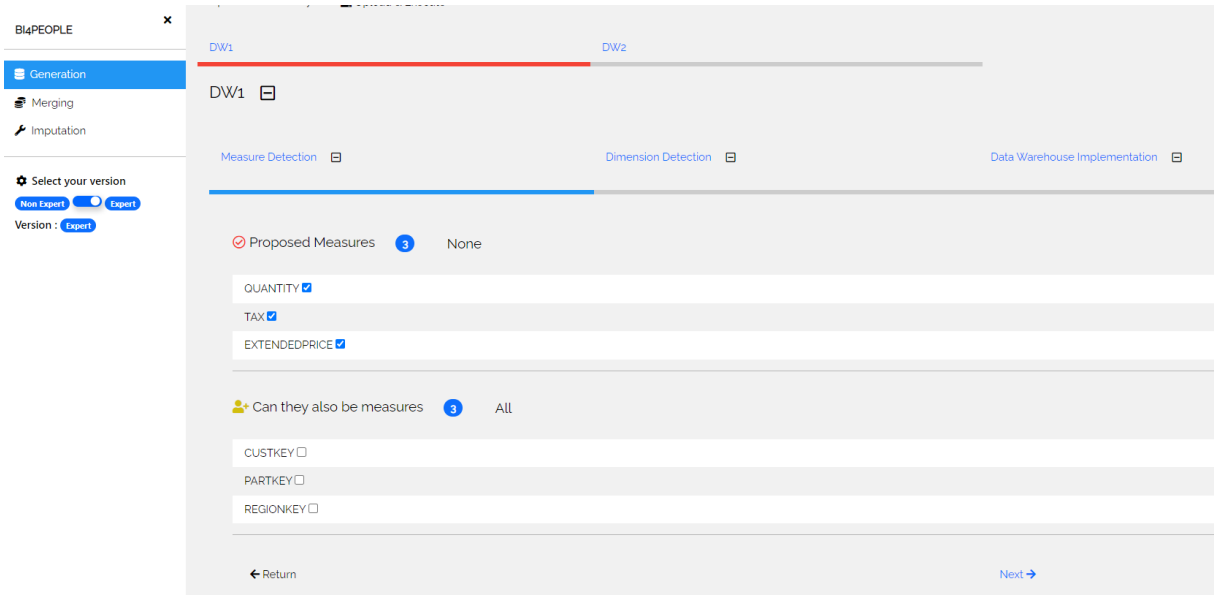


Figure V.6: Measure detection in expert version

The dimension detection results of non-expert and expert versions are respectively shown in Fig. V.7 and Fig. V.9. In the non-expert version, the vocabularies "Axis", "Analysis vision", "Levels", "Supplementary information" are used to respectively describe dimensions, hierarchies, parameters and weak attributes. When the user move the mouse onto the information icons, some explications of the multidimensional components are shown to the non-expert user. Since the hierarchies contain much information, the detail information about the parameters and weak attributes can be consulted or hidden by expand bars. The names of the dimensions, hierarchies and facts do not exist in the original data, they are generated automatically. For example, the dimensions have the name of "D1","D2", "D3". The application allows the user to modify the names manually.

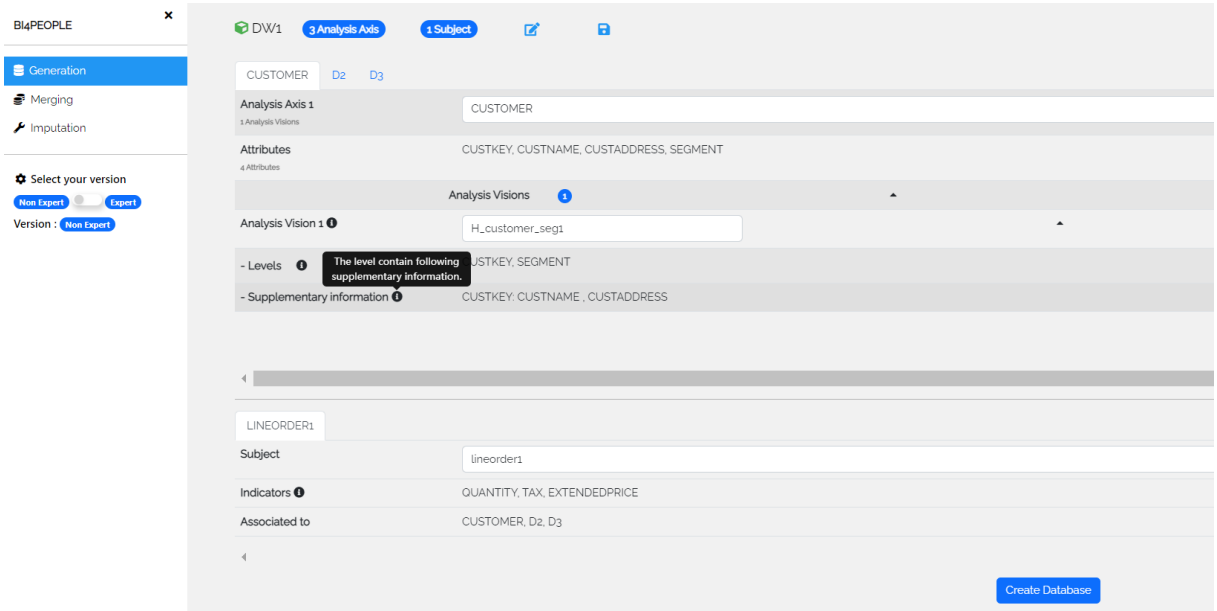


Figure V.7: Dimension detection in non-expert version

If there is a date dimension, the application also proposes some possible date granularity to choose like shown in Fig. V.8. The pre-defined hierarchies are created based on the chosen granularities. This functionality is also provided in the expert version.

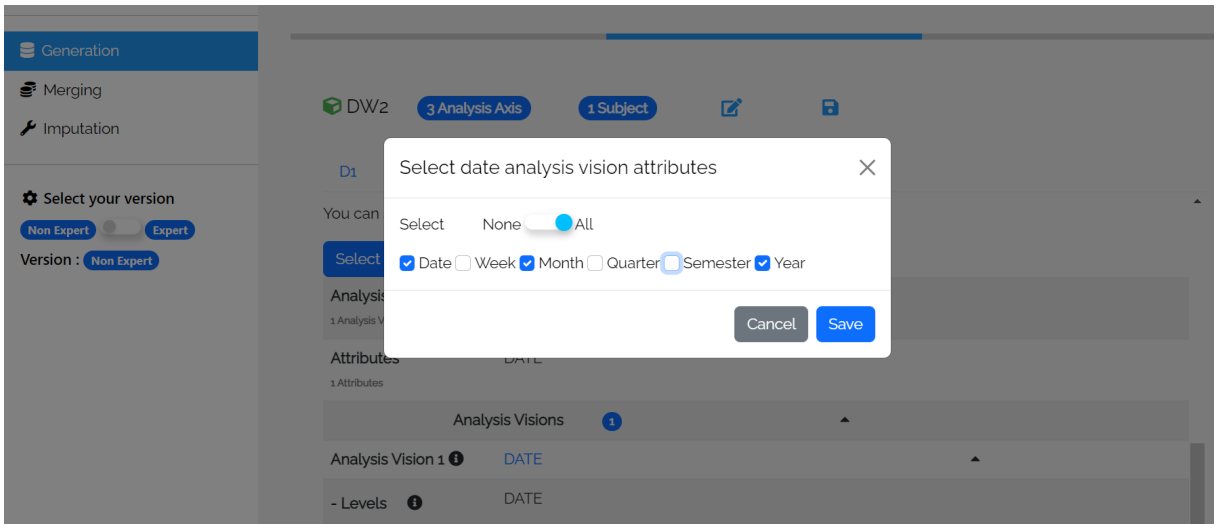


Figure V.8: Date granularity selection

In the expert version (Fig. V.9), there is an additional functionality of schema editing. By clicking the button “Edit schema”, the expert user can edit the schema, which is in a JSON-like format like shown in Fig. V.10

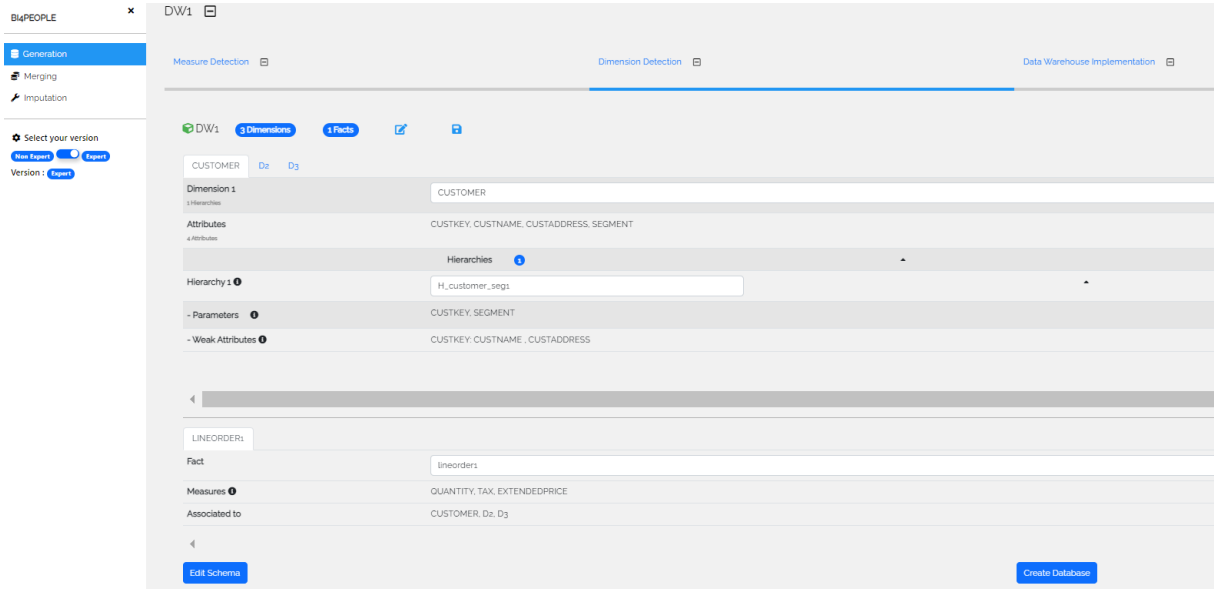


Figure V.9: Dimension detection in expert version

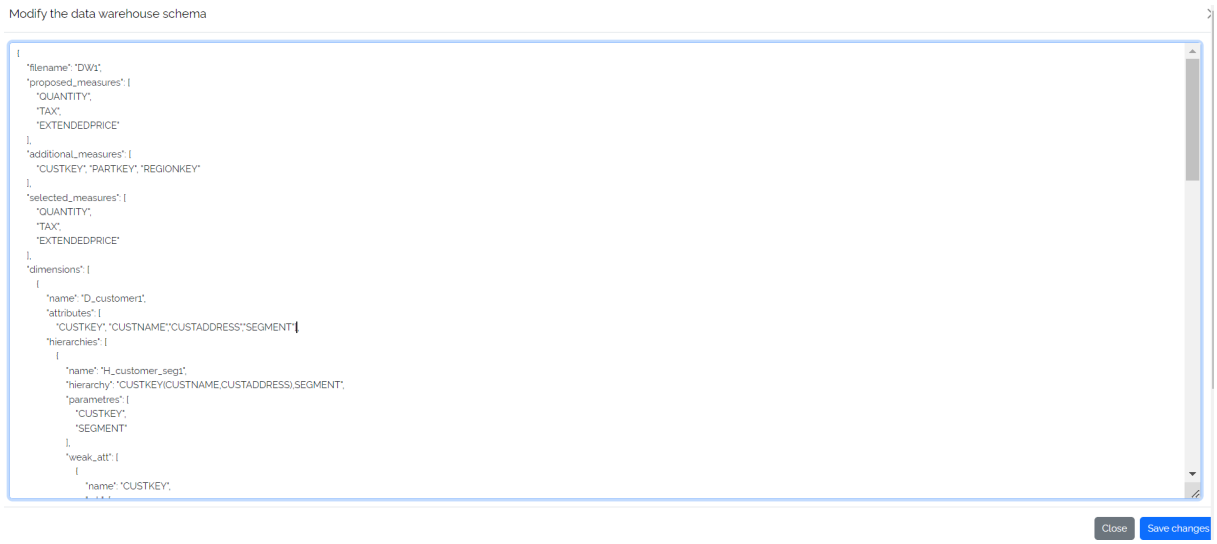


Figure V.10: Schema editing

Finally, the user can click “create database” to implement the DW. A window is displayed which asks the user to enter the database name and password. A confirmation interface is then shown as in Fig. V.11

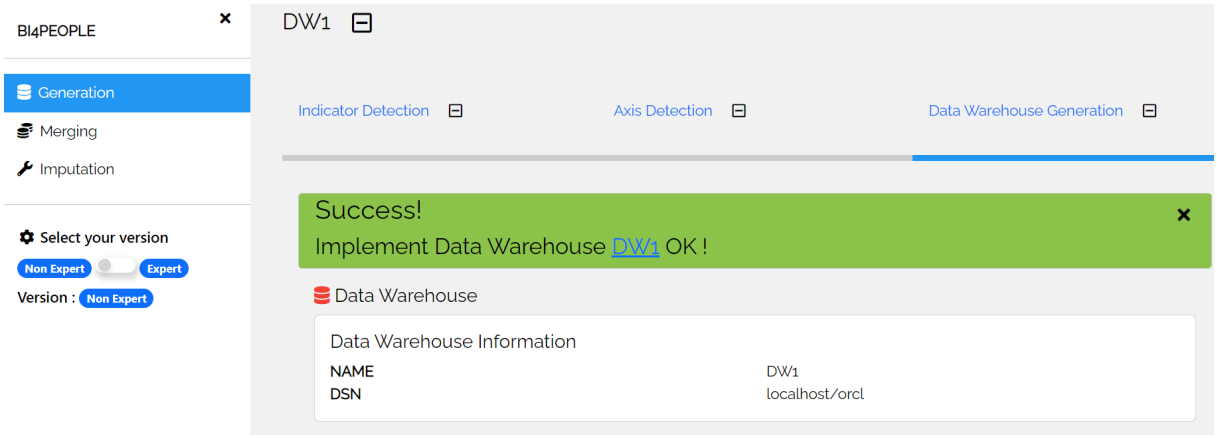


Figure V.11: DW implementation

2.2 Back-end

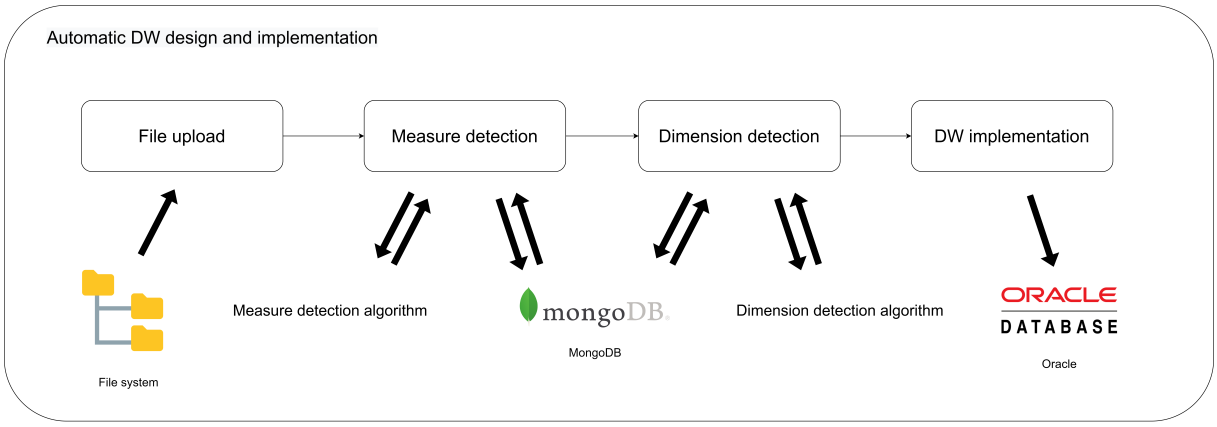


Figure V.12: Back-end illustration of automatic DW design and implementation

We illustrate the back-end in Fig. V.12. When the user selects the files, the file information is extracted from the file system and is first processed by the measure detection algorithm (Algo. 3 $line_1$). The detected measures and other numerical column information are sent back to the front-end. When the user validate the measures, the final measure information is returned to the application and is stored in MongoDB. Then when the user continue the next step, the dimension detection (Algo. 3 $line_{2-31}$) is carried out to detect the hierarchies and identify attributes as parameters or weak attributes. Then the multidimensional schema is generated and is stored in MongoDB. If the user finish modifying the schema, the schema information in MongoDB will be updated. When the user enter the database name and password, a new database will be created in Oracle to implement the detected DW by creating a fact table, dimension tables as well as table relationships, and key constraints.

3 Automatic DW Merging

3.1 Front-end

The front-end contains a user interface where all existing DWs are shown (Fig. V.13).

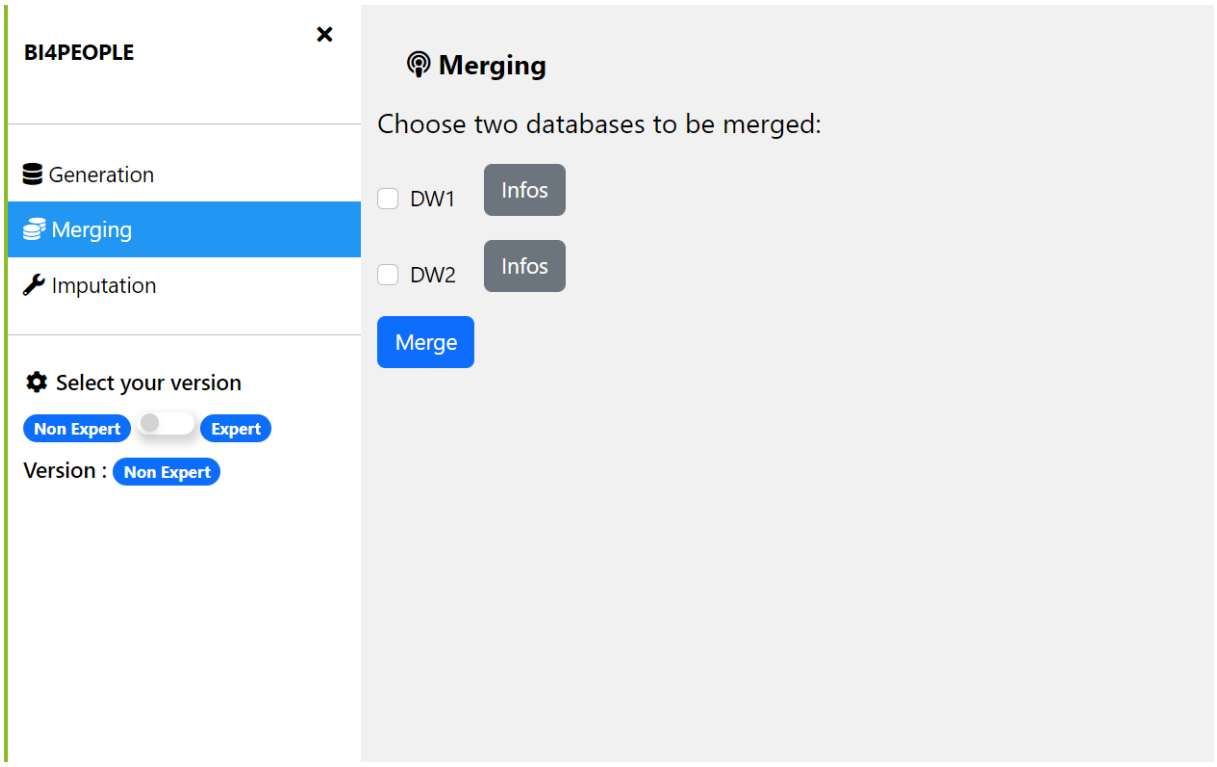


Figure V.13: DW selection

The user can consult the DW schema information by clicking the button “info” as shown in Fig. V.14.

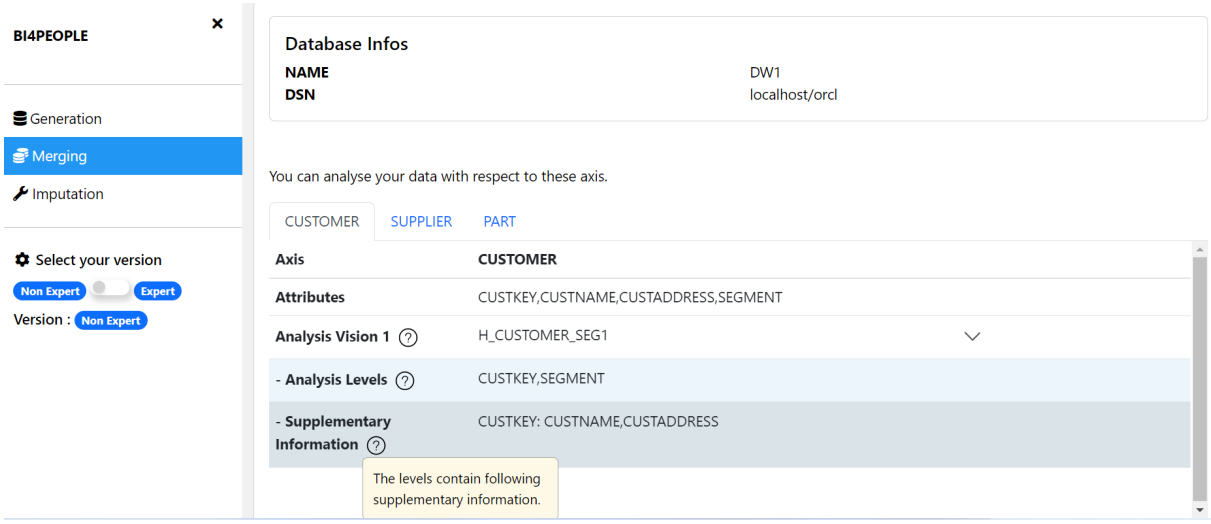


Figure V.14: DW schema information

The user can select the DWs to be merged and click the button “merge”. Then a window of confirmation is shown (Fig. V.15) where there are the DWs’ name. The user can enter the name and the password of the database for the merged DW. By clicking “Yes”, the DWs will be merged and the merged DW will be implemented.

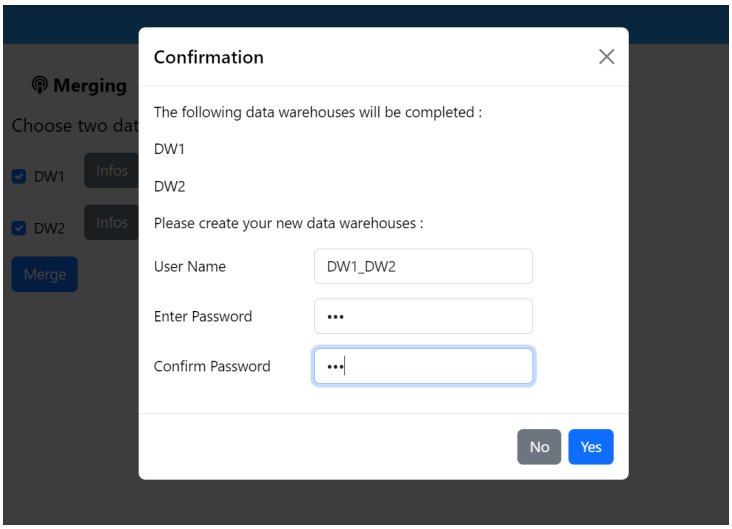


Figure V.15: Confirmation window

The merged DW schema is then shown in the interface as shown in Fig. V.16 and Fig. V.17. We can see that a constellation schema is created where there are two facts (Fig. V.17). The user can also modify the name of the multidimensional components.

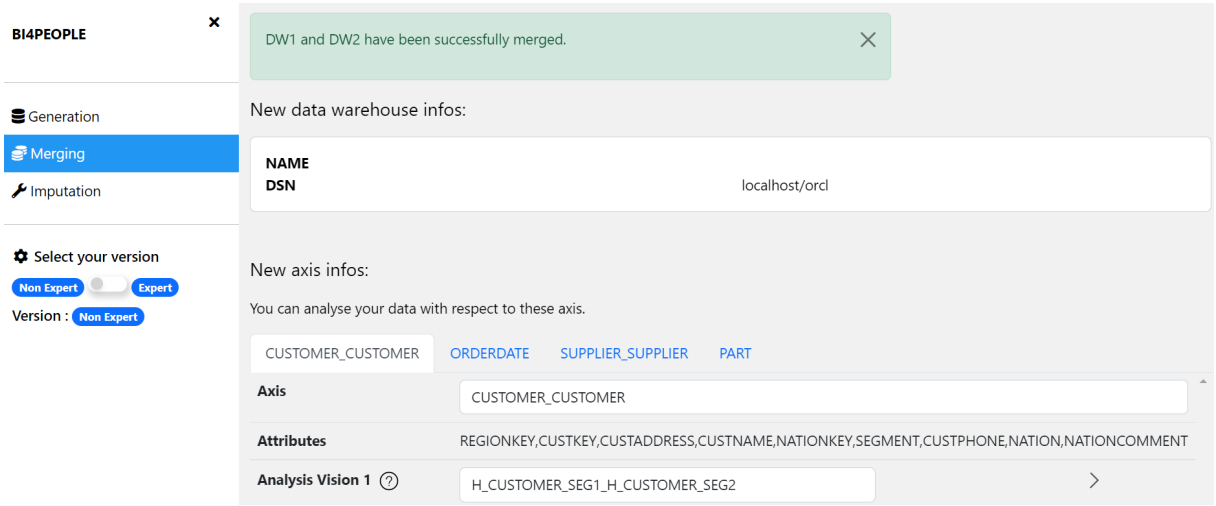


Figure V.16: Merged DW

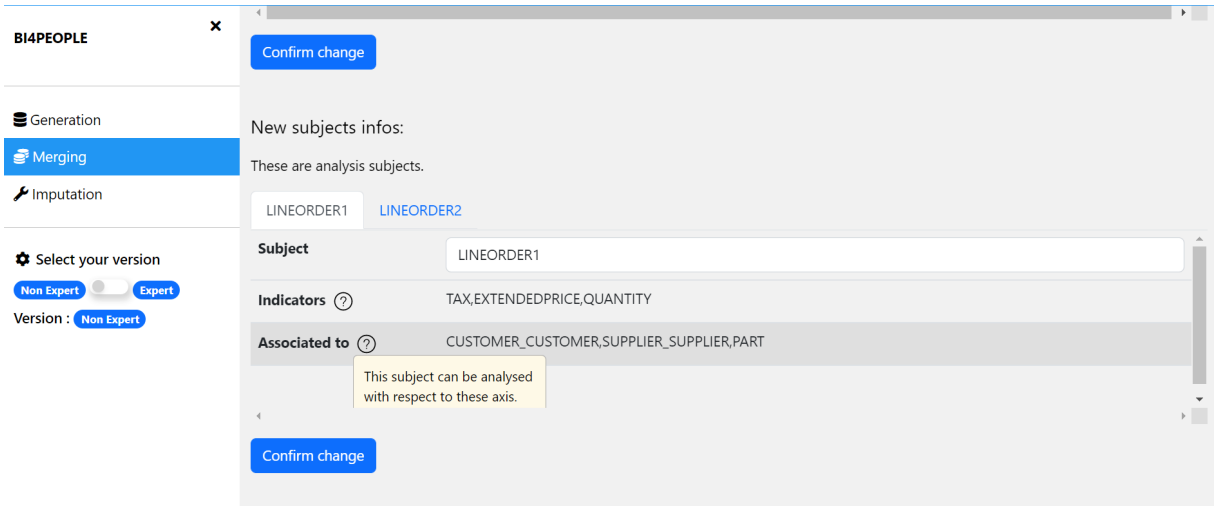


Figure V.17: Merged DW

In the expert version, there is an important difference with respect to the non-expert one that the user can choose the analysis form as shown in Fig. V.18

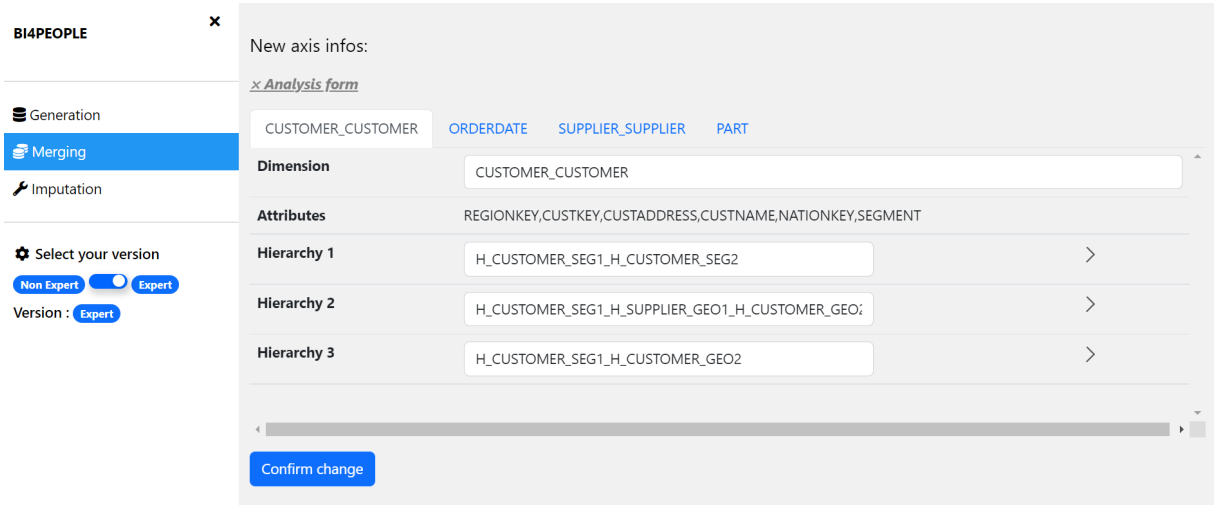


Figure V.18: Analysis form

3.2 Back-end

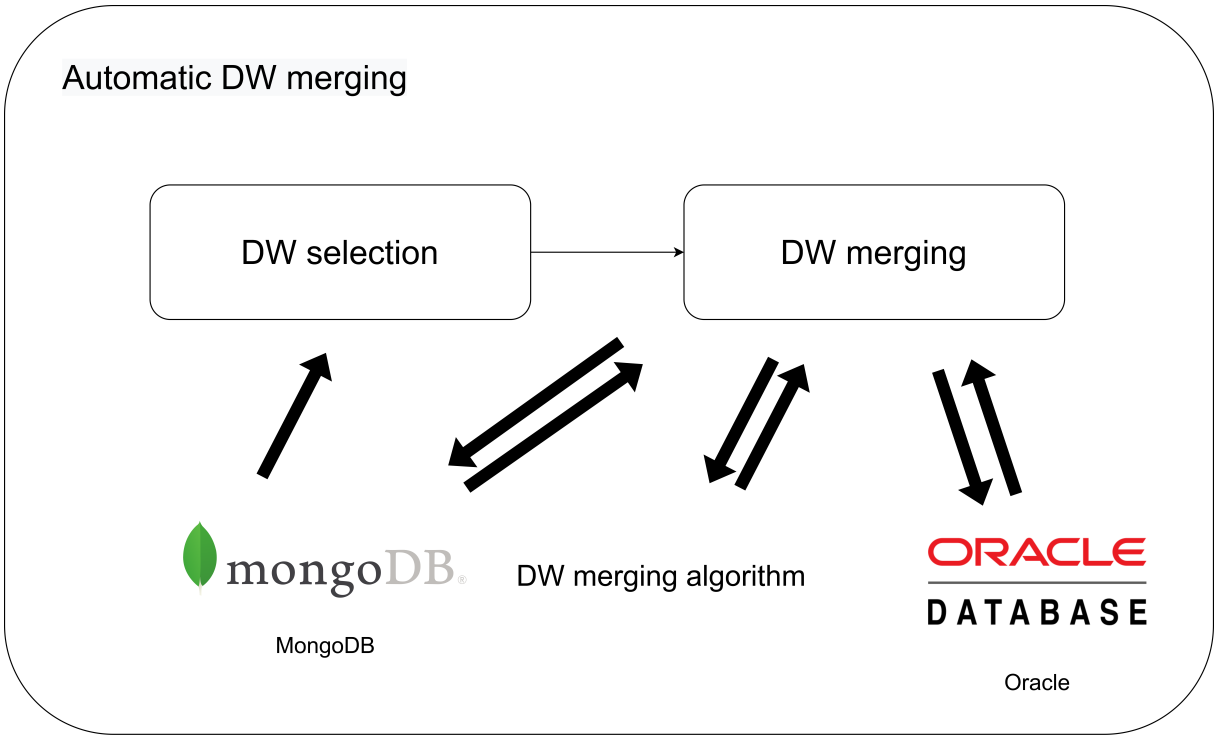


Figure V.19: Back-end illustration of automatic DW merging

We illustrate the back-end in Fig. V.19. The schema of the DWs are extracted from MongoDB and are shown to the user. When the user selects the DWs to be merged and confirm the selection, the DW information will be sent to the DW merging algorithm (Algo. 8) will be carried out. The schema are obtained from MongoDB and the instances are obtained from Oracle. The name and password of the merged DW that the user enters are also collected. At the schema level, a multidimensional schema of the merged DW is created and is stored in the MongoDB database and its information is sent back to the front-end. At the instance level, a new database is created based on the entered merged DW name and password in Oracle. The merged DW instances are obtained and are inserted into the new database.

4 Dimensional Data Imputation

4.1 Front-end

The front-end contains a user interface which allows the user to choose the DW for the imputation as shown in Fig. V.20

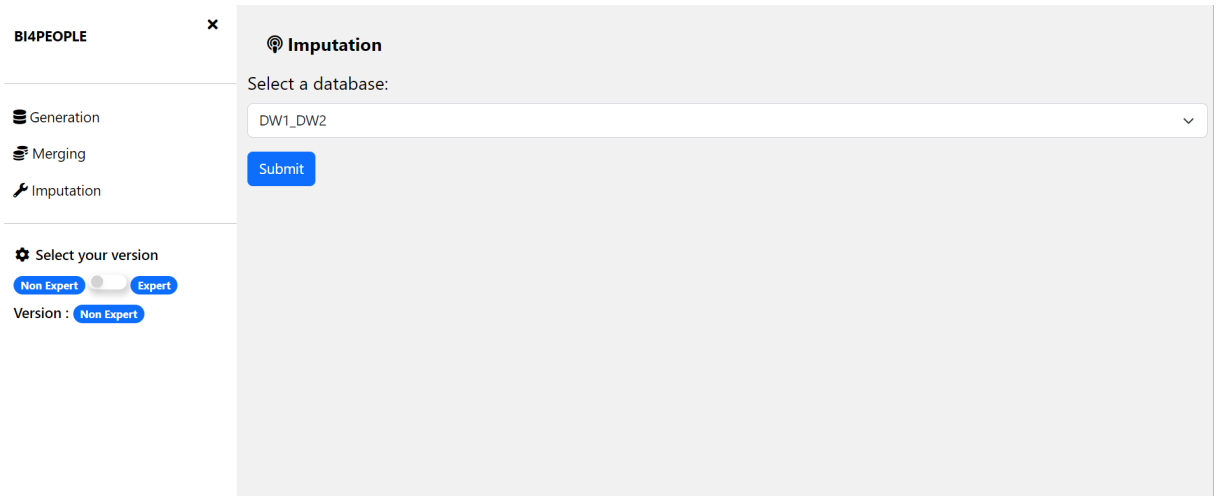


Figure V.20: DW selection

Once the user select the DW, the DW schema information is shown in the interface (Fig. V.21). The missing value number of each attribute as well as the total record name of the dimension are also shown in the interface. The user can then choose the attributes to be replaced.

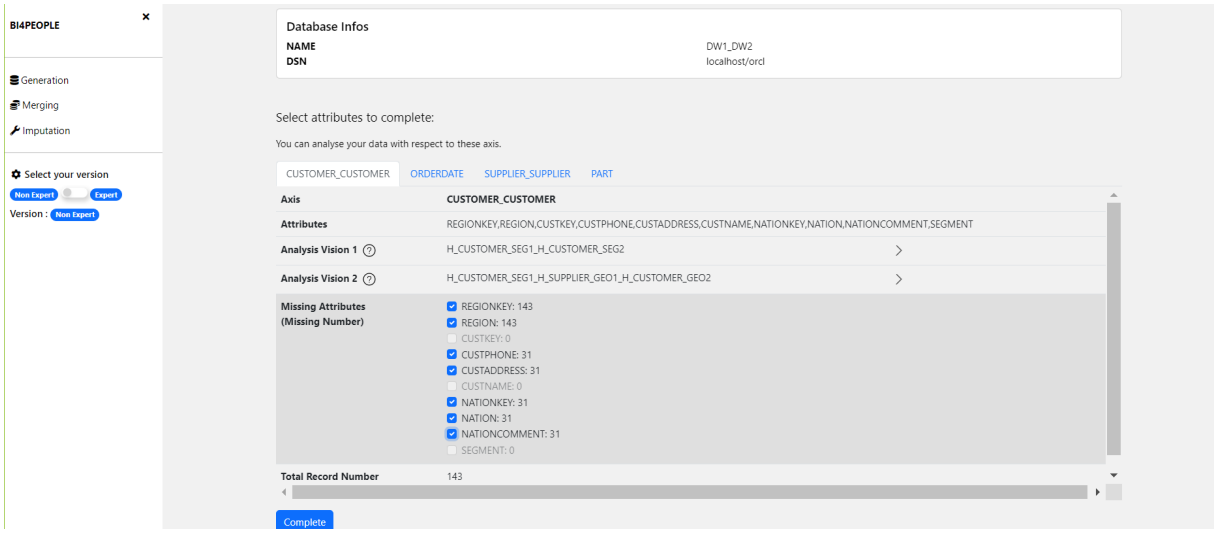


Figure V.21: Attribute selection

Next, the user will see a confirmation window as shown in Fig. V.22. We provide the user with the choice of applying only hierarchical imputation which replaces missing data with exact values or applying Hie-OLAPKNN which replaces as many missing values as possible. In the non-expert version, these two imputation approaches are called “Complete with exact values” and “Complete as many values as possible” so that the non-expert user can understand the advantages of these two approaches. If the non-expert choose using Hie-OLAPKNN, the parameters k and hierarchy level weight are set with default values 2 and w_c since they are the most used optimal values in our experiments.

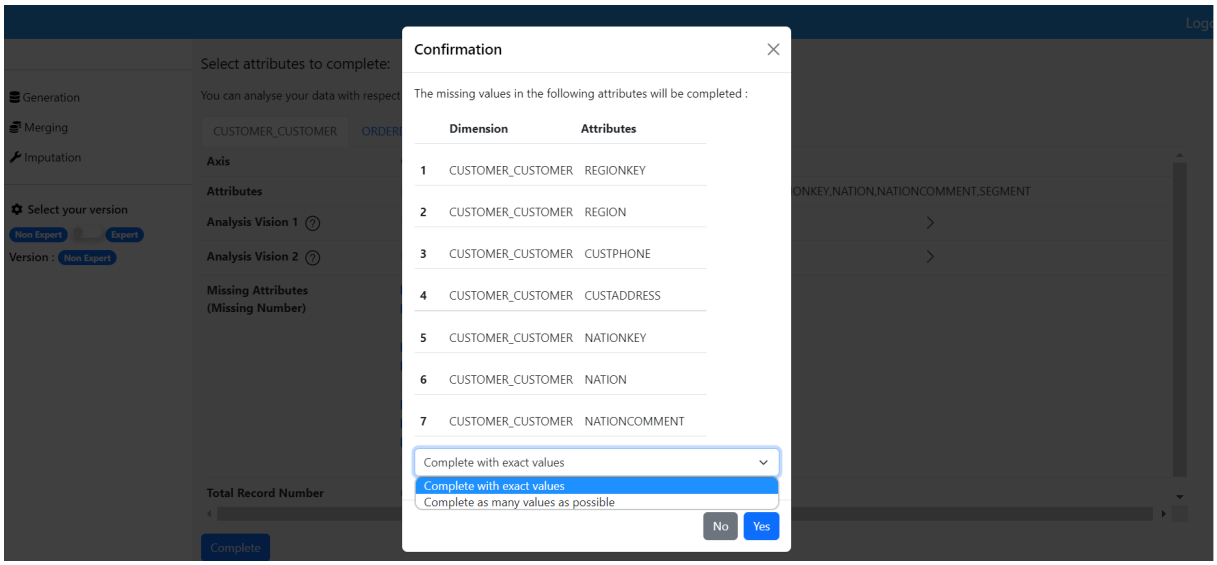


Figure V.22: Imputation confirmation in non-expert version

The confirmation window in the expert version is shown in Fig. V.23. In this version, the two approaches are displayed as their original names “hierarchical imputation” and “Hie-OLAPKNN”. When the user chooses using Hie-OLAPKNN, the user can manually change the value of k and choose the hierarchy level weight.

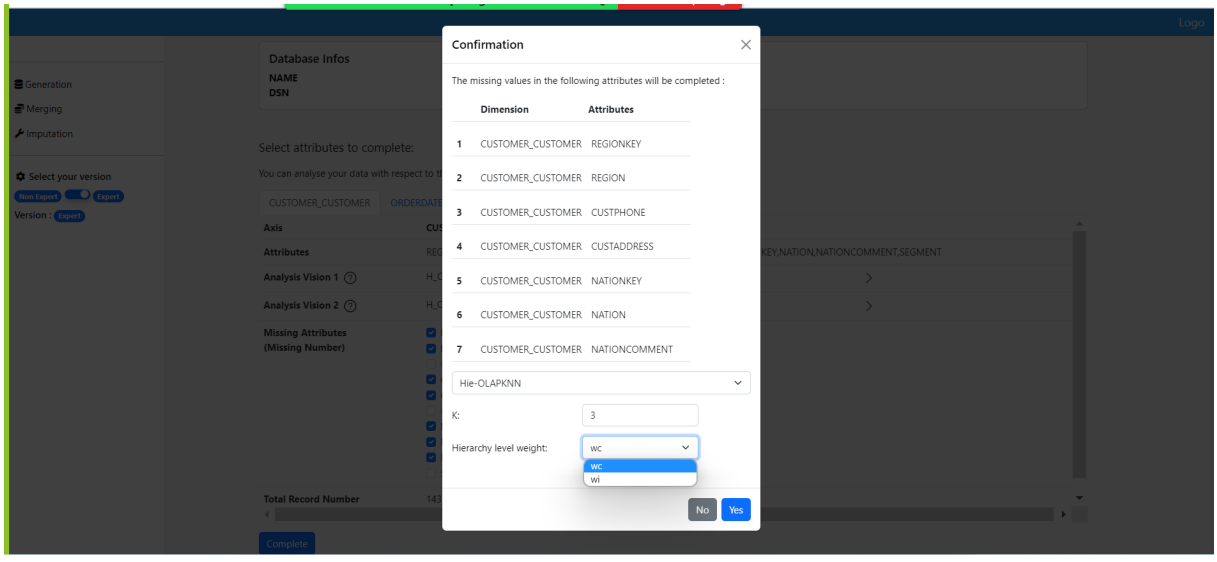


Figure V.23: Imputation confirmation in expert version

Finally, after the confirmation, the imputation is carried out and an interface of result is shown (Fig. V.24). The interface shows the missing value number, the replaced value number and the replaced value rate for each attribute.

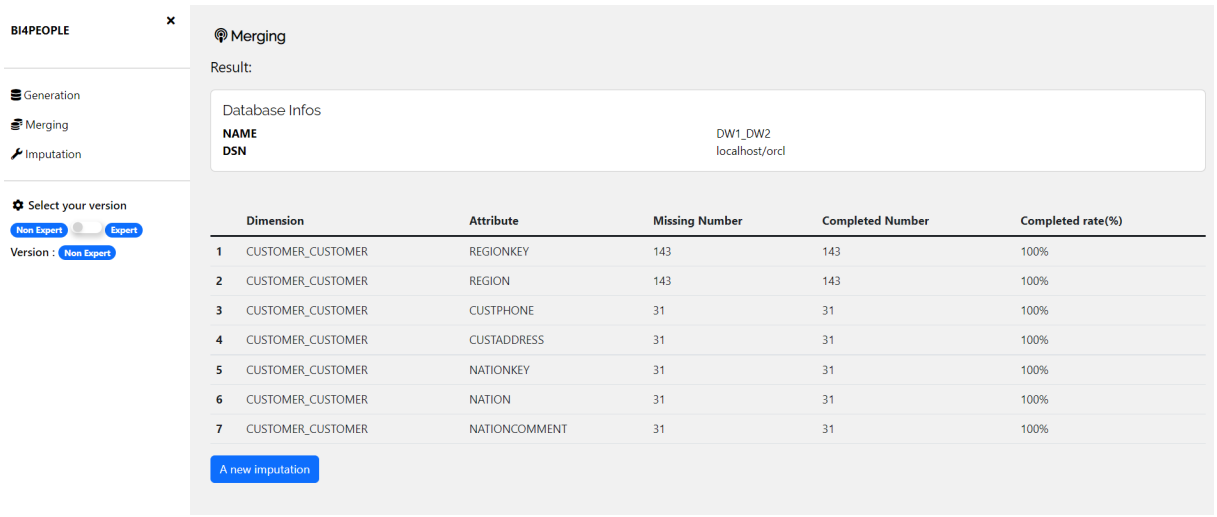


Figure V.24: Imputation result

4.2 Back-end

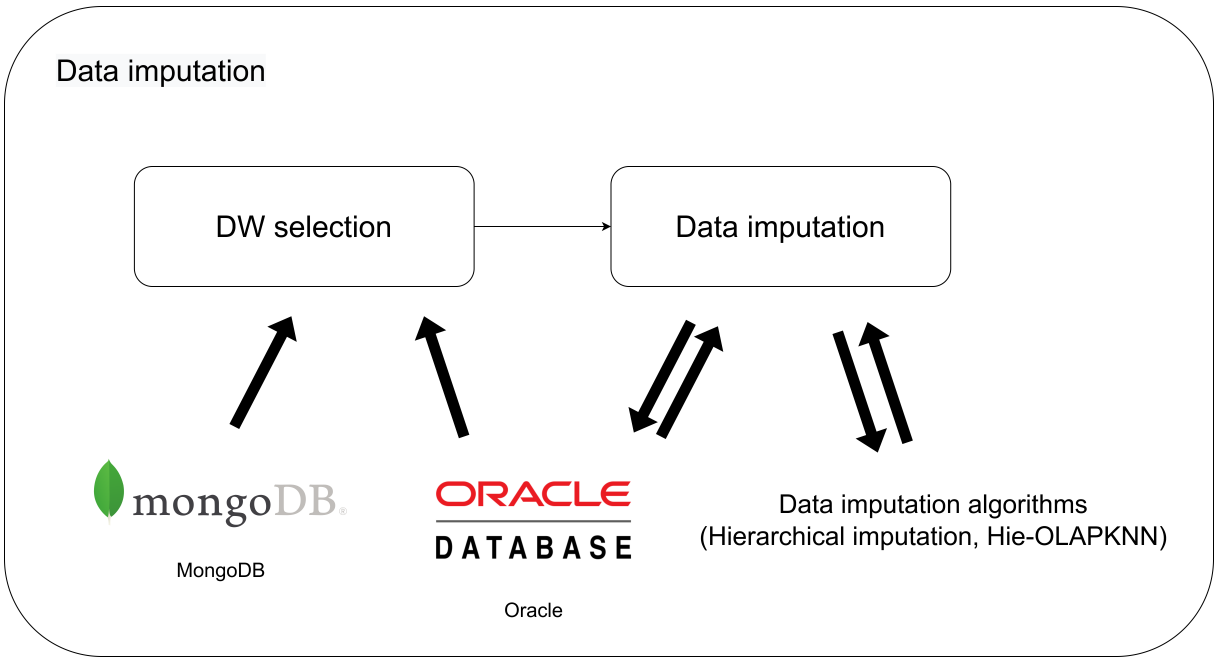


Figure V.25: Back-end illustration of data imputation

We illustrate the back-up in Fig. V.25. When the user selects a DW, the application will receive the information of the DW to connect to the Oracle database. Then the DW schema information is extracted from MongoDB. And the missing value number of each attribute is obtained by SQL queries from Oracle. When the user selects the attributes to be replaced and the imputation algorithm, the selected attributes and imputation algorithm will be sent to the back-end. If the user selects to use hierarchical imputation, the inter-dimensional imputation (Algo. 10) and intra-dimensional imputation (Algo. 9) are carried out. If the user selects to apply Hie-OLAPKNN imputation, the algorithm param-

eters are obtained by the expert user's input or by the default values. The hierarchical imputation Algo. 10 and Algo. 9 first run, then the OLAPKNN algorithm (Algo. 11) is carried out with the obtained parameters.

5 Conclusion

In this chapter, we illustrated the implementation of our solution. We developed an application by integrating all parts of our solution including automatic DW design and implementation, automatic DW merging and data imputation. The application provides a user-friendly interface which allows the user to easily use our proposed three functionalities. Since our target users are non-expert users, we created a non-expert version where the technical vocabularies are replaced by non-technical ones and where there are supplementary explications. We also created an expert version which provides the user with more operation choices. The front-end interface of each functionality was shown with an example of TPCH data to illustrate the use of the application and the back-end functioning was also explained.

Chapter VI

Conclusion

Contents

1 Contributions

It is hard for small companies and organisations to take advantage of BI systems to analyse their data mostly in tabular form due to the lack of experts and budget. Thus, it is necessary to automate the DW design and implementation process from tabular data, which induces three main challenges. (1) Automatic DW design requires the detection of different multidimensional components, but tabular data do not have an explicit schema that represents relationships between attributes. (2) Users' data may come from multiple sources, so DWs need to be merged at both schema and instance levels by considering different multidimensional components. (3) The different original DW attributes cause missing data in the merged DW and should be replaced, because missing data make aggregated data incomplete and may lead to inaccurate decision making. Thus, in this thesis, we proposed a complete solution to automate the DW design and implementation from tabular data to allow non-expert users taking advantage of BI systems for decision making. The solution consists of three parts: (1) automatic DW design from tabular data, (2) automatic DW merging and (3) dimensional data imputation.

1.1 Contributions on Automatic DW Design from Tabular Data

First, we proposed a process to automatically design a DW from tabular data without explicit schema. The process is composed of measure detection and dimension detection.

1.1.1 Measure Detection

Measure detection aims to find all potential measures to build facts. Our measure detection approach is based on machine learning. We considered numerical columns as candidate measures. So first, we proposed to carry out a pre-processing step to identify all numerical columns. Second, we proposed three categories of features including general features, statistical features and inter-column features that are defined by the characteristics of measures. Third, features are extracted and fed into machine learning algorithms for model training or measure prediction. Finally, users are asked to validate the measures. The experiment results have shown that (1) random forest is the machine learning algorithm having the best effectiveness for measure detection, with a F-score of 93.65%, and has an F-score augmentation of 17.2% with respect to baseline methods; (2) each category of features has a contribution to measure detection; (3) the trained model is generic regardless of the data source or domain; and (4) the feature values vary with respect to different algorithms and the location ratio is the most important feature in random forest.

1.1.2 Dimension Detection

Our dimension detection consists of hierarchy detection and the distinction of parameters and weak attributes. Hierarchy detection is based on functional dependencies that exist among hierarchy levels. We modelled the discovered functional dependencies as tree

structures and by finding the root-to-leaf paths to retrieve hierarchies. Tree roots are dimension identifiers. The distinction of parameters and weak attributes is carried out for equivalent attributes and for the detected highest-granularity levels. We proposed several syntactic and semantic rules based on the characteristics of parameters and weak attributes. Experiment results have shown that, for the applied datasets, our approach detect all dimension identifiers and attributes in each dimension with 100% precision. Our approach also accurately identifies most of the parameters, weak attributes and accurately detects most hierarchies or equivalent attribute relationships.

1.2 Contributions on Automatic DW Merging

We proposed a process to merge DWs, which operates at both schema and instance levels. The process considers the merging of all multidimensional components and is composed of level merging, hierarchy merging, dimension merging and star schema merging. Level merging merges identical parameters into one parameter with their weak attributes merged. Hierarchy merging creates new hierarchies and applies level merging. Dimension merging is performed based on hierarchy merging. In star schema merging, facts may be merged or not according to their associated dimensions to generate a star or constellation schema. We carried out experiments with the TPC-H benchmark's data. The results revealed that our process is able to correctly merge two DWs at both schema and instance levels and may generate a star or constellation schema.

1.3 Contributions on Dimensional Data Imputation

We proposed a hybrid imputation approach named Hie-OLAPKNN for dimensional missing data. The approach combines a hierarchical imputation (Hie) and a k-nearest neighbor-based imputation (OLAPKNN). Hierarchical imputation is based on the functional dependencies in intra- and inter-dimensional hierarchies. It is thus reliable and replaces missing values with exact values. However, when the distinct value ratios of the parameters are high, the number of missing values that can be replaced is limited. Therefore, OLAPKNN can then be carried out based on missing value instances' nearest neighbors. We defined specific dimension instance distance metrics for looking for nearest neighbors, which takes DW dimension structure and characteristics into account. We also proposed the creation of candidate lists based on the dependency constraints among hierarchy levels. Moreover, we proposed to solve the dependency conflicts between replaced values and their lower-granularity level parameter values. Therefore, OLAPKNN can replace missing values by following dependency constraints. We conducted experiments comparing Hie-OLAPKNN with other approaches from the literature. The results showed that Hie-OLAPKNN outperforms the other approaches in terms of (1) effectiveness, e.g., up to 44.84% higher F-score in the $F1$ dataset; (2) efficiency, e.g., up to 2.51s less run time in the *Adventure* dataset and (3) respect of hierarchy strictness.

1.4 Contributions on Automatic Data Warehousing System

To bring up a complete solution for automatic DW design, we implemented our solution by developing an application that enforces the functionalities of automatic DW design, automatic DW merging and dimensional data imputation. The application provides a user-friendly interface allowing the user to easily carry out different functions. There are two versions in the application. The non-expert version is the default one, where vocabulary is non-technical. Moreover, supplementary explanations help non-expert users understand the DW's information. The expert version offers more operations to expert users so that they can modify the detected schema according to their requirement and set customized algorithm parameters.

2 Future Work

2.1 Short-term Plan

2.1.1 Automatic DW Design Approach Augmentation

In the short term, we intend to enhance our automatic DW design approach. For measure detection, we will consider the automatic detection of textual measures (Ravat et al., 2008b) that may exist in DWs. For dimension detection, we will integrate some commonly used ontologies and apply ontology matching techniques (Euzenat et al., 2007) to help detect hierarchies and identify attributes as parameters or weak attributes. Integration of an ontology can provide pre-defined data semantics that are useful for algorithms to identify relationships between attributes.

2.1.2 Imputation Approach Extension

Our OLAPKNN dimensional data imputation considers data of one dimension, we will extend it by also considering inter-dimensional data relationships such as sequential patterns (Plantevit et al., 2010). Our imputation approach focused on categorical data in dimensions. There are also other types of data in DWs, such as non-categorical textual data in dimensions and numerical data in dimensions and facts. Therefore, we will also propose imputation algorithms to replace missing data of these types. For non-categorical textual data in dimensions, the replaced values usually do not exist in the DW. Thus, we will replace them with the help of external source-based imputation such as crowdsourcing-based approaches (Ye and Wang, 2014; Ye et al., 2020) and web information-based approaches (Li et al., 2014; Tang et al., 2017; Liu et al., 2018). For numerical data in dimensions and measures, we will apply statistical-based (Graham et al., 2009; Lin, 2010; Schneider, 2001) and machine learning-based imputation approaches (Miao et al., 2018; Lin and Tsai, 2020; Osman et al., 2018) by considering DW structure characteristics.

2.2 Mid-term Plan

In the middle term, we intend to consider the data evolution that may occur at the schema and instance levels. The user may obtain new data in existing or non-existing attributes, which leads to DW evolution at the instance level or at both schema and instance levels, respectively. We will thus propose an automatic process to update DWs at both levels. In terms of updated DW, the process will propose the choices of updating the original DW generated from tabular data, the merged DW based on the original one, or both ones. In terms of update processing way, we will propose batch or stream update. At the schema level, the process will automatically detect new multidimensional components and relationships and merge them with the original schema. At the instance level, the process will automatically update the DW data by inserting new instances and updating existing instances.

2.3 Long-term Plan

In the long term, we intend to extend our automatic DW design approach so that it can be applied for big data analytics (Cuzzocrea et al., 2013). To do this, we have to first extend our approach for other types of data, since there are various data types involved in big data management solutions such as data lake (Ravat and Zhao, 2019). We will address data types such as semi-structured data type like JavaScript Object Notation (JSON) or non-structure data type like Portable Document Format (PDF) and images. Regarding JSON data, there are existing approaches (Piech and Marcjan, 2018; Bahta and Atay, 2019) that transform JSON data into relational data so that our approach can then be applied. There are also approaches (Frozza et al., 2018; Spoth et al., 2021) discovering schemas from JSON data, which are helpful for detecting multidimensional schemas. Regarding PDF files and images, tables can be extracted by existing approaches (Khusro et al., 2015; Burdick et al., 2020) and then be processed by our approach. In addition, DWs are implemented in a parallel way (Santoso et al., 2017) in distributed big data architectures such as Hadoop. We will have to parallelize the processing and DW implementation of our approach to fit big data architectures (Zhang et al., 2016). Recently, the concept of Lakehouse (Armbrust et al., 2021) is proposed, which combines the low-cost storage advantage of data lakes and the powerful management advantage of data warehouses. Data in lakehouses are stored in Parquet format [1], which also has a tabular structure (Peltenburg et al., 2020). Thus eventually, we can integrate our solution for automatically designing DW schemas in lakehouses.

[1] `https://parquet.apache.org/`

Annexes

Appendix A

Ground truth and Detected Schemas in Dimension Detection

1 Dataset - Example

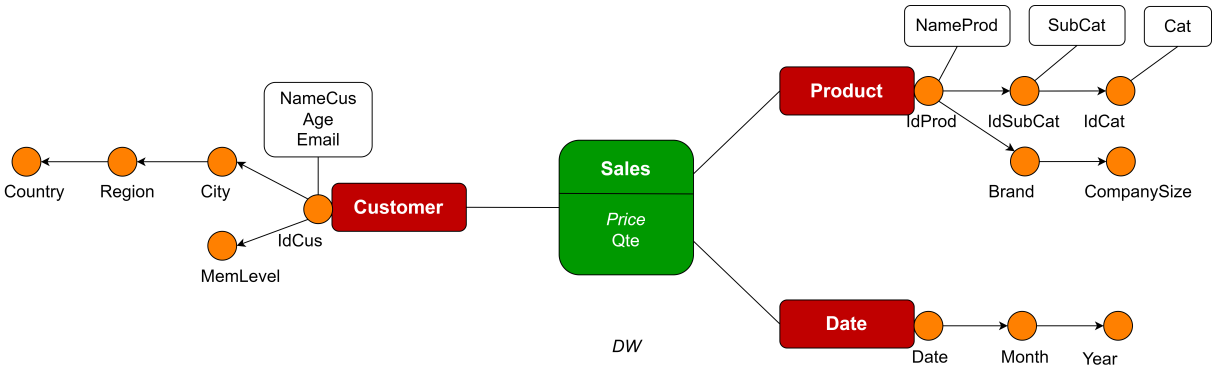


Figure A.1: Ground truth schema of dataset Example

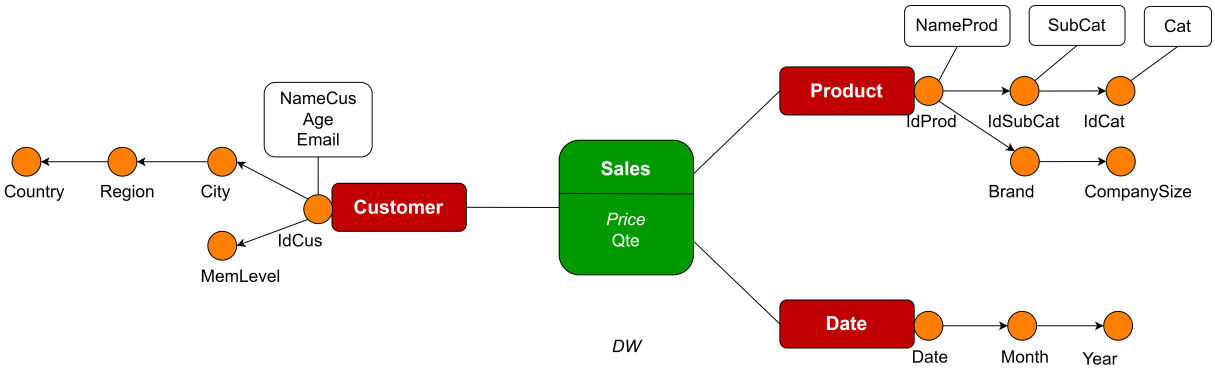


Figure A.2: Detected schema of dataset Example

2 Dataset - Sales1

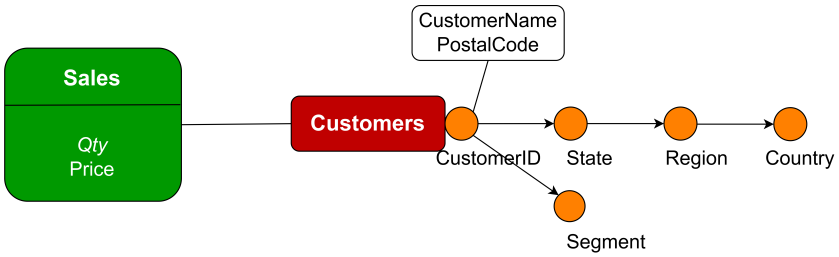


Figure A.3: Ground truth schema of dataset Sales1

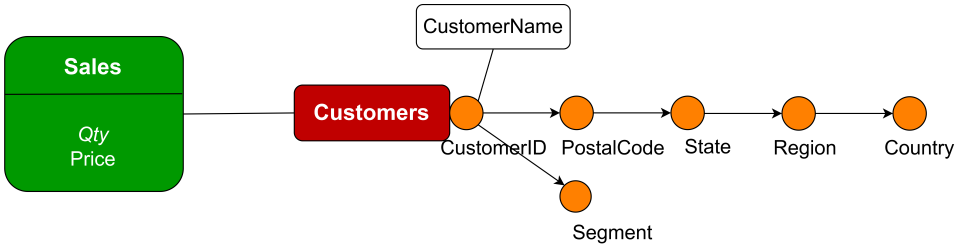


Figure A.4: Detected schema of dataset Sales1

3 Dataset - Sales2

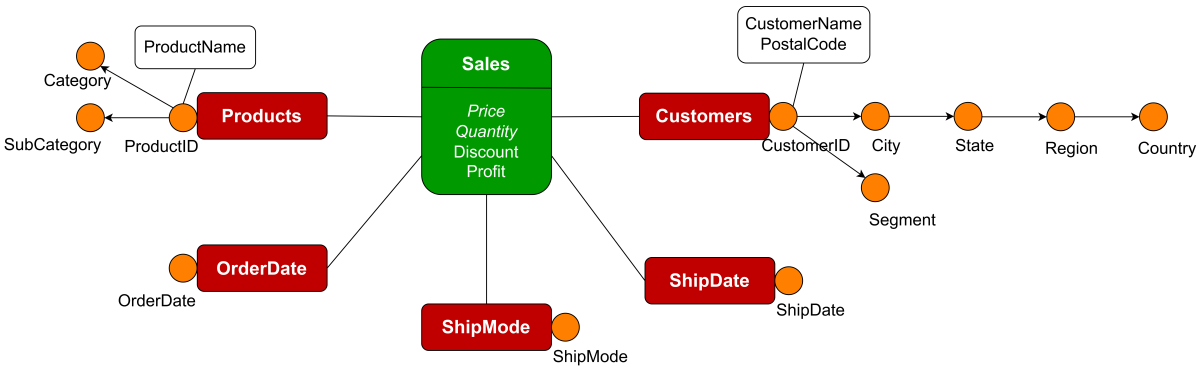


Figure A.5: Ground truth schema of dataset Sales2

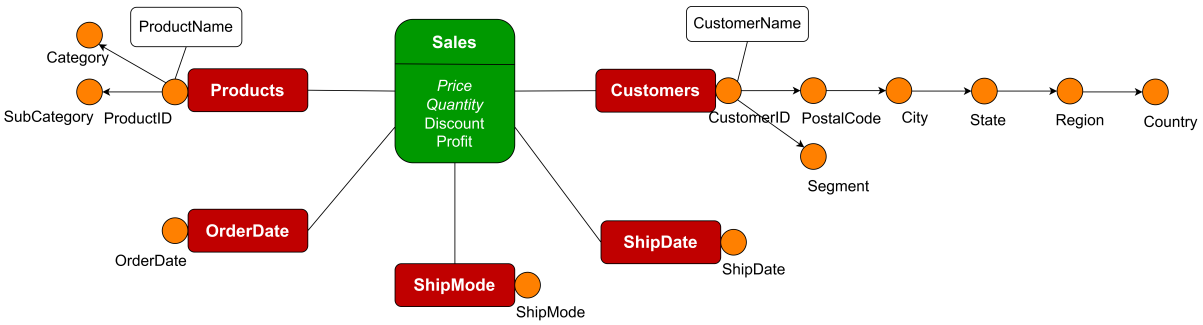


Figure A.6: Detected schema of dataset Sales2

4 Dataset - DevApp

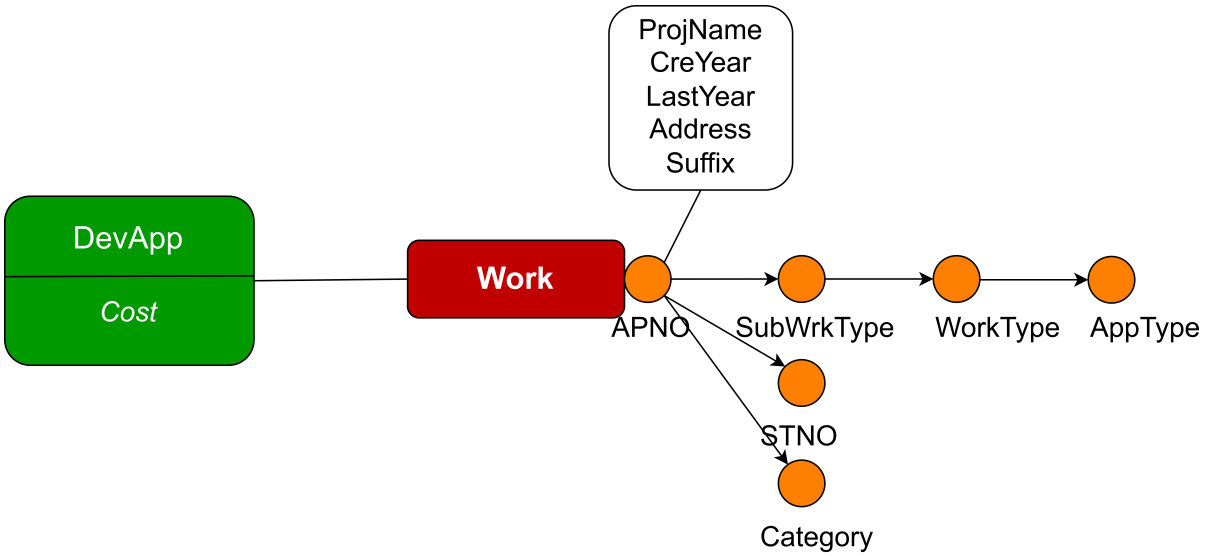


Figure A.7: Ground truth schema of dataset DevApp

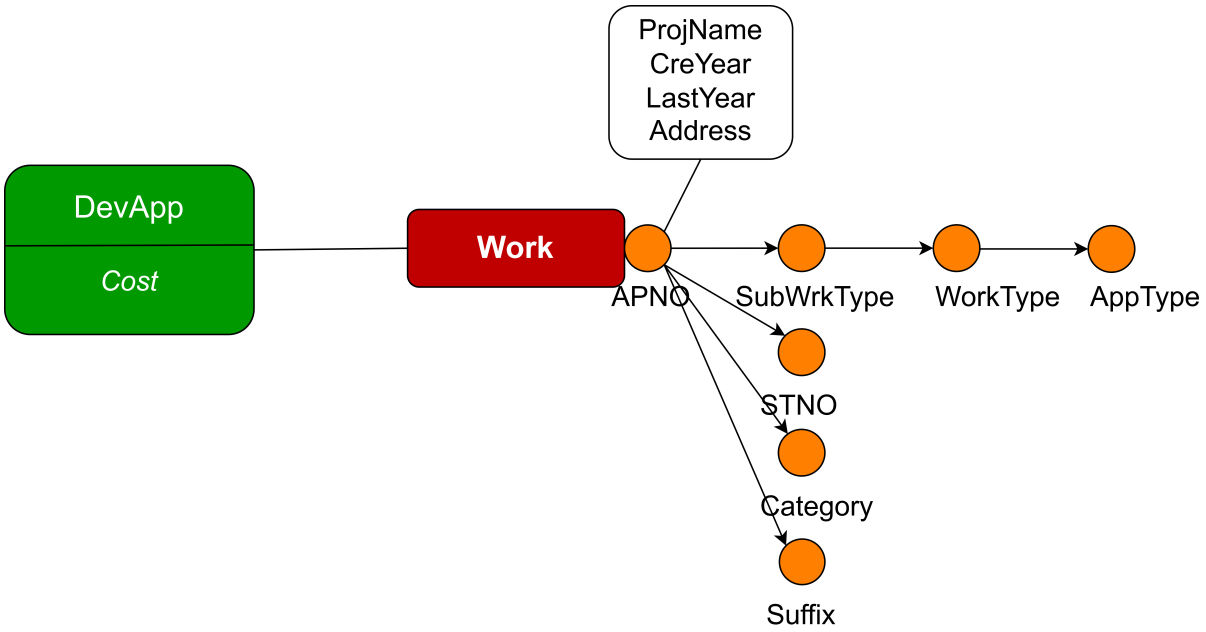


Figure A.8: Detected schema of dataset DevApp

5 Dataset - Countries

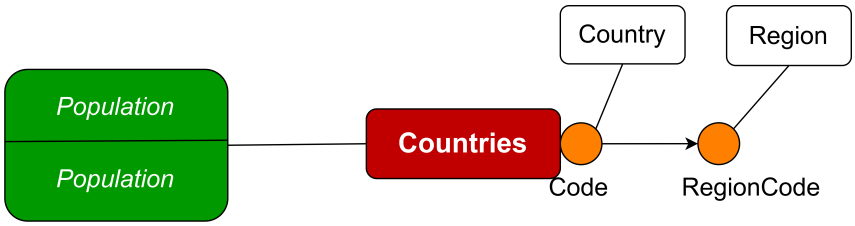


Figure A.9: Ground truth schema of dataset Countries

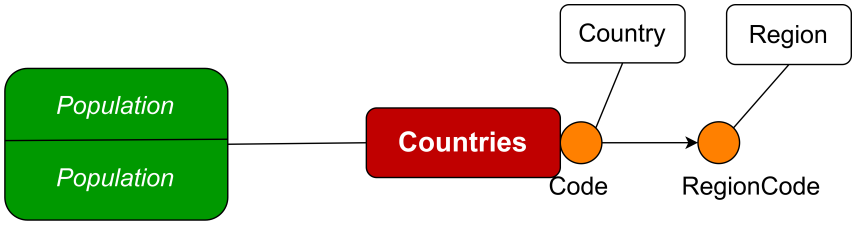


Figure A.10: Detected schema of dataset Countries

6 Dataset - Covid

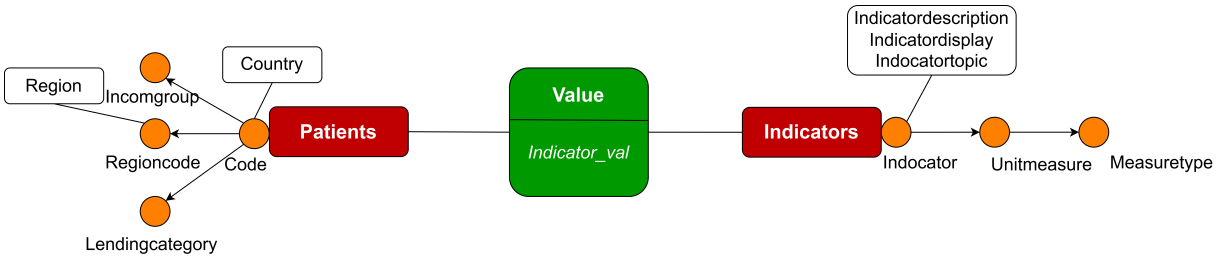


Figure A.11: Ground truth schema of dataset Covid

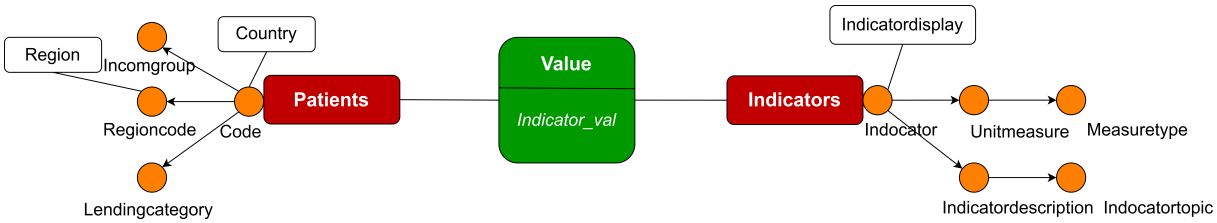


Figure A.12: Detected schema of dataset Covid

Appendix B

DW Schemas in Imputation Experiments

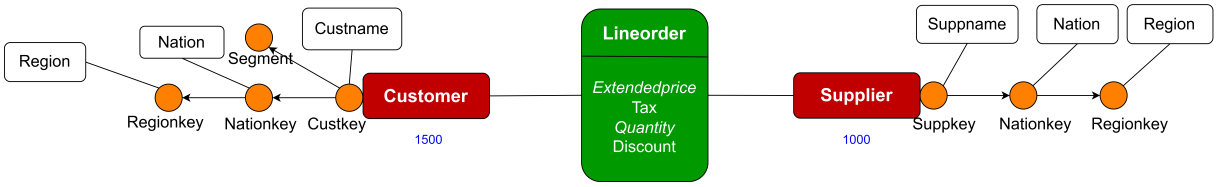


Figure B.1: Schema of dataset TPCH

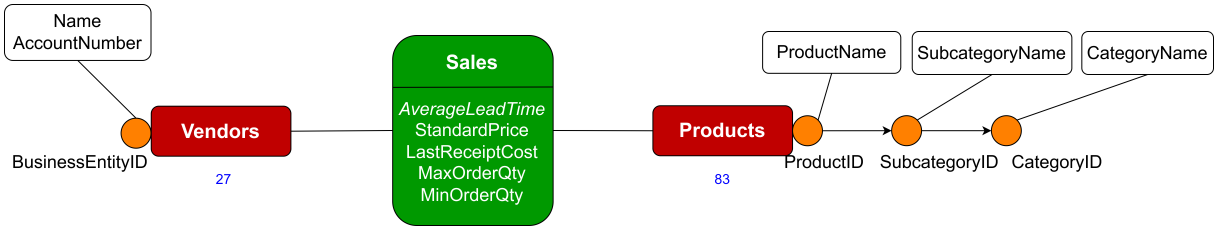


Figure B.2: Schema of dataset Adventure

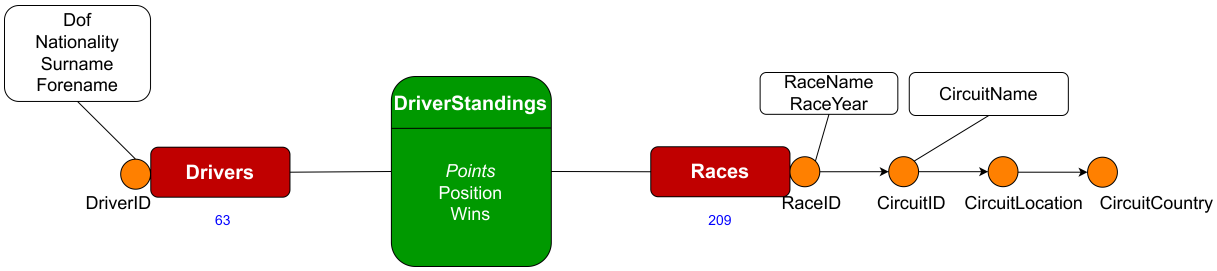


Figure B.3: Schema of dataset F1

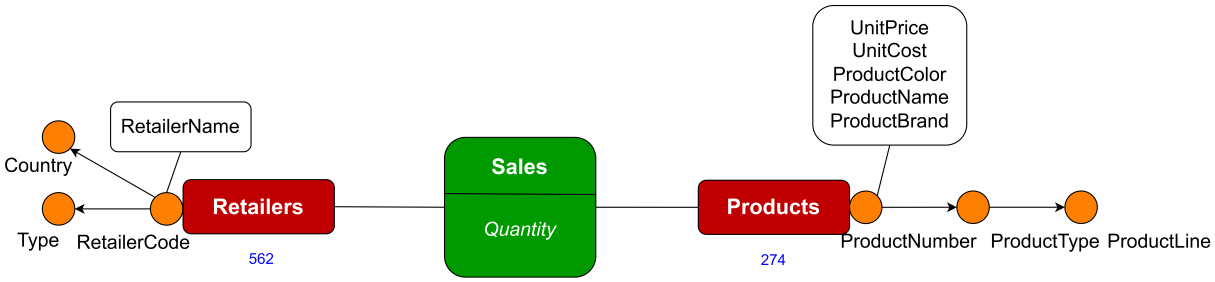


Figure B.4: Schema of dataset GoSales

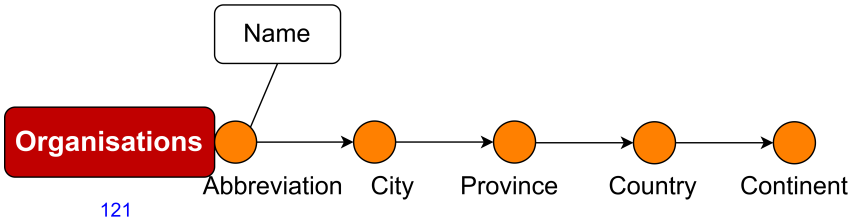


Figure B.5: Schema of dataset Organisation

Bibliography

Abelló, A., Darmont, J., Etcheverry, L., Golfarelli, M., Mazón, J.-N., Naumann, F., Pedersen, T., Rizzi, S. B., Trujillo, J., Vassiliadis, P., and Vossen., G. (2013). Fusion cubes: Towards self-service business intelligence. *International Journal of Data Warehousing and Mining*, 9(12):66–88.

Adelfio, M. D. and Samet, H. (2013). Schema extraction for tabular data on the web. *VLDB Endowment*, 6(6):421–432.

Agrawal, R., Srikant, R., et al. (1994). Fast algorithms for mining association rules. In *Proc. 20th int. conf. very large data bases, VLDB*, volume 1215, pages 487–499. Citeseer.

Allison, P. D. (2005). Imputation of categorical variables with proc mi. *SUGI 30 proceedings*, 113(30):1–14.

Alobaid, A., Kacprzak, E., and Corcho, O. (2019). Typology-based semantic labeling of numeric tabular data. *Semantic Web*, 1(0):1–5.

Amanzougarene, F., Zeitouni, K., and Chachoua, M. (2014). Predicting missing values in a data warehouse by combining constraint programming and knn. In *EDA*.

Andridge, R. R. and Little, R. J. (2010). A review of hot deck imputation for survey non-response. *International statistical review*, 78(1):40–64.

Ariyan, S. and Bertossi, L. (2011). Structural repairs of multidimensional databases. In *5th Alberto Mendelzon Inter. Workshop on Foundations of Data Management*, volume 748.

Armbrust, M., Ghodsi, A., Xin, R., and Zaharia, M. (2021). Lakehouse: a new generation of open platforms that unify data warehousing and advanced analytics. In *Proceedings of CIDR*.

Armstrong, W. W. (1974). Dependency structures of data base relationships. In *IFIP congress*, volume 74, pages 580–583.

Aydilek, I. B. and Arslan, A. (2013). A hybrid method for imputation of missing values using optimized fuzzy c-means with support vector regression and a genetic algorithm. *Information Sciences*, 233:25–35.

Bahta, R. and Atay, M. (2019). Translating json data into relational data using schema-oblivious approaches. In *Proceedings of the 2019 ACM Southeast Conference*, pages 233–236.

Banek, M., Vrdoljak, B., Tjoa, A. M., and Skočir, Z. (2007). Automating the schema matching process for heterogeneous data warehouses. In *Data Warehousing and Knowl-*

edge Discovery, pages 45–54.

Baraldi, A. N. and Enders, C. K. (2010). An introduction to modern missing data analyses. *Journal of school psychology*, 48(1):5–37.

Beaumont, J.-F. (2000). On regression imputation in the presence of nonignorable nonresponse. In *Proceeding of the Survey Research Methods Section, American Statistical Asssociation*, pages 580–585.

Beretta, L. and Santaniello, A. (2016). Nearest neighbor imputation algorithms: a critical evaluation. *BMC medical informatics and decision making*, 16(3):197–208.

Bergamaschi, S., Olaru, M., Sorrentino, S., and Vincini, M. (2011). Semi-automatic discovery of mappings between heterogeneous data warehouse dimensions. *J. of Computing and Information Technology*, pages 38–46.

Bernstein, P. A., Madhavan, J., and Rahm, E. (2011a). Generic schema matching, ten years later. *Proc. VLDB Endow.*, 4(11):695–701.

Bernstein, P. A., Madhavan, J., and Rahm, E. (2011b). Generic schema matching, ten years later. *Proceedings of the VLDB Endowment*, 4(11):695–701.

Bimonte, S., Ren, L., and Koueya, N. (2020). A linear programming-based framework for handling missing data in multi-granular data warehouses. *Data & Knowledge Engineering*, 128.

Bleiholder, J. and Naumann, F. (2009). Data fusion. *ACM computing surveys (CSUR)*, 41(1):1–41.

Boehnlein, M. and Ulbrich-vom Ende, A. (1999). Deriving initial data warehouse structures from the conceptual data models of the underlying operational information systems. In *Proceedings of the 2nd ACM international workshop on Data warehousing and OLAP*, pages 15–21.

Bohannon, P., Fan, W., Geerts, F., Jia, X., and Kementsietsidis, A. (2006). Conditional functional dependencies for data cleaning. In *2007 IEEE 23rd international conference on data engineering*, pages 746–755. IEEE.

Borisov, V., Leemann, T., Seßler, K., Haug, J., Pawelczyk, M., and Kasneci, G. (2021). Deep neural networks and tabular data: A survey. *arXiv preprint arXiv:2110.01889*.

Braunschweig, K., Eberius, J., Thiele, M., and Lehner, W. (2012). The state of open data. *Limits of current open data platforms*.

Burdick, D., Danilevsky, M., Evfimievski, A. V., Katsis, Y., and Wang, N. (2020). Table extraction and understanding for scientific and enterprise applications. *Proceedings of the VLDB Endowment*, 13(12):3433–3436.

Burgette, L. F. and Reiter, J. P. (2010). Multiple imputation for missing data via sequential regression trees. *American journal of epidemiology*, 172(9):1070–1076.

Céret, E., Dupuy-Chessa, S., Calvary, G., Front, A., and Rieu, D. (2013). A taxonomy of design methods process models. *Information and Software Technology*, 55(5):795–821.

Chaudhuri, S., Dayal, U., and Narasayya, V. (2011). An overview of business intelligence technology. *Communications of the ACM*, 54(8):88–98.

Chen, Z. and Cafarella, M. (2013). Automatic web spreadsheet data extraction. In *3rd International Workshop on Semantic Search Over the Web*, pages 1–8.

Chhabra, G., Vashisht, V., and Ranjan, J. (2018). Missing value imputation using hybrid k-means and association rules. In *2018 International Conference on Advances in Computing, Communication Control and Networking (ICACCCN)*, pages 501–509. IEEE.

Choudhury, S. J. and Pal, N. R. (2019). Imputation of missing data with neural networks for classification. *Knowledge-Based Systems*, 182:104838.

Chugh, R. and Grandhi, S. (2013). Why business intelligence?: Significance of business intelligence tools and integrating bi governance with corporate governance. *International Journal of E-Entrepreneurship and Innovation (IJEEI)*, 4(2):1–14.

Cuzzocrea, A., Bellatreche, L., and Song, I.-Y. (2013). Data warehousing and olap over big data: current challenges and future research directions. In *Proceedings of the sixteenth international workshop on Data warehousing and OLAP*, pages 67–70.

de S. Ribeiro, L., Goldschmidt, R. R., and Cavalcanti, M. C. (2011). Complementing data in the etl process. In *DaWaK*, pages 112–123.

Ding, Y. and Simonoff, J. S. (2010). An investigation of missing data methods for classification trees applied to binary response data. *Journal of Machine Learning Research*, 11(1).

Domeniconi, C. and Yan, B. (2004). Nearest neighbor ensemble. In *ICPR*, volume 1.

Dorneles, C. F., Gonçalves, R., and dos Santos Mello, R. (2011). Approximate data instance matching: a survey. *Knowledge and Information Systems*, 27(1):1–21.

Du, L., Gao, F., Chen, X., Jia, R., Wang, J., Zhang, J., Han, S., and Zhang, D. (2021). Tabularnet: A neural network architecture for understanding semantic structures of tabular data. In *27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, page 322–331.

Dudani, S. A. (1976). The distance-weighted k-nearest-neighbor rule. *IEEE Transactions on Systems, Man, and Cybernetics*, (4):325–327.

Elamin, E., Altalhi, A., and Feki, J. (2017). Heuristic based approach for automating multidimensional schemas construction. *International Journal of Computer and Information Technology*, 764.

Elamin, E., Alzaidi, A., and Feki, J. (2018). A semantic resource based approach for star schemas matching. *Inter. J. of Database Management Systems*, 10(6).

Elavarasi, S. A., Akilandeswari, J., and Menaga, K. (2014). A survey on semantic similarity measure. *Inter. J. of Research in Advent Technology*, 2.

Estellés-Arolas, E. and González-Ladrón-de Guevara, F. (2012). Towards an integrated crowdsourcing definition. *Journal of Information science*, 38(2):189–200.

Euzenat, J., Shvaiko, P., et al. (2007). *Ontology matching*, volume 18. Springer.

Fan, W. (2008). Dependencies revisited for improving data quality. In *Proceedings of the twenty-seventh ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 159–170.

Fan, W., Jianzhong, L., Shuai, M., Nan, T., and Wenyuan, Y. (2010). Towards certain fixes with editing rules and master data. *The VLDB Journal*, pages 173–184.

Feki, J., Majdoubi, J., and Gargouri, F. (2005). A two-phase approach for multidimensional schemes integration. In *17th Inter. Conference on Software Engineering and Knowledge Engineering*, pages 498–503.

Fisher, A., Rudin, C., and Dominici, F. (2019). All models are wrong, but many are useful: Learning a variable's importance by studying an entire class of prediction models simultaneously. *Journal of Machine Learning Research*, 20(177):1–81.

Frozza, A. A., dos Santos Mello, R., and da Costa, F. d. S. (2018). An approach for schema extraction of json and extended json document collections. In *2018 IEEE International Conference on Information Reuse and Integration (IRI)*, pages 356–363. IEEE.

Garcia, A. J. and Hruschka, E. R. (2005). Naive bayes as an imputation tool for classification problems. In *Fifth International Conference on Hybrid Intelligent Systems (HIS'05)*, pages 3–pp. IEEE.

García-Laencina, P. J., Sancho-Gómez, J.-L., Figueiras-Vidal, A. R., and Verleysen, M. (2009). K nearest neighbours with mutual information for simultaneous classification and missing data imputation. *Neurocomputing*, 72(7):1483–1493.

Gheyas, I. A. and Smith, L. S. (2010). A neural network-based framework for the reconstruction of incomplete data sets. *Neurocomputing*, 73(16-18):3039–3065.

Golfarelli, M., Rizzi, S., , and Vrdoljak, B. (2001). Data warehouse design from xml sources. In *4th ACM international workshop on Data warehousing and OLAP*, page

40–47.

Golfarelli, M. and Rizzi, S. (2009). *Data Warehouse Design: Modern Principles and Methodologies*. McGraw-Hill, Inc., 1 edition.

Grabova, O., Darmont, J., Chauchat, J.-H., and Zolotaryova, I. (2010). Business intelligence for small and middle-sized entreprises. *SIGMOD Record*, 39(2):39–50.

Graham, J. W. et al. (2009). Missing data analysis: Making it work in the real world. *Annual review of psychology*, 60(1):549–576.

Han, J., Kamber, M., and Pei, J. (2012). Preface. In *Data Mining*. third edition.

Hapfelmeier, A., Hothorn, T., and Ulm, K. (2012). Recursive partitioning on incomplete data using surrogate decisions and multiple imputation. *Computational Statistics & Data Analysis*, 56(6):1552–1565.

Horner, J., Song, I.-Y., and Chen, P. P. (2004). An analysis of additivity in olap systems. In *7th ACM International Workshop on Data Warehousing and OLAP*, page 83–91.

Hruschka, E. R., Hruschka, E. R., and Ebecken, N. F. (2007). Bayesian networks for imputation in classification problems. *Journal of Intelligent Information Systems*, 29(3):231–252.

Huang, C.-C. and Lee, H.-M. (2004). A grey-based nearest neighbor approach for missing attribute value prediction. *Applied Intelligence*, 20(3):239–252.

I.-Y.Song, Khare, R., and Dai, B. (2007). Samstar: A semi-automated lexical method for generating star schemas from an entity-relationship diagram. In *ACM 10th International Workshop on Data Warehousing and OLAP*, pages 9–16.

Jatnika, D., Bijaksana, M. A., and Suryani, A. A. (2019). Word2vec model analysis for semantic similarities in english words. *Procedia Computer Science*, 157:160–167.

Jensen, M. R., Holmgren, T., and Pedersen, T. B. (2004). Discovering multidimensional structure in relational data. In Kambayashi, Y., Mohania, M., and Wöß, W., editors, *Data Warehousing and Knowledge Discovery*, pages 138–148.

Khusro, S., Latif, A., and Ullah, I. (2015). On methods and tools of table detection, extraction and annotation in pdf documents. *Journal of Information Science*, 41(1):41–57.

Kimball, R. and Ross, M. (2011). *The data warehouse toolkit: the complete guide to dimensional modeling*. John Wiley & Sons.

Koci, E., Thiele, M., Romero, O., and Lehner, W. (2016). A machine learning approach for layout inference in spreadsheets. In *International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management*, page 77–88.

Kokla, M., Virtanen, J., Kolehmainen, M., Paananen, J., and Hanhineva, K. (2019). Random forest-based imputation outperforms other methods for imputing lc-ms metabolomics data: a comparative study. *BMC bioinformatics*, 20(1):1–11.

Kwakye, M., Kiringa, I., and Viktor, H. L. (2013). Merging multidimensional data models: A practical approach for schema and data instances. In *5th Inter. Conference on Advances in Databases, Data, and Knowledge Applications*.

Lapa, J., Bernardino, J., and Figueiredo, A. (2014). A comparative analysis of open source business intelligence platforms. In *Proceedings of the International Conference on Information Systems and Design of Communication*, pages 86–92.

Latifi, H. and Koch, B. (2012). Evaluation of most similar neighbour and random forest methods for imputing forest inventory variables using data from target and auxiliary stands. *International Journal of Remote Sensing*, 33(21):6668–6694.

Lautert, L. R., Scheidt, M. M., and Dorneles, C. F. (2013). Web table taxonomy and formalization. *ACM SIGMOD Record*, 42(3):28–33.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521:436–444.

Lee, D. and Chu, W. W. (2000). Constraints-preserving transformation from xml document type definition to relational schema. In *International Conference on Conceptual Modeling*, pages 323–338. Springer.

Lee, J. Y. and Styczynski, M. P. (2018). Ns-knn: a modified k-nearest neighbors approach for imputing metabolomics data. *Metabolomics*, 14(12):1–12.

Li, D., Deogun, J., Spaulding, W., and Shuart, B. (2004a). Towards missing data imputation: A study of fuzzy k-means clustering method. In *RSCTC*, pages 573–579.

Li, D., Deogun, J., Spaulding, W., and Shuart, B. (2004b). Towards missing data imputation: a study of fuzzy k-means clustering method. In *International conference on rough sets and current trends in computing*, pages 573–579. Springer.

Li, Y. and Parker, L. E. (2014). Nearest neighbor imputation using spatial–temporal correlations in wireless sensor networks. *Information Fusion*, 15:64–79.

Li, Z., Qin, L., Cheng, H., Zhang, X., and Zhou, X. (2015). Trip: An interactive retrieving-inferring data imputation approach. *IEEE Transactions on Knowledge and Data Engineering*, 27(9):2550–2563.

Li, Z., Sharaf, M. A., Sitbon, L., Sadiq, S., Indulska, M., and Zhou, X. (2014). A web-based approach to data imputation. *World Wide Web*, 17(5):873–897.

Liao, Z., Lu, X., Yang, T., and Wang, H. (2009). Missing data imputation: a fuzzy k-means clustering algorithm over sliding window. In *2009 Sixth International Conference*

on Fuzzy Systems and Knowledge Discovery, volume 3, pages 133–137. IEEE.

Lin, J. and Mendelzon, A. O. (1998). Merging databases under constraints. *International Journal of Cooperative Information Systems*, 7(01):55–76.

Lin, T. H. (2010). A comparison of multiple imputation with em algorithm and mcmc method for quality of life missing data. *Quality & quantity*, 44(2):277–287.

Lin, W.-C. and Tsai, C.-F. (2020). Missing value imputation: a review and analysis of the literature (2006–2017). *Artificial Intelligence Review*, 53(2):1487–1509.

Little, R. and Rubin, D. (2019). *Statistical Analysis with Missing Data*. Wiley Series in Probability and Statistics. Wiley.

Liu, H., Li, Z., Chen, Q., and Chen, Z. (2018). Automatic web-based relational data imputation. *Frontiers of Computer Science*, 12(6):1125–1139.

Liu, J., Li, J., Liu, C., and Chen, Y. (2012). Discover dependencies from data—a review. *IEEE Transactions on Knowledge and Data Engineering*, 24(2):251–264.

Llave, M. R. (2017). Business intelligence and analytics in small and medium-sized enterprises: A systematic literature review. *Procedia Computer Science*, 121:194–205.

Malinowski, E. and Zimányi, E. (2004). Olap hierarchies: A conceptual perspective. In *Advanced Information Systems Engineering*, pages 477–491.

Markl, V., Ramsak, F., and Bayer, R. (1999). Improving olap performance by multidimensional hierarchical clustering. In *Proceedings. IDEAS'99. International Database Engineering and Applications Symposium*, pages 165–177. IEEE.

McLachlan, G. J. and Krishnan, T. (2007). *The EM algorithm and extensions*. John Wiley & Sons.

Meng, L., Huang, R., and Gu, J. (2013). A review of semantic similarity measures in wordnet. *Inter. J. of Hybrid Information Technology*, 6.

Miao, X., Gao, Y., Guo, S., and Liu, W. (2018). Incomplete data management: a survey. *Frontiers of Computer Science*, 12:4–25.

Miller, G. A. (1995). Wordnet: A lexical database for english. *Communications of the ACM*, 38(11):39–41.

Moody, D. L. and Kortink, M. A. (2000). From enterprise models to dimensional models: a methodology for data warehouse and data mart design. In *DMDW*, page 5.

Negash, S. and Gray, P. (2008). Business intelligence. In *Handbook on decision support systems 2*, pages 175–193. Springer.

Nelson, G. S. (2010). Business intelligence 2.0: Are we there yet. In *SAS global forum*, volume 2010. Citeseer.

Nishanth, K. J. and Ravi, V. (2016). Probabilistic neural network based categorical data imputation. *Neurocomputing*, 218:17–25.

Olaru, M. O. and Vincini, M. (2014). Integrating multidimensional information for the benefit of collaborative enterprises. *Journal of Digital Information Management*, 12(4).

Osman, M. S., Abu-Mahfouz, A. M., and Page, P. R. (2018). A survey on data imputation techniques: Water distribution system as a use case. *IEEE Access*, 6:63279–63291.

Ouaret, Z., Chalal, R., and Boussaid, O. (2014). An approach for generating an xml data warehouse schema using model transformation language. *Journal of Digital Information Management*, 12(6).

Ouyang, M., Welsh, W. J., and Georgopoulos, P. (2004). Gaussian mixture clustering and imputation of microarray data. *Bioinformatics*, 20(6):917–923.

Pan, R., Yang, T., Cao, J., Lu, K., and Zhang, Z. (2015). Missing data imputation by K nearest neighbours based on grey relational structure and mutual information. *Applied Intelligence*, 43(3):614–632.

Pantanowitz, A. and Marwala, T. (2009). Missing data imputation through the use of the random forest algorithm. In *Advances in computational intelligence*, pages 53–62. Springer.

Papenbrock, T., Bergmann, T., Finke, M., Zwiener, J., and Naumann, F. (2015a). Data profiling with metanome. *VLDB Endowment*, 8(12):1860–1863.

Papenbrock, T., Ehrlich, J., Marten, J., Neubert, T., Rudolph, J., Schönberg, M., Zwiener, J., and Naumann, F. (2015b). Functional dependency discovery: An experimental evaluation of seven algorithms. In *VLDB Endowment*, volume 8, page 1082–1093.

Papenbrock, T. and Naumann, F. (2016). A hybrid approach to functional dependency discovery. In *International Conference on Management of Data*, page 821–833.

Pawlak, Z. (1977). Rough set approach to knowledge-based decision support. *European J. of Operational Research*, 99(1):48–57.

Pawlak, Z. (1982). Rough sets. *Inter. J. of Computer & Information Sciences*, 11(5):341–356.

Pawlak, Z. and Skowron, A. (2007). Rudiments of rough sets. *Information Sciences*, 177(1):3–27.

Peguero, K. and Cheng, X. (2021). Electrolint and security of electron applications.

High-Confidence Computing, 1(2):100032.

Pellissier Tanon, T., Weikum, G., and Suchanek, F. (2020). Yago 4: A reason-able knowledge base. In *European Semantic Web Conference*, pages 583–596. Springer.

Peltenburg, J., Van Leeuwen, L. T., Hoozemans, J., Fang, J., Al-Ars, Z., and Hofstee, H. P. (2020). Battling the cpu bottleneck in apache parquet to arrow conversion using fpga. In *2020 international conference on Field-Programmable technology (ICFPT)*, pages 281–286. IEEE.

Phipps, C. and Davis., K. C. (2002). Automating data warehouse conceptual schema design and evaluation. In *4th International Workshop on Design and Management of Data Warehouses*, pages 23–32.

Piech, M. and Marcjan, R. (2018). A new approach to storing dynamic data in relational databases using json. *Computer Science*, 19.

Plantevit, M., Laurent, A., Laurent, D., Teisseire, M., and Choong, Y. W. (2010). Mining multidimensional and multilevel sequential patterns. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 4(1):1–37.

Pottinger, R. A. and Bernstein, P. A. (2003). Merging models based on given correspondences. In *Proceedings 2003 VLDB Conference*, pages 862–873. Elsevier.

Pujolle, G., Ravat, F., Teste, O., Tournier, R., and Zurfluh, G. (2011). Multidimensional database design from document-centric xml documents. In *International Conference on Data Warehousing and Knowledge Discovery*, pages 51–65. Springer.

Qi, Z., Wang, H., Li, J., and Gao, H. (2018). Frog: Inference from knowledge base for missing value imputation. *Knowledge-Based Systems*, 145:77–90.

Qi, Z., Wang, H., Meng, F., Li, J., and Gao, H. (2017). Capture missing values with inference on knowledge base. In *International Conference on Database Systems for Advanced Applications*, pages 185–194. Springer.

Qin, Y., Zhang, S., Zhu, X., Zhang, J., and Zhang, C. (2007). Semi-parametric optimization for missing data imputation. *Applied Intelligence*, 27(1):79–88.

Quix, C., Kensche, D., and Li, X. (2007). Generic schema merging. In *Advanced Information Systems Engineering*, pages 127–141.

Rahm, E. and Bernstein, P. A. (2001). A survey of approaches to automatic schema matching. *the VLDB Journal*, 10(4):334–350.

Raj, R., Wong, S. H., and Beaumont, A. J. (2016). Business intelligence solution for an sme: A case study.

Ravat, F., Teste, O., Tournier, R., and Zurfluh, G. (2008a). Algebraic and graphic lan-

guages for olap manipulations. *Inter. J. of Data Warehousing and Mining*, 4:17–46.

Ravat, F., Teste, O., Tournier, R., and Zurfluh, G. (2008b). Top_keyword: An aggregation function for textual document olap. In *Data Warehousing and Knowledge Discovery*, pages 55–64.

Ravat, F., Teste, O., Tournier, R., and Zurfluh, G. (2009). Designing and implementing olap systems from xml documents. In *New Trends in Data Warehousing and Data Analysis*, pages 1–21. Springer.

Ravat, F. and Zhao, Y. (2019). Data lakes: Trends and perspectives. In *Database and Expert Systems Applications*, pages 304–313.

Riazati, D. and Thom, J. A. (2011). Matching star schemas. In *International Conference on Database and Expert Systems Applications*, pages 428–438. Springer.

Roman, D., Dimitrov, M., Nikolov, N., Putlier, A., Sukhobok, D., Elvesæter, B., Berre, A., Ye, X., Simov, A., and Petkov, Y. (2016). Datagraft: Simplifying open data publishing. In *European Semantic Web Conference*, pages 101–106. Springer.

Romero, O. and Abelló, A. (2007). Automating multidimensional design from ontologies. In *Proceedings of the ACM tenth international workshop on Data warehousing and OLAP*, pages 1–8.

Romero, O. and Abelló, A. (2009). A survey of multidimensional modeling methodologies. *International Journal of Data Warehousing and Mining*, 5(2):1–23.

Romero, O. and Abelló, A. (2010). A framework for multidimensional design of data warehouses from ontologies. *Data & Knowledge Engineering*, 69(11):1138–1157.

Samat, N. A. and Salleh, M. N. M. (2016). A study of data imputation using fuzzy c-means with particle swarm optimization. In *International Conference on Soft Computing and Data Mining*, pages 91–100. Springer.

Sangeetha, M. and Senthil Kumaran, M. (2020). Deep learning-based data imputation on time-variant data using recurrent neural network. *Soft Computing*, 24(17):13369–13380.

Sanprasit, N., Jampachaisri, K., Titijaroonroj, T., and Kesorn, K. (2021). Intelligent approach to automated star-schema construction using a knowledge base. *Expert Systems with Applications*, 182:115 – 226.

Santoso, L. W. et al. (2017). Data warehouse with big data technology for higher education. *Procedia Computer Science*, 124:93–99.

Sautot, L., Faivre, B., Journaux, L., and Molin, P. (2015). The hierarchical agglomerative clustering with gower index: A methodology for automatic design of olap cube in ecological data processing context. *Ecological Informatics*, 26:217–230.

Schafer, J. L. and Graham, J. W. (2002). Missing data: our view of the state of the art. *Psychological methods*, 7(2):147.

Schneider, T. (2001). Analysis of incomplete climate data: Estimation of mean values and covariance matrices and imputation of missing values. *Journal of climate*, 14(5):853–871.

Sefidian, A. M. and Daneshpour, N. (2019). Missing value imputation using a novel grey based fuzzy c-means, mutual information based feature selection, and regression model. *Expert Systems with Applications*, 115:68–94.

Sen, P. C., Hajra, M., and Ghosh, M. (2020). Supervised classification algorithms in machine learning: A survey and review. In *Emerging Technology in Modelling and Graphics*, pages 99–111.

Sentas, P. and Angelis, L. (2006). Categorical missing data imputation for software cost estimation by multinomial logistic regression. *Journal of Systems and Software*, 79(3):404–414.

Shen, J.-J., Chang, C.-C., and Li, Y.-C. (2007). Combined association rules for dealing with missing values. *Journal of Information Science*, 33(4):468–480.

Skiena, S. S. (2008). The algorithm design manual. *The Algorithm Design Manual:*.

Song, S. and Sun, Y. (2020). Imputing various incomplete attributes via distance likelihood maximization. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 535–545.

Spoth, W., Kennedy, O., Lu, Y., Hammerschmidt, B., and Liu, Z. H. (2021). Reducing ambiguity in json schema discovery. In *Proceedings of the 2021 International Conference on Management of Data*, pages 1732–1744.

Tang, F. and Ishwaran, H. (2017). Random forest missing data algorithms. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 10(6):363–377.

Tang, Y., Wang, H., Zhang, S., Zhang, H., and Shi, R. (2017). Efficient web-based data imputation with graph model. In *International Conference on Database Systems for Advanced Applications*, pages 213–226. Springer.

YANG, Y., Abdelhédi, F., Darmont, J., Ravat, F., and Teste, O. (2021a). Internal data imputation in data warehouse dimensions. In *DEXA*, pages 237–244.

YANG, Y., Abdelhédi, F., Darmont, J., Ravat, F., and Teste, O. (2022a). Automatic machine learning-based olap measure detection for tabular data. In *International Conference on Big Data Analytics and Knowledge Discovery*, pages 173–188. Springer.

YANG, Y., Darmont, J., Ravat, F., and Teste, O. (2020). Automatic Integration Is-

sues of Tabular Data for On-Line Analysis Processing. In *16e journées EDA Business Intelligence & Big Data (EDA 2020)*, volume B-16, pages 5–18.

YANG, Y., Darmont, J., Ravat, F., and Teste, O. (2021b). An automatic schema-instance approach for merging multidimensional data warehouses. In *25th International Database Engineering & Applications Symposium*, pages 232–241.

YANG, Y., Darmont, J., Ravat, F., and Teste, O. (2022b). Dimensional data knn-based imputation. In *European Conference on Advances in Databases and Information Systems*, pages 315–329. Springer.

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

Troyanskaya, O., Cantor, M., Sherlock, G., Brown, P., Hastie, T., Tibshirani, R., Botstein, D., and Altman, R. B. (2001). Missing value estimation methods for DNA microarrays. *Bioinformatics*, 17(6):520–525.

Trujillo, J., Palomar, M., Gomez, J., and Song, I.-Y. (2001). Designing data warehouses with oo conceptual models. *Computer*, 34(12):66–75.

Tutunea, M. F. and Rus, R. V. (2012). Business intelligence solutions for sme's. *Procedia economics and finance*, 3:865–870.

Twala, B. (2009). An empirical comparison of techniques for handling incomplete data using decision trees. *Applied Artificial Intelligence*, 23(5):373–405.

Twala, B., Cartwright, M., and Shepperd, M. (2005). Comparison of various methods for handling incomplete data in software engineering databases. In *2005 International Symposium on Empirical Software Engineering, 2005.*, pages 10–pp. IEEE.

Ullman, J. D. (1983). *Principles of database systems*. Galgotia publications.

Usman, M., Asghar, S., and Fong, S. (2010). Data mining and automatic olap schema generation. In *2010 Fifth International Conference on Digital Information Management (ICDIM)*, pages 35–43. IEEE.

Usman, M., Pears, R., and Fong, A. C. M. (2013). A data mining approach to knowledge discovery from multidimensional cube structures. *Knowledge-Based Systems*, 40:36–49.

Verpoort, P., MacDonald, P., and Conduit, G. J. (2018). Materials data validation and imputation with an artificial neural network. *Computational Materials Science*, 147:176–185.

Vrdoljak, B., Banek, M., and Rizzi, S. (2003). Designing web warehouses from xml schemas. In *International Conference on Data Warehousing and Knowledge Discovery*, pages 89–98.

Wang, H.-Z., Qi, Z.-X., Shi, R.-X., Li, J.-Z., and Gao, H. (2017). Cosset+: Crowdsourced missing value imputation optimized by knowledge base. *Journal of Computer Science and Technology*, 32(5):845–857.

Wang, J. and Dong, Y. (2020). Measurement of text similarity: a survey. *Information*, 11(9):421.

Wang, X., Li, A., Jiang, Z., and Feng, H. (2006). Missing value estimation for dna microarray gene expression data by support vector regression imputation and orthogonal coding scheme. *BMC bioinformatics*, 7(1):1–10.

Wang, Z., Dong, H., Jia, R., Li, J., Fu, Z., Han, S., and Zhang, D. (2021). Tuta: Tree-based transformers for generally structured table pre-training. In *27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, page 1780–1790.

Watson, H. J. and Wixom, B. H. (2007). The current state of business intelligence. *Computer*, 40(9):96–99.

Wijsen, J. (2005). Database repairing using updates. *ACM Transactions on Database Systems (TODS)*, 30(3):722–768.

Wu, C.-H., Wun, C.-H., and Chou, H.-J. (2004). Using association rules for completing missing data. In *Fourth International Conference on Hybrid Intelligent Systems (HIS'04)*, pages 236–241. IEEE.

Wu, J., Song, Q., and Shen, J. (2007). An novel association rule mining based missing nominal data imputation method. In *Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD 2007)*, volume 3, pages 244–249. IEEE.

Wu, S., Feng, X., Han, Y., and Wang, Q. (2012). Missing categorical data imputation approach based on similarity. In *2012 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 2827–2832. IEEE.

Wu, X. and Barbará, D. (2002). Modeling and imputation of large incomplete multidimensional datasets. In *DaWak*, pages 286–295.

Xia, J., Zhang, S., Cai, G., Li, L., Pan, Q., Yan, J., and Ning, G. (2017). Adjusted weight voting algorithm for random forests in handling missing values. *Pattern Recognition*, 69:52–60.

Yan, X., Xiong, W., Hu, L., Wang, F., and Zhao, K. (2015). Missing value imputation based on gaussian mixture model for the internet of things. *Mathematical Problems in Engineering*, 2015.

Ye, C. and Wang, H. (2014). Capture missing values based on crowdsourcing. In *International Conference on Wireless Algorithms, Systems, and Applications*, pages 783–792.

Springer.

Ye, C., Wang, H., Lu, W., and Li, J. (2020). Effective bayesian-network-based missing value imputation enhanced by crowdsourcing. *Knowledge-Based Systems*, 190:105199.

Yujian, L. and Bo, L. (2007). A normalized levenshtein distance metric. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(6):1091–1095.

Zhang, A., Song, S., Sun, Y., and Wang, J. (2019). Learning individual models for imputation. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 160–171. IEEE.

Zhang, S. (2012). Nearest neighbor selection for iteratively knn imputation. *Journal of Systems and Software*, 85(11):2541–2552.

Zhang, S., Zhang, J., Zhu, X., Qin, Y., and Zhang, C. (2008). Missing value imputation based on data clustering. In *Transactions on computational science I*, pages 128–138. Springer.

Zhang, W., Yang, Y., and Wang, Q. (2011). Handling missing data in software effort prediction with naive bayes and em algorithm. In *Proceedings of the 7th International Conference on Predictive Models in Software Engineering*, pages 1–10.

Zhang, Y., Cao, T., Li, S., Tian, X., Yuan, L., Jia, H., and Vasilakos, A. V. (2016). Parallel processing systems for big data: a survey. *Proceedings of the IEEE*, 104(11):2114–2136.

Zhang, Y. and Liu, Y. (2009). Data imputation using least squares support vector machines in urban arterial streets. *IEEE Signal Processing Letters*, 16(5):414–417.

Zhao, Q. and Bhowmick, S. S. (2003). Association rule mining: A survey. *Nanyang Technological University, Singapore*, 135.

Zhuang, Y., Ke, R., and Wang, Y. (2019). Innovative method for traffic data imputation based on convolutional neural network. *IET Intelligent Transport Systems*, 13(4):605–613.