A Pattern Model and Algebra for Representing and Querying Relative Information Completenes

Fatma-Zohra Hannou

Sorbonne Université

Laboratoire Informatique Paris 6

THÈSE DE DOCTORAT

DISCIPLINE: INFORMATIQUE

A Pattern Model and Algebra for Representing and Querying Relative Information Completeness

par Fatma-Zohra HANNOU

Rapporteurs:	Nicole BIDOIT-TOLLU	Professeur, Université Paris-Sud
	Dimitris KOTZINOS	Professeur, Université Cergy Pontoise
Examinateurs:	Ladjel BELLATRECHE	Professeur, ENSMA, Poitiers
	Laure BERTI-EQUILLE	Professeur, Université Aix-Marseille
	Christophe MARSALA	Professeur, Sorbonne Université
Directeur de thèse:	Bernd AMANN	Professeur, Sorbonne Université
Encadrant de thèse:	Mohamed-Amine BAAZIZI	MdC, Sorbonne Université

”Have no fear of perfection, you will never reach it.”

Marie Skłodowska-Curie

Nobel Prizes in Physics and Chemistry

Abstract

Information incompleteness is a major data quality issue which is amplified by the increasing amount of data collected from unreliable sources. Assessing the completeness of data is crucial for determining the quality of the data itself, but also for verifying the validity of query answers over incomplete data. While there exists an important amount of work on modeling data completeness, deriving this completeness information has not received much attention. In this work, we tackle the issue of extracting and reasoning about complete and missing information under *relative information completeness* setting. Under this setting, the completeness of a dataset is assessed with respect to a complete reference dataset. Few works have been dedicated to representing data completeness under this setting, and we advance the field by proposing two contributions: a *pattern model* for providing minimal covers summarizing the extent of complete and missing data partitions and a *pattern algebra* for deriving minimal pattern covers for query answers to analyze their validity.

The completeness pattern framework presents an intriguing opportunity to achieve many applications, particularly those aiming at improving the quality of tasks impacted by missing data. In our work, we address the problem of repairing query results obtained from incomplete data. Data imputation is a well-known technique for repairing missing data values but can incur a prohibitive cost when applied to large data sets. Query-driven imputation offers a better alternative as it allows for fixing only the data that is relevant for a query. We adopt a rule-based query rewriting technique for imputing the answers of analytic queries that are missing or suffer from incorrectness due to data incompleteness. We present a novel query rewriting mechanism that is guided by the completeness pattern model and algebra. Our solution strives to infer the broadest possible set of missing answers while improving the precision of incorrect ones.

In the last contribution, we investigate the generalization of our pattern model for summarizing any data fragments. The generalized pattern model can be used to produce pattern summaries of data fragments over any subset of attributes and these summaries can be queried to analyze and compare data fragments in a synthetic and flexible way.

Keywords: Relative Information, Completeness Assessment, Pattern model, Pattern Algebra, Imputation, Summarization

Résumé

L'incomplétude des données est un problème majeur de qualité qui s'amplifie par la quantité croissante de données collectées par des sources peu fiables. L'évaluation de l'exhaustivité des données est cruciale pour déterminer leur qualité mais aussi la validité des réponses de requêtes qui en découlent. Dans le contexte de l'information relative, la complétude d'une base de données est évaluée en comparaison à une base référence. Nous apportons deux principales contributions à ce domaine: un modèle de motifs produisant des couvertures minimales résumant l'étendue des partitions de données complètes et manquantes, ainsi qu'une algèbre de motifs permettant de dériver des couvertures minimales pour l'analyse de la validité des réponses des requêtes.

Ce modèle de motifs offre une opportunité intéressante pour réaliser de nombreuses applications, en particulier celles visant à améliorer la qualité des tâches affectées par les données manquantes. Nous adoptons une technique de réécriture de requêtes à base de règles pour imputer les réponses des requêtes d'agrégation manquantes ou présentant des valeurs incorrectes.

Nous étudions également la généralisation de notre modèle de motifs pour effectuer la synthèse des fragments de données. Les résumés peuvent être interrogés pour analyser et comparer les fragments de données de manière synthétique et flexible.

Mots Clés: Information Relative, Complétude de données, Modèle de motifs, Algèbre de motifs, Imputation, Synthétisation

Dédicaces

À **Nour-Filastine**,

Pour toutes les fois où je lisais en te berçant,
Où je réfléchissais en te nourrissant,
Où j'écrivais en te parlant,
Pour toutes les fois où cette thèse a partagé nos petits moments,
Pour toutes les fois où je t'ai demandé de m'aider sans que tu comprennes,
Où sur ce document tu as tenté d'écrire sans que tu y parviennes.

Après dieu, qui remercier si ce n'est toi, tu m'as appris le sens de la force, l'espoir, la détermination, l'amour... Peu de gens peuvent comprendre, mais tu m'as fait sourire lors de mes échecs, et pleurer de bonheur avec ton seul sourire...
Les matins parisiens glacials, le son de ta poussette, tes larmes la nuit, la fièvre, les dents, tes pas, tes mots...
Je pourrais écrire un livre juste pour te dire merci, pour tout ce que tu m'as apporté, mais peu de mots ressembleront à ce que je ressens.

Ma chère fille,
Depuis que tu es née, chaque jour, c'est pour toi que je m'interdisais de lâcher,
Pour toi que je refusais d'abandonner,
C'est pour toi ma chérie que maman a continuée,
Pour qu'un jour tu saches que dans la vie, on ne laisse jamais tomber,
Dans la vie il faut sans cesse repousser ses limites,
Aujourd'hui, toutes les deux, nous l'avons fait....

Comme pour mes jours et mes nuits... je te dédie cette thèse
Maman

Remerciements

الْحَمْدُ لِلهِ الّذِي بِنِعْمَتِهِ تَتِمُّ الصَّالِحَاتُ

J'exprime mon profond respect et ma parfaite gratitude envers mon directeur de thèse, **Bernd Amann**. D'abord, pour la qualité de ses conseils et directives scientifiques, ainsi que sa disponibilité malgré toutes ses responsabilités. Bernd n'a jamais économisé son temps pour me prodiguer ses précieux conseils, et débattre de mes idées. Au-delà de son expertise, je reste impressionnée par sa qualité d'écoute et sa capacité à mettre ses étudiants en confiance pour défendre leurs propositions. Je le remercie pour ses encouragements et sa confiance en moi, pendant toutes les périodes parfois difficiles de ma thèse. Bernd a toujours su me motiver, me pousser au-delà de mes limites, me rassurer quand la science se compliquait, et partager ma joie quand elle me souriait.
Qu'il trouve ici l'expression de ma profonde reconnaissance.

Je remercie également mon encadrant de thèse, **Mohamed-Amine Baazizi**, pour ses conseils scientifiques, ses relectures soignées, et sa participation aux travaux de recherche.

Je remercie les professeurs **Nicole Bidoit-Tollu** et **Dimitris-Kotzinos** d'avoir accepté d'être les rapporteurs de mon manuscrit, et pour la qualité de leurs remarques malgré des délais serrés. Je remercie également tous les membres du Jury, pour m'avoir fait l'honneur d'assister à ma soutenance afin d'évaluer mon travail et me faire profiter de leur grande expertise.

Je remercie l'équipe du projet **Ebita**, qui a permis de financer cette thèse, et ses membres qui ont participé à l'amorce de cette thèse, à enrichir le débat sur les problématiques scientifiques que j'ai traité.

J'ai eu la chance d'avoir passé mes années de thèses au sein de l'équipe Base de Données du laboratoire LIP6, dans un environnement propice à l'apprentissage. J'ai beaucoup appris en côtoyant les talentueux chercheurs de mon équipe que je remercie pour leur accueil et convivialité. Je garderai de ma thèse les agréables souvenirs de nos riches discussions sur la science et parfois la vie, avec ou sans caféine...
Merci à **Anne, Camélia, Hubert, Li Ke et Stéphane**.

Quoi que j'ai pu accomplir dans ma modeste vie, j'ai trouvé à mes côtés une famille aimante et supportrice qui m'a toujours donné l'envie de persévérer.

Je salue d'abord la mémoire de ma **grande-mère** qui m'a inculqué les plus belles valeurs, et donné les plus beaux exemples de courage. J'aurai aimé te dire que j'ai franchis une nouvelle étape, que dieu t'accueille dans son vaste paradis.

Papa
À mon héros, à celui qui m'a appris ce qu'aucune école ne m'a apprise, ni aucun livre ne m'a transmis,
Merci pour tes sacrifices, ton amour, ta sagesse et ton soutien,
Merci parce que tu m'as appris depuis mes premiers pas, comment grimper les plus hautes montagnes et viser toujours plus loin que le ciel...

Maman
À ma fidèle supportrice et ma meilleure amie, merci pour ton amour inconditionnel et ton soutien indéfectible.
Je sens tes prières avec chaque pas je mets en avant et tu me rassures plus que tout le monde dans mes moments de doutes.
Je resterai à jamais ta fille qui ne grandira qu'aux yeux des autres...

Ilhem, Meriem, Hadjer, Mouloud, Yasmine
Même séparés de centaines de kilomètres, votre joie de vivre, votre énergie débordante et votre amour m'éclairent tous les jours,
C'est vous que j'appelle toujours avant mes épreuves, et vous que j'appelle d'abord pour partager mes réussites,
Peu importe où la vie nous mènent, je porterai nos promesses enfantines comme une étoile au cou,
Ma fierté est sans égale...

Mohammed
Merci d'être mon pilier et mon repère,
Pour tout ce qu'on a partagé ensemble, et ce qu'il nous reste à bâtir,
Pour ton soutien, ta confiance, ton enthousiasme à l'égard de mes travaux,
Merci pour la fierté sincère qui brille dans ton regard,
Ensemble on ira aussi loin que nos rêves les plus fous...

Contents

1 Introduction and Motivation

> *If we knew what it was we were doing it would not be called research, would it?*
>
> **— Albert Einstein**
> Nobel Prize in Physics

Contents

1.1 General Context and Motivation

"The world's most valuable resource is no longer oil, but data." This statement from The Economist [Eco] fully demonstrates the importance of data in our society. Social web-applications and connected objects have changed our daily life and the fourth industrial revolution is transforming static production processes into dynamic and data-driven manufacturing workflows. Millions of users order food on the internet, do shopping on Amazon, ask Google to find an Italian restaurant near their place, and exchange messages on Facebook. Objects become smart, houses self-regulate their energy consumption, cars self-drive, and soon robots will make a medical diagnosis. Modern airplanes, like the "A380", are equipped with 25 000 sensors and generate almost 2.5TB of data per day [Man] for ensuring the aircraft maintenance.

This data revolution is supported by technological advances, new algorithms, and abundant data storage capabilities which enable the creation of new services, tools, and industries producing and consuming huge amounts of data. One major challenge in this context is to maximize the quality of the data. For example, IBM indicates that the loss of 3.1 trillion dollars per year in the USA [Har] can be mainly attributed to inaccurate, outdated, or incomplete data that do not fit specific task requirements. The report argues that data quality is one of the most significant obstacles for the development of a company, coming before material tools or human expertise. Despite the abundance of produced data, "missing data" is a frequent quality issue [Her+07], which emanates from multiple reasons: physical anomalies, database design, human errors, lack of sources, or privacy rules. Incomplete data problems generate several interesting research challenges concerning the representation and processing of missing information. Whereas many data models have been developed for representing any kind of complex data, the exact representation of missing information within these models is in general difficult. A first solution is to introduce placeholders to indicate missing information that should be filled in. This kind of placeholder has been introduced in the relational data model by E.F. Codd [Cod79] in the form of a "missing information" symbol $null$. Codd's $null$-"values" represent missing or unknown attribute values and remain the most frequently used representation for missing information in databases. A significant drawback of this solution is the difficulty to agree on a unique meaning of the $null$ symbol and its query semantics. For example, simple filtering conditions like $A = 3$ cannot be evaluated to true or false if A is a $null$ value (unknown). Another problem concerns aggregation functions, which produce incorrect results with $null$ values. These limitations led to the development of "stronger" representation systems for describing missing data more precisely and better understanding their influence on query results. For example, c-tables [Imi+88a] use "marked" nulls to describe missing values, which can be shared by different tuples attributes. The completeness assessment under this setting does not cover missing data tuples, which are considered as false (Closed World Assumption). The

Open World Assumption is a paradigm that supposes and accepts the existence of additional data tuples not included in the database (missing tuples), but only a few queries find complete answers due to the absence of knowledge about what is missing. A middle-ground assumption was first introduced in [Mot89] to provide a better theoretical foundation for describing missing tuples. The proposed model assumes the existence of a virtual database with a complete set of tuples, which can be compared against the available, incomplete database. Under this setting, known as the Partially Closed World Assumption, a high number of representation systems have been proposed to model data incompleteness. The concept of relative completeness has been proposed [Fan+10a]. Instead of a virtual reference database, relative completeness is defined with respect to a materialized reference dataset which allows for a more effective and precise quality assessment process [Fan15].

In this thesis, we adopt the relative completeness approach to address several challenges concerning the representation of incomplete information for annotating and repairing query answers. We introduce these challenges in the following section through a concrete application scenario.

1.2 EBITA and Smart Campus

This thesis has been financed by the EBITA project (2016-2018), a French-German research project, associating Sorbonne University to the german Fraunhofer Institute [Ebi]. EBITA was a two-year project which strived to explore database and machine learning research opportunities in various Smart IoT application domains like mobility, environment, energy consumption.

One use case of the project was a Smart Campus scenario for the Jussieu site of Sorbonne University. The Jussieu campus is equipped with a sensor network that measures multiple energy and environment indicators: temperature, pressure, electricity consumption (lighting, heating, power-supply), water consumption. These sensors continuously produce several measurements per hour, and a database is daily updated with the most recent values. The Jussieu campus counts 96 buildings, and sensors are variably distributed across these buildings. As part of the project, we had access to the data produced by 5,000 sensors, located in the buildings highlighted on the campus map (Figure 1.1). As it is shown in the map, buildings are situated between numbered towers and each building is identified by the numbers of the two towers it connects. For example, building 1323 connects tower 13 with tower 23.

Multiple other data sources on room occupation, meeting rooms planning, localities areas *etc*. have also been gathered for enriching the raw sensor data. Table 1.1 shows some general statistics about the campus locations and sensors.

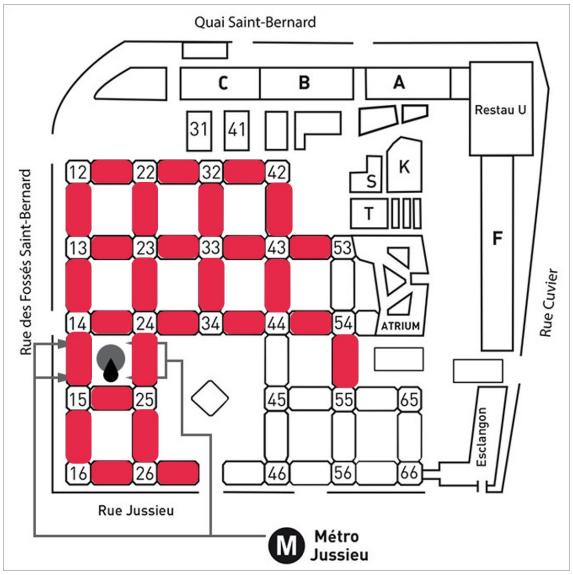


Fig. 1.1.: Campus map coverage by sensors

Within the Smart Campus scenario, the EBITA project aimed to develop a decision-aid application that integrates different available data sources to fulfill the following tasks:

- Creation of analysis reports on the sensor network.
- Spatial and temporal visualization of statistical indicators and their financial and environmental cost.
- Identification of location-based resource consumption profiles to build targeted energy reduction strategies.
- Analyzing and explaining resource consumption variations using contextual metadata like campus events, meteorological data, room occupation etc.

The short-term goal was to develop a decision support system for managers responsible for establishing energy optimization strategies. In a longer term, the application was planned to evolve towards a full Smart Campus system including automating energy optimization strategies by extending the existing sensor network with actuator network. A screen-shot of a web application we developed as a first milestone of the system is illustrated in Figure 1.2.

First experiments on real sensor data allowed us to identify various data quality problems and their significant impact on the Smart Campus analysis tasks. Indeed, our experiments confirmed that raw sensor data suffers from multiple quality problems as syntactic errors, schema encoding issues, missing or outlier data. Missing data was a particular problem for the generation of analytic reports with aggregate queries. Aggregated results were incorrect, and few reliable decisions could

Location Type	Number of units
Buildings	96
Floors	635
Rooms	10 755

Location statistics

Room Occupation	Number of units
Administration	3562
Researcher office	1053
Teaching	615
Meeting	221

Major rooms occupation activities

Sensor type	Number	Number of measures (year 2015)
Temperature	665	8 798 715
Electrical counter	456	4 035 398
water	118	707 715
pressure	48	70 465

Some Sensors types numbers and measures size

Tab. 1.1.: Jussieu campus data general statistics

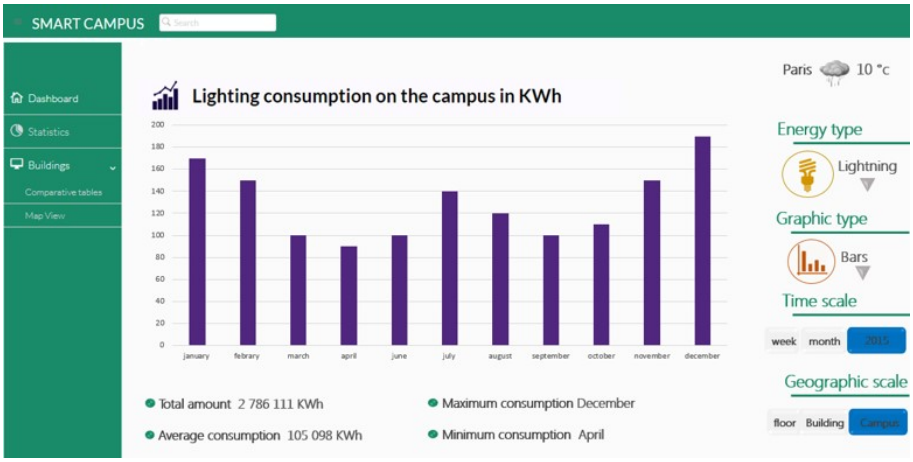


Fig. 1.2.: An overview of the smart campus user interface

be made. Starting from these general observations, the following section introduces the scientific challenges and our contributions through a simple practical usage scenario.

1.3 Challenges by Example

In this section, we introduce the research problems and our contributions using a example scenario inspired by the Smart Campus scenario. Anna is a data analyst at the university in charge of resource consumption monitoring. Her tasks are to interpret sensor data series, identify trends and particular events, and create analytic reports. She regularly acquires data from the sensor network and other related sources to maintain a database system feeding reporting services. The quality of data is a central concern in data analysis, and Anna wants to associate data quality metrics

to her reports, to avoid wrong interpretations. She wants to take advantage of the multiple available data sources to annotate sensor measures with completeness and missing data information.

1.3.1 Challenge #1: Complete and Missing Data Representation

In order to identify missing sensor data fragments, Anna thinks about formulating queries comparing available sensor measures with external *reference* datasets like maps and calendars. Anna repeats this operation manually for various locations, time intervals, energy types, etc., to generate various reports and visualizations. The first problem that arises then is to build synthetic and useful descriptions of *complete* and *missing* data extents. These descriptions should be rich enough to accomplish complex quality analysis and compact enough to achieve a fast and straightforward interpretation. Given a data table $Elec$ that contains all Electricity measures for the floor 5 in the building 2526, we illustrate in Tables 1.2 and 1.3 an overview of possible representation summarizing respectively all available and missing measures fragments. Table 1.2

Building	Floor	Room	Day
2526	5	1	$*$
2526	5	3	Monday
2526	5	3	Tuesday
2526	5	3	Thursday
2526	5	3	Friday
2526	5	4	$*$
2526	5	5	$*$

Tab. 1.2.: Available data for building 2526

describes all measures that are available in $Elec$ with respect to some reference map and calendar. Each tuple or *pattern* characterizes a complete data fragment. For example, the data table contains the all measures regarding room 1. For the 3rd room, this is only true for Monday, Tuesday, Thursday and Friday.

Building	Floor	Room	Day
2526	5	2	$*$
2526	5	3	Wednesday

Tab. 1.3.: Missing data for building 2526

Table 1.3 represents the "complement" of Table 1.2 and summarizes the missing data extents. For example, we can see that that measure is recorded for room 2.

1.3.2 Challenge #2: Query Result Annotation

Anna now wants to understand how the missing data impacts the result of certain queries. Take the example of the previously described $Elec$ table. The corresponding time series is illustrated in Figure 1.3. Anna observes significant consumption variations between different rooms and defines the following query that normalizes the electricity consumption with respect to the room surface (table Area in Figure 1.4):

```
SELECT Building B,Floor F, Room R, Day D, KwH/area
FROM Elec E JOIN Area A
ON E.B=A.B and E.F=A.F and E.R=A.R
```

Listing 1.1: Query Q_{norm}

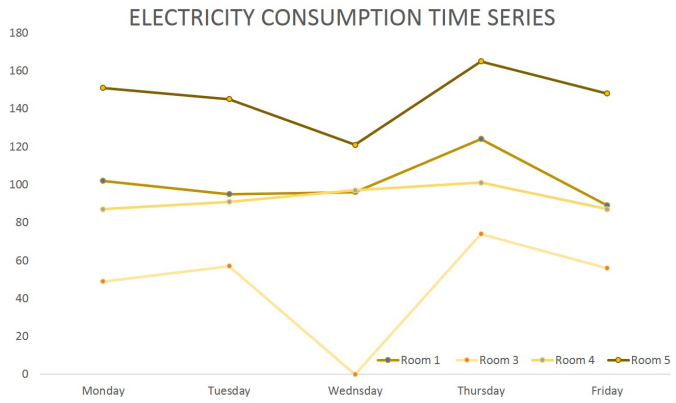


Fig. 1.3.: Electricity consumption evolution: raw time series

Building	Floor	Room	m^2
2526	5	1	15
2526	5	2	10
2526	5	4	15
2526	5	5	20

Area Table

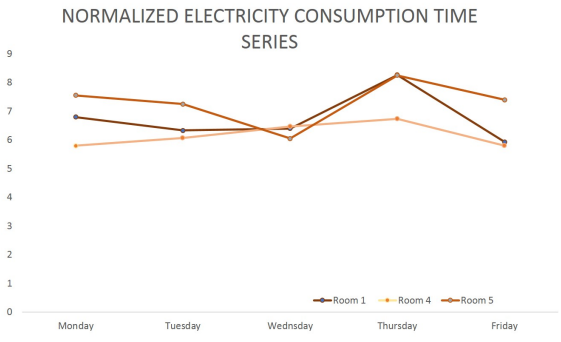


Fig. 1.4.: Area table and normalized electricity consumption times series

The right figure in Figure 1.4 represents the new normalized time series. Observe that the two tables join on locality attributes (building, floor, room). If a room does not occur in table $Elec$ or

Building	Floor	Room	Day
2526	5	5	*
2526	5	4	*
2526	5	1	*

Building	Floor	Room	Day
2526	5	2	*
2526	5	3	*

Tab. 1.4.: Complete and missing data representations for Q_{norm} result

$Area$, the query obviously cannot compute its normalized consumption value. Based on the extent of complete and missing partitions in both tables, we can derive the complete and missing patterns as illustrated in Table 1.4. For example, we can see that the generated result is complete for rooms 1, 4, and 5 of floor 5 and misses all measures of room 2 and 3.

Anna can generate such annotations for analyzing the other query results combining sensor data with room occupation data, event planning information, etc. One issue in this use case is to define an efficient solution which allows annotating query results in an interactive setting.

1.3.3 Challenge #3: Aggregate Queries Correctness

Aggregate queries are frequently used in sensor networks to generate reports, which aggregate measures at different granularity levels. Missing data in raw tables lead to incorrect query results. Indeed, some results might be missing for empty partitions, but an incomplete data table might also produce incorrect results for partially complete partitions. Similarly to the previous join queries, Anna expects to obtain annotations for missing and incorrect results. Take the example of the following query, which returns the total electricity consumption amount for each room:

```
SELECT Building B,Floor F, Room R, Sum(KwH)
FROM Elec E
GROUP BY Building, Floor, Room
```

Listing 1.2: Query Q_{agg}

Figure 1.5 shows a primary bar graph representing the annotated result of the query Q_{agg}: the result of rooms 1, 4 and 5 are correct, the result of room 2 is missing and the result of room three is incorrect.

Table 1.5 shows the complete and missing data annotation for the query Q_{agg} result. We can see that room 2 only appears in the missing partition pattern table, whereas room 3 appears in both tables. The patterns that belongs to both, the complete and missing pattern tables, correspond to partially complete partitions and lead to *incorrect* aggregated results.

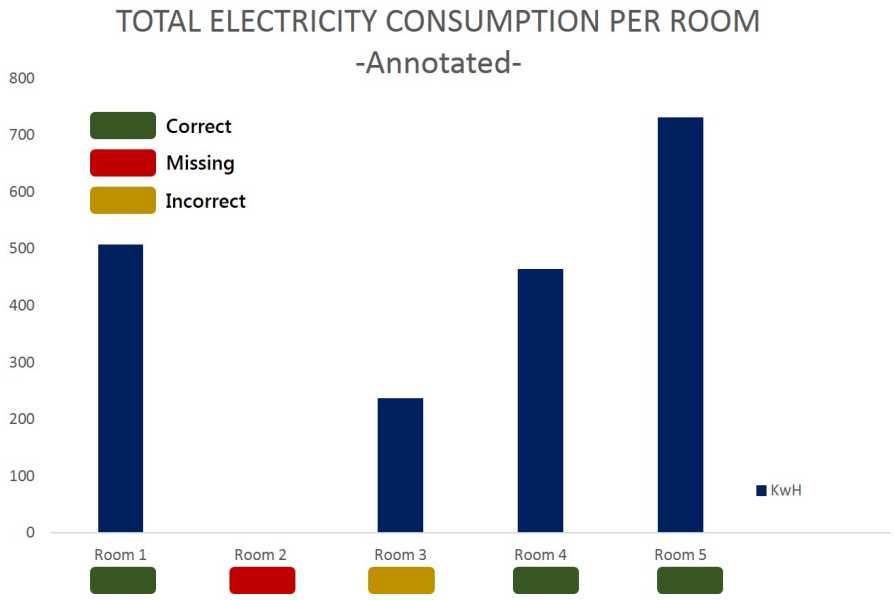


Fig. 1.5.: Annotated total electricity consumption per room

Building	Floor	Room
2526	5	1
2526	5	3
2526	5	4
2526	5	5

Building	Floor	Room
2526	5	2
2526	5	3

Tab. 1.5.: Complete and missing data representations for the query Q_{agg} result

1.3.4 Challenge #4: Aggregate Query Imputation

During the years, Anna has obtained substantial knowledge and expertise about the resource consumption on the campus. She also has access to various metadata tables, allowing for a better understanding of consumption patterns and trends. She wants to use this knowledge to repair her aggregated results and to provide correct visualizations.

Expert rules are a well-known technique for data imputation that joins human domain expertise and an automatic repairing procedure. Consider again query Q_{agg} in Listing 1.2. Anna estimates from historical time series that room 3 has a median consumption profile compared to other rooms in the floor and that Room 2 in building 2526 is a file storage room as the same room of the building 2600 with similar electricity consumption profiles. Anna formulates these two observations by two imputation rules as shown in Table 1.6. Rule r_1 simply copies the value of room 2 in building 2600 to estimate the value of room 2 in building 2526. Rule r_2 estimates the value of room 3 in any building and floor (b, f and r are variables) by the average correct values of the other rooms in the same building and floor.

	Imputation Rule		
r_1	(Building: 2526, Floor: 5, Room: 2), kWh	$\leftarrow$	(Building: 2600, Floor: 5, Room: 2), kWh
r_2	(Building: b, Floor: f, Room: 3), kWh	$\leftarrow$	(Building: b, Floor: f, Room: r), Avg(kWh)

Tab. 1.6.: Imputation rules examples

By using the solution of Challenge 3, it is possible to build an inference mechanism which identifies all incorrect and missing query results, chooses the minimal set of rules that can be applied to repair these results and instantiates these rules by applying them to maximal set of correct answers. For example, the system will automatically apply rule r_1 to repair the missing answer of room 2 and rule r_3 to repair room 3 for producing the repaired query result shown in Figure 1.6.

Aggregate queries may apply to large datasets, and their results are considerably reduced compared to the initial data tables size. Repairing missing data measures observed in raw sensing output can become expensive, and covers data tuples not required for reporting queries. A query-driven strategy intends to correct query results by estimating their results using expert rules. We tackle the problem of extending the representation system for allowing the expression of cleaning tools as the imputation rules previously described. This approach should operate at the *query answer level*, for a query-driven repair.

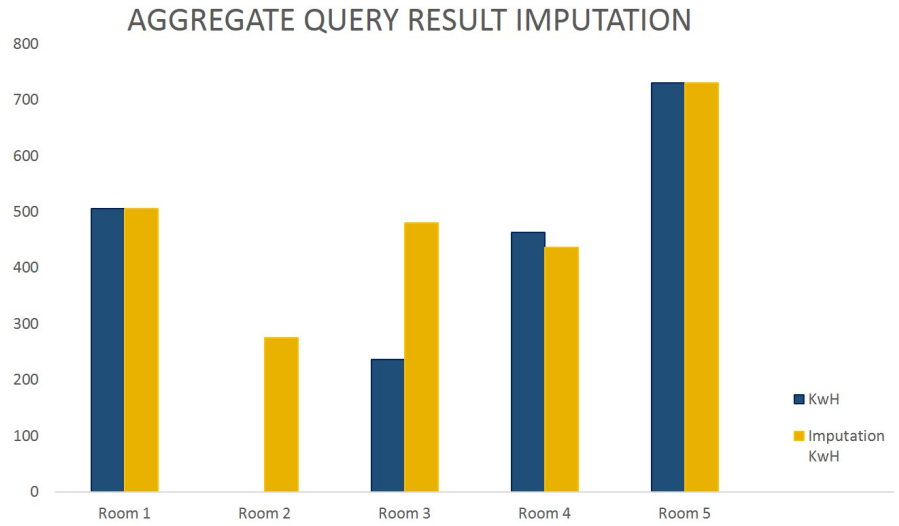


Fig. 1.6.: Repaired aggregate query Q_agg results

1.3.5 Challenge #5: Data Fragments Summarization

An important task of a Smart Campus system consists in establishing an resource consumption profiles for analyzing and deploying more targeted energy optimization strategies. Analogously to complete and missing data summarization, Anna wants to take advantage of the representation system to describe energy profiles in a compact way. For this purpose, she integrates metadata about factors that might impact energy consumption: geographic sector, rooms activity, area, equipment types, etc. For example, Anna might create locality-centric profiles by fixing a consumption threshold for separating locations with high from locations with low consumption. Two summaries for these two profiles are illustrated in Table 1.7.

Sector	Building	Floor	Room	Occupation	Area
North-West	∗	∗	∗	∗	∗
∗	∗	∗	∗	Server	∗
∗	∗	∗	∗	TP classroom	∗

High consumption profile summary

Sector	Building	Floor	Room	Occupation	Area
∗	3242	∗	∗	∗	∗
South	∗	JU	∗	∗	∗
∗	∗	∗	∗	∗	6

Low consumption profile summary

Tab. 1.7.: Building energy profiles summaries

Summaries exhaustively characterize their data fragments and might serve for various tasks. A summary might first have an explanatory use since it allows, for example, establishing the reasons underlying high consumption profiles. For example, Table 1.7 shows that all Server and TP class rooms have a high energy consumption. Summaries also can help in deploying predictive models by identifying particular factors for monitoring like Occupation in the previous example.

1.4 Thesis Contributions

In this section, we summarize our scientific contribution for solving the challenges presented in Section 1.3.

Contribution 1: Pattern-Based Model for Completeness Representation: Our first contribution is a pattern-based model for representing complete and empty data partitions. Our model follows the Partially Closed World Assumption applying the relative information completeness approach, which assumes the existence of materialized reference datasets. Our model introduces pattern tables as shown in Section 1.3 for precisely summarizing available and missing data partitions. Compared to other approaches, the existence of materialized reference datasets increases expressiveness and gives us the possibility to describe more precisely the quality issues generated by missing data for producing query results.

Contribution 2: Pattern Algebra: As a part of the pattern representation framework we extend the relational algebra, with two operators *folding* and *unfolding* for generating and transforming pattern tables. These two operators are central to our model since they provide the "procedural" semantics for implementing our pattern model. In particular, folding allows to generate minimal pattern covers for complete data fragments and unfolding transforms patterns back to raw data tuples. Based on this extended relational algebra, we can express complex pattern queries generating pattern tables for analyzing data completeness and annotating query results. We prove the correctness and soundness of the pattern algebra and show how pattern queries can be optimized and translated into standard SQL queries. The performance of the obtained framework is experimentally evaluated over real-world and synthetic datasets.

Contribution 3: Folding Algorithms: We propose two algorithms for implementing the folding operator defined in our algebra. The first algorithm efficiently computes minimal pattern covers for data tables. The second algorithm directly operates on pattern tables to create minimal sets. We prove the correctness of both proposed algorithms and check their effectiveness and efficiency through experiments on real-world and synthetic datasets.

Contribution 4: Rule-Based Query Result Imputation: We propose an imputation rule model that aims at repairing aggregate query results. The imputation rules leverage our pattern model to achieve query-driven data imputation for aggregate queries. Most of the existing data imputation models operate at the raw data level and do not consider queries for refining the imputation process. We take advantage of the pattern representation model to achieve a complete imputation process, starting with identifying correct, incorrect, and missing query results. Imputation rules are transformed into imputation pattern algebra expressions and translated into optimized SQL queries.

Contribution 5: Fragment Summarization and Reasoning: Finally, we show how we can generalize our completeness pattern model for data summarization. We extend the pattern model to characterize any data fragments. We also introduce a general framework for explaining and comparing fragments using pattern-based fragment summaries.

Publications Our contribution on query-driven answer imputation for aggregate queries will be published in [Han+19c] and our work on exploring and comparing table fragments with fragment summaries is published in[Han+19b]. The work on explaining query answer completeness and correctness with partition patterns will appear in [Han+19a].

1.5 Thesis Outline

The thesis is organized into three parts, each part tackling a subset of the challenges described in Section 1.3:

Part I covers contributions 1, 2, and 3 on relative information completeness representation. The first Chapter 2 surveys the state of the art on data quality and on incomplete data representation models. The remaining chapters summarize our contributions on the pattern model and algebra (Chapter 3) and its implementation and experimental evaluation (Chapter 4).

Part II covers contribution 4 on aggregate query results imputation. We first survey the state-of-the-art on data imputation techniques in Chapter 5. We then formally define our rule-based imputation mechanism to address the problem of repairing analytic query results over incomplete data (Chapter 6).

Part III presents contribution 5; a generalization of the pattern model for reasoning on data fragments.

Part IV presents conclusion and future work.

Part I

Relative Completeness Representation

Data Completeness Representation

2

> *I was taught that the way of progress was neither swift nor easy.*
>
> **— Marie Sklodowska-Curie**
> Nobel Prizes in Physics and Chemistry

Contents

2.1 Introduction

The problem of data incompleteness in databases has been addressed from different perspectives. Several theoretical models have tried to define representation systems with strong expressivity compared to the commonly used Null values. In this thesis, we are interested in approaches that extend the relational model for assessing the completeness of data tables and query results. This chapter is state of the art for the first part of this manuscript, that introduces general quality and completeness concepts, and surveys existing systems with the following structure:

- Section 2.2 gives a general introduction of data quality issues in databases. We review the major taxonomies proposed in the literature and quality dimensions definitions.

- Section 2.3 discusses categories of existing contributions treating data incompleteness, following their purpose: missing data identification, query result assessment, and explanation, repair techniques (Subsection 2.3). We mainly focus on completeness representation systems that allow providing an assessment tool using annotations. The most related works to our research problem are discussed in Subsection2.4.2.

- As a Summary, we draw in Section 2.5 the positioning of our research work compared to the discussed models.

2.2 Data Quality

There is a general agreement for defining the quality of data from the user perspective. In all data-centric applications, regardless of their context, we only talk about poor data quality if data do not meet user expectations, for achieving her tasks. The exact semantic of "*fitness of use*" formulated in [Wan+96b], was adopted in major data quality research works [SC12; Dem82; Gar88; Jur03; Hua+98; Fan+12].

The user-centric vision of quality explains the challenge of studies aiming at quality improvement. In the last decade, quality issues concern is increasing since data is no longer collected for a particular use case. Big data is constantly generated in huge volumes and stored without considering what possible use could be achieved. This setting reduces the adequacy between what an end user gets, and the data she can exploit because data have not necessarily been collected and structured for her needs.

2.2.1 Taxonomies of Data Quality Problems

Assessing the quality of data is a critical task that does not accept making assumptions about a data collection state. It requires identifying a set of problems that determine non-adequacy factors with user metrics for quality. Quality assessment and improvement processes efficiency depends entirely on the exactness and the completeness of the problem identification step. Different data quality taxonomies have been defined in the literature, to cover the variety of data natures, contexts, and sizes. Some taxonomies enumerate common data anomalies for a generic assessment process, and others design fine-grained classifications focusing on a particular domain.

In a recent survey [Gsc+12], authors established a listing of the most popular taxonomies, each considering a particular aspect for anomalies classification. The need for building targeted data cleaning approaches motivates establishing taxonomies that produce anomalies classifications. Given that most existing taxonomies cover structured data anomalies, we enrich the survey with recent advances regarding unstructured data quality problems. We provide in the following paragraphs a simplified summary for data quality taxonomies, according to their underlying classification perspective.

Data granularity Structured data are usually stored in multi-level entities(relations, objects, dimensions, graphs). Let us take the case of the relational model, where data are organized in rows of many attributes values. Anomalies may occur at different granularity levels. Tuple duplicates, wrong attribute values, or referential constraint violation are examples of quality issues at different levels, resulting in variable assessment and cleaning approaches. A set of quality studies follow this classification in their quality assessment approach [Oli+05; Woo+14; Bar+05]. The contribution of Oliveria et al. [Oli+05] uses the same separation at a fine-grained level providing a listing of 33 anomalies. The latter multi-level classifications consider variations in a single data source. With data integration systems, the variability issue grows toward wider anomalies extent. Data can emanate from multiple sources, with heterogeneous local structures, entailing additional quality irregularities. All structural conflicts that may occur locally spread exponentially with distinct sources crossing: naming conflicts, domains adequacy, format misspellings, or duplicates. In [Rah+00], the author suggests a taxonomy of problems sorted following single or multiple sources. Many other works fall in the same classification schema.

Data semantics The taxonomy built in [Mü+05] examines data quality problems by distinguishing syntactic anomalies that impact entities representation (misspellings, wrong schema attributes) from semantic problems which deteriorate data interpretation (wrong values, misfielded values, contradictions or out of range values). The motivation of such classification is providing cleaning strategies with guidance to separate syntax checking operations from repairing tasks requiring

external knowledge for fixing values' errors. Also, coverage anomalies are listed in a third category to indicate that some mandatory data are missing: attribute values or tuples. Missing attributes in the schema are also considered in this category.

Data representation level Barateiro and al. [Bar+05] address the quality problem by detecting schema issues aside from the instance. Authors argue that common schema anomalies, as attributes naming or integrity constraints definition, could be avoided by the database management system *RDBMS* tools. This option may not completely eliminate schema level problems, but the remaining part could be fixed by schema design enhancements which do not require the same workload as for instance anomalies, where the required cleaning cost depends on the values variability factor. For example, an error on an attribute type could be rectified in the design stage, and it will be valid for the entire column. On the other hand, if a misspelling error is frequent in an instance, it is necessary to apply a case-by-case treatment, given the variability of values.

Data structure Unstructured data comprise multiple types of formats as audio files, video, image, or unstructured text. They come from multiple human or machine producing operations, such as medical imagery, social networks posting, or sensor outputs. Unstructured data are nowadays the main material for machine learning tasks, and anomalies are much more hard to identify. The diversity of formats entails the absence of quality assessment standard. Indeed, research in the area remains insufficient, compared to quality assessment needs in data pre-processing tasks, or the well-established taxonomies covering structured data.

Recently, efforts have been paid to propose quality problems identification methods for unstructured data [Bat+16; Son04; Imm+15; Tod+15; Kie16; Tal+18], providing first, an updated definition for Data quality. Since unstructured data are commonly consumed by machine users, the "fitness of use" concept is extended to include data compliance to operating programs requirements such as learning techniques. Quality problems, in this sense, are defined as complex data variations preventing exploration techniques from valuable knowledge extraction. In speech recognition, noise deteriorates the meaning of the extracted text message, image processing requires a minimum pixels resolution for efficient interpretation, and video annotation needs some metadata as environment localization to produce meaningful comments. Furthermore, metadata are useful, in general, for any unstructured data format, serving as a support to understand basic features that a structure would have offered.

We summarise reviewed taxonomy features in Figure 2.1. For a data quality study, adopting a single taxonomy as a baseline may not be the best option. A better alternative should consider data nature and tasks requirements to identify which classification (or set of classifications) fits the most the expressed needs. A good track for data quality designers is to recognize the task context, in order to formulate targetted methodologies that answer user expectations.

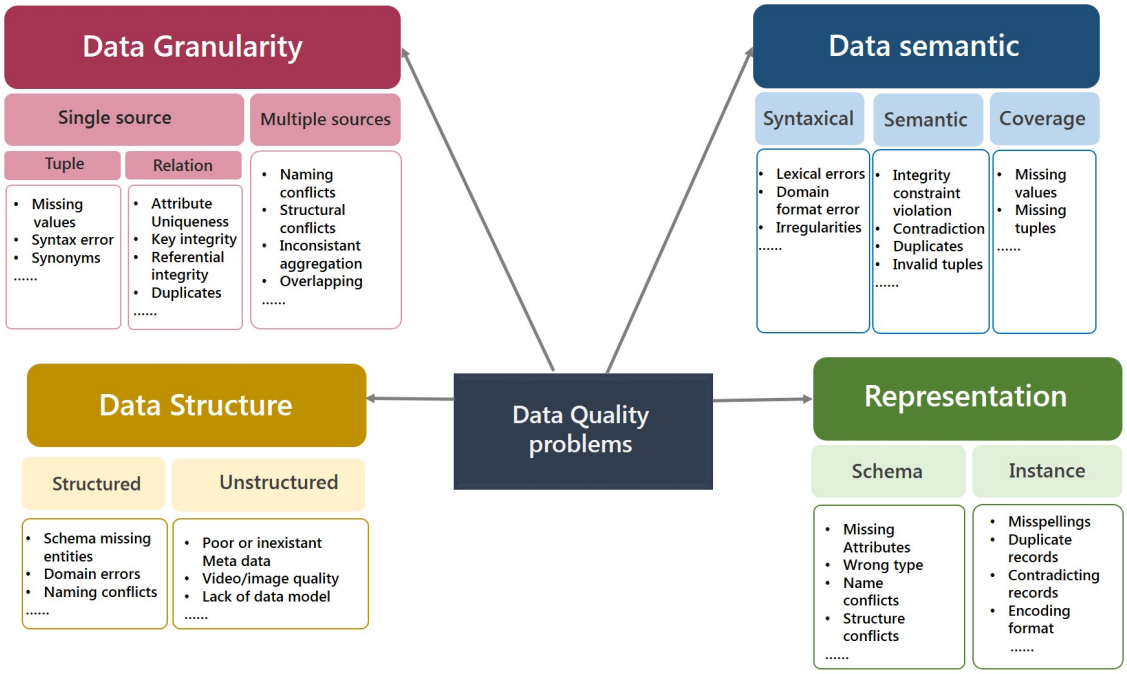


Fig. 2.1.: Some Data quality problems taxonomies

Context-Dependent problems As various as they are, DQ problems taxonomies agree for identifying a data anomaly as a problem to fix. This no longer holds true, in some specific contexts, where a variation in data, yet known to be valid, arises as a problem in particular cases such as country legislation or corporate business rules.

Take the example of missing data; we can not determine if a dataset is complete without referring to a context reference provided by the user. The study in [Ge+07] reviews the classification of DQ problems by splitting anomalies into two categories: those evolving with a context and context-free issues. The first might be detected independently from any awareness of usage purposes, as duplication or syntax errors. In contrast, issues that fall within the scope of context-knowledge, are identified aside, such as business domain constraints.

Context-dependent problems are anomalies that break previously defined "standards" for quality problems. Furthermore, coping with context requires particular attention to develop basic taxonomies toward more flexible classifications, including the use-case settings. The following example illustrates a context-dependent use case.

Example 2.1. *In 2018, the European Union voted a reform called General Data Protection Regulation. This law aims to apply some standards and constraints on personal data collected by websites and Internet applications, for heightened privacy aims. Websites could not share personal data for advertising purposes without adding constraints on personal identity or asking for user permission. In order to include this new reform to their data protection policy, data-centric applications have updated their*

database design constraints to fit the European context. A part of the data collection process remains unchanged outside Europe, being valid in these zones.

The diversity of designed DQ approaches and methodologies can be explained by the complexity of the quality concept. We can understand from the definition "*fitness for use*" that quality is a relative notion, which is comforted by the heterogeneity of taxonomies listing quality issues.The following section explains how problems are reduced into grouped dimensions.

2.2.2 Data Quality Dimensions

The motivation of quality dimensions is grouping similar quality criteria that translate user expectations, to ease the quality assessment process. Quality metrics are consequently assigned to the defined dimensions. As we saw in the previous section, there exists a large number of problems regarding data quality, and mapping these issues into a quality dimension is a necessary step, to ensure effective evaluation and improvement process.

A data quality dimension is a set of DQ attributes, and every single one is the representation of an issue [Wan+96b]. Grouping attributes into dimensions is a step that precedes data analysis to discover measurable criteria to maintain during the task.

There exists a diversity of dimensions that could be explained with context or domain specifications, data nature, and user expectations for target quality criterion. Besides, Batini [Bat+09] observes that there is no general agreement on dimension semantic: each research work tackles DQ problems from its perspective, and associates subsequently a different definition for a given dimension. The author enumerates six major DQ dimension classifications proposed in the following works [Wan+96a; Wan+96b; Red96; Bov+03; Nau03] , and establishes on this basis a common *dimensions core*, that occurs in the succeeding quality studies.

Example 2.2. *In order to illustrate the concept of quality dimensions, we consider the example of a data table representing the number of tourists in points of interest for different cities. The data table is represented in the Figure 2.2, which identifies anomalies occurring in the data table and link them to dimensions.*

The dimensions core identified in [Bat+09] is composed of four dimensions: Completeness, Accuracy, Consistency, and Timeliness and. The following paragraphs provide a general overview of each.

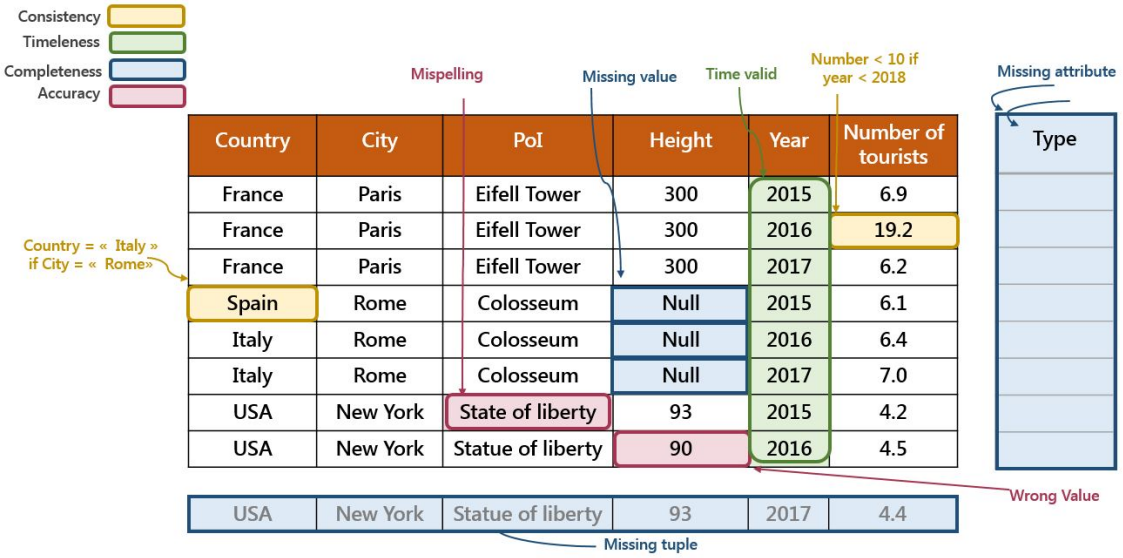


Fig. 2.2.: Tourists dataset quality dimensions illustration

Completeness Completeness is defined as the degree to which a collection of data corresponds to the real world it describes[Bat+09]. In another user-centric definition, [Wan+96b] explains that the completeness of a dataset represents *The extent to which data are of sufficient breadth, depth, and scope for the task at hand*. This dimension is related in relational databases to the notion of coverage, and this includes three levels:

- Schema: does the schema capture all the concepts and attributes of the real world? In the tourist Example 2.2, we need to identify the type of points of interest (museum, library, public building), but the relation schema does not include such an attribute.

- Value: are all the required values describing tuples available? The problem of missing values or null value is very well studied in the literature. A missing value may be non-existing, existing but unrecorded, or with possible existence. In the Example2.2, the height of the monument "Colosseum" is not indicated; this value is missing but exists in the real world.

- Tuple: does the database include the entire population? Missing tuples is also a traditional problem in DQ research. The number of persons that have visited the Statue of Liberty for the year 2017 is unknown. The entire tuple is not included in the data table.

Data completeness is the center of interest in our research work, and we dedicate the next Section 2.3 to data completeness studies.

Accuracy According to [Red96], accuracy is *a measure of proximity of a data value v, to some other value v' that is considered correct* . Accuracy is a dimension concerned with the correctness of values. Data values are of poor quality if they present syntactic or semantic errors.

- Syntax: the value v is incorrect if it does not belong to its corresponding domain. Misspelling errors, for example, bring up new values not included in the domain but approximating existing ones. Observe in Figure 2.2 that the *State of Liberty* is not a monument in New York.

- Semantics: The value v is not correct if it does not match the description of the object it describes. The value 90 m belongs to $Dom(Height)$, but it is the wrong value for *Statue of Liberty*.

As indicated in [Bat+09], the majority of DQ research focuses on the syntactic aspect of the accuracy, to which a quality metric can be naturally associated. Indeed, we can check that a value v is external to the domain, but it is more difficult to check what exact height is the *Statue of liberty*. That checking requires referring to an external knowledge source.

Consistency A dataset is considered consistent if its values respect a set of semantic constraints [Goa+07]. In relational databases, semantic constraints take the form of integrity constraints, which can be either intra-relational or inter-relational [Abi+95]. Intra-relational constraints allow defining a set of accepted values for an attribute, sometimes restricted by other attributes values, in the same relation. Observe that the value $19.1m$ as tourists number for the *Eiffel tower* in the year 2016, presents a violation of the integrity constraint which limits the maximum number of visitors to 10 m up to 2018. The inter-relational integrity constraints, allow the same kind of restrictions involving attributes from multiple relations.

Timeliness Time dimensions are an important aspect of data quality given the high update rate and a large amount of information produced continuously. They include multiple dimensions, and the most used are timeliness, currency, and volatility, which definitions differ and overlap according to research works. For [Wan+96b], timeliness captures the age of data, and it identifies if data are outdated for exploitation. In [Bat+09], it expresses *how current data are for the task at hand*.

2.3 Data Completeness Overview

Given the impact of data completeness on the accuracy and reliability of the analysis, contributions regarding data completeness have a long history. The first interest could be traced back to statistics, where mathematical models were proposed to study the distributions of variables and detect phenomena related to missing data [Afi+66]. In databases, an early representation that extended the relational model for identifying missing values is Null values. It has been quickly shown that this system has not enough expressiveness for addressing the missing data representation problem [Gra77]. In this section, we describe different steps in data completeness studies.

Tackling data completeness is not a single step operation. It requires a pipeline of tasks that allow understanding underlying features, before providing an efficient solution. We retrieve in the literature three main categories of research studies for data completeness: 1- Representation systems(or models), that achieve missing data identification and modeling, 2- Explanatory studies, using correlation-based techniques for detecting the missingness origin and behavior and 3- Cleaning strategies, mostly data enrichment models (figure 2.3).

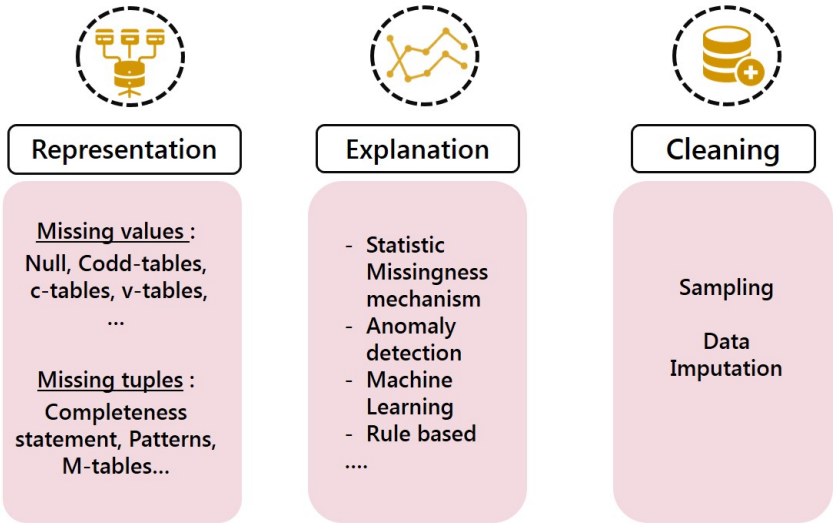


Fig. 2.3.: Data completeness study tasks

(i)-Data Representation The scope of the first part of this thesis extends to completeness representation models. The next Section 2.4 will be dedicated to surveying major advances in the field.

(ii)-Data explanation Missing data occur under various circumstances. Explaining the missing data problem require identifying the origin of incompleteness. Physical anomalies impacting data collection tools, human omitting some values, or data integration system errors are few examples illustrating a possible explanation for incompleteness. Understanding the reasons behind this issue allows providing more efficient cleaning strategies, but also learning about data acquirement and storage methodology, for future enhancements.

Missing data can appear in a population in different forms. According to data missingness mechanisms proposed by [RUB76], we distinguish three missing data types:

- *Missing completely at random*: the fact that a data value is missing cannot be explained by its real value, nor by study variables. Missing data are distributed independently of any study variables value. If a temperature value is missing in a sensor output record, and no other values are missing in the same geographical and time locations, the dataset attributes cannot explain the absence of this value, since no correlation could be established.

- *Missing at random*: the missing data is not related to its real value range, but can be explained by some other study variables values. If a sensor fails, temperature values for the spatial locality it covers, during its failure time window are missing. There exists a correlation between time and geographical attributes explaining the missingness.

- *Missing not at random*: there exists a logical rule delimiting the scope of missing values, allowing for identification. If sensors stop recording temperature value each time it exceeds 30 deg, we can expect that no value of this range is indicated in the dataset. The number of missing data depends on the probability distribution of the temperature variable values.

(iii)-Data cleaning Data cleaning aims to improve the quality of data by removing identified anomalies [Rah+00]. One approach for data completeness cleaning is setting default values for attributes if no value is recorded (0 for example). Another simplification approach for the missing data problem prevents task quality issues by directly ignoring any tuple having missing values. Removing imperfect data tuples from a task domain induces severe quality problems, by introducing significant bias due to data distribution distortion, and false data values categories representation rates.

An alternative approach has been proposed with data integration systems. It consists of finding available data sources that complete the missing required values. Data integration uses mainly source crossing, but finding accurate complementary sources remains expensive and complicated. Besides, data integration may increase the incompleteness problem in some settings [Lem04].

The most used approach for data enrichment is data imputation, which consists of computing and assigning a value to each missing field, exploiting several inference mechanisms: statistics correlation, human surveys, logical rules or machine learning. Data imputation for incomplete data represents the scope of the second part of this thesis manuscript. A state-of-the-art chapter will examine in more details imputation techniques (Chapter 5).

2.4 Data Completeness Representation Models

The interest in representing missing information in databases is as old as the domain itself. The belief that a database contains a complete representation of the world it observes has early intrigued researchers. Data may be missing in two ways: attribute values could be missing, or entire tuples are missing from the database.

The first contribution in missing values representation is the *Null* values formalized by Imielenski and Lipski [Imi+88b]. Widely used since, this unnormalized value indicates the absence of a value for an attribute and allows consequently computing query answers without ignoring incomplete

tuples, and avoiding considering the default 0 which could distort data distribution and false results (aggregations for example). Many works followed, adopting this representation system to build query evaluation systems [Cod79; Imi+84], or quality assessment metrics [Rei86]. Additional missing values were proposed stating the weakness of the Null values representation system. We discuss these ideas in the next subsection.

The assumption that a database includes every observed fact from the real world and can only miss attribute values is known as the *Closed World Assumption*. In other terms, with CWA, "*what is not known to be true must be false*". If we search in an airline company database for a flight linking Paris to New York, we should expect to find all flights ensured by this company. The price of a flight, its duration, or other attribute values could be missing, but all flights are indicated.

It turns out that in practice the *CWA* is rarely checked [Mot89]. In many cases, the database does not represent the whole information but includes only a subset of tuples. In contrast to *CWA*, this assumption is called *Open World Assumption* shortened *OWA*. Suppose we took a flight to New York, and at landing, we want to book a hotel. It is hard to find a database that includes all hotels in New York and its area. It is more practical to accept that a database may miss additional records. However, under this setting, any study becomes much more complicated to conduct considering missing tuples. Indeed, no precise characterization of the real world can state what is complete or missing in the available database, since we ignore the extent of what is outside. Works such as possible worlds theory [Abi+87] try to encounter this lack of knowledge about the ideal representation.

In the next sections, we review in more depth representation models for studying missing values 2.4.1 and missing tuples2.4.2.

2.4.1 Missing Values Representation

The missing values representation in databases motivated a high number of research contributions. It has been the first exploration track for extending the relational model designed by Codd [Cod70], for increasing the data model expressiveness. The null value representation system suffers from several limits [Mey98]. Null values do not allow expressing constraints on missing data, or equality/inequality between multiple null values semantic.

The earliest work considering extending Codd's model was a Codd himself proposal [Cod79]. He suggests using a special symbol *'@'* for characterizing any field with unknown value, creating the *Codd tables*. The semantic of a Codd table could be interpreted as a database of multiple instances of the same relation, each obtained by replacing *'@'* by any value of the corresponding attribute domain. Ilmienski and Lipski [Imi+84] show that such a system does not allow expressing join

queries[1]. Codd tables associate for a '@' symbol occurrence a single interpretation. All variables are distinct, and a variable cannot be indicated for different fields with the same value in one instance, which partially explains the expressiveness limitation. Codd tables are usually related to the semantic of Null values usage in SQL. V-tables overcome the repetition issue stated in Codd-tables, which allow for expressing join queries. Indeed, if two attribute values are marked as missing but with equality, this allows for evaluating the join on this attribute. This extension still does not strongly cover the SPJ fragment of the relational algebra following Ilmienski's criteria.

Ilmienski addresses Codd's and v-tables representation issues by proposing a new system that supports defining constraints over missing values representations: C-tables. He demonstrates that this system is a strong system for SPJUDR queries (Select-Project-Join-Union-Difference-Rename fragment of the relational Algebra). The C-tables include a condition column that expresses the condition on the tuple with a marked null value. This extension offers the possibility to evaluate fully join queries.

We refer to [Mey98] that establishes a comparative table assessing the expressive power of different representation systems extending the relational model, following the study in [Imi+84]. Table 2.1 summarizes a comparison between these systems representation expressiveness. As explained earlier, c-tables achieve the largest extension for the relational model.

Model	Strong representation	Weak representation
Codd Tables	*PR*	*PSR*
ν-tables	*PUR*	$PS^{+}UJR$
c-tables	*PSUJRD*	*PSUJRD*

Tab. 2.1.: Strong and weak representation systems extending the relational model

Example 2.3. *Observe in Table 2.2, three different representations for a sensor measure data table with missing values.*

Room	Day	Value
R1	Mon	15
R1	@	36
R2	@	29
R3	Wed	17
R3	@	17

Codd-table

Room	Day	Value
R1	Mon	15
R1	x	36
R2	y	29
R3	Wed	17
R3	x	17

v-table

Room	Day	Value	Condition
R1	Mon	15	
R1	x	36	$x \neq Mon$
R2	y	29	$y \neq x \wedge y \neq Tue$
R3	Wed	17	
R3	x	17	$x \neq Wed$

c-table

Tab. 2.2.: Different representations for the measure table with missing values

For each table, the possible worlds are table instances obtained by replacing all variables by the one value from the corresponding attribute domain(Day in the example). For ν-tables, the same instance

[1]Detailed explanations of weak/strong representation systems could be found in [Mey98]

have the same valuation for the same variable in all its occurences. In c-tables, the condition must be satisfied by variables valuations to belong to an instance.

2.4.2 Missing Tuples Representation

Adopting a common assumption (CWA/OWA) to study the completeness of a database suggests that all database parts are equal regarding completeness which do not hold true [Den+16]. Some parts of a database may have complete tuples but possibly miss some values and other parts missing entire tuples to provide complete information. Recall the tourist dataset Example 2.2, listing the number of point of interest visitors, all tuples of the data table regarding "City = Rome" are complete for the attribute number of tourists, which is not the case for New York City.

In practice, under the Closed World Assumption, too many constraints impact negatively queries evaluation. Ignoring systematically all records that do not appear in the database considering them as not true, impacts the reliability of queries answers. In the other hand, under *OWA* few queries could be answered with certainty [Fan+10b] because we *do not know if what was returned is false and nothing could be stated as complete for sure*. The complexity of query answering and the poor semantic of their results motivated many proposals to find a middle ground assumption, allowing for accepting the existence of complete database parts, providing complete answers and other parts partially complete, which may lead to incorrect results. The *Partially-Closed World Assumption* first discussed by [Mot89] corresponds to cases where only a subset of the database represents exactly a complete set of tuples.

The *partially-Closed World Assumption* is applied in the literature with two distinct settings. First, the earliest studies since Motro [Mot89] consider an ideal virtual database holding the full description of the real world (all tuples). In order to identify how the available database covers the ideal database, the proposed approaches use annotation models to translate completeness dependencies, such as patterns [Raz+15], completeness statements [Mot89; Lev96] and m-tables annotations [Sun+17]. These systems require manually annotating data with completeness constraints, limiting guarantees for exhaustive data coverage, with the absence of an ideal database. In another hand, relative information completeness deals with incomplete databases in the presence of a materialized ideal database (master data) [Fan+10a]. This setting makes a complete comparison between the two databases possible and allows querying exhaustive annotation sets. Little attention has been paid to relative completeness studies. Our representation model deals with the relative information completeness, under the "partially-Closed World Assumption," and this section surveys similar representation and discusses the major contributions.

We consider the following use case as a running example to showcase some state-of-the-art models.

Example 2.4 (Sensor networks). *Let us consider a running example of a sensor network measuring electrical consumption. The Table 2.3 describes the network recording activity for three rooms.*

Floor	Room	Week	Day	KwH
F1	R1	w1	Mon	10
F1	R1	w1	Tue	12
F1	R1	w2	Mon	9
F1	R2	w1	Mon	15
F1	R2	w1	Tue	17
F2	R1	w1	Mon	11
F2	R1	w1	Tue	13
F2	R1	w2	Mon	8
F2	R1	w2	Tue	9

Tab. 2.3.: Electrical consumption measures M

Query Completeness Statements

To the best of our knowledge, Motro [Mot89] is the first to formalize the concept of partially complete and correct databases. He considers the problem of identifying whether a query answer has integrity or not, which is satisfied if the *an answer contains the whole truth (completeness) and nothing but the truth (validity)*. In order to represent partial completeness, Motro assumes a hypothetical virtual database D_c that represents faithfully all facts that belong to the real world. Any other databases D contains a partial set of real-world data and is defined as a view over D_c. A partially complete database might also contain other data from outside the real world (invalid).

The virtual database D_c cannot be accessed to check whether data are complete (valid) or not. To ensure a basic assessment of data integrity, data in relations are annotated to separate the complete and valid data (hypothetically included in D_c) from others. The model involves a set of meta-relations that summarize the extent of ideal data, each meta-relation storing meta-tuples describing the integrity of a single data table.

Assessing the query completeness is considered as a decision problem. A query Q answer is complete if a possible rewriting using only complete data is possible for Q. This rewriting is performed using meta-tuples views. The author formalizes a mechanism extending the relational algebra operators, with additional operations that manipulate meta tuples, and allow subsequently rewriting the initial queries on database relations, using views over meta relations. These operations constitute a core of the cartesian product, selection, and projection $\{\sigma, \pi, \times\}$. Let us consider a query Q over a relation R. A new query $Q\prime$ is derived using the new operations, over R' (R meta-relation),

and produces a set of meta-tuples. If such a rewriting is possible, then the query Q answer A is complete and has a set of meta tuples A' produced by Q'.

Example 2.5 (Completeness statements). *Let D_c be the virtual complete database, for our running example. We associate completeness c_i and validity constraints v_j to the relation M (see Table 2.3), from the available database. Table 2.4 illustrates the meta-relation M' that lists completeness and validity constraints.*

	Floor	Room	Week	Day	KwH	condition predicate
c1	f2	*	*	*	*	
c2	*	*	w1	*	*	
c3	*	*	*	*	$x1$	$x1 > 13$

Completeness constraints

	Floor	Room	Week	Day	KwH	condition predicate
v1	F2	*	*	*	$x2$	$x2 > 9$

Validity constraints

Tab. 2.4.: Meta-relation $M\prime$: completeness and validity constraints

Consider the following selection queries:

$$Q_1 : \sigma_{floor=f2}M, \quad Q_2 : \sigma_{day=Monday}M$$

To decide whether Q is complete (and valid), we try to retrieve a query rewriting using only views expressed by $\{c_1, c_2, c_3\}(v_1)$. Observe that we can assess the integrity of Q_1, since a rewriting using the view $c1$ (and $v1$) is possible. This is not the case for Q_2 whereas a rewriting using $c_1 and c_2$ does not cover all the query answer, and no validity view is available for the filtering condition.

Discussion Motro uses the concept of a virtual database to represent the real world that he cannot access. However, the evaluation of basic data integrity undeniably involves a reference to ask. The model overcomes this problem by including meta-relation (annotation provided with the data) that describe the extent of completeness and validity. The viability of the entire system depends on the existence of these annotations.

The dynamic system for assessing query integrity is based on a set of operators defined over the meta tuples. This allows querying the meta relations to generate the integrity constraints of the query results. Calculated sets are correct, but does not necessarily cover all complete data in the answer. This is mostly due to some operations where meta-tuples can disappear during the process (Projecting out an attribute discards some meta-relations specifying this attribute)

Another important property relates to the soundness of complete answers. Indeed, the rewriting using views provides correct results: If a query can be rewritten using views, its answer is complete.

Queries for whom no possible rewriting is possible can be complete or not. Since only a subset of queries with complete answers can be identified, the decision mechanism produces sound set but not complete.

Local Completeness Statement

In this same setting as for Motro, Levy [Lev96] investigates the problem of deciding query answer completeness, but with a different technique. His approach uses the query-independent from updates mechanism [Elk90]. A query is independent of updates if its answer does not change with the database update. The fact that no insertion affects some query Q result means that the query asks already for exclusively complete parts in the database, producing a complete answer. In order to formalize this intuition, Levy introduces the *Local completeness statements* as constraints on tuples in a relation that are *guaranteed* to belong to a known complete relation. The tuples satisfying the constraints in a relation are complete.

The query update independence is defined for two types of updates, insertions $IN+$ and deletions $IN-$, with respect to a set of constraints. The author shows that assessing the query answer completeness is equivalent to determining its independence from insertion updates $IN+$. An algorithm *decide-completeness* resolves this problem and theoretical results establish decidability, in specific settings. Results extend naturally to cover the query answer correctness, using the same formalization as for local correctness.

Discussion The approach presented by [Lev96] reuses an interesting paradigm "Query Independent of Updates" **QIU**, adapted to completeness assessment. It improves the decision algorithm for this problem, to polynomial time. The paper suggests deciding completeness in case of deletion updates. No formalization was introduced, but the author seems confident about the easiness of extending his framework to support this case, using Query independent from deletion updates. An efficient algorithm is explained for deciding query completeness based on query independence from insertion updates. However, it puts some restrictions on queries for which a decision can be issued (ex, only constraints involving comparison predicates)

The local completeness statements proposed do not cover all possible constraints that may describe relation completeness, and this limits the scope of possible specifications. Also recall that as for Motro [Mot89], constraints must come with data.

Completeness Patterns

Razniewski et al. work in [Raz+11; Raz+15] under the partial completeness assumption as defined by [Mot89]. They consider the existence of an ideal virtual database representing the real world. However, they do not consider the validity dimension (data included in the available database but not in the real world). In order to represent the extent of completeness for a database relation, the author provides a very similar syntactic form of Motro's *C-completeness statements* under the name *completeness pattern*. Patterns take Levy's work semantic to describe at the relation granularity, complete parts of data. Each relation owns a completeness pattern table.

A pattern tuple encodes a selection on a relation, producing a complete set (exactly the same set that would the ideal database relation return). In order to provide query answers with patterns describing their completeness, a pattern algebra is defined to allow patterns manipulation and inference. Algebra operators are similar to Motro operations on completeness statements: selection (with constant and attribute equality), projection, and joins (equi-join).

The pattern algebra is used to create queries over patterns, producing pattern sets that identify the data query answer completeness extent. Theoretical proofs about the soundness and *computational* completeness of the algebra are provided. The paper presents experiments results over real and synthetic data, showing how compact are pattern sets compared to their respective data sets.

Discussion Completeness patterns represent a very intuitive formalism for capturing completeness, associated with a strong theoretical framework that ensures the dynamicity of the process, allowing to compute completeness annotations for query answers. The Algebra is sound and complete, and allow computations in reasonable times.

Similar to ([Mot89; Lev96]), Razniewski's work assumes the existence of relations annotations as patterns. No automatic derivation process has been proposed. The fact that a set of patterns exist means that covered tuples are complete, but other tuples may miss, and the model cannot assess that the computed set corresponds to *full completeness*, which prevents from deciding whether a query answer is complete or not.

Anomalies Propagation Models

In a research paper entitled *Partial results in database systems* [Lan+14], authors address the problem of labeling query results by identifying how missing data lead to wrong or missing query results. The study illustrates incomplete data case with access anomalies while integrating several

databases. A first contribution is a taxonomy of anomalies that may occur in a query result; a query is performed over an incomplete database. Anomalies are classified according to two dimensions:

- **Cardinality**: corresponds to completeness. When data are not accessible in a database or simply missing, they lead to two types of anomalies in query results. Either the result suffers from *missing tuples* or *phantom tuples*. If the first anomaly is natural to understand, phantom tuples can be generated for example, when an aggregation tuple satisfies the query condition and appears in the result, while it should not with a complete input set. An indeterminate state indicates that the result may include extra tuples or miss required tuples.

- **Credibility**: corresponds to correctness anomalies which are related to aggregation queries. If aggregation has been computed over a complete input set, the aggregation tuple in the query result is credible. If data miss in the input set, aggregation produces a potentially non-credible result.

In order to produce meaningful semantics for a query result, we need to have a minimum of knowledge about the failure that has created the incompleteness of data, and also know how the query operates. This is necessary to trace the origin and transformation of the anomaly. Consequently, we need to create as many analysis models as possible to fit with the detail level of the anomaly identification and observe as the query is executed how it impacts its final result. Authors propose four analysis models with different granularity:

- **Query**: the query is treated as a black box, and no detail is available about the level of the anomaly in input data. The result is only known to be indeterminate.

- **Operator**: at this analysis level, we deal with the query execution plan, operator by operator. We can determine the semantic of the anomaly at the input of each operator, and we know how each one affects the quality of data it outputs.

- **Column**: to observe how data integrity evolves through the execution plan, the operator model treats data as a single bloc, while a more detailed like column level anomalies can sometimes be relevant to identify more specific anomalies such as those directly affected (or created) by a projection.

- **Partition**: represents the finest model proposed. It is based on a horizontal partitioning of data, where a set of tuples create a partition, and anomalies are observed at this level.

The choice of a plan depends on the granularity of acquired anomalies, and the cost one is willing to pay to get this analysis. A trade-off between the precision of explanation and the cost of generating these labels has to be found.

The authors then carry out a study of the behavior of each operator of the relational algebra in the face of the anomalies he receives as input. Thus for each of the unary operators (select, project, extended project and aggregate) and binary operators (Cartesian product, union and set difference), the author explains how an anomaly is kept, removed or transformed. A query on a database can lead to different execution plans, with the guarantee of producing the same result. The models of annotations of the results of the request, progress operators by operators and are thus susceptible to semantics variable according to the scheduling of the operators in the execution plan. The paper provides a discussion of the equivalence of operator scheduling in terms of calculated semantics. All pairs of operators are not described, but for the relevant reordering, the system seems resistant to scheduling variability;

In a more recent work [Lan+15], the author defines a generic cost model for query evaluation, which associates the cost of the plan with that of the data retriever. The partial result penalty model relies on the user to translate its preferences for query result annotation, into penalties to be incorporated into the cost model. The user can thus choose which anomalies he tolerates to appear in the query result and associates them a cost. This approach joins the concept of data quality guided by user satisfaction.

Discussion The partial result concept introduced offers an extension for the true default result that allows a better understanding of anomalies generated with a query over an incomplete database. Despite a well-illustrated discussion of how relational algebra operators propagate anomalies, no complete deterministic approach has been proposed for automatic label generation. The indeterminate state remains omnipresent in many cases, which represents a completeness issue for query answer labeling. The work considers the partial world assumption without any reference, and propose a scenario to simulate incomplete information generation as access failures to some database parts. In practice, to adapt the model usage, we need to provide some prior knowledge about the completeness of raw data in other contexts that the one mentioned by authors.

M-tables

In [Sun+17], authors present a new representation system for incomplete databases, largely inspired by the conditional tables of [Imi+84]. It allows representing both missing values and missing tuples whether their cardinality is known or not. The first element of this generic representation system is a symbol **m** which is used to represent any missing value, with as a distinction, the possibility of representing any value of the domain or even several values. An extended tuple in this system $\widehat{t}$ is a classical tuple that can take **m** as a value for an attribute whose value is initially missing.

Since this specific symbol **m** can be associated with several attributes in the same extended tuple, it is necessary to distinguish its range using constraints. Constraints encode prior knowledge about the range of possible values in the ideal database. An annotation system on the extended tables is proposed to overcome this problem, by translating the constraints on the domains of **m** but also the inter-attribute and inter-table constraints. Annotation of the extended relation goes through a schema that specifies which attributes are affected by the uncertainty or incompetence. On this new schema, one defines polynomials, with variables representing the attributes concerned, and constraints on the values of its attributes which can if possible delimit the possible tuples. Some tuples are designated by the polynomial (1). Each monome represents an attribute or several and its coefficient the degree of the multiplicity of this one. Constraints take the form of classical comparison predicates (<,>, =,..).

From the basic annotations of missing tables, we can derive the same type of annotation to characterize the completeness of query results. Each operator applying to the extended tuple also applies to its annotation, by performing a specific transformation to its semantics. Accepted queries limit to the positive fragment of the relational algebra $\mathcal{RA}^+$ (selection, projection, join, union)

A simple labeling algorithm allows automatic generation of two types of labels, *certain* and *possible*, for a straightforward interpretation of the query results computed annotations. The paper also presents theoretical results and proofs for the model expressiveness and completeness, compared to the traditional c-tables.

Discussion In this work, a new representation model for capturing missing data and values is introduced. It is widely inspired by the conditional tables formalism but offers a better expressiveness. The model is theoretically well defined, and satisfies the properties of completeness and closure for bag and set semantics. Query results can have their own annotations, but this only covers positive queries. Queries with negation or aggregation queries, for example, cannot be annotated. It is also important to understand that no exhaustive knowledge could be guaranteed with initial annotations, due to the absence of a reference and this makes the completeness of the derived annotations dependent on the completeness of the primary set.

Relative Information Completeness

The relative information completeness studies consider the existence of master data[Los10; Fan+10a; Fan+12; Den+16]. In addition to the potentially incomplete database, subject of study D, a master database containing only complete data tables is available D_m. In this setting, both databases are materialized. In [Fan+10a] a set of containment constraints describe the extent to which D includes complete data from D_m. However, all data in D does not necessarily satisfy these

constraints. Authors have a particular perspective of this setting: the part of D satisfying constraints are considered to be a closed world, and the other part with no inclusion constraints regarding master data to be open world (and not necessarily incomplete).

Both queries Q and containment constraints can be expressed in the different query languages considered in the model: Conjunctive queries, union of conjunctive queries, positive existential FO queries, first-order logic queries, and datalog queries. We refer to L_c as the chosen language for constrainsts V and L_Q for queries expression language.

Given a master data D_m, a database D, and a set of containment constraints V, the paper deals with two completeness decision problems for the query Q:

- $RCDP(\mathcal{L}_\mathcal{Q}, \mathcal{L}_\mathcal{C})$: checks whether the database D is complete for the query Q under (D_m,V).
- $RCQP(\mathcal{L}_\mathcal{Q}, \mathcal{L}_\mathcal{C})$: Decide if a database D exists w.r.t (D_m,V), and complete for Q.

In a more recent work, [Den+16], authors extend their model to accept the presence of missing values, as one aspect of database incompleteness. A study of the complexity of both models was discussed in [Cao+14].

Discussion Works on relative information completeness exploit the presence of master data to describe the completeness of databases. This assumption allows an interesting representation system, with decision tasks that can be extended to multi-instances, for one master database. Yet, the model suffers from its own setting: the non-completeness of containment constraints restrict query answer completeness assessment possibilities. We can only infer knowledge about the part of the database covered by containment constraints. The process is not complete,the non-existence of a rewriting does not mean that the query answer is necessarily incomplete.

The presence of an ideal database could introduce many possibilities for representing and manipulating missing data, which is unfortunately not taken into consideration is this model [Fan+10a], nor in its extension [Fan15]. A powerful component of this framework remains the study of the decision problem complexity, considering different languages in which queries and constraints can be expressed. [Den+16]

2.5 Summary

Data quality represents the extent to which data fit user requirements for achieving her tasks. We surveyed in this chapter, data quality studies contributions listing data quality problems and dimensions. We proposed a common taxonomy that integrates state-of-the-art representations, following data structure, granularity, semantics, and context. Since we focus on the completeness dimension study, we put considerable effort into gathering and summarizing contributions advancing the completeness study field.

Working under the Closed World Assumption for completeness study does not fit our problem requirements. The assumption that only values can be missing is both restrictive and oversimplified. In another hand, under the Open World Assumption, no information extracted from an available database can be trusted to be complete. A similar setting complicates every descriptive study and makes all queries results obtained from the database, of indeterminate quality.

The partially-Closed World Assumption is a middle ground setting, where parts of the database can be assessed as complete and queried safely while remaining parts are under the Open World Assumption. Many representation models have been proposed under this setting, that corresponds to the assumption we adopt. We discussed six major contributions, tackling similar problems as those we address (Chapter 1), with similar study purposes. We summarize these representation models in a comparative Table 2.5. We establish this comparison according to four criteria:

- How the primary knowledge about data completeness is extracted/generated? We check whether **Annotations** are injected manually or generated automatically.
- Which **formalism** is used for inferring knowledge about the query answer quality (completeness and derived dimensions)?
- What is the **result** associated with a query answer? a yes or no answer or a more low granularity label?
- What **guarantees** should the user expect while implementing the model? Soundness, completeness, or both?

Most of these works run the study with the assumption that the ideal database is virtual and proceed with an annotation process that is often manual to describe the extent of complete data or what coverage guarantee available data regarding the reference [Mot89; Lev96; Raz+15]. The absence of a materialized reference does not only costs an extensive effort for creating completeness annotations but also prevents from describing the extent of missing data. Except for [Sun+17],

Reference Model	Annotations	Formalism	Result	Guarantees
[Mot89]	Completeness statements	Query rewriting Using views	Decision task Yes/No	sound
[Lev96]	Local statements	Query Independent from updates	Decision task Yes/No	sound
[Lan+14]	physical anomalies logs	operations models	Labels:complete ,correct, phantom	sound and complete
[Raz+15]	completeness patterns	pattern algebra	Pattern sets	sound and complete
[Sun+17]	m-annotations	m-operators	polynome annotations	sound
[Fan+10a]	containment constrainsts	Query rewriting	decision task	sound

Tab. 2.5.: Comparative table for missing data representation models

no contribution has considered annotating query results with missing data. Besides, completeness annotations such as patterns in [Raz+15] are not guaranteed to be complete.

Regardless of the missing data origin (sensor failure, storage system access), relative information completeness proposes to study data completeness against a materialized database. [Fan+10a] argue that such knowledge guarantees the exhaustivity of completeness annotations. We distinguish two types of results regarding query results annotations:

- *Decision task*: aims to provide a Boolean answer to this question: Does the query result returned match exactly the complete answer that the reference would return? for many applications, this information is sufficient for assessing the task quality [Mot89; Lev96].

- *Local descriptions*: Other works [Raz+15; Lan+14] target a more detailed explanation. For incomplete query answers, they provide more information by identifying complete answers from those impacted by missing data. This takes the shape of different annotations forms: incorrect incomplete labels in [Lan+14], completeness patterns in [Raz+15]. Even if more complex to derive, these models do not necessarily ensure the underlying decision task: we know what parts are complete but not always if these parts constitute a global complete query answer.

Pattern Model and Algebra

3

> *Have no fear of perfection; you will never reach it.*
>
> — **Marie Sklodowska-Curie**
> Nobel Prizes in Physics and Chemistry

Contents

3.1 Introduction

In this chapter, we describe two major contributions:

1. The pattern-based model for representing the relative information completeness. This model is defined over constrained tables, and pattern sets identify the extent of complete and missing parts of the incomplete table.

2. The pattern algebra that allows defining pattern queries over complete and missing pattern tables. We prove that this algebra is sound and complete regarding the SPJUD fragment of the relational algebra.

This chapter is organized as follows. Section 3.2 introduces constrained tables as a means to represent relative information data. Section3.3 presents partition patterns and minimal covers which are used for describing complete and missing data table partitions. Section 3.4 presents the pattern algebra which revisits the standard relation algebra and introduces two new operators, *fold* and *unfold*, for mapping standard relational tables to patterns tables. Rewriting rules allowing for optimized pattern queries are also studied in section. Section 3.5 illustrates how pattern queries can be implemented with SQL queries. Section 3.6 deals with the setting where a reference is expressed as a cross-product of independent tables and sketches some query optimization opportunities. A summary of the chapter is provided in Section 3.7.

3.2 Relative Information Model

Our contribution falls within the setting of relative information completeness which is a particular setting of the "Partially-Closed World Assumption" (see Chapter 2) in which the ideal database is no longer virtual but materialized. We start by defining the notion of *Constrained Tables* which is essential in this chapter.

3.2.1 Constrained Tables

The main extension that distinguishes our data model compared to the relational model is the possibility to define *reference tables* for representing completeness constraints over *data tables*.

Definition 3.1 (Reference and Constrained Tables)**.** *Let D and R be two relational tables and A the set of attributes of R. If A is a key in table D, table R is called a* reference table *for* data table D *and the pair $T = (D, R)$ is called a* constrained table.

In the sequel, given a table T, we denote with $Atts(T)$ the set of attributes of T.

A reference is either readily available or may be obtained by querying an existing database. For example, when studying the *spatio-temporal* completeness of some data, the reference is the spatial and temporal coverage of data, as illustrated in the following examples. One may define different references based on the purpose of the analysis that is carried out while respecting the condition that the schema of the reference table must coincide with the schema of the data table restricted to its primary key attributes.

Example 3.1 (Constrained tables)**.** *Consider Table 3.1 which presents several data tables with spatial or temporal attributes, and a data table $Measures$, whose completeness is to be analyzed w.r.t. to different possible references. The primary key of $Measures$ is $\{Floor, Room, Week, Day, KWH\}$. Consider the following candidate reference tables for the $Measures$ defined as follows:*

- R_1 *= Map* $\times$ *Weeks* $\times$ *Days, which contains all combination of localities and days,*
- $R_2 = \Pi_{Floor,Room}$ *Meeting* $\times$ *Weeks* $\times$ *Days, restricts to room R1 in both floors*
- $R_3 = \Pi_{Floor,Room}$ *Sensor, considers an external table defining spatial locations of sensors.*

Only R_1 and R_2 whose schema is {Floor, Room, Week, Day} can be considered as a reference for $Measures$ while R_3 which does not contain the temporal attributes Week and Day can not be a reference for this table. However, R_3 can serve a reference for $\pi_{Week,Day}Measures = \{(F1, R1), (F1, R2), (F2, R1)\}$ since $Atts(R_3) = \{Week, Day\}$.

Incompleteness may come in different flavors: missing tuples or missing values. Under some restrictions, the notion of constrained tables defined above can be adapted to the case of missing values. Precisely, when in a constrained table $T = (D, R)$, $\pi_{Atts(R)}D$ is a key for D, one can extend D with all tuples from $R - D$ while assigning Null to the $Atts(D) - Atts(R)$.

Any data table M with a set of key attributes A and other attributes accepting null values, could be transformed into a constrained data table. The table M could be split into a reference R= $\pi_A M$, and a table D= $\sigma_{\wedge_{a\in(B-A)}a\neq Null}M$.

Example 3.2 (Misssing Values)**.** *Observe the Table 3.2. In the data table M, all spatial and temporal attributes values are indicated and the unrecorded measures are indicated as $Null$ values for the KWH attribute. M can be transformed into a constrained table $T = (Measures, R)$ with:*

- *Reference attributes A= { Floor, Room, Week, Day} ,*
- *Data table $Measures = \sigma_{Kwh\neq null}M$,*

Floor	Room
F1	R1
F1	R2
F1	R3
F2	R1
F2	R2

Map

Week
W1
W2

Weeks

Day
Mon
Tue
Wed
Thu
Fri

Days

Floor	Room	area
F1	R1	23
F2	R1	19

Meeting

Floor	Room	Sensor
F1	R1	s165
F2	R1	s196
F3	R1	s318

Sensor

Floor	Room	Week	Day	KwH
F1	R1	W1	Mon	10
F1	R1	W1	Tue	12
F1	R1	W2	Mon	9
F1	R2	W1	Mon	15
F1	R2	W1	Tue	17
F2	R1	W1	Mon	11
F2	R1	W1	Tue	13
F2	R1	W2	Mon	8
F2	R1	W2	Tue	9

$Measures$

Tab. 3.1.: A data table and its candidate reference tables

- *Reference table* $R = \Pi_A M$.

3.2.2 Assessing Data Completeness

The notion of constrained tables provides the means for assessing completeness of data in a rather intuitive way as presented below.

Definition 3.2 (Table Completeness)**.** *A constrained table* $T = (D, R)$ *with reference attributes* A *is* complete *iff* $R \subseteq \pi_A(D)$.

Proposition 3.2.1. *(Transitivity) Completeness is transitive: if* $R' \subseteq R$ *and* $T = (D, R)$ *is complete, then the constrained table* $T' = (D, R')$*. The same holds for any* D' *such that* $D \subseteq D'$.

Example 3.3 (Table Completeness)**.** *Observe data and reference tables in Table 3.1. The Table 3.3 depicts the complete and incomplete constrained tables among candidates.*

3.3 The Pattern Model

The completeness representation system we define aims to provide a compact representation for complete and missing partitions in a data table, where a data partition is a table fragment obtained

Floor	Room	Week	Day	KwH
F1	R1	W1	Mon	10
F1	R1	W1	Tue	12
F1	R1	W2	Mon	9
F1	R1	W2	Tue	Null
F1	R2	W1	Mon	15
F1	R2	W1	Tue	17
F1	R2	W2	Mon	Null
F1	R2	W2	Tue	Null
F2	R1	W1	Mon	11
F2	R1	W1	Tue	13
F2	R1	W2	Mon	8
F2	R1	W2	Tue	9

M

Floor	Room	Week	Day	KwH
F1	R1	W1	Mon	10
F1	R1	W1	Tue	12
F1	R1	W2	Mon	9
F1	R2	W1	Mon	15
F1	R2	W1	Tue	17
F2	R1	W1	Mon	11
F2	R1	W1	Tue	13
F2	R1	W2	Mon	8
F2	R1	W2	Tue	9

$Measures$

Tab. 3.2.: Transforming a table M with Null values into an equivalent constrained table.

Incomplete	Complete
$(Measures, R_1)$	(Measure,$\sigma_{week=W1}R_2$)
$(Measures, R_2)$	($\Pi_{Floor,Room,KwH}$ Measure, $\sigma_{Floor=F1 \vee Floor=F2}R_3$)

Tab. 3.3.: Example of complete and incomplete constrained tables

by a select-query. In this section, we introduce the pattern model we propose for completeness representation.

3.3.1 Partition Patterns

Definition 3.3 (Partition Pattern)**.** *Let $A = \{a_1, a_2, ..., a_n\}$ be a set of reference attributes where the domain of each attribute is extended by a distinguished value $*$called wildcard. A* (partition) pattern *$p = [a_1 : v_1, a_2 : v_2, ..., a_n : v_n]$ over A is a tuple which assigns to each reference attribute $a_i \in A$ a value $v_i \in dom(a_i) \cup \{*\}$ in the extended domain of a_i.*

A set of partition patterns $P(A) = \{p_1, p_2, \ldots, p_k\}$ over a set of reference attributes A is called a pattern table.

In the following we will denote by [$*$] the *wildcard pattern* p= [a_1:$*$,..., a_n:$*$] where all attributes a_i are assigned to wildcards. Any data tuple can also be considered as a pattern, without any wildcards. A pattern table might include only data tuples and a data table is also considered a pattern table.

Example 3.4 (Pattern Tables)**.** *Recall the reference table R_1 from Example 3.1. The reference attributes have the following domains: $Dom(Floor)$={ F1, F2}, $Dom(Room)$={ R1,R2},$Dom(Week)$*

= {W1,W2 }, and $Dom(Day)$={Mon, Tue, Wed, Thu, Fri }. Based on these domains, we can represent multiple pattern tables with variable sizes, Table 3.4 illustrates one example.

Floor	Room	Week	Day
F1	∗	∗	∗
F2	∗	∗	∗
∗	R1	∗	∗
∗	R2	∗	∗
∗	∗	W1	∗
∗	∗	W2	∗
∗	∗	∗	Mon
F1	∗	∗	Tue
F2	∗	W1	∗
F1	R1	W2	Mon

Tab. 3.4.: A pattern table example

The following definition introduces the syntactic relationships between partition patterns: generalization and specialization.

Definition 3.4 (Pattern Generalization and Specialization)**.** *A pattern tuple p_1* generalizes *a pattern tuple p_2 if p_1 can be obtained from p_2 by replacing zero or more constants by wildcards. Inversely, p_1* specializes *p_2 if p_1 can be obtained from p_2 by replacing zero or more wildcards by constants.*

According to this definition, we deduce the following properties:

- Reflexivity: p generalizes and specializes p.
- Transitivity: if p_1 specializes p_2 and p_2 specializes p_3, then p_1 specializes p_3 (the same for generalization).

Example 3.5 (Pattern generalization)**.** *Let us consider the following patterns: p_1 = [F1,∗,∗,∗], p_2 = [F1,R1,∗,∗] , p_3 = [F1,R1,W1,∗]. We can easily check that p_2 and p_3 are two specializations of p_1, since they can be obtained by replacing the wildcard for attributes* Room *and (*Room,Week*) respectively, by constants. This naturally implies that p_1 is a generalization of p_2 and p_3.*

3.3.2 Pattern Semantics

Patterns are syntactic representations that take their semantic when associated with a constrained table. We explain in the following definitions the pattern instance in a data table, and how a constrained table *satisfies* a pattern tuple.

Definition 3.5 (Pattern Instance). *The* instance $I(p, S)$ *of a pattern* p *in some table* S *is the* subset *of tuples* $t \in S$ *which are specializations of* p.

The instance $I(p, S)$ of a pattern $p = [a_1 : v_1, a_2 : v_2, ..., a_n : v_n]$ in some table S can be computed by a relational selection $I(p, S) = \sigma_{cond(p)}(S)$ with filtering condition $cond(p) = \bigwedge_{p.a_i \neq *}(a_i = p.a_i)$. It is then easy to show the following properties of I for all patterns p and tables S and S':

- $I([*], S) = S$ since the filtering condition is empty and no selection is applied;
- $I(p, I(p, S)) = I(p, S)$ thanks to the idempotence of σ;
- $S \subseteq S' \Rightarrow I(p, S) \subseteq I(p, S')$ thanks to the monotonicity of σ.

The notion of instance can naturally be extended from pattern tuples to pattern tables P and constrained tables $T = (D, R)$: $I(P, S) = \bigcup_{p \in P} I(p, S)$ and $I(p, T) = (I(p, D), I(p, R))$.

Definition 3.6 (Pattern Satisfaction). *A constrained table* $T = (D, R)$ *over a set of reference attributes* A satisfies *a* completeness *pattern* p*, denoted* $T \models p$*, iff* $I(p, R) \subseteq I(p, D)$. *A constrained table* T satisfies *a completeness pattern table* P *if* T *satisfies all patterns in* P*:* $T \models P \equiv \forall p \in P : T \models p$.

This definition corresponds to a *completness pattern* satisfaction, this term shortcuts the "*complete partition*" pattern. In Section 3.3.3, we explain how a constrained table satisfies a "missing partition" pattern or a *missing* pattern.

The following proposition provides an equivalent definition for the complete table Definition 3.2.

Proposition 3.3.1. *A constrained table is* complete *if it satisfies wildcard pattern.*

Example 3.6 (Pattern Instance, Pattern Satisfaction). *Consider the constrained table* $T = (Measures, R)$ *presented in Table 3.5, where* $R = Map \times Weeks \times Days$*. The table* $Measures$ *is* incomplete *w.r.t.* R *whereas the partition* $\sigma_{floor='F2'}Measures$ *is* complete *w.r.t.* R *since*

$$\sigma_{Floor='F2'}R = \sigma_{Floor='F2'}(\pi_{Floor,Room,Week,Day}Measures)$$

Therefore, T *satisfies the pattern [F2,*,*,*].*

The semantic counterpart of the (purely syntactic) notion of generalization/specialization is pattern *subsumption* which is defined as follows.

Definition 3.7 (Pattern Subsumption). *A pattern* p_2 subsumes *a pattern* p_1*, denoted* $p_1 \sqsubseteq p_2$*, iff* for all *constrained tables* T*:* $T \models p_2 \Rightarrow T \models p_1$. *Two patterns are* equivalent*, denoted* $p_1 \equiv p_2$*, iff* $p_2 \sqsubseteq p_1 \wedge p_1 \sqsubseteq p_2$.

Floor	Room	Week	Day	KwH
F1	R1	W1	Mon	10
F1	R1	W1	Tue	12
F1	R1	W2	Mon	9
F1	R2	W1	Mon	15
F1	R2	W1	Tue	17
F2	R1	W1	Mon	11
F2	R1	W1	Tue	13
F2	R1	W2	Mon	8
F2	R1	W2	Tue	9

Measures

Floor	Room
F1	R1
F1	R2
F2	R1

Map

Week
W1
W2

Weeks

Day
Mon
Tue

Days

Tab. 3.5.: A constrained table.

According to this definition, the wildcard pattern $[*]$ subsumes all other patterns since its instance is the full data table.

The following proposition links between the notion of subsumption and that of instance.

Proposition 3.3.2 (Subsumption and Instance)**.** *There exist a relationship between pattern subsumption and patterns instances:* $p_1 \sqsubseteq p_2 \Rightarrow \forall S : I(p_1, S) \subseteq I(p_2, S)$

Proof: We show that if there exists a table S where $I(p_1, S) \not\subseteq I(p_2, S)$, then $p_1 \not\sqsubseteq p_2$. For showing $p_1 \not\sqsubseteq p_2$, we define a constrained table $T = (D, R)$ such that $I(p_2, R) \subseteq I(p_2, D)$ and $I(p_1, R) \not\subseteq I(p_1, D)$. Let $R = S$ and $D = I(p_2, R)$. Then, $I(p_2, D) = I(p_2, I(p_2, R)) = I(p_2, R)$ (by idempotency).

Now we have to show that $I(p_1, R) \not\subseteq I(p_1, D)$. Based on the initial assumption $I(p_1, S) \not\subseteq I(p_2, S)$ and $S = R$ we conclude $I(p_1, R) \not\subseteq I(p_2, R)$ and it is sufficient to show that $I(p_1, D) \subseteq I(p_2, R)$: $I(p_1, D) = I(p_1, I(p_2, R)) \subseteq I(p_2, R) = I(p_2, R)$. □

The following proposition states the relationship between the syntactic notion of specialization and the semantic notion of subsumption.

Proposition 3.3.3. *If* p_1 *is a specialization of* p_2 *then* $p_1 \sqsubseteq p_2$.

Proof: We show that if p_1 is a specialization of p_2, then $p_1 \sqsubseteq p_2$. If p_1 is a specialization of p_2, then for all S, $I(p_1, S) = I(p_1, I(p_2, S))$ (then the filtering condition of p_1 is subsumed by the filtering condition of p_2). Then, if $I(p_2, R) \subseteq I(p_2, D)$ we know by monotonicity of I that $I(p_1, I(p_2, R)) \subseteq I(p_1, I(p_2, D))$ which is equivalent to $I(p_1, R) \subseteq I(p_1, D)$. □

Observe that the converse does not hold: pattern subsumption does always entail pattern specialization.

Definition 3.8 (Pattern Table Equivalence)**.** *Two pattern tables P_1 and P_2 are* equivalent *with respect to a reference table R, denoted $P_1 \equiv_R P_2$, iff $I(P_1, R) = I(P_2, R)$.*

Observe that there might be several equivalent pattern tables for a given reference table R. For example, if R contains only one tuple t, then all non-empty pattern tables containing t and/or generalizations of t are equivalent w.r.t. R.

Example 3.7 (Patterns Subsumption and Equivalence)**.** *Consider the following set of patterns, w.r.t the constrained table T=$(Measures, R_1)$: p_1 = [F2,∗,∗,∗], p_2 = [F2,R1,∗,∗]. Observe that:*

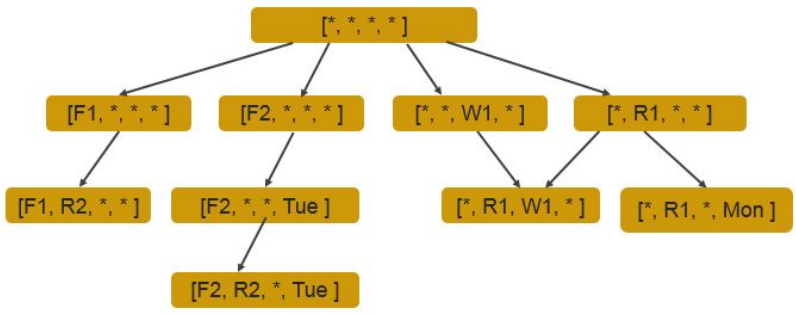
- *p_1 subsumes p_2 (and p_1 is a generalization of p_2)*
- *$p_1 \equiv_{R_1} p_2$ whereas p_1 and p_2 are not equivalent in general. For instance, equivalence does not hold for R_2.*

Pattern Lattice The subsumption relationship is a partial order since it is reflexive and transitive. The induced semi-lattice provides a convenient graphical tool for analyzing pattern tables as illustrated in the following example.

Example 3.8 (Pattern lattice)**.** *Consider the pattern table and its corresponding semi-lattice presented in Figure 3.1. An arrow linking a pattern p_1 to another pattern p_2 denotes that p_1 subsumes p_2. The top of the lattice contains the wildcard pattern while the leaves the most specific patterns.*

Floor	Room	Week	Day
*	*	*	*
F1	*	*	*
F2	*	*	*
F1	R2	*	*
*	*	W1	*
*	R1	W1	*
*	R1	*	*
*	R1	*	Mon
F2	*	*	Tue
F2	R2	*	Tue

Pattern table P



Pattern lattice P1

Fig. 3.1.: Pattern lattice

3.3.3 Pattern Covers

Given a constrained table T, T satisfies a pattern table P, means that P describes the completeness of some partitions in D, w.r.t R. The following definitions introduce particular pattern tables regarding T.

Definition 3.9 (Cover). *A pattern table P* covers *a constrained table T iff* for all patterns p *satisfied by T there exists a pattern $p' \in P$ subsuming p: $\forall p : T \models p \Rightarrow \exists p' \in P : p \sqsubseteq p'$.*

Observe that a pattern table P covering a constrained table T is not necessarily satisfied by T. In particular, any pattern table containing the universal pattern covers all constrained tables T (over the same set of reference attributes).

Any covering pattern table P for a constrained table T, contains at least all patterns satisfied by T (or their generalization). However, it may also includes other patterns that are not satisfied by T. We see with the following definition, the *strict cover* pattern table, that restricts P to only patterns satisfied by T.

Definition 3.10 (Strict Cover). *A pattern table P* strictly covers *a constrained table T iff P covers T and $P \models T$.*

It is simple to show that if P strictly covers constrained table $T = (D, R)$ and P' is equivalent to P w.r.t. R, then P' also strictly covers T.

Definition 3.11 (Reduced Pattern Table). *A pattern table P is* reduced *if there exists no pair of distinct patterns $p \in P$ and $p' \in P$ such that p is a generalization of p'.*

Proposition 3.3.4 (minimal cover). *For each constrained table T, there exists a unique reduced strict cover $\mathcal{P}^*(T)$ called the* minimal cover *of T.*

Proof: By contradiction using the notion of cover and subsumption. Suppose that there exist two minimal strict covers $\mathcal{P}^*(T)_1$ and $\mathcal{P}^*(T)_2$. Then there exists a pattern $p_1 \in \mathcal{P}^*(T)_1 - \mathcal{P}^*(T)_2$ and a pattern $p_2 \in \mathcal{P}^*(T)_2 - \mathcal{P}^*(T)_1$ such that $p_1 \sqsubseteq p_2$ (otherwise $\mathcal{P}^*(T)_2$ would not be a cover). Since $p_1 \neq p_2$ and by Proposition 3.3.3, we can conclude that $p_1 \sqsubset p_2$. By Definition 3.9 there must exist a third pattern $p_1' \in \mathcal{P}^*(T)_1$ such that $p_2 \sqsubseteq p_1'$ (otherwise $\mathcal{P}^*(T)_1$ would not be a cover). Then, we obtain $p_1 \sqsubset p_2 \sqsubseteq p_1'$ where p_1 and p_1' are two distinct patterns in $\mathcal{P}^*(T)_1$ and p_1' subsumes p_1. This is in contradiction with the claim that $\mathcal{P}^*(T)_1$ is a minimal cover. □

Example 3.9 (Pattern Covers). *Consider the constrained table in Table 3.5. Then, P_{strict} is a strict cover for T while $\mathcal{P}^*(T)$ is the minimum strict cover for T.*

Missing Partitions Patterns One interesting application of our model is the ability to capture partitions of missing data. This is allowed by the use of a reference which upper bounds the data in a given table. Interestingly, the notion of satisfaction can be generalized to the case of missing partitions by considering the complement of data w.r.t. the reference.

Floor	Room	Week	Day
F2	R1	∗	∗
F2	R2	∗	∗
∗	∗	W1	Mon
∗	∗	W1	Tue

Pattern strict cover P_{strict}

Floor	Room	Week	Day
F2	∗	∗	∗
∗	∗	W1	∗

Pattern minimal cover $\mathcal{P}^*(T)$

Tab. 3.6.: Constrained table T cover, strict cover and minimal cover pattern tables

Definition 3.12 (Missing Patterns Satisfaction)**.** *Let $T = (D, R)$ be a constrained table. A* satisfies *a missing pattern p^m, denoted $T \models_m p^m$, iff $I(p^m, R) \subseteq I(p^m, R - D)$. This entails $I(p^m, D) = \emptyset$. A constrained table T* satisfies *a missing pattern table P^m if T satisfies all patterns in P^m: $T \models_m P^m \equiv \forall p^m \in P^m : T \models_m p^m$.*

Based on this definition, all results and properties introduced for completeness patterns can be naturally generalized to missing patterns. For instance, each constrained table has a unique missing partition pattern minimal cover.

Example 3.10 (Missing Partition Patterns Minimal Covers)**.** *The minimal cover for the missing partition patterns of the constrained table of Example 3.6 is given as follows.*

Floor	Room	Week	Day
F1	R1	W2	Tue
F1	R2	W2	∗

$P^{*m}T$

3.4 The Pattern Algebra

The standard relational algebra is not sufficient for querying pattern tables due to the semantics of the wildcard $*$. The following example motivates the need for extending the relational algebra with specific operators taking into consideration the semantics of patterns.

Example 3.11 (Querying Pattern Tables)**.** *Consider the pattern table P from which we want to retrieve all patterns referring to week W1. The naive solution is to select the patterns in P based on an equality condition such as $\sigma_{Week='W1'}P$. However, such an expression would fail to return patterns where the fact Week='W1' is implicit, like p_0. Indeed, the standard relational semantics relies on an exact matching of the constants and has no interpretation of the wildcard.*

To define the pattern algebra, we introduce two operators that bridge the gap between pattern tables and data tables:

	Floor	Room	Week	Day
p_0	F2	$*$	$*$	$*$
p_1	$*$	$*$	W1	$*$
p_2	F1	R1	W2	$*$

Pattern table P

	Floor	Room	Week	Day
p_1	$*$	$*$	W1	$*$

$Q_0(P)$

- A *folding* operator, denoted with $\triangleright$, which, when applied on a constrained table T, returns its minimal cover ;

- An *unfolding* operator, denoted with $\triangleleft$, which, when applied on a pattern table P, returns the constrained table satisfying the P.

The table folding and unfolding operators rely on attribute folding and unfolding operators, respectively.

Definition 3.13 (Attribute unfolding $\triangleleft_A$)**.** *Let P be a pattern and a reference R. The* unfolding *of P w.r.t. R, denoted with $\triangleleft_A(P, R)$, generates an* equivalent *pattern table P' where all values of attributes $a_i \in A$ are constant values.*

The unfolding $\triangleleft_A(P, R)$ of a pattern table P on some attribute set A w.r.t. its reference table R can be defined by the following relational algebra expression:

$$\triangleleft_A (P, R) = \pi_{R.A, P.\neg A}(P \bowtie_{\theta_\triangleleft} R) \tag{3.1}$$

where $\theta_\triangleleft = \bigwedge_{a_j \in A}(P.a_j = * \vee P.a_j = R.a_j)$ for all attributes in A and $\pi_{R.A,P.\neg A}$ denotes the projection on attributes A of R and on all attributes of P except A.

Intuitively, unfolding a pattern table P over all its attributes generates a data table D satisfying P.

Example 3.12 (Unfolding patterns)**.** *Consider the pattern table P in Table 3.7 and the reference table of Example 3.2. Unfolding P on Floor expands the first pattern of P, [$*$,$*$,W1,$*$], into two patterns [F1,$*$,W1,$*$] and [F2,$*$,W1,$*$] ; the two remaining patterns are not modified since Floor is already a constant.*

Unfolding a minimal pattern table does not preserve minimality as one can observe from the previous example where [F2,$*$,W1,$*$] is subsumed by [F2,$*$,$*$,$*$].

Floor	Room	Week	Day
∗	∗	W1	∗
F2	∗	∗	∗
F1	R1	∗	Mon

P

Floor	Room	Week	Day
F1	∗	W1	∗
F2	∗	W1	∗
F2	∗	∗	∗
F1	R1	∗	Mon

$\triangleleft_{\{Floor\}}(P,R)$

Tab. 3.7.: A pattern table and the result of its unfolding

Operator fold $\triangleright_{a_i}$ is the inverse operator of $\triangleleft_{a_i}$ and *generalizes*, when possible, all subsets S of patterns $p \in S$ which are equal for all attributes except for attribute a_i into a single pattern $p_{a_i:*}$ with a wildcard value for attribute $a_i = *$:

Definition 3.14 (Attribute folding $\triangleright_{a_i}$)**.** *The* fold *operator* $\triangleright_{a_i}(P, R)$ *generates for a given pattern table* P *and reference table* R *an* equivalent *pattern table* $P' \equiv_R P$ *where there exists no pattern* $p_{a_i:*}$ *and subset* $S \subseteq P'$ *of specializations* p *of* $p_{a_i:*}$ *where* $p_{a_i:*} = *$ *and* $p_{a_i:*}$ *is equivalent to* S*:* $\nexists p_{a_i:*}, S \subseteq P' : p_{a_i:*} = * \wedge \{p_{a_i:*}\} \equiv_R S$.

To compute $\triangleright_A(P, R)$ we need to compute the set of all reference tuples missing in D

$$M = R - I(P, R) = R - D$$

Then, let $p_{a_i:*}$ denote the pattern obtained from p by replacing the constant value v_i of attribute a_i in p by a wildcard $*$. The semi-join expression $\overline{G}(a_i) = \sigma_{a_i \neq *}(P) \ltimes_{\theta_\triangleright} M$ where $\theta_\triangleright = \bigwedge_{i \neq j}(P.a_j = * \vee P.a_j = M.a_j)$ returns all patterns p in P which *cannot* be generalized on a_i: condition $\theta_\triangleright$ is true for all patterns $p \in P$ where the $p_{a_i:*}$ is incomplete (its instance in M is not empty).

Then $G(a_i) = \sigma_{a_i \neq *}(P) - \overline{G}(a_i)$ returns the set of patterns p where $p_{a_i:*}$ is complete and we can define the folding operator as follows:

$$\triangleright_{a_i}(P, R) = \sigma_{a_i = *}(P) \cup \overline{G}(a_i) \cup \{[a_i : *]\} \times \pi_{\neg a_i}(G(a_i))$$

Observe that if there is no pattern $p \in P$ where a_i has a constant value, then $\overline{G}(a_i)$ and $G(a_i)$ are empty and $\triangleright_{a_i}(P, R) = P$.

Example 3.13. *Consider the pattern table* P' *below.*

Observe that:

- $\sigma_{ro=*}(P')$ ={ *[F2,∗,∗,∗]*},
- $\overline{G}(ro)$ ={ *[F1,R1,∗,Mon]*} *and*

Floor	Room	Week	Day
F2	$*$	$*$	$*$
F1	R1	$*$	Mon
$*$	R1	W1	$*$
$*$	R2	W1	$*$

- $G(ro)$ ={[$*$,R1,W1,$*$], [$*$,R2,W1,$*$]} *and thus* $\{[a_i : *]\} \times \pi_{\neg a_i}(G(a_i)) = \{[*, *, W1, *]\}$.

Therefore, $\triangleright_{ro}(P', R)$ *returns* E.

As for unfold, the fold operation is associative and can be generalized on a set of attributes $A = \{a_1, a_2, ..., a_n\}$:

$$\triangleright_A(P, R) = \begin{cases} P & \text{for } A = \emptyset \\ \bigcup_{a_i \in A_h} (\triangleright_{a_i}(\triangleright_{A-a_i}(P, R), R)) & \text{otherwise} \end{cases} \tag{3.2}$$

In the following, $\triangleleft(P, R)$ (unfold all) and $\triangleright(P, R)$ (fold all) will denote the unfold and fold operations over *all* reference attributes in P (and R). Using this extended relational algebra RA_{ext}, we can now define a *pattern algebra* RA_{patt} which contains for each data table operator $op \in= \{\sigma, \pi, \bowtie, \cup, \cap, -\}$ its pattern-table counterpart $\widehat{op} \in \{\widehat{\sigma}, \widehat{\pi}, \widehat{\bowtie}, \widehat{\cup}, \widehat{\cap}, \widehat{-}\}$.

3.4.1 Pattern Operators

Let $T = (D, R)$ be a constrained table and Q be a relational query we would like to evaluate on T. To compute the minimal cover $\mathcal{P}^*(T')$ of the result $T' = Q(T)$, one can either evaluate Q over T then apply $\triangleright$ over T' or adopt an alternative solution by rewriting $Q(D)$ into a new query $\widehat{Q}$ and evaluate directly on a minimal cover $\mathcal{P}^*(T)$ to produce $\mathcal{P}^*(T')$ (see blue solid line in Figure 3.2). To do so, one needs to introduce set of pattern operators.

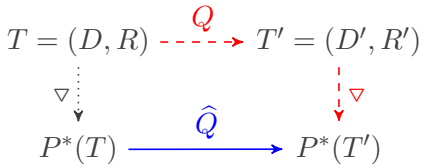


Fig. 3.2.: Pattern queries commutativity diagram

As mentioned in the previous section, we use the extended algebra $RA_{ext} = RA \cup \{\triangleright, \triangleleft\}$ to define a new *pattern algebra* RA_{patt} over pattern tables.

Definition 3.15 (Pattern Algebra operators)**.** *Let* P *and* P' *be two minimal covers of constrained tables* $T = (D, R)$ *and* $T' = (D', R')$*. Then we define the following pattern algebra* $RA_{patt} = \{\widehat{\sigma}, \widehat{\pi},$

$\widehat{\bowtie}, \widehat{\cup}, \widehat{\cap}, \}$ *where each operator* $\widehat{op}$ *is defined by using its relational counterpart* op *and operators* $\triangleright$ *and* $\triangleleft$*:*

$$\widehat{op}\,(P) = \triangleright\,(\,op\,(\triangleleft(P,R)),\ op\,(R)) \tag{3.3}$$

$$P\,\widehat{op}\,P' = \triangleright\,(\triangleleft(P,R)\ op\ \triangleleft\,(P',R'), R\,op\,R') \tag{3.4}$$

Observe that the previous definition does not include set difference. Instead, we introduce a *complement* operator $\overline{P}$ which generates the "complement" of a partition table P. Using this operator and intersection we define pattern difference as follows:

$$\overline{P} = \triangleright\,(R - \triangleleft(P,R), R) \tag{3.5}$$

$$P\widehat{-}P' = P\widehat{\cap}\overline{P'} \tag{3.6}$$

Theorem 3.4.1. RA_{patt} *is sound and complete: for all relational operators* $op \in \{\sigma, \pi, \bowtie, \cup, \cap\}$*, constrained tables* $T = (D,R)$ *and* $T' = (D',R')$ *with covers* P *and* P' *respectively, the following equations are true:*

$$\widehat{op}\,(P) = \triangleright\,(\,op\,(D),\ op\,(R)) \tag{3.7}$$

$$P\,\widehat{op}\,P' = \triangleright\,(D\,op\,D', R\,op\,R') \tag{3.8}$$

Proof: *We show that for all relational operators* $op \in \{\sigma, \pi, \bowtie, \cup, \cap, \}$*, constrained tables* $T = (D,R)$ *and* $T' = (D',R')$ *with covers* P *and* P' *respectively, equations 3.9 and 3.10 are true:*

$$\widehat{op}\,(P) = \triangleright\,(\,op\,(D),\ op\,(R)) \tag{3.9}$$

$$P\,\widehat{op}\,P' = \triangleright\,(D\,op\,D', R\,op\,R') \tag{3.10}$$

For proving soundness and completeness we use the two equalities $P = \triangleright(D,R)$ *and* $D = \triangleleft(P,R)$

Soundness:

$$\begin{aligned}\widehat{op}\,(P) &= \triangleright\,(\,op\,(\triangleleft(P,R)),\ op\,(R)) = \triangleright(\,op\,(D),\ op\,(R))\\ P\,\widehat{op}\,P' &= \triangleright\,(\triangleleft(P,R)\ op\ \triangleleft\,(P',R'), R\,op\,R')\\ &= \triangleright\,(D\,op\,D', R\,op\,R')\end{aligned}$$

The first equality in both equations is obtained by definitions 3.3 and 3.4, respectively.

Completeness:

$$\triangleright(\ op\,(D),\ op\,(R)) = \triangleright\,(\ op\,(\triangleleft(P,R)),\ op\,(R)) = \widehat{op}\,(P) \tag{3.11}$$

$$\triangleright(D\ op\ D', R\ op\ R') = \triangleright\,(\triangleleft(P,R)\ op\ \triangleleft\,(P',R'), R\ op\ R') \tag{3.12}$$

$$= P\ \widehat{op}\ P' \tag{3.13}$$

The last equality in both equations is obtained by definitions 3.3 and 3.4, respectively.

□

3.4.2 Rewriting Rules and Optimization

The standard semantics of pattern operators uses a unfolding step before applying the equivalent relational operator. In general, it is possible to avoid this step or to restrict it on the subset of attributes expressed in the pattern operator. We introduce, for each operator, the possible logical optimization which avoids unfolding on unnecessary attributes. This logical optimization is of course guided by the semantics of the pattern operator, as we will show.

Unary operators:

- Selection $\widehat{\sigma}$: Let θ_σ be the filtering condition and $A_\sigma = \{a_i, ..., a_k\}$ be the set of attributes used in the filtering condition.

$$\widehat{\sigma}_{\theta_\sigma}(P,R) = (\sigma_{\theta_\sigma}(\triangleleft(P,R)), \sigma_{\theta_\sigma}(R)) \tag{3.14}$$

$$= (\sigma_{\theta_\sigma}(\triangleleft_{A_\sigma}(P,R)), \sigma_{\theta_\sigma}(R)) \tag{3.15}$$

$$= (\pi_{R.A_\sigma, P.\neg A_\sigma}(P \bowtie_{\theta_\triangleleft} (\sigma_{\theta_\sigma}(R))), \sigma_{\theta_\sigma}(R)) \tag{3.16}$$

Observe that selection needs an unfold on all filtering attributes A_σ and the filtering condition θ_σ can be pushed inside the unfold operation. If the condition is a conjunction of equality predicates $a_i = c_i$, all pattern attributes $a_i \in A_\sigma$ can be replaced by $a_i : *$ and join can be replaced by semi-join:

$$\widehat{\sigma}_{\theta_\sigma}(P,R) = (\{[A_\sigma : *]\} \times (\pi_{P.\neg A_\sigma}(P \ltimes_{\theta_\triangleleft} (\sigma_{\theta_\sigma}(R))), \sigma_{\theta_\sigma}(R))) \tag{3.17}$$

The pattern selection does not preserve the pattern table minimality, a folding operation may be required to recover a minimality.

- Projection $\widehat{\pi}$: Let A_π denote the removed attributes.

$$\widehat{\pi}_{\neg A_\pi}(P, R) = (\pi_{\neg A_\pi} P, \pi_{\neg A_\pi}(R)) \tag{3.18}$$

  Projection does not require any a-priori unfolding phase, but, as we will explain, this may lead to ambiguity when removing attributes whose values are constant.

  Differently for pattern selection, pattern projection preserves minimality.

Binary operators: As for unary operators, the goal for binary operations is to avoid the generation of large intermediate tables through unfolding. As in the relational setting, one solution is to push selections into binary operations. The definitions are as follows:

- Cartesian product $\widehat{\times}$ is only defined if A_1 and A_2 are disjoint. Then, R_1 is by definition disjoint from R_1 and it is not necessary to do a fold and unfold.

$$(P_1, R_1)\widehat{\times}(P_2, R_2) = (P_1 \times P_2, R_1 \times R_2) \tag{3.19}$$

  If both arguments are minimal covers, the result is already a minimal cover.

- Natural join $\widehat{\bowtie}$ only requires unfold over the shared (joining) attributes:

  Let $A_\bowtie = A_1 \cap A_2 = \{a_1, ..., a_k\}$ be the set of join attributes:

$$(P_1, R_1)\widehat{\bowtie}(P_2, R_2) = (\triangleleft_{A_\bowtie}(P_1, R_1) \bowtie \triangleleft_{A_\bowtie}(P_2, R_2), R_1 \bowtie R_2) \tag{3.20}$$

  Observe that join can be implemented as a composition of Cartesian product, selection and projection.

- Union $\widehat{\cup}$ is only defined over two dimension tables sharing the same set of attributes A:

$$(P_1, R_1)\widehat{\cup}(P_2, R_2) = (P_1 \cup P_2, R_1 \cup R_2) \tag{3.21}$$

  We can also show that union doesn't require unfold if $R_1 = R_2 = R$. Observe that this definition of union is an opportunity to combine pattern covers generated independently for different data set fragments over the same reference.

- Intersection $\widehat{\cap}$ is only defined over two dimension tables sharing the same set of attributes A and requires unfold even if $R_1 = R_2$:

$$(P_1, R_1)\widehat{\cap}(P_2, R_2) = (\triangleleft(P_1, R_1) \cap \triangleleft(P_2, R_2), R_1 \cup R_2) \tag{3.22}$$

  Observe that the reference of intersection (and the following operators) is the *union* of the argument reference tables.

- Set difference $\widehat{-}$ is only defined over two dimension tables sharing the same set of attributes A and requires unfold even if $R_1 = R_2$:

$$\overline{P} = \triangleright\,(R - \triangleleft(P, R), R) \tag{3.23}$$

$$P\widehat{-}P' = P\widehat{\cap}\overline{P'} \tag{3.24}$$

3.4.3 Safe Projection

The standard semantics of projection may yield patterns that do not capture complete partitions. This occurs when the projected-out attributes contain constant values. To avoid this situation, we need to perform selection for filtering all patterns that, when projected, may describe incomplete partitions.

Example 3.14. *To illustrate the potential problem of projection, consider the pattern table P with a reference R, and the query $Q_1 = \widehat{\pi}_{Floor,Room}(P, R)$. Evaluating Q_1 with the standard semantics yields the patterns [F1,R1], [F1,R2] and [F2,R1]. Only [F1,R1] was obtained from a pattern describing a complete partition whereas the two others were obtained from patterns which are not complete w.r.t the Week and Day attributes.*

	Floor	Room	Week	Day
p_1	F1	R1	$*$	$*$
p_2	F1	R2	W1	$*$
p_3	F1	R2	W2	Tue
p_4	F2	R1	W2	$*$

P

	Floor	Room	Week	Day
t_1	F1	R1	W1	Mon
t_2	F1	R1	W1	Tue
t_3	F1	R1	W2	Mon
t_4	F1	R1	W2	Tue
t_5	F1	R2	W1	Mon
t_6	F1	R2	W1	Tue
t_7	F1	R2	W2	Mon
t_8	F1	R2	W2	Tue
t_9	F2	R1	W1	Mon
t_{10}	F2	R1	W1	Tue
t_{11}	F2	R1	W2	Mon
t_{12}	F2	R1	W2	Tue

R

In the case of projection, $\widehat{Q_2}$ simply remove attributes Week and Day. The result interpretation could be misleading, since no filtering were applied to keep only complete partitions regarding Week and Day. Indeed, the presence of [F2,R1] in the result may indicate that this partition is complete, which is not the case because only W2 measures are available.

To address this problem, we define a specific operator called *safe projection* which filters all complete partitions before projection. Let A_π denote the attributes removed by some projection. Then $\theta_\pi = \bigwedge_{a_i \in A_\pi}(a_i = *)$ filters all patterns which are incomplete for attributes A_π. The *safe projection* operator $\widehat{\pi}^*$ first folds all patterns over the attributes which are projected out and filters all incomplete dimensions before projection. This guarantees that the result only contains patterns corresponding to partitions which were complete w.r.t. the removed attributes:

$$\widehat{\pi}^*_{\neg A_\pi}(P, R) = (\pi_{\neg A_\pi}(\sigma_{\theta_\pi}(\triangleright_{A_\pi}(P, R))), \pi_{\neg A_\pi}(R)) \tag{3.25}$$

Observe also that if P is already a minimal cover, projection produces a minimal cover without requiring a final fold operation.

3.5 Pattern Queries

The pattern algebra is defined to allow querying pattern tables, to understand the extent of completeness and missing information in a data table.Given a constrained table T=(D,R), pattern queries allow checking the completeness and emptiness of data partition with respect to the reference, using only the pattern minimal cover $\mathcal{P}^*(T)$, and its complement ($\bar{\mathcal{P}}^*(T)$). Considering a relational query Q over the data table D, it is possible to check the complete and missing informations in the result $Q(D)$, by using a pattern query expressed in the relational algebra.

Proposition 3.5.1 (Pattern Query)**.** *Given a constrained table T=(D,R), the minimal pattern cover $\mathcal{P}^*(T)$ and a relational query Q, there exists a pattern query $\widehat{Q}$ defined with the pattern algebra, such as $\widehat{Q}(\mathcal{P}^*(T))$ produces the minimal cover of the query result $Q(D)$ with respect to a complete answer $Q(R)$.*

$$\widehat{Q}(\mathcal{P}^*(T)) = \mathcal{P}^*(Q(D), Q(R)) \tag{3.26}$$

Example 3.15. *In addition to the constrained table $T = (Measures, R)$ in Example 3.2, consider the table $T' = (area, \pi_{Floor,Room}R)$, and its minimal cover $\mathcal{P}^*(T')$, in Table 3.8.*

The following query retrieves the daily electrical consumption by square meter.

Floor	Room	Area
F1	R1	14
F1	R2	9

$Area$

Floor	Room
F1	R1
F1	R2
F2	R1

R'

Floor	Room
F1	$*$

$\mathcal{P}^*(T')$

Floor	Room
F2	$*$

$\bar{\mathcal{P}}^*(T')$

Tab. 3.8.: The Area constrained table ($Area$,R') and its minimal cover $\mathcal{P}^*(Area, R')$

```
SELECT Floor,Room,Week,Day, Kwh/area
FROM Measures m JOIN Area a
ON m.Floor = a.Floor and m.Room = a.Room
```

The Table 3.9 shows the query result $Q(Measures, Area)$. The following pattern queries compute pattern minimal covers for respectively complete and missing partition patterns in the query answer $Q(Measures, Area)$:

$$\mathcal{P}(Q(Measures, Area), Q(R, R')) = (\triangleleft_{Floor,Room}(\mathcal{P}^*(T), R) \bowtie \triangleleft_{Floor,Room}(\mathcal{P}^*(T'), R'), R \bowtie R')$$

$$\bar{\mathcal{P}}((Q(Measures, Area), Q(R, R'))) = (\triangleleft_{Floor,Room}(\bar{\mathcal{P}}^*(T), R) \bowtie \triangleleft_{Floor,Room}(\bar{\mathcal{P}}^*(T'), R'), R \bowtie R')$$

Floor	Room	Week	day	Kwh/m
F1	R1	W1	Mon	0.7
F1	R1	W1	Tue	0.9
F1	R1	W2	Mon	0.6
F1	R2	W1	Mon	1.7
F1	R2	W1	Tue	1.9

Query result $Q(Measures, Area)$

Floor	Room	Week	Day
F1	R1	w1	$*$
F1	R2	w1	Mon
F1	R1	W2	$*$

Complete $\mathcal{P}(Q(Measures, Area), Q(R, R'))$

Floor	Room	Week	Day
F2	R1	$*$	$*$
F1	R2	w2	$*$
F1	R1	w2	Tue

missing $\bar{\mathcal{P}}((Q(Measures, Area), Q(R, R')))$

Tab. 3.9.: The query result and its pattern tables

Observe that the pattern tables are not minimal covers, since the folding all operator can not be expressed in the relational algebra. Furthermore, we showed that a final folding operation is enough

for computing the minimal cover. In next chapter, we perform experiments to study the efficiency of this option.

3.6 Independent References

Fold ($\triangleright$) and unfold ($\triangleleft$) comprise costly joins with reference tables. In many real world settings, reference tables $R = R_1 \times R_2 \times ... \times R_n$ are defined by the Cartesian product of independent reference tables R_i corresponding to spatial, temporal and other dimensions. These reference tables R_i are obviously much smaller than the generated reference table R and introduce optimization opportunities for reducing unfolding/folding costs.

Definition 3.16 (Pattern Unfold $\triangleleft_A$)**.** *Let P be a pattern table with a reference table $R = R_1 \times R_2 \times ... \times R_n$. The unfolding of a pattern table P on some attribute set $A = A_1 \cup A_2 \cup ... \cup A_k$ where A_i are non-empty subsets of attributes of sub-table R_i (*wlg. *unfolding is done over the first k reference tables R_i) is defined as follows:*

$$\triangleleft_A(P, R) = \pi_{R_i.A_i, P.\neg A}(P \bowtie_{\theta^1_\triangleleft} R_1 \bowtie_{\theta^2_\triangleleft} R_2 \bowtie_{\theta^3_\triangleleft} ... \bowtie_{\theta^k_\triangleleft} R_k)) \tag{3.27}$$

where $\theta^i_\triangleleft = \bigwedge_{a_j \in A_i}(P.a_j = * \vee P.a_j = R_i.a_j)$ and $\pi_{R_i.A_i, P.\neg A}$ denotes the projection on attributes A_i of R_i and on all attributes of P except A.

Observe that $\triangleleft_A$ only joins with reference tables R_i which contain at least one attribute $a \in A$.

Definition 3.17 (Pattern Fold $\triangleright_{a_i}$)**.** *The folding of a pattern table P on some attribute a_i of a reference table R_j is defined as in definition 3.14, except that the missing tuples can directly be computed from the reference table R_j without considering the other tables: $M = R_j - \pi_{A_j}(D)$.*

Then, similarly to unfold, the fold operator $\triangleright_A$ only needs to access reference tables R_j which contain at least one attribute $a_i \in A$.

Definition 3.18 (Pattern Operators Optimization)**.** *If all attribute domains are independent and the input pattern tables are minimal covers, selection with equality, projection and Cartesian product can be expressed using the relational algebra (without $\triangleleft$) and generate minimal covers.*

3.7 Summary

In this chapter, we described a completeness approach under the partially closed world assumption. The following contributions have been made:

- We formalized a partition pattern model for creating data tables completeness annotations. The theoretical foundations guarantee the completeness of the pattern table (exhaustive set), and we prove the existence and unicity of a pattern minimal cover for each constrained data table. Pattern subsumption property defines a partial order on pattern sets, allowing for a lattice representation.

- The partition pattern model is used to create two types of annotations: complete partitions and missing partitions. All properties apply to both types, and each constrained table has strictly one completeness pattern minimal cover and one missing partitions patterns minimal cover (complement).

- We extended the relational algebra with two special operators. Fold data $\triangleright$, that allows folding a data table, following a set of attributes, into a pattern table. Folding on all reference attributes returns the pattern minimal cover. Unfold pattern $\triangleleft$, unfolds a pattern table following a set of attributes. Unfolding on all reference attributes returns the pattern table data instance.

- Using the extended relational algebra, we defined a pattern algebra P.A=$\{\widehat{\sigma},\widehat{\pi}, \widehat{\bowtie}, \widehat{\cup}, \widehat{-}\}$, that defines pattern queries over pattern tables. We prove the completeness and soundness of this Algebra regarding the SPJUD fragment of the relational algebra. For each relational query over a constrained data table, a corresponding pattern query allows computing the minimal pattern cover for the query result. The query result is annotated with completeness and missing partition patterns, independently from the relational query evaluation since the pattern query derives the result pattern data table minimal cover.

- Based on relational algebra rewriting rules and pattern properties, we formalized optimization definitions for pattern operators. An additional optimization was introduced with the independent reference case.

Recall the comparative table from the summary Section 2.5 of the state-of-the-art Chapter 2. Table 3.10 adds a line for our contribution.

Reference Model	Annotations	Formalism	Result	Guarantees
[Mot89]	Completeness statements	Query rewriting Using views	Decision task Yes/No	sound
[Lev96]	Local statements	Query Independent from updates	Decision task Yes/No	sound
[Lan+14]	physical anomalies logs	operations models	Labels:complete ,correct, phantom	sound and complete
[Raz+15]	completeness patterns	pattern algebra	Pattern sets	sound and complete
[Sun+17]	m-annotations	m-operators	polynome annotations	sound
[Fan+10a]	containment constrainsts	Query rewriting	decision task	sound
[**hannou2018**]	Partition Patterns	Pattern Algebra	Pattern Minimal covers	sound and complete

Tab. 3.10.: Comparative table for missing data representation models

4 Pattern Algebra Implementation and Experiments

> *Nothing in life is to be feared, it is only to be understood.*
>
> — **Marie Skłodowska-Curie**
> Nobel Prizes in Physics and Chemistry

Contents

4.1 Introduction

In the previous chapter, we introduced the notions of constrained tables T and minimal pattern covers $\mathcal{P}^*(T)$ for describing complete and missing partitions within dome data table w.r.t. some reference table. We proposed a pattern algebra over pattern covers to derive query results completeness annotations. The pattern algebra can be expressed in relational algebra except for the folding operator $\triangleright$ that implies a recursive table browsing. In this section, we describe an implementation of the pattern algebra in SQL. This implementation comprises two steps:

1. Translating query expressions based on the pattern algebra fragment $R.A. + \triangleleft$ (without $\triangleright$) into SQL queries. Remind that the results of these SQL pattern queries might not be minimal since no folding is applied.

2. Defining two algorithms for implementing the fold operator: a *fold data* algorithm that folds constrained data tables into minimal pattern covers; and a *fold pattern* algorithm that reduces pattern tables (such as SQL pattern query results) into minimal pattern tables.

We evaluate our implementation over the sensor network dataset of our campus Jussieu (to the Introduction Chapter 1). Experiments show the effectiveness and the efficiency of algorithms and pattern algebra queries, for describing the data tables and query results completeness.

This chapter is organized as follows. Section 4.2 shows, through examples, how pattern queries are translated into SQL. Section 4.3 presents the Algorithm $FoldData$implementing the fold operator applied on data tables. In Section 4.4, the second Algorithm $FoldPatterns$allows for folding pattern tables (as pattern query results) into minimal pattern tables without a preliminary unfolding step. Section 4.5 summarizes the experiments on our Campus Sensor data set. Finally, a chapter summary is given in Section 4.6.

4.2 Translating Pattern Algebra Expression into SQL

The pattern algebra computes pattern covers of query results without completely evaluating these queries on the data and their reference tables. All pattern algebra operators except the folding operator can be expressed in the relational algebra (Section 3.5 of Chapter 3). We have also shown that all pattern query expressions can be rewritten into an equivalent expression with a single final folding step to produce a minimal cover result[1] We therefore can apply the same rules for

[1]Note that the special operator safe projection $\widehat{\pi^*}$ is an exception to this rule since it requires as input a minimal cover to ensure the presence of all generalizations. We have shown in Section 3.4.3 that one solution to this restriction is to implement safe projection by a join query over a pattern table and its complement.

translating pattern algebra expressions into SQL as defined for the relational algebra. The following examples illustrates some examples for this translation.

Example 4.1. *Consider the constrained data table $T=(D,R)$ and its minimal pattern cover $\mathcal{P}^*(T)$, shown in Table 4.1.*

Floor	Room	Week	Day	kwH
F1	R1	W1	Mon	10
F1	R1	W1	Tue	12
F1	R1	W2	Mon	9
F1	R2	W1	Mon	15
F1	R2	W1	Tue	17
F2	R1	W1	Mon	11
F2	R1	W1	Tue	13
F2	R1	W2	Mon	8
F2	R1	W2	Tue	9

Data table D

Floor	Room	Week	Day
F1	R1	W1	Mon
F1	R1	W1	Tue
F1	R1	W2	Mon
F1	R1	W2	Tue
F1	R2	W1	Mon
F1	R2	W1	Tue
F1	R2	W2	Mon
F1	R2	W2	Tue
F2	R1	W1	Mon
F2	R1	W1	Tue
F2	R1	W2	Mon
F2	R1	W2	Tue

Reference table R

Floor	Room	Week	Day
F2	$*$	$*$	$*$
$*$	$*$	W1	$*$
$*$	R1	$*$	Mon

Minimal pattern cover $\mathcal{P}^*(T)$

Tab. 4.1.: A constrained data table T and its minimal pattern cover $\mathcal{P}^*(T)$

Consider the following pattern query $\widehat{Q}_1$ that filters patterns of floor F1, and aggregates on Day attribute(groups on Floor, Room, Week):

$$\widehat{Q}_1 = \widehat{\pi}_{Floor,Room,Week}(\widehat{\sigma}_{Floor=F1}(P,R)) \tag{4.1}$$

This pattern query implies two steps: a selection that restricts pattern tuples to those covering $Floor =' F1'$, and a projection that removes attribute Day. The reference R is the Cartesian product of two independent reference tables $R = R_S \times R_T$, which enables additional optimizations as described in Section 3.6. We first translate the PA expression $\widehat{Q}_1$ into an equivalent Relational Algebra (RA) expression Q_1 as an intermediate step before translating into SQL. We replace the pattern selection by its RA definition, using the spatial reference only, and then apply the pattern projection as a relational projection:

$$Q_1 = \pi_{R_S.Floor,P.Room,P.Week}(P \bowtie_{cond} \sigma_{floor=F1}(R_S))) \tag{4.2}$$

where $cond = (R_S.Floor = P.Floor \vee P.Floor = *) \wedge (R_S.Room = P.Room \vee P.Room = *)$.

The relational algebra query Q_1 can then directly be translated into the following SQL query:

```
SELECT  '*', P.Room, P.Week
FROM P p JOIN R_S  r
ON (p.Floor=r.Floor OR p.Floor= '*') AND (p.Room=r.Room OR p.Room= '*')
```

```
    WHERE r.Floor = F1
```

Listing 4.1: SQL Query for Q_1

The query result is shown in table 6.2.

Floor	Room	Week
*	*	W1
*	R1	*

Tab. 4.2.: Result of Query $\widehat{Q}_1$

Example 4.2 (the minimal cover problem)**.** *Suppose that we modify the query Q_1 from the previous example, by replacing the selection filtering condition by (Floor = F2):*

$$Q_2 = \widehat{\pi}_{Floor,Room,Week}(\widehat{\sigma}_{Floor=F2}(P,R)) \tag{4.3}$$

The translation of this pattern query is analogous to the previous translation, and the result is shown in Table 4.3.

Floor	Room	Week
*	*	*
*	*	W1
*	R1	*

Tab. 4.3.: Result of Query $\widehat{Q}_2$

Notice that the result of the query $\widehat{Q}_2$ is not minimal. The first pattern (∗,∗,∗) tuple means that the query result is complete and suffices for describing the completeness. The remaining pattern tuples are subsumed by the wildcard pattern and need to be removed from the result. This task should be achieved by the fold operator ▷ which is not implemented in SQL. In the following, algorithms implementing this operator will be proposed.

4.3 Folding Data

The folding data algorithm to implements the fold operator $\triangleright_A$ presented in Section 3.3 (Chapter 3). Given a constrained table $T = (D, R)$, folding $\triangleright_A(T)$ a strict cover $\mathcal{P}^*(T)$ according to the set of reference attributes A. In case A represents the entire set of attributes in T, the folding algorithm produces the minimal cover of T.

The intuition behind the algorithm $FoldData$ is to systematically check the completeness of data partitions, starting by the most "tuple-covering" partition. Each data partition corresponds to a candidate pattern, which is only *satisfied* if $I(p, R) \subseteq I(p, D)$. The $FoldData$algorithm browses the pattern subsumption lattice L_D generated by the active attribute domains in the data table D. It starts from the wildcard pattern $[*]$ (level 0) and explores *top-down* and *breadth-first* the lattice. Each level l corresponds to all candidate patterns p with l constants (remaining attributes being valued as wildcards $*$). The candidate patterns are then subject of completeness satisfaction checking, to decide whether it belongs to the pattern cover $\mathcal{P}^*(T)$ or not.

Satisfaction check: For simplifying completeness checking procedure, we assume that the data table D contains exclusively tuples from the reference table R, i.e. $D \subseteq R$. In this case, a data table satisfies a pattern p iff $I(p, D) = I(p, R)$ and the algorithm $FoldData$ can check pattern satsifaction by comparin the cardinality of $I(p)$ in D and R.

Search Space Pruning: If a pattern p is satisfied at some level l, all its specializations $p\prime$ are also satisfied, but not required to appear in the resulting pattern table. To optimize browsing, the algorithm automatically prunes the complete partition (I(p,D)) after p is generated, preventing additional unnecessary checking, in next levels $l\prime > l$.

Algorithm $FoldData$ uses the following functions:

- $powerSet(A, N)$ produces all subsets of A of cardinality N.
- $patterns(A, D)$ produces for a set of attributes A, all patterns $\pi_A(D) \times \{[*]\}$
- $checkComp(p, D, R)$ checks if $I(p, D) = I(p, R)$ (True if $|I(p, D)| = |I(p, R)|$).
- $prune(P, D)$ deletes from D all tuples satisfied by patterns $p \in P$.

Observe that operations $checkComp$ and $patterns$ can be implemented by standard SQL queries. In particular, $patterns$ is a simple projection on D and $checkComp$ can be implemented by comparing the result of two $count$-queries on D and R (we suppose that $D \subseteq R$).

The exploration strategy is as follows: the algorithm starts by checking the universal pattern (level 0). If this pattern is satisfied, it stops (all tuples in D are pruned). Otherwise it checks all patterns generated by one attribute (level 1). After finishing this level, the algorithm can again safely delete all tuples in D which are subsumed by the found patterns and proceed to the next level until D is empty.

As one can observe, this process is guided by the schema of the reference (line 2) and follows a breadth-first traversal of the lattice in increasing the size of the subsets of attributes (line 3).

At each iteration (lattice depth), a set of pattern candidates are derived: first relevant tuples are retrieved from R (line 4), then extended with the necessary $*$to obtain valid patterns (line 5). Each pattern is then verified (line 6) and added to the current list (line 7), in case it is satisfied. At the time, this same pattern is marked in the to-be-excluded set so that subsequent explorations do not derive patterns that are subsumed by it (line 8). This pruning is ensured by passing to the next exploration phase an updated reference where all all tuples which have already generated patterns are removed (line 12), guaranteeing minimality of the derived pattern-set.

Algorithm 1: Algorithm $FoldData$

Data: constrained table $T = (D, R)$, attribute set A
Result: minimal cover $\mathcal{P}^*(T)$

```
1  P := ∅ ;
2  for level := 0 to |A| do
3  |   X := ∅ ;
4  |   for B ∈ powerSet(A, level) do
5  |   |   for p ∈ patterns(B, D) do
6  |   |   |   if checkComp(p, D, R) then
7  |   |   |   |   P := P ∪ {p} ; X := X ∪ {p} ;
8  |   |   |   end
9  |   |   end
10 |   end
11 |   prune(X, D) ;
12 end
13 return P
```

Example 4.3 (Fold Data running steps)**.** *Recall the constrained table in the Example 4.1. Let us explain how the minimal cover $\mathcal{P}^*(T)$ is obtained using the $FoldData$ algorithm (table 4.4).*

At level 0, only the wildcard pattern in examined. This pattern is not satisfied, because the primary data partition, does not contain all referenced spatio-temporal locations. At level 1, we generate attributes combinations of length 1, which gives $B = \{\{Floor\}, \{Room\}, \{Week\}, \{Day\}\}$. This step requires checking as much candidate pattern as the total size of their respective active domains in the data *table. Data partitions satisfy two patterns $[F2, *, *, *]$ and $[*, *, W1, *]$. After this completeness checking, the algorithms prunes all data partitions that are assessed to be complete (all data tuples satisfying $Floor =' F2' or Week =' W2'$). At level 2, all attributes combinations of length 2 are checked: $B = \{\{Floor, Room\}, \{Floor, Week\}, \{Floor, Day\}, \{Room, Week\}, \{Room, Day\}, \{Week, Day\}\}$ for the remaining tuple. Only values of the data table active domain are used to create candidate patterns and among the 6 candidate patterns, only pattern $[*, R1, *, Mon]$ is satisfied. Notice that the pruning steps are performed on the data table and the reference table, because there is no need to keep reference tuples that have already been matched with complete data partitions. The algorithm stops once the data table is empty.*

Level 0

Floor	Room	Week	Day	KwH
F1	R1	W1	Mon	10
F1	R1	W1	Tue	12
F1	R1	W2	Mon	9
F1	R2	W1	Mon	15
F1	R2	W1	Tue	17
F2	R1	W1	Mon	11
F2	R1	W1	Tue	13
F2	R1	W2	Mon	8
F2	R1	W2	Tue	9

Floor	Room	Week	Day
F1	R1	W1	Mon
F1	R1	W1	Tue
F1	R1	W2	Mon
F1	R1	W2	Tue
F1	R2	W1	Mon
F1	R2	W1	Tue
F1	R2	W2	Mon
F1	R2	W2	Tue
F2	R1	W1	Mon
F2	R1	W1	Tue
F2	R1	W2	Mon
F2	R1	W2	Tue

*	*	*	*

Level 1

Floor	Room	Week	Day	KwH
F1	R1	W1	Mon	10
F1	R1	W1	Tue	12
F1	R1	W2	Mon	9
F1	R2	W1	Mon	15
F1	R2	W1	Tue	17
F2	R1	W1	Mon	11
F2	R1	W1	Tue	13
F2	R1	W2	Mon	8
F2	R1	W2	Tue	9

Floor	Room	Week	Day
F1	R1	W1	Mon
F1	R1	W1	Tue
F1	R1	W2	Mon
F1	R1	W2	Tue
F1	R2	W1	Mon
F1	R2	W1	Tue
F1	R2	W2	Mon
F1	R2	W2	Tue
F2	R1	W1	Mon
F2	R1	W1	Tue
F2	R1	W2	Mon
F2	R1	W2	Tue

F1	*	*	*
F2	*	*	*
*	R1	*	*
*	R2	*	*
*	*	W1	*
*	*	W2	*
*	*	*	Mon
*	*	*	Tue

Level 2

Floor	Room	Week	Day	KwH
F1	R1	W2	Mon	9

Floor	Room	Week	Day
F1	R1	W2	Mon
F1	R1	W2	Tue
F1	R2	W2	Mon
F1	R2	W2	Tue

F1	R1	*	*
F1	*	W2	*
F1	*	*	Mon
*	R1	W2	*
*	R1	*	Mon
*	*	W2	Mon

Level 3

Floor	Room	Week	Day	KwH

Floor	Room	Week	Day
F1	R1	W2	Tue
F1	R2	W2	Mon
F1	R2	W2	Tue

Tab. 4.4.: Example: Folding data algorithm running steps

Complexity Analysis In the worst case, $FoldData$ explores (almost) the whole pattern lattice L_D that is generated by all attribute/value combinations in the fragment. The number of patterns $size(L_D)$ of L_D depends on the active attribute domains in the fragment D and the number of attributes $n = |A|$: $size(L_D) = \sum_{i=1}^{n}(C_i^n) * D_i$ where D_i is the maximum size of the Cartesian product of the active domain of i attributes in the data table. The size of the source table influences the cost of checking pattern satisfaction. We also can estimate an upper bound for the fragment summary size as follows. Each tuple in the fragment generates between 0 (for tuples that are subsumed by patterns generated by other tuples) and k patterns, where k is the number of identifiers of the tuple in the source (reference) table. In the worst case, the size of the generated summary is $max_{1\leq i\leq n}C_i^n \simeq C_{n/2}^n$ times the size of the fragment where $n = |A|$ is the number of attributes in $|A|$. Such a worst case scenario corresponds to the particular case of random missing data with highly correlated attribute values and no pruning opportunities. If all attributes are necessary to identify any tuple in the source table (independent attribute domains), the fragment summary cannot get bigger than the fragment. As we show in our experiments, real-world data generally follows more regular incompleteness schemes, which increase the compression rate and folding performance.

Proposition 4.3.1 (Correctness)**.** *Algorithm $FoldData$ generates the unique minimal cover.*

Proof: *We can show that each level only generates patterns that are not subsumed by the patterns of the previous levels (minimality), the algorithms stops when all data tuples are subsumed by at least one generated pattern (cover) and the algorithm does not generate any pattern which does not cover at least one data tuple (strict cover).* □

4.4 Folding Patterns

The minimality property is not preserved after applying pattern algebra operators. Pattern selection for example, needs to partially unfold minimal pattern covers over the attributes in the filtering condition. The output of these operators has to be rectified in terms of minimality by applying a fold operator (see algebra definitions in Section3.3 Chapter 3 for detailed explanations). To do this, one could use Algorithm $FoldData$ on the data obtained applying a complete unfold. This unfolding/folding strategy obviously is inefficient, in particular for pattern tables with a high compactness ratios ($|D| \div |P| >> 1$).

A pattern table P is not a minimal cover for two main reasons:

Cover issue: it might not be a cover, *i.e.* there might exist a subset of patterns $S \subseteq P$ which could be merged into a single generalized equivalent pattern $p \notin P$.

Minimality issue: The pattern table might not be reduced, *i.e.* contain two patterns p_1 and p_2 such that one pattern is subsumed by the other. The existence of specializations entails a redundancy contradicting the minimality.

The Algorithm $FoldPatterns$ deals with these issues in two separate steps. The first *merge* step solves the *Cover issue* by recursively checking for each pattern if it can be replaced by a more general pattern (see details below). The *Minimality issue* is solved in a second *reduce* step.

The *merge* step (lines 1 to 8) proceeds by checking if the instance $I(S, R)$ of a subset $S \subseteq P$ is equivalent to the instance $I(p, R)$ of a pattern $p \notin P$. The basic idea is to explore the patterns in P *bottom-up* starting from the most specialized pattern (at the lowest level) and by recursively merging sets S of patterns which differ only on the constant of one attribute. As soon as S can be merged into one pattern p, we add p to P, without deleting S. S can not be deleted at this step, because some patterns can be used to merge with other patterns outside S.

Regarding the Minimality issue, Algorithm $FoldPatterns$ *reduces* P by removing all patterns $p_1 \in P$ which specialize another pattern $p_2 \in P$. These patterns include those which were merged in the first step. This can be done by a simple auto-join on P (lines 14 to 24). This *reduce* step is implemented by the second outer loop in $FoldPatterns$. Algorithm $FoldPatterns$ uses the following functions:

- $getPatt(P, level)$ returns all patterns $p \in P$ with $level$ constant attributes.
- $isGen(p1, p2)$ checks if $p1$ is a generalization of $p2$ (Definition 3.4 in Section 3.2 from Chapter 3).
- $gen(p, a)$ generalizes pattern p by replacing the constant attribute a by a wildcard.
- $getConstAttrs(p)$ finds all constant attributes of pattern p.
- $checkComp(X, p, T, R)$ checks if the instance of pattern set X in data table T is equal to the instance of pattern p in R.
- $getSimPatt(P, p, a)$ returns all patterns in P which differ from p by a different constant value for attribute a.

Example 4.4 (Fold Pattern running steps)**.** *Consider the problem of folding the pattern table regarding the reference table in Table 4.5. This table represents a strict but not minimal cover for our constrained table T in 4.1 since there exist patterns specializing other patterns in the same table.*

We run the Algorithm $FoldPatterns$ on the pattern table P. The Table 4.6 shows step by step the merge and reduce stages of the algorithm. Merging patterns starts from the lowest granularity:

Algorithm 2: Algorithm $FoldPatterns$

Data: pattern table P, reference table R, data table T, attribute set A
Result: minimal cover $\mathcal{P}^*(I(P,R))$

```
for level := |A| to 0 do
    for p ∈ getPatt(P, level) do
        for a ∈ getConstAttrs(p) do
            p_{a_i:*} := gen(p, a)
            S := getSimPatt(P, p, a)
            if p_{a_i:*} ∉ P) then
                if checkComp(S, p_{a_i:*}, T, R) then
                    P := P ∪ {p_{a_i:*}}
                end
            end
        end
    end
end
for level1 := 0 to |A| do
    for p1 ∈ getPatt(P, level1) do
        for level2 := level1 + 1 to |A| do
            for p2 ∈ getPatt(P, level2) do
                if isGen(p1, p2) then
                    P := P − {p2}
                end
            end
        end
    end
end
return P
```

Floor	Room	Week	Day
F2	∗	∗	∗
F2	R1	∗	∗
F2	∗	W1	∗
F1	R1	W1	∗
F1	R2	W1	∗
F1	R1	W2	∗
F1	R1	W2	Mon

Pattern table P

Floor	Room	Week	Day
F1	R1	W1	Mon
F1	R1	W1	Tue
F1	R1	W2	Mon
F1	R1	W2	Tue
F1	R2	W1	Mon
F1	R2	W1	Tue
F1	R2	W2	Mon
F1	R2	W2	Tue
F2	R1	W1	Mon
F2	R1	W1	Tue
F2	R1	W2	Mon
F2	R1	W2	Tue

Reference table R

Floor	Room	Week	Day
F2	∗	∗	∗
∗	∗	W1	∗
F1	R1	W2	∗

Minimal pattern cover $\mathcal{P}^*(T)$

Tab. 4.5.: Reducing a pattern table into a minimal table

- *Merge 1:* $[F1, R1, W1, *]$ *and* $[F1, R2, W1, *]$ *are merged to constitute* $[F1, *, W1, *]$.
- *Merge 2:* $[F1, *, W1, *]$ *and* $[F2, *, W1, *]$ *constitute a full complete partition* $[*, *, W1, *]$.
- *Merge 3:* $[F2, *, W1, *]$ *and* $[F2, *, W2, *]$ *merge to generate* $[F2, *, *, *]$.

Notice that $[F2, *, W1, *]$ *participates in two merging operations, which explains why patterns are not replaced by their generalization immediatly after merge. Merging patterns creates new generalization patterns but does not entail the deletion of patterns at their origin. This contributes to the increase of the number of subsumed patterns, contradicting the propriety of minimality. The second reduce phase deletes the redundant specializations.*

Observe for example in Table 4.6 that the pattern [F2, ∗, ∗, ∗] is a generalization of three other patterns [F2, R1, ∗, ∗], [F2, ∗, W1, ∗], [F2, ∗, W2, ∗], which leads to three reducing operations.

Analysis: In the worst case, the reduce step generates $O(|P|^2)$ generalization tests. Similar to the top-down algorithm $FoldData$, the size $size(L_P)$ of the pattern lattice L_P explored by the second steps can be estimated by $size(L_P) = \sum_{i=1}^{n}(C_n^i) * D_i$ where n is the number of attributes, and D_i is the maximum size of the Cartesian product of the active domain of i attributes in the *pattern* table. Since the D_i over the pattern table P is in general much smaller than D_i over the data set D, computing the minimal cover for P (without unfold) is in general much more efficient than computing the minimal cover on the data set D. This observation is confirmed by our experiments.

Proposition 4.4.1. *Algorithm* $FoldPatterns$ *is correct.*

Floor	Room	Week	Day
F2	*	*	*
F2	R1	*	*
F2	*	W1	*
F1	R1	W1	*
F1	R2	W1	*
F1	R1	W2	*
F1	R1	W2	Mon

Merge 1

Floor	Room	Week	Day
F2	*	*	*
F2	R1	*	*
F2	*	W1	*
F1	R1	W1	*
F1	R2	W1	*
F1	*	W1	*
F1	R1	W2	*
F1	R1	W2	Mon

Floor	Room	Week	Day
F2	*	*	*
F2	R1	*	*
F2	*	W1	*
F1	R1	W1	*
F1	R2	W1	*
F1	*	W1	*
F1	R1	W2	*
F1	R1	W2	Mon

Merge 2

Floor	Room	Week	Day
F2	*	*	*
F2	R1	*	*
F2	*	W1	*
F1	R1	W1	*
F1	R2	W1	*
F1	*	W1	*
F1	R1	W2	*
F1	R1	*	*
F1	R1	W2	Mon

Floor	Room	Week	Day
F2	*	*	*
F2	R1	*	*
F2	*	W1	*
F1	R1	W1	*
F1	R2	W1	*
F1	*	W1	*
F1	R1	W2	*
F1	R1	W2	Mon

Merge 3

Floor	Room	Week	Day
F2	*	*	*
F2	R1	*	*
*	*	W1	*
F2	*	W1	*
F1	R1	W1	*
F1	R2	W1	*
F1	*	W1	*
F1	R1	W2	*
F1	R1	*	*
F1	R1	W2	Mon

Floor	Room	Week	Day	
F2	*	*	*	
F2	R1	*	*	x
*	*	W1	*	
F2	*	W1	*	x
F1	R1	W1	*	x
F1	R2	W1	*	x
F1	*	W1	*	x
F1	R1	W2	*	x
F1	R1	*	*	x
F1	R1	W2	Mon	

Reduce

Floor	Room	Week	Day
F2	*	*	*
F1	*	W1	*
F1	R1	W2	Mon

Tab. 4.6.: Folding pattern algorithm running steps on table (p, R)

Proof: *We can show that after the reduce step, a pattern p can only be generated if there exists an attribute a and a subset of patterns $S \subseteq P$ such that p generalizes all patterns in S on attribute a and S is equivalent to p. Then by following a recursive bottom-up strategy we guarantee that all possible generalizations are tested. The second reduce step guarentees minimality.* □

4.5 Experiments

We created an experimental protocol to evaluate the following features:

- Pattern table *compactness* (Section 4.5.2).
- Folding algorithms performance (Section 4.5.2 and Section 4.5.4)
- Query result completeness patterns generation performance (Secion 4.5.4).

We ran our experiments on a standard Linux machine equipped with a 2.4 GhZ dual-core CPU, 8GB of RAM and 350 GB of standard storage. The algorithms are implemented in Python 2.6 whereas data and patterns were managed in PostgreSQL [Sto+86] and accessed using the psycopg2+ library of Python. For storing and querying data and pattern tables between the database and our programs rely on the psycopg2 open source library.

4.5.1 Datasets

We use both a *real-world* and a *synthetic* dataset. The real-world dataset corresponds to sensor measurements of different kinds as electricity, heating, or temperature, collected during one year at our University campus. The Smart Campus scenario was introduced in Introduction Chapter 1. This dataset features both spatial and temporal incompleteness since not at all parts of the campus are covered by sensors, and many of the sensors operate erratically. The synthetic dataset is generated from the real one by introducing more randomness for the purpose of studying the impact of data distribution on pattern compactness.

We restrict the study to measures pertaining to temperatures collected in 12 out of 96 buildings and refer to this data with **Temp**. We build two reference tables with different spatial coverage and an identical temporal span. The first reference, noted R_{All}, includes all spatial locations of the campus regardless of the existence of temperature sensors. The second reference, noted R_{Temp},

restricts on localities equipped with a *temperature* sensor, that is, localities present in **Temp**. The schema of the data and the reference tables together with their sizes are reported in 4.7.

$\mathbf{Temp}(\underline{building, floor, room, year, month, day, hour}, value)$

$\mathbf{Loc}(\underline{building, floor, room})$ $\mathbf{Cal}(\underline{year, month, day, hour})$

$Sch(R_{All}) = Sch(R_{Temp}) = Sch(\mathbf{Loc}) \cup Sch(\mathbf{Cal})$

variant x	$\lvert Loc_x \rvert$	$\lvert Cal_x \rvert$	$\lvert R_x \rvert = \lvert Loc_x \rvert \times \lvert Cal_x \rvert$
all	10,757	8,760	94,231,320
$Temp$	2,810	8,760	24,615,600

Tab. 4.7.: Size of reference tables R_{all} and R_{Temp}

Naturally, the choice of the reference dataset has an impact on the pattern derivation performance. We start by investigating the performance by studying the variation of the compaction ratio when varying the size of the data and its associated reference. To do so, we build two smaller data tables by restricting Temp *spatially*, selecting only the measures of one building, and *temporally*, by keeping the measures covering a single month. We refer to the resulting tables with, respectively, T_OneBlg and T_OneMon. Their cardinalities are reported in Table 4.8 together with their completeness ratio $CR(ds, R) = |ds|/|R_x|$ regarding their references R_x. We denote by R^{ds}_{All} and R^{ds}_{Temp} the reference tables obtained by using the same spatial or temporal restriction as the dataset ds. For example, $R^{T\_OneBlg}_{All} = \sigma_{building='25'}(R_{All})$.

dataset ds	$\lvert ds \rvert$	$CR(ds, R^{ds}_{all})$	$CR(ds, R^{ds}_{Temp})$
Temp	1,321,686	1.4%	5.36%
T_OneBlg	341,640	21.43%	21.43%
T_OneMon	88,536	1.4%	4.23%

Tab. 4.8.: Sizes and completeness ratio

As expected, the closer data is to its reference, the better is the completeness ratio. We observe that the spatial restriction allows for achieving the highest completeness ratio (21.43%).

4.5.2 Pattern table generation

We perform a preliminary experiment to measure the completeness of different datasets D and the compactness ratio of the corresponding complete and missing pattern tables P and $\overline{P}$. We define the *compactness ratio* $\Gamma(P, D)$ of a completeness pattern table P by the ratio $|D|/|P| \in [1, D]$ where $|P|$ is the size (cardinality) of the pattern table and $|D|$ is the size of the data table. The *completeness* $\Omega(D)$ of a measure table D with respect to its reference table R is defined by the ratio $|D|/|R| \in [0, 1]$. In addition to $Temp$, we consider a subset $OneBlg$ of all measures in building 2232

and a subset $OneMonth$ of all measures collected during January. The corresponding reference tables are built by extracting the reference subsets corresponding to the same building and month respectively.

D	$\Omega(D)$	$\lvert D\rvert$	$\lvert P\rvert$	$\Gamma(P, D)$	$\lvert\overline{P}\rvert$	$\Gamma(\overline{P}, R - D)$
$Temp$	5.36%	1,321,686	11,269	117	10,777	2,161
$OneBlg$	21.43%	341,640	39	8760	55	22,776
$OneMonth$	4.23%	88,536	119	744	370	5,390

Tab. 4.9.: Patterns tables sizes and compactness ratios

The completeness ratio is significantly higher for building 2232 than for the overall campus average which can be explained by a better sensor coverage in this building. Completeness is not uniformly distributed over months of the year, many sensors experience periods of no recording activity (failure) or are installed after January, leading to a lower monthly completeness rate than for other months. Observe in Table 4.9 that the completeness ratio and the data size are not sufficient to explain the compactness ratio since the compactness ratio is governed by the distribution of missing data over the spatial and temporal localities.

We define two "real" measure datasets **Temp_0** (empty temperature table), **Temp_**50% (containing the first 50% of $Temp$ sorted by time and space), **Synthetic_0**% (empty table) and two "synthetic" datasets **Synthetic_30**% (containing a random 30% sample of the reference table). Starting from these four datasets with a fixed completeness ratio, we build two series of datasets obtained by successively inserting and deleting tuples from the dataset. The insertion and deletions follow two strategies: i) a sequential strategy which selects the (inserted or deleted) tuples using their spatial and temporal domain order preserving the original data distribution in **Temp_0**% and **Temp_**50%, and ii) a random strategy which randomly picks these tuples for **Synthetic_0**% and **Synthetic_30**%.

Figures 4.1 and 4.2 depict the evolution of compactness w.r.t. completeness for each dataset. In the synthetic datasets (Figures 4.1), the compactness of a random dataset with 30% completeness evolves symmetrically in both directions (insertion and deletion): successive insertions/deletions generate/remove tuples which give rise to new patterns. At some point, these insertions/deletions will cause the merging of fine-grained patterns to coarser-grained ones increasing the compactness ratio to achieve maximum compactness at both extremities. In the real datasets, we observe the same trend with a lower amplitude for a dataset with 50% initial completeness: insertions lead to faster completion of the partial partitions (thanks to ordering sensitive updates) and thus to the faster derivation of coarser patterns without deriving all their subsumed patterns.

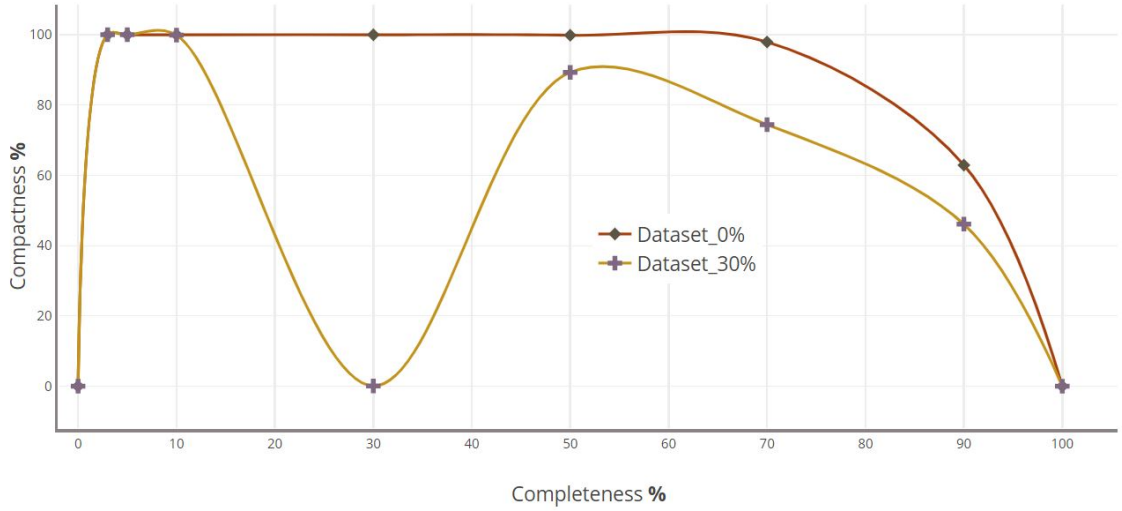


Fig. 4.1.: Synthetic datasets: Data missing randomly

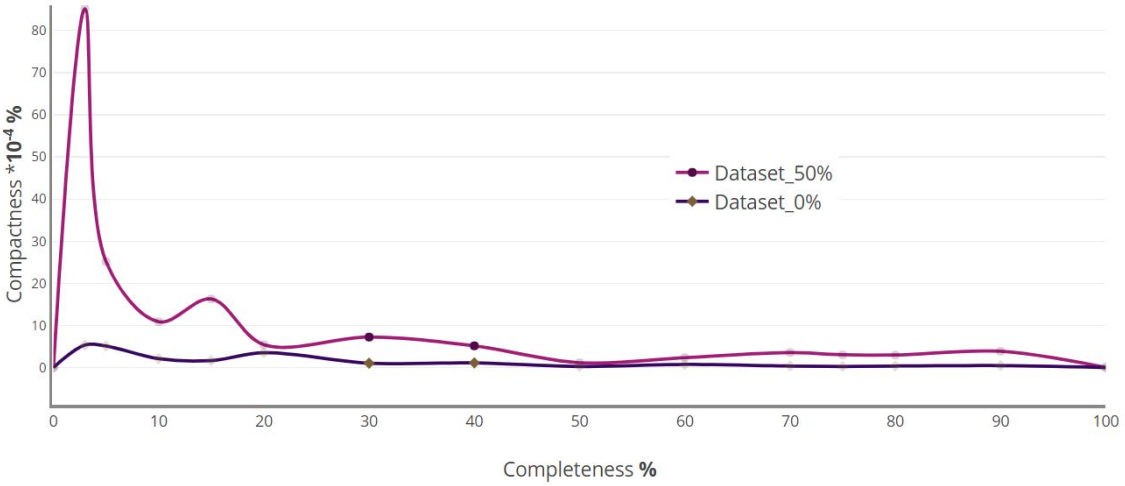


Fig. 4.2.: Real datasets: missing data following sensor failures

Performance In the following experiment we evaluate the performance of algorithm $FoldData$. From the original dataset Temp, we derived 30 datasets grouped into three categories, each with approximately the same completeness rate, but different dataset sizes.

Figure 4.3 shows the running time of $FoldData$ for all datasets according to the number of generated patterns. Categories are represented by points of different colors ($orange = 15\%$, $violet = 10\%$ and $green = 3\%$ completeness rate).

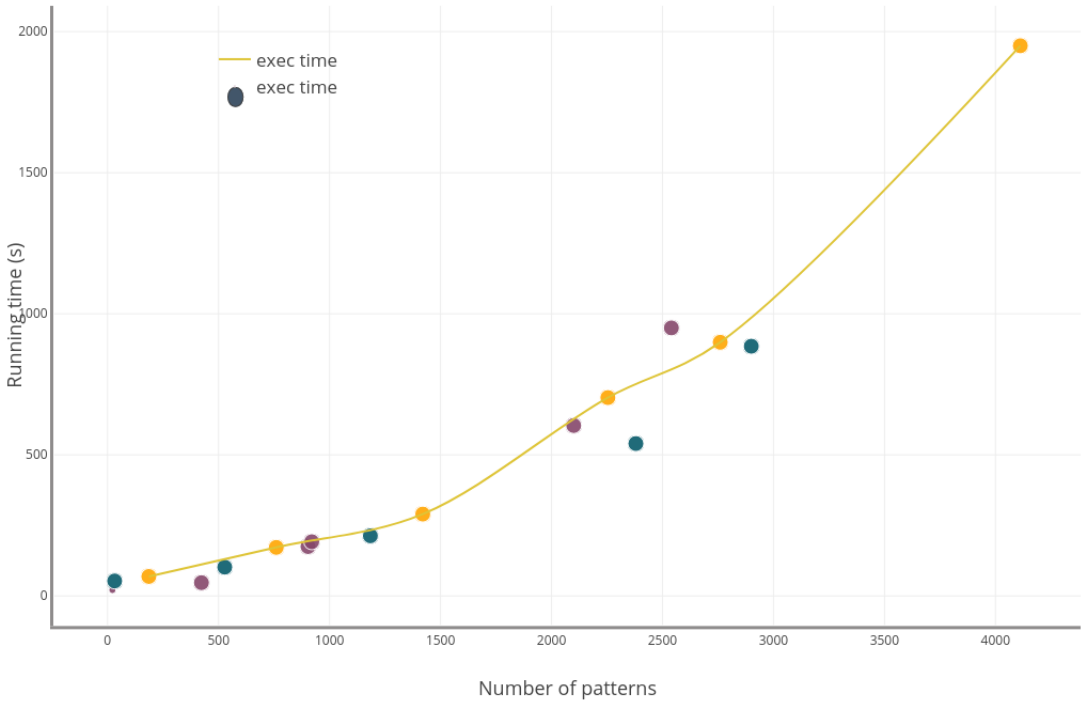


Fig. 4.3.: $FoldData$ performance

Notice that execution time is not impacted by the data completeness but grows exponentially with the number of generated patterns.

4.5.3 Pattern Query Processing

The following experiment measures the efficiency of processing pattern queries for producing minimal covers for query results over constrained tables. We compare the pattern-based query plans (blue solid path in Figure 3.2) using the techniques described in Section 3.4 from Chapter 3 by comparing it with the "naive" strategy of computing the minimal cover from the results of the query applied to the data and reference tables (red dashed path in Figure 3.2). We tested both approaches on the queries below and report the result in Table 4.11. The reported execution times correspond to the queyr answer completeness pattern table generation cost (Fold Answer), and to

the sum of pattern query evaluation cost and the Fold Pattern cost necessary to produce a minimal pattern set (Pattern Algebra).

Q_1:	$\sigma_{b=2223}(Temp)$	Q_2:	$\sigma_{b=2223 \wedge f=1}(Temp)$
Q_3:	$\sigma_{b\in(1213,3334)\wedge(m\in(11,12)}(Temp)$	Q_4:	$\pi_{b,f,r,m}\sigma_{bin(1213,2324)}(Temp)$
Q_5:	$\pi_{f,r,m,d}(Temp)$	Q_6:	$\pi_{b,f,r,area,temp}Temp \bowtie_b LocArea$
Q_7:	$OneBlg - OneMonth$		

Tab. 4.10.: Data Queries

Assessing the completeness of queries with the pattern algebra outperforms the naive approach for all of the tested queries. Queries Q_1 and Q_2 only refer to the spatial dimension and both methods (Fold Answer and Pattern Algebra) and can be optimized by exploiting attribute domain independence as described in Section 3.6. For Q_3 the gain is less important since it needs partial unfolding over both reference tables which incurs in an important overhead for Fold Answer. Queries Q_4 and Q_5 need no unfolding which explains the performance gain of the pattern algebra approach. For Q_5, the pattern algebra evaluation is much more efficient because of the compactness of the pattern covers and the fact that the corresponding pattern query doesn't need unfolding (no selection) in contrast with Q_4. The performance gain for the last two queries Q_6 and Q_7 is less significant since both imply accessing two tables, leading to performing joins between corresponding pattern tables. Pattern queries are independent of the data size. For Q_7 the data size is much larger than for Q_6, but the pattern queries have similar run time since both queries have pattern tables of similar size.

	Complete		*Missing*		Execution time (sec)	
Query	$\|Answer\|$	$\|P(Q_i)\|$	$\|Missing\|$	$\|\overline{P}(Q_i)\|$	Fold Answer	Pattern Algebra
Q_1	96,360	11	1,103,760	66	7.410	0.091
Q_2	8,760	1	191,808	15	0.250	0.002
Q_3	16,025	217	584,250	91	156.060	13.700
Q_4	144	12	3,228	46	1.700	0.140
Q_5	25,342	114	101,678	763	143.920	9.890
Q_6	327	11	10,415	578	10.090	8.630
Q_7	312,624	39	1,146,288	143	23.520	9.870

Tab. 4.11.: Complete and Missing Query Answer Patterns

4.5.4 Folding pattern query results

The last experiments set aims at digging deeper in the efficiency of pattern queries by analyzing the overhead of $FoldPatterns$. In general, $FoldPatterns$ operates on small pattern tables produced by queries. Nevertheless, it remains important to show that it doesn't influence the performances

of the query answer pattern computation. We consider different pattern table sizes with variable compactness values (see Table 4.9). We measure the $FoldPatterns$ phase execution time while keeping track of the exact number of merge and reduce operations (see Section 4.4).

We can observe in Table 4.12 that run time grows with the number of patterns to minimize. The table also shows the number of reduced and the number of merged patterns. As expected, the run time grows with the number of patterns to minimize and merging patterns is much more expensive than reducing patterns, due to the cost of querying the reference table.

P. size	P_{min}.size	Compac.	time	**m**erges /**r**educes
106	22	20.75%	0.29s	7 ***m***
238	32	13.44%	0.32s	9***m***+ 79 ***r***
570	30	5.2%	0.38s	45***m***
992	864	87%	0.47s	6 ***m*** + 32 ***r***
10961	3921	35.77%	1.33s	6 ***m*** + 7040 ***r***
11285	11178	99.01%	0.35s	107 ***r***
12054	11440	94.90%	6.59s	38 ***m*** + 158 ***r***

Tab. 4.12.: Pattern Fold algorithm performances

4.6 Summary

This chapter describes the implementation of contributions presented in Chapter 3. We show, by using examples, how pattern queries are translated into SQL. All pattern operators can be implemented in SQL, except the fold operator which requires recursive scan of data tables. We proposed a first Algorithm $FoldData$ that implements the fold operator over data tables. We showed that the pattern table size is bounded by the data table size. We also defined a second Algorithm $FoldPatterns$, which implements a variant of the fold operator that can directly be applied to pattern tables. This algorithm reduces pattern sets by grouping specializations and eliminating semantically redundant patterns.

We performed a set of experiments on real datasets, recording sensing output activity in the Smart Campus use case (refer to Chapter 1). The results show the effectiveness of using patterns for representing the complete and missing partitions for data tables constrained by spatial and temporal references. The compactness ratio for pattern table is very high for all tested data tables. We also tested the impact of multiple study features on the compactness of generated pattern tables and used synthetic datasets to this purpose. We observed that an important factor impacting the size of pattern tables is the missing data distribution. Random distribution in synthetic datasets provokes an explosion of the number of patterns ($|P| = |D|$ being the upper bound). In contrast, in real datasets, where missing data is caused by physical anomalies that do not occur in a random fashion, the compactness remains high.

Finally, we compared two approaches for computing query result patterns. The first approach applies $FoldData$ on the query result, whereas the second approach uses pattern queries. Our results show the efficiency of using the pattern queries compared to folding data results.

Part II

Incomplete Query Result Imputation

5 Data and query result imputation techniques

> *Research is to see what everybody else has seen, and to think what nobody else has thought*
>
> **— Albert szent-Györgyi**
> Nobel prize in physiology or medicine

Contents

5.1 Introduction

Regardless of the way data is collected, data incompleteness remains an important issue impacting the quality of data processes and the reliability of most data analysis tasks. Data incompleteness studies are manifold. We reviewed in a previous Chapter 2, the state-of-the-art contributions regarding data incompleteness representation. Data imputation consists of enriching incomplete data sets with new values gathered from multiple sources by applying appropriate estimation techniques.

Data imputation techniques are in general complex and costly. Applying such techniques equally to all data to large datasets makes this task sometimes inefficient compared to the expected output. For example, repairing a whole dataset for improving the quality of some queries over particular data fragments might not be efficient. Query-driven imputation overcomes this issue by restricting the cleaning task to data required in the query evaluation. The purpose behind is to limit the reparation cost by only considering the data necessary for evaluating the query, which is usually smaller than the entire dataset [Wan+14]. Result estimation approaches use database techniques such as sampling, query plan optimization, or relational algebra extension for missing data manipulation.

This chapter describes state of the art for the second part of the thesis, covering contributions around incompleteness improvement in databases. The chapter is organized as follows:

- Section 5.2 discusses general solutions for handling missing data and the various data cleaning parameters.
- Section 5.3 introduces data imputation and surveys different approaches addressing this subject and their techniques. We distinguish between approaches requiring a human to participate in the cleaning task, and those relying on automatic algorithms.
- Section 5.4 describes recent advances in query result estimation for incomplete databases. We discuss major contributions sharing our perspective for addressing the missing data problem.
- In Section 5.5, we propose a comparative study between the surveyed approaches.

5.2 Handling the Missing Data Problem

Identifying and representing missing data is a preliminary data cleaning task and additional knowledge about missing data is necessary for elaborating effective and efficient imputation strategies. In general, three criteria are considered for analyzing data imputation tasks:

Missing data distribution: In Chapter 2 we saw that ”data missingness” varies between missing totally at random, missing at random and the missing dependent on the value itself. One rarely can establish if a dataset includes enough explaining factors (attributes) to understand the missing data distribution [Gel+06]. Limiting the study to recorded attributes deteriorates the data correlation discovery process, and automatic learning is considered in many cases to overcome this limitation.

Data analysis goal: Data quality is a relative concept which depends on the data sets and the tasks to be achieved. Tasks requirements represent an essential factor for deciding which behavior to adopt regarding missingness. For example, in electricity consumption monitoring, it is more relevant to apply an enrichment strategy to complete room locations instead of completing sensors characteristics. The analysis task requires the availability of spatial and temporal attributes and does not exploit sensor properties metadata. Any effort put in a cleaning strategy must fit task specifications for cost optimization reasons.

Incompleteness reasons: Many reasons can lead to missing data [Wan+17]: physical anomalies in collecting devices (sensors, connected objects), human carelessness during manually filling data, access failures in data integration process, denormalized database schemas, etc. It is crucial to understand why a value is represented as missing and what methods were initially deployed for collecting data to avoid reusing redundant, ineffective tools.

There exists a significant number of research surveys for data cleaning methodologies[Gsc+12; Hel08; Lak+99]. Most of these works agree that handling missing data issues have been tackled in two ways, either by discarding incomplete data items (e.g., tuples with null values) or by repairing them. The following paragraph shortly explains the missing data discarding approach and its limitation, while the Section 5.3 covers data imputation techniques.

Discarding Missing Values A naive solution for resolving the missing data problem considers discarding the data tuples with missing values. Excluding missing data corresponds in statistics to two notable approaches [Lak+99]:

1. *complete case analysis* that ignores all data entries with missing values, and

2. *available case analysis* which consists of ignoring all variables (attributes) with a high rate of missing values.

In addition to the fact that the remaining data set might be considerably reduced, the question of data representativeness arises. For example, in a sensor network data table, discarding all measure tuples of a given floor number might produce biased results when aggregating measure values.

There exists well-documented literature for addressing missing data in the statistical area [Lit88; Lit+89; Lan+97; Lit+14; RUB76; Roy+04]. Sampling [Ach+99; Con+10] reduces the missing data ratio by choosing a representative sample of the population. The sampling involves errors and introduces a bias for queries evaluation, especially if the query is not considered as an input for sampling [Wan+14].

5.3 Data Imputation

Repairing missing data using data imputation technique consists in replacing missing values ($nulls$) with new values by applying an inference model and external reference (knowledge, heuristics, data). This reference is used

A frequently used approach for data imputation is the single imputation model that substitutes each missing value with one imputation output. The effectiveness of the repairing process relies exclusively on the inference quality. Other approaches [Rub04] strive at increasing the reliability and accuracy of the completion method by applying and aggregating the results of multiple imputation models. Distinct outputs are ordered following their estimation accuracy or can be aggregated into a mean estimation. Multiple imputations may guarantee better quality estimations but are more costly to achieve due to the number of simultaneous imputation models that operate.

Moreover, according to [Fel+76; Lit88], two families of imputation approaches emerge, models involving human intervention, presented in the following Section 5.3.1, and automatic inference models that will be surveyed in Section 5.3.2.

5.3.1 Human Based Imputation

In most situations, collecting tools are pointed out as the origin of data incompleteness.Human-based imputation starts from the assumption that specific data imputation scenarios cannot exclusively rely on automatic imputation tools and request human intervention to address the incompleteness problem [Fan+10b; Jef+08; All00].

Recent works argue that human intervention is needed beyond quality motivations. It represents an answer to a feasibility concern: automatic models perform well for missing values with enough correlation captured in data set but evolve less under the missing tuples setting [Li+16], where additional data tuples are needed to explain the missing data correlations. Human-based imputation is a family of techniques where the human is the reference for the inference process. We distinguish between two inference methods:

- Direct imputation or Crowdsourcing, where humans directly generate missing tuples and values manually.

- Indirect imputation, or Rule-Based Imputation, where domain experts encode their knowledge about existing data correlations in the form of logical rules that support the imputation inference.

In both cases, the reliability of the process depends entirely and exclusively on the human knowledge and understanding of data characteristics. We explain both approaches in the following paragraphs.

Crowdsourcing

With the emergence of web-based data services, crowdsourcing has become in the last decade a strong ally for data collection, cleaning, and quality assessment [Bra08; Chi+16]. Crowdsourcing platforms like Amazon Mechanical Turk [Amt; Buh+11], Figure Eight [Fig], Gengo [Gen] or Upwork [Upw] allow a set of individuals (workers) with different expertise levels and various specializations to perform problem-solving tasks related to producing or cleaning data sets (human intelligence tasks). They enable tasks assignation and their quality control.

Crowdsourcing implies paying the workers, and its supporters argue that investing in tasks is less expensive than any similar procedure involving direct employment and more valuable and efficient than automatic data processing software [Chi+16]. Using crowdsourcing for repairing missing data can be achieved in two forms [Li+16]:

- Fill tasks: where workers are provided with data sets and asked to fill missing values.

- Enumerating tasks: where users propose new data entries, disposing of a set of descriptions restricting the expected data.

We describe two examples of each form.

Filling Missing Values: Park and Widom [Par+14] propose a system for collecting structured data from the crowd. They provide workers with an incomplete data table and ask them to fill missing values. A candidate table is generated, where workers observe the current state of the table, *i.e.*, all previous updates of workers are available, and they can choose either to insert a new entry or to up-vote an existent record if they agree with. Attributes with $null$ values are subject to predefined constraints, included in the system and visible for workers. Once all workers update

the candidate table, the final table is created by keeping complete rows associated with the highest score obtained among all rows with a similar primary key.

Enumerating Additional Tuples: The interesting part of incompleteness that the previous form of crowdsourcing cleaning does not address is creating new records that the data table does not consider at all. In this Open World Assumption setting, workers must enumerate a set of records with predefined constraints or description. For example, the worker might be asked to list all point of interests in Europe. A similar system to upvoting can be used for collaboration and quality control. There remains a limitation: while the filling crowdsourcing can be assessed as complete when all missing values are informed, it is more challenging to consider an upper bound for enumerating tasks. Under the Open World Assumption, we do not have any prior knowledge about the full extent of the expected answer. One track for resolving this problem is proposed in [Tru+13]. The authors are inspired by the species estimation problem, known in biology, for estimating the unknown size of the population. They repeat a counting process involving all workers, to achieve this estimation. However, they point a limitation regarding human behavior that does not include one element in several enumerations. The work offers an improvement by designing a new estimator that only considers unique worker answers.

Rule based cleaning and imputation

Crowdsourcing allows humans to be directly involved in the data completion process by informing new values or records. While this solution generally produces results of high quality, it remains an expensive means. There exists an alternative way of taking advantage of human knowledge, without including costly and time-consuming manual effort. Rule-based cleaning relies on experts for injecting a set of constraints, such as logical rules, as an input for an automatic and declarative inference process. Indeed, defined logical rules can be reused even when data are updated, without requiring human intervention, as long as the produced records correspond to the original rules settings.

Many works are proposing to use a set of rules for data reparation, under different formalisms [Rah+00]. Functional dependencies are used in [Wij05] to define constraints allowing repairing and completing data tuples without deleting them, through update queries. Fan et al. [Fan+08] extends functional dependencies by defining Conditional Functional Dependencies (CFD) and explains how they allow a better data cleaning task. CFD's imply semantic values binding, giving more expressiveness to the repairing process than using traditional functional dependencies. Driven by the repairing cost evaluation and improvement, [Boh+05] deals with functional dependencies and inclusion dependencies for data cleaning. The authors focus on value modification for cost

assessment, which differs from other contributions where only tuple update costs are considered. The experimental study consolidates the effectiveness of a value modification guided data repair.

The common characteristics of rule-based cleaning are its dependency on a particular formalism for rules language. A notable work [Dal+13] points the lack of expressiveness engendered by this approach, due to its syntactic and semantic restrictions. The authors argue that adopting a single formalism for defining logical constraints ignores any rule that constitutes a violation of these constraints, naturally restricting repairing and completion possibilities. In order to handle the expressiveness limits, *Nadeef* is a framework that proposes mixing different formalisms for repairing rule expression. The rules toolbox goes further by allowing the user that defines these rules to customize the framework by proposing its proper rule encoding schema.

While previous works exploit user-defined rules for detecting and repairing incorrect or missing data values, the scope of new entailed values, in general, limited to the incomplete data attributes active domain. A recent contribution [Fan+10b], considers an additional input than the expert rule, to overcome this limitation. Master data are used to feed the repairing system with accurate values each time a tuple matches with a pattern stating its inconsistency. The formalism used for the system is editing rules, used to indicate what are replacing possibilities from master data to real incomplete data. The system is enriched with reasoning capabilities over editing rules, increasing the inference opportunities.

In addition to its high cost, the human intervention for repairing data is not always efficient. Take the example of time series data sets; it is uncertain that surveyed humans can help to fill the electrical consumption series. Even experts might fail, without prior automatic processing, in providing essential rules that capture existing data correlations. If we have several data sources, we can consider referring to data integration techniques for crossing multiple sources, which increases the possibilities for finding exact values.

In situations where human intervention does not fix the problem, automatic inference models based on statistics(or machine learning) are used. We review some significant approaches in this family in the next section.

5.3.2 Automatic Data Imputation

Statistical Imputation

Statistical imputation consists of attributing to each missing variable value a new value computed over the variable active domain values. Mean imputation is a widely used method [Lit+89] where missing values are replaced by the mean values computed over the variable active domain values.

Other formulas are also used as the most common value or the minimal/maximal observed value. Such quasi-random imputation approaches are efficient since they consist of setting and applying formulas for completion regardless of particular data properties. The major disadvantage of this approach is the distribution bias introduced: if a variable misses many values in many records, all of them are replaced with the same imputation (mean, max).

To overcome the randomness of the statistical imputation, other contributions propose to take into account the variable correlations. In this sense, [Buc60] introduced conditional mean imputation. The *Hot deck* technique [Gow71] applies a two-stage imputation process. First, a clustering step based on variable covariance is performed, before applying the imputation formula, with respect to the formed clusters. This generates a more precise imputation result. For example, instead of taking the most recurrent value in the dataset for imputation, it replaces missing values by the most frequent value occurring in their respective clusters.

Learning Model Imputation

The majority of these techniques rely exclusively on data samples. There exist alternatives where estimation values are extracted from external samples, appreciated for a level of completeness and accuracy higher than the classical task. Extensive work has been dedicated to applying machine learning techniques to deal with missing data imputation [Lit+14; VB18; Don+06; Jer+10; Sch97]. Statistical inference achieves missing data imputation by modeling the estimation impact of values and *discovering* imputation functions through a learning process. The imputation goal is then to build a predictive estimator that relies on the observed values of the subject (characteristics) to predict the missing variable values. Various estimators have been used for imputation. We present some notable approaches listed in [Jer+10]:

- Regression based imputation requires the use of reference data in addition to the incomplete dataset in order to build the regression model. It aims at estimating missing values as a prediction function output. Regression models are multiple, for example, [Rag+01] uses linear regression based on the Expectation Maximization principle [Dem+77].

- Multilayer Perceptrons (MLP) [Car+83] are artificial neural networks that train data sets to discover neural weights and can serve for several tasks such as classification or dimensionality reduction. [SR+11; Jun+04] are a few examples of works that have applied MLP for filling incomplete data. Their experimental results illustrate the efficiency of this approach, especially for categorical variables imputation, where other statistical techniques (mean/mode) fail to achieve good estimations.

- Nearest neighbors [Dix79] apply similarity measures to perform imputation where the most *similar* n objects are retained for estimating missing values. Similarity values are weighted with the number of missing values to take account of the estimation quality.

Other techniques for multiple data imputation use iterative decision trees [Ssa+08], sequential regression trees [Bur+10], self organization maps [Koh97], or predictive time series [Dua+07].

BayesDB

Machine learning and statistical models have the advantage of automatic imputation procedures but require experts to train the framework before use. From a database perspective, BayesDB [May+10; Yak+11] is the first approach which connects statistical learning models software and databases by offering non-experts a user inference for running inference models. Inference models can be monitored using an SQL-like language and do not require users to master the applied learning models. Additional features allow experts to input domain knowledge information to customize the models and achieve better estimations. Using machine learning and automatic inference models for missing data imputation relegates the significant works to statistical models but does not exclude user participation in the process. The case of BayesDB is not the only case where users can, a work where the user can set up confidence values for controlling repairing algorithm outputs.

5.3.3 Summary

We reviewed two main approaches dealing with the task of data imputation. The first approach requires humans to understand data correlations and either repair missing data manually (Crowd-sourcing) or by defining logical imputation rules. The second family of approaches considers automatically discovering data correlations in order to infer new values for missing data. We summarize both families techniques in a taxnomy Figure 5.1.

5.4 Query-driven Imputation

Data imputation strategies as those described in the previous section often are costly and fail to remain efficient for large datasets. The primary goal of cleaning data is to increase the quality of some predefined data processing tasks. Query-driven imputation consists of taking account of the query to improve improving query results over incomplete datasets. The goal is to achieve a targeted and more efficient imputation process, especially over large datasets. The primary motivation is to provide estimations for complex queries in a fast way and to avoid to repair the entire dataset. This is particularly relevant for aggregate queries. Instead of applying imputation to the whole dataset,

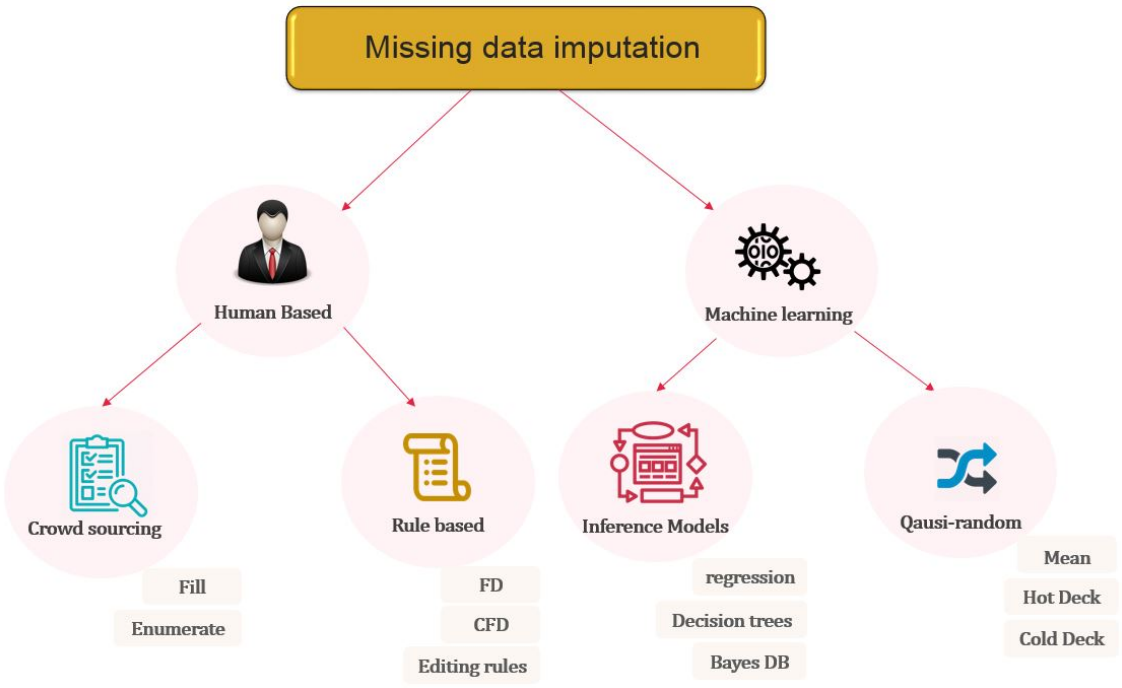


Fig. 5.1.: A taxonomy of data imputation techniques

query-driven imputation approaches aim at selecting a subset of data that deserve imputation for increasing a targeted quality metric. In this section, we discuss some works directly related to our work described in Chapter 6 to achieve data imputation at the query answer level.

5.4.1 Approximate Query Processing

Approximate query processing [Bab+03; Gar+01; Cha+01] consists of evaluating queries over data samples instead of the entire dataset to minimize execution time. Statistics can be used for query optimization, and for controlling the data flow. A frequent use case consists of computing summary statistics for online data, restricting data using time windows [Hel+00; Ram+02]. Statistical sampling techniques guarantee representative datasets with limited bias error. However, these approaches do not consider errors at the data level. In particular, missing or incorrect data entries are not identified to evaluate their impact on approximate query results.

More recently, SampleClean [Wan+14] integrates sampling with data cleaning to propose an approach for query result estimation that addresses obvious errors in the data model. Data cleaning can be achieved with different techniques, as listed in the previous section. Sampleclean proposes to reduce the imputation costs by concentrating the cleaning process on a representative data sample. This framework is implemented in two ways. *Clean estimation* replaces the query result to be computed over the entire data set, with the result obtained from the cleaned sample, and *Error correction* materializes the raw query result but associates a bias observed on the approximate query

result. In both cases, the approach error is captured by measuring how far the sample represents the full dataset. The framework is tested by including eventual cleaning errors, where the cleaning technique does not achieve 100% error elimination.

5.4.2 Dynamic Imputation

The recent work of [Cam+17] tackles the imputation problem from another angle. It incorporates missing data imputation operations into a query optimization engine. In addition to the core relational algebra operators, two operators are defined to allow the replacement *on the fly* of missing or incorrect values, targeting only data involved in the query evaluation. The *Drop* operator deletes any tuple containing null values, while *Impute* uses statistical inference mechanisms for replacement values computation. Since the framework is built as a query plane optimizer, the classical optimization function taking into account evaluation cost is extended to cover quality requirements. By integrating these parameters, the user can choose her own trade-off between quality and cost for query evaluation.

5.4.3 Missing Tuples Impact on Query Results

Most works regarding data completeness improvement generally focus on missing values since their representation is included in usual data models, which is not the case of missing tuples. Under the Open World Assumption, i.e., without master data, no clue is available about the number of missing records from an available set of data, nor about their nature. Nevertheless, their impact on the quality of query results is not marginal. In [Chu+18], a new perspective of missing data is proposed, introducing the study of the impact of unknown unknowns on query results.

5.5 Summary

Data incompleteness has a high negative influence on data analysis tasks. The missing data problem has first been identified and treated by statisticians. Studies on missing data distributions allowed to perform explanatory tasks and simple formulations have been used to repair incompleteness such as mean or most frequent value imputation. Data imputation recognizes two leading families of approaches, those involving the human to participate in the data repair process, and others, where inference algorithms are designed to discover data correlations and generate reliable estimations of missing values.

- Human-based imputation takes two forms. Crowdsourcing platforms ask workers to manually repair data by filling missing observations. In the other hand, rule-based techniques rely on experts to translate their domain knowledge by defining logical rules, which can be used for automatic repairing, and while running databases updates.
- Automatic imputation regroups all machine learning and basic statistical techniques, where algorithms attempt to explain missing data distributions and its correlation with available observations. Learning outputs are used to feed the inference mechanism, for creating new values.

Most data imputation techniques operate at data level without taking into account the query semantics. This implies a considerable effort and cost for repairing data tables, regardless of their future use rate. Query-driven imputation is concerned with estimating the query results that might be impacted by poor data quality (as missing data). One notable example is dynamic query imputation [Cam+17], where data imputation is injected into the query evaluation plan as optimization operators.

We summarize in Table 5.1, research contributions surveyed in this chapter.

	Technique	Reference	Inference Model	Missing data
Data Imputation	Crowdsourcing	Human	Manual	Values, Tuples
	Rule-Based	Human	Automatic	Values
	Relative Rule Based	Human + Master data	Automatic	Values
	Statistics	Data + Estimation Function	Automatic	Values
	Machine learning	Data + Estimation Model	Automatic	Values, Tuples
Query-driven Imputation	Sample Clean	Cleaned sample	Automatic	Values
	Dynamic Imputation	Data	Automatic	Values
	Rule based Query result	Human + Data	Automatic	Values, Tuples

Tab. 5.1.: Comparative table for missing data representation models

6 Query Result Imputation for Aggregation Queries

> *There are 360 degrees, so why stick to one?*
>
> **— Zaha Hadid**
> Pritzker architecture prize

Contents

6.1 Introduction

Data incompleteness naturally leads to query results of poor quality, and repairing missing data is a common data cleaning task. Data imputation designates a family of approaches that aims at repairing missing data by inferring new values from the available dataset, sometimes with human intervention. In the previous chapter, we surveyed multiple methods for data imputation and showed that imputation techniques are generally complex and do not scale up to large datasets. In particular, global data imputation might become inefficient for repairing task-specific input datasets.

Query result estimation techniques address the missing data problem at the query result level. They show that focusing the repairing task on data used for the query evaluation can drastically reduce cleaning efforts. We consider in this chapter the problem of data imputation for repairing aggregate query results, obtained from incomplete data. Missing data leads in general to missing query results, but in the case of aggregate queries, they also create incorrect aggregations.

In the first part of this thesis, we propose a pattern model for describing available and missing data extents and an algebra allowing to derive annotations for query results. Our motivation for taking the direction of rule-based imputation is the opportunity of applying this pattern model to identify missing partitions at different aggregation levels and to integrate this functionality in a general inference process for generating query-driven imputation strategy.

The chapter is structured as follows:

- Section 6.2 enumerates the challenges and motivates our contributions through an example.
- Section 6.3 extends the pattern data model by introducing imputation rules which can be applied to repair a large class of aggregate queries, the syntax. We define the precise semantics of imputation rules and their translation into imputation queries.
- Section 6.4 describes the imputation process that consists of four steps: result annotation, candidate imputation query generation, imputation strategy selection, and imputation queries generation.
- Section 6.5 explains how the imputation strategy is implemented using the pattern algebra defined in Chapter 3.
- Section 6.6 summarizes a set of experiments run over a benchmark dataset. The experiments evaluate the imputation strategy, and results are provided for each process step.
- Finally, the chapter contributions are summarized in Section 6.7.

6.2 Motivation

Recall the Smart Campus scenario introduced in Chapter 1. Data table $Energy$ stores daily electrical consumption for buildings at the room level. It might miss some measures for floors which are indicated with a $null$ value for the KwH attribute in Table 6.1. As shown in Section 3.2, we can analyze data table $Energy$ with respect to a reference table R that represents all tuples on reference attributes $A = \{Building, Floor, Room, Week, Day\}$. Suppose that D represents the subset of tuples in $Energy$ where the KwH attribute is not $null$. Using the pattern model defined in Chapter 3, we can run the folding algorithm presented in Chapter 4 to compute complete and missing minimal covers $\mathcal{P}^*(D, R)$, and $\bar{\mathcal{P}}^*(D, R)$ for the constrained table (D,R).

Example 6.1. *Table 6.1 shows respectively a representation of table $Energy$ (ordered by week and floor), completeness patterns $\mathcal{P}^*(D, R)$, and missing partition patterns $\bar{\mathcal{P}}^*(D, R)$. Each floor contains one room, and the week only counts three days. For example, for week 1, $Energy$ contains all measures of floor 1, misses one measure for floor 2.*

Tab. 6.1.: $Energy$ table and its pattern minimal covers

Data table $Energy$

B	F	R	W	D	kWh
25	1	1	1	1	12.3
25	1	1	1	2	10.1
25	1	1	1	3	9.6
25	2	1	1	1	8.3
25	2	1	1	2	6.4
25	2	1	1	3	null
25	3	1	1	1	5.3
25	3	1	1	2	7.2
25	3	1	1	3	6.1
25	1	1	2	1	null
25	1	1	2	2	null
25	1	1	2	3	null
...	...	...	...	...	...

$\mathcal{P}^*(D, R)$

B	F	R	W	D
25	1	*	1	*
25	2	*	1	1
25	2	*	1	2
25	3	*	1	*
25	3	1	2	*
25	5	*	1	*
26	*	*	*	*

$\bar{\mathcal{P}}^*(D, R)$

B	F	R	W	D
25	1	*	2	*
25	2	*	1	3
25	*	*	3	*
25	3	1	2	*
25	4	*	*	*

Consider the following SQL query Q_{kwh} that computes the total weekly energy consumption for three floors in building 25:

```
SELECT Building B, Floor F, Week W, Sum(kWh) as kWh
FROM Energy
WHERE B = 25 and F in (1,2,3)
GROUP BY B, F, W
```

Listing 6.1: Aggregate KwH per floor

The query answer $Q_{kwh}(Energy)$ is illustrated in Table 6.2. Each tuple in the query answer is obtained by aggregating a partition of the input data and annotated as correct if the partition is complete, missing if the partition is empty and incorrect if the partition is partially complete.

Tab. 6.2.: The query $Q_{KwH}answer$

$Q_{kWh}(Energy)$

B	F	W	kWh	label
25	1	1	32.0	correct
25	2	1	14.7	incorrect
25	3	1	18.6	correct
25	1	2	null	missing
...	...	...	...	...

Query-driven result imputation tries to avoid spending effort in, which is not relevant to a query. This is especially useful for aggregate queries, where results can be estimated by exploiting available correct aggregated results. Imputation rules could be expressed at the result level, exploiting the knowledge about the domain at the aggregation level. For example, an expert could define rules that estimate the mean temperature at the floor level, without having to estimate each room value, but by using similar floors values.

We propose a rule-based approach for data imputation, similar to the one defined in [Dal+13]. The main new contribution is the use of patterns for analyzing aggregate queries and choosing imputation rules to estimate aggregated values for missing or partially complete partitions.

For example, an expert can state by the following imputation rule stating that some missing or incorrect query result for a given floor can be repaired by the average value of all available and correct results computed for other floors.

$$r_0 : (B : x, F : \_, W : y) \leftarrow (B : x, F : \_, W : y), avg(kWh)$$

This rule has the following imputation semantics: any missing or incorrect measure for a given floor in building x and for week y which matches the left-hand side of the rule r_0 can be replaced by the average of all correct results for the same building x during the same week y (partitions matching the right-hand side of the rule).

The rest of this chapter aims at defining the imputation model for aggregate queries results, for repairing missing and incorrect results.

6.3 Imputation Model

In this section, we will introduce the basic definitions for building the imputation process. For understanding these concepts, we suggest the reader refer to Chapter 3 for all definitions related to constrained tables, partition patterns, minimal covers, and pattern algebra.

6.3.1 Aggregate Queries and Query Patterns

Our imputation model is defined for a particular sub-class of SQL aggregate queries:

Definition 6.1 (valid aggregate query)**.** *Let Q be a valid SQL aggregate query of the form*

```
SELECT S, agg(m) FROM T WHERE P GROUP BY G
```

such that the `WHERE` *condition P uses only equality predicates with constants.*

Without loss of generality, we assume that P is in disjunctive normal form.

Example 6.2 (aggregate query example)**.** *Query Q_{kWh} can be rewritten by transforming the* `WHERE` *clause into* `(B = 25 AND F=1) OR (B = 25 AND F=2) OR (B = 25 AND F=3)`*.*

Recall from the Chapter 3 that a pattern is a tuple whose attributes can take the wildcard value $*$ for summarizing complete and missing partitions. For defining imputation rules, we generalize this notion of pattern to that of query pattern, with the possibility to assign variables to attributes.

Definition 6.2 (query pattern)**.** *A* query pattern *is a tuple $q = (a_1 : x_1, ..., a_n : x_n)$ where for each attribute a_i, its values $x_i \in dom(a_i) \cup V \cup \{*\}$ is (1) a constant in the domain of attribute a_i or (2) a distinct variable in a set of variables V or (3) the wildcard symbol $*$. We denote by C_p, V_p and W_p the set of constant, variable and wildcard attributes in p.*

Example 6.3. *Expression $(B : 25, F : x, R : *)$ is a query pattern where $x \in V$ is a variable.*

We generalize the notion of query patterns to query pattern sets.

Definition 6.3 (query pattern set)**.** *Let Q be some valid SQL aggregate query of some table T as defined in Definition 6.1 and A be a key of the input table T containing all attributes in Q except the aggregated attribute. We can then define a set of query patterns $\mathcal{Q}$ over A which contains a query pattern $q_i \in \mathcal{Q}$ for each conjunction d_i in the* `WHERE` *clause such that (1) all attributes A_j in d_i are represented by the corresponding constants c_j in q_i, (2) all other attributes in the* `GROUP BY` *clause are distinct variable attributes and (5) all attributes in A and not in Q are wildcard attributes.*

Example 6.4 (query as a query pattern set)**.** *For example, suppose that* $\{B, F, R, W, D\}$ *is a key of table* $Energy$ *(see Section 6.2). Then, the SQL query* Q_{kWh} *generates the query pattern set*

$$\begin{aligned}\mathcal{Q} = \{&(B : 25, F : 1, R : *, W : x_w, D : *),\\ &(B : 25, F : 2, R : *, W : x_w, D : *),\\ &(B : 25, F : 3, R : *, W : w_w, D : *)\}\end{aligned}$$

Observe that all query patterns of a query share the same wildcard attributes (with value $*$), and if q does not contain any wildcard $*$ attribute, then the corresponding SQL query corresponds to a simple conjunctive query which returns the measured values of the matching tuples (without aggregate and group-by clause).

The instance of a query pattern q defines a subset of the partitions generated by the `GROUP BY` clause over the tuples filtered by the `WHERE` clause of the corresponding SQL query. This filtered partitioning can formally be defined by an equivalence relation over the query input tuples:

Definition 6.4 (pattern tuple equivalence)**.** *A tuple* t matches *a query pattern* q*, denoted* $match(t, q)$*, if* $t.a_i = q.a_i$ *for all* constant *attributes in* q*. Two tuples* t *and* t' *matching some query pattern* q *are* equivalent *in* q*, denoted* $t \equiv_q t'$*, if* $t.a_j = t'.a_j$ *for all* variable *attributes* a_j *in* q *(*t *and* t' *only can differ for wildcard attributes).*

A pattern p defines for each matching tuple t an equivalence class $\Phi_q(t) = \{t' | t \equiv_p t'\}$.

Definition 6.5 (Query pattern instance)**.** *The instance of a query pattern* q *in some table* D*, denoted* $I(D, q)$*, contains all equivalence classes (partitions) of tuples in* D*.*

Example 6.5. *For example, the equivalence class* $\Phi_q(t)$ *of tuple* $t = (B : 25, F : 1, R : 1, W : 2, D : 3)$ *defined by pattern* $q = (B : 25, F : 1, R : *, W : x_w, D : *)$ *contains all tuples of building 25, floor 1 and week 2. The equivalence class* $\Phi_q(t')$ *with the same pattern* q *for tuple* $t' = (B : 26, F : 1, R : 1, W : 2, D : 3)$ *is empty. Finally,* $q' = (B : 25, F : 1, R : *, W : 2, D : *)$ *defines a unique equivalence class of all tuples of floor* 1 *in building* 25 *and week* 2*.*

It is easy to see that (1) when p does not contain any wildcard attribute, then $I(D, q) = \{\{t\} | t \in I(q, D)\}$ contains a singleton for each matching tuple in D, and (2) when q does not contain any variable, $I(q, D) = \{\Phi_q\}$ contains a single partition $\Phi_q \subseteq D$.

6.3.2 Imputation Rules and Imputation Queries

Imputation rules repair the results of aggregate queries by estimating the aggregated values of missing partitions and repairing the incorrect aggregations. They are defined using *query patterns* characterizing the results that should be repaired and those that could be used for their reparation.

Definition 6.6 (imputation rule)**.** *An* imputation rule *for some set of reference attributes A and some measure attribute m is an expression of the form $r : q_m \leftarrow q_a, imp$ with:*

(1) q_m and q_a are query patterns over A without wildcards*,*

(2) all variables shared by q_m and q_a are bound to the same attribute in q_m and q_a and

(3) imputation expression *imp is an aggregation function transforming a set of values in the domain of m into a single value in the domain of m.*

In the following, we use the anonymous variable _ for denoting non-shared variables.

Example 6.6. *For example, we can define the following imputation rules for missing kWh values:*

$$r_1 : (B:x, F:\_, W:y) \leftarrow (B:x, F:\_, W:y), (max(kWh) + min(kWh))/2$$
$$r_2 : (B:x, F:y, W:3) \leftarrow (B:x, F:y, W:2), kWh$$
$$r_3 : (B:\_, F:4, W:x) \leftarrow (B:\_, F:4, W:x), avg(kWh)$$

Imputation rule r_1 produces an estimation of the total electricity consumption by week of some floor in some building using the midrange of all correct *consumption values over* other *floors of the same building and the same week. Rule r_2 takes the correct consumption of a floor in week 2 for estimating the value of the same floor at week 3 (the aggregation function is the identity). Finally, rule r_3 takes the average of all correct values for floor 4 in all buildings to repair the value of the same floor in some building for the same week.*

Implementing the semantics of an imputation rule is defined with respect to a query Q, a table M of result tuples to be repaired by the rule and a table $AvailableD$ of all correct values which can be used for imputation. Table M contains all results generated by incomplete partitions and all missing results corresponding to empty partitions whereas table $AvailableD$ contains all possible correct tuples. A formal definition for computing these tables w.r.t. a query pattern will be explained in the imputation process (Section 6.4).

Definition 6.7 (imputation rule semantics)**.** *Let M and $AvailableD$ be two tables that contain the tuples to be repaired and the tuples which can be used for reparation. Then, the semantics of an imputation rule $r : q_m \leftarrow q_a, imp$ is defined by the following* imputation query $Q(r)$ *over M and $AvailableD$ where A contains all attributes in q_m and S is the set of variable attributes shared by q_m and q_a:*

```
SELECT x.A, imp AS m
FROM M x, AvailableD y
WHERE match(x, q_m) AND match(y, q_a) AND x.S = y.S
GROUP BY x.A
```

The previous imputation query joins all tuples $x \in M$ matching q_m with the set of tuples y in $AvailableD$ matching q_a over the shared attributes S, partitions the obtained table over all rule attributes and finally applies the imputation expression imp to estimate a value for m.

Example 6.7. *The imputation rule r_2 from example 6.6 can be rewritten into the following SQL query :*

```
SELECT x.B, x.F, x.W, y.kWh
FROM M x, AvailableD y
WHERE x.W = 3 AND y.W=2 AND x.B = y.B AND x.F = y.F ;
```

where $AvailableD$ corresponds to the correct results generated by query Q_{kWh} without its filtering condition.

6.4 Query Imputation Process

Given a data table D, and its related minimal pattern covers $\mathcal{P}^*(D)$ and $\bar{\mathcal{P}}^*(D)$, an imputation process is concerned with repairing the result of any aggregate query Q, using a set of imputation rules R. Our imputation strategy makes the following assumption. Any partition of the query result that is identified as incorrect or missing has lower quality than the result of the imputation. Subsequently, incorrect partitions results are replaced by the imputation result. The imputation process is illustrated in Figure 6.1.

The imputation strategy is decomposed into four steps:

- Step 1 consists in identifying the set of all partition patterns $ImputeP(Q)$ summarizing the partitions to be repaired and the set of partition patterns $AvailableP(Q)$ of partitions that can be used for reparation.

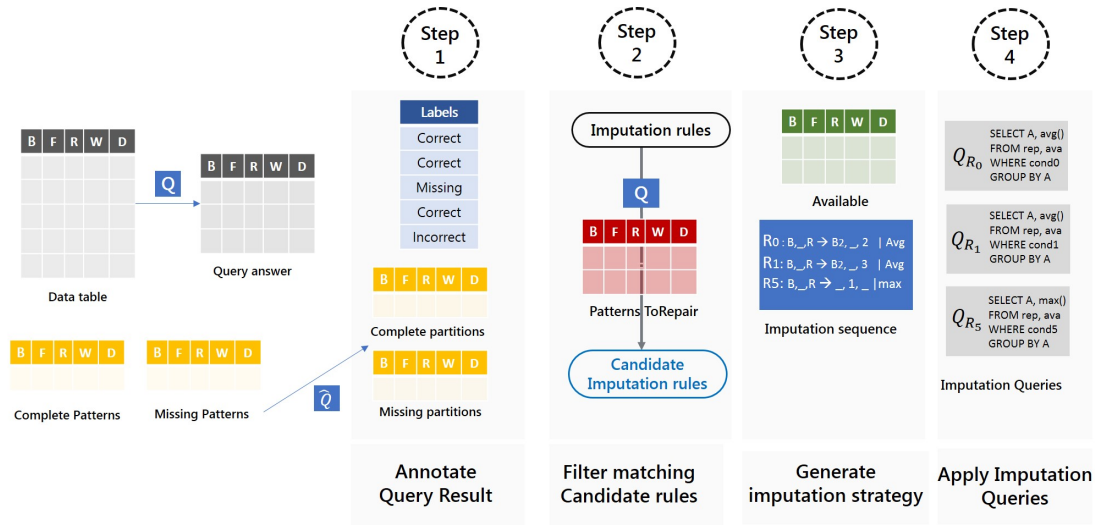


Fig. 6.1.: The imputation process illustration

- Step 2 consists of identifying the set of all rules that can be used for repairing $ImputeP(Q)$ by using $AvailableP(Q)$. A rule is chosen if and only if it can repair at least one answer tuple and if there exists at least one correct value that can be used for imputation. The result of this step is a set of *candidate imputations*.

- Step 3 consists of creating a *sequence of candidate imputations* which repair the missing and incorrect tuples. Observe that several queries might repair a tuple, and we assume that each imputation query overwrites conflicting repaired tuples generated by the previous queries.

- Step 4 generates the imputation queries following the imputation strategy.

In the rest of this section, we describe step-by-step the query-driven imputation process to generate efficient imputation strategies using partition patterns and imputation rules.

6.4.1 Step 1: Annotating Query Results

The first step requires identifying partition patterns corresponding to correct, incorrect, and missing query results. The following definition introduces the matching property between a partition pattern and a query pattern.

Definition 6.8 (Partial/full match)**.** *A query pattern* q matches *a partition pattern* p*, denoted by* $match(q, p)$*, if for all* constant *attributes* $q.a_i$ *in* q*,* $q.a_i = p.a_i$ *or* $p.a_i = *$*. If* $match(q, p)$*, we can define a mapping* ν *from the variable attributes* a_i *in* q *to the attributes in* p *such that* $\nu(q.a_i) = p.a_i$*. Then, a query pattern* q*:*

1. fully matches *partition pattern* p*, denoted by* $full(q, p)$*, if* $\nu(q)$ *matches* p *and*

2. partially matches *pattern* p*, denoted by* $partial(q, p)$*, otherwise. Partition pattern* $\nu(p)$ *is called the* matching pattern *of* q *for* p*.*

Definition 6.9 (extended query pattern)**.** *Let* $W(Q)$ *be the set of wildcard attributes in the query patterns* $\mathcal{Q}$ *of some query* Q *and* q *be a query pattern over all variable and constant attributes in the query patterns of* Q*. Then we denote by* q^* *the query pattern where all attributes in* W *are wildcard attributes. Pattern* q^* *is called the extension of* q *in* Q*.*

Example 6.8. *The extension of query pattern* $q = (B:25, F:2, W:\_)$ *is* $q^* = (B:25, F:2, R:*, W:\_, D:*)$ *and the extension of tuple* $t = (B:25, F:1, W:1)$ *in* Q *is pattern* $t^* = (B:25, F:1, R:*, W:1, D:*)$*.*

Proposition 6.4.1 (patterns classification)**.** *Given a query* Q *over some constrained table* $T = (D, R)$ *with complete pattern summary* $\mathcal{P}^*(T)$ *and missing pattern summary* $\bar{\mathcal{P}}^*(T)$*. Let* $\mathcal{Q}$ *be the query pattern set of* Q*. Then, for any tuple* t *in the reference table of* Q *the following conditions hold:*

- t *is in the result of* Q *and correct iff* t^* *matches a pattern* $p \in full(\mathcal{P}^*(T), \mathcal{Q})$*;*
- t *is in the result of* Q *and incorrect iff* t^* *matches a pattern* $p \in partial(\bar{\mathcal{P}}^*(T), \mathcal{Q})$ *(or equivalently* $p \in partial(\mathcal{P}^*(T), \mathcal{Q})$*);*
- t *is missing in the result of* Q *iff a pattern* $p \in full(\bar{\mathcal{P}}^*(T), \mathcal{Q})$ *matches* t^**.*

In addition to partition patterns identification, imputation rules need to identify larger partitions sets in data, not necessary appearing in the query answer. Indeed, the missing partition patterns need to be repaired by a set of correct available partitions, from the data table. The following definition introduces these pattern sets.

Definition 6.10 (missing/available pattern sets)**.** *Given query* Q *over some table* $T = (D, R)$ *with pattern tables* $\mathcal{P}^*(T)$ *and* $\bar{\mathcal{P}}^*(T)$ *and query pattern set* $\mathcal{Q}$*. We can then define the following sets of patterns for* Q*:*

- $ImputeP(Q) = full(\bar{\mathcal{P}}^*(T), \mathcal{Q}) \cup partial(\bar{\mathcal{P}}^*(T), \mathcal{Q}) = full(\bar{\mathcal{P}}^*(T), \mathcal{Q})$
- $AvailableP(Q) = \{p | p \in \mathcal{P}^*(T) \wedge \forall A \in W(Q) : p.A = *\}$

$ImputeP(Q)$ contains all patterns describing incomplete or missing partitions (to be repaired) in the result of Q whereas $AvailableP(Q)$ describes all complete partitions that can be used for repairing Q.

6.4.2 Step 2: Generate Candidate Imputations

Missing and incorrect answers of some aggregate query Q (query pattern set $\mathcal{Q}$) are estimated by imputation queries. Each imputation query is generated by an imputation rule and repairs some missing and incorrect tuples. We assume that the complete and missing data partitions are represented by a complete and missing pattern summary as defined before.

We first define the notion of candidate imputation.

Definition 6.11 (candidate imputation)**.** *Let $ImputeP(Q)$ be the imputation pattern set and $AvailableP(Q)$ the reparation pattern set of Q. A* rewriting *ω for $p_m \in ImputeP(Q)$ is an expression $\omega : p_m \leftarrow^r P_a$ where there exists an imputation rule $r : q_m \leftarrow q_a, f_{imp}$ such that the extended query pattern q_m^* matches p_m with ν and $P_a \subseteq \mathcal{P}^*(T)$ is a non-empty set of complete patterns in $\mathcal{P}^*(T)$ that are matched by $\nu(q_a^*)$.*

We say that rule r generates rewriting ω and call $\nu(q_m^*)$ the *imputation pattern of* ω and $\nu(q_a^*)$ the *repair pattern* of ω. All rules r where there exists at least one rewriting are called *candidate imputations* for Q.

Example 6.9. *For example, $\omega_1 : e_1 \leftarrow^{r_1} \{c_3, c_7\}$ is a candidate imputation for $e_1 : (25, 1, *, 2, *)$ generated by rule r_1 with imputation pattern $\nu(q_m^*) = e_1 : (25, 1, *, 2, *)$, repair pattern $\nu(q_a^*) = (25, \_, *, 2, *)$ and $P_a = \{c_3 : (25, 2, *, 2, *), c_7 : (25, 5, *, 2, *)\}$, Similarly, $\omega_2 : e_2 \leftarrow^{r_2} \{c_3, c_7\}$ is a candidate imputation for $e_2 : (25, *, *, 3, *)$ using rule r_2 with imputation pattern $\nu(q_m^*) = e_2 : (25, *, *, 3, *)$, repair pattern $\nu(q_m^*) = (25, *, *, 2, *)$ and $P_a = \{c_3 : (25, 2, *, 2, *), c_7 : (25, 5, *, 2, *)\}$ and Finally, $\omega_3 : e_2 \leftarrow^{r_3} \{c_8\}$ is second a candidate imputation for $e_2 : (25, *, *, 3, *)$ using rule r_3 with imputation pattern $\nu(q_m^*) = e_2 : (25, *, *, 3, *)$, repair pattern $\nu(q_a^*) = (\_, 4, *, 2, *)$ and $P_a = \{c_8 : (26, *, *, *, *)\}$. .*

6.4.3 Step 3: Imputation Strategy

The result of step 2 is a set of candidate imputation rules where there exists at least one rewriting. Given a set of candidate imputations $\mathcal{R}$ for some aggregate query Q, the goal is to define an ordered sequence of candidate imputations for repairing the answer of Q. This sequence is called an *imputation strategy*. The goal of a strategy is to solve two kinds of conflicts. First, there might exist several candidate imputations for the same partition pattern $p_m \in ImputeP(Q)$ as shown in the example above for pattern e_2. Second, patterns in $ImputeP(Q)$ might not be disjoint and repair a subset of shared tuples. For example, missing patterns $e_2 : (25, *, *, 3, *)$ and $e_4 : (25, 4, *, *, *)$ might share the partition $(25, 4, 3)$. A standard way for solving such conflicts is to apply a multiple-imputation strategy which consists in applying all candidate imputations and combining

the estimated results through some statistical methods. In this article, we adopt a different strategy which consists in regrouping all candidate imputations of for each rule and in evaluating these imputation groups following some order defined over the imputation rules. We can show that this process is deterministic since each imputation rule repairs any tuple at most once (generates at most one imputation value).

Imputation rules can be ordered in different ways. For example, one might prefer "specialized" rules to more "generic" rules where specialization can be expressed by the number of constants in the and shared variables. For example, rule r_3 is then considered more specialized than rule r_1 since it contains more constants whereas rule r_2 is more specialized than r_3 since it contains more shared variables (with the same number of constants). Another strategy is to order the rules using some statistical estimations about data distribution, bias, and completeness or domain-specific expert knowledge about the system generating the data. For example, if the kWh values for floor 4 are quite similar overall buildings for a given week, rule r_d might be preferable to rule r_c. Rule r_1 might be preferred to the other rules if the kWh values do not vary over the floors of the same building.

6.4.4 Step 4: Imputation Query Generation

As shown in Definition 6.7, each candidate imputation $r : q_m \leftarrow q_a, f_{imp}$ generates an imputation query joining the table M of values to be repaired with the table $AvailableD$ containing all correct values.

As explained in Section 6.3, table $AvailableD$ is shared by all imputation queries and can be obtained by removing the filter condition (where clause) of query Q and matching the result with the pattern table $AvailableP(Q)$ (see Definition 6.10). For performance reasons, we precompute this table once and store the result, and reuse it for all imputation queries. Table M can be obtained by matching the result Q with pattern table $ImputeP(Q)$. Each rule $r : q_m \leftarrow q_a, f_{imp}$ then generates the following imputation query over tables $ImputeP(Q)$, the result table $Result$ of Q and $AvailableD(Q)$ where S is the set of variable attributes shared by q_m and q_a and A is the set of remaining attributes in q_m:

```
SELECT x.A, x.S, f_imp AS m
FROM ImputeP(Q) p, Result x, AvailableD y
WHERE match(x,q_m) AND match(y,q_a) AND x.S=y.S AND match(x,p)
GROUP BY x.A, x.S
```

In the implementation 6.5, we explain a variant of imputation queries which returns the pattern cover of partitions to be repaired. This setting is more efficient since partitions covered by the

same pattern are imputed with the same value. An example of such a rewriting is shown in the experiments.

6.5 Implementation

In this section, we describe an implementation of the imputation process using the pattern algebra defined in Section 3.4.

6.5.1 Partition Patterns Classification

The first step of the process corresponds to the classification of partition patterns into {Correct, Missing, Incorrect} query results. This classification depends on the query. The following definition shows how pattern queries can be used to achieve this classification.

Definition 6.12 (partition classification). *Given a data table D, its minimal pattern covers $\mathcal{P}^*(T)$ and $\bar{\mathcal{P}}^*(T)$ and an aggregate query Q generating partitions over attributes A (`GROUP BY` attributes), we can define the following pattern sets using pattern queries:*

- *Correct results patterns:* $Corr(Q(D)) = \widehat{\Pi}_A(\widehat{\sigma}_{cond}(\mathcal{P}^*(T)))\widehat{-}\widehat{\Pi}_A(\widehat{\sigma}_{cond}(\bar{\mathcal{P}}^*(T)))$
- *Missing results patterns:* $Miss(Q(D)) = \widehat{\Pi}_A(\widehat{\sigma}_{cond}(\bar{\mathcal{P}}^*(T)))\widehat{-}\widehat{\Pi}_A(\widehat{\sigma}_{cond}(\mathcal{P}^*(T)))$
- *Incorrect results patterns:* $Inc(Q(D)) = \widehat{\Pi}_A(\widehat{\sigma}_{cond}(\bar{\mathcal{P}}^*(T)))\widehat{\cap}\widehat{\Pi}_A(\widehat{\sigma}_{cond}(\mathcal{P}^*(T)))$

where $cond = \bigwedge_{a_i=c_i \vee a_i=*}$ *for each predicate* $a_i = c_i$ *in the* `WHERE` *condition of* Q.

Example 6.10. *Consider the query Q_{KwH}, and pattern $\mathcal{P}^*(D)$, $\bar{\mathcal{P}}^*(D)$ tables in Section 6.2. The following SQL[1] queries compute pattern sets that describe correct, incorrect and missing results.*

Correct results $Corr(Q(D))=$

```
SELECT Building B, Floor F, Week W
FROM P*(D) C
WHERE (B=25 or B=*) and (F in (1,2,3,*))
GROUP BY B,F,W
EXCEPT
SELECT Building, Floor, Week
FROM P̄*(D) M
WHERE (B=25 or B=*) and (F in (1,2,3,*))
GROUP BY B,F,W
```

[1] Refer to Chapter4 for pattern queries translation into SQL

Listing 6.2: SQL query computing correct results

Incorrect results $Inc(Q(D))=$

```
SELECT Building B, Floor F, Week W
FROM P̄*(D) C
WHERE (B=25 or B=*) and (F in (1,2,3,*))
GROUP BY B,F,W
EXCEPT
SELECT Building, Floor, Week
FROM P*(D) M
WHERE (B=25 or B=*) and (F in (1,2,3,*))
GROUP BY B,F,W
```

Listing 6.3: SQL query computing incorrect results

Missing results $Miss(Q(D)) =$

```
SELECT Building B, Floor F, Week W
FROM P̄*(D) C
WHERE (B=25 or B=*) and (F in (1,2,3,*))
GROUP BY B,F,W
INTERSECT
SELECT Building, Floor, Week
FROM P*(D) M
WHERE (B=25 or B=*) and (F in (1,2,3,*))
GROUP BY B,F,W
```

Listing 6.4: SQL query computing missing results

The resulting sets are illustrated in Table 6.3.

Tab. 6.3.: Pattern results classes for query Q result

$Corr(Q(D))$

B	F	W
25	1	1
25	2	1
25	2	2

$Miss(Q(D))$

B	F	W
25	1	2
25	*	3

$Inc(Q(D))$

B	F	W
25	2	1

The following definition uses the previous classification queries for computing the pattern cover $P(AvailableP(Q))$ of available correct results $AvailableP(Q)$ and the pattern cover $P(ImputeP(Q))$ of missing and incorrect results $ImputeP(Q)$ called the imputation sets. We will show in Section 6.5.2 how these pattern covers can be used to implement the final imputation step.

Definition 6.13 (Available and toRepair pattern sets)**.** *Given a data table D, its minimal pattern cover $\mathcal{P}^*(D)$ and $\bar{\mathcal{P}}^*(D)$, and pattern results sets defined in Definition 6.12. Pattern tables covering $ImputeP(Q)$ and $AvailableP(Q)$ are defined as follows:*

Imputation patterns: $P(ImputeP(Q)) = Miss(Q(D)) \cup Inc(Q(D))$

Availabble patterns: $P(AvailableP(Q)) = \pi_A \sigma_(a_j = *) \mathcal{P}^*(D)$, *with* $a_j \notin A$.

Example 6.11. *Table 6.4 shows the pattern covers for $ImputeP(Q)$ and $AvailableP(Q)$ tables.*

Tab. 6.4.: Pattern covers for partitions *"to impute" and* "available"

$P(ImputeP(Q))$

B	F	W
25	1	2
25	*	3
25	2	1

$P(AvailableP(Q))$

B	F	W
25	1	1
25	3	1
25	5	1
26	*	*

6.5.2 Imputation Query SQL Implementation

The imputation strategy is generated after steps 2 and 3. In step 4, imputation queries join available correct partitions with empty and incomplete data partitions to impute missing and incorrect values. The number of such imputation queries corresponds to the number of data partitions to repair, which might lead to important imputation costs. We take advantage of the pattern cover, which is in general more compact than the corresponding data tables, and perform imputation queries directly on pattern sets. By this, instead of using $AvailableD$and Mdatasets, we use their respective pattern covers in defined in Definition 6.13. The pattern imputation query is split into two queries. The first query is performed on the pattern covers and implies joining and aggregating correct partition. The second query then simply reports the results computed for each pattern to data partitions without requiring an aggregation. The following example illustrates this optimization we also use in our experiments.

Example 6.12. *Recall the imputation rule r_3 from Example 6.6 and two pattern sets $P(ImputeP(Q))$ and $P(AvailableP(Q))$ for some query Q. The following query computes a first intermediate "repaired" pattern set extended with new estimated values for its instances:*

```
CREATE TABLE repairedPatt as
 SELECT r.b, a.f, a.w, avg(a.KwH)
 FROM P(ImputeP(Q)) r, P(AvailableP(Q)) a
 WHERE (a.b = '*') AND (a.f = r.f OR r.f = '*')
   AND (a.w = r.w OR r.w = '*')
GROUP BY r.b, a.f, a.w
```

Listing 6.5: Repaired Patterns

The following query then joins $repairedPatt$ with the result $Result$ of query Q:

```
CREATE TABLE repairedResult as
 SELECT r.b, r.f, r.w, p.temp
 FROM repairedPatt p, Result r
 WHERE (r.b = p.b OR p.b = '*') AND (r.f = p.f OR p.f = '*')
   AND (r.r = p.r OR p.r = '*') AND (r.d=p.d OR p.d = '*')
```

Listing 6.6: Repaired Data

6.6 Experiments

In this section, we investigate the effectiveness and efficiency of our pattern-based approach for repairing analytic query answers. We consider the same dataset as for Chapter 4, to follow the quality improvement process. The temperature dataset $Temp$ is extracted from the Smart Campus use case.

In addition to the temperature measures, we consider a second data set $Occ(building, floor, room, occupation, area)$ that records campus rooms areas and occupations. This dataset is considered to support the imputation process.

Complete and empty pattern summaries are computed by the pattern generation algorithm described in Chapter 4. This algorithm produces pattern summaries with respect to the campus map and the calendar. Data and pattern tables cardinalities are reported in Table 6.5, and we can see table $Temp$ only covers about 5% of the spatiotemporal reference, whereas table Occ is almost complete.

variant x	Data D_x	Reference R_x	Complete P_x	Missing $\bar{P}_x$
Temp	1,321,686	24,615,600	11,268	10,777
Occ	10,131	10,742	1,109	263

Tab. 6.5.: Data and pattern tables cardinalities

Imputation rules: We designed a set of imputation rules over attributes in $Temp$ and Occ. Rules have variable schemas, allowing to match with different query patterns. The rules in Table 6.6 are listed in priority order following implicit (expert) knowledge about campus locations.

Take the example of rules r_2 and r_3 in Table 6.6. In the same floor, rooms are numbered sequentially in each floor side, where rooms in one side have odd numbers, and the other rooms side have even numbers, room 12 is next room 10. The room planning allows defining rules such as r_9 since the occupation of "TP" rooms is nearly the same as for "TD", which allows experts to assume the correlation between their temperature measures.

building, floor, room:

rule	b	f	r	←	b	f	r	agg
r_0	3334	x_f	x_r		_	x_f	x_r	min(temp)
r_1	_	x_f	_		_	x_f	_	avg(temp)

building, floor, room, month, day:

rule	b	f	r	m	d	←	b	f	r	m	d	agg
r_2	x_b	x_f	10	x_m	x_d		x_b	x_f	12	x_m	x_d	temp
r_3	x_b	x_f	11	x_m	x_d		x_b	x_f	13	x_m	x_d	temp
r_4	x_b	x_f	x_r	8	_		x_b	x_f	x_r	_	_	max(temp)
r_5	x_b	x_f	x_r	x_m	_		x_b	x_f	x_r	x_m	_	avg(temp)
r_6	_	_	_	x_m	x_d		_	_	_	x_m	x_d	avg(temp)

building, floor, room, month, occupation:

rule	b	f	r	m	o	←	b	f	r	m	o	agg
r_7	x_b	x_f	x_r	_	"TD"		x_b	x_f	_	_	"TP"	avg(temp)
r_8	x_b	x_f	_	_	"TD"		x_b	x_f	_	_	_	avg(temp)

Tab. 6.6.: Imputation rules for temperature measures

Queries: For our experiments, we define a set of aggregate queries over data tables *Temp* and *TempOcc* = $Temp \bowtie Occ$ (Table 6.7). All queries aggregate temperature measures, along with various filtering conditions (spatial, temporal, occupation, area).

```
 SELECT b, f, r, agg(temp)
 FROM temp
 WHERE b=3334
GROUP BY b, f, r
```

Listing 6.7: Query Q_1

```
 SELECT b, f, r,m, d, max(temp)
 FROM temp
 WHERE m in (6,7,8)
GROUP BY b, f, r, m, d
```

Listing 6.8: Query Q_2

```
 SELECT b, f, r,m, d, avg(temp)
 FROM temp
 WHERE f in (4,5) and r in (10,11)
GROUP BY b, f, r, m, d
```

Listing 6.9: Q_3

```
 SELECT b, f, r,m, avg(temp)
 FROM tempOcc
 WHERE b in (5354,5455) and o = ''TD''
GROUP BY b, f, r, m
```

Listing 6.10: Query Q_4

Tab. 6.7.: List of queries

6.6.1 Query Result Annotation

The query result annotation step consists in classifying each answer tuple as *correct*, *incorrect* and *missing*. We run an identification algorithm that implements queries as shown in Proposition 6.4.1 in Section 6.5. Table 6.8 classifies result patterns and partitions of each query in Table 6.7 into missing and correct categories. The answer data partitions are distributed between two classes

	$\|answer\|$	Correct		incorrect		missing		time (sec)
		patts	data	patts	data	patts	data	
Q_1	8	0	0	8	8	24	108	1.6×10^{-2}
Q_2	1,012	119	1 012	0	0	132	256,588	10.0×10^{-2}
Q_3	1,602	4	377	7	1,225	116	5,333	2.9×10^{-2}
Q_4	44	19	22	22	22	66	220	4.3×10^{-2}

Tab. 6.8.: Correct, incorrect, missing patterns, and data

correct and *incorrect*. Missing data is by definition not part of the query answer since they do not belong to the data table (when using $null$ values for representing missing information, missing data would correspond to $null$ values in the result).

Observe that the number of patterns does not represent the number of corresponding data partitions. Pattern summarize completeness of data partitions at different sizes ([25,*,*] covers much more data than [25,1,10] which corresponds to one room partition). More general patterns belong to a category set, more full they cover data partitions and imputing this single pattern extends to all subsumed data partitions. The running time is not impacted by the data table size (Q_3 vs. Q_4).

6.6.2 Query Result Imputation

The imputation strategy algorithm generates an ordered set of imputation queries to apply for each query "to repair" pattern set. Since all imputation queries share the same pattern summaries, we precompute the join between both data tables and the corresponding pattern tables and use the result in the imputation queries. Recall that rules are applied in the inverse order of their definition order. Take the example of the query Q_2. The ordered set of rules to repair the answer is $\{r_6, r_5, r_4, r_3, r_2\}$.The imputation process described in Section 6.4 is optimized in our experiments. Two imputation queries are executed. First, table $repairedPatt$ stores an aggregation estimation obtained by joining the pattern table $torepair$ with data table $available$. The obtained pattern table with freshly computed temperature values is then joined with the result table $Result$ to generate the final table $repairedResult$. This pre-aggregation at the pattern level improves query performance since it avoids the redundant aggregation of partitions, which are covered by the same patterns in the $Result$:

In Table 6.9, column *match patt.* records the number of patterns that can be repaired and column *cov. part.* shows the number of repaired partitions. The number of imputed partitions (column *imp. part.*) depends on the number of available correct partitions matching the rule's RHS for the repairing process. The number of remaining patterns (column *rem.*) corresponds to patterns that no rule has repaired.

Analysis Observe from the set of rules that only r_1 and r_0 are applicable for the first query. We start by applying the rule r_1 with less priority, imputing 109 partitions over 136. The rule r_0 repairs fewer tuples since it requires repairing a room with the average observed temperature for the same room during the year. Many rooms are not equipped with sensors at all, which explains the poor number of imputation update achieved with this rule. In the end, 27 results remain without any estimation. We found for example that all missing partitions matching the patterns $(3334, JU, *)$, $(3334, SS, *)$ and $(3334, SB, *)$ could not be imputed since no temperature measure is available for these floors in all campus buildings. Note that both applied rules require a completion using the same floor, but no recording sensor is available for these floors, preventing imputation. All other queries could be repaired completely by applying all matching imputation rules. These experiments

Query	rule	match. patt.	cov. part.	imp. part.	rem.	run time (10^{-3} sec)
Q_1	r_1	32	136	109	27	2.40
	r_0	32	136	40		1.58
Q_2	r_6	132	256,588	256 588	0	27,910.00
	r_5	132	256,588	9936		720.00
	r_4	132	86459	10261		3,260.00
	r_3	25	9292	920		1.74
	r_2	25	10212	920		1.84
Q_3	r_6	127	6558	6558	0	13,890.00
	r_5	127	6558	1084		2,240.00
	r_4	25	465	10261		3.70
	r_3	123	5333	331		1,590.00
	r_2	74	1225	342		170.00
Q_4	r_8	88	242	242	0	4.78
	r_7	88	242	66		0.15

Tab. 6.9.: Imputation results

demonstrate that the utility of imputation rules depends on the existence of correct answers and the expert's knowledge about the sensor network configuration and behavior.

6.7 Summary

In this chapter, we explored using imputation rules for improving the completeness and the correctness of aggregate query results. We proposed a new imputation model that extends the pattern model defined in Part 1 for representing the completeness of data and query answers.

The imputation rules translate expert knowledge at different partition levels, corresponding to variable aggregation levels. The rules semantic include to aggregate different partitions results, thanks to an imputation function. We formalized a complete strategy that starts with assessing the completeness of a query result. The strategy includes four steps: 1-partitions classification { correct, missing, incorrect}, 2-candidate rules choice, 3- strategy establishment, and 4- imputation queries generation.

The implementation of the imputation process is ensured by pattern algebra. In the first step, three pattern queries allow classifying query results as correct, incorrect, or missing partitions. We proposed an optimized version for applying imputation queries to pattern sets instead of using data tables. This avoids computing queries with aggregation over large datasets (correct partitions used for imputation).

We conducted a set of experiments on real datasets and took advantage of our knowledge of the campus, and the consumption history in localities to propose temperature imputations rules. The experimental results show that the imputation process was successful at repairing most of the queries results, in reasonable times, compared to queries running time. Our analysis leads to the conclusion that the richness of imputation rules and their adequacy with data features, is a strong condition for achieving a successful imputation.

Multiple imputation rules are applied to the same query result. We chose in this work to assign the imputation rule result with the highest priority to repair matching partitions. An interesting alternative could consider the multiple imputation mechanism, that merges many imputation results, using the mean value, or other weighted formulas. This option could offer a better imputation quality but implies additional cost.

Part III

Reasoning With Fragment Summaries

7 Summarizing and comparing data fragments using patterns

> *Go as far as you can see, and you will see further.*
>
> — **Zig Zigler**
> Writer

Contents

7.1 Introduction

The main contribution of this thesis is the pattern model and algebra for representing relative completeness. Data summarization is a database research approach concerned with producing compact representations for data tables, for optimizing queries evaluations or performing analysis tasks.

We explore in this chapter an opportunity to generalize our pattern model for achieving summarization. In the original pattern model, two data fragments are considered, built according to one attribute value. Data partitions having a null value for this attribute belong to the missing partitions fragment, whereas partitions with constant values constitute the complete partitions fragment. We extend this model, to cover more fragment types generated by classifying data parts according to any filtering conditions. Summarization then aims at providing fragments with exhaustive pattern descriptions to achieve analysis tasks. Unlike classical summarization contributions, we do not consider the compactness as a guideline, but we strive to produce complete summaries. The advantage of accessing complete summaries is the possibility to query summary tables, for analysis, but also to exploit reasoning rules for explanation and prediction tasks.

This chapter is organized as follows:

- Section 7.2 shows a use case that motivates our research direction;
- Section 7.3 formalizes the data model by generalizing constrained tables to constrained fragments. We also define how pattern tables can be used as summaries.
- Section 7.4 defines a formal framework for reasoning with fragment summaries.
- Section 7.5 describes experimentation results conducted over a benchmark dataset.
- The contributions are summarized in Section 7.7.

7.2 Motivation

In order to motivate the need for a generalized model, let us consider an example that goes beyond the sensor network use case. The ADULT dataset is a public dataset [1] with census data about population income. It counts $32,561$ rows with 14 attributes. We restrict to seven representative attributes commonly used in other studies. The remaining attributes are correlated with others or include a high rate of missing values. The considered dataset attributes are one numerical attribute

[1] https://archive.ics.uci.edu/ml/index.php

age and seven categorical attributes *workclass*, *education*, *marital-status*, *occupation*, *race*, *sex* and *income* { <50k, ≥50k }.

Consider the analysis task that aims to explore the relationship between the income and the other dataset features. The dataset can be fragmented into two classes. The high-income class refers to all individuals with an income >= 50k, and the remaining individuals are in the low-income category. To explain each fragment profile, we think first about computing pattern minimal covers, in a similar way to completeness summaries study (detailed definitions are provided in Chapters 3 and 4).

However, we face a setting issue. The data table does not fit our model for a constraint violation. Indeed, in our data model, we require data tables to include a primary key to clearly distinguish the fragment of tuples with values from the fragment of tuples with null values (no tuple can have both a constant value and a null value for the same attribute). For the current dataset, two individuals with the same attribute values may belong to both, the high and the low-income fragment.

To deal with this issue, we consider a third data fragment that groups Non-Distinguishable individuals. If a data tuple appears in both fragments, it cannot be used to characterize one of the two main categories (low-income and high-income) and is discarded from both. By eliminating conflicts, pattern summaries can be created for each fragment. We show in Table 7.1 an illustration of the resulting summaries, describing the fragments High, Low, and ND generated by the subset of Male-White-Married individuals.

High	age	workc	education	occupation
	40-50	*	*	Armed-Forces
	*	federal-gov	doctorate	*

High income

Low	age	workc	education	occupation
	< 20	*	*	*
	*	never-worked	*	*
	*	*	preschool	*

Low income

ND	age	workc	education	occupation
	*	*	masters	*
	> 60	*	doctorate	*

Non Distinguishable

Tab. 7.1.: "Male white married" fragments summaries

We learn from these summaries what follows:

1. all white married male soldiers between 40 and 50 and all employees of the federal government with a Doctorate have a high income,

2. all young white males who are married or have never worked or with a preschool diploma have a low income and

3. it is not possible to decide for white married males with a Masters, or which are old with a Doctorate whether they belong to the high or low-income class.

We show in Section 7.4 how this kind of detailed analysis can be done by evaluating simple SQL queries over pre-computed fragment summaries.

In the next Section 7.3, we define the data fragment model, and generalize the pattern model, by allowing constrained tables with no primary key, introducing by the same the Non-Distinguishable fragment.

7.3 Fragment and Summary Model

In this section, we introduce the notion of fragment summary as a comprehensive description of all complete data categories in a data fragment.

7.3.1 Data Fragments

Definition 7.1 (data fragments)**.** *Let D be a relational data table with attributes X and $A \subset X$ be a subset of attributes of F. A fragment F over D is defined as a subset of tuples in D which satisfy a filtering condition $cond$ projected on A : $F = \pi_A(\sigma_{cond}(D))$. We call F a data fragment, and the couple (F,D) a constrained fragment.*

The complementary fragment $\bar{F}$ of a fragment F is defined as $\bar{F} = \pi_A(\sigma_{\neg cond}(D))$. Then, by definition, $F \cup \bar{F} = \pi_A(D)$.

Definition 7.2 (conflict fragments)**.** *Let (F, D) and (F', D) be two constrained fragments of some data table D defined over the same set of attributes. The conflict fragment of F and F' is the constrained fragment (R, D) which contains all tuples which appear in F and F': $R = F \cap F'$.*

Example 7.1 (Adult dataset fragments)**.** *Recall the example in Section 7.2. Data table ADULT has schema: X={age, workclass, maritalStatus education, occupation, sex, race, income }. Let $A = X - \{income\}$ and*

- $F_{High} = \pi_A(\sigma_{income='+50'}(Adult))$

- $F_{Low} = \pi_A(\sigma_{income='-50'}(Adult))$

Since A is not a key in $ADULT$, the rest $R = F_{High} \cap F_{Low}$ might not be empty.

7.3.2 Fragment Summaries

We assume for the following that the reader is familiar with the pattern model proposed in Chapter 3.

Definition 7.3 (Pattern satisfaction)**.** *A constrained fragment $T = (F, D)$ over fragment attributes A* satisfies *a pattern p if the instance of p in the fragment F is equal to the instance of p in $\pi_A(D)$: $I(p, \pi_A(D)) = I(p, F)$. By extension, a constrained fragment T* satisfies *a pattern table P if T satisfies all patterns in P.*

We also say that a pattern p or a pattern table *characterizes* the constrained fragment (F, D).

Observe that Definition 7.3 exactly corresponds to Definition 3.6 of pattern satisfaction in Chapter 3. We apply the notion of minimal pattern cover based on the modified definition of pattern satisfaction:

Definition 7.4 (Fragment summary)**.** *Let $T = (F, D)$ be a constrained fragment of some data table D. The minimal cover $\mathcal{P}^*(T)$ (Proposition 3.3.4, page 50) of the constrained fragment T) is called a* fragment summary *and noted $\mathcal{P}^*(T)$.*

The fragment summary $\mathcal{P}^*(T)$ is not necessarily compact since there might exist a subset of patterns $P' \subset \mathcal{P}^*(T)$ where $I(P', T) = I(\mathcal{P}^*(T), T)$ (all categories described $\mathcal{P}^*(T)$ are "subsumed" by the patterns in P'). This makes our summarization model different from other models which try to maximize the compression ratio whereas our fragment summaries are compact representations of *all* characteristic data categories.

By Definition 7.3, a fragment summary might only cover a strict subset of its fragment (F, D), *i.e.* $I(\mathcal{P}^*((F, D)) \subset T[A]$. This happens when the rest fragment is not empty.

Definition 7.5 (Non-Distinguishable summary)**.** *Let $T = (F, D)$ and $T' = (F', D)$ be two constrained fragments of some data table D and $R = F \cap F'$ be the conflict fragment. The fragment summary $ND(F, F') = \mathcal{P}^*(R, D)$ is called the non-distinguishable (ND) summary of F and F'.*

7.4 Reasoning with Fragment Summaries

7.4.1 Formal Reasoning Model

Fragment summaries are concise characterizations of data fragments and can be used for analyzing and comparing data fragments extracted from a given reference data set. By the previous definition of fragment summary, the following constraints hold for all constrained fragments $T = (F, D)$ where $D \neq \emptyset$:

If $p \in \mathcal{P}^*(T)$:

- for $p' = p$ and any specialization p' of p, its instance $I(p', F)$ in F is *complete* with respect to D.
- for any strict generalization p' of p ($p \sqsubset p'$), its instance $I(p', F)$ in F is *incomplete* with respect to D.

If $p \in ND(F, F')$:

- for $p' = p$ and any specialization p' of p, its instance $I(p, F)$ in F and its instance $I(p, F')$ in F' are *identical* ($I(p, F) = I(p, F')$) and *complete* with respect to D.

Based on these properties, all patterns in the summary of a fragment T can be classified into:

1. complete (C_F) patterns which completely characterize the fragment,
2. incomplete (I) patterns which have specializations in both fragments summaries, and
3. non_distinguishable ($ND(F, F')$) patterns which characterize tuples in the conflict table of two fragments F and F'.

Example 7.2. *Recall the adult dataset from Section 7.2. Table 7.1 shows subsets from each fragment summary: "high income" (High), "low income" (Low) and "non distinguishable" (ND). For space reason, we assume in the following that these tables represent the entire fragment summaries, to allow a clear representation and fast reasoning. The pattern table in Table 7.2 shows some patterns from the ADULT dataset and the class of each pattern is defined before. Figure 7.1 is a tree representation of the same patterns where each node at some level i corresponds to a pattern of length i. Each level i corresponds to patterns with i constant attributes. The wildcard pattern $[*]$ is the root (level 0 with no constant attributes), the first level corresponds to patterns $[*, *, masters, *]$, $[40 - 50, *, *, *]$, $[*, *, doctorate, *]$ and $[< 20, *, *, *]$ (with one attribute). All patterns in summary $\mathcal{P}^*(High)$ are labeled C_{high} (in blue) and all patterns in summary $\mathcal{P}^*(Low)$ are labeled C_{low} (in red). All ancestors of both kinds of patterns*

	age	workc	education	occupation
I	$*$	$*$	$*$	$*$
C_{low}	< 20	$*$	$*$	$*$
I	$40 - 50$	$*$	$*$	$*$
C_{low}	$*$	$*$	preschool	$*$
ND	$*$	$*$	masters	$*$
I	$*$	$*$	doctorate	$*$
C_{low}	$40 - 50$	$*$	preschool	$*$
ND	> 60	$*$	doctorate	$*$
C_{high}	$*$	federal-gov	doctorate	$*$
C_{high}	$40 - 50$	$*$	$*$	armed-forces
ND	$40 - 50$	never-Worked	$*$	armed-forces

Tab. 7.2.: A set of patterns in the adult dataset

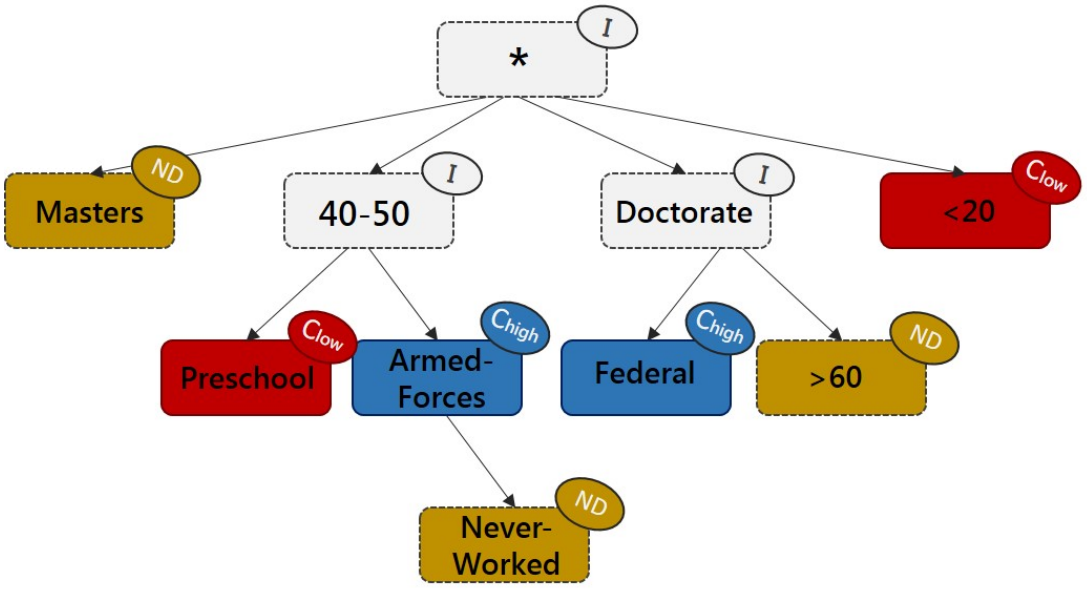


Fig. 7.1.: Labeled fragment summary lattice

nodes are I-patterns. Patterns $[*, *, masters, *]$ *and* $[> 60, *, doctorate, *]$ *are non-distinguishable (ND) (in yellow). This means that for all categories of people with a Masters degree or with a Doctorate and older than 60 years, there exist at least one individual with a low income and one individual with a high income. The pattern* $[40-50, never-worked, *, armed-forces]$ *is also non-distinguishable (ND) since it specializes the high income pattern* $[40-50, *, *, armed-forces]$ *and the a low income pattern* $[*, never-worked, *, *]$.

7.4.2 Reasoning with Queries

Let $RA_{ext} = RA \cup \{\triangleright, \triangleleft\}$ be the relational algebra extended by two operators $\triangleright$ and $\triangleleft$ where

1. $\triangleleft_A(P)$ generates for a given pattern table P an *equivalent* pattern table P' where all values of attributes $a_i \in A$ are constant values and

2. $\triangleright_A(P)$ generates for a given pattern table P an *equivalent* pattern table where there exists no pattern p and subset $S \subseteq P'$ with more than one pattern which is equivalent to p : $\nexists p, S \subseteq P', |S| > 1 : \{p\} \equiv S$.

Using this extended algebra, we can define queries over fragment summaries. First we can define two operators $\triangleright(T) = \triangleright_A(T)$ and $\triangleleft(P) = \triangleleft_A(P)$ which compute the summary of some fragment T and the instance of a pattern table P respectively. Unfolding $\triangleleft$ can directly be translated into the relational algebra by joining the pattern table with the data table, whereas folding $\triangleright$ over a set of attributes needs recursion which is not expressible in relational algebra (see Section 4.3 in Chapter 4 for implementations of $\triangleright$). Based on this formalization and the equivalence of the relational algebra and SQL, it is then possible to rewrite any pattern query *without folding* into a relational SQL query over summaries and their data tables. We will illustrate this by two examples.

First, selection can be applied for checking if some given pattern p is a specialization/generalization of a pattern $p' \in P$. For example, when considering the summary P in Table 7.2, pattern $[40-50, *, doctorate, armed-forces]$ is complete in fragment $High$ if the result of following query over the summary $\mathcal{P}(High)$ is not empty:

```
select * from P(High)
where (age='40-50' or age='*')
and (education='Doctorate' or education ='*')
and (occupation='Armed-Forces' or occupation='*')
```

It is easy to see that the result contains pattern $[40-50, *, *, armed-forces]$.

Joining two summaries needs unfolding. Consider two summaries $P_1(age, workc)$ and $P_2(workc, education)$ of two fragments F_1 and F_2 of data table ADULT. The natural join of these two summaries generates a new summary $P(age, workc, education)$ characterizing the fragment $D_1 \bowtie D_2$:

```
select P1.age,R.workc,P2.education
from P1, P2, Adult
where (P1.age=Adult.age or P1.age=*)
and (P1.workc=Adult.workc or P1.workc=*)
and (P2.workc=Adult.workc or P2.workc=*)
and (P2.education=Adult.education or P2.education=*)
```

Observe that we have to join both sumaries with the data set on attribute $workc$ to avoid the generation of complete but empty patterns. It is also easy to see that this table might not be minimal and has to be re-folded over attribute $workc$ to obtain a minimal summary.

7.5 Experiments

We run a set of experiments on the ADULT dataset to evaluate the effectiveness and the efficiency of the fragment summary model. The ADULT dataset counts one numerical attribute, *age* ranging from 17 to 90, and seven categorical attributes *workclass* (private, federal-gov...), *education* (Bachelors, Doctorate,...), *marital-status* (married,divorced..), *occupation* (tech-support, Sales,...), *race* (black,white,...), *sex* (female,male), *income* ($<$50k,$\geq$50k).

The $incomeclass$ summarization considers two fragments $High$ and Low following the income attribute. We run the pattern generation algorithms described in Chapter 3, first with the numerical age attribute (raw dataset), and a second time by aggregating ages into ten years intervals $]20, 30],]30, 40], ...$ (binned dataset). We also vary the number of attributes by creating three datasets, where D_1 is the original dataset including all attributes, D_2 removes attributes $workclass$ and $race$, and D_3 removes the attribute sex from D_2. The obtained results are reported in Table 7.3 and Figures 7.2 and 7.3. We can see that by decreasing the number of attributes, the coverage of the corresponding summaries decreases, and the size of the rest increases. We also can see that the size of the summaries (number of patterns) might become higher than the data set it describes. For example, the summary for High income in $D1$ is larger than the fragment it describes. This size is due to the fact that a summary precisely characterizes the fragment with respect to the whole data set and therefore contains more information about the data fragment than the fragment itself. The table also shows that reducing the domain of attribute Age by aggregating values (numerical to categorical) leads to increasing the size of *ND*, which can be explained by the fine-grained correlation between the age attribute and the income.

data set	Age : numerical					
	High income		Low income		Not-Distinguishable	
	Data	Patterns	Data	Patterns	Data	Patterns
D_1	4382	5485	20848	10924	7544	1995
D_2	2283	1786	18164	4736	12114	1859
D_3	1712	1106	16964	3131	13858	1762

data set	Age : categorical					
	High income		Low income		Not-Distinguishable	
	Data	Patterns	Data	Patterns	Data	Patterns
D_1	1591	1813	15227	4096	15743	1454
D_2	394	284	11235	971	20932	736
D_3	184	133	9363	493	23014	645

Ag:age, **Wo**:Workclass, **Ed**:Education,**MS**:Marital-status,**Oc**:Occupation,**Ra**:Race,Se:Sex

Tab. 7.3.: Income classes summaries with variable attributes sets

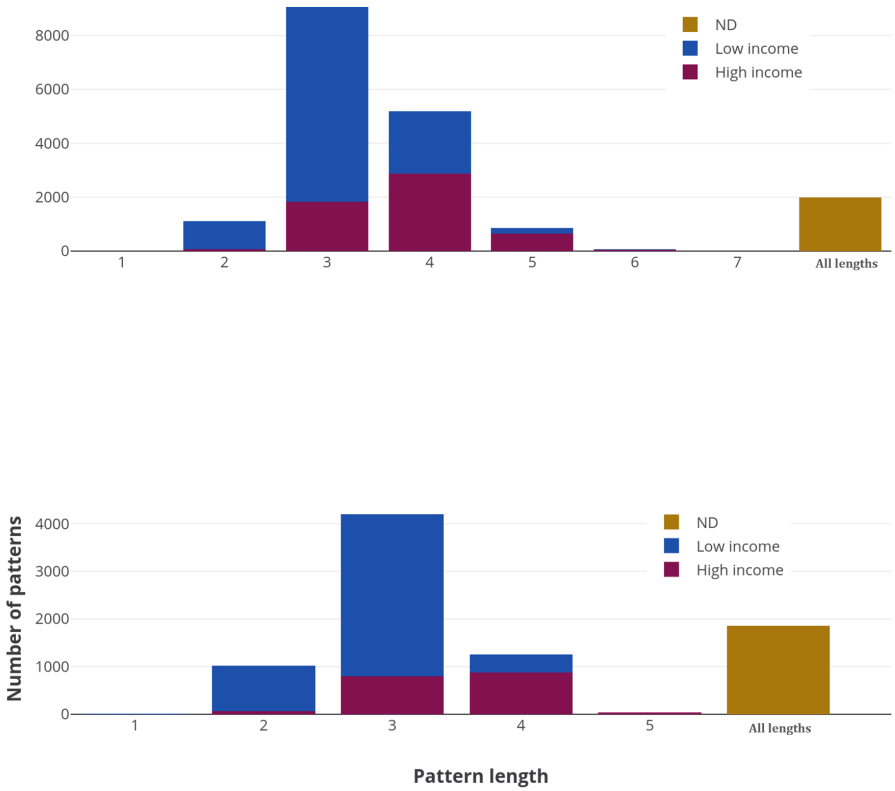


Fig. 7.2.: Income classes pattern summaries with variable attributes sets (raw dataset)

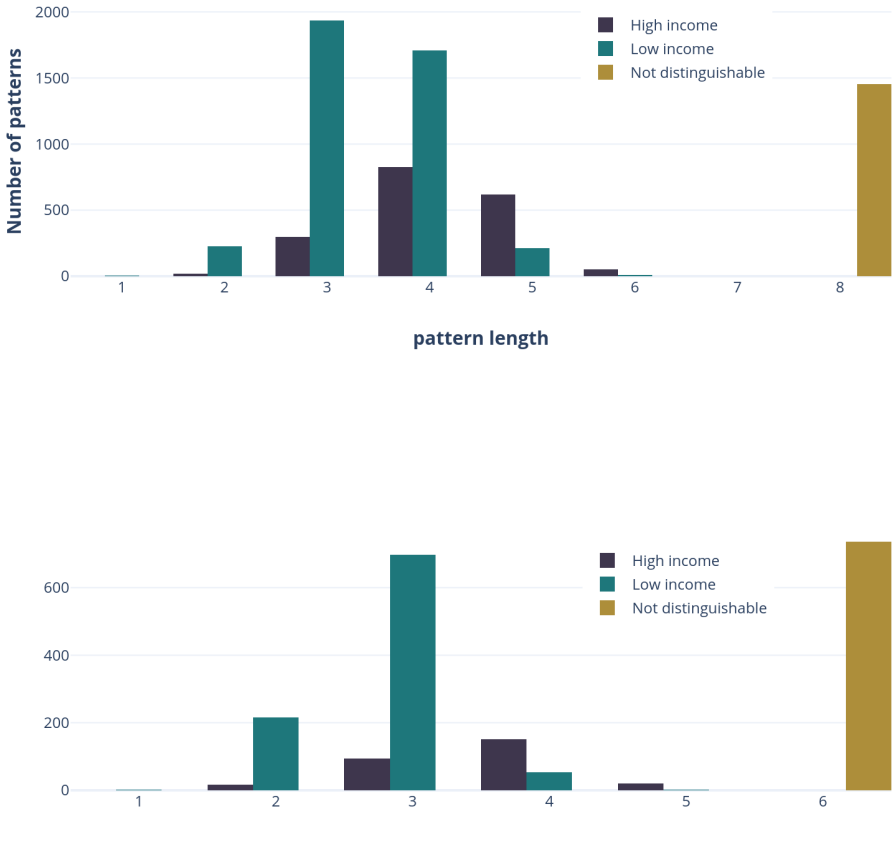


Fig. 7.3.: Income classes pattern summaries with variable attributes sets (binned dataset)

Figures 7.3 and 7.3 show the distribution of patterns according to their length (number of constant values in a pattern). For example, in D_2 we infer that 74 patterns of length 2, are in the high-income summary, while $1,033$ belong to the low-income summary. This means that for data covered by both patterns sets, we can decide about their income by knowing only two attributes among 7. We also observe the evolution of the size of the pattern set corresponding to non-distinguishable data (in yellow), increasing with attribute set restriction.

The execution time increases with the number of attributes. The larger the attribute set is, the more attribute combinations have to be checked during pattern generation. The Table 7.4 summarizes the running times for fragment Low in all data sets.

Data set	Number of attributes	Running time (s)
D_1	7	259.19
D_2	5	60.96
D_3	4	32.87

Tab. 7.4.: Execution time depending on attributes number

7.6 Related Work

The general shared goal of summarization is to reduce the data size by preserving enough useful information for a specific goal like supporting approximate query answering [Vog+06]. In general, we can distinguish between several families of summarization approaches for structured data. *Semantic* approaches exploit external semantic knowledge such as fuzzy thesauri and linguistic variables to generate a concise approximate human-understandable description of a large data-set [Ras+02; SP+05]. They can also use OLAP hierarchies for guiding the summarization of multidimensional data based on a class hierarchy [Buc+03; Cuz+08].

Hierarchical summarization is another family that was investigated in [Mah+11] and [SC+09] and consist in building value lattices to cluster and aggregate table fragments [Mah+11; SC+09]. *Statistical summarization* is more related to the notion of "summarizability" property [Len+97] and studies the mathematical properties of aggregation functions to ensure that they yield "correct" values when applied to partitioned data. Similarly, quotient cubes introduced in [Lak+02] are concerned with building a lossless summary of hierarchical OLAP tables while preserving the roll-up/drill-down semantics of the cube. The summary is meant to have an optimal size for evaluating monotone aggregation functions (COUNT, MIN, MAX, SUM) and locally optimal for non-monotone functions (AVG).

Table reduction approaches like Tabsum [Lo+00] allow for compressing tables for the sake of being displayed on small devices. Compression can be performed on tuples by compressing their values using interval encoding or on columns by merging them; both numerical and categorical attributes are addressed. Another family of approaches that build a succinct and *lossless* representation of relational data is factorized databases [Olt+16]. The underlying idea is to exploit the properties of the relational algebra, in particular, distributivity of the Cartesian product over union to represent the compute query answers in an effective and efficient manner. This representation opens new opportunities for query optimization of complex join and aggregation queries. A slightly different notion of summary is introduced in [Mam+13]. It consists in vertically partitioning an input table into as many tables as there are sub-set of attributes with a strong correlation in the input table. This approach, however, seems to be very close to the well-known problem of normalization using approximate functional dependencies.

Our work is more reminiscent to the family of *pattern mining* approaches [Lee+14; Koo+09; Che+13] where patterns are used for summarizing data under different perspectives. In [Che+13], patterns are used for data compressing, and a measure of representativeness is used for selecting only a subset of patterns based on their coverage of the data. The approach uses the Minimum Description Length principle for guiding the extraction process and searches for a minimal patterns-set with maximal informativeness. Differently from [Che+13], our approach is concerned with extracting an exhaustive set of the most general patterns characterizing a table fragment w.r.t. the entire data. Doing so allows for reasoning about data fragments using the patterns as will be described later.

Query evaluation with summaries Summaries are extensively used for optimizing aggregation queries in statistical databases and OLAP. The main objective here is to generate summaries which allow to estimate the information loss and generate approximate query answers. Factorized Databases [Olt+16] are succinct lossless representations of relational data and can be used for rewriting standard SQL queries of the source data into optimized queries over the factorized data representation as described in [Olt+16]. As illustrated in the introduction, our fragment summaries can directly be used for annotating analytic query results with meta-data about the result completeness and correctness.

7.7 Summary

In this chapter, we explored the opportunity to generalize our pattern model for data summarization purposes. We first extended the data model to fit additional fragment types, obtained from a data table by applying a particular class of queries. The generalization process led to consider data tables that do not include a primary key, entailing the particular notion of Non-Distinguishable fragment. This new setting required adapting the pattern model, to propose summaries that respect minimal covers properties. These sets are by definition reduced pattern sets, covering entirely the data fragment, without redundancy, and are disjoint.

A formal reasoning framework allows for inferring knowledge about patterns that do not belong to fragment summaries. Thanks to pattern properties (specialization, generalization), the model guarantees the completeness of the inference process; we can decide for any pattern tuple whether it belongs to a particular fragment or the non-distinguishable pattern set.

For our experiment, we use the adult dataset public dataset to evaluate our fragment summaries model. We executed the fragment summaries computation to three datasets with variable schemas, to check the variability of summaries according to the number of attributes. We also resized data categories, by using a dataset with the age attribute within a categorical domain, instead of a numerical, to show the domain length influence on the non-distinguishable fragment. Varying the number of attributes showed that 1- the Non-Distinguishable fragment is more significant with fewer attributes, since fewer features allow to distinguish the income class, 2-the more attributes we consider, the less a summary is compact, since the tuples redundancy decreases. Experiments allowed to check the effectiveness of our model, given the semantics of the generated covers that correspond to our observations, and similar contributions results on this dataset.

Part IV

Conclusion and Future Work

Conclusion and perspectives

8

> *Winning the prize was not half as exciting as doing the work itself.*
>
> — **Maria Goeppert Mayer**
> Nobel prize in Physics

Contents

8.1 General Conclusion

In this thesis, we addressed several problems related to data completeness in a relative information setting. The first contribution regards the completeness representation, where we proposed a pattern model for representing the complete and missing partitions in data tables. Compared to other pattern-based completeness models [Raz+16], our model is based on reference datasets which enables the automatic derivation of minimal pattern covers for describing complete and *missing* data, which is not possible in the Partially Open World Assumption. To support the pattern model, we extended the relational algebra by defining two new operators that operationally connect data tables to pattern tables. A fold operator can generate the pattern cover of a data table, and inversely, the unfold operator returns a pattern table instance. Based on this extended algebra, a pattern algebra to query pattern tables has been introduced. This algebra is complete and sound regarding the SPJUD fragment [Abi+95] of the relational algebra. Pattern queries allow computing pattern minimal covers for assessing the completeness of query results. All pattern algebra operators, except $fold$, can be expressed in the relational algebra and we proposed two algorithms that implement the folding operator. Extensive experiments have been conducted to check the efficiency of these algorithms and evaluating the cost of applying pattern queries for result annotation. Our experimental results confirmed the efficiency and effectiveness of our proposed contributions and confirmed our assumptions about high compactness of pattern tables for real-world datasets.

We dedicated the second part of this manuscript to our contribution to query result imputation. We defined a query-driven imputation model and strategy for repairing aggregate query results. We took advantage of the pattern model to define a rule-based imputation model for data partitions. The imputation rules aim at repairing missing and incorrect results by new values, which at estimated by aggregating correct aggregated values. We created a complete process that starts by classifying aggregate results into correct, incorrect, and missing categories using pattern queries. The imputation process counts four steps and returns a set of SQL imputation queries. A set of experiments have been performed to check the correctness of our model, and the results show that rule-based imputation strategies can significantly improve the completeness and the correctness of aggregate query results.

The final part of this manuscript presents our last contribution, where we explored the possibility of generalizing the pattern model to summarize and compare any data fragments. The pattern model is used to define pattern minimal covers for data fragments as summaries. We also defined a first formal framework for reasoning about fragment summaries.

In the following sections, we present some future research directions for advancing our contributions.

8.2 Perspectives on the Completeness Model

This section covers future directions for contributions summarized in Part 1.

8.2.1 User-Friendly Interface

We consider the development of a user interface that implements the pattern model and algebra for completeness representation. Users can access database tables and create constrained tables for their completeness studies. The Figure 8.1 illustrates an overview of this interface. Observe that the user can constitute constrained tables $T(D, R)$ either by browsing database tables or by creating views. Then, she can run the $FoldData$ Algorithm (refer to Section 4.3 from Chapter 4) for computing minimal pattern covers of complete and missing partitions. The user can browse the entire produced pattern tables, or define SQL queries for analyzing the results.

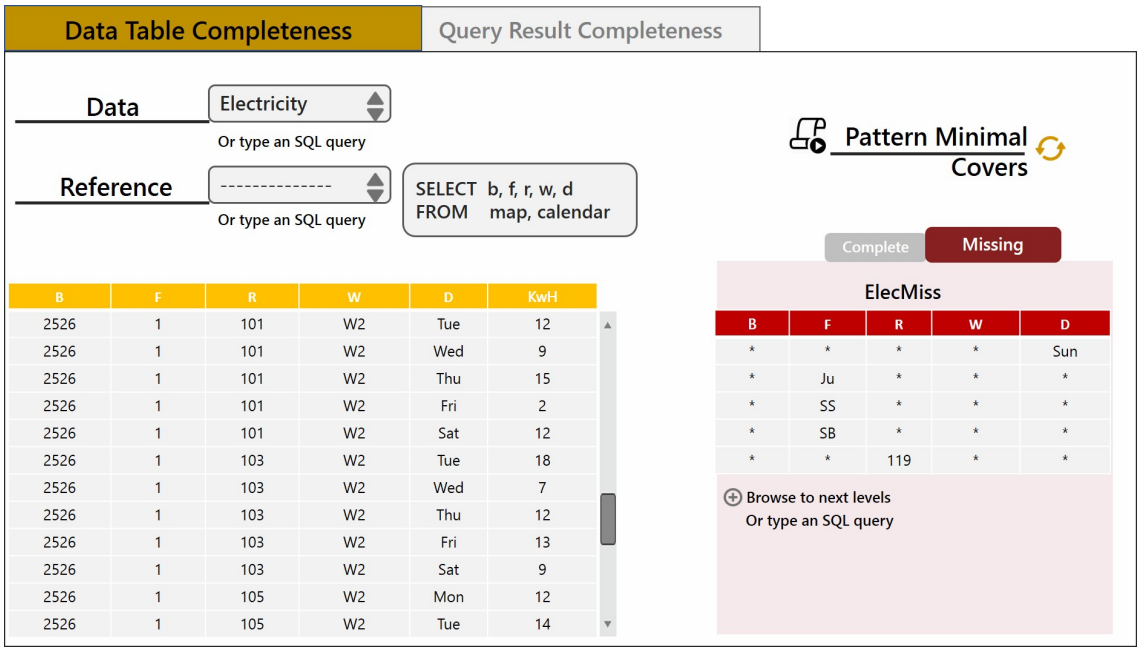


Fig. 8.1.: User interface overview: Data completeness

The second major feature of the user interface is the completeness assessment for query results (Figure 8.2). The user can express her SQL queries over data tables and visualize the results. The application should automatically produce the corresponding pattern query that computes the minimal covers for the query result. Similar to data covers, the user can then browse the provided pattern tables or ask SQL queries.

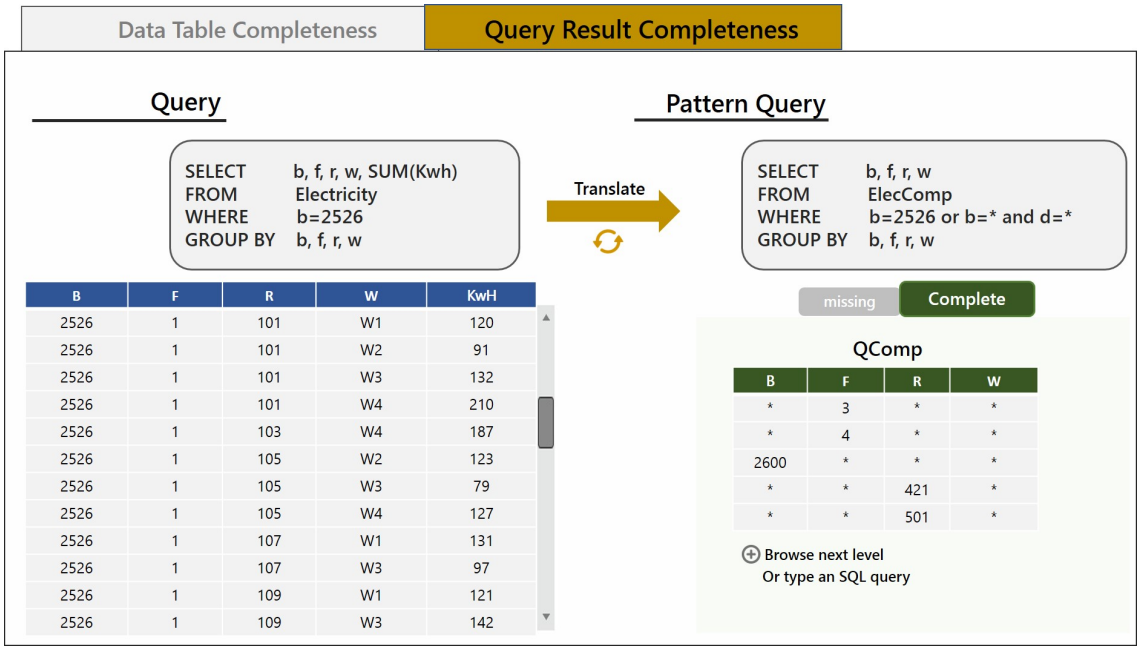


Fig. 8.2.: User interface overview: Query Result completeness

Pattern Query Optimizer The previously described user interface implies the implementation of an algorithm that automatically derives a pattern query $\hat{Q}$ for computing the pattern cover of a data query Q. Currently, for each SQL query defined over constrained tables, a pattern query can be defined using the pattern algebra as shown in Section 3.5. Pattern queries are related to relational queries by the following formula:

$$\widehat{Q}(\mathcal{P}^*(T)) = \triangleright(Q(\triangleleft(\mathcal{P}^*(T))))$$

We have shown for pattern operators how this formula could be applied along with relational rewriting rules and pattern fold/unfold operators properties to derive optimized pattern operators definitions (Section 3.4). The challenge is then to define an optimizer that uses these rules to return an optimized pattern query for each data query.

Translating Pattern Queries into SQL The optimized pattern query can be translated into SQL, as shown by the examples in Chapter 4. Since any pattern query (without $folding$) can be expressed in the relational algebra, this compilation step could directly apply well-known rules for rewriting algebraic expressions into SQL. The final folding step can be implemented using the $FoldPatterns$ algorithm presented in Chapter 4.

8.2.2 Incremental Minimal Covers

In Chapter 4, we proposed a folding algorithm that implements the fold operator for pattern minimal cover computation. Given a constrained table $T(D, R)$, the minimal covers $\mathcal{P}^*(T)$ and $\bar{\mathcal{P}}^*(T))$ strictly cover all complete, respectively missing, data partitions. These pattern minimal cover lose their coverage property when the constrained table is updated. We can distinguish between the following cases:

- Updates completing/reducing the data table D over the same reference R: For example, in a sensor network, we can acquire new data measures for last year records. The temporal and spatial references remain the same, but the completeness of the covered localities increases. For insertion, some missing patterns might become complete, and some complete patterns might be generalized. Symmetrically, for deletion, some complete patterns might become incomplete or missing, and some missing patterns might be merged int a generalized.

- Updates extending/reducing the reference: If a new sensor is installed in a new locality, the reference should be extended to this new area and patterns describing spatial completeness (wildcards for spatial attributes), should be rechecked. In this case, some initially complete pattern may be split (specialized), or some missing partitions patterns may be generalized (merged)

- Updates on both data and reference: The previous changes may occur together and trigger the generalization and specialization of patterns.

The current folding algorithms do not take account of these updates, and pattern tables have to be recomputed from scratch. This is inefficient, especially when the data and/or reference tables are often updated. An incremental algorithm could avoid recomputing the entire cover. In all situations, updates impact only a subset of tables, and some patterns remain valid for the new constrained table. Only patterns that are no longer satisfied should be fixed, for efficiency.

Query Independence of Updates [Lev+93] is a mechanism that allows deciding if a query is independent of certain data update, i.e., the result of the query is different before and after the update. A. Levy et al. [Lev96] uses this approach for assessing query answer completeness. A challenging direction is to adapt to this problem setting to our updates issue. A pattern can be considered as a constraint view on a data table and reference table, which remains valid if its instance is independent of insertion or deletion updates. We can explore this intuition for defining an algorithm that can decide if a given update impacts a given pattern.

8.3 Perspectives on Query Result Imputation

This section describes some possible future directions for the contributions of Chapter 6.

8.3.1 Imputation Quality Model

The data imputation strategy defined in Section 6.4 makes the assumption that the imputation output is always more accurate than the original query result to impute (missing or incorrect). This assumption is too strong, and an open research direction consists of defining more precise quality metrics for assessing the accuracy of the estimated imputation values.

The imputation rules are defined to repair missing or incorrect aggregations, using correct partition results. A possible quality metric quantifies the size of the correct data partitions satisfying the imputation rule and involved in the imputed value computation. For example, for the electricity measure table, consider a query that asks for weekly consumption per floor. The rule $r : (B : \_, F : f, M : m, W : \_) \leftarrow (B : \_, F : f, M : m, W : \_), avg(KwH)$ repairs incorrect results with all correct results obtained for the same floor and month. We can consider that the estimation accuracy then depends on the number of available correct estimations.

Given the imputation rule $r : q_m \leftarrow q_a$, to use over a data table D, with a pattern cover $\mathcal{P}^*(D)$. To repair the query Q answer, a simple quality metric could be defined as a function:

$$f(r, Q, P) = \sum_{p_i \in match(q_a)} |I(p_i)|$$

where $match(q_a)$ is the set of pattern matching the rule right-hand side. This metric favors imputation rules that estimate values over more data partitions.

The user asking the query should be informed by the quality of the achieved imputation, and decide to keep the original result if the proposed improvement is not satisfying.

8.3.2 Imputation Strategy

The current imputation strategy uses a statically defined priority order for choosing conflicting imputation rules. This setting restricts the process effectiveness.

We started a formalization of more ambitious strategies. Given a set of imputation rules R, we can consider the following two strategies:

1- Quality-driven strategy: Considering the previously described quality metrics, each result partition is assigned with the most accurate imputation rule. The candidate patterns are ordered following their estimated accuracy.

The current strategy applies rules in descending order, which leads to multiple estimations for the same data item where old values can be overwritten by new values. Another solution is to detect application conflicts before applying the rules and generate for each rule an independent pattern set. Indeed, if the same data partition is covered by two patterns that match two different rules, only the rule with the highest quality should be allowed to repair this partition. For this, we can create a graph describing imputation rules dependencies. Each graph node represents the rule's left hand-side q_m, and each edge represents a conflict between different rules. If two rules repair the same partition, this partition should be discarded from the imputation query with the lowest accuracy score.

2- Cost-driven strategy: Similar to the quality metrics, we can define a cost model for the imputation strategy. Given a set of candidate rules, a possible cost metric quantifies the number of generated imputation queries. We can adapt the strategy in this sense, by minimizing the number of imputation queries and prefer rules with the highest coverage. This requires to resolve the following problem. Consider a set of candidate rules R, and P_m patterns describing partitions to impute. Given the graph of candidate rewritings (p, r) ($p \in P_m$, $r \in R$), with (p matches q_m), *find the smallest subgraph that maximally covers* P_m. In other terms, we need to maximize the following function:

$$Cost(R, P_m) = \alpha \times Coverage(R, P_m) - \beta \times |R|$$

with coverage a function that returns the number of pattern in P_m matching rules left-hand side, and α, β weighting parameters.

A possible enhancement considers both, quality and cost metrics, to create a balanced imputation strategy. In this case, we can associate each candidate strategy with the following score:

$$SS = \gamma \times \sum_{r_i \in R} Quality(r_i, Q, P_a) + \theta \times Cost(R, P_m)$$

where γ and θ are weighting parameters.

8.3.3 Shared Query Result Imputations

Currently, the imputation strategy only can repair the result of an aggregation query at the same granularity. For example, it is not possible to use correct or repaired results obtained at the room level for repairing the aggregated values at the floor level. A challenging research direction is to rewrite aggregate queries to take advantage of previously repaired results at the same or at other granularity levels. An interesting first direction would be to study the "query rewriting using views" approach proposed in [Coh+00] for aggregate queries.

Résumé en Français

A

"La ressource la plus précieuse au monde n'est plus le pétrole, mais les données" [Eco]. Cette observation démontre pleinement l'importance des données dans nos sociétés actuelles. La prolifération des réseaux sociaux et des objets connectés a transformé nos habitudes quotidiennes, et révolutionné nos industries. Les usines du futur bannissent de plus en plus les processus de production traditionnels en faveur des flux dynamiques basés sur les données. Des millions d'utilisateurs commandent de la nourriture sur Internet, font leurs courses sur Amazon, interrogent Google pour trouver un restaurant italien près de chez eux et échangent des messages instantanés sur Facebook. Les objets deviennent intelligents, les maisons règlent automatiquement leur consommation énergétique, les voitures deviennent autonomes, et bientôt les machines établiront un diagnostic médical sans intervention humaine. Les avions modernes, comme l'A380, sont équipés de 25 000 capteurs et génèrent près de 2,5 téraoctets de données par jour [Man] pour assurer la maintenance de l'avion.

Cette révolution des données s'appuie sur les avancées technologiques, les nouveaux algorithmes d'apprentissage et les capacités de stockage de données qui permettent la création de nouveaux services produisant et consommant d'énormes quantités de données. Un défi majeur dans ce contexte est de maximiser la qualité des données. Par exemple, IBM indique une perte de 3,1 milliards de dollars par an aux États-Unis [Har], qui peut être principalement attribuée à des données inexactes, obsolètes ou incomplètes, ne répondant pas aux exigences des tâches à accomplir. Le rapport affirme que la qualité des données est l'un des obstacles les plus importants au développement d'une entreprise, se classant devant les outils matériels ou l'expertise humaine.

En dépit de l'abondance des données produites, les données manquantes constituent un problème de qualité fréquent [Her+07], qui trouve son origine dans plusieurs causes : des anomalies physiques, une mauvaise conception de la base de données, des erreurs humaines, un manque de sources ou la nature des règles de confidentialité appliquées. La problématique des données incomplètes génère plusieurs défis de recherche intéressants portant sur la représentation et le traitement des informations manquantes. Alors que de nombreux modèles de données ont été développés pour représenter tout type de données complexes, la prise en compte des données manquantes dans ces modèles demeure compliquée. Une première solution consiste à introduire des symboles réservés pour indiquer des données manquantes nécessitant d'être renseignée. Ce type de symbole a été introduit dans le modèle de données relationnel par E.F. Codd [Cod70]

sous la forme d'un symbole d'"information manquante" $null$. Les valeurs $null$ représentent des valeurs d'attributs manquantes ou inconnues et restent la représentation la plus fréquemment utilisée pour les informations manquantes dans les bases de données. Un inconvénient majeur de cette modélisation est la difficulté de convenir d'une signification unique du symbole $null$ et de sa sémantique dans les langages de requêtes. Par exemple, des conditions de filtrage simples telles que $A = 3$ ne peuvent pas être évaluées comme vraies ou fausses si A accepte les valeurs $null$. Un autre problème concerne les fonctions d'agrégation, qui peuvent produire des résultats incorrects avec la présence des valeurs $null$. Ces limitations ont conduit à la mise au point de systèmes de représentation plus puissants permettant de décrire plus précisément les données manquantes et de mieux comprendre leur influence sur les résultats de la requête. Par exemple, les c-tables [Imi+88a] utilisent des valeurs $null$ "marquées" qui peuvent être partagées par les différents n-uplets en indiquant que c'est la même valeur manquante.

La modélisation de la complétude des données avec des valeurs $null$ ne couvre pas les n-uplets manquantes dans une base de données, qui sont considérés comme faux (hypothèse du monde fermé). Un modèle plus flexible pour représenter l'information manquante a été introduite pour la première fois dans [Mot89], offrant une base théorique plus élargie pour la représentation des n-uplets manquants. Le modèle proposé suppose l'existence d'une base de données virtuelle avec un ensemble complet de n-uplets, pouvant être comparée à la base de données incomplète disponible. Dans ce contexte, connu sous le nom d'hypothèse du monde partiellement fermé, un grand nombre de systèmes de représentation ont été proposés pour modéliser l'incomplétude des données. Le concept de complétude relative [Fan+10a] exploite l'existence de données de référence pour l'analyse de la complétude d'une base de données. La complétude relative est définie par rapport à une base de données de référence matérialisée qui permet d'accomplir une évaluation plus efficace et plus précise [Fan15].

Dans cette thèse, nous adoptons l'approche de la complétude relative pour relever plusieurs défis concernant la représentation d'informations incomplètes pour l'annotation et la réparation de réponses des requêtes. Ce manuscrit présente les contributions en trois parties :

1. La Partie 1 présente les contributions pour la représentation de la complétude de l'information relative. Elle comporte un chapitre d'état de l'art, ainsi qu'un positionnement relatif à notre problématique. Un chapitre est dédié à la formalisation du modèle de patterns (motifs) qui permet une représentation compacte des partitions complètes ou manquantes dans une table de données comparée à une table de référence. Une algèbre de patterns est définie pour dériver les informations de complétude des résultats des requêtes directement à partir des descriptions associées au tables de données. Toutes les définitions et les résultats théoriques sont détaillés et illustrées par des exemples. Le dernier chapitre de cette partie est consacrée à l'implantation de l'algèbre de patterns. Deux algorithmes de calcul de couverture minimale

de patterns sont définis, qui opèrent respectivement sur des tables de données et des tables de patterns. Le chapitre résume également une série d'expérimentations qui ont été réalisées pour vérifier la validité et l'efficacité des contributions.

2. La Partie 2 présente les contributions portant sur une extension du modèle de patterns pour assurer réparation des résultats de requêtes en utilisant des règles d'imputation. Un premier chapitre est dédié à l'état de l'art sur les techniques d'imputation et le deuxième chapitre présente le modèle des règles d'imputations. Le modèle d'imputation proposé se focalise sur la réparation de requêtés d'agrégation, et nous présentons un processus en 4 étapes, ainsi qu'une partie expérimentale qui démontre la faisabilité de l'approche sur des données réelles.

3. La Partie 3 présente une généralisation du modèle de patterns pour la caractérisation de fragments de données. La caractérisation des fragments consiste à produire des résumés compactes pour analyser et comparer des fragments de données. L'objectif est la production de résumés exhaustifs minimaux à travers les couvertures de patterns, pour permettre de réaliser de l'inférences sur l'appartenance de données à diverses catégories. Un module de raisonnement basées sur les requêtes SQL est défini ainsi qu'une série d'expérimentations sur des données réelles.

Représentation de la complétude de l'information relative

La particularité des modèles de complétude réside dans le fait que tout processus d'évaluation dépend de l'existence d'une définition de la sémantique de données complètes et manquantes. L'identification de l'étendue des n-uplets manquants nécessite l'existence de données de référence, et l'exhaustivité relative de l'information est une configuration particulière de "l'hypothèse du monde partiellement fermé", qui offre des garanties supplémentaires. En effet, la base de données de référence (supposée idéale) n'est pas virtuelle mais peut être accessible partiellement ou entièrement.

Vu que nous développons notre modèle sous cette hypothèse, nous proposons en premier lieu un modèle de données qui capture l'existence de tables référence en plus des tables de données classiques. Sous cette configuration, nous étudions la complétude de tables contraintes, constituées de couples de tables données/références. Sur cette base, nous définissons un système de représentation de complétude pour fournir une modélisation compacte des partitions complètes et manquantes dans une table de données. Une partition de données peut être n'importe quel fragment de table obtenu par une requête de sélection. Un pattern est un tuple relationnel classique à l'exception de la présence d'un symbole particulier $*$, qui s'ajoute au domaines de tous les attributs. Ce symbole universel $*$ représente l'ensemble de valeurs possibles qu'un attribut peut prendre, en fonction des valeurs des attributs restants, suivant leur présence dans la table de référence. Un

ensemble de propriétés syntaxiques et sémantiques des patterns permet de définir le concept de la couverture minimale de table de données. Une table de données contrainte possède une unique couverture minimale et nous avons montré que la taille de la couverture minimale est bornée par la taille de la table de données.

L'algèbre relationnelle standard ne suffit pas pour interroger des tables de patterns, en raison de la sémantique particulière du symbole $*$. C'est pourquoi, nous définissions une extension de l'algèbre relationnelle qui comporte deux nouveaux opérateurs : un opérateur de pliage $\triangleright$ qui permet de produire une table de patterns à partir de table de données classique (en créant des $*$ pour représenter de façons compacte des partitions), et un opérateur de dépliage $\triangleleft$ qui défait les symboles $*$ pour retrouver leurs instances. Cette algèbre relationnelle étendue forme un noyau pour la définition d'opérateurs spécifiques aux tables de patterns, permettant d'exprimer des requêtes directement applicables à ces représentations. La tâche associée aux requêtes pattern, est la dérivation directe des couvertures minimales pour la partitions complètes et manquantes dans les résultats de requête sur les données. L'algèbre de patterns peut être exprimée en algèbre relationnelle, à l'exception de l'opérateur de pliage qui implique un parcours récursif de la table de données. Hormis cet opérateur, toute expression basée sur le fragment d'algèbre de patterns $R.A.+\triangleleft$ (sans $\triangleright$) peut être traduite en SQL. Cependant, cette traduction ne garantit pas la minimalité du résultat en l'absence de l'opérateur de pliage. Une contribution importante de cette partie est la définition d'un algorithme efficace pour l'opérateur de pliage sur des tables de données contraintes. Une variante de cet algorithme est aussi proposée pour s'appliquer aux tables de patterns, dans le but d'éliminer les redondances sémantiques et syntaxiques pour garantir la minimalité.

Nous évaluons notre implantation sur un jeu de données collecté par un réseau de capteurs installé dans notre campus Jussieu. Les expériences ont permis de mesurer l'efficacité de l'algorithme de calcul de couverture minimale et la fiabilité des requêtes de patterns pour la dérivation automatiques des couvertures des résultats de requêtes de données. Nous avons également testé l'impact de plusieurs propriétés des tables de données sur la compacité des tables de patterns générées pour des jeux de données réelles et simulées. Nous avons observé que la distribution des données manquantes était le facteur le plus important influant sur la taille des tables de patterns. Une distribution aléatoire dans les ensembles de données synthétiques provoque une explosion du nombre de patterns ($|P| = |D|$ étant la limite supérieure). En revanche, dans les jeux de données réels, où les données manquantes sont causées par des anomalies physiques, qui ne se produisent généralement pas de manière aléatoire, la compacité reste élevée. Nous avons considéré un ensemble de requêtes de données diverses (sélection, agrégation, jointures) pour évaluer l'efficacité des requêtes de pattern. Les résultats obtenus montrent l'efficacité de l'utilisation des requêtes de modèles comparés à une approche naïve.

Imputation des résultats des requêtes d'agrégation

Les données incomplètes donnent naturellement lieu à des résultats de requêtes de mauvaise qualité, et la réparation des données manquantes est une tâche de nettoyage courante. L'imputation des données désigne une famille d'approches visant à réparer les données manquantes en déduisant de nouvelles valeurs à partir des jeux de données disponibles, parfois avec une intervention humaine. Il existe plusieurs méthodes d'imputation des données et les techniques d'imputation sont généralement complexes et ne peuvent pas forcément s'étendre à de grands ensembles de données. En particulier, l'imputation globale des données peut devenir inefficace pour réparer des tables destinées à des tâches spécifiques portant sur un sous ensemble.

Le problème des données manquantes a d'abord été identifié et traité par des statisticiens. Des études sur les distributions de données manquantes ont permis d'effectuer des tâches explicatives et des formulations simples ont été utilisées pour réparer les données incomplètes telles que l'imputation à la valeur moyenne ou la plus fréquente. L'imputation des données comporte deux grandes familles d'approches, celles impliquant une participation humaine dans le processus de réparation des données, et d'autres, dans lesquelles des algorithmes d'inférence sont conçus pour découvrir des corrélations de données et générer des estimations fiables des valeurs manquantes. L'imputation manuelle par l'humain prend deux formes. Les plates-formes de crowdsourcing demandent aux travailleurs de réparer manuellement les données en renseignant les observations manquantes. D'autre part, les techniques basées sur des règles font appel à des experts pour traduire leurs connaissances de domaine en définissant des règles logiques, qui peuvent être utilisées pour la réparation automatique et lors de l'exécution de mises à jour de bases de données. L'imputation automatique regroupe toutes les techniques d'apprentissage automatique et statistiques de base, dans lesquelles des algorithmes tentent d'expliquer les distributions de données manquantes et leur corrélation avec les observations disponibles. Les résultats d'apprentissage servent à alimenter le mécanisme d'inférence, à créer de nouvelles valeurs.

La plupart des techniques d'imputation opèrent au niveau des données sans prendre en compte la sémantique de la requête. Cela implique des efforts et des coûts considérables pour la réparation des tables de données, quel que soit leur taux d'utilisation futur. L'imputation basée sur les requêtes consiste à estimer les résultats de la requête susceptibles d'être affectés par une mauvaise qualité des données (données manquantes). Les contributions montrent que concentrer la tâche de réparation sur les données utilisées pour l'évaluation de la requête peut considérablement réduire les efforts de nettoyage.

Nous traitons le problème de la réparation des résultats des requêtes d'agrégation obtenues à partir de données incomplètes. Les données manquantes entraînent généralement des résultats

de requête manquants, mais dans le cas des requêtes d'agrégation, elles créent également des agrégations incorrectes.

La première contribution de cette thèse, portait sur un modèle de patterns pour identifier et représenter les étendues de données complètes et manquantes et une algèbre permettant de déduire des annotations pour les résultats de la requête. Notre exploitons l'opportunité d'étendre ce modèle pour effectuer l'imputation basée sur des règles, suite à l'identification des partitions manquantes ou incorrectes dans le résultat. Nous définissons une stratégie d'imputation globale qui permet de réparer les résultats de requêtes moyennant des règles d'imputations à différents niveaux de granularités.

Les règles d'imputation traduisent les connaissances des experts à différents niveaux de partitions, correspondant à des niveaux d'agrégation variables. Les règles sémantiques supportent d'agréger différents résultats d'imputations, grâce à une fonction d'imputation. Nous avons formalisé une stratégie complète qui commence par l'évaluation de la complétude du résultat d'une requête. La stratégie comprend quatre étapes : classification des partitions {correcte, manquante, incorrecte }, choix de règles d'imputations candidates, établissement de la stratégie, et enfin la génération de requêtes d'imputation SQL.

La mise en œuvre du processus d'imputation est assurée par l'algèbre des patterns. Dans la première étape, trois requêtes du modèle permettent de classer les résultats de la requête en partitions correctes, incorrectes ou manquantes. Les partitions à réparer sont celles manquantes et incorrectes, car ces dernières sont supposées de moindre qualité que le résultat d'une imputation. Sur cette base, les règles d'imputation sont filtrées pour garder celles qui correspondent au niveau et éventuellement aux filtres exprimés dans les partitions à réparer. Les règles d'imputation candidates sont triés suivant la stratégie d'imputation avant d'être traduites en requêtes SQL. Nous avons proposé une version optimisée pour l'application de requêtes d'imputation à des ensembles de patterns au lieu d'utiliser des tables de données. Cela évite les requêtes d'agrégation sur des ensembles de données volumineux (partitions correctes utilisées pour l'imputation).

Nous avons mené une série d'expériences sur des ensembles de données réelles et tiré parti de notre connaissance du campus et de l'historique de la consommation dans ses localités pour proposer des règles d'imputation pour les mesures de température. Les résultats expérimentaux montrent que le processus d'imputation a réussi à réparer la plupart des résultats des requêtes, dans des délais raisonnables, par rapport au temps d'exécution des requêtes. Notre analyse nous a permis de conclure que la richesse des règles d'imputation et leur adéquation avec les caractéristiques des données constituent une condition essentielle à la réussite de l'imputation.

Les règles d'imputation multiple sont appliquées au même résultat de requête. Dans ce travail, nous avons choisi d'attribuer le résultat de la règle d'imputation avec la priorité la plus élevée

pour réparer les partitions correspondantes. Une alternative intéressante pourrait envisager le mécanisme d'imputation multiple, qui fusionne de nombreux résultats d'imputation, en utilisant la valeur moyenne, ou d'autres formules pondérées. Cette option pourrait offrir une meilleure qualité d'imputation mais impliquerait des coûts supplémentaires.

Récapitulation des fragments de données

La synthèse de données est une approche de recherche de base de données visant à produire des représentations compactes pour les tables de données, afin d'optimiser les évaluations de requêtes ou d'effectuer des tâches d'analyse.

Nous explorons l'opportunité de généraliser notre modèle de patterns pour obtenir des résumés divers au-delà des représentations des informations complètes ou manquantes. Dans le modèle de patterns d'origine, deux fragments de données sont considérés, construits suivant les valeurs d'un seul attribut. Les partitions de données ayant une valeur nulle pour cet attribut appartiennent au fragment de partitions manquant, alors que les partitions avec des valeurs constantes constituent le fragment de partitions complet. Nous étendons ce modèle pour couvrir davantage de types de fragments générés en filtrant les données par de plus riches conditions de sélection. La synthèse vise ensuite à fournir des fragments avec des descriptions exhaustives pour réaliser des tâches d'analyse. Contrairement aux contributions de synthèse classiques, nous ne considérons pas la compacité comme une ligne directrice, mais nous nous efforçons de produire des résumés complets. L'avantage d'avoir accès à des résumés complets est la possibilité d'interroger des ensembles de synthèse, à des fins d'analyse, mais aussi d'exploiter des règles de raisonnement pour les tâches d'explication et de prédiction.

Le processus de généralisation a conduit à considérer les tables de données qui n'incluent pas de clé primaire, impliquant la notion particulière de fragment indiscernable. Cette configuration a nécessité l'adaptation du modèle de patterns pour proposer des résumés respectant les propriétés de couverture minimales. Ces ensembles sont par définition des ensembles de patterns réduits, couvrant entièrement le fragment de données, sans redondance et sont disjoints.

Un module de raisonnement formel permet de déduire des connaissances sur des patterns ou des données qui n'appartiennent pas aux résumés de fragments. Grâce aux propriétés syntaxiques et sémantiques du pattern (spécialisation, généralisation, satisfaction), le modèle garantit l'exhaustivité du processus d'inférence ; nous pouvons décider, pour tout tuple de pattern, s'il appartient à un fragment particulier ou à l'ensemble de motifs non distinguables.

Pour notre expérience, nous utilisons l'ensemble de données public pour adultes afin d'évaluer notre modèle de résumés de fragments. Nous avons exécuté le calcul des résumés de fragments

dans trois jeux de données avec des schémas variables, afin de vérifier la variabilité des résumés en fonction du nombre d'attributs. Nous avons également redimensionné les catégories de données en utilisant un jeu de données avec l'attribut age dans un domaine catégorique, au lieu d'un nombre, pour montrer l'influence de la longueur du domaine sur le fragment non discernable. En variant le nombre d'attributs, on a montré que 1 - le fragment non distinguable est plus significatif avec moins d'attributs, car moins d'éléments permettent de distinguer la classe de revenu, 2 - plus nous considérons d'attributs, moins un résumé est compact, car la redondance des tuples diminue. Les expériences ont permis de vérifier l'efficacité de notre modèle, compte tenu de la sémantique des couvertures générées correspondant à nos observations, ainsi que des résultats de contributions similaires sur cet ensemble de données.

Bibliography

[Abi+87] Serge Abiteboul, Paris C. Kanellakis, and Gösta Grahne. „On the Representation and Querying of Sets of Possible Worlds“. In: *Proceedings of the Association for Computing Machinery Special Interest Group on Management of Data 1987 Annual Conference, San Francisco, CA, USA, May 27-29, 1987*. 1987, pp. 34–48 (cit. on p. 27).

[Abi+95] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995 (cit. on pp. 24, 142).

[Ach+99] Swarup Acharya, Phillip B Gibbons, Viswanath Poosala, and Sridhar Ramaswamy. „The Aqua approximate query answering system“. In: *ACM Sigmod Record*. Vol. 28. 2. ACM. 1999, pp. 574–576 (cit. on p. 90).

[Afi+66] A. A. Afifi and R. M. Elashoff. „Missing Observations in Multivariate Statistics I. Review of the Literature“. In: *Journal of the American Statistical Association* 61.315 (1966), pp. 595–604. eprint: `https://doi.org/10.1080/01621459.1966.10480891` (cit. on p. 24).

[All00] Paul D Allison. „Multiple imputation for missing data: A cautionary tale“. In: *Sociological methods & research* 28.3 (2000), pp. 301–309 (cit. on p. 90).

[Bab+03] Brian Babcock, Surajit Chaudhuri, and Gautam Das. „Dynamic sample selection for approximate query processing“. In: *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*. ACM. 2003, pp. 539–550 (cit. on p. 96).

[Bar+05] José Barateiro and Helena Galhardas. „A Survey of Data Quality Tools“. In: *Datenbank-Spektrum* 14 (2005), pp. 15–21 (cit. on pp. 19, 20).

[Bat+09] Carlo Batini, Cinzia Cappiello, Chiara Francalanci, and Andrea Maurino. „Methodologies for data quality assessment and improvement“. en. In: *ACM Computing Surveys* 41.3 (July 2009), pp. 1–52 (cit. on pp. 22–24).

[Bat+16] Carlo Batini and Monica Scannapieco. *Data and Information Quality - Dimensions, Principles and Techniques*. Data-Centric Systems and Applications. Springer, 2016 (cit. on p. 20).

[Boh+05] Philip Bohannon, Wenfei Fan, Michael Flaster, and Rajeev Rastogi. „A cost-based model and effective heuristic for repairing constraints by value modification“. In: *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*. ACM. 2005, pp. 143–154 (cit. on p. 92).

[Bov+03] Matthew Bovee, Rajendra P. Srivastava, and Brenda Mak. „A conceptual framework and belief-function approach to assessing overall information quality“. In: *Int. J. Intell. Syst.* 18.1 (2003), pp. 51–74 (cit. on p. 22).

[Bra08] Daren C Brabham. „Crowdsourcing as a model for problem solving: An introduction and cases“. In: *Convergence* 14.1 (2008), pp. 75–90 (cit. on p. 91).

[Buc+03] Francesco Buccafurri, Filippo Furfaro, Domenico Sacca, and Cristina Sirangelo. „A Quad-tree Based Multiresolution Approach for Two-dimensional Summary Data“. In: *Proceedings of the 15th International Conference on Scientific and Statistical Database Management*. SSDBM '03. Washington, DC, USA: IEEE Computer Society, 2003, pp. 127–140 (cit. on p. 136).

[Buc60] Samuel F Buck. „A method of estimation of missing values in multivariate data suitable for use with an electronic computer“. In: *Journal of the Royal Statistical Society. Series B (Methodological)* (1960), pp. 302–306 (cit. on p. 94).

[Buh+11] Michael Buhrmester, Tracy Kwang, and Samuel D Gosling. „Amazon's Mechanical Turk: A new source of inexpensive, yet high-quality, data?“ In: *Perspectives on psychological science* 6.1 (2011), pp. 3–5 (cit. on p. 91).

[Bur+10] Lane F Burgette and Jerome P Reiter. „Multiple imputation for missing data via sequential regression trees“. In: *American journal of epidemiology* 172.9 (2010), pp. 1070–1076 (cit. on p. 95).

[Cam+17] José Cambronero, John K Feser, Micah J Smith, and Samuel Madden. „Query optimization for dynamic imputation“. In: *Proceedings of the VLDB Endowment* 10.11 (2017), pp. 1310–1321 (cit. on pp. 97, 98).

[Cao+14] Yang Cao, Ting Deng, Wenfei Fan, and Floris Geerts. „On the data complexity of relative information completeness“. In: *Inf. Syst.* 45 (2014), pp. 18–34 (cit. on p. 37).

[Car+83] Jaime G Carbonell, Ryszard S Michalski, and Tom M Mitchell. „An overview of machine learning“. In: *Machine Learning, Volume I*. Elsevier, 1983, pp. 3–23 (cit. on p. 94).

[Cha+01] Kaushik Chakrabarti, Minos Garofalakis, Rajeev Rastogi, and Kyuseok Shim. „Approximate query processing using wavelets“. In: *The VLDB Journal—The International Journal on Very Large Data Bases* 10.2-3 (2001), pp. 199–223 (cit. on p. 96).

[Che+13] Jieying Chen, Jia-Yu Pan, Christos Faloutsos, and Spiros Papadimitriou. „TSum: fast, principled table summarization“. en. In: *Proceedings of the Seventh International Workshop on Data Mining for Online Advertising - ADKDD '13*. Chicago, Illinois: ACM Press, 2013, pp. 1–9 (cit. on p. 137).

[Chi+16] Anand Inasu Chittilappilly, Lei Chen, and Sihem Amer-Yahia. „A survey of general-purpose crowdsourcing techniques“. In: *IEEE Transactions on Knowledge and Data Engineering* 28.9 (2016), pp. 2246–2266 (cit. on p. 91).

[Chu+18] Yeounoh Chung, Michael Lind Mortensen, Carsten Binnig, and Tim Kraska. „Estimating the impact of unknown unknowns on aggregate query results“. In: *ACM Transactions on Database Systems (TODS)* 43.1 (2018), p. 3 (cit. on p. 97).

[Cod70] E. F. Codd. „A Relational Model of Data for Large Shared Data Banks“. In: *Commun. ACM* 13.6 (1970), pp. 377–387 (cit. on pp. 27, 149).

[Cod79] E. F. Codd. „Extending the Database Relational Model to Capture More Meaning“. In: *ACM Trans. Database Syst.* 4.4 (1979), pp. 397–434 (cit. on pp. 2, 27).

[Coh+00] Sara Cohen, Werner Nutt, and Alexander Serebrenik. „Algorithms for rewriting aggregate queries using views“. In: *Current Issues in Databases and Information Systems*. Springer, 2000, pp. 65–78 (cit. on p. 148).

[Con+10] Tyson Condie, Neil Conway, Peter Alvaro, et al. „Online aggregation and continuous query support in mapreduce“. In: *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. ACM. 2010, pp. 1115–1118 (cit. on p. 90).

[Cuz+08] Alfredo Cuzzocrea and Domenico Saccà. „H-IQTS: A Semantics-aware Histogram for Compressing Categorical OLAP Data“. In: *Proceedings of the 2008 International Symposium on Database Engineering & Applications*. IDEAS '08. New York, NY, USA: ACM, 2008, pp. 209–217 (cit. on p. 136).

[Dal+13] Michele Dallachiesa, Amr Ebaid, Ahmed Eldawy, et al. „NADEEF: a commodity data cleaning system“. In: *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013, New York, NY, USA, June 22-27, 2013*. 2013, pp. 541–552 (cit. on pp. 93, 104).

[Dem+77] Arthur P Dempster, Nan M Laird, and Donald B Rubin. „Maximum likelihood from incomplete data via the EM algorithm“. In: *Journal of the royal statistical society. Series B (methodological)* (1977), pp. 1–38 (cit. on p. 94).

[Dem82] W Edwards Deming. „Quality, productivity, and competetive position“. In: *MIT Center for Advanced Engineering, Cambridge* (1982) (cit. on p. 18).

[Den+16] Ting Deng, Wenfei Fan, and Floris Geerts. „Capturing Missing Tuples and Missing Values“. In: *ACM Trans. Database Syst.* 41.2 (2016), 10:1–10:47 (cit. on pp. 29, 36, 37).

[Dix79] John K Dixon. „Pattern recognition with partly missing data“. In: *IEEE Transactions on Systems, Man, and Cybernetics* 9.10 (1979), pp. 617–621 (cit. on p. 95).

[Don+06] A Rogier T Donders, Geert JMG Van Der Heijden, Theo Stijnen, and Karel GM Moons. „A gentle introduction to imputation of missing values“. In: *Journal of clinical epidemiology* 59.10 (2006), pp. 1087–1091 (cit. on p. 94).

[Dua+07] Songyun Duan and Shivanath Babu. „Processing forecasting queries“. In: *Proceedings of the 33rd international conference on Very large data bases*. VLDB Endowment. 2007, pp. 711–722 (cit. on p. 95).

[Elk90] Charles Elkan. „Independence of Logic Database Queries and Updates“. In: *Proceedings of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, April 2-4, 1990, Nashville, Tennessee, USA*. 1990, pp. 154–160 (cit. on p. 32).

[Fan+08] Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. „Conditional functional dependencies for capturing data inconsistencies“. In: *ACM Transactions on Database Systems (TODS)* 33.2 (2008), p. 6 (cit. on p. 92).

[Fan+10a] Wenfei Fan and Floris Geerts. „Relative Information Completeness“. In: *ACM Trans. Database Syst.* 35.4 (Oct. 2010), 27:1–27:44 (cit. on pp. 3, 29, 36, 37, 39, 63, 150).

[Fan+10b] Wenfei Fan, Jianzhong Li, Shuai Ma, Nan Tang, and Wenyuan Yu. „Towards certain fixes with editing rules and master data“. In: *Proceedings of the VLDB Endowment* 3.1-2 (2010), pp. 173–184 (cit. on pp. 29, 90, 93).

[Fan+12] Wenfei Fan and Floris Geerts. *Foundations of Data Quality Management*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2012 (cit. on pp. 18, 36).

[Fan15] Wenfei Fan. „Data quality: From theory to practice“. In: *Acm Sigmod Record* 44.3 (2015), pp. 7–18 (cit. on pp. 3, 37, 150).

[Fel+76] Ivan P Fellegi and David Holt. „A systematic approach to automatic edit and imputation“. In: *Journal of the American Statistical association* 71.353 (1976), pp. 17–35 (cit. on p. 90).

[Gar+01] Minos N Garofalakis and Phillip B Gibbons. „Approximate Query Processing: Taming the TeraBytes.“ In: *VLDB*. 2001, pp. 343–352 (cit. on p. 96).

[Gar88] David A Garvin. *Managing quality: The strategic and competitive edge*. Simon and Schuster, 1988 (cit. on p. 18).

[Ge+07] Mouzhi Ge and Markus Helfert. „A Review of Information Quality Research - Develop a Research Agenda“. In: *Proceedings of the 12th International Conference on Information Quality, MIT, Cambridge, MA, USA, November 9-11, 2007*. 2007, pp. 76–91 (cit. on p. 21).

[Gel+06] Andrew Gelman and Jennifer Hill. *Data analysis using regression and multilevel/hierarchical models*. Cambridge university press, 2006 (cit. on p. 89).

[Goa+07] Virginie Goasdoué, Sylvaine Nugier, Dominique Duquennoy, and Brigitte Laboisse. „An Evaluation Framework For Data Quality Tools“. In: *Proceedings of the 12th International Conference on Information Quality, MIT, Cambridge, MA, USA, November 9-11, 2007*. 2007, pp. 280–294 (cit. on p. 24).

[Gow71] John C Gower. „A general coefficient of similarity and some of its properties“. In: *Biometrics* (1971), pp. 857–871 (cit. on p. 94).

[Gra77] John Grant. „Null Values in a Relational Data Base“. In: *Inf. Process. Lett.* 6.5 (1977), pp. 156–157 (cit. on p. 24).

[Gsc+12] Theresia Gschwandtner, Johannes Gärtner, Wolfgang Aigner, and Silvia Miksch. „A taxonomy of dirty time-oriented data“. In: *International Conference on Availability, Reliability, and Security*. Springer, 2012, pp. 58–72 (cit. on pp. 19, 89).

[Han+19a] Fatma-Zohra Hannou, Bernd Amann, and Mohamed-Amine Baazizi. „Explaining Query Answer Completeness and Correctness With Partition Patterns“. In: *Database and Expert Systems Applications - 30th International Conference, DEXA 2019, Linz, Austria, August 26-29, 2019, Proceedings, Part I*. 2019 (cit. on p. 13).

[Han+19b] Fatma-Zohra Hannou, Bernd Amann, and Mohamed-Amine Baazizi, eds. *Exploring and Comparing Table Fragments With Fragment Summaries*. 2019 (cit. on p. 13).

[Han+19c] Fatma-Zohra Hannou, Bernd Amann, and Mohamed-Amine Baazizi. „Query-Oriented answer imputation for aggregate queries“. In: *Advances in Databases and Information Systems - 23rd European Conference, ADBIS 2019, Bled, Slovenia, September 8-11, 2019, Proceedings*. 2019 (cit. on p. 13).

[Hel+00] Joseph M. Hellerstein, Michael J. Franklin, Sirish Chandrasekaran, et al. „Adaptive query processing: Technology in evolution“. In: *IEEE Data Eng. Bull.* 23.2 (2000), pp. 7–18 (cit. on p. 96).

[Hel08] Joseph M Hellerstein. „Quantitative data cleaning for large databases“. In: *United Nations Economic Commission for Europe (UNECE)* (2008) (cit. on p. 89).

[Her+07] Thomas N Herzog, Fritz J Scheuren, and William E Winkler. *Data quality and record linkage techniques*. Springer Science & Business Media, 2007 (cit. on pp. 2, 149).

[Hua+98] Kuan-Tse Huang, Yang W Lee, and Richard Y Wang. „Quality information and knowledge“. In: (1998) (cit. on p. 18).

[Imi+84] Tomasz Imielinski and Witold Lipski Jr. „Incomplete Information in Relational Databases“. In: *J. ACM* 31.4 (1984), pp. 761–791 (cit. on pp. 27, 28, 35).

[Imi+88a] Tomasz Imieliński and Witold Lipski. „Incomplete information in relational databases“. In: *Readings in Artificial Intelligence and Databases*. Elsevier, 1988, pp. 342–360 (cit. on pp. 2, 150).

[Imi+88b] Tomasz Imieliński and Witold Lipski. „Incomplete information in relational databases“. In: *Readings in Artificial Intelligence and Databases*. Elsevier, 1988, pp. 342–360 (cit. on p. 26).

[Imm+15] Anne Immonen, Pekka Pääkkönen, and Eila Ovaska. „Evaluating the Quality of Social Media Data in Big Data Architecture“. In: *IEEE Access* 3 (2015), pp. 2028–2043 (cit. on p. 20).

[Jef+08] Shawn R Jeffery, Michael J Franklin, and Alon Y Halevy. „Pay-as-you-go user feedback for dataspace systems“. In: *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. ACM. 2008, pp. 847–860 (cit. on p. 90).

[Jer+10] José M Jerez, Ignacio Molina, Pedro J García-Laencina, et al. „Missing data imputation using statistical and machine learning methods in a real breast cancer problem“. In: *Artificial intelligence in medicine* 50.2 (2010), pp. 105–115 (cit. on p. 94).

[Jun+04] Heikki Junninen, Harri Niska, Kari Tuppurainen, Juhani Ruuskanen, and Mikko Kolehmainen. „Methods for imputation of missing values in air quality data sets“. In: *Atmospheric Environment* 38.18 (2004), pp. 2895–2907 (cit. on p. 94).

[Jur03] Joseph M Juran. *Juran on leadership for quality*. Simon and Schuster, 2003 (cit. on p. 18).

[Kie16] Cornelia Kiefer. „Assessing the Quality of Unstructured Data: An Initial Overview.“ In: *Proceedings of the LWDA 2016 Proceedings (LWDA), CEUR Workshop Proceedings*. 2016 (cit. on p. 20).

[Koh97] Teuvo Kohonen. „Exploration of very large databases by self-organizing maps“. In: *Proceedings of International Conference on Neural Networks (ICNN'97)*. Vol. 1. IEEE. 1997, PL1–PL6 (cit. on p. 95).

[Koo+09] Arne Koopman and Arno Siebes. „Characteristic Relational Patterns“. In: *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '09. New York, NY, USA: ACM, 2009, pp. 437–446 (cit. on p. 137).

[Lak+02] Laks VS Lakshmanan, Jian Pei, and Jiawei Han. „Quotient cube: How to summarize the semantics of a data cube“. In: *Proceedings of the 28th international conference on Very Large Data Bases*. VLDB Endowment, 2002, pp. 778–789 (cit. on p. 136).

[Lak+99] Kamakshi Lakshminarayan, Steven A Harp, and Tariq Samad. „Imputation of missing data in industrial databases“. In: *Applied intelligence* 11.3 (1999), pp. 259–275 (cit. on p. 89).

[Lan+14] Willis Lang, Rimma V. Nehme, Eric Robinson, and Jeffrey F. Naughton. „Partial results in database systems“. en. In: *Proceedings of the 2014 ACM SIGMOD international conference on Management of data - SIGMOD '14*. Snowbird, Utah, USA: ACM Press, 2014, pp. 1275–1286 (cit. on pp. 33, 39, 63).

[Lan+15] Willis Lang, Rimma V. Nehme, and Ian Rae. „Database Optimization in the Cloud: Where Costs, Partial Results, and Consumer Choice Meet“. In: *CIDR 2015, Seventh Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 4-7, 2015, Online Proceedings*. 2015 (cit. on p. 35).

[Lan+97] Lawrence R Landerman, Kenneth C Land, and Carl F Pieper. „An empirical evaluation of the predictive mean matching method for imputing missing values“. In: *Sociological Methods & Research* 26.1 (1997), pp. 3–33 (cit. on p. 90).

[Lee+14] Matthijs van Leeuwen and Jilles Vreeken. „Mining and Using Sets of Patterns through Compression“. en. In: *Frequent Pattern Mining*. Ed. by Charu C. Aggarwal and Jiawei Han. Cham: Springer International Publishing, 2014, pp. 165–198 (cit. on p. 137).

[Lem04] Domenico Lembo. „Dealing with inconsistency and incompleteness in data integration“. In: (2004) (cit. on p. 26).

[Len+97] H.-J. Lenz and Arie Shoshani. „Summarizability in OLAP and statistical data bases“. In: *Scientific and Statistical Database Management, 1997. Proceedings., Ninth International Conference on*. IEEE, 1997, pp. 132–143 (cit. on p. 136).

[Lev+93] Alon Y Levy and Yehoshua Sagiv. „Queries independent of updates“. In: *VLDB*. Vol. 93. Citeseer. 1993, pp. 171–181 (cit. on p. 145).

[Lev96] Alon Y. Levy. „Obtaining complete answers from incomplete databases“. In: *VLDB*. Vol. 96. Citeseer, 1996, pp. 402–412 (cit. on pp. 29, 32, 33, 38, 39, 63, 145).

[Li+16] Guoliang Li, Jiannan Wang, Yudian Zheng, and Michael J Franklin. „Crowdsourced data management: A survey“. In: *IEEE Transactions on Knowledge and Data Engineering* 28.9 (2016), pp. 2296–2319 (cit. on pp. 90, 91).

[Lit+14] Roderick JA Little and Donald B Rubin. *Statistical analysis with missing data*. Vol. 333. John Wiley & Sons, 2014 (cit. on pp. 90, 94).

[Lit+89] Roderick JA Little and Donald B Rubin. „The analysis of social science data with missing values“. In: *Sociological Methods & Research* 18.2-3 (1989), pp. 292–326 (cit. on pp. 90, 93).

[Lit88] Roderick JA Little. „A test of missing completely at random for multivariate data with missing values“. In: *Journal of the American statistical Association* 83.404 (1988), pp. 1198–1202 (cit. on p. 90).

[Lo+00] Ming-Ling Lo, Kun-Lung Wu, and Philip S Yu. „Tabsum: A flexible and dynamic table summarization approach“. In: *Distributed Computing Systems, 2000. Proceedings. 20th International Conference on*. IEEE. 2000, pp. 628–635 (cit. on p. 137).

[Los10] David Loshin. *Master data management*. Morgan Kaufmann, 2010 (cit. on p. 36).

[Mah+11] A. N. Mahmood, C. Leckie, R. Islam, and Z. Tari. „Hierarchical summarization techniques for network traffic“. In: *2011 6th IEEE Conference on Industrial Electronics and Applications*. June 2011, pp. 2474–2479 (cit. on p. 136).

[Mam+13] Michael Mampaey and Jilles Vreeken. „Summarizing categorical data by clustering attributes“. en. In: *Data Mining and Knowledge Discovery* 26.1 (Jan. 2013), pp. 130–173 (cit. on p. 137).

[May+10] Chris Mayfield, Jennifer Neville, and Sunil Prabhakar. „ERACER: a database approach for statistical inference and data cleaning“. In: *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. ACM. 2010, pp. 75–86 (cit. on p. 95).

[Mey98] Ron van der Meyden. „Logical approaches to incomplete information: A survey“. In: *Logics for databases and information systems*. Springer, 1998, pp. 307–356 (cit. on pp. 27, 28).

[Mot89] Amihai Motro. „Integrity = Validity + Completeness“. In: *ACM Trans. Database Syst.* 14.4 (Dec. 1989), pp. 480–502 (cit. on pp. 3, 27, 29, 30, 32, 33, 38, 39, 63, 150).

[Mü+05] Heiko Müller and Johann-Christph Freytag. *Problems, methods, and challenges in comprehensive data cleansing*. Professoren des Inst. Für Informatik, 2005 (cit. on p. 19).

[Nau03] Felix Naumann. *Quality-driven query answering for integrated information systems*. Vol. 2261. Springer, 2003 (cit. on p. 22).

[Oli+05] Paulo Oliveira, Fátima Rodrigues, and Pedro Rangel Henriques. „A Formal Definition of Data Quality Problems“. In: *Proceedings of the 2005 International Conference on Information Quality (MIT ICIQ Conference), Sponsored by Lockheed Martin, MIT, Cambridge, MA, USA, November 10-12, 2006*. 2005 (cit. on p. 19).

[Olt+16] Dan Olteanu and Maximilian Schleich. „Factorized Databases“. In: *SIGMOD Rec.* 45.2 (Sept. 2016), pp. 5–16 (cit. on p. 137).

[Par+14] Hyunjung Park and Jennifer Widom. „Crowdfill: collecting structured data from the crowd“. In: *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. ACM. 2014, pp. 577–588 (cit. on p. 91).

[Rag+01] Trivellore E Raghunathan, James M Lepkowski, John Van Hoewyk, and Peter Solenberger. „A multivariate technique for multiply imputing missing values using a sequence of regression models“. In: *Survey methodology* 27.1 (2001), pp. 85–96 (cit. on p. 94).

[Rah+00] Erhard Rahm and Hong Hai Do. „Data cleaning: Problems and current approaches“. In: *IEEE Data Eng. Bull.* 23.4 (2000), pp. 3–13 (cit. on pp. 19, 26, 92).

[Ram+02] Vijayshankar Raman and Joseph M Hellerstein. „Partial results for online query processing“. In: *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*. ACM. 2002, pp. 275–286 (cit. on p. 96).

[Ras+02] Guillaume Raschia and Noureddine Mouaddib. „SAINTETIQ: a fuzzy set-based approach to database summarization“. In: *Fuzzy sets and systems* 129.2 (2002), pp. 137–162 (cit. on p. 136).

[Raz+11] Simon Razniewski and Werner Nutt. „Completeness of Queries over Incomplete Databases“. In: *PVLDB* 4.11 (2011), pp. 749–760 (cit. on p. 33).

[Raz+15] Simon Razniewski, Flip Korn, Werner Nutt, and Divesh Srivastava. „Identifying the Extent of Completeness of Query Answers over Partially Complete Databases“. en. In: *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data - SIGMOD '15*. Melbourne, Victoria, Australia: ACM Press, 2015, pp. 561–576 (cit. on pp. 29, 33, 38, 39, 63).

[Raz+16] Simon Razniewski, Ognjen Savkovic, and Werner Nutt. „Turning The Partial-closed World Assumption Upside Down“. In: CEUR-WS, 2016 (cit. on p. 142).

[Red96] Thomas C. Redman. *Data quality for the information age*. Artech House, 1996 (cit. on pp. 22, 23).

[Rei86] Raymond Reiter. „A sound and sometimes complete query evaluation algorithm for relational databases with null values“. In: *J. ACM* 33.2 (1986), pp. 349–370 (cit. on p. 27).

[Roy+04] Patrick Royston et al. „Multiple imputation of missing values“. In: *Stata journal* 4.3 (2004), pp. 227–41 (cit. on p. 90).

[Rub04] Donald B Rubin. *Multiple imputation for nonresponse in surveys*. Vol. 81. John Wiley & Sons, 2004 (cit. on p. 90).

[RUB76] DONALD B. RUBIN. „Inference and missing data“. In: *Biometrika* 63.3 (1976), pp. 581–592 (cit. on pp. 25, 90).

[SC+09] K Selçuk Candan, Huiping Cao, Yan Qi, and Maria Luisa Sapino. „Alphasum: size-constrained table summarization using value lattices“. In: *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*. ACM. 2009, pp. 96–107 (cit. on p. 136).

[SC12] Laura Sebastian-Coleman. *Measuring data quality for ongoing improvement: a data quality assessment framework*. Newnes, 2012 (cit. on p. 18).

[Sch97] Joseph L Schafer. *Analysis of incomplete multivariate data*. Chapman and Hall/CRC, 1997 (cit. on p. 94).

[Son04] Daniel Sonntag. „Assessing the Quality of Natural Language Text Data“. In: *INFORMATIK 2004 - Informatik verbindet, Band 1, Beiträge der 34. Jahrestagung der Gesellschaft für Informatik e.V. (GI), Ulm, 20.-24. September 2004*. 2004, pp. 259–263 (cit. on p. 20).

[SP+05] Régis Saint-Paul, Guillaume Raschia, and Noureddine Mouaddib. „General Purpose Database Summarization“. In: *Proceedings of the 31st International Conference on Very Large Data Bases*. VLDB '05. Trondheim, Norway: VLDB Endowment, 2005, pp. 733–744 (cit. on p. 136).

[SR+11] Esther-Lydia Silva-Ramírez, Rafael Pino-Mejías, Manuel López-Coello, and María-Dolores Cubiles-de-la Vega. „Missing value imputation on missing completely at random data using multilayer perceptrons“. In: *Neural Networks* 24.1 (2011), pp. 121–129 (cit. on p. 94).

[Ssa+08] George Ssali and Tshilidzi Marwala. „Computational intelligence and decision trees for missing data estimation“. In: *Neural Networks, 2008. IJCNN 2008.(IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*. IEEE. 2008, pp. 201–207 (cit. on p. 95).

[Sto+86] Michael Stonebraker and Lawrence A. Rowe. „The Design of POSTGRES“. In: *SIGMOD Rec.* 15.2 (June 1986), pp. 340–355 (cit. on p. 77).

[Sun+17] Bruhathi Sundarmurthy, Paraschos Koutris, Willis Lang, Jeffrey Naughton, and Val Tannen. „m-tables: Representing missing data“. In: *LIPIcs-Leibniz International Proceedings in Informatics*. Vol. 68. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017 (cit. on pp. 29, 35, 38, 39, 63).

[Tal+18] Ikbal Taleb, Mohamed Adel Serhani, and Rachida Dssouli. „Big Data Quality Assessment Model for Unstructured Data“. In: *2018 International Conference on Innovations in Information Technology (IIT)*. IEEE. 2018, pp. 69–74 (cit. on p. 20).

[Tod+15] Ion-George Todoran, Laurent Lecornu, Ali Khenchaf, and Jean-Marc Le Caillec. „A Methodology to Evaluate Important Dimensions of Information Quality in Systems“. In: *J. Data and Information Quality* 6.2-3 (2015), 11:1–11:23 (cit. on p. 20).

[Tru+13] Beth Trushkowsky, Tim Kraska, Michael J Franklin, and Purnamrita Sarkar. „Crowdsourced enumeration queries“. In: *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*. IEEE. 2013, pp. 673–684 (cit. on p. 92).

[VB18] Stef Van Buuren. *Flexible imputation of missing data*. Chapman and Hall/CRC, 2018 (cit. on p. 94).

[Vog+06] W. A. Voglozin, G. Raschia, L. Ughetto, and N. Mouaddib. „Querying a summary of database“. en. In: *Journal of Intelligent Information Systems* 26.1 (Jan. 2006), pp. 59–73 (cit. on p. 136).

[Wan+14] Jiannan Wang, Sanjay Krishnan, Michael J Franklin, et al. „A sample-and-clean framework for fast and accurate query processing on dirty data“. In: *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. ACM. 2014, pp. 469–480 (cit. on pp. 88, 90, 96).

[Wan+17] Hong-Zhi Wang, Zhi-Xin Qi, Ruo-Xi Shi, Jian-Zhong Li, and Hong Gao. „COSSET+: Crowdsourced Missing Value Imputation Optimized by Knowledge Base“. In: *Journal of Computer Science and Technology* 32.5 (2017), pp. 845–857 (cit. on p. 89).

[Wan+96a] Yair Wand and Richard Y. Wang. „Anchoring Data Quality Dimensions in Ontological Foundations“. In: *Commun. ACM* 39.11 (1996), pp. 86–95 (cit. on p. 22).

[Wan+96b] Richard Y. Wang and Diane M. Strong. „Beyond Accuracy: What Data Quality Means to Data Consumers“. In: *J. of Management Information Systems* 12.4 (1996), pp. 5–33 (cit. on pp. 18, 22–24).

[Wij05] Jef Wijsen. „Database repairing using updates“. In: *ACM Transactions on Database Systems (TODS)* 30.3 (2005), pp. 722–768 (cit. on p. 92).

[Woo+14] Philip Woodall, Martin A. Oberhofer, and Alexander Borek. „A classification of data quality assessment and improvement methods“. In: *IJIQ* 3.4 (2014), pp. 298–321 (cit. on p. 19).

[Yak+11] Mohamed Yakout, Ahmed K Elmagarmid, Jennifer Neville, Mourad Ouzzani, and Ihab F Ilyas. „Guided data repair“. In: *Proceedings of the VLDB Endowment* 4.5 (2011), pp. 279–289 (cit. on p. 95).

Sites internet

[Amt] *Amazon Mechanical Turk*. URL: `https://www.mturk.com/` (cit. on p. 91).

[Ebi] *Ebita project website*. Accessed: 2019-04-30. URL: `https://www.iosb.fraunhofer.de/servlet/is/56787/` (cit. on p. 3).

[Eco] *Economist Article*. URL: `https://www.economist.com/leaders/2017/05/06/the-worlds-most-valuable-resource-is-no-longer-oil-but-data` (cit. on pp. 2, 149).

[Fig] *Figure Eight*. URL: `https://www.figure-eight.com/` (cit. on p. 91).

[Gen] *Gengo*. URL: `https://gengo.ai/` (cit. on p. 91).

[Har] *Harvard Business Review article*. Accessed: 2019-05-06. URL: https://hbr.org/2016/09/bad-data-costs-the-u-s-3-trillion-per-year (cit. on pp. 2, 149).

[Man] *Planes Technology Data*. Accessed: 2019-05-05. URL: https://www.aerospacemanufacturinganddesign.com/article/millions-of-data-points-flying-part2-121914/ (cit. on pp. 2, 149).

[Upw] *Upwork*. URL: https://www.upwork.com/ (cit. on p. 91).

List of Figures

List of Tables

Abstract

Information incompleteness is a major data quality issue which is amplified by the increasing amount of data collected from unreliable sources. Assessing the completeness of data is crucial for determining the quality of the data itself, but also for verifying the validity of query answers over incomplete data. While there exists an important amount of work on modeling data completeness, deriving this completeness information has not received much attention. In this work, we tackle the issue of extracting and reasoning about complete and missing information under *relative information completeness* setting. Under this setting, the completeness of a dataset is assessed with respect to a complete reference dataset. Few works have been dedicated to representing data completeness under this setting, and we advance the field by proposing two contributions: a *pattern model* for providing minimal covers summarizing the extent of complete and missing data partitions and a *pattern algebra* for deriving minimal pattern covers for query answers to analyze their validity.

The completeness pattern framework presents an intriguing opportunity to achieve many applications, particularly those aiming at improving the quality of tasks impacted by missing data. In our work, we address the problem of repairing query results obtained from incomplete data. Data imputation is a well-known technique for repairing missing data values but can incur a prohibitive cost when applied to large data sets. Query-driven imputation offers a better alternative as it allows for fixing only the data that is relevant for a query. We adopt a rule-based query rewriting technique for imputing the answers of analytic queries that are missing or suffer from incorrectness due to data incompleteness. We present a novel query rewriting mechanism that is guided by the completeness pattern model and algebra. Our solution strives to infer the broadest possible set of missing answers while improving the precision of incorrect ones.

In the last contribution, we investigate the generalization of our pattern model for summarizing any data fragments. The generalized pattern model can be used to produce pattern summaries of data fragments over any subset of attributes and these summaries can be queried to analyze and compare data fragments in a synthetic and flexible way.

Keywords: Relative Information, Completeness Assessment, Pattern model, Pattern Algebra, Imputation, Summarization