THE UNIVERSITY
of EDINBURGH

Toward relevant answers to queries on incomplete databases

Etienne Toussaint

Doctor of Philosophy

THE UNIVERSITY OF EDINBURGH

2022

To Abigail, Judith, Charlie, Pepe and Lison

Etienne

Abstract

Incomplete and uncertain information is ubiquitous in database management applications. However, the techniques specifically developed to handle incomplete data are not sufficient. Even the evaluation of SQL queries on databases containing NULL values remains a challenge after 40 years. There is no consensus on what an answer to a query on an incomplete database should be, and the existing notions often have limited applicability.

One of the most prevalent techniques in the literature is based on finding answers that are certainly true, independently of how missing values are interpreted. However, this notion has yielded several conflicting formal definitions for certain answers. Based on the fact that incomplete data can be enriched by some additional knowledge, we designed a notion able to unify and explain the different definitions for certain answers. Moreover, the knowledge-preserving certain answers notion is able to provide the first well-founded definition of certain answers for the relational bag data model and value-inventing queries, addressing some key limitations of previous approaches. However, it doesn't provide any guarantee about the relevancy of the answers it captures.

To understand what would be relevant answers to queries on incomplete databases, we designed and conducted a survey on the everyday usage of NULL values among database users. One of the findings from this socio-technical study is that even when users agree on the possible interpretation of NULL values, they may not agree on what a satisfactory query answer is. Therefore, to be relevant, query evaluation on incomplete databases must account for users' tasks and preferences.

We model users' preferences and tasks with the notion of regret. The regret function captures the task-dependent loss a user endures when he considers a database as ground truth instead of another. Thanks to this notion, we designed the first framework able to provide a score accounting for the risk associated with query answers. It allows us to define the risk-minimizing answers to queries on incomplete databases. We show that for some regret functions, regret-minimizing answers coincide with certain answers. Moreover, as the notion is more agile, it can capture more nuanced answers and more interpretations of incompleteness.

A different approach to improve the relevancy of an answer is to explain its provenance. We propose to partition the incompleteness into sources and measure their respective contribution to the risk of answer. As a first milestone, we study several models to predict the evolution of the risk when we clean a source of incompleteness. We implemented the framework, and it exhibits promising results on relational databases

and queries with aggregate and grouping operations. Indeed, the model allows us to infer the risk reduction obtained by cleaning an attribute. Finally, by considering a game theoretical approach, the model can provide an explanation for answers based on the contribution of each attributes to the risk.

Acknowledgements

This Thesis would not have been possible without the academic and emotional support of many people. First, I wish to thank every researcher who patiently listened to the mathematical errand of my mind. I have been lucky to encounter Amélie Gheerbrant, who first introduced me to the Principles of Data Management. Since then, she and many others from the community have blessed me with invaluable feedback (and fun moments).

I am grateful to Leonid Libkin, Paolo Guagliardo, and Juan Sequeda for being actively involved in the research projects of this Thesis. But more importantly, they went far behind their academic duties and provided crucial emotional support.

A special place in my heart belongs to Luke Darlow, Piete Cyrus, and Patricia Rubbish for their ongoing daily support during the Ph.D. Even as a father, Christian Toussaint, went far behind his responsibilities and, despite his lack of expertise, spent countless hours discussing ideas with me.

Finally, my academic life would have stopped before undergrad without Philippe Toussaint. He had no social or academic responsibilities. He simply helped due to his kind heart. For that and much more, he earned my highest esteem and gratitude.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

Etienne Toussaint

Contents

Figures and Tables

Figures

Tables

Chapter 1

Introduction

Mostly known as computer science, the term Informatics seems to be more appropriate to describe the field today. Indeed information is at heart of most IT applications, as a consequence Data management has become one of the most relevant fields in computer science Abiteboul (2013). In April 2016, a community of researchers working in the area of Principles of Data Management (PDM) identified the study of Uncertain Information as one of the most important research directions Abiteboul et al. (2017). Indeed Database applications need to handle incomplete data, this is especially true these days: Many sources of incompleteness arise due to the "big data" phenomenon: for instance, integrating or exchanging large datasets is a common task nowadays, and resulting databases almost invariably have lots of missing data. Likewise, data found in large repositories available on the Web tend to be incomplete and contain many gaps Abiteboul, Buneman, and Suciu (2000). Even if the problem has been acknowledged early on Codd (1975, 1979), the current way to deal with incomplete information leaves much to be desired. It has been estimated that handling dirty and incomplete data costs US businesses alone more than $600B each year Eckerson (2002).

SQL uses `NULL` as a single placeholder object for representing incomplete information, and problems persist after so many years. When it comes to handling `NULL`s, SQL's behavior has been described as *"fundamentally at odds with the way the world behaves"* Date and Darwen (1996), or capable of *"ruining everything"* Celko (2010), and recommendations to avoid nulls altogether are not uncommon Darwen and Date (1995). The origin of SQL's controversial behavior with `NULL` can be traced back to Codd's original 12 rules for relational database management system.

> **Rule 3:** Null values (distinct from the empty character string or a string of blank characters and distinct from zero or any other number) are supported in fully relational DBMS for representing missing information and inapplicable information in a systematic way, independent of data type.

The third rule stated that **NULL** values should represent both “missing” and “inapplicable” information, but those two interpretations are often at odds with each other. The question has been addressed in the research literature, going all the way back to Zaniolo (1984) which defined three types of nulls: non-applicable, no-information, and those representing existing but currently unknown values. To go even further, consider a hypothetical example: we have a table with information about employees, and the salary of the CEO is given as **NULL**. This could have different meanings:

- *Non-applicable* (**NA**). The CEO may not receive a regular salary and use another remuneration scheme. Then **NULL** indicates a field that is non-applicable, for which a value does not exist.
- *Existing unknown* (**EU**). The CEO salary cannot be disclosed for privacy reasons. In such cases, **NULL** denotes an existing but currently unknown constant.
- *Existing known constant* (**C**). The CEO salary may not be specified because it depends on changing financial results of company operations. Here **NULL** denotes an existing, and known, value.
- *Dirty* (**D**). The CEO may receive a regular salary but the data source from which the table is populated may be dirty.
- *No-information* (**NI**). We may be in a situation when we know nothing at all about the reasons why that **NULL** is in the database.

In an attempt to deal with the many possible interpretations of **NULL**, the implementation of most SQL’s features is based on 3-valued-logic. As it is commonly assumed that programmers tend to think in terms of the familiar two-valued logic, SQL’s 3-valued-logic is often consider responsible when SQL behavior is not satisfactory. In practice the SQL evaluation of most queries would return answers compatible with the “inapplicable” (**NA**) interpretation of **NULL**. Moreover one can always consider a setting where non-applicable **NULL**s have been removed – e.g., by using the techniques of Franconi and Tessaris (2012). Hence most theoretical works on incomplete databases has traditionally focused on the interpretation of **NULL** as "missing" values (**EU**).

Since the early days of database field, the standard approach to answering queries over databases with **NULL** representing existing but unknown values has been based on finding answers one can be *certain* about, regardless of the interpretation of the missing data. This notion was introduced about 40 years ago Codd (1979); Grant (1977); Lipski (1979) and has since been studied on its own, leading to several conflicting mathematical definitions of certain answers. Abiteboul, Segoufin, and Vianu (2006); Grahne (1991); Imielinski and Lipski (1984); Libkin (2016a, 2016b); Lipski (1984); van der Meyden (1998). Moreover all these definitions are build with a single setting in mind: first-order (or closely related) queries over relational databases, in-

terpreted under set semantics. While understanding certainty and its computational properties in this setting was very useful Console, Guagliardo, Libkin, and Toussaint (2020), it nonetheless falls short of what one needs to deal with in realistic everyday queries, like those written in SQL. There are key shortcomings to existing techniques:

Data-model: Real-life relational databases operate with *bags* rather than sets. Syntactically straightforward extensions of certainty notions have been studied over bags Console, Guagliardo, and Libkin (2017) but they were not properly justified unlike their set-theoretic counterparts (and we shall see that this indeed leads to serious problems with existing definitions).

Query Language: Existing definitions only work for queries that essentially manipulate data. Real-life queries also *generate* new data values, by means of, for example, arithmetic operations or aggregates. In fact, in the standard TPC-H benchmark for evaluating SQL-database performance, over 90% of queries are value-inventing “TPC Benchmark™ H Standard Specification” (2018).

Pertinence: Despite its maturity, foundational research on incomplete data has not yet properly translated to practice. This gap between practice and theory is often justified by the fact that most theoretical solutions are inefficient (computing certain answers is coNP-hard even for first order queries) and rely on abstract model (marked nulls). The reality is that expectations of database practitioners with respect to incomplete data have not been systematically documented. It is not known if existing research results can readily be used to address the relevant practical challenges.

We have to address each of those shortcomings to provide more relevant answers for queries on incomplete databases.

In Chapter 3, we leverage our understanding of the notion of certainty for queries in SQL-like languages. We consider incomplete databases whose information content may be enriched by additional knowledge. The knowledge order among them is derived from their semantics, rather than being fixed a priori. The resulting framework allows us to capture and justify existing notions of certainty, and extend these concepts to other data models and query languages. As natural applications, we provide for the first time a well-founded definition of certain answers for the relational bag data model and for value-inventing queries on incomplete databases.

In Chapter 4, we present the first milestone to study the expectations of database practitioners with respect to incomplete data. We present the results of a survey we designed and conducted on the everyday usage of `NULL` values among SQL database users. We reach the conclusion that while `NULL` values are ubiquitous and problematic in real-life scenarios, foundational research on `NULL`s and has been addressing problems of limited practical relevance. The community has mostly focused on the "miss-

ing" value interpretation for NULL that only a minority of users encounter. Furthermore, theoretical solutions have been based on the assumption that, in the presence of NULL values and for a specific semantics, all users want the same answers. The results of the survey provide evidence that this assumption does not hold in practice.

In Chapter 5, we address the problem of the interpretation of incomplete data. Most existing frameworks are tight to the assumption that incompleteness should be interpreted as missing data, which can be completed thanks to database semantics and additional knowledge. To overcome this limitation, we model incompleteness on the level of database and query by considering a semantics function that maps a query and a database to a set of databases representing its possible answers. Especially, the query evaluation semantics can capture the no-information interpretation of NULL in relational databases. Another critical component to obtain pertinent answers is the notion of similarity between database objects. Indeed, the most pertinent answer is often defined as the database that is the least dissimilar to every other possible answer. For instance, the certain answers are based on a boolean similarity measure, namely the less-informative pre-order. To obtain a more flexible notion of answers, we propose a numerical similarity measure called regret. The regret capture the task-dependent loss a user endures when he considers a database as ground truth instead of another. The notion of answer defined with regret similarity measures is called risk-minimizing answers. We show that for some regret functions, the regret-minimizing answers coincide with the certain answers. Moreover, because regret functions are numerical, the risk-minimizing notion can define more nuanced and pertinent answers.

In Chapter 6, we leverage the practical advantages of using the risk to represent the degree of incompleteness. The risk can be computed for any answer. We can especially compute the risk of the answers returned by existing database management systems. Then instead of trying to change the behavior of a system, we propose to explain and improve the answers it returns. We lay down the theoretical foundation to partition the query evaluation semantics additional knowledge into incompleteness sources. Then we reason about each source of incompleteness and determine how they impact the risk of answers. We defined two measures: The importance of cleaning is a prediction of the risk evolution if we remove a source of incompleteness. The contribution to the risk is a game theoretical perceptive on the correlated impact of each source of incompleteness on the risk. To study the framework's applicability and the precision of its inference models, we experimented with relational databases and queries with aggregate and grouping operations. Despite some remaining challenges, the results are promising.

"There is a cat in a box . . .
Some argue that the cat may be dead or alive. Others consider that the cat is either certainly dead or certainly alive. Schrodinger insists that the cat can be both dead and alive, while Bohr denies the cat's existence altogether. Thankfully, SQL accommodates all and proclaims the cat to be **NULL**.*"*

Luke Darlow & Etienne Toussaint

Example 1. Let the following table representing the information about employees:

Person	Age	Salary
John	**NULL**	2000
Jane	**NULL**	**NULL**

Let a query q_1 returning the name of all employees older than 18, with the standard SQL evaluation and the **NA** interpretation of **NULL**, the answer is the empty set. Similarly the answer to a query q_2 returning the name of all employees younger than 18 is also empty. Finally, the evaluation of tautology $q_1 \cup q_2$ returns the empty set. Without explicit access to the database, one could easily conclude that the company does not have any employees which seems at odd even with the **NA** interpretation of **NULL**. On the other hand, if we consider the notion of certain answers with nulls, then the answer to $q_1 \cup q_2$ is $\{\text{John}, \text{Jane}\}$.
If we ask for the average salary in the company, SQL assumes that Jane's salary is non-applicable and would return 2000. Therefore the SQL evaluation loses the information about the existence of a **NULL** placeholder in the Salary column. Depending on the user's interpretation of **NULL** and his preferences, the placeholder **NULL** could also be a valuable answer. In this thesis, we explore various frameworks to define relevant answers on incomplete databases, and we argue that there is no good answer if no explanation about its origin can be provided.

1.1 Related work

The subject of incomplete data has been addressed in the literature, starting from foundational papers in the 1970s Codd (1975); Grant (1977); Lipski (1979), and several surveys are available; e.g., Console et al. (2020); Greco, Molinaro, and Spezzano (2012); van der Meyden (1998). In this overview of related work, we concentrate on the interpretations that are commonly accepted, and especially outline what has been done for the existing unknown semantics of nulls, which is the subject of most of the material in the aforementioned surveys and the basis of this thesis.

Nulls in SQL. Even prior to the development of SQL, an ANSI committee working on the then new relational model suggested 14 different interpretations of nulls "Interim Report: ANSI/X3/SPARC Study Group on Data Base Management Systems" (1975). Most of these fall into the categories we introduced earlier, with some exceptions that have been made obsolete by subsequent developments. While "Interim Report: ANSI/X3/SPARC Study Group on Data Base Management Systems" (1975) did not provide any semantics of operations on nulls, nor a logic for their handling, this was suggested in two later documents Cannan, Dee, and Kerridge (1987); *Introduce named null definitions* (1990) from the SQL standardization process; these proposals date back to the late 1980s, but no action with respect to the Standard was taken. They proposed user-defined nulls, a model moving in the direction of marked nulls, with user-defined meanings, and SQL's `NULL` as the fallback position.

Shortcomings of SQL's `NULL`s are anecdotally known in the community, with only few exceptions trying to systematize them in the research literature. Among those are Brass and Goldberg (2006), which looks into issues experienced by SQL programmers in general, not with nulls directly, and Neumann (2018), which studies how aggressive optimization techniques result in incorrect query evaluation in the presence of nulls.

***Existing unknown* EU and *Dirty* D: imputation.** A well-established line of research that concerns these two interpretations is to impute null values. That is, nulls are replaced with actual constants according to some statistical model. In the end, this gives us a database without nulls that can be queried and analyzed assuming the data is clean and complete. Most of this work is seen in the statistical and data cleaning literature Gao and Miao (2018); van der Loo and de Jonge (2018) and these approaches are fundamentally geared towards machine learning tasks on data derived from a database.

***Non-applicable* (NA) and *No-information* (NI) nulls.** Some early work on query evaluation with nulls under the **NI** semantics was done in Zaniolo (1984), which proposed the trichotomy of **NI**, **NA**, and **EU** nulls. This was followed by work on **NA** in Lerat and Jr. (1986), essentially treating them as problems with the database design. There are different lines of work on **NA** and **NI**, each going in its own direction. For example, Gottlob and Zicari (1988) extended truth tables and arithmetic operations to the **NA** null, and Atzeni and Morfuni (1984); Hartmann and Link (2012) looked at functional dependencies over databases with **NA** and **NI** nulls. One direction of work is to capture the semantics of different nulls by extending the 3-valued logic of SQL with more values and even paraconsistent logics Arieli, Avron, and Zamansky (2010); Con-

sole, Guagliardo, and Libkin (2016); Gessert (1990); Thalheim and Schewe (2010); Yue (1991). However it was shown recently that SQL's 3-valued logic is the most rational choice of a propositional logic that does not hamper query evaluation Console, Guagliardo, and Libkin (2022).

On the query evaluation side, we mention Candan, Grant, and Subrahmanian (1997); Franconi and Tessaris (2012); Klein (2001). The approach of Klein (2001) is based on the notion of subsumption of tuples with nulls, already present in Buneman, Jung, and Ohori (1991); Zaniolo (1984), and defined the semantics of **EU** and **NI** by different ways of lifting this subsumption from tuples to relations. This led to a well-defined notion of query answering and approximation for the **EU** semantics, but not for the **NI** semantics, where the discussion stopped at the level of investigating a number of concrete SQL queries. A step further was made in Candan et al. (1997), which looked at **NI** added as either a substitute for a value, or a disjunction of several values (this, by itself, is a model of incompleteness known as *or-sets* Imielinski, Naqvi, and Vadaparty (1991); Libkin and Wong (1996)). These can be expressed by means of extra constraints attached to tuples in the spirit of conditional tables Imielinski and Lipski (1984). The paper shows then how to use such conditions to evaluate relational algebra queries; the result is again a conditional table, which can complicate understanding the output from a user's perspective.

Last but not least, Franconi and Tessaris (2012) formally shows that answering queries (in relational algebra and calculus) on databases where nulls are interpreted under the **NA** semantics is equivalent to naively evaluating a rewritten query on a decomposed database without nulls, in which the "absence" of values is implicitly encoded in the schema. From a practical point of view, however, no concrete implementation and experimental evaluation of such rewriting and decomposition algorithms were devised.

***Existing unknown* EU and certain answers.** Theoretical work on incomplete databases has traditionally focused on the interpretation of nulls as missing values, in the sense of the **EU** semantics: a value exists, but it is currently unknown. The latest survey in that area is only two years old, and we refer the reader to it for multiple additional details and references Console et al. (2020), while we here outline only the part relevant to the thesis. The prevalent model in this line of research is that of so-called *marked nulls*, first systematically studied in Imielinski and Lipski (1984), where each missing value is represented by a special null symbol with an associated identifier. For example, the symbols in the "age" column of the table in Figure 1.1a express the fact that John, Mary and Jane do have an age, but we do not know how old they actually are.

Person	Age
John	$\perp_1$
Mary	$\perp_2$
Jane	$\perp_1$

(a)

Person	Age
John	NULL
Mary	NULL
Jane	NULL

(b)

Figure 1.1: Incomplete tables with **EU** nulls: (a) in the model with marked nulls, and (b) in SQL.

There is a clear mismatch between the model of nulls used in theoretical database research and that of SQL; even if we interpret SQL nulls solely under the **EU** semantics, marked nulls are more expressive, because they allow co-reference of nulls, which is not possible in SQL. For example, the table in Figure 1.1b does not capture the fact that John and Jane have the *same* (but unknown) age.

It is folklore in the database theory community that SQL nulls can be modeled by non-repeating marked nulls (i.e., with distinct identifiers). This, however, is not the case, as was demonstrated in Guagliardo and Libkin (2019). Nonetheless, the marked nulls model dominated not only research on incomplete information, but also work in areas where incomplete databases naturally arise. Examples of these include ontology-based data access Poggi et al. (2008) and data exchange Fagin, Kolaitis, Miller, and Popa (2005). We refer the reader to specific surveys of these areas Arenas, Barceló, Libkin, and Murlak (2014); Bienvenu and Ortiz (2015), with hundreds of further references, and detailed explanations of the usage of marked nulls under the **EU** interpretation.

For marked nulls, the preferred model of query answering is that of *certain answers* Imielinski and Lipski (1984); Lipski (1979). These are answers that are true under all interpretations of nulls, i.e., all possible ways of assigning known values to them. The definition of certainty itself is not unique Libkin (2016a); Lipski (1984), but in all cases query answering is computationally hard; e.g., coNP-complete for relational algebra queries Abiteboul, Kanellakis, and Grahne (1991). The standard query evaluation (a.k.a. *naïve*), where nulls are treated like regular constants, does sometimes produce answers that come with certainty guarantees; this is the case for unions of conjunctive queries (i.e., select-project-join-union queries) Imielinski and Lipski (1984), or their extension with relational algebra division Gheerbrant, Libkin, and Sirangelo (2014). Given the general hardness of computing certain answers, recent efforts concentrated on finding efficient approximations Feng, Huber, Glavic, and Kennedy (2019); Fiorentino, Greco, Molinaro, and Trubitsyna (2018); Guagliardo and Libkin (2016).

But, there are no developed algorithms, nor even a generally accepted definition, for extending certainty to queries with aggregation, which are extremely common in applications. Thus, despite much solid research in this area, its outcomes are often not directly applicable.

Chapter 2

Preliminaries

2.1 Incomplete Databases

2.1.1 Database domain

We present the general framework of Libkin (2016a) by considering databases as abstract "objects"; these could be relational databases, or graphs, or XML documents, or other. Later we look at the relational setting, here to make clear the distinction with concrete relational databases, we denote database objects with lowercase letters x, y, and so on.

Definition 2.1.1 (Database domain)**.** *A* database domain $\mathbf{D}$ *is a triple* $(\mathbb{I}, \mathbb{C}, [\![\cdot]\!])$*, where* $\mathbb{I}$ *is a set of database objects,* $\mathbb{C} \subseteq \mathbb{I}$ *is the set of complete objects (over which queries are defined), and* $[\![\cdot]\!]$ *is a semantic function from* $\mathbb{I}$ *to the powerset of* $\mathbb{C}$ *such that* $x \in [\![x]\!]$ *for every* $x \in \mathbb{C}$*.*

For convenience of notation, we also use $\mathbb{I}_{\mathbf{D}}$, $\mathbb{C}_{\mathbf{D}}$, $[\![\cdot]\!]_{\mathbf{D}}$ to refer to the set of incomplete objects, the set of complete objects, and the semantic function, respectively, of a database domain $\mathbf{D}$.

The elements of $[\![x]\!]$ are referred to as the *possible worlds* of x. Intuitively, the more possible worlds a database object represents, the more ambiguous it is. To make this intuition formal, with each database domain $\mathbf{D} = (\mathbb{I}, \mathbb{C}, [\![\cdot]\!])$, we associate an information pre-order $\preceq_{\mathbf{D}}$ (that is, a reflexive and transitive relation) on $\mathbb{I}$, defined as follows: $x \preceq y$ iff $[\![y]\!] \subseteq [\![x]\!]$. That is, $x \in \mathbb{I}$ is *less informative* than $y \in \mathbb{I}$ if every possible world of y is also a possible world of x.

Given two database domains $\mathbf{S}$ (for source) and $\mathbf{T}$ (for target), a *query* from $\mathbf{S}$ to $\mathbf{T}$ is a mapping q from $\mathbb{C}_{\mathbf{S}}$ to $\mathbb{C}_{\mathbf{T}}$, i.e., it maps complete databases of a source database domain $\mathbf{S}$ to complete databases of a target database domain $\mathbf{T}$. The natural question that arises is how to define relevant answers for query on database which are not complete.

2.1.2 Certain anwers

A natural way to define query answers on an incomplete database object x is to consider answers that are "true" in all possible worlds of x. To this end, for a set C of complete database objects, we let

$$q(C) = \{\, q(x) \mid x \in C \,\}.$$

Then, to formalize the notion of truth, we assume, for each database domain, the existence of a set $\mathbb{F}$ of formulae expressing *knowledge* about incomplete objects, and a satisfaction relation $\models$ indicating when a formula ϕ is true in a possibly incomplete database object x, written $x \models \phi$.

For a set X of database objects we write $X \models \phi$ if $x \models \phi$ for each $x \in X$. Similarly, for a set Φ of formulas, we write $x \models \Phi$ if $x \models \phi$ for each $\phi \in \Phi$. The *theory* of X is the set $\mathsf{Th}(X) = \{\phi \mid X \models \phi\}$, and the *models* of Φ are given by $\mathsf{Mod}(\Phi) = \{x \mid x \models \Phi\}$. Intuitively, $\mathsf{Th}(X)$ contains the knowledge we have about objects of X, and $\mathsf{Mod}(\Phi)$ consists of all objects satisfying the knowledge expressed by Φ. When the database domain is not clear from the context, we will explicitly indicate it as superscript in Th by writing, e.g., $\mathsf{Th}^{\mathbf{D}}$.

For a database domain $\mathbf{D}$, we want the knowledge $\mathbb{F}$ to satisfy some minimal requirements, such as being compatible with the information pre-order $\preceq_{\mathbf{D}}$, and being expressive enough to capture at least the semantics of incompleteness $[\![\cdot]\!]_{\mathbf{D}}$. To this end, we require that

1. for every $\phi \in \mathbb{F}$ and every $x, y \in \mathbb{I}_{\mathbf{D}}$, if $x \preceq_{\mathbf{D}} y$ and $x \models \phi$ then $y \models \phi$;
2. for every $x \in \mathbb{I}_{\mathbf{D}}$ there exists a formula $\delta_x \in \mathbb{F}$ equivalent to $\mathsf{Th}(x)$ (i.e., $\mathsf{Mod}\big(\mathsf{Th}(x)\big) = \mathsf{Mod}(\delta_x)$), furthermore if $x \not\preceq_{\mathbf{D}} y$, then $y \not\models \delta_x$.

We are finally ready to define what is certainly true when answering queries on incomplete database objects.

Definition 2.1.2. *Let q be a query from $\mathbf{S}$ to $\mathbf{T}$, and let $x \in \mathbb{I}_{\mathbf{S}}$. The* certain knowledge *of q on x is the set* $\mathsf{Th}^{\mathbf{T}}\big(q([\![x]\!]_{\mathbf{S}})\big)$.

Thus, the certain knowledge of q on x is the set of formulae that are true in all query answers over the possible world of x . However, in practice, we expect the answer to a query on a database object to be an object itself, ideally one that captures the entire certain knowledge of the query, that is, $o \in \mathbb{I}_{\mathbf{T}}$ such that $\mathsf{Th}^{\mathbf{T}}(o) = \mathsf{Th}^{\mathbf{T}}(q([\![x]\!]_{\mathbf{S}})$. Unfortunately, such an object need not exist in general. To overcome this issue, we define the certain answer as the most informative object (w.r.t. $\preceq_{\mathbf{T}}$) whose theory is a subset of the certain knowledge: $\mathsf{Th}^{\mathbf{T}}(o) \subseteq \mathsf{Th}^{\mathbf{T}}(q([\![x]\!]_{\mathbf{S}})$. Therefore, the definition of certain answers as an object is an under-approximation of the certain knowledge; the most informative one allowed by the database domain, but an approximation nonetheless.

Definition 2.1.3. *Let q be a query from $\mathbf{S}$ to $\mathbf{T}$, and let $x \in \mathbb{I}_{\mathbf{S}}$. The* information-based certain answer *to q on x is*

$$\mathsf{cert}_{\square}(q, x) \;=\; \mathsf{glb}_{\preceq_{\mathbf{T}}}\, q([\![x]\!]_{\mathbf{S}})\,, \tag{2.1}$$

where the greatest lower bound glb *is with respect to $\preceq_{\mathbf{T}}$.*

That is, the most informative object that is less informative than every possible answer. This notion satisfies the expected property of answers on incomplete databases: more informative query inputs yield more informative query answers.

Proposition 1 (Libkin (2016a))**.** *Let q be a query from $\mathbf{S}$ to $\mathbf{T}$, and let $x, y \in \mathbb{C}_{\mathbf{S}}$ be such that $x \preceq_{\mathbf{S}} y$. If* $\mathsf{cert}_{\square}(q, x)$ *and* $\mathsf{cert}_{\square}(q, y)$ *exist, then* $\mathsf{cert}_{\square}(q, x) \;\preceq_{\mathbf{T}}\; \mathsf{cert}_{\square}(q, y)$.

Even though the certain answers as object are an approximation of the certain knowledge, this does not guarantee their existence. Next, we discuss this problem in the context of relational databases.

2.2 Set Relational Databases

Despite the fact that commercial relational database management system are based on bags (a.k.a. multiset), and a single placeholder object to represent incomplete data **NULL**, the standard theoretical model for relational database is based on sets and *marked nulls*. Formally, we consider that relational databases are populated by two kinds of values, *constants* and *marked nulls*. These come from two disjoint countably infinite sets, Const and Null, respectively. We denote the elements of Null using the symbol $\perp$ with subscripts.

A relational database schema is a finite set of table names with associated arities, and a k-ary table is a finite set of k-tuples over $\mathsf{Const} \cup \mathsf{Null}$. Then, a set relational database d over a given schema maps each table name R in the schema to a table R^d of appropriate arity. We write $\mathsf{Const}(d)$ and $\mathsf{Null}(d)$ for the set of constants and nulls occurring in d, respectively. The *active domain* of d is $\mathsf{adom}(d) = \mathsf{Const}(d) \cup \mathsf{Null}(d)$. For convenience of notation, we sometimes represent a set relational database as a set of facts; e.g., $d = \{R(1, \perp_1), S(\perp_1, 2), R(1, \perp_1)\}$ is the database d such that $R^d = \{(1, \perp_1), (1, \perp_1)\}$ and $S^d = \{(\perp_1, 2))\}$.

A relational databases d is said to be incomplete when some of its values are elements of the set Null. An element $\perp_i \in \mathsf{Null}(d)$ is a placeholder object to represent incomplete information. However this incompleteness can have different interpretation, it could mean that the information is 'inapplicable' but also that it is 'missing'. Most of the literature consider that null values represent 'missing' information, more specifically the null is used as a placeholder for an existing constant which is currently unknown.

In the context of null as 'missing' values, an incomplete relational databases can be seen as a compact representation of many possible worlds. We define a *valuation* v on a relational database d as a partial mapping $v : \text{Null}(d) \to \text{Const}$ that assigns constant values to nulls occurring in the database. If v is defined on all elements of $\text{Null}(d)$, we say that v is d-*complete*. By $v(d)$ we denote the result of replacing each null $\perp_i$ with $v(\perp_i)$ in d. The *semantics* $[\![d]\!]$ of an incomplete database d is the set of all complete databases it can represent, i.e.,

$$[\![d]\!]_{\text{CWA}} = \{v(d) \mid v \text{ is a } d\text{-}\textit{complete} \text{ valuation}\}\,.$$

This is known as the closed-world semantics of incompleteness, or semantics under CWA, or *closed-world assumption* Reiter (1977).

Under CWA, the facts stored in a database are assumed to be the whole truth. When used on the target domain of queries, this would provide a precise semantics for answers; however, in such a case, information-based certain answers may not exist even for very simple queries Libkin (2016a). To see this, consider the database $d = \{R(\perp_i)\}$ and the Boolean query q returning true iff $R(2)$ belongs to the database. Then, the certain knowledge of q on d is the set of formulae satisfied by both $\{()\}$ (when $\perp_i \mapsto 2$) and $\emptyset$ (otherwise); therefore, this knowledge is empty and, under CWA, there exists no database whose theory is the empty set. The reason for this is that all facts in a database are true, but at the same time, under CWA, all other facts not in it are assumed to be false. So, the theory of the empty database is not empty under CWA.

To overcome this difficulty, we often consider the *open-world assumption* of incompleteness, under which the facts stated in a database are true but they are not assumed to be the whole truth. The semantics of incompleteness under OWA, or *open-world assumption* is defined as

$$[\![d]\!]_{\text{OWA}} = \{\text{complete } d' \mid v(d) \subseteq d' \text{ for a } d\text{-}\textit{complete} \text{ valuation } v\}\,.$$

When OWA semantics is used on the target domain, the more facts an answer contains, the more knowledge it provides. This fits in well with the idea of approximating query answers: finding additional tuples in the answer makes approximations more informative. Also, with OWA, unlike with CWA, the theory of the empty database is the empty set: $\text{Th}(\emptyset) = \emptyset$. Therefore, as we shall see, the information-based certain answers exist for larger classes of queries.

Definition 2.2.1. *A* set relational *database domain* $\mathbf{D}$ *is such that* $\mathbb{I}_{\mathbf{D}}$ *consists of set relational databases populated with elements of* Const $\cup$ Null*,* $\mathbb{C}_{\mathbf{D}}$ *is the subset of* $\mathbb{I}_{\mathbf{D}}$ *of databases without nulls, and* $[\![\cdot]\!]_{\mathbf{D}}$ *is either the* CWA *or* OWA *semantics of incompleteness.*

2.2.1 Certain answers for relational databases

A k-ary *query* is a map that, with a database d, associates a subset of k-tuples over its elements, i.e., a subset of $\mathsf{adom}(d)^k$. Queries in standard languages such as relational algebra, calculus, Datalog, etc., cannot invent new values, i.e., return constants that are not in the active domain. Such a query is called *generic* if it commutes with permutations of the domain of constants.

Definition 2.2.2 (Generic queries)**.** *A query* q *is* generic *if* $q\big(\pi(d)\big) = \pi\big(q(d)\big)$ *whenever* $\pi\colon \mathsf{Const} \to \mathsf{Const}$ *is a bijection. The class of generic queries will be denoted by* GEN*.*

For the source domain $\mathbf{S}$, either the semantics CWA or OWA can be used: the former assumes we completely know the real world, while the latter allows for the possibility that we may be missing some facts. When this choice is relevant for the results, we indicate it by saying that a query, or class of queries, is "under CWA" (resp., under "OWA").

Proposition 2 (Amendola and Libkin (2018))**.**

(a) The information-based certain answer always exists for generic queries under CWA*;*

(b) There exists a generic query such that the information-based certain answer does not exist under OWA*.*

Thus, OWA on the source makes a huge difference in the class of queries for which the information-based certain answers exist. The reason for this is that we can find a generic query q and a database d (under OWA) for which the certain knowledge contains infinitely many non-equivalent formulae. Then, for every finite answer database a whose theory is contained in this infinite set, we can always find a finite a' such that:

$$\mathsf{Th}(a) \subset \mathsf{Th}(a') \subseteq \mathsf{Th}\big(q([\![d]\!]_{\mathsf{OWA}})\big)$$

and therefore the greatest lower bound does not exist.

Other notions of certain answers exist in the literature. The most common ones are intersection-based certain answers Abiteboul et al. (1991); Imielinski and Lipski (1984); Lipski (1979); Reiter (1986), and certain answers with nulls Libkin (2016b); Lipski (1984). While these are specific to the relational domain, and therefore less general than information-based certain answers, they have the advantage of existing for larger classes of queries.

Definition 2.2.3. *The* intersection-based certain answer *to a query q on a relational database d is the intersection of the answers to q on every possible world represented by d:*

$$\mathsf{cert}_{\cap}(q,d) = \bigcap \{ q(d') \mid d' \in [\![d]\!]_{\mathbf{S}} \} \tag{2.2}$$

Observe that $\mathsf{cert}_{\cap}(q,d)$ consists solely of constants. When the target domain of queries allows only for databases without nulls, the intersection-based certain answers are precisely the information-based ones:

Proposition 3 (Amendola and Libkin (2018))**.** *Let $\mathbf{S}$ and $\mathbf{T}$ be relational database domains such that*

(a) $\mathbf{S}$ is under either OWA *or* CWA*,*

(b) $\mathbf{T}$ is under OWA*, and $\mathbb{I}_{\mathbf{T}}$ consists of databases without nulls.*

Then, for every generic query q from $\mathbf{S}$ to $\mathbf{T}$ and for every database $d \in \mathbb{I}_{\mathbf{S}}$, we have that $\mathsf{cert}_{\square}(Q,D)$ *and* $\mathsf{cert}_{\cap}(Q,D)$ *exist and coincide.*

One of the problems with intersection-based certain answers is that they only consist of constants and, for this reason, may miss tuples that are in fact certain. To see this, consider the database $d = R(\bot_i)$ and the query q that simply returns R. Then $\mathsf{cert}_{\cap}(q,d) = \emptyset$, even though we are certain that $\bot_i$ is in R no matter which missing value it represents. To overcome this shortcoming, nulls in certain answers can be allowed as follows:

Definition 2.2.4. *The* certain answers with nulls *to a query q on a relational database d is the table* $\mathsf{cert}_{\bot}(q,d)$ *such that, for every d-complete valuation v and for every $c \in [\![v(d)]\!]_{\mathbf{S}}$, each tuple in $v\big(\mathsf{cert}_{\bot}(q,d)\big)$ is also in $q(c)$:*

$$\mathsf{cert}_{\bot}(q,d) = \big\{ \bar{t} \mid v(\bar{t}) \in q(c) \text{ *for every } d\text{-complete valuation } v \text{ and for every } c \in [\![v(d)]\!]_{\mathbf{S}} \big\}$$

When the source relational database semantics is that of CWA, the definition becomes

$$\mathsf{cert}_{\bot}(q,d) = \big\{ \bar{t} \mid v(\bar{t}) \in q\big(v(d)\big) \text{ for every } d\text{-complete valuation } v \big\}.$$

With the above example of d containing $R(\bot_i)$, and the query q that simply returns R, we have $\mathsf{cert}_{\bot}(R,d) = \{\bot_i\}$, i.e., we keep the certain information about $\bot_i$ being in R.

While $\mathsf{cert}_{\bot}(q,d)$ may contain nulls and constants, these may only come from element of the database d. Thus, certain answers with nulls exist for every generic query q and for every database d.

Unlike intersection-based certain answers, certain answers with nulls cannot be captured by the information-based ones. This is due to the fact that it does not keep track of what nulls are mapped to. With the above example of d containing $R(\bot_i)$, and the query q that simply returns R, we have $\text{cert}_\square(R, d) = \{\bot_j\}$, where $\bot_j$ is a fresh null. Since null labels are not preserved, there is no relationship between $\bot_j$ in the output, and $\bot_1$ in the input. In particular, it is not enforced that the value $\bot_j$ in the output and the value $\bot_i$ in the input must be equal.

2.2.2 Complexity of Query evaluation

The standard language for query on the set relational model is called *relational algebra* (RA) and has the following syntax:

$$\begin{aligned} q := R \mid \pi_{\bar{\alpha}}(q) \mid \sigma_{\alpha_i=\alpha_j}(q) \mid \sigma_{\alpha_i\neq\alpha_j}(q) \\ \mid q \times q \mid q \cup q \mid q \cap q \mid q - q \end{aligned}$$

where α_i and α_j are attributes, and $\bar{\alpha}$ is possibly empty tuple of attributes.

The semantics of a (well-formed) RA query q is given by inductively defining the quantity set $q(d)$, which is the the result of applying q to a database d. This is done as follows:

$$\begin{aligned} R(d) &= R^d \\ \pi_{\bar{\alpha}}(q)(d) &= \{\bar{t} \mid \exists \bar{t}' \in q(d), \pi_{\bar{\alpha}}(\bar{t}') = \bar{t}\} \\ \sigma_{\alpha_i=\alpha_j}(q)(d) &= \{\bar{t} \mid \bar{t} \in q(d), \bar{t}[\alpha_i] = \bar{t}[\alpha_j]\} \\ \sigma_{\alpha_i\neq\alpha_j}(q)(d) &= \{\bar{t} \mid \bar{t} \in q(d), \bar{t}[\alpha_i] \neq \bar{t}[\alpha_j]\} \\ (q \times q')(d) &= \{\bar{t}\,\bar{t}' \mid \bar{t} \in q(d), \bar{t}' \in q'(d)\} \\ (q \cup q')(d) &= \{\bar{t} \mid \bar{t} \in q(d) \vee \bar{t} \in q'(d)\} \\ (q \cap q')(d) &= \{\bar{t} \mid \bar{t} \in q(d) \wedge \bar{t} \in q'(d)\} \\ (q - q')(d) &= \{\bar{t} \mid \bar{t} \in q(d) \wedge \bar{t} \notin q'(d)\} \end{aligned}$$

where $\bar{t}[\alpha_i]$ denotes the α_i element of $\bar{t}$, and $\pi_{\alpha_1,\cdots,\alpha_i}(\bar{t})$ is the tuple $(\bar{t}[\alpha_1], \cdots, \bar{t}[\alpha_i])$.

The size of a query is often negligible with respect to the size of the database we often consider *data complexity*.

Definition 2.2.5 (M. Y. Vardi (1986))**.** *Let* $\mathbf{S}$ *and* $\mathbf{T}$ *be relational database domains, and* L *a set of queries from* $\mathbf{S}$ *to* $\mathbf{T}$*. The* data complexity *of a query evaluation algorithm* $\text{Eval}\colon \text{L} \times \mathbb{I}_\mathbf{S} \to \mathbb{I}_\mathbf{T}$ *is defined for every* k*-query* q *in* L *as the complexity of the decision problem* $\text{Eval}[q]$*:*

- ***Input:*** *A database* $d \in \mathbb{I}_\mathbf{S}$*, and a tuple* $\bar{t} \in (\text{Const} \cup \text{Null})^k$*.*

- ***Output:*** $\bar{t} \in \text{Eval}(q, d)$ *?*

Theorem 1 (M. Y. Vardi (1986))**.** *For every query* $q \in$ RA*, given a relational database without null* c*, and a tuple* $\bar{t}$*, deciding whether* $\bar{t} \in q(c)$ *is in* LOGSPACE*.*

In one hand the evaluation of relational algebra queries over complete databases is really efficient. On the other hand the evaluation of the miscellaneous notion of certain answers is untractable.

By definition, the information-based certain answer is the most informative database consistent with all query answers. Since answers are interpreted under OWA, more informativeness means more tuples, which of course comes at a cost in space.

Theorem 2 (Amendola and Libkin (2018); Arenas, Botoeva, Kostylev, and Ryzhikov (2017))**.** *For generic queries, under the* CWA *semantics of input databases, the size of the information-based certain answer is at most doubly exponential in the size of the database. Moreover, under both* OWA *and* CWA *interpretation of input databases, there exist a relational algebra query* $q \in$ RA *and a database* d *for which the size of* $\text{cert}_{\square}(q, d)$ *is exponential in the size of* d*.*

At the moment we do not know how to close the gap between the single-exponential lower bound and the double exponential upper bound for queries on CWA databases. This depends on some unresolved problems related to families of cores of graphs Hell and Nešetřil (2004); see Amendola and Libkin (2018) for more details.

On the other hand, as the certain answer with nulls consists only of tuples over the domain of the database, the size of $\text{cert}_{\perp}(q, d)$ is at most polynomial in the size of d. However, computing it is intractable in data complexity under CWA, and undecidable (still in data complexity, i.e., for a fixed query) under OWA.

Theorem 3 (Abiteboul et al. (1991); Gheerbrant and Libkin (2015))**.** *Under* OWA*, there exist a (fixed) relational algebra query* $q \in$ RA *for which, given a database* d *and a tuple* $\bar{t}$*, it is undecidable whether* $\bar{t} \in \text{cert}_{\perp}(q, d)$ *(resp.,* $\bar{t} \in \text{cert}_{\square}(q, d)$*) (resp.,* $\bar{t} \in \text{cert}_{\cap}(q, d)$*).*

Under CWA*, there exist a (fixed) query* $q \in$ RA *such that deciding, given a database* d *and a tuple* $\bar{t}$*, whether* $\bar{t} \in \text{cert}_{\perp}(q, d)$ *(resp.,* $\bar{t} \in \text{cert}_{\square}(q, d)$*) (resp.,* $\bar{t} \in \text{cert}_{\cap}(q, d)$*) is* coNP*-complete.*

The intractability of computing certain answers yield to the study of fragment of relation algebra for which its computation is tractable. The most common restriction is called *union of conjunctive queries* denoted UCQ and has the following syntax:

$$q := R \mid \pi_{\bar{\alpha}}(q) \mid \sigma_{\alpha_i = \alpha_j}(q)$$
$$\mid q \times q \mid q \cup q$$

Proposition 4 (Gheerbrant et al. (2014))**.** *For every union of conjunctive queries* $q \in$ UCQ *under both* OWA *and* CWA *interpretation of input databases, deciding, given a database* d *and a tuple* $\bar{t}$ *whether* $\bar{t} \in \mathsf{cert}_{\perp}(q, d)$ *(resp.,* $\bar{t} \in \mathsf{cert}_{\square}(q, d)$*) (resp.,* $\bar{t} \in \mathsf{cert}_{\cap}(q, d)$*) is in* PTIME*.*

The tractable algorithm to compute certain answers over union of conjunctive queries is based on *naive evaluation*. The idea of naive evaluation is simple: treat nulls as new values, and evaluate the query by using normal evaluation techniques on databases with nulls. For example, if we have a graph with edges $\{(1, \perp_1), (\perp_1, 2), , (\perp_2, 3)\}$ then evaluating a query q naively amounts to changing $\perp_1$ to a new constant c_1 and $\perp_2$ to a new different constant c_2, then evaluating q on the database $\{(1, c_1), (c_1, 2), (c_2, 3)\}$.

Formally, we say that $v : \mathsf{Null}(d) \to \mathsf{Const}$ is a *bijective valuation* if it is a bijection and $v\big(\mathsf{Null}(d)\big)$ is disjoint from $\mathsf{adom}(D)$ and all the constants mentioned in q. Then

$$\mathsf{naive}(q, d) \;=\; v^{-1}\Big(q\big(v(d)\big)\Big)\,.$$

It is easy to see that for queries that are generic, i.e., that are invariant under permutations of the domain (see Section 2.2.2), this definition does not depend on the choice of a particular valuation v.

Chapter 3

Knowledge-preserving certain answers

Based on 'Knowledge-Preserving Certain Answers for SQL-like Queries' published in KR 2020, Toussaint, Guagliardo, and Libkin (2020).

The key shortcomings of existing techniques to handle incomplete relational databases are of two kinds. To start with, real-life databases operate with *bags* rather than sets. Syntactically straightforward extensions of certainty notions have been studied over bags Console et al. (2017) but they were not properly justified unlike their set-theoretic counterparts (and we shall see later that this indeed leads to serious problems with existing definitions). Furthermore, existing notions only work for queries that essentially manipulate data. Real-life queries also *generate* new data values, by means of, for example, arithmetic operations or aggregates. In fact, in the standard TPC-H benchmark for evaluating SQL-database performance, over 90% of queries are value-inventing "TPC Benchmark™ H Standard Specification" (2018), and all of them use bag semantics.

Our goal is thus twofold. We want to build an abstract framework for justifying the notion of certainty, and upon validating it, we shall use it to explain what certainty is for realistic SQL-like queries that use bag semantics and value-invention. By validating we mean that it should capture existing notions in the setting where they are well understood, namely relational databases under set semantics, and no value invention in queries.

We follow the approach presented in Preliminaries (see. Section 2.1.1) that has the advantage of being applicable in different data models. The key concept is that of informativeness of databases: a database x is more informative than a database y if all the possible worlds it represents are also possible worlds represented by y. Intuitively the more informative a database is, the fewer worlds it may represent and the less ambiguous it is. Then the certain answer to a query on an incomplete database x is the most informative database which is not more informative than the query answer

on every possible world of x. This however may be too permissive as it misses the reason, or *explanation*, why a complete database is a possible world for an incomplete database x. In general an explanation can turn an incomplete object into a more informative object; explanations compose, and certain answers must preserve the informativeness order provided by the explanation. Such an order essentially says that applying an explanation to a database results in a more informative database. Using this idea, Amendola and Libkin (2018) developed a framework that was capable of explaining the commonly used definition of certainty of Lipski (1984), for relational database queries under set semantics and informativeness orders naturally imposed by such a model. The main drawback of the approach is that it is too closely tied to a particular data model (relations as sets) and the informativeness orders it imposes, and to particular features of query languages (no bags, no value invention).

To overcome those problems, we refine the information-based framework in a way that opens up an approach to defining certainty for most typical queries that occur in SQL-like languages over real-life databases. The key concept is that of *knowledge preservation* to improve the information content of answers to queries on incomplete databases. Intuitively, the interpretation of missing data may be specified by some additional knowledge, and hence the information content of a database is defined with respect to this knowledge. Consequently, certainty of an answer to a query q on a database x means two things. First, it is no more informative than answers to q in all possible worlds x. Second, if c is a possible world for x obtained by providing some new knowledge, then the answer to q on c must also be a possible world for the answer to q on x with this new knowledge.

The main distinguishing feature of this approach is that we derive the knowledge-preserving information pre-order from the semantics of data, rather than have it as a basic notion in the model. The latter restricted us to a handful of cases where such an ordering already existed, such as familiar open- and closed-world interpretation of incomplete databases under set semantics. The new approach allows us to unify already existing notions of certain answers without using any specific database model. The framework is very natural to instantiate for other data models and for expressive query languages. We demonstrate it by showing that it allows us for the first time to give a well-founded notion of certain answers on bag relational databases. We demonstrate its additional power with respect to query language features by showing that we can define, again for the first time, a notion of certain answers for value-inventing queries. We can further use the approach to devise an implementation-friendly approximation scheme for such certain answers.

3.1 Abstract Framework

An incomplete database x is less informative than y if every possible world of y is also a possible world of x. Suppose now that we discover some new information that reduces the ambiguity of x by eliminating some possible worlds; nothing ensures that this knowledge would also reduce the incompleteness of y. Indeed, it may well be the case that, in the presence of some additional knowledge, x would become more informative than y. Informally the notion of additional knowledge allows us to capture potentially uncertain information that the database model can not capture. For instance, the additional knowledge may constrain the interpretation of some **NULL** to be **EU**.

To make this intuition formal, we define the notion of *universal knowledge* $\mathcal{K}$. Its elements are viewed as pieces of knowledge that can be applied to a database to make it potentially more informative. Such pieces of knowledge can be concatenated: applying $\omega\omega'$ means applying ω first and then applying ω' to the result of applying ω. Finally we have the empty knowledge ϵ: applying it to any database x does not change x. That is, $\mathcal{K}$ is a *monoid* as it has a binary concatenation operation $\omega\omega'$ (which we assume to be associative) and the identity ϵ satisfying $\epsilon\omega = \omega\epsilon = \omega$.

Then, for a database domain $(\mathbb{I}, \mathbb{C}, [\![\cdot]\!])$, the $\mathcal{K}$*-semantics* of an incomplete database $x \in \mathbb{I}$ under knowledge $\omega \in \mathcal{K}$ is the set $[\![x]\!]^\omega$ of complete databases in $\mathbb{C}$ represented by x when ω is known. We assume that, for any given database domain, such a semantic function from $\mathbb{I} \times \mathcal{K}$ to $2^{\mathbb{C}}$ always exists and it is such that $[\![x]\!]^\epsilon = [\![x]\!]$ for every $x \in \mathbb{I}$. Intuitively, the empty knowledge does not give us any extra information about the semantics of incomplete databases.

As $\mathcal{K}$ contains all possible knowledge, it would be reasonable to assume that the $\mathcal{K}$-semantics, while always existing, is not always fully known. Thus, we consider sub-monoids of $\mathcal{K}$, in particular those whose knowledge increases the information content of database objects.

Definition 3.1.1. *A sub-monoid $\mathcal{A}$ of the universal knowledge $\mathcal{K}$ is* additional knowledge *for a database domain $\mathbf{D}$ if*

$$\emptyset \subsetneq [\![x]\!]^{\omega\omega'}_{\mathbf{D}} \subseteq [\![x]\!]^{\omega}_{\mathbf{D}} \tag{3.1}$$

for every $x \in \mathbb{I}_{\mathbf{D}}$ and for every $\omega, \omega' \in \mathcal{A}$.

Intuitively, the information content of an incomplete database can only increase with the additional knowledge in $\mathcal{A}$ and is consistent with it. Note that $\{\epsilon\}$ is additional knowledge for every database domain and corresponds to the situation in which we only have “empty” knowledge.

We can now define the relative informativeness of incomplete databases in the presence of additional knowledge.

Definition 3.1.2. *Let $\mathcal{A}$ be additional knowledge for $\mathbf{D}$, and let $x, y \in \mathbb{I}_{\mathbf{D}}$. We say that x is* less $\mathcal{A}$-informative than *y, and write $x \preceq^{\mathcal{A}}_{\mathbf{D}} y$, if $[\![y]\!]^{\omega}_{\mathbf{D}} \subseteq [\![x]\!]^{\omega}_{\mathbf{D}}$ for every $\omega \in \mathcal{A}$.*

In other words, x is less informative than y under the additional knowledge $\mathcal{A}$ if, for every piece of knowledge $\omega \in \mathcal{A}$, each possible world of y under ω is also a possible world of x under ω. We omit the subscript in $\preceq^{\mathcal{A}}_{\mathbf{D}}$, and simply write $\preceq^{\mathcal{A}}$, when the database domain is clear from the context.

As shown below, additional knowledge conservatively extends the information-based framework of Libkin (2016a): if we do not have any knowledge beyond ϵ, we get the standard information pre-order $\preceq$; but the more we discover – by means of additional knowledge – of the $\mathcal{K}$-semantics of incomplete databases, the less permissive the information pre-order becomes.

Proposition 5. *The following are true:*

(a) $\preceq^{\mathcal{A}}$ is a pre-order;
(b) $\preceq^{\{\epsilon\}}$ is a equivalent to $\preceq$;
(c) If $x \preceq^{\mathcal{A}} y$, then $x \preceq^{\mathcal{B}} y$ for every sub-monoid $\mathcal{B}$ of $\mathcal{A}$.

Proof of Proposition 5.

(a) Let $x, y, z \in \mathbb{I}$ such that $x \preceq^{\mathcal{A}} y$ and $y \preceq^{\mathcal{A}} z$. By definition for every $\omega \in \mathcal{A}$ we have $[\![z]\!]^{\omega}_{\mathbf{D}} \subseteq [\![y]\!]^{\omega}_{\mathbf{D}} \subseteq [\![x]\!]^{\omega}_{\mathbf{D}}$. Hence we have $[\![z]\!]^{\omega}_{\mathbf{D}} \subseteq [\![x]\!]^{\omega}_{\mathbf{D}}$ and $x \preceq^{\mathcal{A}} z$.
(b) Simply recall that for every databases $[\![x]\!]^{\epsilon}_{\mathbf{D}} = [\![x]\!]_{\mathbf{D}}$.
(c) By definition for every $\omega \in \mathcal{A}$ we have $[\![y]\!]^{\omega}_{\mathbf{D}} \subseteq [\![x]\!]^{\omega}_{\mathbf{D}}$. As $\mathcal{B} \subseteq \mathcal{A}$, especially for every $\omega \in \mathcal{B}$ we have $[\![y]\!]^{\omega}_{\mathbf{D}} \subseteq [\![x]\!]^{\omega}_{\mathbf{D}}$, hence $x \preceq^{\mathcal{B}} y$.

□

With this in place, our goal is to capture answers that are consistent with the answers on every possible world under every extra knowledge.

Definition 3.1.3 (Knowledge-preserving certain answer)**.** *Let q be a query from $\mathbf{S}$ to $\mathbf{T}$, let $x \in \mathbb{I}_{\mathbf{S}}$, and let $\mathcal{A}$ be additional knowledge for both $\mathbf{S}$ and $\mathbf{T}$. The* $\mathcal{A}$-preserving certain answer *to q on x, denoted by* $\mathsf{cert}_{\mathcal{A}}(q, x)$*, is the most informative database with respect to $\preceq^{\mathcal{A}}$ that satisfies the following:*

$$q\big([\![x]\!]^{\omega}_{\mathbf{S}}\big) \subseteq [\![\mathsf{cert}_{\mathcal{A}}(q, x)]\!]^{\omega}_{\mathbf{T}} \tag{3.2}$$

for every $\omega \in \mathcal{A}$.

This notion generalizes the information-based certain answer $\mathsf{cert}_{\square}$ and it relies only on the existence of a possible-worlds semantics for the additional knowledge in $\mathcal{A}$.

The $\mathcal{A}$-preserving certain answer can also be interpreted as a kind of "synchronized" greatest lower bound, with respect to $\preceq^{\mathcal{A}}$, of the query answers on the possible worlds of the input:

$$\mathsf{cert}_{\mathcal{A}}(q, x) = \mathsf{glb}_{\preceq^{\mathcal{A}}}\, q(x)$$

where, for every knowledge $\omega \in \mathcal{K}$, the $\mathcal{K}$-semantics of the artifact $q(x)$ is given as $[\![q(x)]\!]^{\omega}_{\mathbf{T}} = q([\![x]\!]^{\omega}_{\mathbf{S}})$.

The expected behavior of query answering under incomplete information is that more informative query inputs yield more informative query outputs. We show that this is indeed the case for certain answers that preserve additional knowledge. Moreover, the information-based certain answers are precisely the certain answers that preserve the empty knowledge.

Proposition 6 (Information preservation)**.** *Let $\mathcal{A}$ be a sub-monoid of $\mathcal{K}$, and let q be a query from $\mathbf{S}$ to $\mathbf{T}$. Then, for all $x, y \in \mathbb{I}_{\mathbf{S}}$ and every $\omega \in \mathcal{A}$, all of the following hold:*

(a) If $\mathsf{cert}_{\mathcal{A}}(q, x)$ *and* $\mathsf{cert}_{\mathcal{A}}(q, y)$ *exist and* $x \preceq^{\mathcal{A}} y$*, then* $\mathsf{cert}_{\mathcal{A}}(q, x) \preceq^{\mathcal{A}} \mathsf{cert}_{\mathcal{A}}(q, y)$*;*

(b) $\mathsf{cert}_{\mathcal{A}}(q, x) \preceq^{\mathcal{B}} \mathsf{cert}_{\mathcal{B}}(q, x)$ *for every $\mathcal{B} \subseteq \mathcal{A}$;*

(c) $\mathsf{cert}_{\square}(q, x) = \mathsf{cert}_{\{\epsilon\}}(q, x)$*.*

Proof of Proposition 6.

(a) By definition, for every $\omega \in \mathcal{A}$ we have $q\big([\![x]\!]^{\omega}_{\mathbf{S}}\big) \subseteq [\![\mathsf{cert}_{\mathcal{A}}(q, x)]\!]^{\omega}_{\mathbf{T}}$. Moreover we know that $x \preceq^{\mathcal{A}} y$ and $[\![y]\!]^{\omega}_{\mathbf{S}} \subseteq [\![x]\!]^{\omega}_{\mathbf{S}}$. Especially for every $\omega \in \mathcal{A}$ we have $q\big([\![y]\!]^{\omega}_{\mathbf{S}}\big) \subseteq [\![\mathsf{cert}_{\mathcal{A}}(q, x)]\!]^{\omega}_{\mathbf{T}}$. And by definition $\mathsf{cert}_{\mathcal{A}}(q, y)$ is the most informative database w.r.t. $\preceq^{A}$ which satisfies this property. Therefore $\mathsf{cert}_{\mathcal{A}}(q, x) \preceq^{\mathcal{A}} \mathsf{cert}_{\mathcal{A}}(q, y)$.

(b) By definition, for every $\omega \in \mathcal{A}$ we have $q\big([\![x]\!]^{\omega}_{\mathbf{S}}\big) \subseteq [\![\mathsf{cert}_{\mathcal{A}}(q, x)]\!]^{\omega}_{\mathbf{T}}$. As $\mathcal{B} \subseteq \mathcal{A}$, especially for every $\omega \in \mathcal{B}$ we have $q\big([\![x]\!]^{\omega}_{\mathbf{S}}\big) \subseteq [\![\mathsf{cert}_{\mathcal{A}}(q, x)]\!]^{\omega}_{\mathbf{T}}$. And by definition $\mathsf{cert}_{\mathcal{B}}(q, y)$ is the most informative database w.r.t. $\preceq^{B}$ which satisfies this property. Therefore $\mathsf{cert}_{\mathcal{A}}(q, x) \preceq^{\mathcal{B}} \mathsf{cert}_{\mathcal{B}}(q, x)$.

(c) Simply recall that the pre-order $\preceq$ and $\preceq^{\{\epsilon\}}$ are equivalent.

□

The above also implies that information-based certain answers are the most informative answers one could expect with empty knowledge; however, they do not preserve non-empty knowledge in general. The following example shows why this is important.

Example 2. Consider the database $d = \{R(\bot_1, 1)\}$ and a query q that returns the first and second columns of R plus an extra column with their sum. The most relevant answer one could expect is $(\bot_1, 1, \bot_1 + 1)$. But the information-based notion $\mathsf{cert}_\Box(q, \cdot)$ gives us a tuple of the form $(\bot_i, 1, \bot_j)$, where $\bot_i$ and $\bot_j$ are fresh nulls. However, since null labels are not preserved, there is no relationship between $\bot_i$ and $\bot_j$ in the output, and $\bot_1$ in the input. In particular, it is not enforced that the value $\bot_i$ in the output and the value $\bot_1$ in the input must be equal (and that $\bot_j$ is equal to $\bot_1 + 1$).
Now consider the extra piece of knowledge ω encoding the fact that the value of $\bot_j$ is 0. Clearly, the information content of the original database d is not increased by this new knowledge, since $\bot_j \notin \mathsf{Null}(d)$. But $\bot_j$ occurs in the information-based certain answer $(\bot_i, 1, \bot_j)$, whose possible worlds under ω always contain the tuple $(\bot_i, 1, 0)$. Thus, the information-based certain answer under additional knowledge is not less informative than all possible answers under the same knowledge, and it may contain false tuples.

What this means is that information-based certain answers become irrelevant once we discover additional knowledge: they must be recomputed on a more informative database, if it exists. Knowledge-preserving certain answers do not have this shortcoming, as we show next.

Theorem 4. *Let q be a query from $\mathbf{S}$ to $\mathbf{T}$, let $x \in \mathbb{I}_\mathbf{S}$, and let $\mathcal{A}$ be additional knowledge for both $\mathbf{S}$ and $\mathbf{T}$. If $\mathsf{cert}_\mathcal{A}(q, x)$ exists, then*

$$q([\![x]\!]^\omega_\mathbf{S}) \subseteq [\![\mathsf{cert}_\mathcal{A}(q, x)]\!]^\omega_\mathbf{T}$$

for every $\omega \in \mathcal{A}$. Moreover, there exist a query q, database $x \in \mathbb{I}_\mathbf{S}$ and $w \in \mathcal{A}$ such that:

(a) $q([\![x]\!]^\omega_\mathbf{S}) \not\subseteq [\![\mathsf{cert}_\Box(q, x)]\!]^\omega_\mathbf{T}$*, and*
(b) $[\![\mathsf{cert}_\Box(q, x)]\!]_\mathbf{T} \subseteq [\![\mathsf{cert}_\mathcal{A}(q, x)]\!]^\omega_\mathbf{T}$.

Informally, knowledge-preserving certain answers remain consistent when new information is discovered. Moreover, there are queries for which this notion gives more informative answers under additional knowledge.

Proof of Theorem 4. For every $\omega \in \mathcal{A}$ we have that $[\![\mathsf{cert}_\mathcal{A}(q, x)]\!]^w_\mathbf{T} \subseteq q([\![x]\!]^\omega_\mathbf{S})$ is immediate from the definition of $\mathsf{cert}_\mathcal{A}(q, x)$.

An example of a query and a database such that $[\![\mathsf{cert}_\Box(q, x)]\!]^\omega_\mathbf{T} \not\subseteq q([\![x]\!]^\omega_\mathbf{S})$ is given in Example 2. And we obtain $[\![\mathsf{cert}_\mathcal{A}(q, x)]\!]^\omega_\mathbf{T} \supseteq [\![\mathsf{cert}_\Box(q, x)]\!]_\mathbf{T}$ as soon as $\mathsf{cert}_\mathcal{A}(q, x)$ contains some marked-null values. □

Next, we apply our abstract framework to relational databases, under the set semantics. We do so in a setting where additional knowledge is given by partial valuations of nulls. We then show that the well-known certain answers with nulls corresponds to the valuation-preserving certain answers.

3.2 Certainty in Relational Databases

Recall that an incomplete relational database is a finite set of tables populated by constants and nulls, and its semantics of incompleteness is given by instantiating nulls with constants by means of valuations.

Definition 3.2.1. *The monoid of partial valuations $\mathcal{V} \subseteq \mathcal{K}$ consists of all partial functions from* Null *to* Const*. Moreover, for every $v, v' \in \mathcal{V}$, the $\mathcal{K}$-semantics of incomplete relational databases is $[\![d]\!]^{vv'} = [\![v'(v(d))]\!]$.*

We immediately have that the knowledge encoded by $\mathcal{V}$ is additional to the relational database domain for both closed and open world semantics. Below we use the notion of valuation-preserving certain answers to justify the use of certain answers with nulls.

Proposition 7. *Let $\mathbf{S}$ be the set-relational databases domain with either* OWA *or* CWA *semantics and $\mathbf{T}$ be the set-relational databases domain with* OWA *semantics. Then, for every query q from $\mathbf{S}$ to $\mathbf{T}$ and for every $d \in \mathbb{I}_{\mathbf{S}}$*

(a) either both $\mathsf{cert}_{\mathcal{V}}(q, d)$ *and* $\mathsf{cert}_{\perp}(q, d)$ *exist and coincide, or neither of them exists.*
(b) for every database d' we have $d' \subseteq \mathsf{cert}_{\perp}(q, d)$ iff. $d' \preceq^{\mathcal{V}} \mathsf{cert}_{\mathcal{V}}(q, d)$

Proof. This proposition is a special case of proposition 9 for which a proof is given. □

We also justify the intersection-based certain answer:

Proposition 8. *Let $\mathbf{S}$ be the set-relational databases domain with either* OWA *or* CWA *semantics and $\mathbf{T}$ be the set-relational databases domain with* OWA *semantics and without null. Let $\mathcal{A}$ an additional knowledge $\mathbf{S}$ and $\mathbf{T}$, then, for every query q from $\mathbf{S}$ to $\mathbf{T}$ and for every $d \in \mathbb{I}_{\mathbf{S}}$, either both* $\mathsf{cert}_{\mathcal{A}}(q, d)$ *and* $\mathsf{cert}_{\cap}(q, d)$ *exist and coincide, or neither of them exists.*

The intersection-based certain answer is the maximal knowledge-preserving answer containing only constant tuples.

Proof of Proposition 8. By proposition 6 we have that $\mathsf{cert}_{\mathcal{A}}(q, d) \preceq^{\{\epsilon\}} \mathsf{cert}_{\{\epsilon\}}(q, d) = \mathsf{cert}_{\Box}(q, d)$ and by proposition 3, for databases without nulls as target we have $\mathsf{cert}_{\Box}(q, d) = \mathsf{cert}_{\cap}(q, d)$. Therefore we have $\mathsf{cert}_{\mathcal{A}}(q, d) \preceq^{\{\epsilon\}} \mathsf{cert}_{\cap}(q, d)$ hence $[\![\mathsf{cert}_{\mathcal{A}}(q, d)]\!]_{\mathsf{OWA}} \subseteq [\![\mathsf{cert}_{\cap}(q, d)]\!]_{\mathsf{OWA}}$. Recall that $\mathsf{cert}_{\mathcal{A}}(q, d)$ and $\mathsf{cert}_{\cap}(q, d)$ are complete databases, therefore we obtain $\mathsf{cert}_{\mathcal{A}}(q, d) \subseteq \mathsf{cert}_{\cap}(q, d)$

$\mathsf{cert}_\cap(q,d) = \bigcap \{q(d') \mid d' \in [\![d]\!]_{\mathbf{S}}\}$, therefore for every $d' \in [\![d]\!]_{\mathbf{S}}$, $\mathsf{cert}_\cap(q,d) \subseteq q(d')$. Recall that by definition of Additional knowledge we have that for every $\omega \in \mathcal{A}$, $[\![d]\!]^{\omega}_{\mathbf{S}} \subseteq [\![d]\!]_{\mathbf{S}}$, therefore for every $\omega \in \mathcal{A}$, for every $d' \in [\![d]\!]^{\omega}_{\mathbf{S}}$ we have $\mathsf{cert}_\cap(q,d) \subseteq q(d')$. Especially we have, $\mathsf{cert}_\cap(q,d) \subseteq q([\![d]\!]^{\omega}_{\mathbf{S}})$ for every $\omega \in \mathcal{A}$. Moreover by definition of the knowledge preserving certain answer, we have $q([\![d]\!]^{\omega}_{\mathbf{S}}) \subseteq [\![\mathsf{cert}_{\mathcal{A}}(q,d)]\!]^{\omega}_{\mathbf{T}}$. Now recall that $\mathsf{cert}_{\mathcal{A}}(q,d)$ is a complete database, therefore we obtain $q([\![d]\!]^{\omega}_{\mathbf{S}}) \subseteq \mathsf{cert}_{\mathcal{A}}(q,d)$ and we conclude that $\mathsf{cert}_\cap(q,d) \subseteq \mathsf{cert}_{\mathcal{A}}(q,d)$. □

3.2.1 Bag Relational Databases

The model of bag relational database is closely related to set relational database, here we present the differences. A bag relational database schema is a finite set of table names with associated arities, and a k-ary table is a finite bag (a.k.a. multiset) of k-tuples over Const∪Null. The number of occurrences of a tuple $\bar{t}$ in a table T is denoted by $\#\big(R,\bar{t}\big)$. Then, a bag relational database d over a given schema maps each table name R in the schema to a table R^d of appropriate arity.

For convenience of notation, we sometimes represent a bag relational database as a bag of facts; e.g., $d = \{\!\{R(1,\bot_1), S(\bot_1,2), R(1,\bot_1)\}\!\}$ is the database d such that $R^d = \{\!\{(1,\bot_1),(1,\bot_1)\}\!\}$ and $S^d = \{\!\{(\bot_1,2))\}\!\}$.

Relational algebra for Bag Relational Databases

The relational algebra language on the bag relational model denoted Bag-RA has the following syntax:

$$\begin{aligned} q := R \mid \pi_{\bar{\alpha}}(q) \mid \sigma_{\alpha_i=\alpha_j}(q) \mid \sigma_{\alpha_i\neq\alpha_j}(q) \\ \mid q \times q \mid q \uplus q \mid q \cup q \mid q \cap q \mid q - q \end{aligned}$$

where α_i and α_j are attributes, and $\bar{\alpha}$ is possibly empty tuple of attributes.

The semantics of a (well-formed) Bag-RA query q is given by inductively defining the quantity $\#\big(q(d), \bar{t}\big)$, which is the number of occurrences of a tuple $\bar{t}$ (of appropriate arity) in the result of applying q to a database d. This is done as follows:

$$
\begin{aligned}
\#\big(R(d), \bar{t}\big) &= \#\big(R^d, \bar{t}\big) \\
\#\big(\pi_{\bar{\alpha}}(q)(d), \bar{t}\big) &= \sum_{\bar{t}' \,:\, \pi_{\bar{\alpha}}(\bar{t}') = \bar{t}} \#\big(q(d), \bar{t}'\big) \\
\#\big(\sigma_{\alpha_i = \alpha_j}(q)(d), \bar{t}\big) &= \begin{cases} \#\big(q(d), \bar{t}\big) & \text{if } \bar{t}[\alpha_i] \overset{\text{Const}}{=} \bar{t}[\alpha_j] \\ 0 & \text{otherwise} \end{cases} \\
\#\big(\sigma_{\alpha_i \neq \alpha_j}(q)(d), \bar{t}\big) &= \begin{cases} \#\big(q(d), \bar{t}\big) & \text{if } \bar{t}[\alpha_i] \overset{\text{Const}}{\neq} \bar{t}[\alpha_j] \\ 0 & \text{otherwise} \end{cases} \\
\#\big((q \times q')(d), \bar{t}\,\bar{t}'\big) &= \#\big(q(d), \bar{t}\big) \cdot \#\big(q'(d), \bar{t}'\big) \\
\#\big((q \uplus q')(d), \bar{t}\big) &= \#\big(q(d), \bar{t}\big) + \#\big(q'(d), \bar{t}\big) \\
\#\big((q \cup q')(d), \bar{t}\big) &= \max\{\#\big(q(d), \bar{t}\big), \#\big(q'(d), \bar{t}\big)\} \\
\#\big((q \cap q')(d), \bar{t}\big) &= \min\{\#\big(q(d), \bar{t}\big), \#\big(q'(d), \bar{t}\big)\} \\
\#\big((q - q')(d), \bar{t}\big) &= \max\{\#\big(q(d), \bar{t}\big) - \#\big(q'(d), \bar{t}\big), 0\}
\end{aligned}
$$

where $\bar{t}[\alpha_i]$ denotes the α_i element of $\bar{t}$, $\pi_{\alpha_1, \cdots, \alpha_i}(\bar{t})$ is the tuple $(\bar{t}[\alpha_1], \cdots, \bar{t}[\alpha_i])$, and the tuples $\bar{t}$ and $\bar{t}'$ in the rule for $q \times q'$ have the same arity as $q(d)$ and $q'(d)$, respectively.

The most common restriction of bag relational algebra is called *union of conjunctive queries* denoted UCQ and has the following syntax:

$$
\begin{aligned}
q := R \mid \pi_{\bar{\alpha}}(q) \mid \sigma_{\alpha_i = \alpha_j}(q) \\
\mid q \times q \mid q \uplus q
\end{aligned}
$$

Incompleteness in Bag Relational Databases

The semantics of incompleteness for bag relational database are also defined by the mean of valuation such that:

(a) $[\![d]\!]_{\mathsf{CWA}} = \{v(d) \mid v \text{ is a } d\textit{-complete} \text{ valuation}\}$.
(b) $[\![d]\!]_{\mathsf{OWA}} = \{\text{complete } d' \mid v(d) \subseteq d' \text{ for a } d\textit{-complete} \text{ valuation } v\}$.

The notion of certain answers with null can also be extended to the bag model in the following way:

Definition 3.2.2. *The* certain answers with nulls *to a query q on a bag relational database d is the table* $\mathsf{cert}_\perp(q,d)$ *such that, for every d-complete valuation v and for every $c \in [\![v(d)]\!]_{\mathbf{S}}$, the multiplicity of each tuple in $v\big(\mathsf{cert}_\perp(q,d)\big)$ is less than or equal to its multiplicity in $q(c)$:*

$$\#\big(\mathsf{cert}_\perp(q,d),\bar{t}\big) \;= \min\big\{\,\#\big(q(c),v(\bar{t})\big) \mid c \in [\![v(d)]\!]_{\mathbf{S}},\; v \textit{ is a } d\textit{-complete valuation}\big\}$$

However, so far we have not explained how a valuation v is applied to a bag relational database d. When tables T are constrained to be sets, there is no ambiguity on how to construct $v(T)$. But when they are bags, there are several possibilities, each leading to different semantics Hernich and Kolaitis (2017).

3.2.2 Collapsing and Additive Semantics

The first construction we study prescribes that, when a valuation is applied to a table, distinct tuples that become equal (i.e., unify) under the valuation are "collapsed" together, so that the maximum number of occurrences of each such tuple appears in the result.

Definition 3.2.3. *The* collapsing application *of a valuation v on a table T is the table $v(T)$ such that for every tuple $\bar{t}$*

$$\#\big(v(T),\bar{t}\big) = \max\big\{\,\#\big(T,\bar{u}\big) \mid v(\bar{u}) = \bar{t}\big\}$$

For example, consider a table T given by the following bag:

$$\{\!|(1,2),(\perp_1,2),(1,\perp_2),(1,\perp_2)|\!\} \tag{3.3}$$

and a valuation v such that $v(\perp_i) = i$ for every i. Then, the collapsing application of v to T is the table $v(T)$ given by the bag $\{\!|(1,2),(1,2)|\!\}$.

Intuitively, when valuations are applied in such a way, incomplete facts only represent new information if they do not unify with already existing data. Below we show that, under this semantics of valuations, the certain answers with nulls coincide with the valuation-preserving ones.

Proposition 9. *Let $\mathbf{S}$ be the bag-relational databases domain under the collapsing application of valuations with either* OWA *or* CWA *semantics and $\mathbf{T}$ be the bag-relational databases domain under the collapsing application of valuations with* OWA *semantics. Then, for every query q from $\mathbf{S}$ to $\mathbf{T}$ and for every $d \in \mathbb{I}_{\mathbf{S}}$, either both* $\mathsf{cert}_{\mathcal{V}}(q,d)$ *and* $\mathsf{cert}_\perp(q,d)$ *exist and coincide, or neither of them exists.*

We remark that the above result holds for databases under the bag data model, but also – in particular – when tables are constrained to be sets.

Proof. Due to its length and for sake of readability, the proof is given at the end of the chapter 3.2.4. □

Since the collapsing semantics negates the importance of a tuple's identity by disregarding – to some extent – its multiplicity, the most commonly used semantics is instead one where the multiplicities of tuples that become equal under a valuation are added up in the result.

Definition 3.2.4. *The* additive application *of a valuation v on a table T is the table $v(T)$ such that for every tuple $\bar{t}$*

$$\#\big(v(T), \bar{t}\big) = \sum_{\bar{u}\colon\, v(\bar{u})=\bar{t}} \#\big(T, \bar{u}\big)$$

To see the difference with the collapsing semantics, consider again the table T given by the bag in (3.3), and the same valuation v such that $\bot_i \mapsto i$. The additive application of v to T results in a table $v(T)$ consisting of 4 (as opposed to 2) occurrences of the tuple $(1, 2)$. When valuations are applied in such an additive way, incomplete facts always represent new information even if they unify with already existing data, in contrast with the collapsing semantics discussed earlier.

However, while more natural, this semantics of valuations leads to problems in terms of certain answers, as shown below.

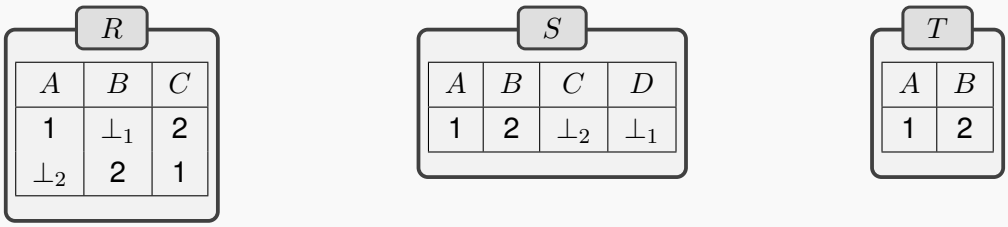
Theorem 5. *Let $\mathbf{S}$ be the bag-relational databases domain under the additive application of valuations with either* OWA *or* CWA *semantics and $\mathbf{T}$ be the bag-relational databases domain under the additive application of valuations with* OWA *semantics. There exists a union of conjunctive queries with negation such that:*

(a) *the valuation-preserving certain answer does not exist;*
(b) *the $\{\epsilon\}$-preserving certain answer does not exist.*
(c) *the certain answer with nulls is not less informative than every possible answer.*

Proof. Due to its length and for sake of readability, the proof is given at the end of the chapter 3.2.4. □

Below we give an example in which the certain answer with nulls to a query q on a database d is not less informative than the answer to q on some possible world of d.

Example 3. Consider the following database D:



R

A	B	C
1	$\perp_1$	2
$\perp_2$	2	1

S

A	B	C	D
1	2	$\perp_2$	$\perp_1$

T

A	B
1	2

and the following relational algebra query q:

$$\pi_{A,B}\big(\sigma_{A\neq C\wedge B\neq C}(R)\big) \uplus \pi_{A,B}\big(\sigma_{A=C\wedge B=D}(S)\big) \uplus T$$

Then, $\mathsf{cert}_{\perp}(q, D)$ is the bag $\{\!\{(1,2),(1,\perp_1),(\perp_2,2)\}\!\}$. If we take a valuation v such that $v(\perp_1) = 2$ and $v(\perp_2) = 1$, then $\{\!\{(1,2);(1,2)\}\!\}$ belongs to $q\big([\![v(D)]\!]\big)$. But with the additive semantics $[\![\mathsf{cert}_{\perp}(q, D)]\!]$ contains at least three tuples and so it cannot be less informative than $q\big(v(D)\big)$.
The answers $A_1 = \{\!\{(1,2)\}\!\}$ and $A_2 = \{\!\{(1,\perp_i),(\perp_k,2)\}\!\}$ are both less informative than every element of $q([\![d]\!])$. Thus, $\mathsf{cert}_{\square}(q, D)$ should be more informative than both A_1 and A_2. We clearly have that $A_1 \not\preceq A_2$ and $A_2 \not\preceq A_1$; moreover, it is easy to check that we cannot increase the information content of A_1 or A_2 and still be less informative than the answer to q on every possible world described by D. Therefore, the information-based certain answer does not exist.

The above result tells us that, under additive valuations, the notion of certain answers with nulls is not well-founded, as even for simple queries they may not be less informative than the query answers on every possible world. Thus, some fact in the certain answers with nulls might be false.

Moreover, Example 2 showed that the information-based certain answer can be strictly less informative than the certain answer with nulls, and Proposition 6 states that, for every knowledge, the knowledge-preserving certain answer is no more informative than the information-based certain answer. Therefore, there is no additional knowledge such that knowledge-preserving certain answers captures certain answers with nulls. We argue that this is due to the additive semantics itself.

To see this, consider two incomplete relational databases d_1 and d_2 of the same schema. Then, under OWA and collapsing valuations, we can always build a database d_3 such that $[\![d_3]\!] = [\![d_1]\!] \cap [\![d_2]\!] = [\![d_1 \cup d_2]\!]$, where $\cup$ denotes the union-max operator: $\#\big((T_1 \cup T_2), \bar{t}\big) = \max\big(\#\big(T_1, \bar{t}\big), \#\big(T_2, \bar{t}\big)\big)$. This ensures that, if we have a finite number of possible answers to a query, then the certain answers always exist. However,

under additive valuations, we lose this property: as shown in Example 3, there exist databases d_1 and d_2 such that, for every database d_3, we have $[\![d_3]\!] \neq [\![d_1]\!] \cap [\![d_2]\!]$. This can be interpreted as a mismatch between the semantics and the set of incomplete objects.: there are not enough objects to capture the complexity of the semantics.

Now that we have identified the problem, we can finally capture the notion of certain answers with nulls by changing the semantics of target. To do this under additive valuations, we need to interpret query answers under the collapsing semantics.

Proposition 10. *Let* $\mathbf{S}$ *be the bag-relational databases domain under the additive application of valuations with either* OWA *or* CWA *semantics and* $\mathbf{T}$ *be the bag-relational databases domain under the collapsing application of valuations with* OWA *semantics. Then, for every query* q *from* $\mathbf{S}$ *to* $\mathbf{T}$ *and for every database* $d \in \mathbb{I}_{\mathbf{S}}$*, either both* $\mathrm{cert}_{\mathcal{V}}(q, d)$ *and* $\mathrm{cert}_{\perp}(q, d)$ *exist and coincide, or neither of them exists.*

Proof. Due to its length and for sake of readability, the proof is given at the end of the chapter 3.2.4. □

We have been able to justify the notion of certain answers with nulls for both the additive and the collapsing semantics of valuations, but neither is entirely satisfactory. On the one hand, collapsing valuations decreases the expressiveness of the relational data model based on bags. On the other hand, certain answers with nulls under additive valuations must be interpreted using the collapsing semantics on target, which is counter-intuitive especially on top of the open world assumption.

We will now propose a new semantics of incompleteness for relational databases that overcomes these shortcomings.

3.2.3 Mixed Semantics

The main idea behind our proposed semantics is that applying a valuation to a table does not produce *one* table, as is the case with the collapsing and additive semantics, but rather a *set* of tables.

Definition 3.2.5. *The* mixed application *of a valuation* v *to a* k*-ary table* T *is the set* $v(T)$ *consisting of all tables* B *of arity* k *such that, for every* k*-ary tuple* $\bar{t}$*:*

$$\max\{\, \#(T, \bar{u}) \mid v(\bar{u}) = \bar{t} \,\} \leq \#(B, \bar{t}) \leq \sum_{\bar{u}\colon v(\bar{u})=\bar{t}} \#(T, \bar{u})$$

This extends to databases d *as follows:* $v(d)$ *is the set of all databases* d_1 *of the same schema as* d *such that* R^{d_1} *belongs to* $v(R^d)$ *for every table name* R*.*

Finally, the mixed semantics of incompleteness under OWA and CWA are given as follows:

$$[\![d]\!]_{\mathsf{CWA}} = \{d_1 \text{ complete} \mid d_1 \in v(d),\ v \text{ is a valuation}\}$$
$$[\![d]\!]_{\mathsf{OWA}} = \{d_2 \text{ complete} \mid d_2 \supseteq d_1 \in v(d),\ v \text{ is a valuation}\}$$

As the name suggests, the mixed semantics combines the collapsing and the additive ones, by taking into account that incomplete facts unifying with already existing data may or may not represent new information.

Below we show that, when using the mixed semantics of valuations on both the source and target domain of queries, the valuation-preserving certain answers and the certain answers with nulls coincide.

Theorem 6. *Let* $\mathbf{S}$ *be the bag-relational databases domain under the mixed application of valuations with either* OWA *or* CWA *semantics and* $\mathbf{T}$ *be the bag-relational databases domain under the mixed application of valuations with* OWA *semantics. Then, for every query* q *from* $\mathbf{S}$ *to* $\mathbf{T}$ *and for every database* $d \in \mathbb{I}_{\mathbf{S}}$*, either both* $\mathsf{cert}_{\mathcal{V}}(q,d)$ *and* $\mathsf{cert}_{\perp}(q,d)$ *exist and coincide, or neither of them exists.*

Proof. Due to its length and for sake of readability, the proof is given at the end of the chapter 3.2.4. □

Note that when we consider the open world assumption on $\mathbf{S}$, the collapsing and mixed semantics are equivalent. And for every valuation from $\mathcal{V}$ their $\mathcal{K}$-semantics are also the same. However, as illustrated in Example 4, there exists a query such that valuation-preserving certain answers upon additive, collapsing and mixed semantics are different for closed-world semantics.

Example 4. Let d be a relational database such that

$$R^d = \{\!\{1, \perp_1, \perp_2\}\!\}\,;\quad S^d = \{\!\{1, 1\}\!\}\,;$$

and consider the following relational algebra query q:

$$\pi_{\emptyset}\big((R - S) \uplus (S - R)\big)$$

It is easy to see that the certain answers of q are empty iff there exists a possible world c of d such that $R^c = S^c$. Since S^d does not contain nulls, we focus on R.
Under the additive semantics of valuations, for every possible world c of d there are at least 3 tuples in R^c. As there are always exactly 2 tuples in S^c, we have that $R^c - S^c$ cannot be empty. Thus, the valuation-preserving certain answer to q on d is non-empty.

Under the collapsing semantics of valuations, $\{\!\{1,1\}\!\} \neq v(R^d)$ for every valuation v because, even if $v(\bot_1) = 1$ or $v(\bot_2) = 1$, they collapse into a single occurrence of value 1. Hence, for every possible world c of d, we have that $S^c - R^c$ cannot be empty, and therefore the valuation-preserving certain answer to q on d is non-empty.
Under the mixed semantics of valuations, we consider every possible multiplicity for each tuple, and $\{\!\{1,1\}\!\} \in v(R^d)$ when $v(\bot_1) = v(\bot_2) = 1$. Thus, the empty bag is a possible answer to q, and the valuation-preserving certain answer to q on d is empty.

In light of the above, we argue that the semantics of valuations one should use in the context of relational databases is the mixed semantics, because it behaves consistently independently of whether a (relational) database domain is the source or the target of queries. However, under mixed semantics, the valuation-preserving certain answers coincide with the certain answers with nulls, and these are still unsatisfactory as they cannot produce answers involving values that were not present in the original data, as Example 2 illustrates.

3.2.4 Proof of section 3.2: Certainty in relational databases

The proof for bag relational database domain are a bit more involved and require some domain specific lemma:

Lemma 1. *For every relational database $d, d' \in \mathbb{I}$ we have*

(a) *with open world assumption the valuation-information pre-order is equivalent to the subset pre-order ie. $d \preceq^{\mathcal{V}}_{\mathsf{OWA}} d' \Leftrightarrow d \subseteq d'$.*

(b) *with closed world assumption the valuation-information pre-order is equivalent to the subset pre-order ie. $d \preceq^{\mathcal{V}}_{\mathsf{CWA}} d' \Leftrightarrow d = d'$.*

Proof.

Definition 3.2.6. *We say that a d-complete valuation is naive, if for every $\bot_i, \bot_j \in$ Null(d) we have $v(\bot_i) \neq v(\bot_j)$ and $v(\bot_i), v(\bot_j) \notin$ Const(d) where* Const(d) *the set of constant value appearing in d. As d is finite and* Const *is infinite, such a valuation always exists.*

The $(\Leftarrow)$ is trivial for each pre-order.

We want to prove $d \preceq^{\mathcal{V}}_{\mathsf{OWA}} d' \Rightarrow d \subseteq d'$. By contradiction assume that $d \not\subseteq d'$ then there exists a tuple $\bar{t}$ and a relation R such that $\#\big(R^d, \bar{t}\big) > \#\big(R^{d'}, \bar{t}\big)$. By definition of a $d \uplus d'$-complete naive valuation v, we know that $\bar{t}$ does not unify with any tuple of R^d and $R^{d'}$ after the application of v. Therefore we have $\#\big(v(R^d), v(\bar{t})\big) > \#\big(v(R^{d'}), v(\bar{t})\big)$, hence $v(d') \notin [\![v(d)]\!]$ and $[\![v(d')]\!]_{\mathsf{OWA}} \not\subseteq [\![v(d)]\!]_{\mathsf{OWA}}$. Finally we have $[\![d']\!]^v_{\mathsf{OWA}} \not\subseteq [\![d]\!]^v_{\mathsf{OWA}}$ and we conclude that $d \not\preceq^{\mathcal{V}}_{\mathsf{OWA}} d'$.

The proof of $d \preceq^{\mathcal{V}}_{\mathsf{CWA}} d' \Rightarrow (d = d')$ is similar as neither $d \not\subseteq d'$ or $d' \not\subseteq d$ and we do not use property of the OWA semantic in the proof. □

Lemma 2. *For every relational database d we have*

$$[\![d]\!] = \bigcup_{v \text{ a } d\text{-complete valuation}} [\![d]\!]^v \tag{3.4}$$

Proof. Recall that

$$[\![d]\!]_{\mathsf{OWA}} = \{d' \text{ complete} \mid \exists \text{ valuation } v \text{ s.t. } v(d) \subseteq d'\}$$
$$[\![d]\!]_{\mathsf{CWA}} = \{d' \text{ complete} \mid \exists \text{ valuation } v \text{ s.t. } v(d) = d'\}$$

In both definition d' is a complete database, hence $v(d)$ is a complete database and v is a d-complete valuation. Moreover by lemma 1 the inclusion and equality can be rewritten as $d' \in [\![v(d)]\!] = [\![d]\!]^v$. □

Lemma 3. *For every relational databases $d, d' \in \mathbb{I}$ if for all $\big(d \uplus d'\big)$-complete valuation v we have $[\![d']\!]^v \subseteq [\![d]\!]^v$ then $d \preceq^{\mathcal{V}} d'$.*

Proof. Let a partial valuation v, by lemma 2 we have that:

$$[\![v(d)]\!] = [\![d]\!]^v = \bigcup_{v' \text{ a } v(d)\text{-complete valuation}} [\![v'(v(d))]\!] \tag{3.5}$$

And by hypothesis for every v' a $\big(v(d) \uplus v(d')\big)$-complete valuation we have $[\![v'(v(d'))]\!] \subseteq [\![v'(v(d))]\!]$. Therefore for every valuation v we have $[\![v(d')]\!] \subseteq [\![v(d)]\!]$. □

Proof of Proposition 9

This proof is a bit more involved than one might expect. It mostly rely on the fact that for collapsing semantic with the open world assumption we have the following property, which essentially says that we have enough incomplete objects to capture the semantics.

Lemma 4. *For every databases $A_1, A_2 \in \mathbb{I}_{\mathbf{T}}$ and every valuation v with the collapsing application and the* OWA *semantics we have $[\![A_1]\!]^v_{\mathsf{OWA}} \cap [\![A_2]\!]^v_{\mathsf{OWA}} = [\![\big(A_1 \cup A_2\big)]\!]^v_{\mathsf{OWA}}$.*

Proof. Let $A_1, A_2 \in \mathbb{I}_{\mathbf{T}}$, we have $A_1 \subseteq \big(A_1 \cup A_2\big)$ and $A_2 \subseteq \big(A_1 \cup A_2\big)$. Therefore by lemma 1, with OWA semantics, for every valuation v we have $[\![\big(A_1 \cup A_2\big)]\!]^v_{\mathsf{OWA}} \subseteq [\![A_1]\!]^v_{\mathsf{OWA}}$ and $[\![\big(A_1 \cup A_2\big)]\!]^v_{\mathsf{OWA}} \subseteq [\![A_2]\!]^v_{\mathsf{OWA}}$. Hence $[\![\big(A_1 \cup A_2\big)]\!]^v_{\mathsf{OWA}} \subseteq \big([\![A_1]\!]^v_{\mathsf{OWA}} \cap [\![A_2]\!]^v_{\mathsf{OWA}}\big)$.

Let a $\big(A_1 \uplus A_2\big)$-complete valuation v, and a complete database $C \in \mathbb{C}_{\mathbf{T}}$ such that $C \in [\![v(A_1)]\!]_{\mathsf{OWA}}$ and $C \in [\![v(A_2)]\!]_{\mathsf{OWA}}$. As C is complete, by lemma 1, we have $v(A_1) \subseteq C$ and $v(A_2) \subseteq C$. Moreover, with the collapsing semantics, for every tuple $\bar{t}$ we have $\#\big(v(A_1 \cup A_2), \bar{t}\big) \leq \max\big(\#\big(v(A_1), \bar{t}\big); \#\big(v(A_2), \bar{t}\big)\big)$, therefore $\#\big(v(A_1 \cup A_2), \bar{t}\big) \leq \#\big(C, \bar{t}\big)$

and $v(A_1 \cup A_2) \subseteq C$. Hence we have $C \in [\![v(A_1 \cup A_2)]\!]_{\mathsf{OWA}}$ and $C \in [\![(A_1 \cup A_2)]\!]^v_{\mathsf{OWA}}$. So we can conclude $[\![A_1]\!]^v_{\mathsf{OWA}} \cap [\![A_2]\!]^v_{\mathsf{OWA}} \subseteq [\![(A_1 \cup A_2)]\!]^v_{\mathsf{OWA}}$ for every $(A_1 \uplus A_2)$-complete valuation v. And by lemma 3 we have $[\![A_1]\!]^v_{\mathsf{OWA}} \cap [\![A_2]\!]^v_{\mathsf{OWA}} \subseteq [\![(A_1 \cup A_2)]\!]^v_{\mathsf{OWA}}$ for every valuations. □

Now that we can compose bag with union-max operator and compute its semantics, we can prove the proposition:

Proposition. *Let* $\mathbf{S}$ *and* $\mathbf{T}$ *be relational database domains under the collapsing application of valuations. Then, for every query* q *from* $\mathbf{S}$ *to* $\mathbf{T}$ *and for every* $d \in \mathbb{I}_{\mathbf{S}}$*, either both* $\mathsf{cert}_{\mathcal{V}}(q,d)$ *and* $\mathsf{cert}_{\perp}(q,d)$ *exist and coincide, or neither of them exists.*

Proof. For every tuple $\bar{t}$ and every bag B we denote $B_{\bar{t}}$ the bag such that $\#(B_{\bar{t}}, \bar{t}) = \#(B, \bar{t})$ and $\#(B_{\bar{t}}, \bar{t}') = 0$ if $\bar{t} \neq \bar{t}'$. Then by definition we have

$$B = \bigcup_{\bar{t} \in \mathcal{T}^*} B_{\bar{t}} \tag{3.6}$$

with $\cup$ the union-max operation for bags.

By definition of $\mathsf{cert}_{\perp}(q,d)$ for every tuple $\bar{t}$ and every complete valuation v we have $q([\![x]\!]^v) \subseteq [\![\mathsf{cert}_{\perp}(q,d)_{\bar{t}}]\!]^v$. Therefore we have $\mathsf{cert}_{\perp}(q,d)_{\bar{t}} \preceq^{\mathcal{V}} q(d)$. Hence by lemma 4 we have $\mathsf{cert}_{\perp}(q,d) \preceq^{\mathcal{V}} q(d)$. And by definition of valuation preserving certain answers as greatest lower bound, we can conclude that $\mathsf{cert}_{\perp}(q,d) \preceq^{\mathcal{V}} \mathsf{cert}_{\mathcal{V}}(q,d)$.

To prove that $\mathsf{cert}_{\mathcal{V}}(q,d) \preceq^{\mathcal{V}} \mathsf{cert}_{\perp}(q,d)$, by contradiction we assume there exists a tuple $\bar{t}$ such that $\mathsf{cert}_{\mathcal{V}}(q,d)_{\bar{t}} \not\subseteq \mathsf{cert}_{\perp}(q,d)_{\bar{t}}$. Hence by definition of $\mathsf{cert}_{\mathcal{V}}(q,d)_{\bar{t}}$ we have $\mathsf{cert}_{\mathcal{V}}(q,d)_{\bar{t}} \preceq^{A} \mathsf{cert}_{\mathcal{V}}(q,d)$ and $\mathsf{cert}_{\mathcal{V}}(q,d)_{\bar{t}} \preceq^{\mathcal{V}} q(x)$. Especially for all d-complete valuation v, $v(\mathsf{cert}_{\mathcal{V}}(q,d)_{\bar{t}}) \subseteq q(v(d))$. And if $\mathsf{cert}_{\mathcal{V}}(q,d)_{\bar{t}} \not\subseteq \mathsf{cert}_{\perp}(q,d)_{\bar{t}}$ it contradicts the maximality definition of certain answers with nulls. □

Proof of Theorem 5

The example 3 gives such query and therefore a proof for the Theorem.

Proof of Proposition 10 and Theorem 6

The proof are similar to the proof of Proposition 9. Indeed notice that as mixed semantic and collapsing semantic are the same with the open assumption, lemma 4 is applicable. And the rest of the proof apply disregarding the semantic of the source as soon at the valuation monoid is additional to it.

3.3 Certainty for Value-Inventing Queries

We would like to capture all the information-based certain answers, while keeping the valuation-preservation property of the certain answers with nulls. By Proposition 6, we know that with the relational database domain as target it is impossible to be valuation-preserving and more informative at the same time. Therefore, we must modify the target domain. To handle queries whose answers may contain values not appearing in the original input, we extend relational databases with the notion of *persistent nulls*. Informally, these are information placeholders with the same semantics as marked nulls, but their incompleteness does not decrease in the presence of new knowledge: every interpretation of a persistent null is consistent with any additional knowledge.

Relational database domains with persistent nulls are simply relational database domains, where databases are populated by constants, nulls and persistent nulls. The latter values come from a countably infinite set PNull. We denote the elements of PNull using the symbol $\top$ with subscripts. For a database d, we denote by PNull(d) the set of persistent nulls appearing in it.

A P-*valuation* is a partial function v from Null $\cup$ PNull to Const. We denote by $v(d)$ the database obtained from d by replacing each element e of Null $\cup$ PNull with $v(e)$, if this is defined. We introduce P-valuations only as a means to define the semantics of relational database domains with persistent nulls; such P-valuations, however, do not provide additional knowledge about the persistent nulls, as their incompleteness cannot be reduced. Indeed, we consider again the monoid $\mathcal{V}$ of partial valuations introduced in Section 3.2, which is additional to relational database domains with persistent nulls.

In what follows, we will only use relational database domains with persistent nulls as target, and we focus on open-world semantics as before. For simplicity, we use the collapsing semantics of P-valuations, which is defined below and is equivalent to the mixed one.

Definition 3.3.1. *The* collapsing application *of a* P*-valuation* v *to a* k*-ary table* T *is the* k*-ary table* $v(T)$ *such that, for every tuple* $\bar{t} \in$ Const^k *we have:*

$$\#\big(v(T), \bar{t}\big) = \max\big\{\, \#\big(T, \bar{u}\big) \mid v(\bar{u}) = \bar{t} \,\big\}$$

The corresponding OWA *semantics of incompleteness is defined as follows:*

$$[\![d]\!]^{\mathsf{OWA}} = \{d' \textit{ complete} \mid d' \supseteq v(d),\ v \textit{ is a } \mathsf{P}\textit{-valuation}\}$$

With all of this in place, we can capture information-based certain answers while still preserving knowledge from partial valuations.

Below, we show that the valuation-preserving certain answers with persistent nulls are as informative as information-based certain answers, and more informative than valuation-preserving certain answers without persistent nulls.

Proposition 11. *Let $\mathbf{R}$ be a relational database domain with either* OWA *or* CWA *semantics, and let $\mathbf{P}$ be a relational database domain with persistent nulls with* OWA *semantics, such that $\mathbb{C}_{\mathbf{R}} = \mathbb{C}_{\mathbf{P}}$ and $\mathbb{I}_{\mathbf{R}} \subseteq \mathbb{I}_{\mathbf{P}}$. Let q be a query from $\mathbf{R}$ to $\mathbf{R}$, and let q' be a query identical to q, but from $\mathbf{R}$ to $\mathbf{P}$. Then, for every $d \in \mathbb{I}_{\mathbf{R}}$, all of the following hold:*

(a) $\mathsf{cert}_{\square}(q, d) \equiv_{\mathbf{P}} \mathsf{cert}_{\mathcal{V}}(q', d)$*;*
(b) $\mathsf{cert}_{\mathcal{V}}(q, d) \preceq^{\mathcal{V}}_{\mathbf{P}} \mathsf{cert}_{\mathcal{V}}(q', d)$*.*

Proof. Due to its length and for sake of readability, the proof is given at the end of the chapter 3.3.4. □

Since we are now able to capture information-based certain answers, the valuation-preserving certain answers will not be empty. As an application, we look at a class UCQ-$\mathcal{F}$ of value-inventing queries defined by unions of conjunctive queries with function application.

3.3.1 Query Answering for UCQ-$\mathcal{F}$

Let $\mathcal{F}$ be a set of functions where each function f of arity k is from Const^k to Const.

Definition 3.3.2. *The language* UCQ-$\mathcal{F}$ *of* union of conjunctive queries with function application *is defined by the following grammar:*

$$q := R \mid q \times q \mid q \uplus q \mid \pi_{\bar{\alpha}}(q) \mid \sigma_{\alpha_i = \alpha_j}(q)$$
$$\mid \mathsf{Apply}_{\bar{\alpha}}(f, q) \quad \textit{with } f \in \mathcal{F}$$

where α_i and α_j are attributes, and $\bar{\alpha}$ is a tuple of attributes.

All of the operations above, except Apply*, are defined in subsection 3.2.1. The semantics of $q' = \mathsf{Apply}_{\alpha_1,\ldots,\alpha_k}(f, q)$, for every complete relational database d and for every tuple $\bar{t}$ of the same arity as q, is defined as follows:*

$$\#\big(q'(d), (\bar{t}, t)\big) = \begin{cases} \#\big(q(d), \bar{t}\big) & \textit{if } t = f\big(\bar{t}[\alpha_1], \ldots, \bar{t}[\alpha_k]\big) \\ 0 & \textit{otherwise} \end{cases}$$

While the query language UCQ-$\mathcal{F}$ is conceptually quite simple, computing the valuation-preserving certain answers for UCQ-$\mathcal{F}$ queries is intractable even if $\mathcal{F}$ consists of just one unary function.

Proposition 12. *There is a unary function f such that computing the $\{\epsilon\}$-preserving certain answer and the valuation-preserving certain answer to Boolean queries in* UCQ-$\{f\}$*, on relational databases with or without persistent nulls, is coNP-hard in data complexity.*

Proof. Due to its length and for sake of readability, the proof is given at the end of the chapter 3.3.4. □

For UCQ-$\mathcal{F}$ queries, the standard SQL evaluation is polynomial, so it cannot compute any knowledge-preserving notion of certain answers, which may result in counter-intuitive results on database with nulls. An approach to approximate certain answers when they are intractable is to build a query evaluation algorithm with correctness guarantees that runs in polynomial time.

Definition 3.3.3. *Let $\mathbf{S}$ and $\mathbf{T}$ be database domains, and $\mathcal{Q}$ be a query language. A query evaluation algorithm* Eval$\colon \mathcal{Q} \times \mathbb{I}_{\mathbf{S}} \to \mathbb{I}_{\mathbf{T}}$ *has* $\mathcal{A}$-preserving correctness guarantees *for $\mathcal{Q}$ if, for every query $q \in \mathcal{Q}$, all of the following hold:*

(a) Eval$(q, x) \preceq^{\mathcal{A}}_{\mathbf{T}} \mathrm{cert}_{\mathcal{A}}(q, x)$ *for every $x \in \mathbb{I}_{\mathbf{S}}$, and*
(b) Eval$(q, c) = q(c)$ *for every $c \in \mathbb{C}_{\mathbf{S}}$.*

In other words, a query evaluation algorithm with correctness guarantees produces answers that are consistent (w.r.t. the $\mathcal{A}$-preserving information ordering on the target domain) with the query answers on incomplete databases, and equal to them on complete databases.

In order to obtain a polynomial-time algorithm, we extend the idea of naive evaluation on relational database domains based on a free algebra of terms.

3.3.2 Relational Databases over Free Algebra

We consider a free algebra of terms $\mathcal{T}$, where the basis of $\mathcal{T}$ is the set Const $\cup$ Null of constants and (marked) nulls, and the operations are functions from $\mathcal{T} \times \cdots \times \mathcal{T}$ to $\mathcal{T}$. The set of such functions is denoted by Γ. Then, a relational database is over $\mathcal{T}$ if it is populated with elements of $\mathcal{T}$.

Definition 3.3.4. *An* interpretation $\cdot^{\mathcal{I}}$ *of* Γ *over* Const *associates each* $\gamma \in \Gamma$ *of arity* k *with a function* $\gamma^{\mathcal{I}} \colon \mathsf{Const}^k \to \mathsf{Const}$. *The* grounding *of* $\mathcal{T}$ *under* $\mathcal{I}$ *is the function* grd *that maps each* $t \in \mathcal{T}$ *to an element of* Const $\cup$ Null $\cup$ PNull *as follows:*

$$\mathsf{grd}(t) = \begin{cases} t & \text{if } t \in \mathsf{Const} \cup \mathsf{Null} \\ \gamma^{\mathcal{I}}(c_1, \ldots, c_k) & \text{if } t = \gamma(t_1, \ldots, t_k) \text{ and} \\ & c_i = \mathsf{grd}(t_i) \in \mathsf{Const} \\ & \text{for every } i \in \{1, \ldots, k\} \\ \top_{\mathsf{grd}(t)} & \text{otherwise} \end{cases}$$

This naturally extends tables and relational databases over $\mathcal{T}$. *The grounding of a table* T *is the table* $\mathsf{grd}(T)$ *with persistent nulls such that, for every tuple* $\bar{t}$ *of the same arity as* T, *we have:*

$$\#\big(\mathsf{grd}(T), \bar{t}\big) = \sum_{\bar{u} \colon \mathsf{grd}(\bar{u}) = \bar{t}} \#\big(T, \bar{u}\big)\big)$$

Finally, the grounding of a relational database d *over* $\mathcal{T}$ *is the database* $\mathsf{grd}(d)$ *of the same schema such that, for every table name* R, *we have* $R^{\mathsf{grd}(d)} = \mathsf{grd}(R^d)$.

A relational database domain $\mathbf{D}$ is over the free algebra $\mathcal{T}$ if every database in $\mathbb{I}_{\mathbf{D}}$ is over $\mathcal{T}$ and its grounding belongs to $\mathbb{C}_{\mathbf{D}}$; in addition, $[\![\cdot]\!]_{\mathbf{D}}$ uses OWA with the collapsing semantics of valuations as defined below.

Definition 3.3.5. *For every valuation* $v \colon \mathsf{Null} \to \mathsf{Const}$ *and for every term* $t = \gamma(t_1, \ldots, t_k) \in \mathcal{T}$, *we let* $v(t)$ *denote the term* $\gamma\big(v(t_1), \ldots, v(t_k)\big)$.

The collapsing application *of* v *to a* k*-ary table* T *over* $\mathcal{T}$ *is the* k*-ary table* $v(T)$ *such that, for every* $\bar{t} \in \mathcal{T}^k$, *we have:*

$$\#\big(v(T), \bar{t}\big) = \max\big\{\, \#\big(T, \bar{u}\big) \mid v(\bar{u}) = \bar{t} \big\}$$

The corresponding OWA *semantics of incompleteness is defined as follows:*

$$[\![d]\!]^{\mathsf{OWA}} = \{d' \textit{ complete} \mid d' \supseteq v(d),\ v \textit{ is a valuation}\}$$

We will now use the relational database domain over $\mathcal{T}$ to define naive evaluation for value-inventing queries.

3.3.3 Approximation Algorithms for UCQ-$\mathcal{F}$

For the rest of this section, we consider the free algebra of terms $\mathcal{T}$ defined as above, with a set Γ of operations whose interpretation $\mathcal{I}$ over Const is assumed to be computable in polynomial time. We take $\mathcal{F} = \{\gamma^{\mathcal{I}} \mid \gamma \in \Gamma\}$ as the set of functions for UCQ-$\mathcal{F}$ queries.

Definition 3.3.6. *The* interpreted naive evaluation *of a* UCQ- $\mathcal{F}$ *query* q *is the query* $q_{\textit{i-naive}}$ *from incomplete relational databases to incomplete relational databases over* $\mathcal{T}$*, where* $q_{\textit{i-naive}}$ *is the standard naive evaluation for relational algebra (i.e., nulls are treated as new constants) and such that, for every relational database* d *and for every tuple* $\bar{t}$ *of appropriate arity, we have:*

- *for* $q = \mathsf{Apply}_{\alpha_1,\ldots,\alpha_k}(\gamma^{\mathcal{I}}, q')$*;* $\#\big(q_{\textit{i-naive}}(d), (\bar{t}, t)\big)$ *is*

$$\begin{cases} \#\big(q'_{\textit{i-naive}}(d), \bar{t}\big) & \textit{if } t = \gamma(\bar{t}[\alpha_1], \ldots, \bar{t}[\alpha_k]) \\ 0 & \textit{otherwise} \end{cases}$$

- *for* $q = \sigma_{\alpha_i = \alpha_j}(q')$*;* $\#\big(q_{\textit{i-naive}}(d), \bar{t}\big)$ *is*

$$\begin{cases} \#\big(q'_{\textit{i-naive}}(d), \bar{t}\big) & \textit{if } \mathsf{grd}(\bar{t}[\alpha_i]) = \mathsf{grd}(\bar{t}[\alpha_j]) \\ 0 & \textit{otherwise} \end{cases}$$

Informally, the interpreted naive evaluation behaves like the classical naive evaluation for relational algebra, except that it interprets the free algebra terms when checking for equality in selections.

Proposition 13. *For* UCQ-$\mathcal{F}$ *query* q *and for every relational database* d*, the interpreted naive evaluation of* q *can be computed in polynomial time in the size of* d*.*

Proof. Due to its length and for sake of readability, the proof is given at the end of the chapter 3.3.4. □

We have built a polynomial-time evaluation algorithm for UCQ-$\mathcal{F}$ queries on relational database domains over $\mathcal{T}$. In addition, by means of grounding, we can provide an evaluation algorithm with correctness guarantees.

Theorem 7. *The grounding of the interpreted naive evaluation is a query evaluation algorithm with correctness guarantees for* UCQ-$\mathcal{F}$*. Thus, for every query* $q \in$ UCQ-$\mathcal{F}$ *and for every relational database* d*, we have:*

(a) $\mathsf{grd}\big(q_{\textit{i-naive}}(d)\big) \preceq^{\mathcal{V}}_{\mathbf{P}} \mathsf{cert}_{\mathcal{V}}(q, d)$*; and*
(b) *if* d *is complete, then* $\mathsf{grd}\big(q_{\textit{i-naive}}(d)\big) = q(d)$*.*

Proof. Due to its length and for sake of readability, the proof is given at the end of the chapter 3.3.4. □

We conclude this section with a fully worked-out example of how UCQ-$\mathcal{F}$ queries are evaluated and grounded.

Example 5. Consider a database D with a relation $R = \{(3,0,1,2), (1,\bot_1,1,\bot_1), (2,\bot_2,\bot_2,2)$ over attributes A, B, C, D, and the following value inventing query q':

$$\pi_{(A+B);(C+D)}\big(\mathsf{Apply}_{(C,D)}(+, \mathsf{Apply}_{(A,B)}(+, R))\big)$$

where $+(c, c') = c + c'$ for every $c, c' \in \mathsf{Const}$. The naive evaluation of q' on D gives the following answers:

$$q'_{\text{i-naive}}(D) = q'_{\text{naive}}(D) =$$

$A+B$	$C+D$
$+(3,0)$	$+(1,2)$
$+(1,\bot_1)$	$+(1,\bot_1)$
$+(2,\bot_2)$	$+(\bot_2,2)$

The naive evaluation of $q = \sigma_{(A+B)=(C+D)}(q')$ on D and its grounding are:

$q_{\text{naive}}(D)$

$A+B$	$C+D$
$+(1,\bot_1)$	$+(1,\bot_1)$

$\mathsf{grd}(q_{\text{naive}}(D))$

$A+B$	$C+D$
$\top_i$	$\top_i$

These return unsatisfactory answers, because they miss some complete tuples. Naive evaluation cannot capture the fact that $+(0,3) = +(1,2) = 3$, as it does not interpret the function. On the other hand, the interpreted naive evaluation and its grounding are:

$q_{\text{i-naive}}(D)$

$A+B$	$C+D$
$+(3,0)$	$+(1,2)$
$+(1,\bot_1)$	$+(1,\bot_1)$

$\mathsf{grd}(q_{\text{i-naive}}(D))$

$A+B$	$C+D$
3	3
$\top_j$	$\top_j$

The interpreted naive evaluation is able to capture all complete tuples. It recognizes that constant terms such as $+(3,0)$ and $+(1,2)$ are equal. Moreover, for this query, the interpreted-naive evaluation captures more tuples than the certain answers with nulls, but strictly less tuples than the valuation-preserving certain answers on the relational database domain with persistent nulls:

$\mathsf{cert}_{\perp}(q, D)$

$A + B$	$C + D$
3	3

$\mathsf{cert}_{\mathcal{V}}(q, D)$

$A + B$	$C + D$
3	3
$\top_j$	$\top_j$
$\top_k$	$\top_k$

The valuation-preserving certain answer captures more tuples, because it is able to recognize that $+(2, \perp_2)$ and $+(\perp_2, 2)$ are equal for every valuation.

We have been able to build a polynomial query evaluation algorithm with correctness guarantee for the applied union of conjunctive query language UCQ-$\mathcal{F}$. Moreover as illustrated in Example 5, there exist queries where the interpreted naive evaluation algorithm is a strict improvement from the certain answers with nulls: it returns more certain tuples.

3.3.4 Proof of section 3.3: Value-inventing queries

For sake of clarity we denote:

1. $\mathbf{R}$ the relational database domain.
2. $\mathbf{P}$ the relational database domain with persistent nulls.
3. $\mathbf{F}$ the relational database domain with free algebra terms.

Proof of Proposition 11

Proposition. *Let $\mathbf{R}$ be a relational database domain, and let $\mathbf{P}$ be a relational database domain with persistent nulls, such that $\mathbb{C}_{\mathbf{R}} = \mathbb{C}_{\mathbf{P}}$. Let q be a query from $\mathbf{R}$ to $\mathbf{R}$, and let q' be a query identical to q, but from $\mathbf{R}$ to $\mathbf{P}$. Then, for every $d \in \mathbb{I}_{\mathbf{R}}$, all of the following hold:*

(a) $\mathsf{cert}_{\square}(q, d) \equiv_{\mathbf{P}} \mathsf{cert}_{\mathcal{V}}(q', d)$*;*
(b) $\mathsf{cert}_{\mathcal{V}}(q, d) \preceq^{\mathcal{V}}_{\mathbf{P}} \mathsf{cert}_{\mathcal{V}}(q', d)$*.*

Proof. We first prove: $\mathsf{cert}_{\mathcal{V}}(q', d) \preceq_{\mathbf{P}} \mathsf{cert}_{\square}(q, d)$. Notice that for every database d we have $[\![d]\!]_{\mathbf{P}} = [\![d]\!]_{\mathbf{R}}$. Therefore $\mathsf{cert}_{\square}(q', d) = \mathsf{cert}_{\square}(q, d)$. And by lemma 4 we can conclude.

We now prove: $\text{cert}_\square(q,d) \preceq_{\mathbf{P}} \text{cert}_{\mathcal{V}}(q',d)$. Let a function m which maps every null $\bot_i \in \text{Null}(\text{cert}_\square(q,d))$ to a distinct P-null $\top_i \in \text{PNull}$. Then we have $m(\text{cert}_\square(q,d)) \in \mathbb{I}_{\mathbf{P}}$ and as there is no null value in $m(\text{cert}_\square(q,d))$, for every valuation we have $[\![m(\text{cert}_\square(q,d))]\!]^v_{\mathbf{P}} = [\![\text{cert}_\square(q,d)]\!]_{\mathbf{P}}$. Therefore we also have $m(\text{cert}_\square(q,d)) \preceq^{\mathcal{V}}_{\mathbf{P}} q(d) = q'(d)$. And by definition of valuation preserving certain answers as greatest lower bound: $m(\text{cert}_\square(q,d)) \preceq^{\mathcal{V}}_{\mathbf{P}} \text{cert}_{\mathcal{V}}(q',d)$. Hence we can conclude $\text{cert}_\square(q,d) \preceq_{\mathbf{P}} \text{cert}_{\mathcal{V}}(q',d)$.

We want to show: $\text{cert}_{\mathcal{V}}(q,d) \preceq^{\mathcal{V}}_{\mathbf{P}} \text{cert}_{\mathcal{V}}(q',d)$. We notice that $\mathbb{I}_{\mathbf{R}} \subset \mathbb{I}_{\mathbf{P}}$, especially we have $\text{cert}_{\mathcal{V}}(q,d) \in \mathbb{I}_{\mathbf{P}}$. And it verifies $\text{cert}_{\mathcal{V}}(q,d) \preceq^{\mathcal{V}}_{\mathbf{P}} q(d) = q'(d)$. Hence by definition of greatest lower bound we have $\text{cert}_{\mathcal{V}}(q,d) \preceq^{\mathcal{V}}_{\mathbf{P}} \text{cert}_{\mathcal{V}}(q',d)$. □

Proof of Proposition 12

Proposition. *There is a unary function f such that computing the $\{\epsilon\}$-preserving certain answer and the valuation-preserving certain answer to boolean queries in* UCQ-$\{f\}$*, on relational databases with or without persistent nulls, is coNP-hard in data complexity.*

Proof. We consider the modulo operation f such that for every constant $c \in \text{Const}$ we have $f(c) = c \mod 3$. Then it is easy to build a boolean query in UCQ-$\{f\}$ which is not empty if and only if the numbering of a graph is not a valid 3-colouring. Therefore by reduction to 3-colouring computing the certain answers to such query is coNP-hard in data-complexity. □

Proof of proposition 13

Proposition. *For* UCQ-$\mathcal{F}$ *query q and for every relational database d, the interpreted naive evaluation of q can be computed in polynomial time in the size of d.*

We focus on the selection operation and prove that computing it for each tuple can be done in polytime with respect to the size of the query.

Definition 3.3.7. *For each term $t \in \mathcal{T}$ we define the metric of t denoted $\|t\|$ such that:*

$$\|t\| \quad = \begin{cases} 1 + \sum_{i=1}^{k} \|t_i\| & \text{if } t = \gamma(t_1, \cdots t_k) \\ 1 & \text{otherwise} \end{cases}$$

Lemma 5. *For every query q in* UCQ-$\mathcal{F}$ *and every database d, the metric of the term in $q_{\text{i-naive}}(d)$ is independent of the size of d.*

Proof. By induction assume that q is such that for every term $t \in q_{\text{i-naive}}(d)$ we have $\|t\|$ independent of the size of d. Let $Q = \text{Apply}_{(\alpha_1,\cdots,\alpha_k)}(\gamma, q)$. Let $t \in Q_{\text{i-naive}}(d)$, then by definition of Q and interpreted naive evaluation, we either have $t \in q_{\text{i-naive}}(d)$, or $t = \gamma(\bar{t})$ with $\bar{t} \in q_{\text{i-naive}}(d)$. The first case is immediate, hence assume $t = \gamma(\bar{t})$ with

$\bar{t} \in q_{\text{i-naive}}(d)$; then by definition:

$$\|t\| = 1 + \sum_{t' \in \bar{t}} \|t'\| \tag{3.7}$$

And by hypothesis of induction $\|t'\|$ is independent of the size of d, hence $\|t\|$ is also independent. □

By definition of interpreted naive evaluation, checking selection can then by done linearly in the size of the term. And by lemma previous lemma we can conclude that computing selection can be done in polytime with respect to size of d.

Proof of Theorem 7

This proof is more involved: we first have to prove that $q_{\text{i-naive}}(d) \preceq^{\mathcal{V}}_{\mathbf{F}} \text{cert}_{\mathcal{V}}(q_{\text{i-naive}}, d)$, and then that $\text{grd}(\text{cert}_{\mathcal{V}}(q_{\text{i-naive}}, d)) \preceq^{\mathcal{V}}_{\mathbf{P}} \text{cert}_{\mathcal{V}}(q, d)$.

Lemma 6. *For every free relational database $d, d' \in \mathbb{I}_{\mathbf{F}}$, the valuation-information pre-order is equivalent to the subset pre-order ie. $d \preceq^{\mathcal{V}}_{\mathbf{F}} d' \Leftrightarrow d \subseteq d'$.*

Proof. The proof is similar to the one for $\mathbf{R}$. □

Lemma 7. $q_{\textit{i-naive}}(d) \subseteq \text{cert}_{\mathcal{V}}(q_{\textit{i-naive}}, d)$

Proof. The proof is done by induction on the structure of q; we assume q, q' such that for every database we have $q_{\text{i-naive}}(d) \subseteq \text{cert}_{\mathcal{V}}(q_{\text{i-naive}}, d)$ and $q'_{\text{i-naive}}(d) \subseteq \text{cert}_{\mathcal{V}}(q'_{\text{i-naive}}, d)$

Case 1. Let $Q_{\text{i-naive}} = q_{\text{i-naive}} \uplus q'_{\text{i-naive}}$.
By definition of $Q_{\text{i-naive}}$ we have $Q_{\text{i-naive}}(d) = q_{\text{i-naive}}(d) \uplus q'_{\text{i-naive}}(d)$. And by induction hypothesis we conclude $Q_{\text{i-naive}}(d) \subseteq \text{cert}_{\mathcal{V}}(q_{\text{i-naive}}, d) \uplus \text{cert}_{\mathcal{V}}(q'_{\text{i-naive}}, d)$.

Case 2. Let $Q_{\text{i-naive}} = \text{Apply}_{(\alpha_1, \cdots, \alpha_k)}(\gamma, q_{\text{i-naive}})_{\text{i-naive}}$. Then by definition for every tuple $\bar{t} \in \mathcal{T}^*$ we have

$$\#\big(Q_{\text{i-naive}}(d), (\bar{t}, t)\big) = \begin{cases} \#\big(q_{\text{i-naive}}(d), \bar{t}\big) & \text{if } t = \gamma(\bar{t}[\alpha_1], \cdots, \bar{t}[\alpha_k]) \\ 0 & \text{otherwise.} \end{cases}$$

By induction hypothesis we have $\#\big(q_{\text{i-naive}}(d), \bar{t}\big) \leq \#\big(\text{cert}_{\mathcal{V}}(q_{\text{i-naive}}, d), \bar{t}\big)$ hence:

$$\#\big(Q_{\text{i-naive}}(d), (\bar{t}, t)\big) \leq \begin{cases} \#\big(\text{cert}_{\mathcal{V}}(q_{\text{i-naive}}, d), \bar{t}\big) & \text{if } t = \gamma(\bar{t}[\alpha_1], \cdots, \bar{t}[\alpha_k]) \\ 0 & \text{otherwise.} \end{cases}$$

Therefore we have $\#\big(Q_{\text{i-naive}}(d), (\bar{t}, t)\big) \leq \#\big(\text{cert}_{\mathcal{V}}(Q_{\text{i-naive}}, d), (\bar{t}, t)\big)$, and we conclude $Q_{\text{i-naive}}(d) \subseteq \text{cert}_{\mathcal{V}}(Q_{\text{i-naive}}, d)$.

Case 3. Let $Q_{\text{i-naive}} = \sigma_{\alpha_i=\alpha_j}(q_{\text{i-naive}})_{\text{i-naive}}$. Then by definition for every tuple $\bar{t} \in \mathcal{T}^*$ we have:

$$\#\big(Q_{\text{i-naive}}(d), \bar{t}\big) = \begin{cases} \#\big(q_{\text{i-naive}}(d), \bar{t}\big) & \text{if } \text{grd}(\bar{t}[\alpha_i]) = \text{grd}(\bar{t}[\alpha_j]) \\ 0 & \text{otherwise.} \end{cases}$$

By induction hypothesis we have $\#\big(q_{\text{i-naive}}(d), \bar{t}\big) \leq \#\big(\text{cert}_{\mathcal{V}}(q_{\text{i-naive}}, d), \bar{t}\big)$ hence:

$$\#\big(Q_{\text{i-naive}}(d), \bar{t}\big) \leq \begin{cases} \#\big(\text{cert}_{\mathcal{V}}(q_{\text{i-naive}}, d)), \bar{t}\big) & \text{if } \text{grd}(\bar{t}[\alpha_i]) = \text{grd}(\bar{t}[\alpha_j]) \\ 0 & \text{otherwise.} \end{cases}$$

Therefore we have $\#\big(Q_{\text{i-naive}}(d), (\bar{t}, t)\big) \leq \#\big(\text{cert}_{\mathcal{V}}(Q_{\text{i-naive}}, d), (\bar{t}, t)\big)$, and we conclude $Q_{\text{i-naive}}(d) \subseteq \text{cert}_{\mathcal{V}}(Q_{\text{i-naive}}, d)$. Note that for the selection the converse is not true, in general $\text{cert}_{\mathcal{V}}(Q_{\text{i-naive}}, d) \not\subseteq Q_{\text{i-naive}}(d)$

The other cases are similar. □

Lemma 8. *grd*$(\text{cert}_{\mathcal{V}}(q_{\textit{i-naive}}, d)) \preceq^{\mathcal{V}}_{\mathbf{P}} \text{cert}_{\mathcal{V}}(q, d)$

Proof.

Claim 1. *For every free algebra database $d \in \mathbb{I}_{\mathbf{F}}$ and every d-complete valuation v we have*

$$\textit{grd}([\![v(d)]\!]_{\mathbf{F}}) \subseteq [\![v(\textit{grd}(d))]\!]_{\mathbf{P}}$$

By definition of the interpreted naive evaluation we have $c \in \mathbb{C}_{\mathbf{R}}$ we have $\text{grd}(q_{\text{i-naive}}(c)) = q(c)$. Notice that it is not true for naive evaluation, we only have $\text{grd}(q_{\text{naive}}(c)) \subseteq q(c)$.
For every every d-complete valuation v we have $q_{\text{naive}}(v(d)) \in [\![\text{cert}_{\mathcal{V}}(q_{\text{i-naive}}, d)]\!]^v$.
Therefore we have $\text{grd}(q_{\text{naive}}(v(d))) \in \text{grd}([\![\text{cert}_{\mathcal{V}}(q_{\text{i-naive}}, d)]\!]^v)$ and $q(v(d)) \in \text{grd}([\![\text{cert}_{\mathcal{V}}(q_{\text{i-naive}}, d)]\!]^v)$.
By claim 1 we can obtain $q(v(d)) \in [\![\text{grd}(\text{cert}_{\mathcal{V}}(q_{\text{i-naive}}, d))]\!]^v$.
Therefore we have $\text{grd}(\text{cert}_{\mathcal{V}}(q_{\text{i-naive}}, d)) \preceq^{\mathcal{V}}_{\mathbf{P}} q(d)$, and by definition of greatest lower bound we conclude that $\text{grd}(\text{cert}_{\mathcal{V}}(q_{\text{i-naive}}, d)) \preceq^{\mathcal{V}}_{\mathbf{P}} \text{cert}_{\mathcal{V}}(q, d)$. □

Theorem 1. *The grounding of the interpreted naive evaluation is a query evaluation algorithm with correctness guarantees for* UCQ-$\mathcal{F}$*. Thus, for every query $q \in$ UCQ-$\mathcal{F}$ and for every relational database d, we have:*

(a) *grd*$(q_{\textit{i-naive}}(d)) \preceq^{\mathcal{V}}_{\mathbf{P}} \text{cert}_{\mathcal{V}}(q, d)$*; and*
(b) *if d is complete, then grd*$(q_{\textit{i-naive}}(d)) = q(d)$*.*

Proof. Combining the previous lemmas we obtain $\mathsf{grd}(q_{\text{i-naive}}(d)) \preceq^{\mathcal{V}}_{\mathbf{P}} \mathsf{grd}(\mathsf{cert}_{\mathcal{V}}(q_{\text{i-naive}}, d)) \preceq^{\mathcal{V}}_{\mathbf{P}} \mathsf{cert}_{\mathcal{V}}(q, d)$ which conclude the proof of Theorem 7 □

Proof of Claim 1

Claim. *For every free algebra database $d \in \mathbb{I}_{\mathbf{F}}$ and every d-complete valuation v we have*

$$\mathit{grd}([\![v(d)]\!]_{\mathbf{F}}) \subseteq [\![v(\mathit{grd}(d))]\!]_{\mathbf{P}}$$

Proof. First by induction on the term we are first proving that for every $\bar{t} \in \mathcal{T}^*$ there exists a P-valuation v_{P} such that $v_{\mathsf{P}}\big(v(\mathsf{grd}(\bar{t}))\big) = \mathsf{grd}(v(\bar{t}))$.

Case 1. Let $c \in \mathsf{Const}$ then we trivially have $v(\mathsf{grd}(c)) = \mathsf{grd}(c) = c = v(c) = \mathsf{grd}(v(c))$.

Case 2. Let $\bot \in \mathsf{Null}$ and $v(\bot) = c$ by definition we have $\mathsf{grd}(\bot) = \bot$. Hence $v(\mathsf{grd}(\bot)) = v(\bot) = c$ and $\mathsf{grd}(v(\bot)) = \mathsf{grd}(c) = c$.

Case 3. Let $\gamma(\bar{t}) \in \mathcal{T}$.
Assume $\mathsf{grd}(\bar{t}) \in \mathsf{Const}^*$. By definition $\mathsf{grd}(\gamma(\bar{t})) \in \mathsf{Const}$ therefore $v(\mathsf{grd}(\gamma(\bar{t}))) = \mathsf{grd}(\gamma(\bar{t}))$. Moreover as $\mathsf{grd}(\bar{t}) \in \mathsf{Const}^*$ we also have $v(\gamma(\bar{t})) = \gamma(\bar{t})$.
Assume $\mathsf{grd}(\bar{t}) \notin \mathsf{Const}^*$. then by definition $\mathsf{grd}(\gamma(\bar{t})) = \top_{\gamma(\bar{t})}$. Let v_{P} the P-valuation such that $v_{\mathsf{P}}(\top_{\gamma(\bar{t})}) = \mathsf{grd}(\gamma(v(\bar{t}))) \in \mathsf{Const}$. Then we have $v_{\mathsf{P}}(v(\top_{\gamma(\bar{t})})) = \mathsf{grd}(\gamma(v(\bar{t})))$.

Case 4. Let $\bar{t} = (t_1, t_2)$ such that for every $i \in \{1, 2\}$ there exists a P-valuation $v_{\mathsf{P}i}$ such that $\mathsf{grd}(v(t_i)) = v_{\mathsf{P}i}\big(v(\mathsf{grd}(t_i))\big)$. By definition $\mathsf{grd}(v(\bar{t})) = \big(\mathsf{grd}(v(t_1)), \mathsf{grd}(v(t_2))\big)$. By induction hypothesis we have: $\mathsf{grd}(v(\bar{t})) = \big(v_{\mathsf{P}1}(v(\mathsf{grd}(t_1))), v_{\mathsf{P}2}(v(\mathsf{grd}(t_2)))\big)$. We consider the P-valuation v_{P} such that $v_{\mathsf{P}}(\top_{t_1}) = v_{\mathsf{P}1}(\top_{t_1})$ and $v_{\mathsf{P}}(\top_{t_2}) = v_{\mathsf{P}2}(\top_{t_2})$. And we obtain $\big(v_{\mathsf{P}1}(v(\mathsf{grd}(t_1))), v_{\mathsf{P}2}(v(\mathsf{grd}(t_2)))\big) = v_{\mathsf{P}}(v(\mathsf{grd}(\bar{t})))$

We have been able to prove that for every $\bar{t} \in \mathcal{T}^*$ there exists a P-valuation v_{P} such that $\mathsf{grd}(v(\bar{t})) = v_{\mathsf{P}}\big(v(\mathsf{grd}(\bar{t}))\big)$. Now we want to extend this result to relations, but it does not hold because of possible unification and therefore collapsing. Hence we prove that for every free-algebra relation there exists a P-valuation v_{P} such that $v_{\mathsf{P}}\big(v(\mathsf{grd}(R))\big) \subseteq \mathsf{grd}(v(R))$. We consider the P-valuation v_{P} such that

$$v_{\mathsf{P}} = \bigcirc_{\bar{t} \in R} v_{\mathsf{P}\bar{t}}$$

where $v_{\mathsf{P}\bar{t}}$ such that $\mathsf{grd}(v(\bar{t})) = v_{\mathsf{P}\bar{t}}\big(v(\mathsf{grd}(\bar{t}))\big)$. Then every tuples in $v_{\mathsf{P}}\big(v(\mathsf{grd}(R))\big)$ and $\mathsf{grd}(v(R))$ are the same. And their multiplicities might be lower in $v_{\mathsf{P}}\big(v(\mathsf{grd}(R))\big)$ because of the collapsing application of v_{P}. Therefore we have

$$v_{\mathsf{P}}\big(v(\mathsf{grd}(R))\big) \subseteq \mathsf{grd}(v(R))$$

From there we can trivially extend this results to free algebra databases: for every free-algebra database d there exists a P-valuation v_{P} such that $v_{\mathsf{P}}\big(v(\mathsf{grd}(d))\big) \subseteq \mathsf{grd}(v(d))$.

Then by definition of $\mathbf{P}$ for every free algebra database $d \in \mathbb{I}_{\mathbf{F}}$and every d-complete valuation v we have

$$\mathsf{grd}(v(d)) \in [\![v(\mathsf{grd}(d))]\!]_{\mathbf{P}}$$

And by definition of the $[\![\cdot]\!]_{\mathbf{P}}$ semantic for every $c' \in \mathbb{C}_{\mathbf{R}}$ such that $\mathsf{grd}(v(d)) \subseteq c'$ we have $c' \in [\![v(\mathsf{grd}(d))]\!]_{\mathbf{P}}$. Also by definition of the $[\![\cdot]\!]_{\mathbf{F}}$ semantic for every $c' \in \mathsf{grd}([\![\cdot]\!]_{\mathbf{F}})$ we have $\mathsf{grd}(v(d)) \subseteq c'$. Hence we can conclude

$$\mathsf{grd}([\![v(d)]\!]_{\mathbf{F}}) \subseteq [\![v(\mathsf{grd}(d))]\!]_{\mathbf{P}}$$

□

3.4 Conclusion

If one looks at real-life database queries (e.g., in standard benchmarks, as those produced by the TPC-H), there are very few queries from the classes for which we have query-answering techniques in the presence of incomplete data. Real-life queries differ in the most basic semantics of the underlying data model and in their features, as crucially, they invent new values that form part of the output. For those, we lacked notions of correctness, or certainty, of query answering.

Our goal here was to remedy this situation by first providing a general framework explaining what correctness is, and second by showing how it can be applied in some cases that go well beyond queries that we had known how to handle. We have done so for the prevalent bag semantics and for queries that can invent new values by means, for example, of arithmetic functions.

The most pressing next question is to extend these techniques to queries with aggregates that can produce new values by applying arithmetic functions to entire columns in relations. These are extremely common in applications, as witnessed again by benchmark queries. Theoretical literature offers no insight into a satisfactory notion of answers for queries with aggregates over incomplete data. The only well-defined notions for queries with aggregates are the information-based certain answers and the valuation-preserving certain answers with persistent nulls. And they produce a fresh null or a persistent null for each aggregation involving incomplete data.

For generic queries, the valuation-preserving certain answers are a strict improvement over the information-based certain answers. They have the same information content and the same computational complexity, but the valuation-preserving certain answers remain relevant and consistent when the values of some nulls are discovered. However, we have only been able to improve certain answers without any computational cost because every element preserving the knowledge from valuations in the certain answers is also in the source database. Indeed, for generic queries, the elements in the valuation-preserving certain answers (null values included) are also elements of the source database.

For queries with aggregates, we do not have a one-to-one injection between the element of the answer and the element of the source database. Actually, the size of an element preserving the knowledge from valuations can be exponential with respect to the size of the source database. Therefore, for aggregate queries, if we want to improve the information-based certain answers without a too high computational cost, we have to consider a different additional knowledge.

In order to know what would be a relevant additional knowledge, we need to discover what are the database users' expectations.

Chapter 4

SQL incompleteness: A Socio-technical study

Based on 'Troubles with nulls, views from the users' published in VLDB 2022, Toussaint, Guagliardo, Libkin, and Sequeda (2022).

Despite its maturity, foundational research on incomplete data has not yet properly translated to practice. While we may expect academics to be familiar with concepts such as relational algebra or even certain answers, we should not expect it from practitioners. In order to study what real-life users expect from incomplete databases, we need to consider the SQL database management system that practitioners are familiar with. Afterward, we will be able to infer how to improve and implement the theoretical models in a relevant way.

Anecdotally, one hears that the way in which SQL handles nulls creates a myriad of problems in everyday applications of database systems. To the best of our knowledge, however, the actual shortcomings of SQL, as perceived by database practitioners, have yet to be systematically documented, and it is not known if existing research results can readily be used to address the practical challenges.

Our goal *is to improve our understanding of database practitioner's expectations and approaches to incomplete information in relational databases, namely* `NULL` *values*, in order to know how we can further bridge the gap between theory and practice. We wish to validate or refute the common assumptions made by the database research community about the causes and potential solutions to the problem. Towards achieving our goal, we seek to answer four questions:

(Q1) How commonly are SQL's `NULL` and related features used?
(Q2) What do nulls mean to users?
(Q3) Are users satisfied with SQL's handling of nulls and, if not, why?
(Q4) Are there readily available solutions to mitigate the problems?

As a first milestone to find answers to these questions, we designed an online survey which ran for four months and attracted 175 participants, with three quarters of them being database practitioners and one quarter academics. Our major findings, in response to the above questions, are summarized below.

SQL's **NULL** *features usage.* Our participants acknowledge that **NULL** values appear often in their databases. They use some **NULL**-specific operators of SQL (most commonly **IS NULL** tests), but rely more on schema constraints to rule out **NULL** values. It appears that **NULL** values are mostly perceived as an inconvenience rather than a feature that one can take advantage of.

Meaning of **NULL** *values.* **NULL** values appear for many reasons, and our participants often ascribe different meanings to them. While there is a near consensus that nulls can represent non-applicable values, only a quarter of respondents think that this is the only interpretation. A majority think that a **NULL** can also represent some unknown value, which may or may not exist. The meaning of **NULL** could also be 0, empty-string, or another constant, but this is rare.

SQL handling of nulls is not satisfactory. SQL's rules for handling **NULL** values are not fully satisfactory. While for simple queries (positive fragment of relational algebra) most of our respondents accept SQL's behavior, for more complex queries, involving either aggregation or negation, many are not satisfied with SQL answers.

No readily available solutions and more research is needed. There is no consensus among the respondents as to what a better behavior of SQL could be. The desired behaviors are diverse, and independent of the users' view of what nulls mean. Some users want the answers to contain more tuples (moving in the direction of possible answers), others want fewer tuples (but with more guarantees), and yet others simply wish for a warning or error message to be given. These different approaches also indicate that the focus of the academic literature, which typically concentrated on the missing value model of nulls and certain answers as the holy grail, is addressing only a very narrow spectrum of the database practitioner's needs.

While the responses we collected and analyzed can only provide indications, the scope of the mismatch between the common assumption made by the research community and the participants' answers provide strong evidence that a problem does exist and deserves to be studied with more resources. Obtaining definitive answers to questions (Q1)–(Q4), requires access to a larger sample of participants and a more in-depth protocol that can include random sampling, interviews, web-data analysis (Twitter, Stack

Overflow, Reddit, etc.) Bryman (2016); Evans and Mathur (2018). Our questionnaire design can serve as the basis for such further studies. Moreover, the design of a social study experiment necessitates some prior information about the target population to minimize bias and maximize efficiency; our survey design and analysis methodology allow to gather such information Evans and Mathur (2005). An obvious step in this direction would be to run our survey on a larger sample, although we note that our sample of 175 respondents is twice as large as those that have previously been seen in database research of this kind Sahu, Mhedhbi, Salihoglu, Lin, and Özsu (2017) and has a higher proportion of practitioners. A different possibility is to run the survey on a more focused sample (e.g., database professionals working in a particular industry, specific DBMS users) to study the problems that different application domains may experience and reduce non-response bias Bhattacherjee (2012). For that reason, the survey and the software for analyzing its results have been made available in the GitHub repository.

4.1 Survey Design and Methodology

In this section, we explain the design of the survey and the methodology used to analyze each type of question. We then describe how participants were recruited, and we analyze the respondents' demographic and their level of engagement with the survey.

4.1.1 Question Types and Analysis Methodology

The survey is an online structured questionnaire consisting of 34 items. To reduce bias and improve the experience of respondents, the design of the survey has been reviewed by a selected group of practitioners and academics. However, biases are inherent to the survey tool, and while some are reduced by our process of answers analysis, we advise readers to discuss our findings.

Multiple-choice These questions were used when there is a fixed number of possible answers, with an option labeled "other" always made available as a fallback. For this kind of questions, we wish to study the prevalence of choices; therefore, for each available option, we compute the proportion of respondents who selected it. When a question allows more than one option to be chosen, we also compute the proportion of participants who selected each subset of all available options. In our analysis, the data collected through multiple-choice questions is mostly used to get demographic information about the participants, and the results are presented as a pie or bar chart.

Consider the following SQL query:

```
SELECT cid, name
  FROM Customers
 WHERE name = alias
```

on the following SQL database:

Customers

cid	name	alias
c1	Etienne	Etienne
c2	Leonid	NULL
c3	Paolo	Juan
c4	NULL	
c5	NULL	NULL

Regardless of what SQL computes, please add [⊕] the rows you would like to be in the answer. You can add each row multiple times.

Rows

cid	name	
c1	Etienne	⊕
c2	Leonid	⊕
c3	Paolo	⊕
c4	NULL	⊕
c5	NULL	⊕
other	other	⊕

Answers

cid	name

(a) Build the answer

Consider the following SQL query:

```
  SELECT O.cid, SUM(O.price) AS result
    FROM Orders AS O
GROUP BY O.cid
```

on the following SQL database:

Orders

oid	cid	price	taxes
o5	NULL	10	0
o6	NULL	20	10

Regardless of what SQL computes, please specify how satisfied you are with each of the following answer tables.

cid	result
NULL	0

★★★★★

cid	result
NULL	30

★★★★★

cid	result
NULL	10
NULL	20

★★★★★

I would prefer the following:

Please describe your answer

★★★★★

(b) Likert interval scale

Figure 4.1: Examples of questions in our online survey.

Frequency scale These are "how-often" questions whose answer is a frequency chosen from an ordinal Likert scale Likert (1932) with options *never*, *infrequently*, *occasionally*, *often*, and *regularly*, which are ordered from the least to the most frequent. Answers to different frequency scale questions can be compared with one another, but we cannot assume that the frequency differential between each subsequent option in the scale is constant; thus, numerical analysis that computes an average frequency score on a single question is largely meaningless Robertson (2012). For each such question and each frequency option, we compute the proportion of participants who selected that option in the question under consideration. As the number of available options is limited, these statistics can be effectively exploited. Moreover, since the same scale is used for several questions, we can compare the relative frequency of some events. The data collected through frequency scale questions is mostly used to obtain information about the participants' demographic as well as to answer question (Q1).

Build the answer In these questions (Figure 4.1a) participants are presented with a relational database and an SQL query, and then asked to construct, by adding default or custom rows to the output table, *the answer they would like to obtain*. This design allows us to gather information about our respondents' expectations from the evaluation of each query. From an abstract point of view, this type of question is a multiple-choice question. The default rows can be seen as the choices of the question, and adding them to the answer is equivalent to selecting the choice. However, due to the nature of the task, even if the prevalence of each row can be of interest, we argue that more emphasis should be put towards SQL's default answers. Therefore, we partition the participants' answers into four groups, according to whether the answer

(1) matches the SQL answer,
(2) is a subset of the SQL answer,
(3) is a superset of the SQL answer, or
(4) is not comparable with the SQL answer.

We then compute the proportion of answers in each of the groups and report the results in a table. The data collected through this kind of questions are mostly used to answer the question (Q3).

Interval Likert scales In these questions (Figure 4.1b) participants are presented with a relational database, a *value-inventing* SQL query (i.e., producing a value not already present in the database, such as an aggregate value over a column) and several tables. The task is to score each table with a value between 0 and 5 stars (in half-star increments) based on how satisfied the participant would be if the scored table were the answer to the query on the given database. This design lets us gather information about our respondents' expectations for the evaluation of value-inventing queries. Each option is displayed with an initial satisfaction score of 0. The respondents can also use an option labeled "other", with an initial score of 5, to provide a better alternative to those presented. We only consider the score of a custom answer if the participant has provided one. For each query we report the average and the quartile values of the satisfaction score obtained by the SQL answer, and we also compute the proportion of participants who would be more satisfied with a different answer. This data is used to answer the question (Q3).

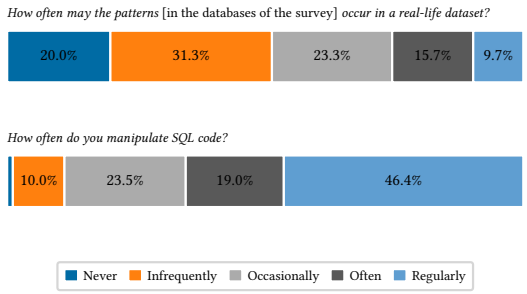
In designing a survey which asks the respondents to evaluate or construct query answers, it is important to ensure that the sample relational databases represent realistic real-life scenarios rather than data patterns that are unlikely to occur. The latter would render the results of the survey less valuable. To this end, for each of the three databases used in the survey, we asked the participants the following frequency question: *How often may the pattern in the given database occur in a real-life dataset?* As shown in the summary (the average for the three databases used) of responses in Figure 4.2a, 80% of our respondents are of the opinion that such patterns can occur in real-life databases, with a varying degree of frequency; about half are of the opinion that such patterns occur frequently.

For all types of questions, depending on the quality of the respondents' sample, the results can be either considered as a representation of the general population (high-quality sample) or, in the case of a small sample, used to identify trends or outliers (choices for multiple-choice questions, events for frequency scale questions, and categories for build the answer and interval Likert scale questions) Evans and Mathur (2005). We will discuss the results we obtained from the participants we managed to recruit, but since the quality of the conclusions one can draw is heavily dependent on the size and quality of the sample, anyone is encouraged to run the survey and its analyses on their own community.

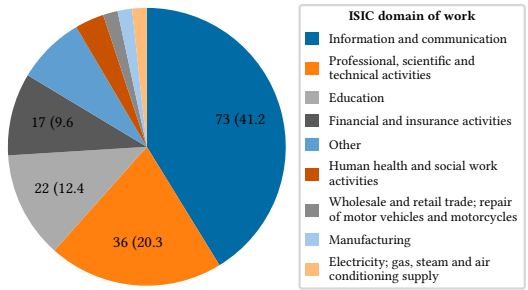
4.1.2 Sample of Respondents

We conducted the survey during a four-month period, and used several different methods to recruit participants. First, we posted a summary of our research on the data.world blog Sequeda (2020) to advertise the survey. We also sent both the survey and the blog post to the data.world mailing list through the monthly digest, reaching their user base. We used mailing lists, such as dbworld, as well as Reddit posts, and several Slack channels dedicated to specific DBMSs.

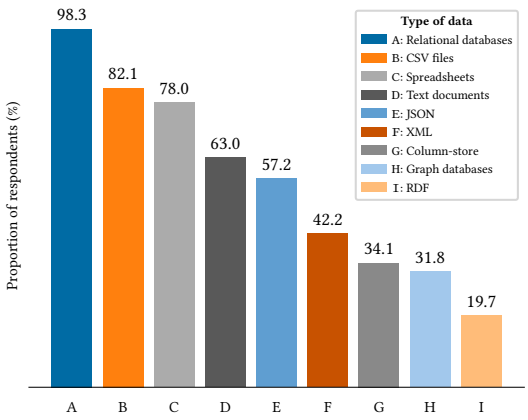
All participants accessed the survey using the same online link, so we cannot tell how many participants were recruited via each individual channel. In the end, there were 175 participants who took the survey, among which 94 completed it in full. Below, we discuss the participants' demographics and how their engagement with the questions evolved during the completion of the survey.



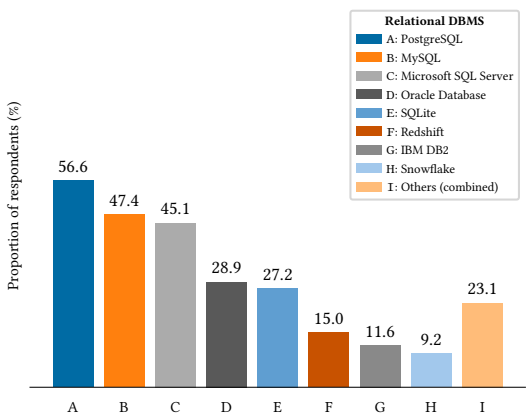
(a) Database patterns and manipulation of SQL code.



(b) Participants' ISIC domain of work.



(c) Types of data users work with.



(d) Popularity of Relational DBMSs.

Figure 4.2: Demographic information

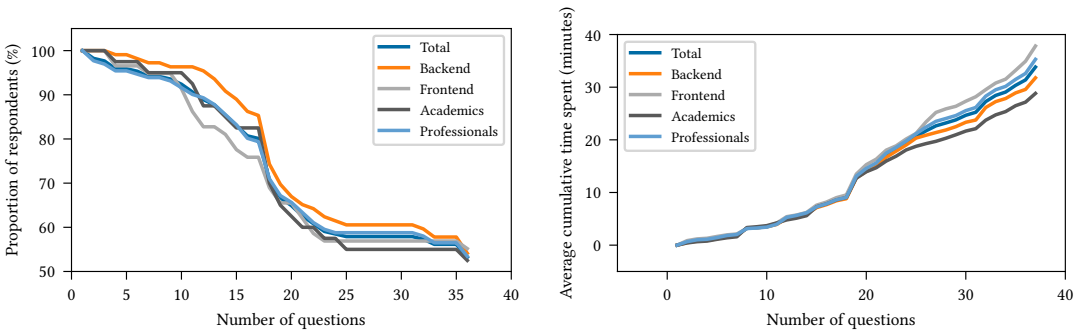
Demographic information We first asked the participants which domain they work in, according to the ISIC classification Nations (2007). As shown in Figure 4.2b, the participants indicated 14 different domains, demonstrating that nulls matter for a wide variety of fields.

Next, we asked the participants what best describes their role in their organization. According to the answers, we split participants into "practitioners" and "academics", where the latter are those who chose *Education* as a domain of activities, or *Professor* or *Student* as a role in their organization. As intended, the sample has a prevalence of practitioners (73%), which our study is geared for.

We asked how often participants manipulate SQL code, to assess their familiarity with the language itself. The answers are shown in Figure 4.2a. We split our participants into two groups:

- *Front-end users*, who manipulate SQL code only occasionally or infrequently; they make up 34% of the respondents (with 1% saying never).
- *Back-end users*, who manipulate SQL code often or regularly; they account for 65% of the respondents with the largest group, almost half, saying regularly; thus, we have reached our target audience.

Finally, we asked what type of data and which relational DBMSs our participants use. As shown in Figure 4.2c and Figure 4.2d, they mostly deal with relational data, and use a variety of systems, with a preference for PostgreSQL, MySQL, and Microsoft SQL Server (followed by Oracle Database, SQLite, and others).[1]



(a) Proportion of respondents for each question.

(b) Average time spent on each question.

Figure 4.3: Participants' engagement

1. The survey erroneously offered both "DB2" and "IBM DB2" as options; 20 respondents chose at least one of them, with 4 selecting both, so we considered 16 as the combined total.

Participants' engagement The complexity and the extent of our study was too ambitious to be kept within the recommended 10-minute survey format Revilla and Ochoa (2017). Despite our best efforts, we knew that our survey would take longer than that to complete. It is surprising that so many participants spent so much time on it. Figure 4.3a and Figure 4.3b show the number of participants and the average time they spent on the survey, respectively; we report the results for each question and each group of participants.

We observe that the engagement is similar for each group of participants, and a significant drop-off in the number of respondents occurs after question 18, which in fact corresponds to the 10-minute mark of average time spent on the survey. In retrospect, the design of question 18 could have been improved: although not complex to answer, it is bulky and cumbersome, and this may have discouraged some participants from continuing.

We also observe that, despite the average completion time being rather high (around 45 minutes), a high proportion (60%) of participants finished the survey. This shows that database practitioners have a strong interest in the problem of nulls in SQL.

4.2 SQL's NULL features usage

The importance of studying nulls stems from their ubiquity in everyday applications; this was confirmed by the survey. Figure 4.4a shows how frequently users encounter nulls; more than 80% of them see **NULL** often or regularly, and less than 20% fall into the *infrequently* and *occasionally* categories. Even if **NULL** does not appear in a dataset, it can be generated by queries, in particular outer joins. Figure 4.4d shows that users frequently deal with **LEFT** and **RIGHT JOIN**; in fact, these are more common than explicitly specified inner joins. Full outer joins are also quite common.

The features offered by RDBMSs to handle **NULL** can be subdivided into the following two categories:

- SQL's DDL constraints, such as **NOT NULL**, to prevent **NULL** from appearing in columns;
- null-specific tests and functions, such as **IS** [**NOT**] **NULL**, **IS** [**NOT**] **DISTINCT**, **COALESCE**(), **ISNULL**(), and **IFNULL**().

Figure 4.4b and Figure 4.4c show the prevalence of using null-prohibiting constraints in the DDL. The use of **NOT NULL** is very common, with almost 70% of respondents regularly or often declaring columns, and just over 5% avoiding the practice. These are also frequently added to keys, foreign keys, and **UNIQUE** declarations. It is interesting to note that, in the case of primary keys, **NOT NULL** is superfluous; yet, the majority of

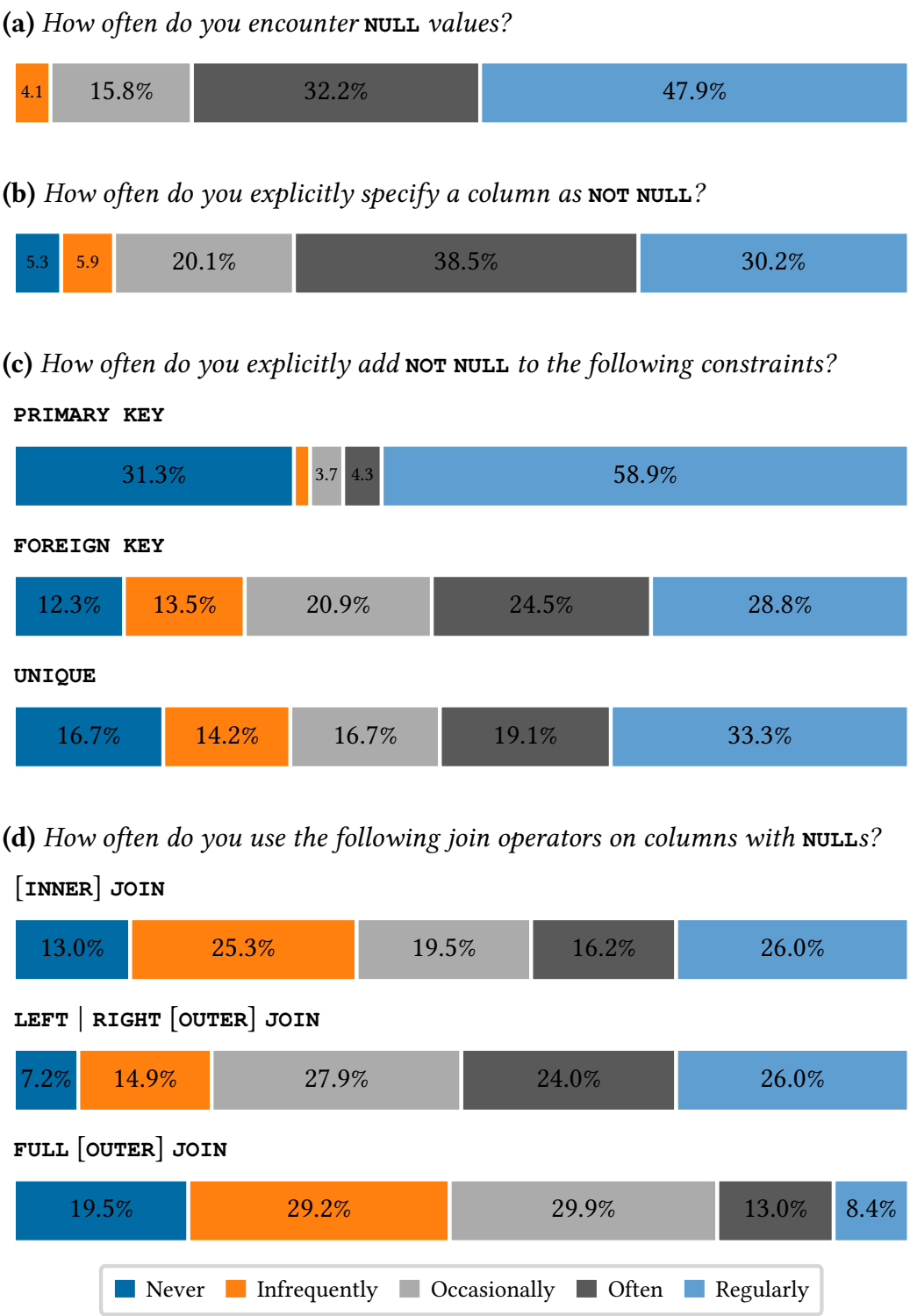


Figure 4.4

users nevertheless includes it explicitly. Adding `NOT NULL` to foreign keys also appears to be a common practice, in all likelihood aimed at avoiding three-valued logic in joins. On the other hand, it is also noteworthy that a non-negligible minority, around a quarter of respondents, never or almost never use null-prohibiting declarations.

(e) *How often do you explicitly add* **NOT NULL** *to the following constraints?*

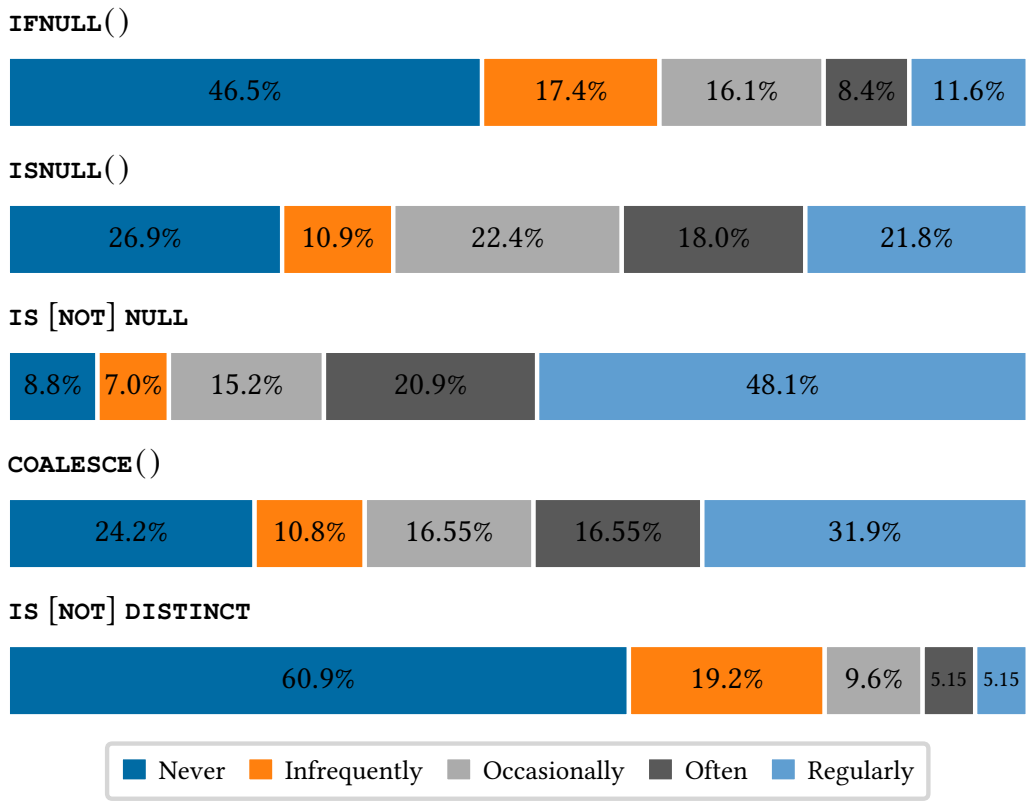


Figure 4.5

In Figure 4.5a we see how commonly SQL's null-related features are used in queries. By far the most common one is checking whether a value is **NULL**. To this end, the **IS NULL** condition, or its negation, are used by almost 70% of the users, often or regularly, and completely avoided by fewer than 10%. The next most common feature is **COALESCE**, with almost half the respondents using it at least often; other operations are less common. We remark that **ISNULL**() and **IFNULL**(), as opposed to **IS** [**NOT**] **NULL**, are non-standard functions only available in some systems.

Conclusion: **NULL** is a common occurrence in relational databases, and users are well aware of it. This manifests itself most commonly in the use of SQL's DDL, by frequently declaring some columns as **NOT NULL**. Null-related features are also fairly common in queries.

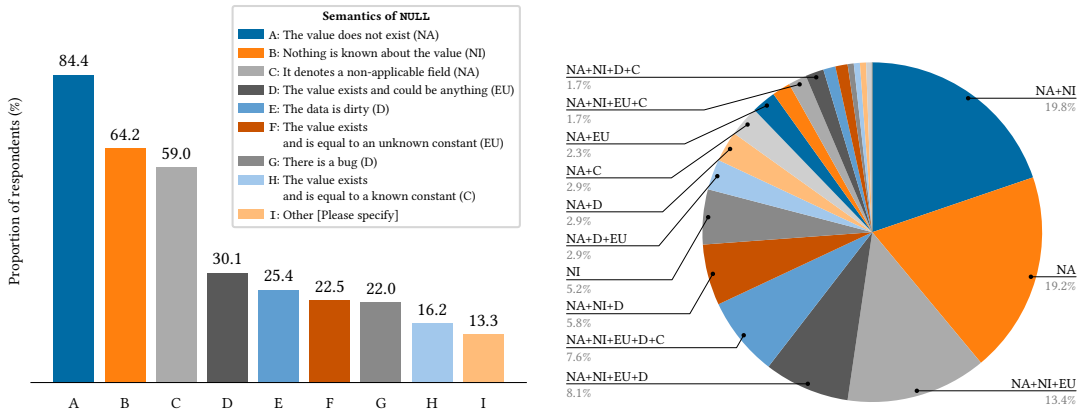
> *"The prevalence of dirty and missing data ought not to be underestimated. For many years, I was in charge of systems for data collection of medical data. Even such a regularized domain frequently had problems managing missing, dirty and suspicious data."*
>
> *A participant*

4.3 Meanings of NULLs

Now that we confirmed the ubiquity of nulls and analyzed operations on nulls, both in the DDL and the query language, we move to the next question: what does `NULL` mean? The question has been addressed in the research literature, going all the way back to Zaniolo (1984) which defined three types of nulls: non-applicable, no-information, and those representing existing but currently unknown values (two of those, non-applicable and unknown values, were adopted early by relational databases Codd (1986)). To see what options we can give to the users asking them a multiple choice question on the meaning of nulls, consider a hypothetical example: we have a table with information about employees, and the salary of the CEO is given as `NULL`. This could have different meanings:

- *Non-applicable* (**NA**). The CEO may not receive a regular salary and use another remuneration scheme. Then `NULL` indicates a field that is non-applicable, for which a value does not exist.
- *Existing unknown* (**EU**). The CEO salary cannot be disclosed for privacy reasons. In such cases, `NULL` denotes an existing but currently unknown constant.
- *Existing known constant* (**C**). The CEO salary may not be disclosed because it depends on changing financial results of company operations. Here `NULL` denotes an existing, and known, value.
- *Dirty* (**D**). The CEO may receive a regular salary but the data source from which the table is populated may have been dirty.
- *No-information* (**NI**). We may be in a situation when we know nothing at all about the reasons why that `NULL` is in the database.

While the different semantics described above are easy to understand in a particular example, a single universally accepted description is not particularly easy to formulate in natural language. Thus, we introduced some redundancy with the options we presented to the users, as shown in Table 4.1.



(a) (b)

Figure 4.6: (a) Popularity of different semantics of `NULL`; (b) Combinations of `NULL` semantics chosen by the participants.

Table 4.1

Sem.	Options presented in the survey
NA	*"the value does not exist"* *"non-applicable field"*
EU	*"the value exists and could be anything"* *"the value exists and is equal to an unknown constant"*
C	*"the value exists and is equal to a known constant"*
D	*"there is a bug"* *"the data is dirty"*
NI	*"nothing is known about the value"*

Figure 4.6a provides the frequencies with which the participants chose each option describing a possible meaning of `NULL`. The **NA** semantics is the most popular one, selected by over 85% of respondents. It is followed by **NI**, which was chosen by more than 60% of the respondents. The **EU** semantics is considered by nearly 40% of the respondents. Finally, the **D** and **C** semantics were chosen by nearly 35% and 20% of the respondents, respectively.

The data in Figure 4.6a does not provide any information about *combinations* of different semantics chosen by the participants. If we take that into account, we obtain 23 different groups, as shown in the pie chart of Figure 4.6b. Not surprisingly, **NA** dominates: alone or in combination with **NI** and **NI**+**EU**, it accounts for more than half

of all the combinations of the semantics seen. We remark that 13% of participants proposed an alternative meaning that does not coincide with any of those we have discussed (see Example 6). In light of this, our list of possible interpretations of **NULL**, while not exhaustive, covers most of the cases one meets in real-life situations.

Example 6. As an example of other meanings proposed by participants, we highlight four given propositions:

> I think of NULL as the "nothing" option of an optional (1 + x) type. If consistently interpreted in this way, there is nothing wrong with NULL (key word: consistently). (A participant)
>
> NULL is not a value, it is a state of the value. Mostly it is the state that no decision has been made yet or the decision was rescinded and there is no decision at this time. (A participant)
>
> NULL is a metadata bit that represents a statement that information about this field cannot be expressed solely through the data domain defined for the field. All of the above (and other meanings) are examples of information that cannot be directly expressed via simple domains. (A participant)
>
> As there is a bug does not completely satisfy me, i prefer to divide it into: - Data collection system (Extract) is not perfectly matching data generation system (data is not collected) - Transformations (Transform) are not OK and loses information (data is lost during process). (A participant)

A formal interpretation of those meanings is not trivial, and they do not seem to fall under the umbrella of our proposed semantics.

Conclusion: The meaning of a **NULL** value is highly varied, and therefore there is no consensus on any one specific interpretation. However, the non-applicable semantics seems to dominate.

> *"It is impossible to know what NULL means without understanding the intent of the one who put it there in the first place. Unfortunately, I've seen it used for all of the above."*
>
> *A participant*

4.4 SQL's handling of NULLs

In order to understand SQL's handling of **NULL** we organize our findings in two parts. Section 4.4.1 addresses data manipulation queries that stay within the standard textbook version of relational algebra, known as *generic* queries Abiteboul, Hull, and Vianu (1995); Arenas, Barceló, Libkin, Martens, and Pieris (2022). These types of queries do not generate new values by means of function application or aggregates. On the other hand, Section 4.4.2 focuses on how **NULL** is dealt with when applying arithmetic functions and aggregates.

Table 4.2: Proportion of answers that differ from SQL.

Operator	(a): 0 or "	(b): Constant	(c): NULL
Equality String	5.1	3.8	**22.2**
Equality Integer	3.3	2.6	**21.5**
Inequality String	**41.6**	**48.7**	11.0
Inequality Integer	**41.6**	**43.0**	8.72

4.4.1 Generic Queries

We look at three different aspects of such queries with respect to NULL handling:

(1) comparisons involving NULL,
(2) positive queries without NOT, and
(3) negative queries using NOT.

The latter are more likely to cause issues given what we know from the literature Libkin (2014, 2016b), due to the way the 3-valued logic is handled in query evaluation.

Comparisons involving NULL We would like to see whether survey participants were satisfied with the way SQL evaluated comparisons between NULL and

- a constant equal to: 0 for numerical types, and empty string for character types;
- another constant;
- another NULL.

We use the results of four build-the-answer questions (Figure 4.1a) to study both equality and disequality comparisons, and are interested in the percentage of participants who *disagree* with the way SQL handles such comparisons; the higher the value is, the more problematic such a comparison. The results, summarized in Table 4.2, show that the desired behavior does not depend on the data-type. Moreover, contrary to a popular belief neither the empty string or the integer value 0 are treated differently than any other constant (even though the former is used as a substitute for NULL in some RDBMSs). Around 20% of our respondents would want SQL to view two NULL values as equal (i.e., use syntactic equality for them), while around 10% would like them to be different. Finally around 40% would like for SQL to evaluate disequality comparisons between NULL and a constant to *false* rather than *unknown*.

When it comes to queries, we aim to learn whether the answer the participants would like to see returned in the presence of NULL

- coincides with the SQL answer, or
- contains the SQL answer, or
- is contained in the SQL answer, or

- is incomparable with the SQL answer.

Positive queries In these queries there is no negation; they are built using joins as well as **IN** and **EXISTS** subqueries. For each individual query testing a particular feature, more than 90% of our participants say that the answer should be the same as the one SQL returns. At the same time, the percentage of participants who say that the result should be the same as SQL for *every* query using those features is 84%. If we consider the whole positive fragment of SQL generic queries, and therefore also take into account the answers obtained with filtering queries on equality conditions (Table 4.2), this percentage drops to 68%. Thus even for fairly uncontroversial queries it is hard to find a uniform agreement across a query workload. But things quickly get much more problematic when we consider the interaction of **NULL** with queries that involve negation.

Queries with negation When queries have negation, the disagreement with SQL query results increases significantly. In the survey we used queries containing **NOT IN** and **NOT EXISTS** subqueries, as well the difference query expressed either via **EXCEPT** under set semantics or **EXCEPT ALL** under bag semantics. In Table 4.3 we report the views of the participants under the four possibilities shown above, with A standing for the answer a participant wanted, and SQL standing for the answer returned for by SQL. For example, the first row in that table says that for a query with a **NOT IN** subquery, 15.4% of participants agree with the result that SQL returned, while 84.5% of participants preferred an output that was a proper superset of SQL's query evaluation.

Looking at **NOT IN** and **NOT EXISTS** subqueries, we see that a large majority are unsatisfied with **NOT IN** and think that it filters out too many results, while the users are generally satisfied with **NOT EXISTS** subqueries. In fact those who are unsatisfied with **NOT IN** and want it to output more results, want it to behave in the spirit of **NOT EXISTS**. We note that **IN** subqueries follow the rules of three-valued logic (the condition in **WHERE** could evaluate to *unknown*) while **NOT EXISTS** subqueries follow the familiar two-valued logic (only *true* and *false* are possible), thus suggesting that the users are more comfortable with two truth values.

Regarding **EXCEPT**, we see that there is a significant variation in results wanted by the users. Recall that set operations treat **NULL** syntactically, so for the purpose of these operations two **NULL** values will be equal, even if testing them explicitly for equality does not return *true*. This sudden change of the semantics may well be at play here. The results for **EXCEPT ALL** seem to be more about handling duplicates than handling

Table 4.3: Proportions of respondents' answers vs SQL.

Query	A=SQL	A⊂SQL	SQL⊂A	A≠ SQL
`NOT IN`	15.4	0	**84.5**	0
`NOT EXISTS`	78.3	15.8	1.7	4.1
`EXCEPT`	53.0	10.4	**27.0**	9.6
`EXCEPT ALL`	10.7	**76.8**	8.9	3.6

nulls: analyzing them in more detail we see that most outputs preferred by survey participants match SQL answers except for duplicates. The desire to have a smaller result, expressed by over 75%, appears to be the desire to eliminate duplicates from the result.

4.4.2 Value-Inventing Queries

We now look at queries that may output values not found in a database. Examples of such would be queries whose **SELECT** clauses contain arithmetic expressions like A+B or aggregates such as **SUM** or **AVG**. For such queries we want to see how satisfied the participants are with the results that SQL produces, and whether they have alternatives. Thus, we ask about users' expectations with respect to queries that evaluate expressions such as 30+**NULL** or **NULL**+**NULL** (whose values, returned by SQL, is **NULL**), or the same expression in the **SUM** aggregate (which, perhaps confusingly for some users, would return 30 for **SUM**(30,**NULL**) and **NULL** for **SUM**(**NULL**,**NULL**)).

The results are presented in Table 4.4. In the left column we have a key feature of a query that depends on handling nulls. For example, we may ask how adding two nulls will behave (**NULL**+**NULL**) or adding a null and a constant (**NULL**+30), or how the same operations are performed under the aggregate **SUM** (**SUM**(**NULL**,30) and **SUM**(**NULL**,**NULL**) respectively). We also ask about queries in which the summation aggregate is applied to values obtained as the result of arithmetic expressions (**SUM**(**NULL**+30,**NULL**+0)), and look at queries with **GROUP BY** where grouping is done on an attribute whose value is **NULL**.

In all of these questions we ask the participants to tell us how satisfied they are with SQL answers on the 0-to-5 scale, with 0 being the least satisfied and 5 the most satisfied. The additional columns in Table 4.4 have the following meaning:

Quartiles/Median This is a box plot displaying the minimum, first quartile, median, third quartile, and maximum. We draw a box from the first quartile to the third quartile. A vertical line goes through the box at the median. The whiskers go from each quartile to the minimum or maximum. For example, looking at the first row of Table 4.4, we see that 75% of respondents gave answers 3, 4, or 5, indicating that they were rather

or very satisfied with the fact that SQL evaluated 30+**NULL** as **NULL**. The next row shows that more than 75% were satisfied with the way SQL evaluated **NULL**+**NULL**. The vertical bar in those results gives the *median* answer to the question; when it is not visible within the blue-colored rectangle, it means the median is 5 (complete satisfaction).

Mean This column provides the mean answer.

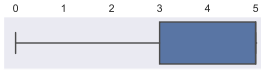
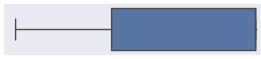
Agreement and Controversy This is our informal interpretation of the results; agreement indicates to what extent people agree with SQL answers and controversy indicates how much of the difference of opinion there is; the possible scores are high, medium, and low. The higher the average and the median score are the higher the agreement of the participants is. The larger the difference between the median value and the lower quartile value is, the higher is the controversy. For example, in the first row of Table 4.4, there is generally high level of agreement as the mean and median tend to be high, with a moderate amount of controversy as for 75% of participants they range from 3 to 5.

Alternative Participants were asked to score several tables for each query (Figure 4.1b). The "Alternative" column gives the percentage of respondents who scored the table produced by SQL lower than (at least one of) the others.

We see from Table 4.4a that the way SQL applies functions whose arguments could be **NULL** (using the rule that the value of any such function is **NULL** then) is fairly uncontroversial and accepted by most. The agreement with SQL's way of handling **NULL** slightly decreases when aggregates are applied to columns containing **NULL** (in which case the rule is not to output **NULL** but rather to ignore all nulls and then compute the aggregate). In the case of aggregates, a significantly higher proportion of participants would be more satisfied with an alternative answer. Most disagreement with SQL occurs in the case of queries with grouping on a column that may contain **NULL**. In this case SQL uses the syntactic equality of nulls, i.e., two **NULL**s are equal, which of course differs from the treatment of **NULL**s elsewhere and thus causes additional problems.

We then look at similar queries but with one important difference. Now **NULL**s are not present in the database but rather created by an **OUTER JOIN** query. Then we ask questions about similar queries involving arithmetic, aggregation, and grouping on **NULL**. Results, in the same way as before, are presented in Table 4.4b. While the trends are the same, we see three notable differences. First, the level of agreement with SQL results is higher: the median in all the cases except grouping on **NULL** moves

Table 4.4: Results obtained with (a) value-inventing queries and NULLs that occur in the database, and (b) value inventing queries and computational NULLs.

Query	Quartiles/Median	Mean	Agreement	Controversy	Alternative	
$30 + \texttt{NULL}$		3.8	high	medium	21.6%	(a)
$\texttt{NULL} + \texttt{NULL}$		4.4	high	low	2.1%	
$\texttt{SUM}(30, \texttt{NULL})$		3.4	medium	high	33.3%	
$\texttt{MIN}(30, \texttt{NULL})$		3.4	medium	medium	30.5%	
$\texttt{SUM}(\texttt{NULL}, \texttt{NULL})$		4.1	high	low	9.5%	
$\texttt{SUM}(30 + \texttt{NULL}, 0 + \texttt{NULL})$		3.6	high	high	23.2%	
$\texttt{GROUP BY}(\texttt{NULL}, \texttt{NULL})$		3.1	medium	high	29.5%	
$\texttt{NULL} + \texttt{NULL}$		4.3	high	low	7.5%	(b)
$\texttt{SUM}(30, \texttt{NULL})$		3.1	high	high	35.5%	
$\texttt{MIN}(30, \texttt{NULL})$		3.3	high	high	31.5%	
$\texttt{SUM}(\texttt{NULL})$		3.2	high	high	37.9%	
$\texttt{GROUP BY}(\texttt{NULL}, \texttt{NULL})$		2.0	low	high	63.2%	

to the highest score 5. Second, the level of controversy increases too (indicated by the widened blue shaded area in the diagram: more different scored are picked by more participants). And third, the percentage of users who prefer an answer that differs from SQL's is now much higher and peaks (at almost two thirds) for the grouping on **NULL**.

Conclusion: SQL's way of handling **NULL**s for the positive generic queries satisfies a vast majority of the users. For negative queries, the amount of dissatisfaction increases significantly. The users do not seem to like that SQL's 3VL filters out so many tuples. Users are reasonably accustomed to SQL's way of handling **NULL**s in arithmetic operations, and slightly less so in aggregation, where the rule changes and **NULL**s are ignored. Their level of dissatisfaction increases when **NULL**s come not from a database but generated by a query. What they do not like is the use of syntactic equality (saying that two **NULL**s are equal) for generating groups using **GROUP BY**.

> *"The NOT IN vs NOT EXISTS trap does nail people in real life, fairly often. I understand why it is the way it is, but I wish SQL had defined NOT IN as a syntactic transform to NOT EXISTS. GROUP BY NULL should cause the user and/or data designer to vanish from existence immediately."*
>
> *A participant*

4.5 Solutions vs Demographics

We now move to the next question: whether there exists any correlation between the demographics for our respondents, and their preferences for the behavior of SQL queries. To address this, we performed a statistical analysis of the received responses. It revealed that, somewhat surprisingly, *there is no correlation* between the users' demographics and their responses to the questions about SQL. This makes the task of fixing SQL even more daunting, if at all possible.

To perform the analysis, we defined eight binary parameters that take into account users' information (academic vs practitioner, front-end vs back-end) and the preferred semantics of **NULL**; we refer to these parameters collectively as "demographic data". We looked at how they correlate with participants' agreement with SQL answers for each of the queries, which is simply labeled as *true* or *false*. For generic queries, participants were considered to agree with SQL if they built precisely the answer SQL produces. For value-inventing queries, participants agreed with SQL if the answers produced by SQL were their most satisfying options.

The first measurement we report is the *uncertainty coefficient* – also called *proficiency coefficient* or *Theil's U* Zellner and Theil (1992) – between the demographic data and the responses for each query. This measure, for two random variables x and y, is the ratio of the mutual information of x and y to the entropy of x. Informally, this is the proportion of the information content of the agreement data that can be explained by the demographics. The demographic data would always correlate with some of the information of the agreement data for each query, simply because it is multi-dimensional while the latter, by definition, is a Boolean value. Thus, to see the real meaning of the uncertainty coefficient, we also compute its value for *randomly* generated data that uses the same distribution as the demographic and agreement data.

We also use machine learning techniques to analyze possible correlations. We study the average accuracy of a Logistic Regression McCullagh and Nelder (2019), a Random Forest Breiman (2001), and a Multi-layer Perceptron (Neural Network) Murtagh (1991) classifier trained on 80% of our data, and then assessing the remaining 20%. To evaluate the quality of the prediction, we compare the accuracy of these classifiers with a dummy classifier algorithm that always returns the highest probability outcome from the training set.

The results, for both generic and value-inventing queries, are presented in Table 4.5. Startlingly, they show that the uncertainty coefficient does not produce better results than those obtained on random data, that is, the real life correlation between demographics and answers is noticeably lower than the one obtained from random data. Likewise, the logistic regression, the random forest and the multi-layer perceptron classifiers fail to outperform the dummy classifier.

Therefore, we have strong evidence that the *demographic data is not sufficient to predict, with a reasonable degree of certainty, the user's preferred behavior of SQL queries in the presence of* **NULL**. As a consequence, the paradigm – prevalent in the academic literature – of deriving answers from the semantics of **NULL** cannot satisfy all users.

> *"Though some of these scenarios give the opportunity to address common frustrations, they have also illustrated how difficult it would be to change the way NULLs are handled without significantly changing how other SQL syntax works. Any changes could also lead to horrendous version control issues."*
>
> *A participant*

Table 4.5: Association and prediction scores obtained with (a) generic queries, (b) value inventing queries and **NULL** values in the database, and (c) value inventing queries and computational **NULL**s.

Query	Theil's U	Theil's U Random	LR Classifier	RF Classifier	MLP Classifier	Dummy Classifier	
=	48%	69%	69%	63%	65%	72%	(a)
<>	45%	73%	61%	51%	50%	52%	
NOT IN	61%	79%	84%	79%	78%	84%	
NOT EXISTS	45%	78%	78%	72%	72%	78%	
EXCEPT	51%	78%	56%	60%	55%	55%	
EXCEPT ALL	54%	92%	89%	85%	87%	89%	
$30 +$ NULL	55%	80%	84%	81%	80%	84%	(b)
NULL + NULL	51%	100%	98%	97%	97%	98%	
SUM$(30,$ NULL$)$	61%	76%	63%	64%	62%	68%	
MIN$(30,$ NULL$)$	44%	83%	70%	66%	65%	72%	
SUM(NULL, NULL)	49%	90%	92%	91%	91%	92%	
SUM$(30 +$ NULL$, 0 +$ NULL$)$	51%	79%	81%	77%	78%	81%	
GROUP BY(NULL, NULL)	57%	81%	71%	61%	56%	73%	
NULL + NULL	55%	95%	93%	90%	89%	93%	(c)
SUM$(30,$ NULL$)$	39%	76%	70%	64%	64%	70%	
MIN$(30,$ NULL$)$	40%	80%	74%	70%	68%	74%	
SUM(NULL)	49%	85%	66%	65%	68%	67%	
GROUP BY(NULL, NULL)	38%	82%	54%	46%	46%	51%	

4.6 Conclusion

Based on the results of the survey, we arrive at the following.

Conclusion 1: `NULL` values are problematic. The data management community has come to this conclusion anecdotally. To the best of our knowledge, ours is the first study that provides strong evidence that this is actually the case in practice. The study provides clues and indications about users' (dis)satisfaction with SQL features:

- The `NULL` values appearing in a single database can have a variety of meanings; we identified over 20 semantic combinations.
- SQL's three-valued logic filters out too many results, which increases users' disagreement with SQL on queries with negation.
- When `NULL` is considered as a syntactic value, as in `GROUP BY` or set operations, dissatisfaction increases.
- While `NULL` as a function argument tends to behave as expected by users, the use of `NULL` in aggregates is more controversial and would benefit from further explanations.

Conclusions 2: Handling `NULL`s in RDBMSs is an open problem. Even though we have identified concrete issues with `NULL` values, there are no satisfying solutions to handle them yet. Most existing research is not addressing the problems that database practitioners are encountering. What exacerbates the problem, is that neither demographics nor the semantics of `NULL` can explain what users want. The bottom line is that more socio-technical studies (e.g., interviews, web-data analysis, online surveys, etc.) are required to address the problem of `NULL` with users in mind.

Conclusion 3: The research spectrum needs to broaden. The research on incomplete information undertaken by the data management community is insufficient, given the problematic state of `NULL` values that practitioners continue to tolerate today. One could argue that research has not yet properly translated to practice. However, research has mostly focused on the **EU** semantics, which only a minority of users encounter, rather than the **NA** and **NI** semantics, which are much more prevalent in practice. Furthermore, for a good part of the last 40 years, research has been based on the assumption that, in the presence of `NULL` values and for a specific semantics, all users should expect the same answers. The results of our survey provide evidence that this assumption does not hold in practice. Thus, we need to tackle the problem in a different way.

The notion previously introduced in this thesis, namely the knowledge-preserving certain answers, is also based on this assumption and the missing information interpretation of incomplete data. While we initially conducted the survey to discover relevant additional knowledge to consider in order to deal with aggregation, we discovered it would not be sufficient to offer satisfactory answers to users. First, we need to refine our model to capture more interpretations of incompleteness. Second, we need a more flexible notion of answers that have users' requirements in mind.

Chapter 5

Answer notions with query-evaluation semantics

In light of the survey results, there are key shortcomings to the notions of certain answers discussed in this thesis. In previous frameworks, an incomplete database is represented by a set of complete possible worlds. Therefore, incompleteness is interpreted as missing information that can be completed. However, for relational databases, merely 2% of the users consider that missing information is the sole interpretation of incompleteness. Moreover, only 40% of the users view it as a possible interpretation. We need a notion of answers able to capture other models of incompleteness, especially the none-applicable and no-information interpretations encountered by most users (85%).

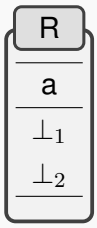
Furthermore, answers to queries on incomplete databases need to return more information, even if it means the information contained is not certain. SQL is filtering too many tuples from the answers, and in most cases, the standard notion of certain answers with the open-world assumption is even more restrictive. We need to propose alternative semantics or assumptions to provide more relevant answers to users.

In previous frameworks, incompleteness is captured on the level of database objects by the mean of a semantics function which maps an incomplete database to complete databases representing its possible interpretations. Upon this model of incompleteness and independently of the query language, each framework defines a notion of answers. Therefore, the incompleteness model does not impact query evaluation. To overcome this limitation, we model incompleteness on the level of database and query by considering a semantics function that maps a query and a database to a set of databases representing its possible answers. This model allows us to capture more interpretations of incompleteness, especially the none-applicable and no-information interpretations (see Example 7). On top of this model of incompleteness, we propose a notion of certain answers and introduce the whole-world assumption for answers on relational databases, resulting in databases containing more tuples.

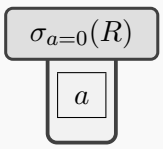
Finally, the survey results show that users have different expectations for answers on incomplete databases. Indeed even if two users agree on the interpretation of incompleteness, what they consider a satisfying answer may differ. We need a more adaptable notion of answers.

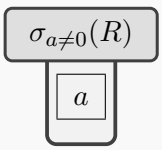
The key component to define answers is the notion of similarity between database objects. Indeed, the most pertinent answer is often defined as the database that is the least dissimilar to every other possible answer. For instance, the certain answers are based on a boolean similarity measure, namely the less-informative pre-order. To obtain a more flexible notion of answers, we propose a numerical similarity measure called regret. The regret capture the task-dependent loss a user endures when he considers a database as ground truth instead of another. The notion of answer defined with regret similarity measures is called risk-minimizing answers. We show that for some regret functions, the regret-minimizing answers coincide with the certain answers. Moreover, because regret functions are numerical, the risk-minimizing notion can define more nuanced and pertinent answers.

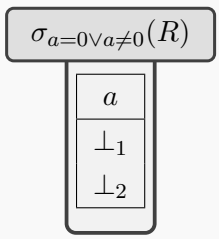
Example 7. Considere the database:



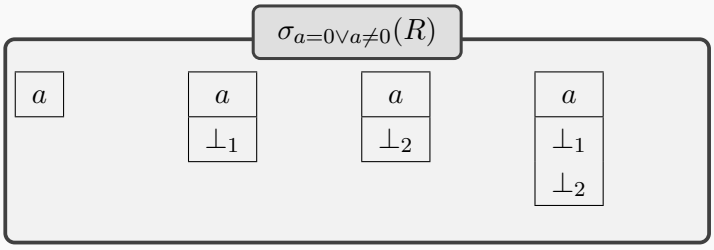
When we consider the traditional semantic (**EU**), as the query is a tautology we obtain the following answers to queries:







The problem arises when we want to capture the **NI** or **NA** semantics. Especially for the tautological query $\sigma_{a=0\vee a\neq 0}(R)$ and the **NI** semantics, we wish the set of possible answers to be:



And it is impossible to capture all those possible answers by only considering a semantic on databases.

5.1 Query-evaluation semantics

Recall that a query q from a source database domain $\mathbf{S}$ to a target database domain $\mathbf{T}$ is a mapping from the source's complete objects to the target's complete objects. For each query q, we define its *evaluation semantics* as a function mapping incomplete objects from the source to the power set of incomplete objects of the target, representing the possible answers. Moreover, the evaluation semantics has to be consistent with the source and target database domain semantics.

Definition 5.1.1 (Evaluation semantics)**.** *The* evaluation semantics *of a query q from $\mathbf{S}$ to $\mathbf{T}$ is a mapping from the source databases $\mathbb{I}_\mathbf{S}$ to the powerset of target databases $\mathbb{I}_\mathbf{T}$, ie. $\mathbb{Q}\colon \mathbb{I}_\mathbf{S} \to 2^{\mathbb{I}_\mathbf{T}}$ such that for every database $x \in \mathbb{I}_\mathbf{S}$, every complete database $c \in \mathbb{C}_\mathbf{S}$:*

(a) $[\![q([\![x]\!]_\mathbf{S})]\!]_\mathbf{T} \subseteq [\![\mathbb{Q}(x)]\!]_\mathbf{T}$
(b) $[\![q([\![c]\!]_\mathbf{S})]\!]_\mathbf{T} = [\![\mathbb{Q}(c)]\!]_\mathbf{T}$

The evaluation semantics of a query q on a database $x \in \mathbb{I}_\mathbf{S}$ contains at least all the complete answers obtained by evaluating the complete interpretations of the database x. Moreover, if the database x is complete, the evaluation semantics of q is equal to the evaluation of q on x.

In chapter 3, we introduced the notion of $\mathcal{K}$-semantics for databases. We considered the elements of the *universal knowledge* $\mathcal{K}$ as pieces of knowledge that can be applied to a database to make it potentially more informative, i.e., restrict its possible interpretations. We now consider that the knowledge in $\mathcal{K}$ can be applied to query evaluation to make it more informative, i.e., restrict the set of possible answers returned by the evaluation semantics. Such pieces of knowledge can be concatenated: applying $\omega\omega'$ means applying ω first and then applying ω' to the result of applying ω. Finally, we have the empty knowledge ϵ: applying it to any query evaluation does not change it. That is, $\mathcal{K}$ is a *monoid* as it has a binary concatenation operation $\omega\omega'$ (which we assume to be associative) and the identity ϵ satisfying $\epsilon\omega = \omega\epsilon = \omega$.

Then for a query q with evaluation semantics $\mathbb{Q}$ and a database $x \in \mathbb{I}_\mathbf{S}$, the $\mathcal{K}$-evaluation semantics under knowledge $\omega \in \mathcal{K}$ is the set $\mathbb{Q}(x, \omega)$ of possible answers in $\mathbb{I}_\mathbf{T}$ to the evaluation of q on x when ω is known. We assume that for any given evaluation semantics $\mathbb{Q}$, such a semantic function from $\mathbb{I}_\mathbf{S} \times \mathcal{K}$ to $2^{\mathbb{I}_\mathbf{T}}$ always exists, and it is such that $\mathbb{Q}(x, \epsilon) = \mathbb{Q}(x)$ for every $x \in \mathbb{I}_\mathbf{S}$. Intuitively, the empty knowledge does not give us any extra information about the semantics of the query evaluation.

As the monoid $\mathcal{K}$ contains all possible knowledge, it is reasonable to assume that the $\mathcal{K}$-operation semantics, while always existing, is not fully known. Thus, we consider sub-monoids of $\mathcal{K}$, in particular those whose knowledge increases the information content of query evaluation.

Definition 5.1.2 (Additional evaluation knowledge)**.** *We say that a sub-monoid of knowledge $\mathcal{A} \subseteq \mathcal{K}$ is an additional evaluation knowledge to an evaluation semantics $\mathbb{Q}$ iff. $\mathcal{A}$ is additional to $\mathbf{S}$ and $\mathbf{T}$ and for every database $x \in \mathbb{I}_{\mathbf{S}}$, every complete database $c \in \mathbb{C}_{\mathbf{S}}$ and every knowledge $\omega, \omega' \in \mathcal{A}$ we have:*

$$[\![\mathbb{Q}(x, \omega\omega'))]\!]^{\omega\omega'}_{\mathbf{T}} \subseteq [\![\mathbb{Q}(x, \omega))]\!]^{\omega}_{\mathbf{T}}$$

Intuitively, the additional knowledge in $\mathcal{A}$ is consistent with the evaluation semantics and can only restrict the set of possible answers returned by the evaluation semantics. Note that $\{\epsilon\}$ is an additional evaluation knowledge for every query and corresponds to the situation in which we only have "empty" knowledge.

Moreover for every q from $\mathbf{S}$ to $\mathbf{T}$ and every sub-monoid of knowledge $\mathcal{A} \subseteq \mathcal{K}$ additional to $\mathbf{S}$ and $\mathbf{T}$, we can always build a trivial evaluation semantics for q denoted $\mathbb{Q}_{\mathrm{t}}$ such that $\mathcal{A}$ is an additional evaluation knowledge. We define $\mathbb{Q}_{\mathrm{t}}$ such that for every database $x \in \mathbb{I}_{\mathbf{S}}$ and every knowledge $\omega \in \mathcal{A}$, $\mathbb{Q}_{\mathrm{t}}(x, \omega) = q([\![x]\!]^{\omega}_{\mathbf{S}})$. Informally the incompleteness of the evaluation semantics is fully contained in the incompleteness of the source database domain.

5.1.1 Evaluation-based Certain answers

Our goal is to capture answers consistent with every possible answer and under every extra knowledge.

Definition 5.1.3 (Evaluation-Based Certain answers)**.** *Let a query q from $\mathbf{S}$ to $\mathbf{T}$ with an evaluation semantics $\mathbb{Q}$. Let a database $x \in \mathbb{I}_{\mathbf{S}}$, and let $\mathcal{A}$ be an additional evaluation knowledge for $\mathbb{Q}$. The* $\mathcal{A}$-evaluation-based certain answer *to $\mathbb{Q}$ on x, denoted by* $\mathsf{cert}_{\mathcal{A}}(\mathbb{Q}, x)$*, is the most informative database with respect to $\preceq^{\mathcal{A}}$ that satisfies the following:*

$$[\![\mathbb{Q}(x, \omega)]\!]^{\omega}_{\mathbf{T}} \subseteq [\![\mathsf{cert}_{\mathcal{A}}(\mathbb{Q}, x)]\!]^{\omega}_{\mathbf{T}} \tag{5.1}$$

for every $\omega \in \mathcal{A}$.

If we consider the trivial evaluation semantics of a query q denoted $\mathbb{Q}_{\mathrm{t}}$ such that for every database $x \in \mathbb{I}_{\mathbf{S}}$ and every knowledge $\omega \in \mathcal{A}$, $\mathbb{Q}_{\mathrm{t}}(x, \omega) = q([\![x]\!]^{\omega}_{\mathbf{S}})$. Then the $\mathcal{A}$-knowledge preserving answers and the $\mathcal{A}$-evaluation-based certain answer of $\mathbb{Q}_{\mathrm{t}}$ coincides. For every database $x \in \mathbb{I}_{\mathbf{S}}$ we either have $\mathsf{cert}_{\mathcal{A}}(q, x) = \mathsf{cert}_{\mathcal{A}}(\mathbb{Q}_{\mathrm{t}}, x)$, or none exists.

The expected behavior of query answering under incomplete information is that more informative inputs yield more informative outputs. We show that this is also true for evaluation-based certain answers. Moreover, if they both exist, the evaluation-based certain answers are always less informative than certain answers based on database semantics.

Proposition 14. *Let q be a query from $\mathbf{S}$ to $\mathbf{T}$ with evaluation semantics $\mathbb{Q}$, and let $\mathcal{A}$ be an additional evaluation knowledge for $\mathbb{Q}$. Then, for every database $x, y \in \mathbb{I}_{\mathbf{S}}$ the followings hold:*

(a) If for every $\omega \in \mathcal{A}$ we have $\mathbb{Q}(x,\omega) \subseteq \mathbb{Q}(y,\omega)$, then $\mathsf{cert}_{\mathcal{A}}(\mathbb{Q}, y) \preceq^{\mathcal{A}} \mathsf{cert}_{\mathcal{A}}(\mathbb{Q}, x)$
(b) $\mathsf{cert}_{\mathcal{A}}(\mathbb{Q}, x) \preceq^{\mathcal{A}'} \mathsf{cert}_{\mathcal{A}'}(\mathbb{Q}, x)$ *for every $\mathcal{A}' \subseteq \mathcal{A}$;*
(c) If $\mathsf{cert}_{\mathcal{A}}(q, x)$ *and* $\mathsf{cert}_{\mathcal{A}}(\mathbb{Q}, x)$ *exist, then* $\mathsf{cert}_{\mathcal{A}}(\mathbb{Q}, x) \preceq^{\mathcal{A}} \mathsf{cert}_{\mathcal{A}}(q, x)$.

Recall that the information-based certain answers coincide with the knowledge preserving certain answers upon the empty knowledge $\{\epsilon\}$. There we can conclude that if both $\mathsf{cert}_{\mathcal{A}}(\mathbb{Q}, x)$ and $\mathsf{cert}_{\square}(q, x)$ exists we have $\mathsf{cert}_{\mathcal{A}}(\mathbb{Q}, x) \preceq \mathsf{cert}_{\square}(q, x)$. Therefore, while capturing more incompleteness models, the evaluation-based certain answers notion is always less informative than the information-based certain answers. However, this only holds when both notions exist. In the next section, we will see that on relational databases, there exists an evaluation semantics for which the evaluation-based certain answers exist, and the information-based ones do not.

5.2 Evaluation semantics for relational databases

The notion of database semantics is strongly tied to the missing interpretation of null, the existing unknown (**EU**) interpretation of incompleteness. With evaluation semantics, we can capture more interpretations of incompleteness, such as the no-information semantics (**NI**). Query evaluation semantics allow us to consider more possible answers, even answers inconsistent with the **EU** interpretation of null. The intuition behind the no-information meanings of null values is that nothing should be assumed about their behavior. More specifically, when a null value is involved in the resolution of a predicate, we should consider both the cases for which the predicate is true and is false as possible answers. We want to account for those two cases even if there exists no complete instance of the database which satisfies the predicate.

5.2.1 Relational Algebra with null

To define an evaluation semantics, we have to consider a query language; we introduce a variation of relational algebra. We recall the grammar of the standard relational algebra for complete databases given in subsection 3.2.1:

$$q := R \mid \pi_{\bar{\alpha}}(q) \mid \sigma_{\alpha_i=\alpha_j}(q) \mid \sigma_{\alpha_i\neq\alpha_j}(q) \\ \mid q \times q \mid q \cup q \mid q \cap q \mid q - q$$

where α_i and α_j are attributes, and $\bar{\alpha}$ is possibly empty tuple of attributes.

It is folklore that for complete databases there is no difference between the queries $\sigma_{\alpha_1=\alpha_2}\big(\sigma_{\alpha_3=\alpha_4}(R)\big)$ and $\big(\sigma_{\alpha_1=\alpha_2}(R)\big) \cap \big(\sigma_{\alpha_3=\alpha_4}(R)\big)$. Informally, the intersection between queries acts as a conjunction between selection operators, and the union acts as a disjunction. Thanks to that, relational algebra has precisely the power of *first-order logic*, despite not explicitly allowing first-order logic formulae in selection.

For incomplete databases, intersection and conjunction carry different interpretations. On the one hand, the intersection is performed on the level of relations, i.e., database objects. Therefore its evaluation should result in a database object whose semantics is equal to the intersection of the source databases' semantics. On the other hand, evaluating a selection operator involves the resolution of a formula with elements of the databases. As those elements may be null, the resolution of predicates and their conjunctions or disjunctions can be specified by an evaluation semantics. For this reason, we allow the selection operator to contain a formula with conjunctions and disjunctions.

Now consider the following SQL query returning the 'id' of each order for which there does not exist a payment entry:

```
SELECT id FROM Orders
WHERE NOT EXISTS (SELECT id FROM Payments
                  WHERE Orders.id = Payments.id)
```

For complete database this query can be written in relational algebra:

$$q = \pi_{\mathsf{id}}(\mathsf{Orders}) - \pi_{\mathsf{id}}(\mathsf{Payments})$$

However, those two queries have different interpretations for incomplete databases. On the one hand„ both $\pi_{\mathsf{id}}(\mathsf{Orders})$ and $\pi_{\mathsf{id}}(\mathsf{Payments})$ are relations ie. they are database objects. Hence the evaluation of q should result in a database object whose semantics contains the database objects in the semantics of $\pi_{\mathsf{id}}(\mathsf{Orders})$ which are not in the semantics of $\pi_{\mathsf{id}}(\mathsf{Payments})$. On the other hand, the evaluation of the SQL query involved the evaluation of the equality predicate between elements of the database. Therefore its resolution can be specified by an evaluation semantics.

We introduce the *Relational Algebra with null* language to be closer to SQL language and semantics.

Definition 5.2.1 (Relational Algebra with null)**.** *Considering a collection of predicates Γ, a relational algebra with null query q is given by the following grammar:*

$$\begin{aligned} q := R \mid q \times q \mid \pi_{\bar{\alpha}}(q) \mid \sigma_{\phi(\bar{\alpha})}(q) \\ \mid \exists_{\phi(\bar{\alpha})}(q,q) \mid \nexists_{\phi(\bar{\alpha})}(q,q) \end{aligned}$$

where $\bar{\alpha}$ is a possibly empty tuple of attributes.

$$\phi(\bar{\alpha}) := \mathcal{P}(\bar{\alpha}) \mid \phi(\bar{\alpha}) \vee \phi(\bar{\alpha}) \mid \phi(\bar{\alpha}) \wedge \phi(\bar{\alpha}) \quad \textit{with } \mathcal{P} \in \Gamma$$

For simplicity, a formula ϕ can not contain negation ($\neg$); however, as there is no constraint on the collection of predicates Γ, one can always force it to be close under negation.

The semantics of a (well-formed) relational algebra with null query q is given by inductively defining the quantity $\#\big(q(d), \bar{t}\big)$, which is the number of occurrences of a tuple $\bar{t}$ (of appropriate arity) in the result of applying q to a possibly with nulls database d.

$$\begin{aligned}
\#\big(R(d), \bar{t}\big) &= \#\big(R^d, \bar{t}\big) \\
\#\big((q \times q')(d), \bar{t}\,\bar{t}'\big) &= \#\big(q(d), \bar{t}\big) \cdot \#\big(q'(d), \bar{t}'\big) \\
\#\big(\pi_{\bar{\alpha}}(q)(d), \bar{t}\big) &= \sum_{\bar{t}' \,:\, \pi_{\bar{\alpha}}(\bar{t}') = \bar{t}} \#\big(q(d), \bar{t}'\big) \\
\#\big(\sigma_{\phi(\bar{\alpha})}(q)(d), \bar{t}\big) &= \begin{cases} \#\big(q(d), \bar{t}\big) & \textit{if } [\![\phi(\bar{\alpha})]\!]_{\bar{t}} = \mathsf{true} \\ 0 & \textit{otherwise} \end{cases} \\
\#\big(\exists_{\phi(\bar{\alpha})}(q, q')(d), \bar{t}\big) &= \begin{cases} \#\big(q(d), \bar{t}\big) & \textit{if } \exists \bar{t}' \in q'(d),\ [\![\phi(\bar{\alpha})]\!]_{\bar{t}\bar{t}'} = \mathsf{true} \\ 0 & \textit{otherwise} \end{cases} \\
\#\big(\nexists_{\phi(\bar{\alpha})}(q, q')(d), \bar{t}\big) &= \begin{cases} \#\big(q(d), \bar{t}\big) & \textit{if } \nexists \bar{t}' \in q'(d),\ [\![\phi(\bar{\alpha})]\!]_{\bar{t}\bar{t}'} = \mathsf{true} \\ 0 & \textit{otherwise} \end{cases}
\end{aligned}$$

The semantics of evaluation for a formula ϕ is given by:

$$[\![\mathcal{P}(\bar{\alpha})]\!]_{\bar{t}} = \begin{cases} \mathcal{P}(\bar{t}[\alpha]) & \textit{if } \forall \alpha_i \in \bar{\alpha},\ \bar{t}[\alpha_i] \in \mathsf{Const} \\ \bot & \textit{otherwise} \end{cases}$$

$$[\![\phi(\bar{\alpha}) \wedge \phi'(\bar{\alpha}')]\!]_{\bar{t}} = \begin{cases} \mathsf{false} & [\![\phi(\bar{\alpha})]\!]_{\bar{t}} = \mathsf{false} \vee [\![\phi'(\bar{\alpha})']\!]_{\bar{t}} = \mathsf{false} \\ \mathsf{true} & \textit{if } [\![\phi(\bar{\alpha})]\!]_{\bar{t}} = [\![\phi'(\bar{\alpha})']\!]_{\bar{t}} = \mathsf{true} \\ \bot & \textit{otherwise} \end{cases}$$

$$[\![\phi(\bar{\alpha}) \vee \phi'(\bar{\alpha}')]\!]_{\bar{t}} = \begin{cases} \mathsf{true} & [\![\phi(\bar{\alpha})]\!]_{\bar{t}} = \mathsf{true} \vee [\![\phi'(\bar{\alpha})']\!]_{\bar{t}} = \mathsf{true} \\ \mathsf{false} & \textit{if } [\![\phi(\bar{\alpha})]\!]_{\bar{t}} = [\![\phi'(\bar{\alpha})']\!]_{\bar{t}} = \mathsf{false} \\ \bot & \textit{otherwise} \end{cases}$$

Note that only the evaluation of the operators σ, $\exists$ and $\nexists$ varies when some elements of the databases are null.

The evaluation of $\exists_{\phi(\bar{\alpha})(q,q')}$ matches the SQL evaluation of the query:

```
SELECT * FROM (q)
WHERE EXISTS (SELECT * FROM (q')
              WHERE φ(...) )
```

and the evaluation of $\nexists_{\phi(\bar{\alpha})(q,q')}$ matches the SQL evaluation of the query:

```
SELECT * FROM (q)
WHERE NOT EXISTS (SELECT * FROM (q')
                  WHERE φ(...) )
```

Indeed in SQL, if the evaluation of a formula ϕ involves **NULL** the behavior of **EXISTS** and **NOT EXISTS** will be at odd. In the case of **EXISTS** the tuple will be discarded, while in the case of **NOT EXISTS** it will be kept. This behavior explain the definition of the semantics for $\nexists_{\phi(\bar{\alpha})(q,q')}$ where we check that no possible evaluation of the formula returns true. In other words, we keep the tuple even if there exists a tuple in $q'(d)$ for which the formula's evaluation returns $\bot$.

Finally, remark that we are not allowing database object operators such as $\cap, \cup$, and $-$. Indeed as mentioned earlier, those operators do not manipulate elements of the databases. Therefore their output is independent of the evaluation semantics and only depends on the semantics of the database. Our goal is to provide an evaluation semantics for the no-information interpretation of nulls, for which database semantics do not exist. Hence we can not consider those operators. However, remark that those operations can be implemented for complete databases thanks to the $\times$, $\exists$, and $\nexists$ operations.

Figure 5.1: A database d containing the relations Orders and Payments.

For every relational algebra with null query q and every relational database d, the database $q(d)$ is well-defined and is a possibly incomplete relational table. However, the straightforward evaluation of q on d given by relational algebra with null is not a consistent evaluation semantics neither for the close-world assumption nor for the open-world assumption.

Example 8. Consider the relational algebra with null query:

$$q = \nexists_{a=b}(\mathsf{Orders}, \mathsf{Payments})$$

Or the equivalent SQL query:

```
SELECT * FROM Orders
WHERE NOT EXISTS (SELECT * FROM Payments
                  WHERE a = b)
```

and the database d from Figure 5.1.
The evaluation of q on d returns the bag $q(d) = \{\!\{(2), (3)\}\!\}$.
The tuple $(1) \in$ Orders is discarded. Indeed if we choose the tuple $(1) \in$ Payments, then we obtain $[\![a = b]\!]_{(1,1)} = \mathsf{true}$.
On the other hand, we have $[\![c = b]\!]_{(2,1)} = \mathsf{false}$ and $[\![a = b]\!]_{(2,\bot)} = \bot$, hence we keep the tuple $(2) \in$ Orders. Similarly we keep tuple $(3) \in$ Orders.
With both CWA and OWA, there exists a possible interpretation of $d = \{\mathsf{Orders}, \mathsf{Payments}\}$ where $\bot \in \mathsf{Payments}^d$ is equal to 2, we denote $v(d) \in [\![d]\!]$ such interpretation.
Then the evaluation of q on $v(d)$ returns the bag $q(d) = \{\!\{(3)\}\!\}$. Indeed if we choose the tuple $(2) \in v(\mathsf{Payments})$, then we obtain $[\![a = b]\!]_{(2,2)} = \mathsf{true}$.
As $[\![\{\!\{(3)\}\!\}]\!] \not\subseteq [\![\{\!\{(2), (3)\}\!\}]\!]$ the evaluation of relational algebra with null queries is not a valid evaluation semantics for neither the CWA nor OWA.

5.2.2 CWA and OWA consistent evaluation semantics

We have to capture more possible answers to propose a valid evaluation semantics for relational algebra with null. Informally we wish to model the no-information meanings of null values. Our interpretation of the no-information meanings of null values is that nothing should be assumed about their behavior. More specifically, when a null value is involved in the resolution of a predicate, we should consider both the cases for which the predicate is true and is false as possible answers. Therefore, we wish to consider all possible resolutions of a predicate when it involves a null value. It results in a set of possible answers, which is then propagated to parent queries.

Definition 5.2.2 (Evaluation semantics for relational algebra with null)**.** *Let a relational algebra with null query q and d a bag relational database. The **NI**-evaluation semantics $q_{\mathbf{NI}}$ is given by induction:*

$$
\begin{aligned}
R_{\mathbf{NI}}(d) &= R^d \\
(q \times q')_{\mathbf{NI}}(d) &= \{a_1 \times a_2 \mid a_1 \in q_{\mathbf{NI}}(d), a_2 \in q'_{\mathbf{NI}}(d)\} \\
(\pi_{\bar{\alpha}})(q)_{\mathbf{NI}}(d) &= \{\pi_{\bar{\alpha}}(a) \mid a \in q_{\mathbf{NI}}(d)\} \\
(\sigma_{\phi(\bar{\alpha})})(q)_{\mathbf{NI}}(d) &= \left\{ a \;\middle|\; \begin{array}{l} \exists a_1 \in q_{\mathbf{NI}}(d), \forall \bar{t} \in a_1, \exists p \in \{\mathsf{true}, \mathsf{false}\}, \\ \#\big(a, \bar{t}\big) = \begin{cases} \#\big(a_1, \bar{t}\big) & \textit{if } [\![\phi(\bar{\alpha})]\!]_{\bar{t}} = \mathsf{true} \vee \big(p \wedge [\![\phi(\bar{\alpha})]\!]_{\bar{t}} = \bot\big) \\ 0 & \textit{otherwise.} \end{cases} \end{array} \right\} \\
(\exists_{\phi(\bar{\alpha})})(q, q')_{\mathbf{NI}}(d) &= \left\{ a \;\middle|\; \begin{array}{l} \exists a_1 \in q_{\mathbf{NI}}(d), \exists a_2 \in q'_{\mathbf{NI}}(d), \forall \bar{t} \in a_1, \exists p \in \{\mathsf{true}, \mathsf{false}\}, \\ \#\big(a, \bar{t}\big) = \begin{cases} \#\big(a_1, \bar{t}\big) & \textit{if } \exists \bar{t}' \in a_2 \textit{ such that} \\ & [\![\phi(\bar{\alpha})]\!]_{\bar{t}\bar{t}'} = \mathsf{true} \vee \big(p \wedge [\![\phi(\bar{\alpha})]\!]_{\bar{t}\bar{t}'} = \bot\big) \\ 0 & \textit{otherwise.} \end{cases} \end{array} \right\} \\
(\nexists_{\phi(\bar{\alpha})})(q, q')_{\mathbf{NI}}(d) &= \left\{ a \;\middle|\; \begin{array}{l} \exists a_1 \in q_{\mathbf{NI}}(d), \exists a_2 \in q'_{\mathbf{NI}}(d), \forall \bar{t} \in a_1, \exists p \in \{\mathsf{true}, \mathsf{false}\}, \\ \#\big(a, \bar{t}\big) = \begin{cases} \#\big(a_1, \bar{t}\big) & \textit{if } \nexists \bar{t}' \in a_2 \textit{ such that} \\ & [\![\phi(\bar{\alpha})]\!]_{\bar{t}\bar{t}'} = \mathsf{true} \vee \big(p \wedge [\![\phi(\bar{\alpha})]\!]_{\bar{t}\bar{t}'} = \bot\big) \\ 0 & \textit{otherwise.} \end{cases} \end{array} \right\}
\end{aligned}
$$

Example 9. Consider the relational algebra with null query:

$$q = \nexists_{a=b}(\mathsf{Orders}, \mathsf{Payments})$$

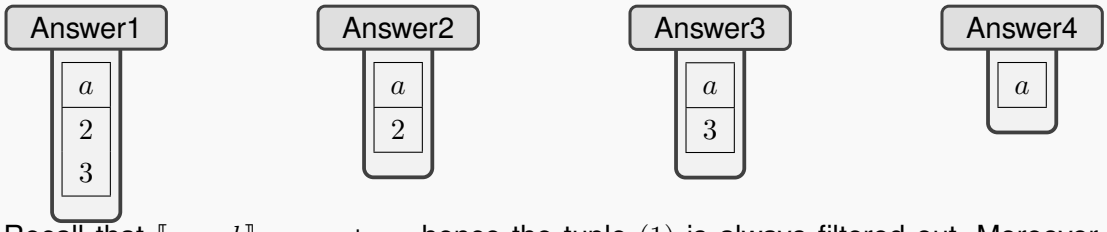
Or the equivalent SQL query:

```
SELECT * FROM Orders
```

```
WHERE NOT EXISTS (SELECT * FROM Payments
                  WHERE a = b)
```

and the database d from figure 5.1.

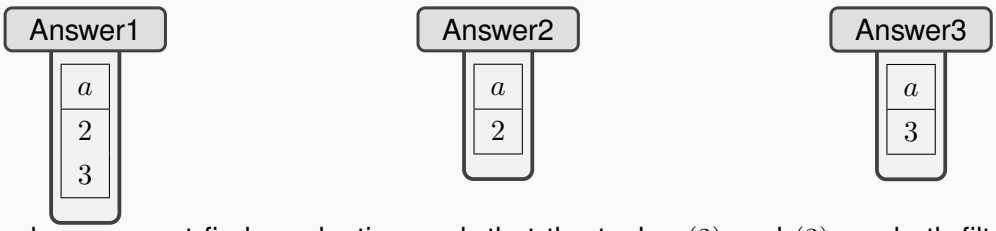
Then the set of possible answers given by the evaluation semantics $q_{\mathbf{NI}}$ is:



Recall that $[\![a = b]\!]_{(1,1)} = \mathsf{true}$, hence the tuple (1) is always filtered out. Moreover, recall that $[\![a = b]\!]_{(2,1)} = \mathsf{false}$ and $[\![a = b]\!]_{(2,\perp)} = \perp$, therefore there does not exists a tuple t in Payments such that $[\![a = b]\!]_{(2,t)} = \mathsf{true}$.

But as $[\![a = b]\!]_{(2,\perp)} = \perp$ we have to consider the cases when (2) is filtered out and when it is kept in the answers. Similarly, with the tuple (3), which gives us the previously given set of possible answers.

On the other hand, the set of answers given by the semantics of d, $q([\![d]\!]_{\mathsf{CWA}})$ is



Indeed we can not find a valuation such that the tuples (2) and (3) are both filtered out. However, we notice that with the open-world-assumption on the target database domain, we retain that:

$$[\![q([\![d]\!]_{\mathsf{CWA}})]\!]_{\mathsf{OWA}} \subseteq [\![q_{\mathbf{NI}}(d)]\!]_{\mathsf{OWA}}$$

Finally, we obtain that the information-based certain answers $\mathrm{cert}_{\square}(q, d)$ are equal to a $\{\!|\perp|\!\}$ and that the $\{\epsilon\}$-evaluation-based certain answers are equal to the empty bag.

Proposition 15. *For every relational algebra with null query q from the source relational database domain $\mathbf{S}$ with the* CWA *semantics to the target relational database domain with* OWA*, the mapping $q_{\mathbf{NI}}$ is an evaluation semantics of q. Moreover, the $\{\epsilon\}$-evaluation-based certain answers,* $\mathrm{cert}_{\{\epsilon\}}(q_{\mathbf{NI}}, d)$ *exists and can be computed in polynomial time.*

Proof. The proof of this proposition is included in the proof of theorem 8. □

Employees

First Name	Second Name	Salary
John	$\perp_1$	2000
$\perp_1$	Peter	1000

Figure 5.2: A database d containing the relation Employees.

We have proposed the first notion of answers compatible with the no-information interpretation of null values. Moreover, we can compute the evaluation-based certain answers with this incompleteness model in polynomial time. However, as the evaluation-based certain answers are less informative than the information-based ones, it does not seem to be a practical notion. But this statement only holds if both the answer notions exist at the same time.

5.2.3 Whole world assumption

Recall that for relational databases, we consider the *open world assumption* over the target database domain. With OWA, an answer contains only true facts but can also miss arbitrary many true facts. Therefore the more facts an answer contains, the more information it carries. Now consider a semantics called *whole world assumption*, such that an answer contains all the true facts but can also contain arbitrary many wrong facts. Then the fewer facts an answer contains, the more information it carries.

Formally the whole-world assumption semantics is defined as follows:

$$[\![d]\!]_{\text{WWA}} = \{\textit{complete } d' \mid d' \subseteq v(d) \textit{ for a } d\textit{-complete valuation } v\}$$

Under WWA, a complete database is a possible interpretation of d, if there exists a valuation v such that it is a subset of $v(d)$

However, when considered on the target databases, the WWA semantic shares a similar caveat as the CWA semantics. The information-based certain answers do not exist even for simple queries, as illustrated in the following Example 10.

Example 10. Consider the generic query q returning the salary of all employees such that the first name is equal to the second name:

$$q = \pi_{\text{Salary}}\big(\sigma_{\text{First Name}=\text{Second Name}}\big)(\text{Employees})$$

on the database given in Figure 5.2

With the valuation $v\colon \bot_1 \mapsto$ Jane, the answer is empty. Therefore if we consider the OWA on the target, the information-based certain answers would exist and be empty.
On the other hand, the set of answers given by the semantics is

$$q([\![d]\!]_{\mathsf{CWA}}) = \{\{\!|1000|\!\}, \{\!|2000|\!\}, \{\!|\}\!\}\}$$

The bag $\{\!|1000, 2000|\!\}$ is not a possible answer of the semantics, because $\bot_1$ can not be evaluated to 'Peter' and 'John' at the same time.
To compute the information-based certain answers, we first have to find the databases which are less informative than all possible answers with the WWA semantic. A trivial one would be $\{\!|1000, 2000|\!\}$, and any superset of it would also be less informative than all possible answers. However, the database $\{\!|\bot_i|\!\}$ is also less informative than all possible answers.
Recall that the certain answer is the most informative database, which is less informative than all possible answers. The problem is that the information content of $\{\!|1000, 2000|\!\}$ and $\{\!|\bot_i|\!\}$ is incomparable. Therefore, the information-based certain answers do not exist.

Intuitively the relational database objects are not able to carry precise enough information. The resulting pre-order is often too sparse, and the greatest lower bound of the set of possible answers do not exist. On the one hand, the easiest way to ensure the existence of certain answers is to use a more expressive model of incompleteness for the target database. However, it comes with a cost, and it pushes us further away from applicability (conditional tables are such a model Imielinski and Lipski (1984)). On the other hand, we can allow more permissive semantics of incompleteness for the source databases. Indeed it will increase the size of the set of possible answers to consider and help toward the existence of a greatest lower bound.

Theorem 8. *For every relational algebra with null query q from the source relational database domain $\mathbb{S}$ with the* CWA *semantics to the target relational database domain with* WWA*, the mapping $q_{\mathbf{NI}}$ is an evaluation semantics of q. Moreover, the $\{\epsilon\}$-evaluation-based certain answers,* $\mathrm{cert}_{\{\epsilon\}}(q_{\mathbf{NI}}, d)$ *exists and can be computed in polynomial time.*

Informally, with the whole world assumption WWA, the evaluation-based certain answers are the smallest database that contains all the true facts. On the other hand, with the open world assumption, OWA, the evaluation-based certain answers are the largest database that only contains true facts.

Example 11. In example 10, the $q_{\mathbf{NI}}$ evaluation semantics enforces that any predicate evaluation involving $\perp_1$ have to be considered true resp. false at some point. Therefore the set of possible answers $q_{\mathbf{NI}}(d)$ contains $\{\!|1000|\!\}$,$\{\!|2000|\!\}$, the empty set, but also $\{\!|1000, 2000|\!\}$. It resolves the previous issue regarding the existence of a greatest lower bound for the WWA semantics, and the evaluation-based certain answers are $\{\!|1000, 2000|\!\}$.

By considering a different assumption on the target database domain, namely the whole world assumption, we have been able to offer a notion of answers for relational database with the no-information interpretation of nulls which filters-out less tuple than previous notions.

5.2.4 Proof of Theorem 8

Theorem 2. *For every relational algebra with null query q from the source relational database domain $\mathbf{S}$ with the* CWA *semantics to the target relational database domain with* WWA*, the mapping $q_{\mathbf{NI}}$ is an evaluation semantics of q. Moreover, the $\{\epsilon\}$-evaluation-based certain answers,* $\mathrm{cert}_{\{\epsilon\}}(q_{\mathbf{NI}}, d)$ *exists and can be computed in polynomial time.*

Proof of Theorem 8. In order to prove the theorem, we inductively define two new query evaluation procedures for relational algebra with null query. We will prove that those evaluation procedures respectively return the evaluation-based certain answers with WWA and the evaluation-based certain answers with OWA.

For every relational algebra with null query q, we define the evaluation procedure q_{true} resp. q_{false} such that their semantics is given by inductively defining the quantity $\#\big(q_{\mathsf{true}}(d), \bar{t}\big)$ resp. $\#\big(q_{\mathsf{false}}(d), \bar{t}\big)$, which is the number of occurrences of a tuple $\bar{t}$ (of appropriate arity) in the result of applying q_{true} resp. q_{false} to a database d.

$$
\begin{aligned}
\#\big(R_*(d), \bar{t}\big) &= \#\big(R^d, \bar{t}\big) \\
\#\big((q \times q')_*(d), \bar{t}\,\bar{t}'\big) &= \#\big(q_*(d), \bar{t}\big) \cdot \#\big(q'_*(d), \bar{t}'\big) \\
\#\big(\pi_{\bar{\alpha}}(q)_*(d), \bar{t}\big) &= \sum_{\bar{t}' \colon \pi_{\bar{\alpha}}(\bar{t}') = \bar{t}} \#\big(q_*(d), \bar{t}'\big) \\
\#\big(\sigma_{\phi(\bar{\alpha})}(q)_{\mathsf{true}}(d), \bar{t}\big) &= \begin{cases} \#\big(q_{\mathsf{true}}(d), \bar{t}\big) & \text{if } [\![\phi(\bar{\alpha})]\!]_{\bar{t}} = \mathsf{true} \vee [\![\phi(\bar{\alpha})]\!]_{\bar{t}} = \bot \\ 0 & \text{otherwise} \end{cases} \\
\#\big(\sigma_{\phi(\bar{\alpha})}(q)_{\mathsf{false}}(d), \bar{t}\big) &= \begin{cases} \#\big(q_{\mathsf{false}}(d), \bar{t}\big) & \text{if } [\![\phi(\bar{\alpha})]\!]_{\bar{t}} = \mathsf{true} \\ 0 & \text{otherwise} \end{cases} \\
\#\big(\exists_{\phi(\bar{\alpha})}(q, q')_{\mathsf{true}}(d), \bar{t}\big) &= \begin{cases} \#\big(q_{\mathsf{true}}(d), \bar{t}\big) & \text{if } \exists \bar{t}' \in q'_{\mathsf{true}}(d),\ [\![\phi(\bar{\alpha})]\!]_{\overline{t\bar{t}'}} = \mathsf{true} \vee [\![\phi(\bar{\alpha})]\!]_{\overline{t\bar{t}'}} = \bot \\ 0 & \text{otherwise} \end{cases} \\
\#\big(\exists_{\phi(\bar{\alpha})}(q, q')_{\mathsf{false}}(d), \bar{t}\big) &= \begin{cases} \#\big(q_{\mathsf{false}}(d), \bar{t}\big) & \text{if } \exists \bar{t}' \in q'_{\mathsf{false}}(d),\ [\![\phi(\bar{\alpha})]\!]_{\overline{t\bar{t}'}} = \mathsf{true} \\ 0 & \text{otherwise} \end{cases} \\
\#\big(\nexists_{\phi(\bar{\alpha})}(q, q')_{\mathsf{true}}(d), \bar{t}\big) &= \begin{cases} \#\big(q_{\mathsf{true}}(d), \bar{t}\big) & \text{if } \nexists \bar{t}' \in q'_{\mathsf{false}}(d),\ [\![\phi(\bar{\alpha})]\!]_{\overline{t\bar{t}'}} = \mathsf{true} \\ 0 & \text{otherwise} \end{cases} \\
\#\big(\nexists_{\phi(\bar{\alpha})}(q, q')_{\mathsf{false}}(d), \bar{t}\big) &= \begin{cases} \#\big(q_{\mathsf{false}}(d), \bar{t}\big) & \text{if } \nexists \bar{t}' \in q'_{\mathsf{true}}(d),\ [\![\phi(\bar{\alpha})]\!]_{\overline{t\bar{t}'}} = \mathsf{true} \vee [\![\phi(\bar{\alpha})]\!]_{\overline{t\bar{t}'}} = \bot \\ 0 & \text{otherwise} \end{cases}
\end{aligned}
$$

Notice that the computation of both q_{false} and q_{true} can be done in polynomial time as we do not need to consider a set of possible answers anymore.

Corollary 1. *For every relational algebra with null query q and database d, for every database $a \in q_{\mathbf{NI}}(d)$ we have*

$$\mathsf{cert}_{\mathsf{OWA}}(q_{\mathbf{NI}}, d) \subseteq a \subseteq \mathsf{cert}_{\mathsf{WWA}}(q_{\mathbf{NI}}, d)$$

where. $\mathsf{cert}_{\mathsf{OWA}}(q_{\mathbf{NI}}, d)$ *resp.* $\mathsf{cert}_{\mathsf{WWA}}(q_{\mathbf{NI}}, d)$ *denotes the $\{\epsilon\}$-evaluation based certain answers with the* OWA*, resp.* WWA *semantics.*

Proof of Corollary 1. Recall that the evaluation-based certain answers $\mathsf{cert}_*(q_{\mathbf{NI}}, x)$, is the most informative database with respect to $\preceq^*$ that satisfies the following:

$$[\![q_{\mathbf{NI}}(d)]\!]_* \subseteq [\![\mathsf{cert}_*(q_{\mathbf{NI}}, d)]\!]_*$$

In other words, $\mathsf{cert}_*(q_{\mathbf{NI}}, d) = \mathsf{glb}_{\preceq^*} q_{\mathbf{NI}}(d)$.

Recall the definition of the OWA semantics:

$$[\![d]\!]_{\mathsf{OWA}} = \{\text{complete } d' \mid v(d) \subseteq d' \text{ for a } d\textit{-complete} \text{ valuation } v\}\,.$$

Therefore for every database d, d' we have $d \preceq^{\mathsf{OWA}} d'$ iff. $d \subseteq d'$ (see Lemma 1 from Chapter 3 for a complete proof).
Which gives us $\mathsf{cert}_{\mathsf{OWA}}(q_{\mathbf{NI}}, d) = \mathsf{glb}_{\preceq^{\mathsf{OWA}}} q_{\mathbf{NI}}(d) = \mathsf{glb}_{\subseteq} q_{\mathbf{NI}}(d)$. Especially, we obtain that for every database $a \in q_{\mathbf{NI}}(d)$ we have $\mathsf{cert}_{\mathsf{OWA}}(q_{\mathbf{NI}}, d) \subseteq a$.

Now recall the definition of the WWA semantics:

$$[\![d]\!]_{\mathsf{WWA}} = \{\textit{complete } d' \mid d' \subseteq v(d) \textit{ for a } d\textit{-complete valuation } v\}$$

Therefore for every database d, d' we have $d \preceq^{\mathsf{WWA}} d'$ iff. $d \supseteq d'$ (see Lemma 1 from chapter 3 for a proof sketch).
Which gives us $\mathsf{cert}_{\mathsf{WWA}}(q_{\mathbf{NI}}, d) = \mathsf{glb}_{\preceq^{\mathsf{WWA}}} q_{\mathbf{NI}}(d) = \mathsf{glb}_{\supseteq} q_{\mathbf{NI}}(d)$. Especially, we obtain that for every database $a \in q_{\mathbf{NI}}(d)$ we have $\mathsf{cert}_{\mathsf{WWA}}(q_{\mathbf{NI}}, d) \supseteq a$.

And we can conclude

$$\mathsf{cert}_{\mathsf{OWA}}(q_{\mathbf{NI}}, d) \subseteq a \subseteq \mathsf{cert}_{\mathsf{WWA}}(q_{\mathbf{NI}}, d)$$

■

As $q_{\mathsf{true}}(d)$ and $q_{\mathsf{false}}(d)$ are possible answers from $q_{\mathbf{NI}}(d)$ and by definition of the evaluation-based certain answers we immediately have

$$\mathsf{cert}_{\mathsf{OWA}}(q_{\mathbf{NI}}, d) \subseteq q_{\mathsf{false}}(d)$$
$$\mathsf{cert}_{\mathsf{WWA}}(q_{\mathbf{NI}}, d) \supseteq q_{\mathsf{true}}(d)$$

We wish to prove inductively that for every relational algebra with null query q and every database d we have $q_{\mathsf{true}}(d) = \mathsf{cert}_{\mathsf{WWA}}(q_{\mathbf{NI}}, d)$ and $q_{\mathsf{false}}(d) = \mathsf{cert}_{\mathsf{OWA}}(q_{\mathbf{NI}}, d)$.

Proof of $\mathsf{cert}_{\mathsf{WWA}}(\sharp_{\phi(\bar{\alpha})}(q, q')_{\mathbf{NI}}, d) = \sharp_{\phi(\bar{\alpha})}(q, q')_{\mathsf{true}}(d)$

Proof. We assume that $\mathsf{cert}_{\mathsf{WWA}}(q_{\mathbf{NI}}, d) = q_{\mathsf{true}}(d)$ and $\mathsf{cert}_{\mathsf{OWA}}(q'_{\mathbf{NI}}, d) = q'_{\mathsf{false}}(d)$.

If $t \in \mathsf{cert}_{\mathsf{WWA}}(Q_{\mathbf{NI}}, d)$, then there exists $a \in Q_{\mathbf{NI}}(d)$ such that $t \in a$. More precisely there exists $a_1 \in q_{\mathbf{NI}}(d)$ and $a_2 \in q'_{\mathbf{NI}}(d)$ and $p \in \{\mathsf{true}, \mathsf{false}\}$ such that $t \in a_1$ and there not exists $t' \in a_2$ such that $[\![\phi(\bar{\alpha})]\!]_{\overline{tt'}} = \mathsf{true} \vee \big(p \wedge [\![\phi(\bar{\alpha})]\!]_{\overline{tt'}} = \bot\big)$.
As we can choose p to be false it gives us, if $t \in \mathsf{cert}_{\mathsf{WWA}}(Q_{\mathbf{NI}}, d)$, then there exists

$a_1 \in q_{\mathbf{NI}}(d)$ and $a_2 \in q'_{\mathbf{NI}}(d)$ such that $t \in a_1$ and there not exists $t' \in a_2$ such that $[\![\phi(\bar{\alpha})]\!]_{\overline{tt'}} = \mathsf{true}$
Which is equivalent to, if $t \in \mathsf{cert}_{\mathsf{WWA}}(Q_{\mathbf{NI}}, d)$, then there exists $a_1 \in q_{\mathbf{NI}}(d)$ and $a_2 \in q'_{\mathbf{NI}}(d)$ such that $t \in a_1$ and for all $t' \in a_2$, $[\![\phi(\bar{\alpha})]\!]_{\overline{tt'}} = \mathsf{false} \vee [\![\phi(\bar{\alpha})]\!]_{\overline{tt'}} = \bot$.

By inductive assumption, $q_{\mathsf{true}}(d) = \mathsf{cert}_{\mathsf{WWA}}(q_{\mathbf{NI}}, d)$. And with corollary 1 (For every database $a \in q_{\mathbf{NI}}(d)$ we have $a \subseteq \mathsf{cert}_{\mathsf{WWA}}(q_{\mathbf{NI}}, d)$.)
We obtain that for every $a \in q_{\mathbf{NI}}(d)$, $q_{\mathsf{true}}(d) \supseteq a$.
Therefore if there exists $a \in q_{\mathbf{NI}}(d)$ such that $t \in a$ then we have $t \in q_{\mathsf{true}}(d)$.

Then we have that if $t \in \mathsf{cert}_{\mathsf{WWA}}(Q_{\mathbf{NI}}, d)$, then $t \in q_{\mathsf{true}}(d)$ and there exists $a_2 \in q'_{\mathbf{NI}}(d)$ such that for all $t' \in a_2$, $[\![\phi(\bar{\alpha})]\!]_{\overline{tt'}} = \mathsf{false} \vee [\![\phi(\bar{\alpha})]\!]_{\overline{tt'}} = \bot$.

By inductive assumption we have $q'_{\mathsf{false}}(d) = \mathsf{cert}_{\mathsf{OWA}}(q'_{\mathbf{NI}}, d)$. And with corollary 1 (For every database $a \in q_{\mathbf{NI}}(d)$ we have $\mathsf{cert}_{\mathsf{OWA}}(q_{\mathbf{NI}}, d) \subseteq a$.)
We obtain that for every $a \in q'_{\mathbf{NI}}(d)$, $q'_{\mathsf{false}}(d) \subseteq a$.
Therefore for every tuple $t \in q'_{\mathsf{false}}(d)$ we have that for every $a \in q'_{\mathbf{NI}}(d)$, $t \in a$.

Then we have that if $t \in \mathsf{cert}_{\mathsf{WWA}}(Q_{\mathbf{NI}}, d)$, then $t \in q_{\mathsf{true}}(d)$ and for all $t' \in q_{\mathsf{false}}(d)$, $[\![\phi(\bar{\alpha})]\!]_{\overline{tt'}} = \mathsf{false} \vee [\![\phi(\bar{\alpha})]\!]_{\overline{tt'}} = \bot$.

And we can conclude that $t \in \nexists_{\phi(\bar{\alpha})}(q, q')_{\mathsf{true}}(d)$ which gives us:

$$\mathsf{cert}_{\mathsf{WWA}}(\nexists_{\phi(\bar{\alpha})}(q, q')_{\mathbf{NI}}, d) = \nexists_{\phi(\bar{\alpha})}(q, q')_{\mathsf{true}}(d)$$

■

Proof of $\mathsf{cert}_{\mathsf{OWA}}(\nexists_{\phi(\bar{\alpha})}(q, q')_{\mathbf{NI}}, d) = \nexists_{\phi(\bar{\alpha})}(q, q')_{\mathsf{false}}(d)$

Proof. We assume that $\mathsf{cert}_{\mathsf{OWA}}(q_{\mathbf{NI}}, d) = q_{\mathsf{false}}(d)$ and $\mathsf{cert}_{\mathsf{WWA}}(q'_{\mathbf{NI}}, d) = q'_{\mathsf{true}}(d)$.

If $t \in Q_{\mathsf{false}}(d)$, then, $t \in q_{\mathsf{false}}(d)$ and there not exists $t' \in q'_{\mathsf{true}}(d)$ such that $[\![\phi(\bar{\alpha})]\!]_{\overline{tt'}} = \mathsf{true} \vee [\![\phi(\bar{\alpha})]\!]_{\overline{tt'}} = \bot$.
Which is equivalent to, if $t \in Q_{\mathsf{false}}(d)$, then, $t \in q_{\mathsf{false}}(d)$ and for every $t' \in q'_{\mathsf{true}}(d)$ we have $[\![\phi(\bar{\alpha})]\!]_{\overline{tt'}} = \mathsf{false}$.

By inductive assumption we have $q_{\mathsf{false}}(d) = \mathsf{cert}_{\mathsf{OWA}}(q_{\mathbf{NI}}, d)$. And with corollary 1 (For every database $a \in q_{\mathbf{NI}}(d)$ we have $\mathsf{cert}_{\mathsf{OWA}}(q_{\mathbf{NI}}, d) \subseteq a$.)
We obtain that for every $a \in q_{\mathbf{NI}}(d)$, $q_{\mathsf{false}}(d) \subseteq a$.
Therefore for every tuple $t \in q_{\mathsf{false}}(d)$ we have that for every $a \in q_{\mathbf{NI}}(d)$, $t \in a$.

Then we have that if $t \in Q_{\mathsf{false}}(d)$, then for every $a \in q_{\mathbf{NI}}(d)$, $t \in a$, and for every $t' \in q'_{\mathsf{true}}(d)$ we have $[\![\phi(\bar{\alpha})]\!]_{\overline{tt'}} = \mathsf{false}$.

By inductive assumption, $q'_{\mathsf{true}}(d) = \mathsf{cert}_{\mathsf{WWA}}(q'_{\mathbf{NI}}, d)$. And with corollary 1 (For every database $a \in q_{\mathbf{NI}}(d)$ we have $a \subseteq \mathsf{cert}_{\mathsf{WWA}}(q_{\mathbf{NI}}, d)$.)
We obtain that for every $a \in q'_{\mathbf{NI}}(d)$, $q'_{\mathsf{true}}(d) \supseteq a$.
Therefore if there exists $a \in q'_{\mathbf{NI}}(d)$ such that $t \in a$ then we have $t \in q'_{\mathsf{true}}(d)$.

Then we have that if $t \in Q_{\mathsf{false}}(d)$, then for every $a \in q_{\mathbf{NI}}(d)$, $t \in a$, for every $a' \in q'_{\mathbf{NI}}(d)$ and for every $t' \in a'$ we have $[\![\phi(\bar{\alpha})]\!]_{\overline{tt'}} = \mathsf{false}$.
Hence if $t \in Q_{\mathsf{false}}(d)$, then for every $a \in \nexists_{\phi(\bar{\alpha})}(q, q')_{\mathbf{NI}}(d)$, $t \in a$.
By defintion of $\mathsf{cert}_{\mathsf{OWA}}(\nexists_{\phi(\bar{\alpha})}(q, q')_{\mathbf{NI}}, d)$ we can conclude that $tt \in \mathsf{cert}_{\mathsf{OWA}}(\nexists_{\phi(\bar{\alpha})}(q, q')_{\mathbf{NI}}, d)$ which gives us:

$$\mathsf{cert}_{\mathsf{OWA}}(\nexists_{\phi(\bar{\alpha})}(q, q')_{\mathbf{NI}}, d) = \nexists_{\phi(\bar{\alpha})}(q, q')_{\mathsf{false}}(d)$$

■

Other cases are omitted because they are similar. QED

We have used the evaluation semantics to provide a notion of certain answers for the **NI** interpretation of nulls. Moreover, with this interpretation, we can consider a target semantics, for which the evaluation-based certain answers exist for a large class of queries and which filters fewer tuples than the previously used one. Finally, we showed that the notion of **NI**-evaluation based certain answers with either the OWA or WWA semantics is easy to compute for a large class of relational queries.

However, one of the shortcomings of the existing techniques to define relevant answers on incomplete databases is that they commonly assume that if users agree on an interpretation of their incompleteness, they should also agree on what a relevant answer to a query is. Results from the survey presented in Chapter 4 provide evidence that this assumption does not hold for most users (Figure 4.5). Even if the evaluation-based certain answers notion offers more flexibility, as a user can choose between the open or whole world assumption on the target databases domain, we argue that it is insufficient.

5.3 Risk minimizing answers

The problem with the notion of certain answers is that it can only provide Manichean answers. Indeed certain answers have to be less informative than every possible answer. And the less-informative pre-order is absolute; a database is either less informative, incomparable, or more informative. The notion does not allow for any nuances. While additional knowledge can explain why a database is less informative or not, it can not help to quantify the information dissimilarity between two databases.

As an illustration, consider the situation from Example 10. The evaluation-based certain answers are completely different depending on the semantics we consider on the target database domain. With the open world assumption, the certain answers are the empty set, and with the whole world assumption, the certain answers are $\{\!|2000, 1000|\!\}$. A user may be more satisfied with an answer in between, such as $\{\!|\bot_i|\!\}$. However, to the best of our knowledge, no existing semantics or incompleteness models can produce this answer.

The problem comes from the notion of similarity we consider between database objects. Indeed, the most pertinent answer is often defined as the database that is the least dissimilar to every other possible answer. For instance, the certain answers are based on a boolean similarity measure, namely the less-informative pre-order. To obtain a more flexible notion of answers, we need to propose a numerical similarity measure.

Based on this observation, we introduce the notion of *regret function* $\mu : \mathbb{I}_{\mathbf{T}} \times \mathbb{I}_{\mathbf{T}} \times \mathcal{O} \to [0, \infty]$,. For any two databases $x, y \in \mathbb{I}_{\mathbf{T}}$, and any extra-knowledge $\omega \in \mathcal{A}$, the regret function returns a value $\mu(x, y, \omega)$ representing the degree of regret that a user would have if deciding according to the information contained in x instead of the information contained in y while also knowing the extra knowledge ω. Therefore for every $x \in \mathbb{I}_{\mathbf{T}}$ and every extra piece of knowledge $\omega \in \mathcal{A}$ we have $\mu(x, x, \omega) = 0$.

The regret functions acts as numerical measures of dissimilarity between possible answers. They also are user-dependent functions. Therefore, it allows us to consider users' preferences.

With the notion of regret, we can now assign a *risk score* to a database we wish to use as an answer to a query on a database. We define the risk associated with an answer a as the maximal regret a user would have if the real answer to q on x is different.

Definition 5.3.1 (Risk of an answers)**.** *Let a query q in from $\mathbf{S}$ to $\mathbf{T}$ with $\mathbb{Q}$ its evaluation semantics. Let $x \in \mathbb{I}_{\mathbf{S}}$, and let $\mathcal{A}$ be additional evaluation knowledge for $\mathbb{Q}$. The risk of a database $a \in \mathbb{I}_{\mathbf{T}}$ as an answer to $\mathbb{Q}$ on x with the evaluation knowledge $\mathcal{A}$ is the value:*

$$\mathsf{risk}_{\mathcal{A}}(\mathbb{Q}, x)[a] = \sup\left\{\mu(a, y, \omega) \mid \omega \in \mathcal{A}, y \in \mathbb{Q}(x, \omega)\right\}$$

Remark that the risk value does not have to rely on the semantics of the database domain. It is dependent on the chosen regret function. This allows us to consider other notions of answers upon incomplete databases.

Definition 5.3.2 (Risk minimizing answer)**.** *Let a query q in from $\mathbf{S}$ to $\mathbf{T}$ with $\mathbb{Q}$ its evaluation semantics. Let $x \in \mathbb{I}_{\mathbf{S}}$, and let $\mathcal{A}$ be additional evaluation knowledge for $\mathbb{Q}$. When it exists the risk-minimizing answer is defined as:*

$$\mathsf{best}_{\mathcal{A}}(\mathbb{Q}, x) = \arg\min \left\{ \mathsf{risk}_{\mathcal{A}}(\mathbb{Q}, x)[a] \mid a \in \mathbb{I}_{\mathbf{T}} \right\}$$

Naively, to find the risk-minimizing answer to a query, one has to compute the risk associated with each possible database object in the target database domain. Unfortunately, the set of objects $\mathbb{I}_{\mathbf{T}}$ is often infinite, but we will see that computing the risk-minimizing answer for reasonable instances of regret functions is possible.

First we prove that for most database target domain (enumerable ones), the notion of risk minimizing answer can capture the notion of information-based certain answers and knowledge preserving certain answers.

Theorem 9. *Let a query q from $\mathbf{S}$ to $\mathbf{T}$ such that the set of databases $\mathbb{I}_{\mathbf{T}}$ is enumerable. Let $\mathcal{A}$ be an additional knowledge to $\mathbf{S}$ and $\mathbf{T}$. Consider the trivial evaluation semantics of q denoted $\mathbb{Q}_t$ such that for every database $x \in \mathbb{I}_{\mathbf{S}}$ and every piece of knowledge $\omega \in \mathcal{A}$ we have $\mathbb{Q}_t(x, \omega) = q([\![x]\!]^{\omega}_{\mathbf{S}})$.*
Then there exists a regret function such that for every database $x \in \mathbb{I}_{\mathbf{T}}$, if the $\mathcal{A}$-preserving certain answer of q on x exists, then the $\mathcal{A}$-risk minimizing answer of $\mathbb{Q}_t$ on x exists and they coincide: $\mathsf{cert}_{\mathcal{A}}(q, x) = \mathsf{best}_{\mathcal{A}}(\mathbb{Q}_t, x)$

Proof of Theorem 9. For clarity, we prove a weaker version of the theorem. We ignore the additional knowledge $\mathcal{A}$ and prove that the risk-minimizing answers and information-based certain answers coincide. The proof with additional knowledge is similar in construction and just adds extra notation to an already tedious proof.

As we will use the cardinality of sets of databases, it is important to only count semantically equivalent databases once. For every database $x \in \mathbb{I}_{\mathbf{T}}$ the set of x equivalent databases denoted $\bar{x}$ is defined as

$$\bar{x} = \{y \in \mathbb{I}_{\mathbf{T}} \mid [\![x]\!]_{\mathbf{T}} = [\![y]\!]_{\mathbf{T}}\}$$

It is clear that for every $y \in \bar{x}$, we have $\bar{y} = \bar{x}$.

As $\mathbb{I}_{\mathbf{T}}$ is enumerable we pick an arbitrary mapping $f \colon \mathbb{I}_{\mathbf{T}} \mapsto \mathbb{N}$. Then for every $i > 0$, we define the finite set of database objects:

$$\mathbb{I}^{i}_{\mathbf{T}} = \{\bar{x} \mid \exists y, f^{-1}(y) < i \wedge y \in \bar{x}\}$$

Note that the cardinality of $\mathbb{I}^i_{\mathbf{T}}$ is bounded by i but not necessarily equal to i. Indeed two databases $x, y \in \mathbb{I}_{\mathbf{T}}$, such that $f^{-1}(x) < i$ and $f^{-1}(y) < i$ may be semantically equivalent and belong to the same class.

For every $z \in \mathbb{I}_{\mathbf{T}}$ we build the directed graph $G^i[z] = (\mathbb{I}^i_{\mathbf{T}} \cup \bar{z}, E[\bar{z}])$ such that there exists a edge between $\bar{x}$ and $\bar{y}$ if and only if we have $[\![\bar{y}]\!]_{\mathbf{T}} \subset [\![\bar{x}]\!]_{\mathbf{T}}$.

$$E[\bar{z}] = \{(\bar{x}, \bar{y}) \in (\mathbb{I}^i_{\mathbf{T}} \cup \bar{z}) \times (\mathbb{I}^i_{\mathbf{T}} \cup \bar{z}) \mid [\![\bar{y}]\!]_{\mathbf{T}} \subset [\![\bar{x}]\!]_{\mathbf{T}}\}$$

Note that for every $z \in \mathbb{I}_{\mathbf{T}}$, the directed graph $G^i[z]$ is acyclic, otherwise we would be able to find $\bar{x} \in \mathbb{I}^i_{\mathbf{T}}$ such that $[\![\bar{x}]\!]_{\mathbf{T}} \subset [\![\bar{x}]\!]_{\mathbf{T}}$ which is absurd.

Corollary 2. *For every $i > 0$ the function $\mu^i \colon \mathbb{I}^i_{\mathbf{T}} \times \mathbb{I}_{\mathbf{T}} \mapsto [0, 1]$ such that, for every $\bar{x} \in \mathbb{I}^i_{\mathbf{T}}$ and every $y \in \mathbb{I}_{\mathbf{T}}$,*

$$\mu^i(\bar{x}, y) = \begin{cases} 1 - \frac{1}{|\textit{longest-path}(G^i[y], \bar{x}, \bar{y})|+1} & \textit{if } x \preceq y \\ 1 \textit{ otherwise.} \end{cases}$$

where longest-path$(G^i[y], \bar{x}, \bar{y})$ is the function returning the longest path between $\bar{x}$ and $\bar{y}$ in $G^i[y]$ when it exists, is well defined.
Moreover for every $\bar{x} \in \mathbb{I}^i_{\mathbf{T}}$ and every $y \in \mathbb{I}_{\mathbf{T}}$ the following holds:

(a) $\mu^i(\bar{x}, y) \leq \mu^{i+1}(\bar{x}, y) \leq 1$
(b) $\mu^i(\bar{x}, y) = 1$ *iff.* $x \not\preceq y$

Proof of Corrollary 2. Let $i \geq 0$ and let $\bar{x} \in \mathbb{I}^i_{\mathbf{T}}$ and $y \in \mathbb{I}_{\mathbf{T}}$.

If $x \not\preceq y$, then $\mu^i(\bar{x}, y)$ is well defined, and for every $j \geq i$ we have $\mu^i(\bar{x}, y) = 1$.
If $\bar{x} = \bar{y}$, by convention we have |longest-path$(G^i[y], \bar{x}, \bar{y})| = 0$, then $\mu^i(\bar{x}, y)$ is well defined, and for every $j \geq i$ we have $\mu^i(\bar{x}, y) = 0$.
If $\bar{x} \neq \bar{y}$ and $\bar{x} \preceq \bar{y}$, then there exists an edge in $G^i[y]$ between $\bar{x}$ and $\bar{y}$. Moreover as $G^i[y]$ is a directed acyclic graph and its size is finite and bounded by i, the longest path between $\bar{x}$ and $\bar{y}$ exists and is bounded by i. Hence $\mu^i(\bar{x}, y)$ is well defined and $\mu^i(\bar{x}, y) < 1$. Moreover for every $j \geq i$ we have $G^i[z] \subseteq G^j[z]$. Therefore we also have |longest-path$(G^i[y], \bar{x}, \bar{y})| \leq$ |longest-path$(G^j[y], \bar{x}, \bar{y})|$. And we can conclude $\mu^i(\bar{x}, y) \leq \mu^j(\bar{x}, y) < 1$. ■

Corollary 3. *For every database $x, y \in \mathbb{I}_{\mathbf{T}}$, the limit $\lim_{i \mapsto \infty} \mu^i(\bar{x}, y)$ exists.*

Proof of Corrollary 3. If $x \not\preceq y$ there exists $i > 0$ such that $\mu^i(\bar{x}, y) = 1$. Then for every $j \geq i$ we have $\mu^j(\bar{x}, y) = 1$. Therefore if $x \not\preceq y$ we have $\lim_{i \mapsto \infty} \mu^i(\bar{x}, y) = 1$
If $x \preceq y$ there exists exists $i > 0$, such that for every $j \geq i$ we have $\mu^j(\bar{x}, y) < 1$. Moreover as we have $\mu^j(\bar{x}, y) \leq \mu^{j+1}(\bar{x}, y)$ (by corrolary 2), by the monotone convergence theorem we can conclude that $\lim_{i \mapsto \infty} \mu^i(\bar{x}, y)$ exists. ■

For every $x, y \in \mathbb{I}_{\mathbf{T}}$, we denote $\mu(x, y) = \lim_{i \mapsto \infty} \mu^i(\bar{x}, y)$

Let $x \in \mathbb{I}_{\mathbf{S}}$, such that $\mathsf{cert}_{\square}(q, x)$ exists. By contradiction assume there exists $\alpha > 0$ and $b \in \mathbb{I}_{\mathbf{T}}$ such that for every database $c \in \mathbb{Q}_{\mathsf{t}}(x, \epsilon) = q([\![x]\!]_{\mathbf{S}})$ we have

$$\mu(b, c) \leq \mathsf{risk}(\mathbb{Q}_{\mathsf{t}}, x)[\mathsf{cert}_{\square}(q, x)] - \alpha < 1$$

Especially $\mu(b, c) \leq 1 - \alpha$, therefore for every $c \in q([\![x]\!]_{\mathbf{S}})$ we have $b \preceq c$. And by definition of the certain answers we must have $b \preceq \mathsf{cert}_{\square}(q, x)$.

By definition of the risk of an answer, there exists $c \in q([\![x]\!]_{\mathbf{S}})$ such that

$$\mathsf{risk}(\mathbb{Q}_{\mathsf{t}}, x)[\mathsf{cert}_{\square}(q, x)] - \frac{\alpha}{2} \leq \mu(\mathsf{cert}_{\square}(q, x), c) < 1$$

Therefore by assumption on b there exists $c \in q([\![x]\!]_{\mathbf{S}})$

$$\mu(b, c) < \mu(\mathsf{cert}_{\square}(q, x), c) < 1$$

Especially, there exists $i > 0$, such that $\mu^i(b, c) < \mu^i(\mathsf{cert}_{\square}(q, x), c)$.
And by definition of μ^i we have that $L^i(\bar{b}, c) < L^i(\mathsf{cert}_{\square}(q, x), c)$. Therefore the longest path between $\bar{b}$ and $\bar{c}$ in $G^i[c]$ is strictly shorter than the path between $\mathsf{cert}_{\square}(q, x)$ and $\bar{c}$ in $G^i[c]$. Which means there can not exists a path between $\bar{b}$ and $\mathsf{cert}_{\square}(q, x)$, hence we have $\bar{b} \not\preceq \mathsf{cert}_{\square}(q, x)$, which is absurd.

Therefore b can not exist, which implies that for all $\alpha > 0$ and for all $b \in \mathbb{I}_{\mathbf{T}}$, there exists a database $c \in \mathbb{Q}_{\mathsf{t}}(x, \epsilon) = q([\![x]\!]_{\mathbf{S}})$ such that

$$\mu(b, c) \geq -\mathsf{risk}(\mathbb{Q}_{\mathsf{t}}, x)[\mathsf{cert}_{\square}(q, x)] - \alpha$$

And we can finally conclude that $\mathsf{cert}_{\square}(q, x) = \mathsf{best}(\mathbb{Q}_{\mathsf{t}}, x)$ QED

As the set of relational databases with null is enumerable, Theorem 9 tells us that we can build a regret function for relational algebra with null for the OWA and the WWA, such that the valuation-preserving certain answers and the risk-minimizing answers coincide. However, we wish to be able to capture answers in between the certain answers obtained with OWA and WWA. In the next sections, we propose a class of regret functions based on the *optimal transportation problem*, which offers more flexibility to the users.

5.4 A Class of Regret functions for relational databases

Example 12. Consider the query q and database d returning the salary of all employees such that the first name is equal to the second name:

$$q = \pi_{\text{Salary}}\big(\sigma_{\text{First Name}=\text{Second Name}}\big)(\text{Employees})$$

on the database:

Employees

First Name	Second Name	Salary
John	$\perp_1$	2000
$\perp_1$	Peter	1000

The **NI**-evaluation semantics of q gives us the following possible answers:

Answer1

a

Answer2

a
1000

Answer3

a
2000

Answer4

a
1000
2000

And it yields the following answers:

$\text{cert}_{\{\epsilon\}}(q_{\mathbf{NI}}, d)$ with OWA

a

$\text{cert}_{\{\epsilon\}}(q_{\mathbf{NI}}, d)$ with WWA

a
2000
1000

A potential problem with those certain answers is that they are not the risk-minimizing ones for some reasonable notion of regret. If we consider the bag $\{\!|1000, 2000|\!\}$ as the answer, we may face a situation where we consider two false tuples as true. If we consider the empty bag as the answer, we may face a situation where we are missing

two true tuples. A user may want to find a compromise between missing some true tuples and considering some false ones. For instance, an answer such as $\{\!|\perp|\!\}$ could be more satisfying for some users. Indeed in the worst-case scenario, we are either missing one tuple and one value or assuming one false tuple. Depending on the regret a user assigns to missing a true tuple, assuming a false tuple and missing a value, $\{\!|\perp|\!\}$ could be a better answer than both $\{\!|1000, 2000|\!\}$ and the empty bag.

Example 12 sums up the intuition behind the class of regret functions we introduce in this section. We assume that a user provides a value $\text{Cost}_{\text{Prod}}$ accounting for the regret he faces when missing some true tuples. The user also provides a value $\text{Cost}_{\text{Sink}}$ accounting for the regret he faces when considering some false tuples. Finally, he provides a value $\text{Cost}_{\text{unify}}$ accounting for the regret he faces when using a null placeholder instead of a constant.

To propose a fair notion of regret for relational databases, we consider each tuple in an answer as an information container to which we assign a mass. The cost of transporting a tuple's mass to another is proportional to their information dissimilarity (using the value $\text{Cost}_{\text{unify}}$). Then the regret value between two bags is the minimal cost of transporting the mass of the tuples in one bag to the tuples in the other bag without overfilling them. To compensate for non-unifying tuples or bags of different sizes, we use the value $\text{Cost}_{\text{Prod}}$ and $\text{Cost}_{\text{Sink}}$ to create special tuples.

Depending on the values of $\text{Cost}_{\text{Prod}}$, $\text{Cost}_{\text{Sink}}$, and $\text{Cost}_{\text{unify}}$ it results in different risk-minimizing answers. Indeed if the regret of missing some tuples $\text{Cost}_{\text{Prod}}$ is low compared to $\text{Cost}_{\text{Sink}}$ then the risk-minimizing answers tend to contain fewer tuples. Moreover, if the regret of using a null instead of a constant $\text{Cost}_{\text{unify}}$ dominates the other values, then the risk-minimizing answers will not contain any null placeholders.

5.4.1 Kantorovich transportation problem

The Kantorovich problem is a formulation for optimal transportation Villani (2009). The notion produced satisfactory results in other domains of computer science Chapel, Alaya, and Gasso (2020); Flamary, Courty, Rakotomamonjy, and Tuia (2014); Vincent-Cuaz, Vayer, Flamary, Corneli, and Courty (2021).

Considering a set of locations $\mathcal{X}$, we define the set of mass assignment over $\mathcal{X}$ denoted $\mathcal{M}$ as the set of functions $\mathbf{a}\colon \mathcal{X} \to \mathbb{R}^+$. Let a cost function $C\colon \mathcal{X} \times \mathcal{X} \in \mathbb{R}^+$ such that $C(x, y)$ is the price for moving a unit of mass from the location x to the location y. Considering two mass assignment $\mathbf{a}, \mathbf{b} \in \mathcal{M}$ we denote m_a resp m_b the total mass of a resp. b such that $m_a = \sum_{x\in\mathcal{X}} \mathbf{a}(x)$ resp. $m_b = \sum_{x\in\mathcal{X}} \mathbf{b}(x)$. We consider the balanced version of the Kantorovich transportation problem, and we assume that $m = m_a = m_b$. We denote the set of admissible m-mass transport $U(\mathbf{a}, \mathbf{b})$ as the set

of assignments $u\colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}^+$ such that:

$$\forall x_{\mathbf{a}} \in \mathcal{X}, \sum_{x_{\mathbf{b}} \in \mathcal{X}} u(x_{\mathbf{a}}, x_{\mathbf{b}}) \leq \mathbf{a}(x_{\mathbf{a}})$$
$$\forall x_{\mathbf{b}} \in \mathcal{X}, \sum_{x_{\mathbf{a}} \in \mathcal{X}} u(x_{\mathbf{a}}, x_{\mathbf{b}}) \leq \mathbf{b}(x_{\mathbf{b}})$$
$$\sum_{x_{\mathbf{a}}, x_{\mathbf{b}} \in \mathcal{X}} u(x_{\mathbf{a}}, x_{\mathbf{b}}) = m$$

The Kantorovich optimal transportation solution is defined as the minimal cost of an admissible m-mass transport:

$$K_C(\mathbf{a}, \mathbf{b}) = \min\{ \sum_{x_{\mathbf{a}}, x_{\mathbf{b}} \in \mathcal{X}} u(x_{\mathbf{a}}, x_{\mathbf{b}}) \cdot C(x_{\mathbf{a}}, x_{\mathbf{b}}) \mid u \in U(\mathbf{a}, \mathbf{b}\}$$

For relational databases, each location will correspond to a tuple, and its mass will match its multiplicity in the bag. Before detailing the specificity of the mass assignment, we introduce a cost function for transporting mass between tuples depending on $\mathsf{Cost}_{\mathsf{unify}}$.

5.4.2 Information dissimilarity between tuples

First, we introduce a notion of dissimilarity between element of an attribute α as a function $\mathbf{C}_\alpha\colon (\mathsf{Const} \cup \{\bot\}) \times (\mathsf{Const} \cup \{\bot\}) \mapsto [0, \infty]$ such that for all $e, e' \in \mathsf{Const} \cup \{\bot\}$:

$$\mathbf{C}_\alpha(e, e') = \begin{cases} 0 & \text{if } e = e' \\ \mathsf{Cost}_{\mathsf{unify}} & \text{if } (e = \bot) \oplus (e' = \bot) \\ \infty & \text{otherwise.} \end{cases}$$

The information dissimilarity is minimal when the two values are the same. It is user-dependent when one value is null, and the other is a constant. Finally, the information dissimilarity is maximal between two different constants because they have incomparable information content.

For tuples $\bar{t}, \bar{t}'$ with attribute $\bar{\alpha}$ we define the information dissimilarity cost as the average cost between each of their attributes.

$$\mathbf{C}_{\bar{\alpha}}(\bar{t}, \bar{t}') = \frac{\sum_{\alpha \in \bar{\alpha}} \mathbf{C}_\alpha(\bar{t}[\alpha], \bar{t}'[\alpha])}{|\bar{\alpha}|}$$

5.4.3 Tuple mass assignment

Let $\mathbf{a}, \mathbf{b}$ two bags of tuple with the same attributes $\bar{\alpha}$. A naive mass assignment would be to attribute to every tuple in $\mathbf{a}$ and $\mathbf{b}$ a mass equal to its multiplicity. However, a problem arises when the size of $\mathbf{a}$ and $\mathbf{b}$ are different. Then the Kantorovich problem is not balanced. We could consider the generalized formulation of the Kantorovich problem to solve the issue. But the difference of size between $\mathbf{a}$ and $\mathbf{b}$ is a piece of valuable information we exploit to propose a relevant notion of regret.

$\mathbf{a}$

From	To	Total
Jane	Alice	1000
$\perp$	Jane	2000

$\mathbf{b}$

From	To	Total
Jane	John	$\perp$
Alice	Jane	$\perp$
$\perp$	Jane	1000

To illustrate the mass assignment situation, consider the answers $\mathbf{a}$ and $\mathbf{b}$ above. First notice that the tuple (Jane,Alice,1000) $\in \mathbf{a}$ do not unify with any tuple from $\mathbf{b}$. Hence the cost of transporting its mass to any tuple in $\mathbf{b}$ is infinite. Therefore, with the trivial mass assignment, any valid transportation plan from $\mathbf{a}$ to $\mathbf{b}$ would be infinite. We argue it is not fair, mainly because we asked our user to provide a value accounting for the regret he faces when considering some false tuples $\text{Cost}_{\text{Sink}}$.

Indeed instead of being infinite, the regret should depend on $\text{Cost}_{\text{Sink}}$. More formally, a transportation plan from $\mathbf{a}$ to $\mathbf{b}$ should be allowed to discard the tuple (Jane,Alice,1000) with a cost $\text{Cost}_{\text{Sink}}$ to account for the fact that the tuple may be false. To make such a transportation plan valid, we add a special tuple to $\mathbf{b}$ called *sinker*. Informally a sinker is an information container that has a fixed transportation cost $\text{Cost}_{\text{Sink}}$ with every tuple in $\mathbf{a}$.

Similarly, notice that the tuple (Jane,John,$\perp$) $\in \mathbf{b}$ do not unify with any tuple from $\mathbf{a}$. The user has provided a value accounting for the regret he faces when missing some true tuple $\text{Cost}_{\text{Prod}}$. A transportation plan from $\mathbf{a}$ to $\mathbf{b}$ should be allowed to create the tuple (Jane,John,$\perp$) with a $\text{Cost}_{\text{Prod}}$ to account for the fact that we are missing a tuple. To make such a transportation plan valid, we add a special tuple to $\mathbf{a}$ called *producer*. Informally a producer is an information container that has a fixed transportation cost $\text{Cost}_{\text{Prod}}$ with every tuple in $\mathbf{a}$.

Definition 5.4.1 (Mass assignment)**.** *For every bag relation* $\mathbf{a}$*,* $\mathbf{b}$ *we define the* mass assignment *as the function* $M[\mathbf{a} \mapsto \mathbf{b}]$ *such that for every tuple with fitting attribute:*

$$M[\mathbf{a} \mapsto \mathbf{b}](\mathbf{a}, \bar{t}) = \begin{cases} |\mathbf{b}| & \textit{if } \bar{t} \textit{ is } \text{producer} \\ \#\big(\mathbf{a}, \bar{t}\big) & \textit{otherwise.} \end{cases}$$

$$M[\mathbf{a} \mapsto \mathbf{b}](\mathbf{b}, \bar{t}) = \begin{cases} |\mathbf{a}| & \textit{if } \bar{t} \textit{ is } \text{sinker} \\ \#\big(\mathbf{b}, \bar{t}\big) & \textit{otherwise.} \end{cases}$$

And we extend the cost function between tuple in the following way:

$$\mathbf{C}_{\bar{\alpha}}(\bar{t}, \bar{t}') = \begin{cases} 0 & \text{if } \bar{t} \text{ is } \textit{producer} \wedge \bar{t}' \text{ is } \textit{sinker} \\ \text{Cost}_{\text{Prod}} & \text{if } \bar{t} \text{ is } \textit{producer} \\ \text{Cost}_{\text{Sink}} & \text{if } \bar{t}' \text{ is } \textit{sinker} \\ \mathbf{C}_{\bar{\alpha}}(\bar{t}, \bar{t}') & \text{otherwise.} \end{cases}$$

The mass of the *producer* is equal to the size of $\mathbf{b}$, so each missing tuple can be produced with a regret $\text{Cost}_{\text{Prod}}$. Similarly, the mass of the *sinker* is equal to the size of $\mathbf{a}$, so each completely false tuple can be discarded with a regret $\text{Cost}_{\text{Sink}}$. Finally, the cost of transporting the mass from the producer to the sinker is 0.

An example of a valid transportation plan from $\mathbf{a}$ to $\mathbf{b}$ would be the following:

$$\begin{aligned} u(\text{(Jane,Alice,1000)} \mapsto \textit{sinker}) &= 1 && \text{(Jane,Alice,1000) is false information} \\ u((\bot\text{,Jane,2000)} \mapsto \text{(Alice,Jane,}\bot)) &= 1 && (\bot\text{,Jane,2000) has a partial match} \\ u(\textit{producer} \mapsto \text{(Jane,John,}\bot)) &= 1 && \text{(Jane,John,}\bot\text{) is a new information} \\ u(\textit{producer} \mapsto (\bot\text{,Jane,1000})) &= 1 && (\bot\text{,Jane,1000)) is a new information} \\ u(\textit{producer} \mapsto \textit{sinker}) &= 1 && \text{sinking the residual producer mass} \end{aligned}$$

Which yield to a a total cost of

$$\text{Cost}_{\text{Sink}} + \frac{2}{3} \cdot \text{Cost}_{\text{unify}} + 2 \cdot \text{Cost}_{\text{Prod}}$$

Remark that with the mass assignment mentioned above, the multiplicity of tuples in each bag matters as they define their respective mass.

5.4.4 Risk minimizing answers with OT regret function

Definition 5.4.2 (OT regret function)**.** *For every Cost$_{unify}$, Cost$_{Prod}$, Cost$_{Sink}$ $\in \mathbb{N}$ the* OT*-regret function $\mu\colon \mathbb{I}_{\mathbf{T}} \times \mathbb{I}_{\mathbf{T}} \mapsto \mathbb{R}^+$ is defined for every bag relational databases $\mathbf{a}, \mathbf{b} \in \mathbb{I}_{\mathbf{T}}$ with attributes $\bar{\alpha}$ as the optimal solution of the transportation problem with the mass assignement $M[\mathbf{a} \mapsto \mathbf{b}]$ and the cost function $\mathbf{C}_{\bar{\alpha}}$:*

$$\mu(\mathbf{a}, \mathbf{b}) = K_{\mathbf{C}_{\bar{\alpha}}}\big(M[\mathbf{a} \mapsto \mathbf{b}](\mathbf{a}), M[\mathbf{a} \mapsto \mathbf{b}](\mathbf{b})\big)$$

The class of OT regret function allows us to capture the notion of certain answers with either OWA and WWA easily.

Proposition 16. *For every relation algebra with null query q and every database d, there exist values $M_1 < M_2$ and N_1 such that for every OT regret function with:*

(a) $0 <$ *Cost$_{Prod}$* $< M_1$ *and Cost$_{Sink}$* $> M_2$
(b) $0 <$ *Cost$_{unify}$* $< N_1$

the regret-minimizing answer coincides with the ***NI****-evaluation-based certain answers with* OWA*.*

Recall that the **NI**-evaluation-based certain answers with OWA contain as many true tuples as possible without containing any false tuple. Intuitively, the condition $0 <$ Cost$_{\text{unify}}$ $< N_1$ ensures that a null placeholder's information content is lower than any constant's information content. The condition $0 <$ Cost$_{\text{Prod}}$ $< M_1$ ensures that the risk-minimizing answers contain as many tuples as possible by punishing missing tuples with a small but not 0 regret Cost$_{\text{Prod}}$. The condition Cost$_{\text{Sink}}$ $> M_2$ ensures that we heavily punish every answer containing a false tuple with a cost Cost$_{\text{Sink}}$, ensuring that the risk-minimizing answers only contain tuples that are always true. Under those conditions, the risk-minimizing answers will match the **NI**-evaluation-based certain answers with OWA.

Proposition 17. *For every relation algebra with null query q and every database d, there exist values $M_1 < M_2$ and $N_1 < N_2$ such that for every OT regret function with:*

(a) $0 <$ *Cost$_{Sink}$* $< M_1$ *and Cost$_{Prod}$* $> M_2$
(b) $0 <$ *Cost$_{unify}$* $< N_1$

the regret-minimizing answer coincides with the ***NI****-evaluation-based certain answers with* WWA*.*

Recall that the **NI**-evaluation-based certain answers with WWA contain all true tuples and as few false tuples as possible. Intuitively, the condition $0 <$ Cost$_{\text{unify}}$ $< N_1$ ensures that a null placeholder's information content is lower than any constant's information content. The condition $0 <$ Cost$_{\text{Sink}}$ $< M_1$ ensures that the risk-minimizing answers contain as few tuples as possible by punishing false tuples with a small but not 0

regret $\text{Cost}_{\text{Sink}}$. The condition $\text{Cost}_{\text{Prod}} > M_2$ ensures that we heavily punish every answer not containing a possibly true tuple with a cost $\text{Cost}_{\text{Prod}}$, ensuring that the risk-minimizing answers contain all true tuples. Under those conditions, the risk-minimizing answers will match the **Nl**-evaluation-based certain answers with WWA.

Proof sketch for Proposition 16 and 17. The proof of both Proposition 16 and Proposition 17 are similar. First we have to prove that there exists $M_1 < M_2$ and N_1 such that for every OT regret function with:

(a) $0 < \text{Cost}_{\text{Sink}} < M_1$ and $\text{Cost}_{\text{Prod}} > M_2$
(b) $\text{Cost}_{\text{unify}} > N_1$

we have:

$$\mu(\mathbf{a}, \mathbf{b}) > M_2 \text{ iff. } a \not\preceq^{\text{WWA}} b$$

and

$$\mu(\mathbf{a}, \mathbf{b}) \leq \mu(\mathbf{a}, \mathbf{c}) \leq M_2 \text{ iff. } a \preceq^{\text{WWA}} c \preceq^{\text{WWA}} b$$

In order to do so, we just have to pick M_2 arbitrarily big. Then we choose M_1 and N_1 such that M_2 dominates them. Then, we will only pay the cost M_2 if no other tuple unify because $N1 << M2$, and we have to produce a tuple. In other word, if we are completely missing some potentially true tuple ie. $a \not\preceq^{\text{WWA}} b$.

Moreover the cost of optimal transport will increase with the number of null in the answers because $\text{Cost}_{\text{unify}} > 0$, and with the size of the answers because $\text{Cost}_{\text{Sink}} > 0$ hence

$$\mu(\mathbf{a}, \mathbf{b}) \leq \mu(\mathbf{a}, \mathbf{c}) \leq M_2 \text{ iff. } a \preceq^{\text{WWA}} c \preceq^{\text{WWA}} b$$

Finally because the set of relational databases is enumerable the proof is a special case of the proof for Theorem 9. □

Thanks to Proposition 16 and Theorem 17, we understand that certain answers are special cases of risk-minimizing answers with extreme OT regret functions. Using Example 12 we look at how the risk-minimizing answers evolve depending on the user-defined values: $\text{Cost}_{\text{Prod}}$, $\text{Cost}_{\text{Sink}}$, and $\text{Cost}_{\text{unify}}$.

Example 13. Consider the query q and database d returning the salary of all employees such that the first name is equal to the second name:

$$q = \pi_{\text{Salary}}\big(\sigma_{\text{First Name}=\text{Second Name}}\big)(\text{Employees})$$

on the database:

Employees

First Name	Second Name	Salary
John	$\perp_1$	2000
$\perp_1$	Peter	1000

The **NI**-evaluation semantics of q gives us the following possible answers:

Answer1

a

Answer2

a
1000

Answer3

a
2000

Answer4

a
1000
2000

First, notice that independently of the query, if $\text{Cost}_{\text{Prod}}$ is equal to 0, then the risk-minimizing answer would always be the empty bag, as its regret with any other bag is 0.

If the regret of using a null placeholder $\text{Cost}_{\text{unify}}$ is lower than $\text{Cost}_{\text{Prod}}$ and $\text{Cost}_{\text{Sink}}$ then the regret between $\{\!\{(1000)\}\!\}$ and $\{\!\{(\perp)\}\!\}$ is given by $\text{Cost}_{\text{unify}}$.

Therefore for a user who considers:

(a) Answers with false tuples are as bad as answers missing true tuples ($\text{Cost}_{\text{Prod}} = \text{Cost}_{\text{Sink}}$).

(b) The information content of a null placeholder is less than a constant, but better than no information ($0 < \text{Cost}_{\text{unify}} < \text{Cost}_{\text{Prod}} = \text{Cost}_{\text{Sink}}$)

Then the risk-minimizing answer for $q_{\mathbf{NI}}$ on the database Employees is:

$$\text{best}_{\mathcal{A}}(q_{\mathbf{NI}}, \text{Employees}) = \{\!\{(\perp)\}\!\}$$

Indeed we have

$$\text{risk}(q_{\mathbf{NI}}, d)[\{\!\{(\perp)\}\!\}] = \sup\left\{\mu(\{\!\{(\perp)\}\!\}, y) \mid y \in q_{\mathbf{NI}}(d)\right\}$$

and we have

$$\begin{aligned}
\mu(\{\!\{(\perp)\}\!\}, \{\!\{\}\!\}) &= \text{Cost}_{\text{Sink}} \\
\mu(\{\!\{(\perp)\}\!\}, \{\!\{(1000)\}\!\}) &= \text{Cost}_{\text{unify}} \\
\mu(\{\!\{(\perp)\}\!\}, \{\!\{(2000)\}\!\}) &= \text{Cost}_{\text{unify}} \\
\mu(\{\!\{(\perp)\}\!\}, \{\!\{(2000), (1000)\}\!\}) &= \text{Cost}_{\text{unify}} + \text{Cost}_{\text{Prod}}
\end{aligned}$$

hence $\text{risk}(q_{\mathbf{NI}}, d)[\{\!\{(\perp)\}\!\}] = \text{Cost}_{\text{unify}} + \text{Cost}_{\text{Prod}}$.

While the risk of other potential answers is higher than $2 \times \text{Cost}_{\text{Prod}}$ resp. $2 \times \text{Cost}_{\text{Sink}}$.

We have shown that the class of OT regret function for relational databases can capture **NI**-evaluation based certain answers and also answers that were not accessible before. Moreover, the user chooses those answers thanks to the values $\text{Cost}_{\text{Prod}}$, $\text{Cost}_{\text{Sink}}$, $\text{Cost}_{\text{unify}}$.

5.5 Conclusion

Thanks to the survey results, we understood the importance of dealing with more interpretations of incompleteness. While the previous notions were tight to the missing data interpretation, our framework based on the notion of query evaluation semantics can capture other models of incompleteness. First, we have shown how to instantiate it on the relational algebra model and provided a model for the no-information interpretation of null values. Then we showed how to efficiently compute the evaluation based certain answers with either the open-world or the whole-world assumption. The latter assumption is filtering out fewer tuples in the answers.

Then, we wanted to offer more possibilities to users. However, the certain answers notion cannot provide flexible answers because of the notion of dissimilarity between databases it considers. Therefore we introduce a new notion of answers on incomplete databases based on a user-defined regret function, the risk-minimizing answers. We have shown that there exists a regret function such that the risk-minimizing answers coincide with the information-based certain answers. Moreover, because the regret function is a numerical measure of dissimilarity, the notion offers more granularity in the answers it can propose to a user.

As natural applications, we presented a class of regret functions based on the optimal-transport problem for relational databases. We showed that depending on user preferences, the risk-minimizing answers can vary between the evaluation-based certain answers with the open-world and the whole-world assumption.

The next obvious step is to use the framework to define risk-minimizing answers for queries with aggregates and groupings. However, the question of what are pertinent answers for value inventing queries is still open. The survey did not provide meaningful input. The results show that, even if not entirely satisfactory, the most popular answers are SQL's. Therefore instead of redefining answers, we use the notion of risk to improve and explain answers returned by database management systems.

Chapter 6

Improving and Explaining Answers

"Authority is no source for Truth." Aristotle. While we recognize the irony of the citation, it provides a good illustration of the query-answering on incomplete databases problem. In the context of relational databases, the commercial implementations of SQL are the Authority. Willingly or not, users have to accept the answers they output. However, despite recent efforts to formalize their language and data model, users have little reason to trust query evaluation systems. Worst on incomplete databases, there is no 'Truth'. When we consider logically-consistent notions of answers on incomplete databases, such as certain answers and risk-minimizing answers, we provide a explanation for answers instead of relying on 'Authority. However, there are two key shortcomings to this approach.

First, a logically-consistent notion of answer provides a *global* explanation. Informally the explanation for an answer is independent of the query and source database. Indeed the explanation of any answer boils down to the interpretation of incompleteness and the measure of dissimilarity between database objects. Then the answers are defined as the most pertinent database object under those assumptions. While a step in the right direction, we want to propose *local* explanation of answers. We wish to provide explanations that depend on the query and source databases.

Second, database management systems are the 'Authority'. Defining and explaining logically-consistent notions of answers is a step forward, but it does not change the answers SQL gives users. We must acknowledge that SQL will not change its evaluation behavior lightly. Too many applications rely on it. If we wish to have a practical impact and improve user satisfaction, we must handle the answers provided by database management systems. Considering DBMS's apparent lack of logical consistency, a satisfying *global* explanation for their answers to queries on incomplete databases seems hard to obtain. However, we can provide *local* explanations.

In this chapter we leverage the advantage of considering a numerical dissimilarity measure between databases, like the *regret*. It allows us to capture the degree of incompleteness of any answer. Indeed the notion of *risk* is well defined for any database object we wish to consider as an answer to a query on an incomplete database. Signi-

ficantly, we can compute the risk of the answers provided by a database management system. While already interesting, the next step would be to improve such answers. In order to do so we propose to explain the degree of the risk of answers by exhibiting and ranking its sources of incompleteness. We represent the sources of incompleteness as the features of a *latent representation* of the additional knowledge.

Then we assign a value of importance to each feature. In isolation, we define the *cleaning-importance* of a source of incompleteness as the diminution of the risk value of a query-evaluation system's answer if we clean the incompleteness from the aforementioned source. In other words, How much would the risk of the SQL answers diminish if we discover the meaning of a specific `NULL`.

The *cleaning-importance* provides valuable information to users who want to clean their data to efficiently reduce the risk of answers. However, it does not offer a satisfying explanation for the risk. Indeed it is not a measure of the contribution of each source. To overcome this difficulty, we introduce a measure of contribution based on the well-known Shapley value, the *risk-contribution* of a source incompleteness. We argue that the *risk-contribution* is a better value to provide an explanation for the risk of an answer.

Finally, the cleaning importance and risk-contribution values can only be inferred. Therefore, we propose simple models to infer its value. To study the framework's applicability and the precision of its inference models, we experimented with relational databases and queries with aggregate and grouping operations. And despite some remaining challenges, the results are promising.

6.1 Risk associated with Knowledge

Recall that with the notion of regret, we can assign a *risk score* to a database we wish to use as an answer to a query on a database. Especially the risk associated with an answer a is the maximal regret a user would have if the real answer to q on x is different.

Definition (Risk of an answers)**.** *Let a query q in from $\mathbf{S}$ to $\mathbf{T}$ with $\mathbb{Q}$ its evaluation semantics. Let $x \in \mathbb{I}_{\mathbf{S}}$, and let $\mathcal{A}$ be additional evaluation knowledge for $\mathbb{Q}$. The risk of a database $a \in \mathbb{I}_{\mathbf{T}}$ as an answer to $\mathbb{Q}$ on x with the evaluation knowledge $\mathcal{A}$ is the value:*

$$\mathsf{risk}_{\mathcal{A}}(\mathbb{Q}, x)[a] = \sup\left\{\mu(a, y, \omega) \mid \omega \in \mathcal{A}, y \in \mathbb{Q}(x, \omega)\right\}$$

While the risk of an answer provides valuable information to users, studying the risk of an answer does not allow us to extrapolate the properties of a DBMS evaluation.

On the other hand, the behavior of a DBMS can be captured by some extra knowledge $\bar{\omega}$. If we assume that the evaluation semantics $\mathbb{Q}$ on x with the knowledge $\bar{\omega}$ returns a single database a, then we can define the risk of a knowledge $\mathsf{risk}_{\mathcal{A}}(\mathbb{Q}, x)[\bar{\omega}]$ such that

$$\mathsf{risk}_{\mathcal{A}}(\mathbb{Q}, x)[\bar{\omega}] = \sup_{\bar{\omega}' \in \mathbf{A}} \big(\mu(\mathbb{Q}(x, \bar{\omega}), \mathbb{Q}(x, \bar{\omega}'), \bar{\omega}')\big) = \mathsf{risk}_{\mathcal{A}}(\mathbb{Q}, x)[a]$$

However in general the evaluation of $\mathbb{Q}$ on x with the knowledge $\bar{\omega}$ does not need to be a singleton.

Definition 6.1.1 (Risk of a knowledge)**.** *Let a query semantics $\mathbb{Q}$, a database $x \in \mathbb{I}_{\mathbf{S}}$ and an additional knowledge $\mathcal{A}$ for the query semantics $\mathbb{Q}$ on the database x. The risk of a a knowledge $\bar{\omega} \in \mathcal{A}$ to the evaluation of $\mathbb{Q}$ on x upon the additional knowledge $\mathcal{A}$ is the value:*

$$\mathsf{risk}_{\mathcal{A}}(\mathbb{Q}, x)[\bar{\omega}] = \sup_{\bar{\omega}' \in \mathcal{A}} \big(\sup_{y \in \mathbb{Q}(x, \bar{\omega}')} (\inf \{\mu(a_i, y, \bar{\omega}') \mid a_i \in \mathbb{Q}(x, \bar{\omega}\}) \big)$$

The risk of a knowledge vector $\bar{\omega}$ is the maximal regret a user endures if the knowledge $\bar{\omega}$ is wrong. It boils down to finding the knowledge vector $\bar{\omega}' \in \mathbf{A}$, which maximizes the regret between the answers obtained with the knowledge $\bar{\omega}$ and the answers obtained with the knowledge $\bar{\omega}'$. Furthermore, because the query evaluation semantics may return a set of possible answers even with the knowledge vector $\bar{\omega}'$, we look for the possible answer which maximizes the regret. Finally, as the evaluation semantics with the knowledge vector $\bar{\omega}$ may also give us several options to choose from, we select the most suitable answer. Hence we consider the infimum of the regret values.

Thanks to the risk of knowledge we can reason about the DBMS behavior. We wish to use this new tool to identify, explain and reduce the risk associated with a query evaluation.

6.1.1 Vectorial Knowledge Query Evaluation Semantics

Why is the risk value so high or so low? How can a user reduce it? To answer those questions, we partition the incompleteness into sources.

To better understand the situation, we present the following analogy. Assume we conducted a sanguine transfusion experiment on a population. First, we mapped each blood sample to the set of individuals who had a bad reaction when transfused with it. It represents a query from a blood sample to a set of individuals, we are selecting all the people who had a bad reaction. Now assume we are mixing some blood samples and wish to infer the set of individuals who may have a bad reaction to the transfusion. Then, the blood samples can be seen as the complete objects of a database domain, and mixing blood samples is an analogy for considering an incomplete object and

its semantics. The logically-consistent notions of answers on incomplete databases provide a *global* explanation for the population it would transfuse. With the certain answers with the open-world assumption, we would transfuse the set of individuals who had a bad reaction to every blood sample contained in the mix. And with the certain answers with the whole-world assumption, we would transfuse the set of individuals who had a bad reaction to at least one blood sample contained in the mix. Depending on the regret one associates with a bad reaction, the risk-minimizing answers may transfuse any set of individuals.

Now assume we receive a new blood sample for which we have no transfusion data. This sample could either be a mix of already studied but unidentified samples or a completely unstudied sample. How do we infer the risk associated with using this blood sample? Which individual may have a bad reaction to it? We may provide some answers depending on our information about the blood sample and our understanding of the blood transfusion mechanism. In this situation, we are providing an additional knowledge to the query evaluation semantics, which captures our understanding of the mechanism for unidentified blood samples. Thanks to this additional knowledge, we can partition the incompleteness into relevant features.

Assuming we still wish to use this unidentified blood sample, we will need to reduce its risk. What test should I conduct on the blood sample to obtain better answers? Assume that our understanding of the transfusion process allows us to know the relevant features of blood. For the sake of simplicity, consider that we know about blood-type (O, A, B, AB) and rhesus factor (+,-). We would like to know which test is more valuable to conduct, rhesus factor or blood-type identification. For example, suppose our population contains only individuals with blood-type (AB). In that case, it is more valuable to conduct a rhesus test (indeed, an individual with group AB can receive blood from all other blood-type). On the other hand, if our population contains only individuals with rhesus factor (+), identifying the group type becomes more important than identifying the rhesus. We understand that the test that will reduce the risk the most depends on the population and the task. Therefore, studying the importance of each test provides a *local* explanation for the risk.

In the context of databases, performing a test is an analogy for cleaning a source of incompleteness. Therefore, finding the most important sources of incompleteness accounts for guessing which would reduce the risk of the query evaluation the most. However, we first have to understand what are sources of incompleteness in an abstract context. For blood transfusion example, the blood type and the rhesus factor are the sources of incompleteness. The blood type and the rhesus factor notion are compatible sources of incompleteness because the pieces of knowledge they contain can not contradict each other. We call them *features*, and they constitute a *vectorial*

knowledge for the blood transfusion problem. Moreover, if the features are enough to completely infer the answers (if after cleaning all features, there is no incompleteness remaining), we say that the vectorial knowledge is a *a latent representation* for blood samples.

Definition 6.1.2 (Vectorial knowledge)**.** *A vectorial knowledge for a query evaluation semantics* $\mathbb{Q}$ *and a database* $x \in \mathbb{I}_{\mathbf{S}}$ *of dimension* $n \geq 0$ *is a* n*-tuple* $\mathbf{A} = (X_1, \cdots, X_n)$ *such that, for every* $i, j \leq n$ *the following hold:*

- X_i *is a subset of the universal knowledge,* $X_i \subseteq \mathcal{K}$ *and* $\epsilon \notin X_i$*.*
- *For every piece of knowledge* $\omega_i \in X_i$*, and* $\omega_j \in X_j$ *we have* $\mathbb{Q}(x, \omega_i\omega_j) = \mathbb{Q}(x, \omega_j\omega_i)$*.*

Moreover when $\mathbf{A} = (X_1, \cdots, X_n)$ *is vectorial knowledge, we call the subsets of knowledge* X_i features*. The elements* $\omega_i \in X_i$ *of the features are called* characteristics*, and the tuples* $(\omega_1, \cdots, \omega_n) \in X_1 \times \cdots \times X_n$ *are called* knowledge vectors*.*

The elements of a knowledge vector are commutable non-empty pieces of knowledge from $\mathcal{K}$. A vectorial knowledge enforces that the characteristics are commutable with respect to the query evaluation semantics. Therefore it ensures that the knowledge contained in each feature can not contradict each other. For instance, in the case of blood samples, we can not consistently specify blood samples with the three feature sets $\{AB, B\}$, $\{O, A\}$, and $\{+, -\}$. While every combination of blood type and rhesus factor is included, a knowledge vector $AB \cdot O \cdot +$ is valid but would be inconsistent. Therefore $\{AB, B\}$, $\{O, A\}$ and $\{+, -\}$ is not a vectorial knowledge.

In the context of incomplete query answering, a vectorial knowledge allows us to define incompleteness sources as features. However, nothing ensures that the pieces of knowledge contained in a vectorial knowledge are enough to capture the whole query evaluation semantics.

Definition 6.1.3 (Latent representation)**.** *Let a sub-monoid of knowledge* $\mathcal{A} \subseteq \mathcal{K}$ *additional to a source domain* $\mathbf{S}$ *and a target domain* $\mathbf{T}$*. A n-tuple* $\mathbf{A} = (X_1, \cdots, X_n)$ *is a latent representation of* $\mathcal{A}$ *if for every query semantics* $\mathbb{Q}$ *from* $\mathbf{S}$ *to* $\mathbf{T}$ *such that* $\mathcal{A}$ *is an evaluation-based additional knowledge to* $\mathbb{Q}$ *and every source database* $x \in \mathbb{I}_{\mathbf{S}}$ *the following hold:*

1. $\mathbf{A}$ *is a vectorial knowledge for* $\mathbb{Q}$ *and* x*.*
2. *For every piece of knowledge* $\omega \in \mathcal{A}$*, there exists a knowledge vector* $(\omega_1, \cdots, \omega_n) \in X_1 \times \cdots \times X_n$ *such that for every* $\omega' \in \mathcal{A}$ *we have* $\mathbb{Q}(x, \omega\omega') = \mathbb{Q}(x, \omega_1 \cdots \omega_n \cdot \omega')$*.*

A vectorial knowledge is a latent representation of $\mathcal{A}$ if, for every piece of knowledge in $\mathcal{A}$, there exists a knowledge vector equivalent to it for every query evaluation semantics on every source database. Intuitively, a latent representation enforces that our features fully capture the knowledge in $\mathcal{A}$. For instance, a vectorial knowledge $\{A, AB, B\}$

and $\{+, -\}$ would not be a latent space for blood sample as there is no knowledge vector to represent the blood samples with blood group $O+$ and $O-$. Remark that a latent representation is a vectorial knowledge for all query semantics with the same additional knowledge and database.

6.2 Improving and explaining answers

While the latent representation formalism provides a sound basis for many applications, we wish to provide a concrete use case to illustrate the real-world applications of the framework. Therefore, the validity of the following sections does not rely on formal proofs but on a concrete implementation and series of experiments presented in the last section. To be as clear as possible, we relax the formalism but will specify important remarks wrt. the approximations we are considering.

We consider a source database domain $\mathbf{S}$, a target database domain $\mathbf{T}$, an additional knowledge $\mathcal{A} \subseteq \mathcal{K}$, and $\mathbf{A}$ a latent representation of $\mathcal{A}$. We assume there exists a knowledge vector $\bar{\omega}_{\mathsf{sys}} \in \mathbf{A}$ which can capture the query evaluation system behavior such that, the for every query semantics $\mathbb{Q}$ and every database $x \in \mathbb{I}_{\mathbf{S}}$, the evaluation $\mathbb{Q}(x, \bar{\omega}_{\mathsf{sys}})$ coincide with the answer returned by the system $\mathbb{Q}(x, \bar{\omega}_{\mathsf{sys}}) = q(x)$. Moreover for the sake of simplicity we will also assume that for any knowledge vector $\omega \in \mathbf{A}$ the set of possible database returned by the semantic $\mathbb{Q}$ is reduce to a singleton.

Remark. Formally proving the existence of a knowledge vector $\bar{\omega}_{\mathsf{sys}} \in \mathbf{A}$, which can capture the query evaluation system behavior, is not reasonable in most cases. It requires a formal verification of the query evaluation system.

6.2.1 Improving answers

As a first step, we focus on a simple use case for latent representations of the knowledge. We want to help the user reduce the risk of the answer produced by its query evaluation system.

Our goal is to predict how much the risk will reduce if we clean the incompleteness from a source. Let a source of incompleteness X_i, assume that after cleaning X_i we discover its true characteristic is $\omega_i \in X_i$. Then the new risk is the value:

$$\mathsf{newrisk}(\mathbb{Q}, x, \omega_i) = \max_{\bar{\omega} \in \mathbf{A}} (\mu(\mathbb{Q}(x, \omega_i \bar{\omega}_{\mathsf{sys}}), \mathbb{Q}(x, \omega_i \bar{\omega}), \omega_i \bar{\omega}))$$

It corresponds to the maximal regret of considering the knowledge $\bar{\omega}_{\mathsf{sys}}$ everywhere but for the feature X_i for which the characteristic is known to be ω_i.

Remark. This first step assumes that the query evaluation semantics returns a single database for every knowledge vector. It also requires the vectorial Knowledge to be a latent representation.

However, we do not know the true characteristic of the feature X_i. Otherwise, it would be complete. Hence our goal is to propose models to infer the value of newrisk for an unknown characteristic. We denote the unknown true risk value after cleaning X_i with an unknown characteristic newrisk$(\mathbb{Q}, x, X_i)$.

Definition 6.2.1 (Feature cleaning-importance)**.** *For every query semantics* $\mathbb{Q}$ *every database* x*, the* cleaning-importance *of a feature* $X_i \in \mathbf{A}$ *is defined as:*

$$\text{ground-truth}(\mathbb{Q}, x, X_i) = \text{risk}_{\mathbf{A}}(\mathbb{Q}, x)[\bar{\omega}_{\text{sys}}] - \text{newrisk}(\mathbb{Q}, x, X_i)$$

Then the *cleaning-importance* of a feature X_i corresponds to the evolution of the risk value when we clean the feature X_i.

Optimistic and pessimistic inferences

As we do not know the true characteristic of the feature X_i, we propose an optimistic and a pessimistic approach to the cleaning-importance of a feature X_i. The pessimistic prediction assumes the worst possible characteristic with respect to risk improvement:

$$\text{Pessim}(\mathbb{Q}, x, X_i) = \text{risk}_{\mathbf{A}}(\mathbb{Q}, x)[\bar{\omega}_{\text{sys}}] - \inf_{\omega_i \in X_i} (\text{newrisk}(\mathbb{Q}, x, \omega_i))$$

The optimistic prediction assumes the best possible characteristic with respect to the risk improvement:

$$\text{Optimi}(\mathbb{Q}, x, X_i) = \text{risk}_{\mathbf{A}}(\mathbb{Q}, x)[\bar{\omega}_{\text{sys}}] - \sup_{\omega_i \in X_i} (\text{newrisk}(\mathbb{Q}, x, \omega_i))$$

And by definition the following holds:

$$\text{Pessim}(\mathbb{Q}, x, X_i) \leq \text{ground-truth}(\mathbb{Q}, x, X_i) \leq \text{Optimi}(\mathbb{Q}, x, X_i)$$

While the optimistic and pessimistic approaches have some guarantees, the gap between the two values can be arbitrarily big. Moreover, even if not critical for data-cleaning tasks, the cost of computing the optimistic and pessimistic cleaning-importances is similar to the cost of computing the risk, hence often intractable. For those reasons, we propose a heuristic approach.

Linear inference

The Linear heuristic for cleaning importance assumes that regret and risk values are aligned in the latent representation of the knowledge. The assumption is illustrated by the following formalization, for every characteristic ω_i, ω_j we have:

$$\begin{aligned}\mu(\mathbb{Q}(x, \bar{\omega}_{\mathsf{sys}}), \mathbb{Q}(x, \omega_i\omega_j\bar{\omega}_{\mathsf{sys}}), \omega_i\omega_j\bar{\omega}_{\mathsf{sys}}) =& \mu(\mathbb{Q}(x, \bar{\omega}_{\mathsf{sys}}), \mathbb{Q}(x, \omega_i\bar{\omega}_{\mathsf{sys}}), \omega_i\bar{\omega}_{\mathsf{sys}}) \\ &+ \mu(\mathbb{Q}(x, \omega_i\bar{\omega}_{\mathsf{sys}}), \mathbb{Q}(x, \omega_i\omega_j\bar{\omega}_{\mathsf{sys}}), \omega_i\omega_j\bar{\omega}_{\mathsf{sys}})\end{aligned}$$

With this assumption the following holds:

$$\mathsf{risk}_{\mathbf{A}}(\mathbb{Q}, x)[\bar{\omega}_{\mathsf{sys}}] = \max_{\omega_i \in X_i} (\mu(\mathbb{Q}(x, \bar{\omega}_{\mathsf{sys}}), \mathbb{Q}(x, \omega_i\bar{\omega}_{\mathsf{sys}}), \omega_i\bar{\omega}_{\mathsf{sys}}) + \mathsf{newrisk}(\mathbb{Q}, x, \omega_i))$$

Then we infer the risk value after cleaning $\mathsf{newrisk}(\mathbb{Q}, x, X_i)$, with the following simplification:

$$\mathsf{risk}_{\mathbf{A}}(\mathbb{Q}, x)[\bar{\omega}_{\mathsf{sys}}] \approx \max_{\omega_i \in X_i} (\mu(\mathbb{Q}(x, \bar{\omega}_{\mathsf{sys}}), \mathbb{Q}(x, \omega_i\bar{\omega}_{\mathsf{sys}}), \omega_i\bar{\omega}_{\mathsf{sys}})) + \mathsf{newrisk}(\mathbb{Q}, x, X_i)$$

And now we can infer the cleaning-importance.

Definition 6.2.2 (Linear heuristic for feature cleaning-importance)**.** *For every query semantics* $\mathbb{Q}$ *every database* x*, the* linear heuristic for cleaning-importance *of a feature* $X_i \in \mathbf{A}$ *is defined as:*

$$\mathsf{Linear}(\mathbb{Q}, x, X_i) = \max_{\omega_i \in X_i} (\mu(\mathbb{Q}(x, \bar{\omega}_{\mathsf{sys}}), \mathbb{Q}(x, \omega_i\bar{\omega}_{\mathsf{sys}}), \omega_i\bar{\omega}_{\mathsf{sys}}))$$

The evaluation of $\mathsf{Linear}(\mathbb{Q}, x, X_i)$ can be arbitrarily hard, but with a reasonable regret function, its complexity only depends on the evaluation of $\mathbb{Q}$ and the size of the set X_i.

Remark. None of the assumptions and simplifications considered by the linear heuristic are correct in general. The goal is not to provide an approximation with theoretical guarantees but to illustrate the intuition behind the heuristic we are considering.

6.2.2 Explaining the risk of answers

The cleaning-importance should be considered when a user wishes to know which features to clean to reduce the risk optimally. However, the cleaning-importance is not a good measurement of the contribution to the risk. Therefore it should not be used to explain the risk values.

First, the values of the cleaning-importance over all the sources of incompleteness do not sum to the risk. Moreover, the cleaning-importance notion neglects the potential correlation between features and considers the cleaning of each source of incompleteness in isolation. Therefore, we wish to define the *risk-contribution* of the feature with Shapley values. Indeed the notion produced satisfactory results not only with databases Livshits, Bertossi, Kimelfeld, and Sebag (2019) but also in other domains of computer science Cesari, Algaba, Moretti, and Nepomuceno (2018); Hunter and Konieczny (2010); Michalak, Aadithya, Szczepanski, Ravindran, and Jennings (2013).

We want to build a cooperative game where each player controls a feature of the latent representation. The value of the coalition game corresponds to the cleaning-importance of the set of features in the coalition. Hence we need to extend the notion of newrisk to sets of characteristics.

Let a set of characteristics $\{\omega_j, \cdots, \omega_k\}$, then the new risk value if we discover that the features $\{X_j, \cdots, X_k\}$ behave according to the characteristics in $\{\omega_j, \cdots, \omega_k\}$ is:

$$\text{newrisk}(\mathbb{Q}, x, \{\omega_j, \cdots, \omega_k\}) = \max_{\bar{\omega} \in \mathbf{A}} (\mu(\mathbb{Q}(x, \omega_j \cdots \omega_k \bar{\omega}_{\text{sys}}), \mathbb{Q}(x, \omega_j \cdots \omega_k \bar{\omega}), \omega_j, \cdots \omega_k \bar{\omega}))$$

It corresponds to the maximal regret of considering the knowledge $\bar{\omega}_{\text{sys}}$ everywhere but for the feature $X_i \in \{X_j, \cdots, X_k\}$ for which the characteristic is known to be ω_i.

We denote the unknown true risk value after cleaning the features $\{X_j, \cdots, X_k\}$ with unknown characteristics newrisk$(\mathbb{Q}, x, \{X_j, \cdots, X_k\})$. Then the *cleaning-importance* of a set of features $\{X_j, \cdots, X_k\}$ is:

$$\text{ground-truth}(\mathbb{Q}, x, \{X_j, \cdots, X_k\}) = \text{risk}_{\mathbf{A}}(\mathbb{Q}, x)[\bar{\omega}_{\text{sys}}] - \text{newrisk}(\mathbb{Q}, x, \{X_j, \cdots, X_k\})$$

It corresponds to the evolution of the risk value when we clean the set of features $\{X_j, \cdots, X_k\}$.

We also extend the linear heuristic for the *cleaning-importance* to a set of features $\{X_j, \cdots, X_k\}$ such that:

$$\text{Linear}(\mathbb{Q}, x, \{X_j, \cdots, X_k\}) = \max_{(\omega_j, \cdots, \omega_k) \in (X_j, \cdots, X_k)} (\mu(\mathbb{Q}(x, \bar{\omega}_{\text{sys}}), \mathbb{Q}(x, \omega_j \cdots \omega_k \bar{\omega}_{\text{sys}}), \omega_j \cdots \omega_k \bar{\omega}_{\text{sys}}))$$

Remark. While the cost of computing the linear heuristic for a single feature is reasonable, for a set of features, computing the linear heuristic is as hard as computing the risk. Indeed if we consider the whole set of features we have:

$$\text{Linear}(\mathbb{Q}, x, \mathbf{A}) = \text{ground-truth}(\mathbb{Q}, x, \mathbf{A}) = \text{risk}_{\mathbf{A}}(\mathbb{Q}, x)[\bar{\omega}_{\text{sys}}]$$

We want to build a function ψ such that $\psi(X_i, \bar{\omega})$ is the contribution of the feature X_i to the risk of the knowledge vector $\bar{\omega}_{\text{sys}}$.

Proposition 18 (Shapley value Characterisation Roth (1988))**.** *Let a database $x \in \mathbb{I}_{\mathbf{S}}$. The Shapley value function ψ is the only function such that for every feature X_i, X_j, and every query semantics $\mathbb{Q}$ and $\mathbb{Q}'$ the following hold:*

1. *Efficiency: The sum of the player's contribution is equal to the game's overall value.*

$$\sum_{i=0}^{n} \psi(\mathbb{Q}, x, X_i) = \text{risk}_{\mathbf{A}}(\mathbb{Q}, x)[\bar{\omega}_{\text{sys}}]$$

2. *Null Player: If a feature contributes for nothing in every coalition, its overall contribution is 0.*

$$\forall S \subseteq \mathbf{A}, \text{ ground-truth}(\mathbb{Q}, x, S) = \text{ground-truth}(\mathbb{Q}, x, S \cup \{X_i\}) \implies \psi(\mathbb{Q}, x, X_i) = 0$$

3. *Symmetry: If two features contribute equally to every coalition, their overall contributions are equal.*

$$\forall S \subseteq \mathbf{A} \setminus \{X_i, X_j\}, \text{ ground-truth}(\mathbb{Q}, x, S \cup \{X_i\}) = \text{ground-truth}(\mathbb{Q}, x, S \cup \{X_j\}) \implies \psi(\mathbb{Q}, x, X_i) = \psi(\mathbb{Q}, x, X_j)$$

4. *Additivity: The contribution of a feature in the additive game is equal to the sum of its marginal contributions.*

$$\psi(\mathbb{Q} + \mathbb{Q}', x, X_j) = \psi(\mathbb{Q}, x, X_j) + \psi(\mathbb{Q}', x, X_j)$$

   Where for every subset of features S the additive game is given by:

$$\text{ground-truth}(\mathbb{Q} + \mathbb{Q}', x, S) = \text{ground-truth}(\mathbb{Q}', x, S) + \text{ground-truth}(\mathbb{Q}', x, S)$$

Moreover the Shapley value has an analytical form:

$$\psi(\mathbb{Q}, x, X_j) = \sum_{S \subseteq \mathbf{A} \setminus X_i} \frac{|S|!\,(n - |S| - 1)!}{n!} (\text{ground-truth}(\mathbb{Q}, x, S \cup \{X_i\}) - \text{ground-truth}(\mathbb{Q}, x, S))$$

The characterization of the Shapley value tells us that it is the only notion of contribution with our desired properties. First, we expect the sum of every feature's contribution to equal the overall risk of the knowledge vector (Efficiency). Second, if a feature does not influence the risk, we expect its contribution to be 0 (Null player). Third, if two features influence the risk similarly, we want their contribution to be equal (Symmetry).

Finally, consider a feature and its cleaning-importance for two different query evaluations. As the two query evaluations are independent, it is reasonable to expect that the feature contribution to the risk of the two query evaluation is equal to the sum of its marginal contributions (Additivity).

Definition 6.2.3 (Feature risk-contribution)**.** *For every query semantics* $\mathbb{Q}$ *every database* x*, the* risk-contribution *of a feature* $X_i \in \mathbf{A}$ *is defined as the shapley value* $\psi(\mathbb{Q}, x, X_j)$ *of the coalition game* ground-truth$(\mathbb{Q}, x, \cdot)$*.*
And the Linear heuristic for the risk-contribution is defines as the shapley value $\psi(\mathbb{Q}, x, X_j)$ *of the coalition game* Linear$(\mathbb{Q}, x, \cdot)$*.*

Intuitively, the Shapley values of the features involved in the computation of a query semantics $\mathbb{Q}$ on a database x allow us to explain the risk. Indeed the Shapley value of a feature X_i does not only depends on its isolated cleaning-importance. It also accounts for its cleaning-importance if we have already cleaned other features.

Example 14. .

Blood Type

Name	Blood Group
John	$AB-$
Alice	$A+$
Jane	$B-$
Trudy	$A-$
Peter	$AB-$
Etienne	$AB-$

We consider a population $\{\text{John}, \text{Alice}, \text{Jane}, \text{Trudy}, \text{Peter}\}$ such that their respective blood type is given above. We wish to use an unidentified blood sample to perform transfusions. We consider a regret equal to the number of individuals who have a bad reaction if we transfuse them with this blood. We performed a test on our blood sample and identified it as $A-$. However, we know our test is not accurate. Therefore, what is the risk associated with the knowledge vector $A-$? We consider an abstract query semantics $\mathbb{Q}$ representing our transfusion mapping with the blood group and rhesus factor features. The database d is the population we are considering.
If we consider that the sample is $A-$, we will transfuse the blood to John, Alice, Peter, Etienne, and Trudy. Therefore if the sample happens to be $AB+$ our regret will be 5 because John, Trudy, Peter, Etienne, and Alice will have a bad reaction: risk$(\mathbb{Q}, d)[A-] = 5$.

Now assume we can perform more expensive tests to accurately identify the blood group $\{A, B, AB, O\}$, and the rhesus factor $\{+, -\}$. The question is, which test should we perform to minimize the risk?

If we perform the blood group $\{A, B, AB, O\}$ test, then we obtain the following new risk value depending on the outcome:

- If the test returns A, our knowledge vector does not change. However, as we can not be wrong about the blood group anymore, the risk diminish and becomes 4. Indeed if we happen to be wrong about the rhesus factor and the blood sample is $A+$ then John, Trudy, Peter and Etienne will have a bad reaction: newrisk$(\mathbb{Q}, d, A) = 4$. If the test returns O, then our knowledge vector becomes $O-$. Therefore we will consider transfusing everybody with the blood sample. And as we can only be wrong about the rhesus factor, the worst-case is when the blood sample is $O+$. Then everybody but Alice have a bad reaction, and the new risk newrisk$(\mathbb{Q}, d, O)$ is equal to 5.
- If the test returns B, our knowledge vector becomes $B-$. Therefore we will consider transfusing John, Jane, Peter, and Etienne. And as we can only be wrong about the rhesus factor, the worst-case scenario is when the blood sample is $B+$. Then they all have a bad reaction, and the new risk newrisk$(\mathbb{Q}, d, B)$ is equal to 4.
- If the test returns AB, then our knowledge vector becomes $AB-$. Therefore we will consider transfusing John, Peter, and Etienne. And as we can only be wrong about the rhesus factor, the worst-case scenario is when the blood sample is $AB+$. Then they all have a bad reaction, and the new risk newrisk$(\mathbb{Q}, d, B)$ is equal to 3.

From there we obtain the optimistic and pessimistic cleaning importance for the blood group source of incompleteness:

$$\text{Optimi}(\mathbb{Q}, d, \{A, B, AB, O\}) = 5 - 3 = 2$$
$$\text{Pessim}(\mathbb{Q}, d, \{A, B, AB, O\}) = 5 - 5 = 0$$

Finally, the Linear cleaning importance is equal to the maximal regret of considering $A-$ while it is either $AB-$,$B-$, or $O-$. The worst case is when the blood sample happens to be $AB-$, and we have:

$$\text{Linear}(\mathbb{Q}, d, \{A, B, AB, O\}) = 2$$

Obviously, we can not know the true value of newrisk$(\mathbb{Q}, d, \{A, AB, O, B\})$, but we know that the ground truth value is either 0, 1, or 2.

Similarly if we perform the rhesus factor test we obtain the following values:

$$\begin{aligned}\text{newrisk}(\mathbb{Q}, d, -) &= 2\\ \text{newrisk}(\mathbb{Q}, d, +) &= 1\\ \text{Optimi}(\mathbb{Q}, d, \{+, -\}) &= 5 - 1 = 4\\ \text{Pessim}(\mathbb{Q}, d, \{+, -\}) &= 5 - 2 = 3\\ \text{Linear}(\mathbb{Q}, d, \{+, -\}) &= 4\end{aligned}$$

Our measures of cleaning-importance do not agree on the values associated with each feature. Still, their ranking is similar, and we should perform the rhesus test to minimize the risk associated with the sample.

Now we wish to compute the Shapley value associated with A and $-$ for the Linear$(\mathbb{Q}, x, \cdot)$ game. We can not compute the Shapley value for the ground-truth$(\mathbb{Q}, x, \cdot)$ game as we need to know the real characteristics of the blood sample.

First, we have to compute the linear heuristic for the cleaning-importance of each subset of characteristics:

$$\begin{aligned}\text{Linear}(\mathbb{Q}, d, \emptyset) &= 0\\ \text{Linear}(\mathbb{Q}, d, \{\{A, AB, O, B\}, \{-, +\}\}) &= 5\end{aligned}$$

Then we use the Shapley value analytical form, and we obtain:

1. The linear heuristic of risk-contribution for rhesus factor feature with the knowledge vector $(A, -)$ is:

$$\psi(\{+, -\}, \mathbb{Q}, x) = \frac{1}{2} \cdot (4 - 0) + \frac{1}{2} \cdot (5 - 2) = 3.5$$

2. The linear heuristic of risk-contribution for the blood group feature with the knowledge vector $(A, -)$ is:

$$\psi(\{A, AB, O, B\}, \mathbb{Q}, x) = \frac{1}{2} \cdot (2 - 0) + \frac{1}{2} \cdot (5 - 4) = 1.5$$

First, notice that the sum of the risk contribution equals the knowledge vector's overall risk:

$$\text{risk}(\mathbb{Q}, d)[A-] = \psi(\{+, -\}, \mathbb{Q}, x) + \psi(\{A, AB, O, B\}, \mathbb{Q}, x)$$

Moreover, the ratio between the risk contributions of A and $-$ and the ratio of their cleaning-importance is different. The relative contribution given by the Shapley value to the blood group characteristic A is lower than its relative cleaning-importance. It is because the Shapley value does not only care about the first test we conduct. Indeed if we first clean the group type feature, $\{A, B, O, AB\}$ it leads to a risk reduction by 2. But cleaning it after the rhesus factor has been cleaned leads to a risk reduction of 1. With this in mind, we understand why the ratios are differents.

By the mean of vectorial knowledge and latent representation, we have been able to provide an abstract notion for the sources of incompleteness in databases, namely the features of the latent representation. Then we propose a notion of cleaning-importance, which should be use if we wish to clean our incompletness one sources at a time. We also proposed a notion of risk-contribution based on the shapley value and should be used to provide an explanation to the risk value. The risk-contribution is a value accounting for the correlation between features.

In the next sections we will provide a latent representation for the non-information interpretation of null value for the relational algebra with null, aggregates and grouping queries. Finally we will run some experiments to study the validity of the framework.

6.3 Applications on SQL database management system

We wish to build vectorial knowledge for the relational database domain with null. Recall that the features of a vectorial knowledge are abstractions of the sources of incompleteness. In the context of relational databases, the straightforward sources of incompleteness are the null placeholders. Assume that every null placeholder in a database d can be uniquely identified (it is always possible, even without the marked null model of incompleteness). Then for every null $\bot_i$ we consider a feature X_i. The set of characteristics in the feature X_i varies depending on the interpretation of null we are considering. For instance, with the existing but unknown interpretation of null, the characteristics of X_i coincide with the set of Const. It represents the fact that each null $\bot_i$ can be instantiated to any constant element. When we consider the no-information interpretation of null, the characteristics of X_i are more complicated. The no-information interpretation of null can only be captured by the mean of the **NI**-evaluation semantics. Therefore the characteristics of a feature X_i, representing the possible behaviors of $\bot_i$, depends on the query one wants to study. For example, considering a query semantics $q_{\mathbf{NI}}$, a characteristic in X_i must specify the expected evaluation of every predicate of $q_{\mathbf{NI}}$ involving $\bot_i$. While we can formally define such a vectorial knowledge, it has little to no practical interest.

First, the size of this vectorial knowledge depends on the source database d. Its number of features is equal to the number of nulls in the databases. It is common to assume that the size of queries is negligible compared to the size of the source databases. If the size of the vectorial knowledge depends on the source databases, we can not use this assumption anymore, and computing the contribution of the features becomes expensive. Formally, we have to consider the combined complexity of the query evaluation instead of the data complexity M. Vardi (1986). Second, we wish to implement the framework on top of existing query evaluation systems. While in theory, we can specify the behavior of each null independently, in practice, there is no RDBMS able to do so. Indeed, RDBMSs use a single null placeholder to represent incomplete information. And if the RDBMS can not differentiate two null placeholders, it must handle them similarly. And RDBMS can not distinguish null placeholders in the same attribute without implementing marked nulls. To the best of our knowledge there is no implementation of relational database system with marked null yet. Therefore we propose a vectorial knowledge for relational databases based on the attributes.

6.3.1 Attribute-consistent NI-evaluation semantics

We wish to build a vectorial knowledge for the relational database domain such that the sources of incompleteness are the attribute of the input databases. For every attribute α, we consider a feature X_α. A characteristic of the feature X_α should specify the behavior of every null in the column α of the database wrt. the evaluation of a query semantics $q_{\mathbf{NI}}$. However, as explained before, it is not reasonable in practice. Hence we impose the evaluation of predicates to be consistent upon every null of the same attribute. Every null in the same column has the same behavior. And We defined the restriction of the **NI**-evaluation semantics, assuming that all the nulls in a column behave similarly.

Remark. With this restriction, we can not build a latent representation for the no-information query evaluation semantics. Indeed some possible answers returned by $q_{\mathbf{NI}}$ can only be obtained if we allow nulls of the same column to behave differently. However, Theorem 8 and Proposition 15 ensure that for queries without aggregates and groupings, we will still be able to capture the largest and the smallest possible answer. Recall that the largest possible answer returned by the no-information query semantics coincides with the evaluation-based certain answers with the whole-world assumption. In the proof of Theorem 8 we provided a query rewriting to compute the largest answer. And the query rewriting is compatible with the assumption that every

null in a column behaves similarly. Because we only allow functions, aggregates, and groupings for the top query level, with most regret functions and answers, we expect the risk to be fairly similar to the risk upon the whole no-information query evaluation semantics.

Definition 6.3.1 (Attribute-consistent **NI**-evaluation semantics)**.** *Let a relational algebra with null query $q \in \mathrm{RA}(\Gamma)$ and d a bag relational database. The Attribute-consistent **NI**-evaluation semantics $q_{\textbf{NI-A}}$ is given by induction:*

$$
\begin{aligned}
R_{\textbf{NI-A}}(d) &= R^d \\
(q \times q')_{\textbf{NI-A}}(d) &= \{a_1 \times a_2 \mid a_1 \in q_{\textbf{NI-A}}(d), a_2 \in q'_{\textbf{NI-A}}(d)\} \\
(\pi_{(t_1,\cdots,t_l)}(q)_{\textbf{NI-A}}(d) &= \{\pi_{(t_1,\cdots,t_l)}(a) \mid a \in q_{\textbf{NI-A}}(d)\} \\
(\sigma_{\phi(\bar{\alpha})})(q)_{\textbf{NI-A}}(d) &= \left\{ a \;\middle|\; \begin{array}{l} \exists a_1 \in q_{\textbf{NI-A}}(d), \exists p \in \{\mathsf{true}, \mathsf{false}\}, \forall \bar{t} \in a_1, \\ \#\big(a, \bar{t}\big) = \begin{cases} \#\big(a_1, \bar{t}\big) & \textit{if } [\![\phi(\bar{\alpha})]\!]_{\bar{t}} = \mathsf{true} \vee \big(p \wedge [\![\phi(\bar{\alpha})]\!]_{\bar{t}} = \bot\big) \\ 0 & \textit{otherwise.} \end{cases} \end{array} \right\} \\
(\exists_{\phi(\bar{\alpha})})(q, q')_{\textbf{NI-A}}(d) &= \left\{ a \;\middle|\; \begin{array}{l} \exists a_1 \in q_{\textbf{NI-A}}(d), \exists a_2 \in q'_{\textbf{NI-A}}(d), \exists p \in \{\mathsf{true}, \mathsf{false}\}, \forall \bar{t} \in a_1, \\ \#\big(a, \bar{t}\big) = \begin{cases} \#\big(a_1, \bar{t}\big) & \textit{if } \exists \bar{t}' \in a_2 \textit{ such that} \\ & [\![\phi(\bar{\alpha})]\!]_{\bar{t}\bar{t}'} = \mathsf{true} \vee \big(p \wedge [\![\phi(\bar{\alpha})]\!]_{\bar{t}\bar{t}'} = \bot\big) \\ 0 & \textit{otherwise.} \end{cases} \end{array} \right\} \\
(\nexists_{\phi(\bar{\alpha})})(q, q')_{\textbf{NI-A}}(d) &= \left\{ a \;\middle|\; \begin{array}{l} \exists a_1 \in q_{\textbf{NI-A}}(d), \exists a_2 \in q'_{\textbf{NI-A}}(d), \exists p \in \{\mathsf{true}, \mathsf{false}\}, \forall \bar{t} \in a_1, \\ \#\big(a, \bar{t}\big) = \begin{cases} \#\big(a_1, \bar{t}\big) & \textit{if } \nexists \bar{t}' \in a_2 \textit{ such that} \\ & [\![\phi(\bar{\alpha})]\!]_{\bar{t}\bar{t}'} = \mathsf{true} \vee \big(p \wedge [\![\phi(\bar{\alpha})]\!]_{\bar{t}\bar{t}'} = \bot\big) \\ 0 & \textit{otherwise.} \end{cases} \end{array} \right\}
\end{aligned}
$$

When null placeholders are involved in the resolution of a predicate, we consider both the cases for which the predicate is true and is false as possible answers. However, notice that the decision to keep or discard the tuples for which the evaluation of a formulae return $\bot$ is done on the relation level. On the one hand, with the $q_{\textbf{NI-A}}$ semantics, the value of $p \in \{\mathsf{true}, \mathsf{false}\}$ is the same for every tuple in a relation. On the other hand, with the $q_{\textbf{NI}}$ semantics (see definition 5.2.2), p can be different for each tuple of a relation.

Vectorial Knowledge with features based on attributes

Observe that if a query does not involve a formula by the mean of $\forall$, $\exists$ or σ operations, its query evaluation semantics is reduced to a single possible answer. We only have to account for these operations to build a vectorial knowledge based on the attributes. Let a relational algebra with null query $q \in \mathsf{RA}(\Gamma)$ from $\mathbf{S}$ to $\mathbf{T}$ The set of $\exists$ resp. $\nexists$ resp. σ operations involved in the computation of q is:

$$\begin{aligned}
\exists[q] =& \{\exists_{\phi(\bar{\alpha})}(q') \mid q', q'' \in \mathsf{RA}_{\mathsf{Agg}}(\Gamma), \exists_{\phi(\bar{\alpha})}(q', q'') \in q\} \\
\nexists[q] =& \{\nexists_{\phi(\bar{\alpha})}(q', q'') \mid q', q'' \in \mathsf{RA}_{\mathsf{Agg}}(\Gamma), \nexists_{\phi(\bar{\alpha})}(q', q'') \in q\} \\
\sigma[q] =& \{\sigma_{\phi(\bar{\alpha})}(q') \mid q' \in \mathsf{RA}_{\mathsf{Agg}}(\Gamma), \sigma_{\phi(\bar{\alpha})}(q') \in q\}
\end{aligned}$$

And we denote $\mathcal{O}[q]$ the union of these sets: $\mathcal{O}[q] = \exists[q] \cup \forall[q] \cup \sigma[q]$.

Definition 6.3.2. *For every relational database $d \in \mathbb{I}_{\mathbf{S}}$ with attributes $\bar{\alpha}$,and every relational algebra with null query $q \in \mathsf{RA}_{Agg}(\Gamma)$, we define the $|\bar{\alpha}| = n$-tuple $\mathbf{A}_d^q = (X_{\alpha_1}, \cdots, X_{\alpha_n})$ such that for every $i \leq n$:*

$$X_{\alpha_i} = \left\{(\omega^{o_1}, \cdots, \omega^{o_m}) \mid m = |\mathcal{O}[q]|, \text{ and } \forall o_j \in \mathcal{O}[q] \text{ we have, } \omega^{o_j} \in \{\mathsf{true}_{\alpha_i}^{o_j}, \mathsf{false}_{\alpha_i}^{o_j}\}\right\}$$

*For every attribute knowledge vector $\bar{\omega} \in \mathbf{A}$ the Attribute-consistent **NI**-evaluation semantics with knowledge $q_{\textbf{NI-A}}$ returns a single answer given by inductively defining the quantity $\#\big(q_{\textbf{NI-A}}(d, \bar{\omega}), \bar{t}\big)$, which is the number of occurrences of a tuple $\bar{t}$ (of appropriate arity) in the result of applying $q_{\textbf{NI-A}}$ to a database d with the knowledge vector $\bar{\omega}$*

$$\begin{aligned}
\#\big(R_{\textbf{NI-A}}(d, \bar{\omega}), \bar{t}\big) &= \#\big(R^d, \bar{t}\big) \\
\#\big((q \times q')_{\textbf{NI-A}}(d, \bar{\omega}), \bar{t}\,\bar{t}'\big) &= \#\big(q_{\textbf{NI-A}}(d, \bar{\omega}), \bar{t}\big) \cdot \#\big(q'_{\textbf{NI-A}}(d, \bar{\omega}), \bar{t}'\big) \\
\#\big((\pi_{\bar{\alpha}})(q)_{\textbf{NI-A}}(d, \bar{\omega}), \bar{t}\big) &= \sum_{\bar{t}' \,:\, \pi_{\bar{\alpha}}(\bar{t}') = \bar{t}} \#\big(q_{\textbf{NI-A}}(d, \bar{\omega}), \bar{t}'\big) \\
\#\big((\sigma_{\phi(\bar{\alpha})})(q)_{\textbf{NI-A}}(d, \bar{\omega}), \bar{t}\big) &= \begin{cases} \#\big(q_{\textbf{NI-A}}(d, \bar{\omega}), \bar{t}\big) & \text{if } [\![\phi(\bar{\alpha})]\!]_{\bar{t}}^{\left(\bar{\omega}, (\sigma_{\phi(\bar{\alpha})})(q)\right)} = \mathsf{true} \\ 0 & \text{otherwise.} \end{cases} \\
\#\big(\exists_{\phi(\bar{\alpha})}(q, q')(d), \bar{t}\big) &= \begin{cases} \#\big(q_{\textbf{NI-A}}(d, \bar{\omega}), \bar{t}\big) & \text{if } \exists \bar{t}' \in q'_{\textbf{NI-A}}(d, \bar{\omega}) \text{ such that} \\ & [\![\phi(\bar{\alpha})]\!]_{\bar{t}\bar{t}'}^{\left(\bar{\omega}, (\exists_{\phi(\bar{\alpha})})(q, q')\right)} = \mathsf{true} \\ 0 & \text{otherwise.} \end{cases} \\
\#\big(\nexists_{\phi(\bar{\alpha})}(q, q')(d), \bar{t}\big) &= \begin{cases} \#\big(q_{\textbf{NI-A}}(d, \bar{\omega}), \bar{t}\big) & \text{if } \nexists \bar{t}' \in q'_{\textbf{NI-A}}(d, \bar{\omega}) \text{ such that} \\ & [\![\phi(\bar{\alpha})]\!]_{\bar{t}\bar{t}'}^{\left(\bar{\omega}, (\nexists_{\phi(\bar{\alpha})})(q, q')\right)} = \mathsf{true} \\ 0 & \text{otherwise.} \end{cases}
\end{aligned}$$

On the query level the knowledge vector $\bar{\omega}$ is just propagated to disambiguate the evaluation of the formulae such that:

$$[\![\mathcal{P}(\bar{\alpha})]\!]_{\bar{t}}^{(\bar{\omega},o)} = \begin{cases} \mathcal{P}(\bar{t}[\alpha]) & \textit{if } \forall \alpha_i \in \bar{\alpha},\ \bar{t}[\alpha_i] \in \mathsf{Const} \\ \mathsf{true} & \textit{else if } \exists \alpha_i \in \bar{\alpha},\ \left(\bar{t}[\alpha_i] = \bot \wedge \mathsf{true}_{\alpha_i}^{o} \in \bar{\omega}\right) \\ \mathsf{false} & \textit{else if } \exists \alpha_i \in \bar{\alpha},\ \left(\bar{t}[\alpha_i] = \bot \wedge \mathsf{false}_{\alpha_i}^{o} \in \bar{\omega}\right) \\ \bot & \textit{otherwise} \end{cases}$$

$$[\![\phi(\bar{\alpha}) \wedge \phi'(\bar{\alpha}')]\!]_{\bar{t}}^{(\bar{\omega},o)} = \begin{cases} \mathsf{false} & \textit{if } [\![\phi(\bar{\alpha})]\!]_{\bar{t}}^{(\bar{\omega},o)} = \mathsf{false} \vee [\![\phi'(\bar{\alpha}')]\!]_{\bar{t}}^{(\bar{\omega},o)} = \mathsf{false} \\ \mathsf{true} & \textit{if } [\![\phi(\bar{\alpha})]\!]_{\bar{t}}^{(\bar{\omega},o)} = [\![\phi'(\bar{\alpha}')]\!]_{\bar{t}}^{(\bar{\omega},o)} = \mathsf{true} \\ \bot & \textit{otherwise} \end{cases}$$

$$[\![\phi(\bar{\alpha}) \vee \phi'(\bar{\alpha}')]\!]_{\bar{t}}^{(\bar{\omega},o)} = \begin{cases} \mathsf{true} & \textit{if } [\![\phi(\bar{\alpha})]\!]_{\bar{t}}^{(\bar{\omega},o)} = \mathsf{true} \vee [\![\phi'(\bar{\alpha}')]\!]_{\bar{t}}^{(\bar{\omega},o)} = \mathsf{true} \\ \mathsf{false} & \textit{if } [\![\phi(\bar{\alpha})]\!]_{\bar{t}}^{(\bar{\omega},o)} = [\![\phi'(\bar{\alpha}')]\!]_{\bar{t}}^{(\bar{\omega},o)} = \mathsf{false} \\ \bot & \textit{otherwise} \end{cases}$$

For every relational database, $d \in \mathbb{I}_{\mathbf{S}}$ and every query $q \in \mathsf{RA}(\Gamma)$ a knowledge vector in $\mathbf{A}_d^q$ encodes the behavior of every attribute of d for every operation involving a formula of q. To control the behavior, during the resolution of a predicate $\mathcal{P}(\bar{t}[\bar{\alpha}])$ we have access to the knowledge $\bar{\omega}$ and the operation involving this predicate o.

If $\bar{t}[\bar{\alpha}]$ only contains constant, then we keep the default evaluation of the predicate.

If one of the values in $\bar{t}[\bar{\alpha}]$ is null, for instance, the value $\bar{t}[\alpha_i]$, then we have to check the content of the knowledge vector $\bar{\omega}$.
If the knowledge $\bar{\omega}[X_{\alpha_i}]^o$ associated with the feature X_{α_i} and the operation o in the vector $\bar{\omega}$ is $\mathsf{true}_{\alpha_i}^{o}$ then we enforce the resolution of the predicate to be true. We ignore the knowledge of any other null value, which may also be in $\bar{t}[\bar{\alpha}]$. It is a design choice; in the context of risk, it has no impact because we always look for the worst or the best possible outcome with knowledge.
If none of the knowledge associated with null values in $\bar{t}[\bar{\alpha}]$ is true_{*}^{o}, then if one of them is false_{*}^{o}, we enforce the resolution of the predicate to be false.

Remark. Our vectorial knowledge controls the behavior of predicate evaluation on the level of the operation. The same knowledge controls every predicate in a formula. Therefore if a null placeholder is involved in evaluating several predicates in the same operation, its knowledge will force every predicate's resolution to the same value true or false. Remark that forbidding different resolutions of predicate only makes sense because we are not allowing explicit negation in formulae.

Claim 2. *For every relational database $d \in \mathbb{I}_{\mathbf{S}}$ and every query $q \in \mathrm{RA}(\Gamma)$ we have:*

1. *$\mathbf{A}_d^q$ is a vectorial knowledge for $q_{\textbf{NI-A}}$ and d.*
2. *The evaluation of q on d coincides with the evaluation of $q_{\textbf{NI-A}}$ on d with the all* false *knowledge vector. Let $\overline{\mathsf{false}} \in \mathbf{A}_d^q$, such that for every operator $o \in \mathcal{O}[q]$ and every attribute α in d we have $\mathsf{false}^o_{\alpha_i} \in \overline{\mathsf{false}}$, then we have:*

$$q(d) = q_{\textbf{NI-A}}(d, \overline{\mathsf{false}})$$

Formally proving that $\mathbf{A}_d^q$ is a vectorial knowledge for $q_{\textbf{NI-A}}$ and d is doable but fairly tedious. It involves an inductive proof on the relational algebra with null grammar. As we wish to validate the framework experimentally, we emancipate from proof. The second claim only holds because we are not allowing explicit negation in formulae. We assume that the formulae have been transformed to equivalent ones without negation (the negations have been pushed to predicate).

Remark. We have not given a latent representation for the relational database domain and the no-information interpretation of null. The definition of a latent representation imposes a bounded number of features. However, if we consider the whole relational database domain, for any $n > 0$, we can find a database with $n + 1$ attributes. This constraint on the latent representation can be lifted by allowing an enumerable number of features for the domain and a finite number of features for each database. But it will not be enough to define a latent representation for the relational database domain. Recall that the vectorial knowledge $\mathbf{A}_d^q$ also depends on q. For each attribute, we only consider the knowledge which applies to the query q. To build a latent representation, we would have to consider an infinite number of characteristics for each attribute to account for every possible query. As we do not need a latent representation for the framework to work, and for simplicity's sake, we choose not to offer one, even if it is theoretically possible.

6.3.2 Aggregates and groupings

Most real-life relational queries involve aggregates and groupings. For instance, if we look at TPC-H "TPC Benchmark™ H Standard Specification" (2018), the standard benchmark for RDBMS, 18 out of the 21 queries use aggregates and/or groupings operations. As we want to validate our framework on TPC-H instances, we quickly extend our query language and database domain to allow aggregates and groupings operation on the top level.

With aggregates and groupings operations, the target database domain $\mathbf{T}$ is populated by elements coming from three different sets. Elements can be constant values from the set Const, or null values from $\{\perp\}$. Elements can also be numeric values from a set Num if they are the output of an aggregate operation.

A Num aggregate operation $\mathcal{F}$ of arity $n > 0$, is a mapping from bags of n-tuple in $(\text{Const} \cup \{\perp\})^n$ to a numeric element of Num. We denote Γ, a set of Num aggregate operations.

Definition 6.3.3 (Relational algebra with top-level aggregates)**.** *The relational algebra with top-level aggregates and grouping* $\text{RA}_{Agg}(\Gamma)$ *is given by the following grammar:*

$$\begin{aligned} q :=\quad & R \mid q \times q \mid \pi_{\bar{\alpha}}(q) \mid \sigma_{\phi(\bar{\alpha})}(q) \\ & \mid \exists_{\phi(\bar{\alpha})}(q,q) \mid \nexists_{\phi(\bar{\alpha})}(q,q) \\ Q :=\quad & \mathsf{Gb}_{\bar{\alpha}}\mathsf{Aggr}\left(\mathcal{F}_1(\bar{\alpha}_1), \cdots, \mathcal{F}_n(\bar{\alpha}_n)\right)(q) \quad \textit{with } \bar{\alpha}_i \cap \bar{\alpha} = \emptyset \textit{ for all } i \in [1,n] \end{aligned}$$

where $\bar{\alpha}, \bar{\alpha}_1, \cdots \bar{\alpha}_n$ *are possibly empty tuples of attributes, and* $\mathcal{F}_1, \cdots, \mathcal{F}_n$ *are aggregates from* Γ*. The semantics of a query* q *is the same as in definition 5.2.1. And for every query* $Q = \mathsf{Gb}_{\bar{\alpha}}\mathsf{Aggr}\left(\mathcal{F}_1(\bar{\alpha}_1), \cdots, \mathcal{F}_n(\bar{\alpha}_n)\right)(q)$ *and every database* d *the semantics of* Q *is given by:*

$$Q(d) = \left\{ (\bar{t}, t_1, \cdots, t_n) \;\middle|\; \begin{array}{l} \bar{t} \in \pi_{\bar{\alpha}}(q)(d) \\ \textit{and } \forall i \in [1,n], \textit{ we have } t_i = \mathcal{F}_i\left(\pi_{\bar{\alpha}_i}(\boldsymbol{B}_{\bar{t}})\right) \\ \textit{with } \boldsymbol{B}_{\bar{t}} \textit{ such that } \forall \bar{t}', \#\big(\boldsymbol{B}_{\bar{t}}, (\bar{t}, \bar{t}')\big) = \#\big(q(d), (\bar{t}, \bar{t}')\big) \end{array} \right\}$$

The evaluation of $\mathsf{Gb}_{\bar{\alpha}}\mathsf{Aggr}\left(\mathcal{F}_1(\bar{\alpha}_1), \cdots, \mathcal{F}_n(\bar{\alpha}_1)\right)(q)$ coincides with the SQL evaluation of the query:

```
SELECT ᾱ, F1(ᾱ1), ··· ,Fl(ᾱ1) FROM (q)
GROUP BY ᾱ
```

Independently of the vectorial knowledge with features attributes, we can define a query evaluation semantics for the **NI**, **EU**, and **NA** interpretations of nulls. Because we wish to study the cleaning-importance and the risk-contribution, in the next section, we present an attribute-consistent NI-evaluation semantics.

Attribute-consistent NI-evaluation semantics for aggregates and groupings

In the context of our framework, we want aggregate operations and numeric elements to be impacted by the attribute-knowledge vectors. While we can not reasonably change the behavior of every aggregate operation in Γ, we can enrich their output with valuable information. We wish for aggregate operations to output a numeric value but also information about the proportion and characteristics of the null element in its input bag. More formally we enrich the set of numeric element Num and we consider the set of enriched numeric element:

$$\mathbf{A}\text{Num} = \{(v, p, q) \mid v \in \text{Num, and } p, q \in [0, 1]\}$$

The enriched numeric elements are tuples (v, p, q), where v represents the numeric element, p the proportion of null elements in the bag, and q the proportion of null elements with at least a characteristic different from false.

Then for every query $Q = \text{Gb}_{\bar{\alpha}}\text{Aggr}\left(\mathcal{F}_1(\bar{\alpha}_1), \cdots, \mathcal{F}_n(\bar{\alpha}_1)\right)(q)$, every database d, and every attribute knowledge vector $\bar{\omega} \in \mathbf{A}$ we define the attribute-consistent NI-evaluation semantics for aggregates and groupings.

$$Q_{\textbf{NI-A}}(d, \bar{\omega}) = \left\{ (\bar{t}, t_1, \cdots, t_n) \;\middle|\; \begin{array}{l} \bar{t} \in (\pi_{\bar{\alpha}})(q)_{\textbf{NI-A}}(d, \bar{\omega}) \\ \text{and } \forall i \in [1, n], \text{ we have } t_i = (\mathcal{F}_i(\pi_{\bar{\alpha}_i}(\mathbf{B}_{\bar{t}})), p_i, q_i) \\ \text{with } \mathbf{B}_{\bar{t}} = \{\!\{(\bar{t}, \bar{t}') \mid (\bar{t}, \bar{t}') \in q(d)\}\!\} \\ \text{and } p_i = \dfrac{|\text{Null}(\pi_{\bar{\alpha}_i}(\mathbf{B}_{\bar{t}}))|}{|\text{Null}(\pi_{\bar{\alpha}_i}(\mathbf{B}_{\bar{t}}))| + |\text{Const}(\pi_{\bar{\alpha}_i}(\mathbf{B}_{\bar{t}}))|} \\ \\ \text{and } q_i = \dfrac{\sum\limits_{\alpha \in \{\alpha \in \bar{\alpha}_i \mid \bar{\omega}[\alpha] \neq \overline{\text{false}}\}} |\text{Null}(\pi_{\alpha}(\mathbf{B}_{\bar{t}}))|}{|\text{Null}(\pi_{\bar{\alpha}_i}(\mathbf{B}_{\bar{t}}))| + |\text{Const}(\pi_{\bar{\alpha}_i}(\mathbf{B}_{\bar{t}}))|} \end{array} \right\}$$

where $|\text{Null}(\pi_{\bar{\alpha}_i}(\mathbf{B}_{\bar{t}}))|$ resp. $|\text{Const}(\pi_{\bar{\alpha}_i}(\mathbf{B}_{\bar{t}}))|$ is the number of null elemets resp. constant elements in the attributes $\bar{\alpha}_i$ of the bag $\mathbf{B}_{\bar{t}}$.

In practice, to obtain the null proportion, we use the `COUNT` operator and the fact that `COUNT(*)` returns the number of elements (nulls included), and `COUNT(`α_i`)` returns the number of constant elements in α_i.

Remark. We are enriching the domain with numerical values and proportions of nulls because we will use this information to refine our regret functions. We do not have to enrich the database domain, as we could encode the relevant information in the additional knowledge and use it later for the regret function. However, with additional knowledge encoding, it didn't feel easy to understand for the reader.

6.3.3 Optimal transport based regret function with numeric element and knowledge

Based on the same principle as the OT-based regret functions introduced in definition 5.4.2, we defined a cost function inductively on the element of the relational database domain.

Generic value dissimilarity with knowledge

First, we introduce a notion of dissimilarity for generic elements in an attribute α as a function $\mathbf{C}_\alpha \colon (\mathsf{Const} \cup \{\perp\}) \times (\mathsf{Const} \cup \{\perp\}) \times \mathbf{A} \mapsto [0, \infty]$ such that, for every elements $e, e' \in \mathsf{Const} \cup \{\perp\}$ and every attribute knowledge vector $\bar{\omega} \in \mathbf{A}$ we have:

$$\mathbf{C}_\alpha(e, e', \bar{\omega}) = \begin{cases} 0 & \text{if } e = e' \text{and } e, e' \in \mathsf{Const} \\ 1 & \text{if } (e = \perp) \oplus (e' = \perp) \\ 0 & \text{if } (e = \perp) \wedge (e' = \perp) \text{ and } \bar{\omega}[\alpha] = \overline{\mathsf{false}} \\ 1 & \text{if } (e = \perp) \wedge (e' = \perp) \text{ and } \bar{\omega}[\alpha] \neq \overline{\mathsf{false}} \\ \infty & \text{otherwise.} \end{cases}$$

The information dissimilarity is minimal when the two constant values are the same. It is equal to 1 when one of the values is null and the other a constant. When the two values are null, we check if the knowledge for the attribute α is $\overline{\mathsf{false}}$. If it is, then the information content of both values is similar, and the cost is 0. If the knowledge is different from $\overline{\mathsf{false}}$, then it means the elements in α have been cleaned. Hence the information dissimilarity between the two values is the same as between a null and a constant. Finally, the information dissimilarity is maximal between two different constant values because they are generic elements; hence they have incomparable information content.

Numeric value dissimilarity

The set of elements Num contains all possible values from arithmetic and aggregation operations. We assume that for every attribute α populated with element Num there exists a normalized distance metric between the elements in α:

$$d^\alpha \colon \mathsf{Num} \times \mathsf{Num} \mapsto [0, 1]$$

Remark. In the implementation, we used the fact that we can compute an upper and lower relation for the set of answers upon the attribute knowledge with q_{false} and q_{true} (see Theorem 8 and Proposition 15). Then for every numerical attribute, we store the result of the aggregate operations **MAX** and **MIN** on q_{false} and q_{true}. This gives us the lower and upper bound of the numerical values in the attribute, and we can normalize the absolute value of the Euclidean distance.

In the case of our database domain, we consider the enriched set of numeric elements $\mathbf{A}$Num. Recall that the element of $\mathbf{A}$Num are tuple (v, p, q) such that v represents the output of an aggregation, p the proportion of null involved in the evaluation, and q the proportion of null with a knowledge different from $\overline{\mathsf{false}}$.

Let two tuples $(v_1, p_1, q_1), (v_2, p_2, q_2) \in \mathbf{A}$Num from an attribute α, a naive notion of dissimilarity would be to consider the normalized distance between the integer values v_1 and v_2. However, it neglects the possible impact of the null values involved in the computation of v_1 and v_2. Because the numeric values are computed with aggregates over bags, our idea to refine the dissimilarity is to, once again, instantiate an optimal transport problem.

We represent a tuple $(v_1, p_1, q_1) \in \mathbf{A}$Num as an histogram $\mathcal{H}((v_1, p_1, q_1))$ with three positions:

- v_1 representing the integer values with a mass $(1 - p_1)$.
- $\bot_{\mathsf{false}}$ representing the nulls with a knowledge $\overline{\mathsf{false}}$ with a mass $(p_1 - q_1)$.
- $\bot$ representing the nulls with a knowledge different than $\overline{\mathsf{false}}$ with a mass q_1.

The cost of transporting the mass of the histograms between two positions is given by a function C as follows:

- For every $v_1, v_2 \in$ Num, $C(v_1, v_2) = d^\alpha$ where d^α is any normalized distance measure on the set of numerical values in α.
- The transporting cost between two positions $\bot_{\mathsf{false}}$ is 0: $C(\bot_{\mathsf{false}}, \bot_{\mathsf{false}}) = 0$
- The transporting cost between two positions $\bot$ is 0: $C(\bot, \bot) = 0$
- The transporting cost is 1 everywhere else.

Then the dissimilarity value between (v_1, p_1, q_1) and (v_2, p_2, q_2) is equal to the cost of the optimal transportation plan between their histogram:

$$\mathbf{C}_\alpha^{\mathsf{Num}}((v_1, p_1, q_1), (v_2, p_2, q_2)) = K_C\big(\mathcal{H}(v_1, p_1, q_1), \mathcal{H}(v_2, p_2, q_2)\big)$$

As all the cost are bounded by 1 and the sum of the weight of each histogram is 1 we have:

$$0 \leq \mathbf{C}_\alpha^{\mathsf{Num}}((v_1, p_1, q_1), (v_2, p_2, q_2)) \leq 1$$

Remark. When we attribute a mass $(1 - p_1)$ to the position v_1 in the histogram $\mathcal{H}((v_1, p_1, q_1))$, we assume that every constant element involved in the aggregate function are equal contributors to the outcome. Therefore we uniformize the contribution of every constant value in the bag. This assumption is generally false, even with simple aggregate functions. For instance, consider an aggregation function returning the maximal value of a bag. Hence there is no mathematical guarantee toward the relevance of our dissimilarity function for Num values. However, building a cost function with mathematical guarantees is possible if a user has a specific use case and does not need to be as general (restrict the set of aggregate functions).

6.3.4 Information dissimilarity between tuples

For every tuple $\bar{t}, \bar{t}'$ with attributes $\bar{\alpha}$, we denote $\bar{\alpha}[\text{Const}]$ the set of attributes with generic elements and $\bar{\alpha}[\text{Num}]|$ the set of attributes with numeric elements. Then the information dissimilarity between the tuples is defined as the $(\text{Cost}_{\text{unify}}, \text{Cost}_{\text{numeric}})$-weighted average of the generic dissimilarity and the numeric dissimilarity.

$$\mathbf{C}_{\bar{\alpha}}(\bar{t}, \bar{t}', \bar{\omega}) = \text{Cost}_{\text{unify}} \cdot \frac{\sum\limits_{\alpha \in \bar{\alpha}[\text{Const}]} \mathbf{C}_\alpha(\bar{t}[\alpha], \bar{t}'[\alpha], \bar{\omega})}{|\bar{\alpha}[\text{Const}]|} + \text{Cost}_{\text{numeric}} \cdot \frac{\sum\limits_{\alpha \in \bar{\alpha}[\text{Num}]} \mathbf{C}_\alpha^{\text{Num}}(\bar{t}[\alpha], \bar{t}'[\alpha])}{|\bar{\alpha}[\text{Num}]|}$$

with $\text{Cost}_{\text{unify}}$, $\text{Cost}_{\text{numeric}}$ accounting for the user preferences. The higher $\text{Cost}_{\text{unify}}$ is with respect to $\text{Cost}_{\text{numeric}}$ the more important the generic values are.

Remark. The cost function for generic elements $\mathbf{C}_\alpha$ depends on the attribute α and the knowledge $\bar{\omega}$. In contrast, the cost function for numeric elements does not depend on the knowledge $\bar{\omega}$. It is a misrepresentation of the situation to simplify the notation. Remember that we enriched our database domain with $\mathbf{A}$Num instead of Num. Thanks to that, the relevant information contained in the knowledge vector $\bar{\omega}$ has been added to the output of the aggregate functions. Hence the cost function between numeric elements also depends on the knowledge.

Tuple mass assignment

Recall the mass-assignment we considered for query without grouping:

Definition 6.3.4 (Mass assignment)**.** *For every bag relation* $\mathbf{a}$*,* $\mathbf{b}$ *we define the* mass assignment *as the function* $M[\mathbf{a} \mapsto \mathbf{b}]$ *such that for every tuple with fitting attribute:*

$$M[\mathbf{a} \mapsto \mathbf{b}](\mathbf{a}, \bar{t}) = \begin{cases} |\mathbf{b}| & \textit{if } \bar{t} \textit{ is} \text{ producer} \\ \#\big(\mathbf{a}, \bar{t}\big) & \textit{otherwise.} \end{cases}$$

$$M[\mathbf{a} \mapsto \mathbf{b}](\mathbf{b}, \bar{t}) = \begin{cases} |\mathbf{a}| & \textit{if } \bar{t} \textit{ is} \text{ sinker} \\ \#\big(\mathbf{b}, \bar{t}\big) & \textit{otherwise.} \end{cases}$$

And the cost function between tuple:

$$\mathbf{C}_{\bar{\alpha}}(\bar{t}, \bar{t}') = \begin{cases} 0 & \text{if } \bar{t} \text{ is } \textit{producer} \wedge \bar{t}' \text{ is } \textit{sinker} \\ \text{Cost}_{\text{Prod}} & \text{if } \bar{t} \text{ is } \textit{producer} \\ \text{Cost}_{\text{Sink}} & \text{if } \bar{t}' \text{ is } \textit{sinker} \\ \mathbf{C}_{\bar{\alpha}}(\bar{t}, \bar{t}') & \text{otherwise.} \end{cases}$$

The mass of the *producer* is equal to the size of $\mathbf{b}$, so each missing tuple can be produced with a regret $\text{Cost}_{\text{Prod}}$. Similarly, the mass of the *sinker* is equal to the size of $\mathbf{a}$, so each completely false tuple can be discarded with a regret $\text{Cost}_{\text{Sink}}$. Finally, the cost of transporting the mass from the producer to the sinker is 0. Moreover, the mass of tuple $\bar{t} \in \mathbf{a}$ is equal to its multiplicity.

Now consider the following answers $\mathbf{a}$ and $\mathbf{b}$.

a

From:Const	**To**:Const	**Total**:Num
Jane	Alice	1000
$\perp$	Jane	2000

b

From:Const	**To**:Const	**Total**:Num
Jane	John	$\perp$
Alice	Jane	$\perp$
$\perp$	Jane	1000

Assume that the bags $\mathbf{a}$ and $\mathbf{b}$ are the answers returned by an aggregate (to compute **Total**) and a grouping operation on the attributes (**From**,**To**). Due to the behavior of group by, $\mathbf{a}$ and $\mathbf{b}$ should not be considered as bags anymore, but as sets. Therefore the tuple (Alice,Jane,$\perp$) $\in \mathbf{b}$ should not be considered as brand new information, as it unifies with ($\perp$,Jane,2000) $\in \mathbf{a}$ and multiplicity of tuples does not matter anymore.

To retain the previous properties concerning missing tuples and false tuples while also discarding the tuple multiplicity, we introduce the concept of *ghost tuples*. For each tuple in the answers, we create a ghost tuple that carries similar information content, except that the cost of producing or sinking a ghost tuple is free. Informally, ghost tuples allow us to use the information of a tuple multiple times without any penalty and retain the property that every tuple should be matched at least once.

Definition 6.3.5 (GroupBy mass assignment)**.** *For every bag relation* $\mathbf{a}$, $\mathbf{b}$ *with group by attributes we define the* GroupBy mass assignment *as the function* $M_{\mathsf{Gb}}[\mathbf{a} \mapsto \mathbf{b}]$ *such that for every tuple with fitting attributes:*

$$M_{\mathsf{Gb}}[\mathbf{a} \mapsto \mathbf{b}](\mathbf{a}, \bar{t}) = \begin{cases} |\mathbf{a}| \cdot |\mathbf{b}| & \textit{if } \bar{t} \textit{ is } \text{producer} \\ |\mathbf{b}| - 1 & \textit{if } \bar{t} \textit{ is } \text{ghost tuple} \\ 1 & \textit{otherwise.} \end{cases}$$

$$M_{\mathsf{Gb}}[\mathbf{a} \mapsto \mathbf{b}](\mathbf{b}, \bar{t}) = \begin{cases} |\mathbf{a}| \cdot |\mathbf{b}| & \textit{if } \bar{t} \textit{ is } \text{sinker} \\ |\mathbf{a}| - 1 & \textit{if } \bar{t} \textit{ is } \text{ghost tuple} \\ 1 & \textit{otherwise.} \end{cases}$$

And we extend the cost function between tuple in the following way:

$$\mathbf{C}_{\bar{\alpha}}(g_{\bar{t}}, \bar{t}', \bar{\omega}) = \begin{cases} 0 & \text{if } \bar{t}' \text{ is } \textit{sinker} \\ \mathbf{C}_{\bar{\alpha}}(\bar{t}, \bar{t}', \bar{\omega}) & \text{otherwise.} \end{cases}$$

$$\mathbf{C}_{\bar{\alpha}}(\bar{t}, g_{\bar{t}'}, \bar{\omega}) = \begin{cases} 0 & \text{if } \bar{t}' \text{ is } \textit{producer} \\ \mathbf{C}_{\bar{\alpha}}(\bar{t}, \bar{t}', \bar{\omega}) & \text{otherwise.} \end{cases}$$

$$\mathbf{C}_{\bar{\alpha}}(g_{\bar{t}}, g_{\bar{t}'}, \bar{\omega}) = \mathbf{C}_{\bar{\alpha}}(\bar{t}, \bar{t}', \bar{\omega})$$

Coming back to our example, a valid transportation plan would be the following:

$$
\begin{aligned}
u(\text{(Jane,Alice,1000)} \mapsto \textit{sinker}) &= 1 && \text{(Jane,Alice,1000) is false information}\\
u((\bot\text{,Jane,2000)} \mapsto (\bot\text{,Jane,1000})) &= 1 && (\bot\text{,Jane,2000) has a generic match}\\
u(g_{(\bot\text{,Jane,2000)}} \mapsto \text{(Alice,Jane,}\bot)) &= 1 && \text{(Alice,Jane,}\bot\text{) has a generic match}\\
u(\textit{producer} \mapsto \text{(Jane,John,}\bot)) &= 1 && \text{(Jane,John,}\bot\text{) is a new information}\\
u(g_{\text{(Jane,Alice,1000)}} \mapsto \textit{sinker}) &= 2 && \text{sinking the residual ghost tuple mass}\\
u(g_{(\bot\text{,Jane,2000)}} \mapsto \textit{sinker}) &= 1 && \text{sinking the residual ghost tuple mass}\\
u(\textit{producer} \mapsto g_{\text{(Jane,John,}\bot)}) &= 1 && \text{producing the residual ghost tuple mass}\\
u(\textit{producer} \mapsto g_{\text{(Alice,Jane,}\bot)}) &= 1 && \text{producing the residual ghost tuple mass}\\
u(\textit{producer} \mapsto g_{(\bot\text{,Jane,1000)}}) &= 1 && \text{producing the residual ghost tuple mass}\\
u(\textit{producer} \mapsto \textit{sinker}) &= 2 && \text{sinking the residual producer mass}
\end{aligned}
$$

Which yield to a regret

$$\mu(\bar{a}, \bar{b}, \overline{\text{false}}) = \text{Cost}_{\text{Sink}} + \Big(\text{Cost}_{\text{numeric}} \cdot d^{\text{Total}}(2000, 1000)\Big) + (\frac{\text{Cost}_{\text{unify}}}{2} + \text{Cost}_{\text{numeric}}) + \text{Cost}_{\text{Prod}}$$

Definition 6.3.6 (OT regret functions with knowledge)**.** *For every $\text{Cost}_{\text{numeric}}$, $\text{Cost}_{\text{unify}}$, $\text{Cost}_{\text{Prod}}$, $\text{Cost}_{\text{Sink}} \in \mathbb{N}$ the OT-regret function with knowledge $\mu \colon \mathbb{I}_{\mathbf{T}} \times \mathbb{I}_{\mathbf{T}} \times \mathbf{A} \mapsto \mathbb{R}^+$ is defined for every set relational databases $\mathbf{a}, \mathbf{b} \in \mathbb{I}_{\mathbf{T}}$ with attributes $\bar{\alpha}$ as the optimal solution to the transportation problem with the mass assignement $M_{\text{Gb}}[\mathbf{a} \mapsto \mathbf{b}]$ and the cost function with knowledge $\mathbf{C}_{\bar{\alpha}}$:*

$$\mu(\mathbf{a}, \mathbf{b}, \bar{\omega}) = K_{\mathbf{C}_{\bar{\alpha}}}\big(M_{\text{Gb}}[\mathbf{a} \mapsto \mathbf{b}](\mathbf{a}), M_{\text{Gb}}[\mathbf{a} \mapsto \mathbf{b}](\mathbf{b})\big)$$

6.4 Experiments

We hypothesize that the attribute-based vectorial knowledge can be used to infer the evolution of the risk associated with a query if we clean the incompleteness in one attribute of the source database. To confirm or falsify our hypothesis, we need to compare the prediction of our models with the actual value obtained when we clean an attribute. Therefore we simulate an environment where we have control over the incompleteness.

6.4.1 Validation procedure

Our environment is based on a randomly generated TPC-H database c. As TPC-H only generates databases without null, the database c is complete. To create some incompleteness, we substitute some constants in the attribute α of the database c with null values (if α is not a Primary key) and create the database: Nullify$(c, \alpha) = d$. In this controlled environment, cleaning an attribute α of the database d accounts to update d with its matching value in c. Such a process can easily be done thanks to the primary keys. Therefore, we have access to a function Nullify^{-1}, such that for every complete database and every attribute α:

$$c = \text{Nullify}^{-1}\left(\text{Nullify}(c, \alpha), \alpha\right)$$

We can trivially extend the Nullify, and Nullify^{-1} process to set of attributes. It allows us to define the ground-truth for the evolution of the risk when we clean an attribute $\alpha_i \in \bar{\alpha}$ in a database $d = \text{Nullify}(c, \bar{\alpha})$:

$$\text{ground-truth}(q, d, \alpha_i) = \text{risk}_{\mathbf{A}}(q_{\textbf{NI-A}}, d)[\overline{\text{false}}] - \text{risk}_{\mathbf{A}^q_d}(q_{\textbf{NI-A}}, \text{Nullify}^{-1}(d, \alpha_i))[\overline{\text{false}}]$$

Then for every prediction model $\in \{\text{Linear}, \text{Optimi}, \text{Pessim}\}$ we can study the error-rate:

$$\text{Error-rate}(\text{model}, q, d, \alpha_i) = \frac{\text{model}(q, d, \alpha_i) - \text{ground-truth}(q, d, \alpha_i)}{\text{risk}_{\mathbf{A}}(q_{\textbf{NI-A}}, d)[\overline{\text{false}}]}$$

When the overall risk is equal to 0 we also have model$(q, d, \alpha_i) =$ ground-truth$(q, d, \alpha_i) = 0$. Therefore when risk$_{\mathbf{A}}(q_{\textbf{NI-A}}, d)[\overline{\text{false}}] = 0$, we defined Error-rate$(\text{model}, q, d, \alpha_i)$ as 0.

The closer to 0 the error rate of a model, the better its predictions are. However, to discuss the validity of the hypothesis, we also need a baseline model. Indeed assume that the error rate returned by a model is $0,00001$. Despite the immediate intuition, this value does not confirm any hypothesis about the model. The prediction task may be trivial, and any model, even the most trivial, would have a low error rate. As a baseline for comparison, we propose the following naive prediction model:

$$\text{baseline}(q, d, \alpha_i) = \frac{|\text{Null}(\pi_{\alpha_i}(d))|}{\sum\limits_{\alpha \in q} |\text{Null}(\pi_{\alpha}(d))|} \cdot \text{risk}_{\mathbf{A}}(q_{\textbf{NI-A}}, d)[\overline{\text{false}}]$$

The baseline prediction model distributes the risk proportionally to the number of nulls involved in the attribute α_i divided by the total number of nulls in the attributes of q. This prediction model being trivial, to falsify the hypothesis, we expect our vectorial knowledge based model to outperform it.

Remark. The results of this experiment do not provide any insight concerning the relevancy of the optimal-transport based regret functions. The ground-truth values used for the validation metric are also computed with optimal-transport based regret function. Hence the relevance of the OT-based regret functions is an assumption of the validation process. It is not a hypothesis that can be falsified or confirmed with the results.

Remark. The Nullify and Nullify^{-1} processes to simulate incompleteness cleaning are not captured with the attribute-based vectorial knowledge. Indeed the constant values used to replace the nulls with the process Nullify^{-1} may not have an attribute-wide consistent behavior. Therefore the model Pessim, and Optimi are not guaranteed to be lower resp. upper bound of the evolution of the risk anymore.

Consider the following SQL query corresponding to Q16 from the TPC-H benchmark:

```
SELECT p_brand,
       p_type,
       p_size,
       Count( ps_suppkey) AS supplier_cnt
FROM   partsupp,
       part
WHERE  p_partkey = ps_partkey
       AND p_brand <> 'Brand#45'
       AND p_type NOT LIKE 'MEDIUM POLISHED%'
       AND p_size IN ( 49, 14, 23, 45,
                       19, 3, 36, 9 )
       AND NOT EXISTS (SELECT *
                       FROM   supplier
                       WHERE  s_comment LIKE '%Customer%Complaints%'
                              AND ps_suppkey = s_suppkey)
GROUP  BY p_brand,
          p_type,
          p_size
ORDER  BY supplier_cnt DESC,
          p_brand,
          p_type,
          p_size;
```

The error rate of each model for each attribute containing null is reported in figure 6.1. Those results tell us that for this specific query, on a specific database and with a specific OT-based regret function, the Linear model outperformed the baseline model. On the other hand, the Optimi and Pessim models performed poorly on the task.

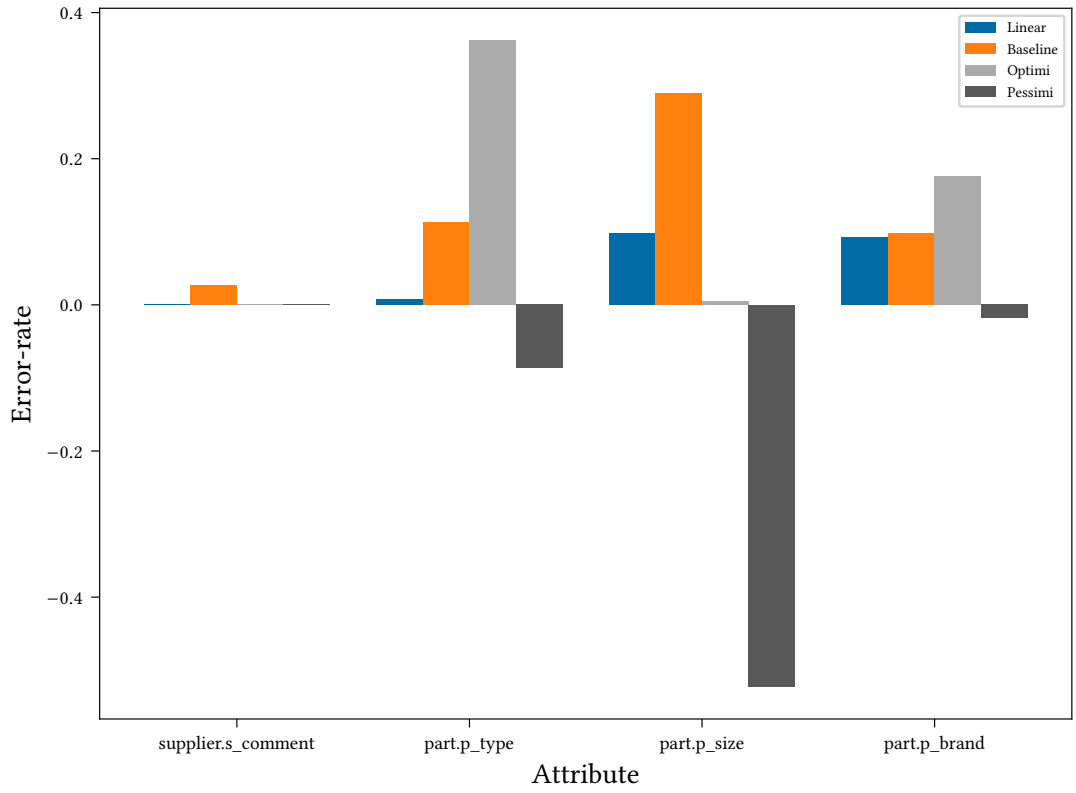


Figure 6.1: Error-rate per attribute for each prediction models on Q16

However, such reports are not sufficient to assess our hypothesis. We need to conduct the experiment on more databases, more queries, and more optimal transport-based regret functions. For the presentation to scale, we introduce average values of the error rate. For every query q, and every database d, the average error rate is:

$$\text{Error-rate}(\text{model}, q, d) = \frac{\sum\limits_{\alpha_i \in q} |\text{Error-rate}(\text{model}, q, d, \alpha_i)|}{|\{\alpha_i \in q\}|}$$

The error rate can be negative or positive; therefore, we consider the average of its absolute value. However, for every database, query, and OT-regret function we tried, the predictions of Pessim were lower than ground-truth, and the predictions of Optimi were higher. Therefore the error rate of Pessim resp. Optimi was always negative resp. positive. That is why in the reports, we are giving negative average values for the Pessim model.

For a batch of query $\mathbf{Q}$ the average of the error rate is given by:

$$\text{Error-rate}(\text{model}, \mathbf{Q}, d) = \frac{\sum\limits_{q \in \mathbf{Q}} \text{Error-rate}(\text{model}, q, d)}{|\mathbf{Q}|}$$

Finally, for a set of databases **D**, the average error rate is given by:

$$\text{Error-rate}(\text{model}, \mathbf{Q}, \mathbf{D}) = \frac{\sum_{d \in \mathbf{D}} \text{Error-rate}(\text{model}, \mathbf{Q}, d)}{|\mathbf{D}|}$$

To validate our prediction models of the attributes' importance, we ran experiments on the TPC-H benchmark. The set of TCP-H queries compatible with our framework is **Q** = {'Q1','Q2','Q3','Q4','Q5','Q6','Q11','Q16','Q19','Q21'} given in appendix A. Other queries contain operations we can not deal with yet, such as nested group-by, views, and cases, or which were too expensive to compute due to the disjunctions we are adding in the queries.

By default the parameters have the following values:

- The cost of producing and discarding a tuple is the same $\text{Cost}_{\text{Prod}} = \text{Cost}_{\text{Sink}} = 1$. It means that the penalty for missing a tuple or having an extra tuple is the same.
- The cost between a null and a constant value is $\text{Cost}_{\text{unify}} = 1$. It means that null values carries information but are still less valuable than constants. It also carries the importance of generic values in the overall regret.
- The weight of the Num component of the answers by default is $\text{Cost}_{\text{numeric}} = 1$.
- The size of the database is Size $= 100$ MB
- The null rate in the incomplete database is NullRate $= 5\%$. It means that 5% of the constant in the non Primary-Key attributes will be replace by a null.
- The probability of null values not being fixed in the database Nullify$^{-}1$ is NullRateFix $= 30\%$. It means that 30 % of the null in a attribute will still be null after the cleaning process.

The validation procedure is the following:

1. Initiate a regret function with $\text{Cost}_{\text{Prod}}$, $\text{Cost}_{\text{Sink}}$, $\text{Cost}_{\text{unify}}$ and $\text{Cost}_{\text{numeric}}$.
2. Generate a set of 3 complete databases **C** of size Size.
3. For every complete database $c \in$ **C**
   - With a probability NullRate nullify element of c to generate an incomplete database Nullify$(c) = d$.
   - For every attribute α_i generate the partially fixed databases Nullify$^{-1}(d, \alpha_i)$
   - Compute the ground-truth value of every attributes and every query in **Q**.
   - Compute the average Error-rate over the set of queries **Q** for each model: Error-rate$(\text{model}, \mathbf{Q}, d)$.
4. Compute the average Error-rate over the set of databases **D**: Error-rate$(\text{model}, \mathbf{Q}, \mathbf{D})$.

6.4.2 Results

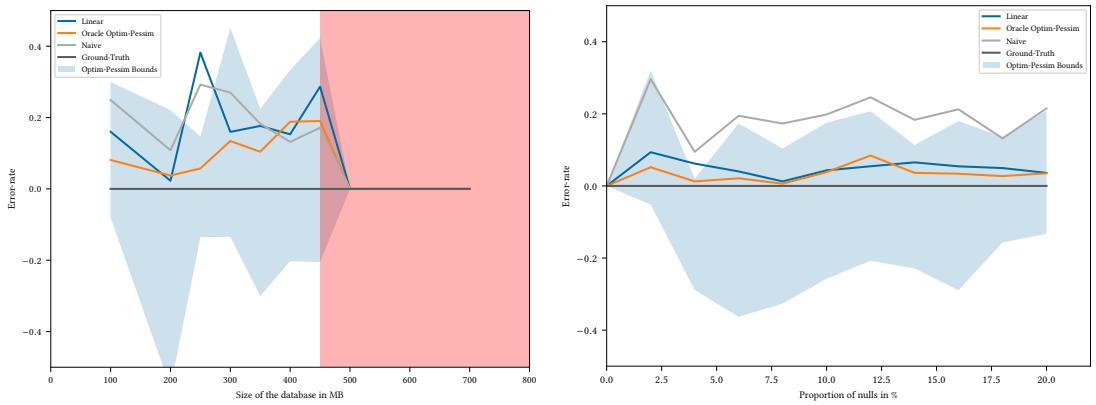
Plot The plot represents the evolution of the average Error-rate of the inference models. Each data point of the curve have been computed with the Validation process described above.

Linear This curve corresponds to the average Error-rate rate obtained with the linear heuristic inference model for the cleaning importance.

Oracle Optim-Pessim This curve corresponds to the average Error-rate rate obtained by choosing the best inference of the risk between the Optimi and Pessim for each attribute cleaning. This does not correspond to any implementable model.

Naive This curve corresponds to the average Error-rate rate obtained with the naive inference model for the cleaning importance.

Optim-Pessim Bounds This area corresponds to the values in-between the always negative Error-rate obtained with the Pessim model and the always positive Error-rate obtained with the Optimi model.



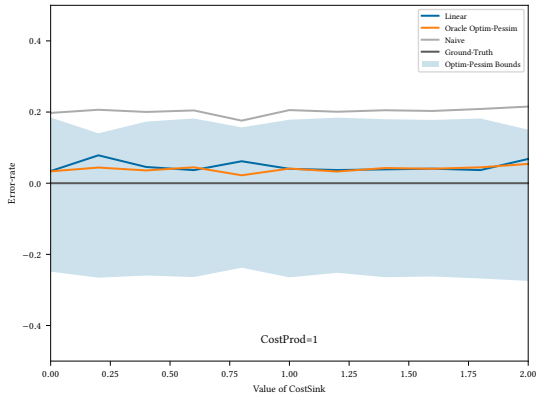
(a) Evolution of the average Error-rate with the Size of the database

(b) Evolution of the average Error-rate with the NullRate in the database

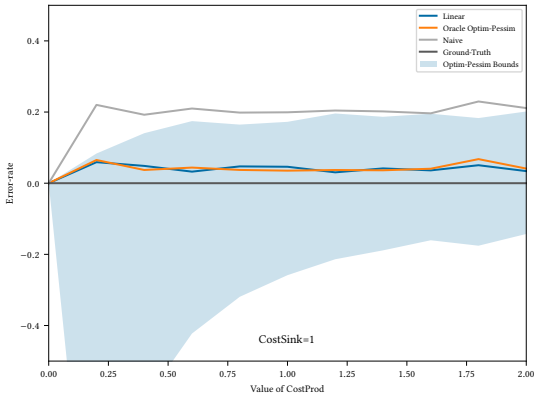
Figure 6.2: Evolution of the average Error-rate with the database parameters

Results

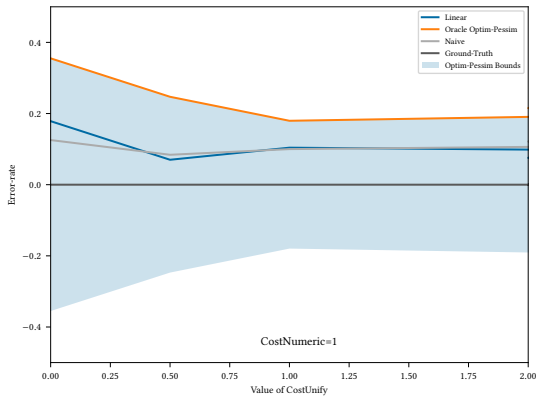
First, we discuss the problematic results. In Figure 6.2a, we remark that the evolution of the Error-rate with the Size of the databases is erratic and unpredictable. The immediate interpretation is that the models for inferring the cleaning-importance of the attributes are not resilient when the size of the database increase. However, Figure 6.2b, shows that the models are resilient to the percentage of null values in the databases. Those two results contradict each others. Indeed an erratic Error-rate is mostly due to the fact that the vectorial knowledge we are considering for inferring the



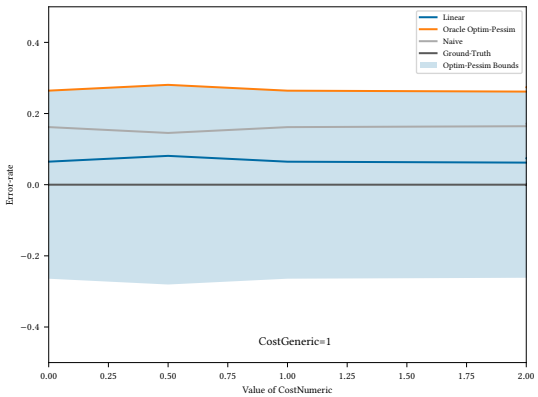
(a) Evolution of the average Error-rate with $\text{Cost}_{\text{Sink}}$



(b) Evolution of the average Error-rate with $\text{Cost}_{\text{Prod}}$



(c) Evolution of the average Error-rate with $\text{Cost}_{\text{unify}}$



(d) Evolution of the average Error-rate with $\text{Cost}_{\text{numeric}}$

Figure 6.3: Evolution of the average Error-rate with the regret function parameters

ground-truth value is attribute based, while we allow the cleaning process to be null based. Hence increasing the number of null in the databases, either with the NullRate or with the Size parameter will yield to worst Error-rate. But we do not notice any correlation between the evolution of the Error-rate with the Size and with the NullRate.

Worst when the Size of the database increases, the naive baseline model inference outperforms the linear heuristic. Our intuition is that the computation of the optimal transport based regret does not scale to big bags. The fact that all the values of risk returned for databases bigger than 500MB are 0 tend to confirm that. To be more specific with suspect that the numerical structure we are using to store the regret values is not good enough (float 32). Moreover, we notice that for small databases <200MB, the implementation is spending most of its running time on the evaluation of queries (ie. waiting for the DBMS results). While for bigger databases, the implementation is spending most of its running time on the evaluation of the optimal transport values. This is a big problem wish require more investigation.

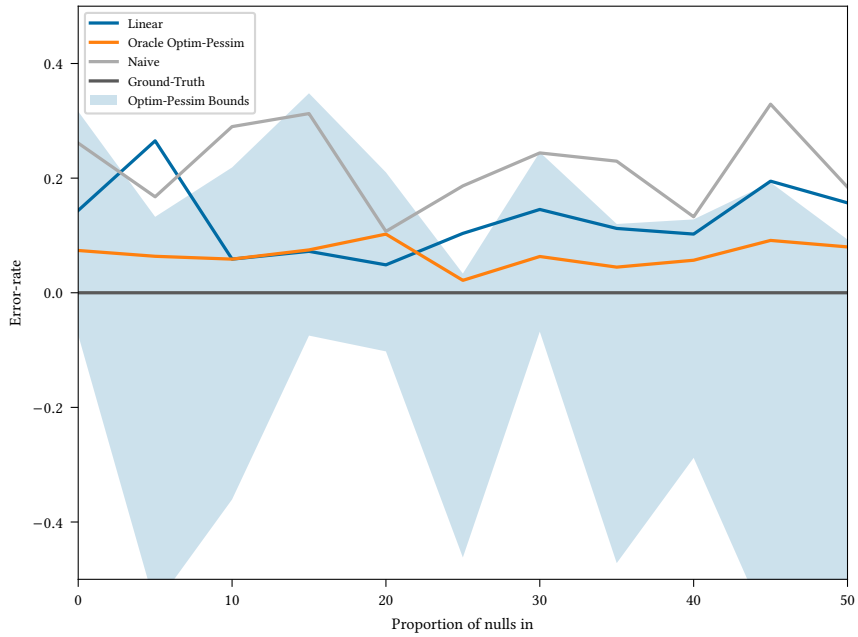


Figure 6.4: Evolution of the average Error-rate with the NullRateFix in the cleaned database

Hopefully solutions exist to help with this scaling issue. First the hardware we are using is far from optimal. Second, the problem is fully parallelizable. Finally, tractable approximation for the optimal transportation problem exists. Most notably the Sinkhorn algorithm consider an entropic regularisation of the transportation plans Cuturi (2013). It pushes the transportation plan of the tuples to be less deterministic. Overall, while we recognize the issue, we think it is more an implementation and technical problem, than a framework problem.

Second we discuss the expected results. The bounds of the cleaning importance of attributes obtain thanks to the models Optimi and Pessim are wide. As expected it gives us an upper and lower bound for the cleaning importance. But considering the difference between the Optimi and Pessim values, they seem to have little practical interest. While not perfect, the linear heuristic is resilient to the proportion of remaining null after the cleaning process (Figure 6.4). It tends to show that the attribute based vectorial knowledge is able to capture the no-information interpretation of null.

Third we discuss the promising results. Our inferences models are resilient with respect to the regret function we are considering. Indeed the values $\text{Cost}_{\text{Prod}}$ and $\text{Cost}_{\text{unify}}$ while having a big influence on the risk values, do not impact significantly the Error-rate of our models (Figure 6.3a and 6.3b). Similarly, the Error-rate are not impacted by the ratio between $\text{Cost}_{\text{numeric}}$ and $\text{Cost}_{\text{unify}}$ (Figure 6.3c and 6.3d). It shows that our models are equally good at inferring the risk of numeric attribute and generic attribute.

Finally, we wish to emphasize on the performances of the linear heuristic for inferring the cleaning importance. While outperforming the naive inference model is already good, the fact that it perform as good as the Oracle-Optim-Pessim model is impressive. Recall that the oracle can not be implemented, it guess optimally which of the optimistic or pessimistic inference of the cleaning importance it picks for each attribute. Despite this, the linear heuristic provides similar performances. It tend to show that to further increase the inference performances, new models should include more information about the attributes and the queries.

Explanations of the risk

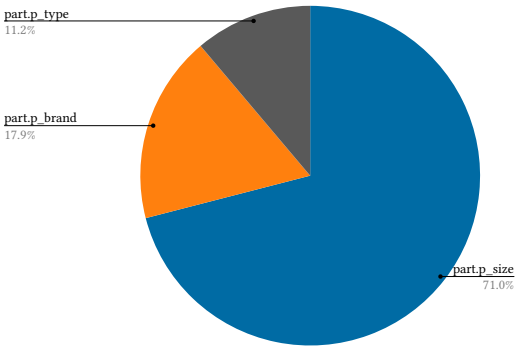
As an example, we consider the query Q16 A and we output the distribution of the Linear heuristic for cleaning-importance over the attributes (Figure 6.5a). If we were to consider this distribution as an explanation of the risk, then it seems that the nulls in the attribute s_comment have no impact on the risk.

However when we look at the distribution of the risk-contribution based on the Shapley values (Figure 6.5b), then it is clear that the attribute s_comment contributes to the the risk. If we consider the distribution of the risk-contribution divided by the number of nulls in the attributes (Figure 6.5c), the contribution of the nulls in s_comment becomes close to 20%. This simple example illustrates the relevance of considering the risk-contribution notion to provide explanations for the risk.

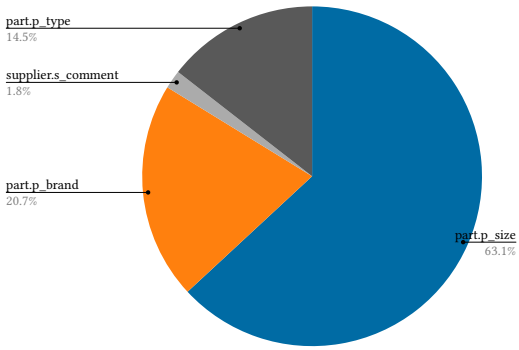
6.5 Conclusion

The SQL answers to queries on incomplete databases are not going to change in a near future. And providing a *global* explanation for them is problematic. For generic queries, SQL seems to behave consistently with the none-applicable interpretation of null. However it does not hold for queries with aggregates and groupings. Therefore we choose to tackle the problem differently and focus on providing *local* explanations for database management system answers.

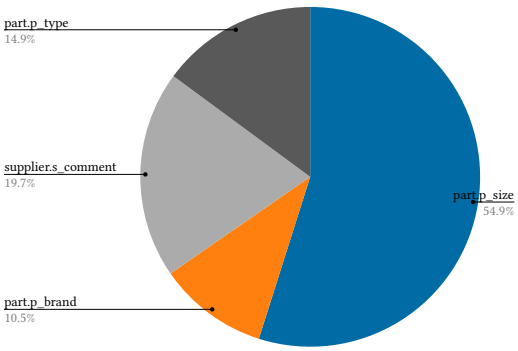
First we abstracted the sources of incompleteness, and defined them as the features of a latent representation of the additional knowledge. Then we defined the explanation of an answer as the contribution of each sources of incompleteness to its risk value. However the contribution of a source of incompleteness depends on the evolution of the risk of an answer if we happen to clean the aforementioned source. And this evolution is unknown, therefore we proposed several inference models.



(a) Distribution of the Linear cleaning-importance.



(b) Distribution of the risk-contribution.



(c) Distribution of the risk-contribution divided by the number of null

Figure 6.5: Comparison between possible explanations of the risk

To validate our inference models, we instantiate the framework on relational databases with nulls. We extended the class of optimal transport based function to work with numerical attributes. Then we implemented the framework and conducting a series of experiments to study the behavior of our inference models. While the implementation has some scaling issues, the results are promising.

Overall the experiments have been conclusive, and we can affirm that the notion of risk and sources of incompleteness can be used to provide local explanations. Our implementation should be seen as a first step toward local explanation of SQL answers on queries with aggregates and groupings. We think it is an exiting new way to tackle the problem of query-answering on incomplete databases, which may involves tools and techniques from other fields of computer science where inference is crucial.

Chapter 7

Conclusion

The handling of incomplete information is a prime example of the rift between practitioners and academics in the field of database management. On the one hand, most of the solutions proposed by the community have not translated to practice. On the other hand, most database systems lack formalism and often claim theoretically impossible features. In this thesis, we studied why such a gap exists and started to build a bridge between practical expectations and research.

At first glance, the results of the survey we conducted on the everyday usage of `NULL` seemed to explain the problem. Indeed researchers in the field have focused on interpretations of incompleteness which most users disagree with. It would explain the apparent lack of interest in the theoretical frameworks. However, the survey data also show that there is no consensus on what is a satisfactory answer to a query on an incomplete database, even for a single interpretation of `NULL` (Chapter 4). We tackled this problem with three complementary approaches.

Users' expectations

First, we can try to change users' expectations. Some users are satisfied with answers inconsistent with their own interpretation of incompleteness. I argue that the community should not compromise with the logical consistency of their frameworks. Logical consistency is the root of our work, and I think most users want consistent answers, but some users simply do not know if a result is consistent or not. We could argue that the resources are available, hence practitioners can learn about it. But I think we should also acknowledge that a lot of the research in the field does not emphasize enough on real-life models, and it tends to discourage practitioners. To help change users' expectations, we have revisited some already existing theoretical frameworks for query answering on incomplete databases. Our goal has been to explain and extend the notion of certain answers to bring it closer to real-life models and applications.

We introduce the concept of knowledge preservation to improve the information content of answers to queries on incomplete databases. Intuitively, the interpretation of incomplete data may be specified by some additional knowledge. Hence the information content of a query evaluation and a database is defined with respect to this knowledge. Consequently, the best answer one could expect is a database consistent with the interpretation of incompleteness and which preserves the additional knowledge. Thanks to this framework, we have been able to offer the first consistent notion of answers on bag relational databases with missing information (Chapter 3). It also allowed us to consider other interpretation of incompleteness, namely the no-information meaning of `NULL` (Chapter 5). Finally, the additional knowledge allowed us to handle query with arithmetic functions (Chapter 3).

Overall we have pushed further the limitations of the certain answers, bringing us a step closer to real-life applications. Moreover the concept of knowledge preservation arguably provides a straightforward intuition to understand certain answers.

User dependent answers

We can offer more flexible answer paradigms without sacrificing logical consistency. If we accept that users can have different expectations, and we still wish to provide a formal notion of answers, we have to model their preferences. In this thesis we assumed that the preferences were specified by the user thanks to a regret function representing his measurement of dissimilarity between databases.

Thanks to this framework, we have been able to offer user dependent answers, namely the risk-minimizing answers (Chapter 5). We prove that certain answers were a special case of risk-minimizing answers. Especially the notions coincide for a user who endures a maximal regret when he is neglecting an interpretation of missing information. For relational databases we proposed a class of customizable regret functions. And we showed that depending on the parameters, the risk-minimizing answers would contain more or less tuples and `NULL` elements.

To the best of our knowledge, it is the first user dependent and logically consistent framework which define answers to queries on incomplete databases.

Explaining and improving answers

The last approach is more convoluted. I argue that if there is no consensus among user toward what is a satisfactory answers. it is because all answers are equally satisfying and equally dissatisfying. The relevancy and the satisfactory value of answers to queries on incomplete databases does not depend on the answers. The most valuable component of an answer is its explanation. Why is this database the answer to my query? The additional knowledge for certain answers, and the regret function for risk-minimizing answers provides a global explanation for the answers. However the path from additional knowledge and regret function to an answer is not always easy to follow.

I think one of the most promising research venue to increase users' satisfaction is to provide easy to understand local explanation for answers. In this thesis we proposed a first concept. We explained SQL answers by outputting their risk values and the contribution of attribute involved in the query to the risk (Chapter 6). Such explanation can help a user decide if he should trust the answer, or if he should try to improve it. In order to facilitate the improvement of answers, we can infer the diminution of the risk if we clean the incompleteness of an attribute.

But our notion of explanation has some shortcomings. Even for SQL, the framework does not provide any mathematical guarantee for the contribution values. Computing the explanation is often hard, and it depends on the user regret function. Finally, we do not know what is a good explanation of an answers for practitioners. It is an exiting new way to tackle the problem of query-answering on incomplete databases, which may involves tools and techniques from other fields of computer science where inference is crucial.

7.1 Future work

In future work, there are several short-term and long-term directions that could be pursued to further improve query-answering over incomplete databases.

Exploiting and Extending risk-minimizing certain answers

While the proposed frameworks have been implemented and tested, there may be other databases and queries for which they may not perform well. Therefore, it is important to test and evaluate the proposed frameworks on a wider range of real-world databases and queries. Moreover, it is known that the complexity of the framework is intractable in most cases; therefore, studying its behavior on real-life datasets is even more important.

Another important direction is to investigate how the proposed frameworks could be extended to handle other data models besides the relational data model, such as graph databases Angles et al. (2017). Graph databases are increasingly popular in a wide range of applications, and their use of graph structures to represent data poses unique challenges for handling incomplete and uncertain information. Specifically, it is important to take into account structural uncertainty, where the incompleteness or uncertainty of one node or edge in the graph can impact the interpretation of the entire graph Fontaine and Gheerbrant (n.d.). Investigating the use of other query languages besides SQL could also be important, particularly for graph databases that use different query languages such as GQL and SPARQL Barceló Baeza (2013); Pérez, Arenas, and Gutierrez (2009). These languages may require different approaches to handle incomplete and uncertain information.

Other metrics for regret should be considered to improve the relevance of query results or to provide stronger mathematical guarantees. For example, exploring the use of uncertainty measures such as entropy, or V-usable information could be an important direction Lin (1991); Xu, Zhao, Song, Stewart, and Ermon (2020).

Finally, it may be interesting to directly incorporate user feedback in the query-answering process. For example, one could design interactive systems where users can provide feedback on query results. Investigating the use of decision theory to model user preferences more explicitly will help develop more accurate and relevant regret functions Benjamin (2019); Slovic, Fischhoff, and Lichtenstein (1977).

Beyond risk-minimizing certain answers

The proposed frameworks for risk-minimizing certain answers and user-dependent answers are relevant for both data exploration and data cleaning tasks. However, data exploration is not always done by experts in the field, and traditional query-answering techniques may not provide satisfactory explanations for non-experts. Therefore, investigating alternative methods for providing explanations of query results could be an important direction for future research. One promising avenue is the use of natural language processing techniques and large language models (LLMs), such as GPT-4 or Llama, to generate easy-to-understand natural language explanations that are tailored to the user's level of expertise and preferences OpenAI (2023); Touvron et al. (2023).

More generally, exploring the use of machine learning techniques to automatically generate explanations or develop metrics for evaluating the quality of explanations could be another interesting avenue for future research. These approaches could make query-answering more accessible and intuitive for non-experts in the field.

Finally, we aim to investigate how the proposed solutions could be applied in real-world settings, such as healthcare or finance, to handle missing data and improve the accuracy and efficiency of queries.

Appendix A

TPC-H Queries:

```
Q1:
select l_returnflag,
       l_linestatus,
       max(l_quantity)                                        as sum_qty,
       max(l_extendedprice)                                   as sum_base_price,
       max(l_extendedprice * (1 - l_discount))                as sum_disc_price,
       max(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge,
       max(l_quantity)                                        as avg_qty,
       max(l_extendedprice)                                   as avg_price,
       max(l_discount)                                        as avg_disc,
       count(*)                                               as count_order
from lineitem
where l_shipdate <= (date '1998-12-01' - interval '90' day)::date
group by l_returnflag,
         l_linestatus
order by l_returnflag,
         l_linestatus;
```

```
Q1:
select l_returnflag,
       l_linestatus,
       max(l_quantity)                                        as sum_qty,
       max(l_extendedprice)                                   as sum_base_price,
       max(l_extendedprice * (1 - l_discount))                as sum_disc_price,
       max(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge,
       max(l_quantity)                                        as avg_qty,
       max(l_extendedprice)                                   as avg_price,
       max(l_discount)                                        as avg_disc,
       count(*)                                               as count_order
from lineitem
where l_shipdate <= (date '1998-12-01' - interval '90' day)::date
group by l_returnflag,
         l_linestatus
order by l_returnflag,
```

```
        l_linestatus;

Q1:
select l_returnflag,
       l_linestatus,
       max(l_quantity)                                          as sum_qty,
       max(l_extendedprice)                                     as sum_base_price,
       max(l_extendedprice * (1 - l_discount))                  as sum_disc_price,
       max(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge,
       max(l_quantity)                                          as avg_qty,
       max(l_extendedprice)                                     as avg_price,
       max(l_discount)                                          as avg_disc,
       count(*)                                                 as count_order
from lineitem
where l_shipdate <= (date '1998-12-01' - interval '90' day)::date
group by l_returnflag,
         l_linestatus
order by l_returnflag,
         l_linestatus;

Q2:
select s_acctbal,
       s_name,
       n_name,
       p_partkey,
       p_mfgr,
       s_address,
       s_phone,
       s_comment
from part,
     supplier,
     partsupp,
     nation,
     region
where p_partkey = ps_partkey
  and s_suppkey = ps_suppkey
  and p_size = 15
  and p_type like '%BRASS'
  and s_nationkey = n_nationkey
  and n_regionkey = r_regionkey
  and r_name = 'EUROPE'
  and ps_supplycost = (
    select min(ps_supplycost)
    from partsupp,
         supplier,
```

```
        nation,
        region
    where p_partkey = ps_partkey
      and s_suppkey = ps_suppkey
      and s_nationkey = n_nationkey
      and n_regionkey = r_regionkey
      and r_name = 'EUROPE'
)
order by s_acctbal desc,
        n_name,
        s_name,
        p_partkey;
```

```
Q3:
SELECT l_orderkey,
       sum(l_extendedprice * (1 - l_discount)) as revenue,
       o_orderdate,
       o_shippriority
FROM customer,
     orders,
     lineitem
WHERE c_mktsegment = 'BUILDING'
  AND c_custkey = o_custkey
  AND l_orderkey = o_orderkey
  AND o_orderdate < date '1995-03-15'
  AND l_shipdate > date '1995-03-15'
GROUP BY l_orderkey,
         o_orderdate,
         o_shippriority
ORDER BY revenue desc,
         o_orderdate;
```

```
Q4:
SELECT o_orderpriority, COUNT(*) AS order_count
FROM orders
WHERE o_orderdate >= '1993-07-01'
  AND o_orderdate < ('1993-07-01'::date + INTERVAL '3' MONTH )::date
  AND EXISTS(
        SELECT *
        FROM lineitem
        WHERE l_orderkey = o_orderkey
          AND l_commitdate < l_receiptdate
    )
GROUP BY o_orderpriority
ORDER BY o_orderpriority;
```

Q5:

```
SELECT n_name,
       sum(l_extendedprice * (1 - l_discount)) as revenue
FROM customer,
     orders,
     lineitem,
     supplier,
     nation,
     region
WHERE c_custkey = o_custkey
  AND l_orderkey = o_orderkey
  AND l_suppkey = s_suppkey
  AND c_nationkey = s_nationkey
  AND s_nationkey = n_nationkey
  AND n_regionkey = r_regionkey
  AND r_name = 'ASIA'
  AND o_orderdate >= date '1994-01-01'
  AND o_orderdate < (date '1994-01-01' + interval '1' year)::date
GROUP BY n_name
ORDER BY revenue desc;
```

Q6:

```
SELECT sum(l_extendedprice * l_discount) as revenue
FROM lineitem
WHERE l_shipdate >= date '1994-01-01'
  AND l_shipdate < (date '1994-01-01' + interval '1' year)::date
  AND l_discount between 0.06 - 0.01 AND 0.06 + 0.01
  AND l_quantity < 24;
```

Q11:

```
SELECT ps_partkey,
       SUM(ps_supplycost * ps_availqty) AS value
FROM partsupp,
     supplier,
     nation
WHERE ps_suppkey = s_suppkey
  AND s_nationkey = n_nationkey
  AND n_name = 'GERMANY'
GROUP BY ps_partkey
HAVING SUM(ps_supplycost * ps_availqty) > (
    SELECT SUM(ps_supplycost * ps_availqty) * 0.0001000000e-2
    FROM partsupp,
         supplier,
         nation
    WHERE ps_suppkey = s_suppkey
```

```
      AND s_nationkey = n_nationkey
      AND n_name = 'GERMANY'
)
ORDER BY value DESC;
```

Q16:

```
SELECT p_brand,
       p_type,
       p_size,
       Count( ps_suppkey) AS supplier_cnt
FROM   partsupp,
       part
WHERE  p_partkey = ps_partkey
       AND p_brand <> 'Brand#45'
       AND p_type NOT LIKE 'MEDIUM POLISHED%'
       AND p_size IN ( 49, 14, 23, 45,
                       19, 3, 36, 9 )
       AND NOT EXISTS (SELECT *
                       FROM   supplier
                       WHERE  s_comment LIKE '%Customer%Complaints%'
                              AND ps_suppkey = s_suppkey)
GROUP  BY p_brand,
          p_type,
          p_size
ORDER  BY supplier_cnt DESC,
          p_brand,
          p_type,
          p_size;
```

Q19:

```
SELECT sum(l_extendedprice * (1 - l_discount)) as revenue
FROM lineitem,
     part
WHERE (
            p_partkey = l_partkey
        AND p_brand = 'Brand#12'
        AND p_container in ('SM CASE', 'SM BOX', 'SM PACK', 'SM PKG')
        AND l_quantity >= 1 AND l_quantity <= 1 + 10
        AND p_size between 1 AND 5
        AND l_shipmode in ('AIR', 'AIR REG')
        AND l_shipinstruct = 'DELIVER IN PERSON'
    )
   OR (
            p_partkey = l_partkey
        AND p_brand = 'Brand#23'
```

```
        AND p_container in ('MED BAG', 'MED BOX', 'MED PKG', 'MED PACK')
        AND l_quantity >= 10 AND l_quantity <= 10 + 10
        AND p_size between 1 AND 10
        AND l_shipmode in ('AIR', 'AIR REG')
        AND l_shipinstruct = 'DELIVER IN PERSON'
    )
   OR (
            p_partkey = l_partkey
        AND p_brand = 'Brand#34'
        AND p_container in ('LG CASE', 'LG BOX', 'LG PACK', 'LG PKG')
        AND l_quantity >= 20 AND l_quantity <= 20 + 10
        AND p_size between 1 AND 15
        AND l_shipmode in ('AIR', 'AIR REG')
        AND l_shipinstruct = 'DELIVER IN PERSON'
    );
```

```
Q21:
select s_name,
       count(*) as numwait
from supplier,
     lineitem l1,
     orders,
     nation
where s_suppkey = l1.l_suppkey
  and o_orderkey = l1.l_orderkey
  and o_orderstatus = 'F'
  and l1.l_receiptdate > l1.l_commitdate
  and exists(
        select *
        from lineitem l2
        where l2.l_orderkey = l1.l_orderkey
          and l2.l_suppkey <> l1.l_suppkey
    )
  and not exists(
        select *
        from lineitem l3
        where l3.l_orderkey = l1.l_orderkey
          and l3.l_suppkey <> l1.l_suppkey
          and l3.l_receiptdate > l3.l_commitdate
    )
  and s_nationkey = n_nationkey
  and n_name = 'FRANCE'
group by s_name
order by numwait desc,
         s_name;
```

Bibliography

Abiteboul, S. (2013). *Sciences des données: de la logique du premier ordre à la toile: Leçon inaugurale prononcée le jeudi 8 mars 2012* (Vol. 226). Fayard.

Abiteboul, S., Arenas, M., Barceló, P., Bienvenu, M., Calvanese, D., David, C., ... others (2017). Research directions for principles of data management (dagstuhl perspectives workshop 16151). *arXiv preprint arXiv:1701.09007*.

Abiteboul, S., Buneman, P., & Suciu, D. (2000). *Data on the web: from relations to semistructured data and xml*. Morgan Kaufmann.

Abiteboul, S., Hull, R., & Vianu, V. (1995). *Foundations of Databases*. Addison-Wesley.

Abiteboul, S., Kanellakis, P., & Grahne, G. (1991). On the representation and querying of sets of possible worlds. *Theoretical Computer Science*, *78*(1), 158-187.

Abiteboul, S., Segoufin, L., & Vianu, V. (2006). Representing and querying XML with incomplete information. *ACM TODS*, *31*(1), 208-254.

Amendola, G., & Libkin, L. (2018). Explainable certain answers. In *Ijcai* (pp. 1683–1690).

Angles, R., Arenas, M., Barceló, P., Hogan, A., Reutter, J., & Vrgoč, D. (2017). Foundations of modern query languages for graph databases. *ACM Computing Surveys (CSUR)*, *50*(5), 1–40.

Arenas, M., Barceló, P., Libkin, L., Martens, W., & Pieris, A. (2022). *Database theory: Querying data*. Freely available from github.com/pdm-book/community.

Arenas, M., Barceló, P., Libkin, L., & Murlak, F. (2014). *Foundations of Data Exchange*. Cambridge University Press.

Arenas, M., Botoeva, E., Kostylev, E., & Ryzhikov, V. (2017). A note on computing certain answers to queries over incomplete databases. In *Amw*.

Arieli, O., Avron, A., & Zamansky, A. (2010). Maximally paraconsistent three-valued logics. In *Principles of knowledge representation and reasoning (kr)*. AAAI Press.

Atzeni, P., & Morfuni, N. M. (1984). Functional dependencies in relations with null values. *Information Processing Letters*, *18*(4), 233-238.

Barceló Baeza, P. (2013). Querying graph databases. In *Proceedings of the 32nd acm sigmod-sigact-sigai symposium on principles of database systems* (pp. 175–188).

Benjamin, D. J. (2019). Errors in probabilistic reasoning and judgment biases. *Handbook of Behavioral Economics: Applications and Foundations 1*, *2*, 69–186.

Bhattacherjee, A. (2012). *Social science research: Principles, methods, and practices*. University of South Florida.

Bienvenu, M., & Ortiz, M. (2015). Ontology-mediated query answering with data-tractable description logics. In *Reasoning web* (pp. 218–307).

Brass, S., & Goldberg, C. (2006). Semantic errors in SQL queries: A quite complete list. *Journal of Systems and Software*, *79*(5), 630–644.

Breiman, L. (2001). Random forests. *Machine learning*, *45*(1), 5–32.

Bryman, A. (2016). *Social research methods*. Oxford university press.

Buneman, P., Jung, A., & Ohori, A. (1991). Using powerdomains to generalize relational databases. *Theoretical Computer Science*, *91*(1), 23-55.

Candan, K. S., Grant, J., & Subrahmanian, V. S. (1997). A unified treatment of null values using constraints. *Information Sciences*, *98*(1-4), 99–156.

Cannan, S., Dee, E., & Kerridge, J. (1987). *A proposal to provide support for multiple NULL states* (Tech. Rep.). ISO/TC97/SC21/WG3-DBL-AMS51.

Celko, J. (2010). *Joe celko's sql for smarties: advanced sql programming*. Elsevier.

Cesari, G., Algaba, E., Moretti, S., & Nepomuceno, J. A. (2018). An application of the shapley value to the analysis of co-expression networks. *Applied network science*, *3*(1), 1–21.

Chapel, L., Alaya, M. Z., & Gasso, G. (2020). Partial optimal tranport with applications on positive-unlabeled learning. *Advances in Neural Information Processing Systems*, *33*, 2903–2913.

Codd, E. F. (1975). Understanding relations (installment #7). *FDT - Bulletin of ACM SIGMOD*, *7*(3), 23-28.

Codd, E. F. (1979). Extending the database relational model to capture more meaning. *ACM Transactions on Database Systems*, *4*(4), 397-434.

Codd, E. F. (1986). Missing information (applicable and inapplicable) in relational databases. *SIGMOD Record*, *15*(4), 53–78.

Console, M., Guagliardo, P., & Libkin, L. (2016). Approximations and refinements of certain answers via many-valued logics. In *Kr* (pp. 349–358). AAAI Press.

Console, M., Guagliardo, P., & Libkin, L. (2017). On querying incomplete information in databases under bag semantics. In *IJCAI* (pp. 993–999). ijcai.org.

Console, M., Guagliardo, P., & Libkin, L. (2022). Propositional and predicate logics of incomplete information. *Artificial Intelligence*, *302*, 103603.

Console, M., Guagliardo, P., Libkin, L., & Toussaint, E. (2020). Coping with incomplete data: Recent advances. In *Proceedings of the 39th ACM symposium on principles of database systems, PODS 2020* (pp. 33–47). ACM.

Cuturi, M. (2013). Sinkhorn distances: Lightspeed computation of optimal transport. *Advances in neural information processing systems*, *26*.

Darwen, H., & Date, C. J. (1995). The third manifesto. *SIGMOD Record*, *24*(1), 39–49.

Date, C. J., & Darwen, H. (1996). *A guide to the SQL standard*. Addison-Wesley.

Eckerson, W. W. (2002). Data quality and the bottom line: Achieving business success through a commitment to high quality data. *The Data Warehousing Institute*, 1–36.

Evans, J. R., & Mathur, A. (2005). The value of online surveys. *Internet research*.

Evans, J. R., & Mathur, A. (2018). The value of online surveys: A look back and a look ahead. *Internet research*.

Fagin, R., Kolaitis, P. G., Miller, R. J., & Popa, L. (2005). Data exchange: semantics and query answering. *Theoretical Computer Science*, *336*(1), 89–124.

Feng, S., Huber, A., Glavic, B., & Kennedy, O. (2019). Uncertainty annotated databases - A lightweight approach for approximating certain answers. In *SIGMOD conference* (pp. 1313–1330). ACM.

Fiorentino, N., Greco, S., Molinaro, C., & Trubitsyna, I. (2018). ACID: A system for computing approximate certain query answers over incomplete databases. In *SIGMOD conference* (pp. 1685–1688). ACM.

Flamary, R., Courty, N., Rakotomamonjy, A., & Tuia, D. (2014). Optimal transport with laplacian regularization. In *Nips 2014, workshop on optimal transport and machine learning.*

Fontaine, G., & Gheerbrant, A. (n.d.). Querying incomplete graphs with data.

Franconi, E., & Tessaris, S. (2012). On the logic of SQL nulls. In *AMW* (Vol. 866, pp. 114–128). CEUR-WS.org.

Gao, Y., & Miao, X. (2018). *Query processing over incomplete databases*. Morgan & Claypool Publishers.

Gessert, G. H. (1990). Four valued logic for relational database systems. *SIGMOD Record*, *19*(1), 29–35.

Gheerbrant, A., & Libkin, L. (2015). Certain answers over incomplete XML documents: Extending tractability boundary. *Theory Comput. Syst.*, *57*(4), 892–926. Retrieved from `https://doi.org/10.1007/s00224-014-9596-y` doi: 10.1007/s00224-014-9596-y

Gheerbrant, A., Libkin, L., & Sirangelo, C. (2014). Naïve evaluation of queries over incomplete databases. *ACM Transactions on Database Systems*, *39*(4), 31:1–31:42.

Gottlob, G., & Zicari, R. V. (1988). Closed world databases opened through null values. In *Vldb* (pp. 50–61). Morgan Kaufmann.

Grahne, G. (1991). *The problem of incomplete information in relational databases* (Vol. 554). Springer.

Grant, J. (1977). Null values in a relational data base. *Information Processing Letters*, *6*(5), 156-157.

Greco, S., Molinaro, C., & Spezzano, F. (2012). *Incomplete data and data dependencies in relational databases*. Morgan & Claypool Publishers.

Guagliardo, P., & Libkin, L. (2016). Making SQL queries correct on incomplete databases: A feasibility study. In *PODS* (pp. 211–223). ACM.

Guagliardo, P., & Libkin, L. (2019). On the codd semantics of SQL nulls. *Information Systems*, *86*, 46–60.

Hartmann, S., & Link, S. (2012). The implication problem of data dependencies over SQL table definitions: Axiomatic, algorithmic and logical characterizations. *ACM Transactions on Database Systems*, *37*(2), 13:1–13:40.

Hell, P., & Nešetřil, J. (2004). *Graphs and homomorphisms*. Oxford University Press.

Hernich, A., & Kolaitis, P. G. (2017). Foundations of information integration under bag semantics. In *LICS* (pp. 1–12). IEEE Computer Society.

Hunter, A., & Konieczny, S. (2010). On the measure of conflicts: Shapley inconsistency values. *Artificial Intelligence*, *174*(14), 1007–1026.

Imielinski, T., & Lipski, W. (1984). Incomplete information in relational databases. *Journal of the ACM*, *31*(4), 761–791.

Imielinski, T., Naqvi, S. A., & Vadaparty, K. V. (1991). Incomplete objects - A data model for design and planning applications. In *Sigmod* (pp. 288–297). ACM Press.

Interim report: Ansi/x3/sparc study group on data base management systems. (1975). *FDT - Bulletin of ACM SIGMOD*, *7*(2).

Introduce named null definitions (Tech. Rep.). (1990). ISO/IEC JTC1 SC21 WG3 LON-111.

Klein, H. (2001). Null values in relational databases and sure information answers. In *Semantics in databases* (Vol. 2582, pp. 119–138). Springer.

Lerat, N., & Jr., W. L. (1986). Nonapplicable nulls. *Theoretical Computer Science*, *46*(3), 67–82.

Libkin, L. (2014). Incomplete information: what went wrong and how to fix it. In *Pods* (p. 1-13).

Libkin, L. (2016a). Certain answers as objects and knowledge. *Artificial Intelligence*, *232*, 1–19.

Libkin, L. (2016b). SQL's three-valued logic and certain answers. *ACM Transactions on Database Systems*, *41*(1), 1:1–1:28.

Libkin, L., & Wong, L. (1996). Semantic representations and query labguages for or-sets. *Journal of Computer and System Sciences*, *52*(1), 125–142.

Likert, R. (1932). A technique for the measurement of attitudes. *Archives of Psychology*, *22*(140), 5–55.

Lin, J. (1991). Divergence measures based on the shannon entropy. *IEEE Transactions on Information theory*, *37*(1), 145–151.

Lipski, W. (1979). On semantic issues connected with incomplete information databases. *ACM Transactions on Database Systems*, *4*(3), 262-296.

Lipski, W. (1984). On relational algebra with marked nulls. In *Pods* (pp. 201–203).

Livshits, E., Bertossi, L., Kimelfeld, B., & Sebag, M. (2019). The shapley value of tuples in query answering. *arXiv preprint arXiv:1904.08679*.

McCullagh, P., & Nelder, J. A. (2019). *Generalized linear models*. Routledge.

Michalak, T. P., Aadithya, K. V., Szczepanski, P. L., Ravindran, B., & Jennings, N. R. (2013). Efficient computation of the shapley value for game-theoretic network centrality. *Journal of Artificial Intelligence Research*, *46*, 607–650.

Murtagh, F. (1991). Multilayer perceptrons for classification and regression. *Neurocomputing*, *2*(5), 183-197.

Nations, U. (2007). International standard industrial classification of all economic activities (ISIC) revision 4 [Computer software manual]. New York. Retrieved from `https://unstats.un.org/unsd/classifications/Econ/isic`

Neumann, T. (2018). Reasoning in the presence of nulls. In *34th IEEE international conference on data engineering* (pp. 1682–1683). IEEE Computer Society.

OpenAI. (2023). *Gpt-4 technical report.*

Pérez, J., Arenas, M., & Gutierrez, C. (2009). Semantics and complexity of sparql. *ACM Transactions on Database Systems (TODS)*, *34*(3), 1–45.

Poggi, A., Lembo, D., Calvanese, D., De Giacomo, G., Lenzerini, M., & Rosati, R. (2008). Linking data to ontologies. *Journal on Data Semantics*, *10*, 133–173.

Reiter, R. (1977). On closed world data bases. In *Logic and data bases* (p. 55-76).

Reiter, R. (1986). A sound and sometimes complete query evaluation algorithm for relational databases with null values. *Journal of the ACM*, *33*(2), 349–347.

Revilla, M., & Ochoa, C. (2017). Ideal and maximum length for a web survey. *International Journal of Market Research*, *59*(5), 557-565.

Robertson, J. (2012). Likert-type scales, statistical methods, and effect sizes. *Communications of the ACM*, *55*(5), 6–7.

Roth, A. E. (1988). *The shapley value: essays in honor of lloyd s. shapley*. Cambridge University Press.

Sahu, S., Mhedhbi, A., Salihoglu, S., Lin, J., & Özsu, M. T. (2017). The ubiquity of large graphs and surprising challenges of graph processing. *Proceedings of the VLDB Endowment*, *11*(4), 420–431.

Sequeda, J. (2020, oct). *Understanding NULL values: a research partnership with the University of Edinburgh.* Retrieved from `https://data.world/blog/understanding-null-values-a-research-partnership-with-the-university-of-edinburgh/`

Slovic, P., Fischhoff, B., & Lichtenstein, S. (1977). Behavioral decision theory. *Annual review of psychology*, *28*(1), 1–39.

Thalheim, B., & Schewe, K. (2010). NULL 'value' algebras and logics. In *20th european-japanese conference on information modelling and knowledge bases (EJC)* (Vol. 225, pp. 354–367). IOS Press.

Toussaint, E., Guagliardo, P., & Libkin, L. (2020). Knowledge-preserving certain answers for sql-like queries. In *Kr 2020-17th international conference on principles of knowledge representation and reasoning* (pp. 758–767).

Toussaint, E., Guagliardo, P., Libkin, L., & Sequeda, J. (2022). Troubles with nulls, views from the users. *Proceedings of the VLDB Endowment*, *15*(11), 2613–2625.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., ... Lample, G. (2023). *Llama: Open and efficient foundation language models.*

Tpc benchmark™ h standard specification (Revision 2.18.0 ed.) [Computer software manual]. (2018). (`http://www.tpc.org/tpc_documents_current_versions/pdf/tpc-h_v2.18.0.pdf`)

van der Loo, M., & de Jonge, E. (2018). *Statistical data cleaning*. Wiley.

van der Meyden, R. (1998). Logical approaches to incomplete information: A survey. In *Logics for databases and information systems* (p. 307-356).

Vardi, M. (1986). Querying logical databases. *Journal of Computer and System Sciences*, *33*(2), 142–160.

Vardi, M. Y. (1986). Querying logical databases. *Journal of Computer and System Sciences*, *33*(2), 142–160.

Villani, C. (2009). *Optimal transport: old and new* (Vol. 338). Springer.

Vincent-Cuaz, C., Vayer, T., Flamary, R., Corneli, M., & Courty, N. (2021). Online graph dictionary learning. In *International conference on machine learning* (pp. 10564–10574).

Xu, Y., Zhao, S., Song, J., Stewart, R., & Ermon, S. (2020). A theory of usable information under computational constraints. *arXiv preprint arXiv:2002.10689*.

Yue, K. (1991). A more general model for handling missing information in relational databases using a 3-valued logic. *SIGMOD Record*, *20*(3), 43–49.

Zaniolo, C. (1984). Database relations with null values. *Journal of Computer and System Sciences*, *28*(1), 142–166.

Zellner, A., & Theil, H. (1992). Three-stage least squares: simultaneous estimation of simultaneous equations. In *Henri theil's contributions to economics and econometrics* (pp. 147–178). Springer.