THE UNIVERSITY of EDINBURGH

Edinburgh Research Explorer

Do We Need Many-valued Logics for Incomplete Information?

Citation for published version:
Console, M, Guagliardo, P & Libkin, L 2019, Do We Need Many-valued Logics for Incomplete Information? in *Proceedings of the Twenty-Eighth International Joint Conferences on Artificial Intelligence.* International Joint Conferences on Artificial Intelligence Organization, pp. 6141-6145, International Joint Conference in Artificial Intelligence, Macao, China, 10/08/19. https://doi.org/10.24963/ijcai.2019/851

Digital Object Identifier (DOI):
10.24963/ijcai.2019/851

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Peer reviewed version

Published In:
Proceedings of the Twenty-Eighth International Joint Conferences on Artificial Intelligence

General rights
Copyright for the publications made accessible via the Edinburgh Research Explorer is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The University of Edinburgh has made every reasonable effort to ensure that Edinburgh Research Explorer content complies with UK legislation. If you believe that the public display of this file breaches copyright please contact openaccess@ed.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 19. Aug. 2022

Do We Need Many-valued Logics for Incomplete Information?*

Marco Console , Paolo Guagliardo , Leonid Libkin
School of Informatics, University of Edinburgh

Abstract

One of the most common scenarios of handling incomplete information occurs in relational databases. They describe incomplete knowledge with three truth values, using Kleene's logic for propositional formulae and a rather peculiar extension to predicate calculus. This design by a committee from several decades ago is now part of the standard adopted by vendors of database management systems. But is it really the right way to handle incompleteness in propositional and predicate logics?

Our goal is to answer this question. Using an epistemic approach, we first characterize possible levels of partial knowledge about propositions, which leads to six truth values. We impose rationality conditions on the semantics of the connectives of the propositional logic, and prove that Kleene's logic is the maximal sublogic to which the standard optimization rules apply, thereby justifying this design choice. For extensions to predicate logic, however, we show that the additional truth values are not necessary: every many-valued extension of first-order logic over databases with incomplete information represented by null values is no more powerful than the usual two-valued logic with the standard Boolean interpretation of the connectives. We use this observation to analyze the logic underlying SQL query evaluation, and conclude that the many-valued extension for handling incompleteness does not add any expressiveness to it.

Introduction

Incomplete information is ubiquitous in applications that involve querying and reasoning about data. It is one of the oldest topics in database research [Codd, 1975], and is essential in many applications such as data integration [Lenzerini, 2002], data exchange [Arenas *et al.*, 2014], inconsistent databases [Bertossi, 2011], and ontology-based data access [Bienvenu and Ortiz, 2015].

When it comes to querying incomplete data, practical solutions, such as relational databases, rely on *many-valued logics* to properly account for the lack of certainty. In fact, every database management system (DBMS) uses a three-valued logic for handling incomplete information, namely Kleene's logic [Bolc and Borowik, 1992]. This was the design choice of SQL, the language of relational DBMSs, which is now written into the SQL Standard [ISO/IEC, 2016], presented in all database textbooks, and implemented in all database products. However, this is far from the only logic to have been considered for representing incomplete information. The use of Kleene's logic was first proposed by Codd [1975], but many other variants appeared afterward. Codd [1987] looked at a four-valued logic, but in the end argued against it due to the additional complexity. Nonetheless, well-documented problems with incomplete information [Date and Darwen, 1996; Date, 2005] led to the search of more appropriate logics for handling incompleteness. For example, Gessert [1990] revisited four-valued logics, while Yue [1991] considered logics with four, five, and seven values, and showed how to encode them with three. A different kind of four-valued logics for missing data was studied by Console *et al.* [2016], while Darwen and Date [1995] suggested dropping nulls altogether and go back to the usual Boolean two-valued logic.

There is also no shortage of many-valued logics that have been proposed in closely related contexts. For example, a variety of many-valued logics were used in the study of default reasoning [Reiter, 1980] or in reasoning about inconsistency [Zamansky and Avron, 2006]. Those are typically based on the notion of bilattices, providing truth and knowledge orderings on the truth values [Arieli and Avron, 1996; Ginsberg, 1988]. A common one is Belnap's bilattice with four truth values [Belnap, 1977; Arieli and Avron, 1998], which also found database applications [Grahne *et al.*, 2015]; but others exist as well, e.g., many generalizations of Kleene's logic based on numerical intervals describing the degree of being true [Fitting, 1991]. A many-valued propositional logic must also provide an interpretation of propositional connectives. To make the general picture even muddier, for different sets of truth values, different semantics of propositional connectives exist, sometimes even non-deterministic ones [Arieli *et al.*, 2010].

*This paper was invited for submission to the *Best Papers from Sister Conferences* track. It is an abridged version of a paper that appeared in KR-2018 [Console *et al.*, 2018].

Thus, we are far from having a clear picture of what to use as a logic of incomplete information in data management applications. Choices are numerous, and there is no final argument as to why the approach of DBMSs that use Kleene's logic is the right one. Hence, the first question we address is:

1) *What is the right many-valued propositional logic for handling incomplete information?*

Now suppose we have a propositional logic that correctly accounts for truth values of statements about incomplete information, and for operations on them. In querying data, however, we use *predicate* logics. Indeed, the core of SQL is essentially a programming syntax for relational calculus, which is another name for first-order (FO) predicate logic.

Of course we know how to lift the semantics of propositional logic to the full predicate calculus by treating existential and universal quantifiers as disjunctions and conjunctions over all elements of the universe. What we do not know is how different choices of propositional logic for incomplete information affect the power of predicate calculus. As one example, consider the version of FO that underlies SQL and is based on Kleene's logic. What extra power does it possess over FO under the usual two-valued Boolean interpretation of the connectives? It was recently argued, by means of rewriting SQL queries, that FO based on Kleene's logic can be encoded in the usual Boolean FO [Guagliardo and Libkin, 2017]. But is there a general result in logic that underlies such a translation, and what is so special about Kleene's logic that makes it work?

Even more generally, the second question we would like to address is:

2) *How does the choice of a propositional logic for incomplete information affect predicate logic?*

Finally, we would like to understand how these theoretical considerations relate to the practice of incomplete data in relational databases. A rough approximation of the core of SQL – the way it is presented in many database textbooks – is first-order logic. But as soon as incomplete information enters the picture, this becomes a many-valued FO. And yet there is even more to it: in SQL queries, answer tuples are split into *true* ones that need to be returned, and others that are not returned, thus collapsing a three-valued logic to two-valued. This leads to our last question:

3) *What is the logic that underlies real-life handling of incomplete information in relational databases (i.e., SQL's logic), and how much more power than the usual two-valued* FO *does it possess?*

The goal of this paper is to address these three questions. Below we outline our main contributions.

Propositional Logic

To understand what a proper propositional logic for reasoning about incomplete information is, we need to define its truth values, and truth tables for its connectives (we shall concentrate on the standard ones, i.e., $\wedge, \vee$, and $\neg$, although we shall see others as well). We follow the approach of Ginsberg [1988] to turn partial knowledge about the truth of a proposition into truth values. If we have a set W of worlds, and two of its subsets T and F in which a proposition is true and false, respectively, this produces a description (T, F, W). It is possible that $T \cup F \neq W$, i.e., we may have partial knowledge about the truth or falsity of a proposition. We require however that $T \cap F = \varnothing$, as here we do not consider inconsistent descriptions.

Taking those descriptions (T, F, W) directly as truth values, however, is not satisfactory: we shall have too many of them. Instead, we want to take as truth values *what we know* about such descriptions. To this end, we use *epistemic theories* that say what is known about a proposition α being possibly or certainly true or false; then, as truth values we take maximally consistent epistemic theories. We show that there are only six such theories, resulting in a six-valued logic, denoted by $\mathbb{L}_{6\text{v}}$. For an arbitrary proposition α, these six theories correspond to the following scenarios.

(1) We know that α is *true in all worlds*. We abstract this as the truth value **t** (*always true*).

(2) We know that $\neg\alpha$ is true in all worlds, hence α is *false in all worlds*. We abstract this as the truth value **f** (*always false*).

(3) We know that there exists a world w in which α is true and there exists a world w' in which its negation is true. Since α cannot be both true and false in the same world, we have $w \neq w'$. We abstract this as the truth value **s** (*sometimes true and sometimes false*).

(4) We know that there is a world in which α is true, but we do not know whether there is a (distinct) world in which its negation is true. Thus, α could be true in all worlds, but we do not know that. We abstract this as the truth value **st** (*sometimes true*).

(5) We know that there is a world in which the negation of α is true and where α is then false, but we do not know whether there is a (distinct) world in which α is true. Thus, α could be false in all worlds, but we do not know that. We abstract this as the truth value **sf** (*sometimes false*).

(6) We do not know whether there exists a world in which α is true nor whether there is one where its negation is true. That is, we have no information at all, and we abstract this as the truth value **u** (*unknown*).

The truth tables of $\mathbb{L}_{6\text{v}}$, shown in Figure 1, are again very naturally derived from epistemic theories of partial knowledge about the truth of propositions. The logic $\mathbb{L}_{6\text{v}}$ contains, as sublogics, the following: (1) the 2-valued Boolean logic, which is the restriction of $\mathbb{L}_{6\text{v}}$ to truth values **t** and **f**, and (2) Kleene's logic, denoted by $\mathbb{L}_{3\text{v}}$, which is the restriction to truth values **t**, **f**, **u**.

As a final step, we look at what makes a many-valued logic database friendly. It needs to be a sublogic of $\mathbb{L}_{6\text{v}}$ and satisfy some basic properties we expect to hold to be able to perform query evaluation and optimization, namely *distributivity* and *idempotency* (see [Jarke and Koch, 1984; Graefe, 1993]).

Theorem 1. *The maximal sublogic of* $\mathbb{L}_{6\text{v}}$ *that includes the truth value* **t** *and in which the binary connectives* $\wedge$ *and* $\vee$ *are distributive and idempotent is Kleene's logic* $\mathbb{L}_{3\text{v}}$.

$\wedge$	**t**	**f**	**s**	**st**	**sf**	**u**
t	**t**	**f**	**s**	**st**	**sf**	**u**
f	**f**	**f**	**f**	**f**	**f**	**f**
s	**s**	**f**	**sf**	**sf**	**sf**	**sf**
st	**st**	**f**	**sf**	**u**	**sf**	**u**
sf	**sf**	**f**	**sf**	**sf**	**sf**	**sf**
u	**u**	**f**	**sf**	**u**	**sf**	**u**

(a)

$\vee$	**t**	**f**	**s**	**st**	**sf**	**u**
t	**t**	**t**	**t**	**t**	**t**	**t**
f	**t**	**f**	**s**	**st**	**sf**	**u**
s	**t**	**s**	**st**	**st**	**st**	**st**
st	**t**	**st**	**st**	**st**	**st**	**st**
sf	**t**	**sf**	**st**	**st**	**u**	**u**
u	**t**	**u**	**st**	**st**	**u**	**u**

(b)

	$\neg$
t	**f**
f	**t**
s	**s**
st	**sf**
sf	**st**
u	**u**

(c)

Figure 1: The truth tables of $\mathbb{L}_{6v}$ for $\wedge$, $\vee$ and $\neg$.

This result justifies, at least at the propositional level, the choice made by SQL designers and standardization committees in choosing the three-valued logic of Kleene as the logic to be implemented in all database products.

Predicate Logic

We have justified Kleene's propositional logic $\mathbb{L}_{3v}$ as the right choice for handling incompleteness in database contexts. But database languages are not propositional: they are based on FO instead. Thus, we look at variants of FO based on propositional many-valued logics such as $\mathbb{L}_{3v}$ and $\mathbb{L}_{6v}$, and compare their expressive power with that of the usual Boolean FO (BFO) with just **t** and **f**. Our main result is that, when added to FO, these many-valued propositional logics add no power: FO based on $\mathbb{L}_{3v}$, or on $\mathbb{L}_{6v}$, or on any other many-valued logic (under some mild restrictions on the connectives) has no more power than BFO.

The need to consider predicate logics of incomplete information arises most commonly in querying incomplete databases, where special values, referred to as *nulls*, indicate incompleteness of some sort. When atomic formulae may involve nulls – e.g., in comparing a null with another value, or in checking whether a tuple with nulls belongs to a relation – the standard approach is not to follow the Boolean semantics of FO, but instead to look for a many-valued semantics that will properly lift a propositional logic to all of FO. While propagating truth values through connectives and quantifiers is completely standard, assigning them to atoms is not unique. We consider three commonly occurring possibilities.

Boolean semantics. This is the standard two-valued FO semantics, with only **t** and **f** as truth values. Given a database D and a tuple $\bar{a}$, the formula $R(\bar{a})$ evaluates to **t** if $\bar{a}$ is tuple in relation R in D, and to **f** otherwise.

Null-free semantics. If $\bar{a}$ is a tuple of constants, it is the same as the Boolean semantics; but if $\bar{a}$ contains nulls, the formula evaluates to **u**.

Unification semantics. This was proposed by Libkin [2016] to enforce certainty guarantees for query answers. It is based on the notion of tuple unification: two tuples *unify* if there exists a mapping from nulls to constants that makes them equal. Then, given a database D, an atom $R(\bar{a})$ evaluates to **t** if $\bar{a}$ is in relation R in D, to **f** if no tuple in R unifies with $\bar{a}$, and to **u** otherwise.

In all of these, instead of relational atoms we could use equality atoms $a = b$ as well, and the same definitions apply.

There is a priori no reason to apply the same semantics to all atoms; instead, one can consider a *mixed semantics* where each atomic formula is interpreted under one of the different "pure" semantics described above. Indeed, this is the case for SQL: relational atoms are interpreted under the Boolean semantics, while equality atoms are interpreted under the null-free semantics. We refer to the mixed semantics used by SQL as the *SQL semantics*.

In the context of many-valued FO, the exact choice of semantics of atoms does not matter: under reasonable assumptions, many-valued predicate logics do not give any extra expressive power compared to BFO, that is, the usual FO under the standard Boolean interpretation of connectives and the Boolean semantics of atomic formulae.

The result is based on capturing a many-valued FO logic by BFO, in the following sense. A formula φ of FO based on a many-valued propositional logic $\mathbb{L}$ is *captured* by BFO under a semantics $[\![\,]\!]$ if for each truth value τ of $\mathbb{L}$ there exists an FO formula φ_τ such that φ evaluates to τ under $[\![\,]\!]$ if and only if φ_τ evaluates to true under the Boolean semantics. Intuitively, if a formula is captured by BFO, this tells us that we do not need a many-valued semantics.

To lift the capturing from atoms to arbitrary formulae of a many-valued FO we only need to impose one condition on the underlying propositional logic, namely that the connectives $\wedge$ and $\vee$ be *weakly idempotent*: for every truth value τ, it must hold that $\tau \wedge \tau \wedge \tau = \tau \wedge \tau$, and likewise for $\vee$. In Boolean logic, we expect these connectives to be idempotent (i.e., $\tau \wedge \tau = \tau$) but logics like $\mathbb{L}_{6v}$ satisfy this weaker condition.

Theorem 2. *Let* $\mathrm{FO}(\mathbb{L})$ *be* FO *based on a many-valued propositional logic* $\mathbb{L}$ *in which the connectives* $\wedge$ *and* $\vee$ *are weakly idempotent, and assume that every atomic* $\mathrm{FO}(\mathbb{L})$ *formula is captured by* BFO *under some (mixed) semantics. Then every* $\mathrm{FO}(\mathbb{L})$ *formula is captured by* BFO *under the same semantics.*

To apply this result to the previously considered semantics, we need to capture atomic formulae, under these semantics, in BFO. This is possible for all of them.

Proposition 1. *Atomic formulae are captured by* BFO *under Boolean, unification, and null-free semantics.*

Finally, this tells us that any mixed semantics (as well as the pure versions, that is, Boolean, null-free, and unification semantics) coupled with any propositional many-valued logic like $\mathbb{L}_{3v}$ or $\mathbb{L}_{6v}$ (as long as it has weakly idempotent conjunction and disjunction) is no more powerful than the standard Boolean semantics over **t** and **f**.

Using this result, we can clarify the question about the expressive power of the logic that underlies real-life database applications that use incomplete information.

The Logic of SQL

We finally apply the above observation to SQL's logic. We explain that it corresponds to a $\mathbb{L}_{3v}$-based FO with an extra connectives that allows one to collapse truth values **f** and **u** into one, but it still has no more power than BFO. Thus, even though SQL designers were justified in choosing Kleene's logic as the propositional logic for reasoning about incomplete information, they overlooked the fact that, when considered within FO, such a logic does not add any expressive power.

Most database texts will claim that the core of SQL is first-order logic FO. This was certainly true in the early stages of SQL design, as it grew out of relational calculus, which is just another name for FO. But then the language gained many features, in particular null values, leading to more complex underlying logics. These logics are still not well understood, as the formalization of SQL mainly took a different route via *relational algebra*, which is the procedural counterpart of FO.

We start with the basic fragment of relational languages that has the power of FO, operating on databases whose values come from a set Const of constants. SQL uses a single null value, denoted here by $\mathbf{N}$. Now we add it; how should the logic change to capture this extension? It depends on who is asked to produce such an extension.

A logician's approach. If the domain is extended by a single constant, we simply consider FO over $\mathsf{Const} \cup \{\mathbf{N}\}$ with a unary predicate $\mathsf{null}(\,)$ that is only true in $\mathbf{N}$. The interpretation of $=$ is simply *syntactic equality*, that is, $\mathbf{N}$ is equal to itself, and not equal to any element of Const. In other words, the logic is the usual BFO, with atomic formulae interpreted under the Boolean semantics. It would thus be seen, by a logician, as an overkill to introduce a many-valued logic to deal with just one extra element of the domain. Nonetheless, this is what SQL did.

SQL approach: a textbook version. The usual explanation of the logic behind SQL is that it adds a new truth value **u** to account for any comparisons involving nulls. In other words, the underlying propositional logic is Kleene's, and the semantics is mixed: Boolean for relational atoms and null-free for equality. We refer to this mixed semantics as the *SQL semantics*.

SQL approach: what really happens. While the textbook approach comes close to describing the logic of SQL, it overlooks one important aspect: in an SQL query, the conditions in the `WHERE` clause are evaluated using Kleene's logic, but only the tuples that evaluated to **t** are kept. To capture this in logic, we need a propositional operator that collapses **f** and **u** into **f**. Such an operator does exist in propositional many-valued logics [Bochvar, 1981] and it is known as an *assertion* operator: $\uparrow p$ for a proposition p evaluates to **t** if p evaluates to **t**, and to **f** otherwise. Kleene's logic with the assertion operator is denoted by $\mathbb{L}_{3v}^{\uparrow}$.

Even though these logics use different sets of truth values, it only matters when formulae evaluate to true, as this determines the output of queries. To this end, we say that two logics are *true-equivalent* if there are translations between their formulae such that each formula, and its translation, evaluate to **t** in the same models. Then, with respect to the truth value **t**, there is no difference between the logics that attempt to model SQL's behavior.

Theorem 3. *The following predicate logics:*

- BFO *under the Boolean semantics,*
- $\mathrm{FO}(\mathbb{L}_{3v})$ *under the SQL semantics, and*
- $\mathrm{FO}(\mathbb{L}_{3v}^{\uparrow})$ *under the SQL semantics,*

are true-equivalent.

Thus, the most natural logical approach to adding the null value to the language does not miss any expressiveness of the more complex solutions based on many-valued logics.

Conclusions

To conclude, let us revisit history. Handling incomplete information by logical languages is an important topic, especially in data management. All commercial database systems that speak SQL offer a solution based on a three-valued propositional logic that is lifted then to full predicate logic. This solution was heavily criticized in the literature, but at the level of the chosen propositional logic.

We proposed a principled approach to justifying a proper logic for handling incomplete information, which resulted in a six-valued logic $\mathbb{L}_{6v}$. However, taking into account the needs of SQL query evaluation (e.g., distributivity laws), the largest fragment of $\mathbb{L}_{6v}$ that does not break traditional evaluation and optimization strategies is Kleene's logic $\mathbb{L}_{3v}$, precisely the one chosen by SQL.

However, even though the SQL designers were justified in their choice of Kleene's logic, they neglected to consider the impact that lifting it to full predicate logic would have. We showed that it leads to no increase in expressive power; had this been known to the SQL designers, perhaps other choices would have been considered too.

But does this mean that we should abandon many-valued logics of incomplete information? Most likely not: while the theoretical complexity of formulae that result from eliminating many-valuedness is the same as that of original many-valued formulae, their *practical* complexity (i.e., if implemented as real life database queries) is likely to be different. This is mainly due to the fact that 40 years of research on query evaluation and optimization had one particular model in mind, and that model used a many-valued logic. However, the observations we made here might have an impact on the design of new languages, since avoiding many-valued logics for handling incompleteness is now an option.

Acknowledgments

This work was partly supported by EPSRC grants M025268 and N023056.

References

[Arenas *et al.*, 2014] Marcelo Arenas, Pablo Barceló, Leonid Libkin, and Filip Murlak. *Foundations of Data Exchange*. Cambridge University Press, 2014.

[Arieli and Avron, 1996] Ofer Arieli and Arnon Avron. Reasoning with logical bilattices. *Journal of Logic, Language and Information*, 5(1):25–63, 1996.

[Arieli and Avron, 1998] Ofer Arieli and Arnon Avron. The logical role of the four-valued bilattice. In *LICS*, pages 118–126, 1998.

[Arieli *et al.*, 2010] Ofer Arieli, Arnon Avron, and Anna Zamansky. Maximally paraconsistent three-valued logics. In *KR*, 2010.

[Belnap, 1977] N. D. Belnap. A useful four-valued logic. In J. M. Dunn and G. Epstein, editors, *Modern Uses of Multiple-Valued Logic*, pages 8–37. D. Reidel, 1977.

[Bertossi, 2011] Leopoldo Bertossi. *Database Repairing and Consistent Query Answering*. Morgan&Claypool Publishers, 2011.

[Bienvenu and Ortiz, 2015] Meghyn Bienvenu and Magdalena Ortiz. Ontology-mediated query answering with data-tractable description logics. In *Reasoning Web*, pages 218–307, 2015.

[Bochvar, 1981] D. A. Bochvar. On a three-valued logical calculus and its application to the analysis of the paradoxes of the classical extended functional calculus. *History and Philosophy of Logic*, 2:87–112, 1981. Translated from *Matematicheskij Sbornik*, 4 (46): 287-–308, 1938.

[Bolc and Borowik, 1992] L. Bolc and P. Borowik. *Many-Valued Logics: Theoretical Foundations*. Springer, 1992.

[Codd, 1975] E. F. Codd. Understanding relations (installment #7). *FDT - Bulletin of ACM SIGMOD*, 7(3):23–28, 1975.

[Codd, 1987] E. F. Codd. More commentary on missing information in relational databases. *SIGMOD Record*, 16(1):42–50, 1987.

[Console *et al.*, 2016] Marco Console, Paolo Guagliardo, and Leonid Libkin. Approximations and refinements of certain answers via many-valued logics. In *KR*, pages 349–358, 2016.

[Console *et al.*, 2018] Marco Console, Paolo Guagliardo, and Leonid Libkin. Propositional and predicate logics of incomplete information. In *KR*, pages 592–601. AAAI Press, 2018.

[Darwen and Date, 1995] Hugh Darwen and C. J. Date. The third manifesto. *SIGMOD Record*, 24(1):39–49, 1995.

[Date and Darwen, 1996] C. J. Date and H. Darwen. *A Guide to the SQL Standard*. Addison-Wesley, 1996.

[Date, 2005] Chris J. Date. *Database in Depth - Relational Theory for Practitioners*. O'Reilly, 2005.

[Fitting, 1991] Melvin Fitting. Kleene's logic, generalized. *J. Log. Comput.*, 1(6):797–810, 1991.

[Gessert, 1990] G. H. Gessert. Four valued logic for relational database systems. *SIGMOD Record*, 19(1):29–35, 1990.

[Ginsberg, 1988] Matthew L. Ginsberg. Multivalued logics: a uniform approach to reasoning in artificial intelligence. *Computational Intelligence*, 4:265–316, 1988.

[Graefe, 1993] Goetz Graefe. Query evaluation techniques for large databases. *ACM Comput. Surv.*, 25(2):73–170, 1993.

[Grahne *et al.*, 2015] Gösta Grahne, Ali Moallemi, and Adrian Onet. Intuitionistic data exchange. In *9th Alberto Mendelzon International Workshop on Foundations of Data Management*, 2015.

[Guagliardo and Libkin, 2017] Paolo Guagliardo and Leonid Libkin. A formal semantics of SQL queries, its validation, and applications. *PVLDB*, 11(1):27–39, 2017.

[ISO/IEC, 2016] ISO/IEC. *ISO/IEC 9075:2016: Information technology – Database languages – SQL*. International Organization for Standardization, 2016.

[Jarke and Koch, 1984] Matthias Jarke and Jürgen Koch. Query optimization in database systems. *ACM Comput. Surv.*, 16(2):111–152, 1984.

[Lenzerini, 2002] Maurizio Lenzerini. Data integration: a theoretical perspective. In *ACM Symposium on Principles of Database Systems (PODS)*, pages 233–246, 2002.

[Libkin, 2016] Leonid Libkin. SQL's three-valued logic and certain answers. *ACM Trans. Database Syst.*, 41(1):1:1–1:28, 2016.

[Reiter, 1980] Raymond Reiter. A logic for default reasoning. *Artif. Intell.*, 13(1-2):81–132, 1980.

[Yue, 1991] Kwok-bun Yue. A more general model for handling missing information in relational databases using a 3-valued logic. *SIGMOD Record*, 20(3):43–49, 1991.

[Zamansky and Avron, 2006] Anna Zamansky and Arnon Avron. Non-deterministic semantics for first-order paraconsistent logics. In *KR*, pages 431–439, 2006.