Databases under the Partial Closed-world Assumption: A Survey

Simon Razniewski
Free University of Bozen-Bolzano
Dominikanerplatz 3
39100 Bozen, Italy
razniewski@inf.unibz.it

Werner Nutt
Free University of Bozen-Bolzano
Dominikanerplatz 3
39100 Bozen, Italy
nutt@inf.unibz.it

ABSTRACT

Databases are traditionally considered either under the closed-world or the open-world assumption. In some scenarios however a middle ground, the partial closed-world assumption, is needed, which has received less attention so far.

In this survey we review foundational and work on the partial closed-world assumption and then discuss work done in our group in recent years on various aspects of reasoning over databases under this assumption.

We first discuss the conceptual foundations of this assumption. We then list the main decision problems and the known results. Finally, we discuss implementational approaches and extensions.

1. INTRODUCTION

Data completeness is an important aspect of data quality. Traditionally, it is assumed that a database reflects exactly the state of affairs in an application domain, that is, a fact that is true in the real world is stored in the database, and a fact that is missing in the database does not hold in the real world. This is known as the closed-world assumption (CWA). Later approaches have discussed the meaning of databases that are missing facts that hold in the real world and thus are incomplete. This is called the open-world assumption (OWA) [16, 7].

A middle view, which we call the partial closed-world assumption (PCWA), has received less attention until recently. Under the PCWA, some parts of the database are assumed to be closed (complete), while others are assumed to be open (possibly incomplete). So far, the former parts were specified using completeness statements, while the latter parts are the complement of the complete parts.

Example. As an example, consider a problem arising in the management of school data in the province of Bolzano, Italy, which motivated the technical work reported here. The IT department of the provincial school administration runs a database for storing school data, which is maintained in a decentralized manner, as each school is responsible for its own data. Since there are numerous schools in this province, the overall database is notoriously incomplete. However, periodically the statistics department of the province queries the school database to generate statistical reports. These statistics are the basis for administrative decisions such as the opening and closing of classes, the assignment of teachers to schools and others. It is therefore important that these statistics are correct. Therefore, the IT department is interested in finding out which data has to be complete in order to guarantee correctness of the statistics, and on which basis the guarantees can be given.

Broadly, we investigated the following research questions:

1. How to describe complete parts of a database?
2. How to find out, whether a query answer over a partially closed database is complete?
3. If a query answer is not complete, how to find out which kind of data can be missing, and which similar queries are complete?

Work Overview. The first work on the PCWA is from Motro [10]. He used queries to describe complete parts and introduced the problem of inferring the completeness of other queries (QC) from such completeness statements. Later work by Halevy [8] introduced tuple-generating dependencies or table completeness (TC) statements for specification of complete parts. A detailed complexity study of TC-QC entailment was done by Razniewski and Nutt [13].

Later work by Razniewski and Nutt has focussed on databases with null values [12] and geographic databases [14].

There has also been work on RDF data [3]. Savkovic et al. [18, 17] have focussed on implementation techniques, leveraging especially on logic programming.

Also the derivation of completeness from data-aware business process descriptions has been discussed [15].

Current work is focussing on reasoning wrt. database instances and on queries with negation [4].

Outline. This paper is structured as follows. In Section 2, we discuss conceptual foundations, in particular the partial closed-world assumption. In Section 3 we present main reasoning problems in this framework and known results. Section 4 discusses implementation techniques. Section 5 presents extension and Section 6 discusses current work and open problems.

In: G. Specht, H. Gamper, F. Klan (eds.): Proceedings of the 26th GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken), 21.10.2014 - 24.10.2014, Bozen, Italy, published at http://ceur-ws.org.

2. CONCEPTUAL FOUNDATIONS

2.1 Standard Definitions

In the following, we fix our notation for standard concepts from database theory. We assume a set of relation symbols Σ, the *signature*. A *database instance* D is a finite set of ground atoms with relation symbols from Σ. For a relation symbol $R \in \Sigma$ we write $R(D)$ to denote the interpretation of R in D, that is, the set of atoms in D with relation symbol R. A *condition* G is a set of atoms using relations from Σ and possibly the comparison predicates $<$ and $\leq$. As common, we write a condition as a sequence of atoms, separated by commas. A condition is *safe* if each of its variables occurs in a relational atom. A *conjunctive query* is written in the form $Q(\bar{s}) :- B$, where B is a safe condition, $\bar{s}$ is a vector of terms, and every variable in $\bar{s}$ occurs in B. We often refer to the entire query by the symbol Q. As usual, we call $Q(\bar{s})$ the *head*, B the *body*, the variables in $\bar{s}$ the *distinguished variables*, and the remaining variables in B the *nondistinguished variables* of Q. We generically use the symbol L for the subcondition of B containing the relational atoms and M for the subcondition containing the comparisons. If B contains no comparisons, then Q is a *relational conjunctive query*.

The result of evaluating Q over a database instance D is denoted as $Q(D)$. Containment and equivalence of queries are defined as usual. A conjunctive query is *minimal* if no relational atom can be removed from its body without leading to a non-equivalent query.

2.2 Running Example

For our examples throughout the paper, we will use a drastically simplified extract taken from the schema of the Bolzano school database, containing the following two tables:

- *student(name, level, code),*
- *person(name, gender).*

The table *student* contains records about students, that is, their names and the level and code of the class we are in. The table *person* contains records about persons (students, teachers, etc.), that is, their names and genders.

2.3 Completeness

Open and closed world semantics were first discussed by Reiter in [16], where he formalized earlier work on negation as failure [2] from a database point of view. The closed-world assumption corresponds to the assumption that the whole database is complete, while the open-world assumption corresponds to the assumption that nothing is known about the completeness of the database.

Partial Database. The first and very basic concept is that of a partially complete database or partial database [10]. A database can only be incomplete with respect to another database that is considered to be complete. So we model a partial database as a pair of database instances: one instance that describes the complete state, and another instance that describes the actual, possibly incomplete state. Formally, a *partial database* is a pair $\mathcal{D} = (D^i, D^a)$ of two database instances D^i and D^a such that $D^a \subseteq D^i$. In the style of [8], we call D^i the *ideal* database, and D^a the *available* database. The requirement that D^a is included in D^i formalizes the intuition that the available database contains no more information than the ideal one.

Example 1. *Consider a partial database $\mathcal{D}_S$ for a school with two students, Hans and Maria, and one teacher, Carlo, as follows:*

$$\begin{aligned} D^i_S &= \{student(Hans, 3, A), student(Maria, 5, C), \\ &\quad person(Hans, male), person(Maria, female), \\ &\quad person(Carlo, male)\}, \\ D^a_S &= D^i_S \setminus \{person(Carlo, male), student(Maria, 5, C)\}, \end{aligned}$$

that is, the available database misses the facts that Maria is a student and that Carlo is a person. □

Next, we define statements to express that parts of the information in D^a are complete with regard to the ideal database D^i. We distinguish query completeness and table completeness statements.

Query Completeness. For a query Q, the *query completeness* statement $Compl(Q)$ says that Q can be answered completely over the available database. Formally, $Compl(Q)$ is *satisfied* by a partial database $\mathcal{D}$, denoted as $\mathcal{D} \models Compl(Q)$, if $Q(D^a) = Q(D^i)$.

Example 2. *Consider the above defined partial database $\mathcal{D}_S$ and the query*

$$Q_1(n) :- student(n, l, c), person(n, 'male'),$$

asking for all male students. Over both, the available database D^a_S and the ideal database D^i_S, this query returns exactly Hans. Thus, $\mathcal{D}_S$ satisfies the query completeness statement for Q_1, that is,

$$\mathcal{D}_S \models Compl(Q_1). \quad \square$$

Abiteboul et al. [1] introduced the notion of certain and possible answers over databases under the open-world assumption. Query completeness can also be seen as a relation between certain and possible answers: A query over a partially complete database is complete, if the certain and the possible answers coincide.

Table completeness. A table completeness (TC) statement allows one to say that a certain part of a relation is complete, without requiring the completeness of other parts of the database [8]. It has two components, a relation R and a condition G. Intuitively, it says that all tuples of the ideal relation R that satisfy condition G in the ideal database are also present in the available relation R.

Formally, let $R(\bar{s})$ be an R-atom and let G be a condition such that $R(\bar{s}), G$ is safe. We remark that G can contain relational and built-in atoms and that we do not make any safety assumptions about G alone. Then $Compl(R(\bar{s}); G)$ is a *table completeness statement*. It has an *associated query*, which is defined as $Q_{R(\bar{s});G}(\bar{s}) :- R(\bar{s}), G$. The statement is satisfied by $\mathcal{D} = (D^i, D^a)$, written $\mathcal{D} \models Compl(R(\bar{s}); G)$, if $Q_{R(\bar{s});G}(D^i) \subseteq R(D^a)$. Note that the ideal instance $\hat{D}$ is used to determine those tuples in the ideal version $R(D^i)$ that satisfy G and that the statement is satisfied if these tuples are present in the available version $R(D^a)$. In the sequel, we will denote a TC statement generically as C and refer to the associated query simply as Q_C.

If we introduce different schemas Σ^i and Σ^a for the ideal and the available database, respectively, we can view the TC statement $C = Compl(R(\bar{s}); G)$ equivalently as the TGD (= tuple-generating dependency) δ_C: $R^i(\bar{s}), G^i \rightarrow R^a(\bar{s})$ from Σ^i to

Σ^a. It is straightforward to see that a partial database satisfies the TC statement C if and only if it satisfies the TGD δ_C.

The view of TC statements is especially useful for implementations.

Example 3. *In the partial database $\mathcal{D}_S$ defined above, we can observe that in the available relation person, the teacher Carlo is missing, while all students are present. Thus, person is complete for all students. The available relation student contains Hans, who is the only male student. Thus, student is complete for all male persons. Formally, these two observations can be written as table completeness statements:*

$$C_1 = Compl(person(n, g); student(n, l, c)),$$
$$C_2 = Compl(student(n, l, c); person(n, 'male')),$$

which, as seen, are satisfied by the partial database $\mathcal{D}_S$. □

One can prove that table completeness cannot be expressed by query completeness statements, because the latter require completeness of the relevant parts of all the tables that appear in the statement, while the former only talks about the completeness of a single table.

Example 4. *As an illustration, consider the table completeness statement C_1 that states that person is complete for all students. The corresponding query Q_{C_1} that asks for all persons that are students is*

$$Q_{C_1}(n, g) :- person(n, g), student(n, l, c).$$

Evaluating Q_{C_1} over D_S^i gives the result {Hans, Maria}. However, evaluating it over D_S^a returns only {Hans}. Thus, $\mathcal{D}_S$ does not satisfy the completeness of the query Q_{C_1} although it satisfies the table completeness statement C_1. □

Reasoning. As usual, a set $\mathcal{S}_1$ of TC- or QC-statements *entails* another set $\mathcal{S}_2$ (we write $\mathcal{S}_1 \models \mathcal{S}_2$) if every partial database that satisfies all elements of $\mathcal{S}_1$ also satisfies all elements of $\mathcal{S}_2$.

Example 5. *Consider the query $Q(n) :- student(n, 7, c)$, person(n,' male') that asks for all male students in level 7. The TC statements C_1 and C_2 entail completeness of this query, because we ensure that all persons that are students and all male students are in the database. Note that these are not the minimal preconditions, as it would be enough to only have male persons in the database who are student in level 7, and students in level 7, who are male persons.*

While TC statements are a natural way to describe completeness of available data ("These parts of the data are complete"), QC statements capture requirements for data quality ("For these queries we need complete answers"). Thus, checking whether a set of TC statements entails a set of QC statements (TC-QC entailment) is the practically most relevant inference. Checking TC-TC entailment is useful when managing sets of TC statements. Moreover, as we will show later on, TC-QC entailment for aggregate queries with count and sum can be reduced to TC-TC entailment for non-aggregate queries. If completeness guarantees are given in terms of query completeness, also QC-QC entailment is of interest.

3. CHARACTERIZATIONS AND DECISION PROCEDURES

Motro [10] introduced the notion of partially incomplete and incorrect databases as databases that can both miss facts that hold in the real world or contain facts that do not hold there. He described partial completeness in terms of *query completeness* (QC) statements, which express that the answer of a query is complete. The query completeness statements express that to some parts of the database the closed-world assumption applies, while for the rest of the database, the open-world assumption applies. He studied how the completeness of a given query can be deduced from the completeness of other queries, which is QC-QC entailment. His solution was based on rewriting queries using views: to infer that a given query is complete whenever a set of other queries are complete, he would search for a conjunctive rewriting in terms of the complete queries. This solution is correct, but not complete, as later results on query determinacy show: the given query may be complete although no conjunctive rewriting exists.

While Levy et al. could show that rewritability of conjunctive queries as conjunctive queries is decidable [9], general rewritability of conjunctive queries by conjunctive queries is still open: An extensive discussion on that issue was published in 2005 by Segoufin and Vianu where it is shown that it is possible that conjunctive queries can be rewritten using other conjunctive queries, but the rewriting is not a conjunctive query [19]. They also introduced the notion of query determinacy, which for conjunctive queries implies second order rewritability. The decidability of query determinacy for conjunctive queries is an open problem to date.

Halevy [8] suggested local completeness statements, which we, for a better distinction from the QC statements, call table completeness (TC) statements, as an alternate formalism for expressing partial completeness of an incomplete database. These statements allow one to express completeness of parts of relations independent from the completeness of other parts of the database. The main problem he addressed was how to derive query completeness from table completeness (TC-QC). He reduced TC-QC to the problem of queries independent of updates (QIU) [5]. However, this reduction introduces negation, and thus, except for trivial cases, generates QIU instances for which no decision procedures are known. As a consequence, the decidability of TC-QC remained largely open. Moreover, he demonstrated that by taking into account the concrete database instance and exploiting the key constraints over it, additional queries can be shown to be complete.

Razniewski and Nutt provided decision procedures for TC-QC in [13]. They showed that for queries under bag semantics and for minimal queries under set semantics, weakest preconditions for query completeness can be expressed in terms of table completeness statements, which allow to reduce TC-QC entailment to TC-TC entailment.

For the problem of TC-TC entailment, they showed that it is equivalent to query containment.

For QC-QC entailment, they showed that the problem is decidable for queries under bag semantics.

For aggregate queries, they showed that for the aggregate functions SUM and COUNT, TC-QC has the same complexity as TC-QC for nonaggregate queries under bag semantics. For the aggregate functions MIN and MAX, they showed that

Problem	Work by	Results
QC-QC	Motro 1989	Query rewritability is a sufficient condition for QC-QC^s
	Razniewski/Nutt 2011	QC-QC^b is equivalent to query containment
TC-TC	Razniewski/Nutt 2011	TC-TC is equivalent to query containment
TC-QC	Levy 1996	Decision procedure for trivial cases
	Razniewski/Nutt 2011	TC-QC^b is equivalent to TC-TC, TC-QC^s is equivalent to TC-TC up to asymmetric cases
	Razniewski/Nutt 2012	Decision procedures for TC-QC^s over databases with nulls

Table 1: Main results

TC-QC has the same complexity as TC-QC for nonaggregate queries under set semantics.

For reasoning wrt. a database instance, they showed that TC-QC becomes computationally harder than without an instance, while QC-QC surprisingly becomes solvable, whereas without an instance, decidability is open.

In [12], Nutt and Razniewski discussed TC-QC entailment reasoning over databases that contain null values. Null values as used in SQL are ambiguous, as they can indicate either that no attribute value exists or that a value exists, but is unknown. Nutt and Razniewski studied completeness reasoning for both interpretations, and showed that when allowing both interpretations at the same time, it becomes necessary to syntactically distinguish between different kinds of null values. They presented an encoding for doing that in standard SQL databases. With this technique, any SQL DBMS evaluates complete queries correctly with respect to the different meanings that null values can carry.

The main results are summarized in Table 1.

4. IMPLEMENTATION TECHNIQUES

Systems for reasoning can be developed from scratch, however it may be useful to implement them using existing technology as far as possible. So far, it was investigated how completeness reasoning can be reduced to answer set programming, in particular using the DLV system.

The MAGIK system developed by Savkovic et al. [18] demonstrates how to use meta-information about the completeness of a database to assess the quality of the answers returned by a query. The system holds table-completeness (TC) statements, by which one can express that a table is partially complete, that is, it contains all facts about some aspect of the domain.

Given a query, MAGIK determines from such meta-information whether the database contains sufficient data for the query answer to be complete (TC-QC entailment). If, according to the TC statements, the database content is not sufficient for a complete answer, MAGIK explains which further TC statements are needed to guarantee completeness.

MAGIK extends and complements theoretical work on modeling and reasoning about data completeness by providing the first implementation of a reasoner. The reasoner operates by translating completeness reasoning tasks into logic programs, which are executed by an answer set engine.

In [17], Savkovic et al. present an extension to MAGIK that computes for a query that may be incomplete, complete approximations from above and from below. With this extension, they show how to reformulate the original query in such a way that answers are guaranteed to be complete. If there exists a more general complete query, there is a unique most specific one, which is found. If there exists a more specific complete query, there may even be infinitely many. In this case, the least specific specializations whose size is bounded by a threshold provided by the user is found. Generalizations are computed by a fixpoint iteration, employing an answer set programming engine. Specializations are found leveraging unification from logic programming.

5. EXTENSIONS AND APPLICATIONS SCENARIOS

Complete generalizations and specializations. When a query is not guaranteed to be complete, it may be interesting to know which similar queries are complete. For instance, when a query for all students in level 5 is not complete, it may still be the case that the query for students in classes 5b and 5c is complete. Such information is especially interesting for interaction with a completeness reasoning system. In [11], Savkovic et al. defined the notion of most general complete specialization and the most specific comple generalization, and discussed techniques to find those.

Completeness over Business Processes. In many applications, data is managed via well documented processes. If information about such processes exists, one can draw conclusions about completeness as well. In [15], Razniewski et al. presented a formalization of so-called *quality-aware processes* that create data in the real world and store it in the company's information system possibly at a later point. They then showed how one can check the completeness of database queries in a certain state of the process or after the execution of a sequence of actions, by leveraging on query containment, a well-studied problem in database theory. Finally, they showed how the results can be extended to the more expressive formalism of colored Petri nets.

Spatial Data. Volunteered geographical information systems are gaining popularity. The most established one is OpenStreetMap (OSM), but also classical commercial map services such as Google Maps now allow users to take part in

the content creation.

Assessing the quality of spatial information is essential for making informed decisions based on the data, and particularly challenging when the data is provided in a decentralized, crowd-based manner. In [14], Razniewski and Nutt showed how information about the completeness of features in certain regions can be used to annotate query answers with completeness information. They provided a characterization of the necessary reasoning and show that when taking into account the available database, more completeness can be derived. OSM already contains some completeness statements, which are originally intended for coordination among the editors of the map. A contribution was also to show that these statements are not only useful for the producers of the data but also for the consumers.

RDF Data. With thousands of RDF data sources today available on the Web, covering disparate and possibly overlapping knowledge domains, the problem of providing high-level descriptions (in the form of metadata) of their content becomes crucial. In [3], Darari et al. discussed reasoning about the completeness of semantic web data sources. They showed how the previous theory can be adapted for RDF data sources, what peculiarities the SPARQL query language offers and how completeness statements themselves can be expressed in RDF.

They also discussed the foundation for the expression of completeness statements about RDF data sources. This allows to complement with *qualitative* descriptions about completeness the existing proposals like VOID that mainly deal with *quantitative* descriptions. The second aspect of their work is to show that completeness statements can be useful for the semantic web in practice. On the theoretical side, they provide a formalization of completeness for RDF data sources and techniques to reason about the completeness of query answers. From the practical side, completeness statements can be easily embedded in current descriptions of data sources and thus readily used. The results on RDF data have been implemented by Darari et al. in a demo system called CORNER [6].

6. CURRENT WORK

In this section we list problems that our group is currently working on.

6.1 SPARQL Queries with Negation

RDF data is often treated as incomplete, following the Open-World Assumption. On the other hand, SPARQL, the standard query language over RDF, usually follows the Closed-World Assumption, assuming RDF data to be complete. What then happens is the semantic gap between RDF and SPARQL. In current work, Darari et al. [4] address how to close the semantic gap between RDF and SPARQL, in terms of certain answers and possible answers using completeness statements. Table 2 shows current results for the relations between query answers, certain answers and possible answers for queries with negation. The queries are assumed to be of the form $Q(\bar{s}) :- P^+, \neg P^-$, where P^+ is the positive part and P^- is the negative part. Then we use letters C and N to indicate which parts are complete. E.g. $Q(\bar{s}) :- N, \neg C$ indicates that the positive part is not complete and the negative part is complete. As the table shows, depending on the complete parts, the query answer may either be equal to the possible answers, to the certain answers, both, or none.

Completeness P Pattern	Relationship between Certain Answers, Query Answers, and Possible Answers
$Q :- C$	$CA = QA = PA$
$Q :- N$	$CA = QA \subseteq PA = inf$
$Q :- N, \neg N$	$\emptyset = CA \subseteq QA \subseteq PA = inf$
$Q :- C, \neg C$	$CA = QA = PA$
$Q :- N, \neg C$	$CA = QA \subseteq PA = inf$
$Q :- C, \neg N$	$\emptyset = CA \subseteq QA = PA$

Table 2: Relation between query result, certain answers and possible answers for queries with negation. The arguments of Q are irrelevant and therefore omitted.

Note that the above results hold for conjunctive queries in general, and thus do not only apply to SPARQL but also to other query languages with negation, such as SQL.

6.2 Instance Reasoning

Another line of current work concerns completeness reasoning wrt. a database instance. We are currently looking into completeness statements which are simpler than TC statements in the sense that we do not contain any joins. For such statements, reasoning is still exponential in the size of the database schema, but experimental results suggest that in use cases, the reasoning is feasible. A challenge is however to develop a procedure which is algorithmically complete.

7. ACKNOWLEDGEMENT

We thank our collaborators Fariz Darari, Flip Korn, Paramita Mirza, Marco Montali, Sergey Paramonov, Giuseppe Pirró, Radityo Eko Prasojo, Ognjen Savkovic and Divesh Srivastava.

This work has been partially supported by the project "MAGIC: Managing Completeness of Data" funded by the province of Bozen-Bolzano.

8. REFERENCES

[1] S. Abiteboul, P.C. Kanellakis, and G. Grahne. On the representation and querying of sets of possible worlds. In *Proc. SIGMOD*, pages 34–48, 1987.

[2] Keith L Clark. Negation as failure. In *Logic and data bases*, pages 293–322. Springer, 1978.

[3] Fariz Darari, Werner Nutt, Giuseppe Pirrò, and Simon Razniewski. Completeness statements about RDF data sources and their use for query answering. In *International Semantic Web Conference (1)*, pages 66–83, 2013.

[4] Fariz Darari, Simon Razniewski, and Werner Nutt. Bridging the semantic gap between RDF and SPARQL using completeness statements. *ISWC*, 2013.

[5] Ch. Elkan. Independence of logic database queries and updates. In *Proc. PODS*, pages 154–160, 1990.

[6] Radityo Eko Prasojo Fariz Darari and Werner Nutt. CORNER: A completeness reasoner for the semantic web (poster). *ESWC*, 2013.

[7] T. Imieliński and W. Lipski, Jr. Incomplete information in relational databases. *J. ACM*, 31:761–791, 1984.

[8] Alon Y. Levy. Obtaining complete answers from incomplete databases. In *Proceedings of the International Conference on Very Large Data Bases*, pages 402–412, 1996.

[9] Alon Y. Levy, Alberto O. Mendelzon, Yehoshua Sagiv, and Divesh Srivastava. Answering queries using views. In *PODS*, pages 95–104, 1995.

[10] A. Motro. Integrity = Validity + Completeness. *ACM TODS*, 14(4):480–502, 1989.

[11] Werner Nutt, Sergey Paramonov, and Ognjen Savkovic. An ASP approach to query completeness reasoning. *TPLP*, 13(4-5-Online-Supplement), 2013.

[12] Werner Nutt and Simon Razniewski. Completeness of queries over SQL databases. In *CIKM*, pages 902–911, 2012.

[13] S. Razniewski and W. Nutt. Completeness of queries over incomplete databases. In *VLDB*, 2011.

[14] S. Razniewski and W. Nutt. Assessing the completeness of geographical data (short paper). In *BNCOD*, 2013.

[15] Simon Razniewski, Marco Montali, and Werner Nutt. Verification of query completeness over processes. In *BPM*, pages 155–170, 2013.

[16] Raymond Reiter. On closed world data bases. In *Logic and Data Bases*, pages 55–76, 1977.

[17] Ognjen Savkovic, Paramita Mirza, Sergey Paramonov, and Werner Nutt. Magik: managing completeness of data. In *CIKM*, pages 2725–2727, 2012.

[18] Ognjen Savkovic, Paramita Mirza, Alex Tomasi, and Werner Nutt. Complete approximations of incomplete queries. *PVLDB*, 6(12):1378–1381, 2013.

[19] L. Segoufin and V. Vianu. Views and queries: Determinacy and rewriting. In *Proc. PODS*, pages 49–60, 2005.