Relative Information Completeness

Wenfei Fan
University of Edinburgh &
Bell Labs
wenfei@inf.ed.ac.uk

Floris Geerts
School of Informatics
University of Edinburgh
fgeerts@inf.ed.ac.uk

Abstract

The paper investigates the question of whether a partially closed database has complete information to answer a query. In practice an enterprise often maintains master data D_m, a closed-world database. We say that a database D is *partially closed* if it satisfies a set V of containment constraints of the form $q(D) \subseteq p(D_m)$, where q is a query in a language $\mathcal{L}_C$ and p is a projection query. The part of D not constrained by (D_m, V) is open, from which some tuples may be missing. The database D is said to be *complete* for a query Q *relative to* (D_m, V) if for all partially closed extensions D' of D, $Q(D') = Q(D)$, *i.e.,* adding tuples to D either violates some constraints in V or does not change the answer to Q.

We first show that the proposed model can also capture the consistency of data, in addition to its relative completeness. Indeed, integrity constraints studied for consistency can be expressed as containment constraints. We then study two problems. One is to decide, given D_m, V, a query Q in a language $\mathcal{L}_Q$ and a partially closed database D, whether D is complete for Q relative to (D_m, V). The other is to determine, given D_m, V and Q, whether there exists a partially closed database that is complete for Q relative to (D_m, V). We establish matching lower and upper bounds on these problems for a variety of languages $\mathcal{L}_Q$ and $\mathcal{L}_C$. We also provide characterizations for a database to be relatively complete, and for a query to allow a relatively complete database, when $\mathcal{L}_Q$ and $\mathcal{L}_C$ are conjunctive queries.

Categories and Subject Descriptors: H.2.3 [**Information Systems**]: Database Management – *Languages*; F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic — *Computational Logic*
General Terms: Languages, Theory, Design.

1. Introduction

One of the issues central to data quality concerns incomplete information. Given a database D and a query Q, we want to know whether Q can be answered by using the data in D. If the information in D is incomplete, one can hardly expect its answer to Q to be accurate. Incomplete information introduces serious problems to enterprises: it routinely leads to misleading analytical results and biased decisions, and accounts for loss of revenues, credibility and customers [4].

The study of incomplete information is almost as old as the relational model itself. There has been a host of work on *missing values*, under the Closed World Assumption (CWA). That is, all the tuples representing real-world entities are assumed already in place, but the values of some fields in these tuples are missing. As a result, facts that are not in the database are assumed to be false. To this end, a number of approaches have been proposed, notably representation systems for a set of possible worlds (*e.g., v*-tables, *c*-tables, OR-object databases, *e.g.,* [16, 17]) and disjunctive logic programming (see [27] for a comprehensive survey).

Equally important to data quality is how to handle *missing tuples*, under the Open World Assumption (OWA). That is, a database may only be a proper subset of the set of tuples that represent real-world entities. While there has also been work on missing tuples (*e.g.,* [15, 18, 22, 28]), this issue has received relatively less attention. Under OWA, one can often expect few sensible queries to find complete answers.

In several emerging applications, however, neither CWA nor OWA is quite appropriate. This is evident in, *e.g.,* Master Data Management (MDM [11, 24, 21]), one of the fastest growing software markets. An enterprise nowadays typically maintains *master data* (*a.k.a. reference data*), a single repository of high-quality data that provides various applications with a synchronized, consistent view of the core business entities of the enterprise. The master data contains complete information about the enterprise in certain categories, *e.g.,* employees, departments and projects.

Master data can be regarded as a closed-world database. Meanwhile a number of other databases may be in use in the enterprise for, *e.g.,* sales, project control and customer support. On one hand, these databases may not be complete, *e.g.,* some sale transactions may be missing. On the other hand, certain parts of the databases are *constrained by* the master data, *e.g.,* employees and projects. In other words, these databases are neither entirely closed-world, nor entirely open-world. It becomes more interesting to decide whether the information available in these databases is complete to answer a query.

Example 1.1: Consider a company that maintains a master data relation $\mathsf{Proj}(\mathsf{emp}, \mathsf{proj})$, which keeps track of people working on projects of a set C_0 of clients, including AL. The company also has a database $\mathsf{Work\_on}(\mathsf{emp}, \mathsf{proj}, \mathsf{cli})$ in which a tuple (e, p, c) indicates that employee e works on project p of client c. $\mathsf{Work\_on}$ is not part of the master data.

Consider a query Q_1 posed on $\mathsf{Work\_on}$ to find all employees who work on project p_0. The query may *not* get a complete answer since some tuples may be missing from

PODS'09, June 29–July 2, 2009, Providence, Rhode Island, USA.

Work_on. However, if we know that p_0 is a project of client AL and if all tuples of $\mathsf{Proj}(e, p_0)$ can be extracted from Work_on, then we can safely conclude that query Q_1 can find a complete answer from Work_on. That is, there is no need to add more tuples to Work_on in order to answer Q_1.

Alternatively, suppose that there is a constraint which asserts that for any project, at most k people can work on it. Then if the answer to Q_1 in Work_on returns k employees, we can also conclude that the answer is complete.

As another example, suppose that the master data contains a relation $\mathsf{Manage}_m(\mathsf{emp}_1, \mathsf{emp}_2)$, in which each tuple (e_1, e_2) indicates that employee e_2 directly reports to e_1. There is another relation $\mathsf{Manage}(\mathsf{emp}_1, \mathsf{emp}_2)$ that is not part of master data, but it contains all tuples in Manage_m. Consider query Q_2 on Manage to find all the people above e_0 in the management hierarchy, *i.e.,* the people to whom e_0 reports, directly or indirectly. Note that if Q_2 is in, *e.g.,* datalog, then we may expect the answer to Q_2 to be complete. In contrast, if Q_2 is a conjunctive query, then the answer to Q_2 is incomplete unless Manage contains the transitive closure of Manage_m. This tells us that the completeness of information is also relative to the query language in use. □

In this context several natural questions have to be answered. Given a query Q posed on a database D that is partially constrained by master data D_m, can we find complete information from D to answer Q? Does there exist a database D at all that has complete information to answer Q? These questions are not only of theoretical interest, but are also important in practice. Indeed, the ability to answer these questions not only helps us determine whether a query can find a complete answer from a particular database, but also provides guidance for what data should be collected in order to answer a query. The increasing demand for MDM highlights the need for a full treatment of the completeness of information *relative to* master data and queries.

Relative completeness. In response to the need, we propose a notion of relative information completeness.

To characterize databases D that are partially constrained by master data D_m, we specify a set V of *containment constraints*. A containment constraint is of the form $q(D) \subseteq p(D_m)$, where q is a query in a language $\mathcal{L}_C$ posed on D, and p is a simple projection query on D_m. Intuitively, the part of D that is constrained by V is bounded by D_m, while the rest is open-world. We refer to a database D that satisfies V as a *partially closed* database *w.r.t.* (D_m, V).

For a query Q in a language $\mathcal{L}_Q$, a partially closed database D is said to be complete *w.r.t.* (D_m, V) if for all partially closed extensions D' of D *w.r.t.* (D_m, V), $Q(D') = Q(D)$. That is, there is no need for adding new tuples to D, since it either violates the containment constraints, or does not change the answer to Q. In other words, D already has complete information necessary for answering Q.

To simplify the discussion, we focus on missing tuples in this paper. As will be addressed in Section 5, the notion of relatively complete information can be extended to accommodate missing values as well, by capitalizing on representation systems for possible worlds [16, 17].

Completeness and consistency. Another critical issue to data quality is the consistency of the data. To answer a query using a database D, one naturally wants the information in D to be both complete and consistent.

To capture inconsistencies one typically makes use of integrity constraints (*e.g.,* [3, 5, 6, 14], see [8, 13] for recent surveys). That is, inconsistencies and errors in the data are detected as violations of the constraints. In light of this one might be tempted to extend the notion of partially closed databases by incorporating integrity constraints.

The good news is that there is no need to overburden the notion with a set of integrity constraints. We show that constraints studied for ensuring data consistency, such as denial constraints [3], conditional functional dependencies [14] and conditional inclusion dependencies [5], are expressible as simple containment constraints. As a result, we can assure that only consistent and partially closed databases are considered, by enforcing containment constraints. That is, in a uniform framework we can deal with both relative information completeness and consistency.

Main results. We investigate two important decision problems associated with the relative completeness of information, and establish their complexity bounds. We also provide characterizations for a database to be relatively complete and for a query to allow a relatively complete database, in certain cases when the decision problems are decidable.

Determining relatively complete databases. One of the two problems, referred to as the *relatively complete database problem*, is to determine, given a query Q, master data D_m, a set V of containment constraints, and a partially closed database D *w.r.t.* (D_m, V), whether or not D is complete for Q relatively to (D_m, V). That is to decide, when Q is posed on D, whether the answer of D to Q is complete.

We parameterize the problem with various $\mathcal{L}_Q$ and $\mathcal{L}_C$, the query languages in which the queries are expressed and in which the containment constraints are defined, respectively. We consider the following $\mathcal{L}_Q$ and $\mathcal{L}_C$, all with equality '$=$' and inequality '$\neq$':

- conjunctive queries (CQ),
- union of conjunctive queries (UCQ),
- positive existential FO queries ($\exists$FO$^+$),
- first-order queries (FO), and
- datalog (FP).

We establish lower and upper bounds for the problem *w.r.t.* all these languages, *all matching*, either Π_2^p-complete or undecidable. The complexity bounds are rather robust: the lower bounds remain intact even when D_m and V are predefined and fixed. The problem is already Π_2^p-complete for CQ queries and containment constraints defined as inclusion dependencies (INDs), when D_m and V are fixed.

Determining relatively complete queries. The other problem, referred to as the *relatively complete query problem*, is to determine, given Q, D_m and V, whether there exists a partially closed database D that is complete for Q relatively to (D_m, V). It is to decide for Q whether it is possible to find a relatively complete database D at all. If such a D exists, Q is said to be a *query relatively complete w.r.t.* (D_m, V).

We present complexity bounds for the problem when $\mathcal{L}_Q$ and $\mathcal{L}_C$ range over CQ, UCQ, $\exists$FO$^+$, FO and FP. The lower and upper bounds are again all matching: coNP-complete, NEXPTIME-complete, or undecidable. In contrast to its counterpart for relative complete databases, fixed D_m and V make our lives easier: the problem becomes Σ_3^p-complete as opposed to NEXPTIME-complete in certain cases.

Characterizations. When $\mathcal{L}_Q$ and $\mathcal{L}_C$ are CQ, we present sufficient and necessary conditions for (a) a partially closed

database D to be complete for a query Q relative to (D_m, V), and (b) a query to be relatively complete *w.r.t.* (D_m, V). As remarked earlier, the characterizations tell us what data should be collected in D in order to answer a query, and whether a query can find a complete answer at all. The characterizations can be extended to UCQ and $\exists$FO$^+$.

To the best of our knowledge, this work is among the first efforts to study the completeness of information in emerging applications such as MDM. Our results provide a comprehensive picture of complexity bounds for important decision problems associated with relatively complete information. Moreover, the results provide guidance for how to make a database relatively complete. A variety of techniques are used to prove the results, including a wide range of reductions and constructive proofs with algorithms.

Related work. Several approaches have been proposed to represent or query databases with missing tuples. In [28], a complete and consistent extension of an incomplete database D is defined to be a database D_c such that $D \subseteq \pi_L(D_c)$ and $D_c \models \Sigma$, where π is the projection operator, L is the set of attributes in D, and Σ is a set of integrity constraints. Complexity bounds for computing the set of complete and consistent extensions of D *w.r.t.* Σ are established there. A notion of *open null* is introduced in [15] to model locally controlled open-world databases: parts of a database D, values or tuples, can be marked with open null and are assumed open-world, while the rest is closed. Relational operators are extended to tables with open null. In contrast to [15], this work aims to model databases partially constrained by master data D_m and consistency specifications, both via containment constraints. In addition, we study decision problems that are not considered in [15].

Partially complete databases D have also been studied in [22], which assumes a virtual database D_c with "complete information", and assumes that part of D is known as a view of D_c. It investigates the query answer completeness problem, the problem for determining whether a query posed on D_c can be answered by an equivalent query on D. In this setting, the problem can be reduced to query answering using views. Along the same lines, [18] assumes that D contains some CQ views of D_c. It reduces the query answer completeness problem to the independence problem for deciding the independence of queries from updates [19]. As opposed to [18, 22], we assume neither D_c with complete information nor that an incomplete database D contains some views of D_c. Instead, we consider D_m as an "upper bound" of certain information in D. Moreover, the decision problems studied here can be reduced to neither the query rewriting problem nor the independence problem (see below).

There has also been work on modeling negative information via logic programming (see [27]), which considers neither partially complete databases nor the decision problems studied in this work.

We now clarify the difference between our decision problems and the independence problem (*e.g.,* [12, 19]). The latter is to determine whether a query Q is independent of updates generated by another query Q^u, such that *for all* databases D, $Q(D) = Q(D \oplus \Delta)$, where Δ denotes updates generated by Q^u. In contrast, we consider relatively complete queries Q, such that *there exists* a database D complete for Q relative to master data D_m and containment constraints V, where D and D_m satisfy V. We want to decide (a) whether for a query Q *there exists* a relatively complete database D, and (b) whether a given D that satisfies V is a witness for Q to be relatively complete. Due to the difference between the problems, results for the independence problem do not carry over to ours, and vice versa.

One may think of an incomplete database as a "view" of a database with complete information. There has been a large body of work on answering queries using views (*e.g.,* [1, 7, 20, 25]), to determine certain answers [1], compute complete answers from views with limited access patterns [10, 20], or to decide whether views determine queries [25] or are lossless [7]. This work differs from that line of research in that one may not find a definable view to characterize a relatively complete database D in terms of the database with complete information. Indeed, D is only *partially* constrained by master data D_m, while D_m itself may not contain the complete information that D intends to represent.

There has also been recent work on consistent query answering (*e.g.,* [3, 6, 8]). That is to decide whether a tuple is in the answer to a query in every repair of a database D, where a repair is a database that satisfies a given set of integrity constraints and moreover, minimally differs from the original D *w.r.t.* some repair model. Master data D_m is not considered there, and we do not consider repairs in this paper. Note that most containment constraints are *not* expressible as integrity constraints studied for consistency.

Organization. In Section 2 we define relatively complete databases and queries, state the decision problems, and show that integrity constraints for capturing inconsistencies can be expressed as containment constraints. We provide complexity bounds and characterizations for determining relatively complete databases in Section 3, and for deciding relatively complete queries in Section 4. Section 5 summarizes the main results of the paper and identifies open problems.

2. Relatively Complete Databases and Queries

In this section we first present the notion of relative completeness of data. We then show that the consistency of the data can be characterized in the uniform framework.

2.1 Relative Completeness

We start with specifications of databases and master data.

Databases and master data. A database is specified by a relational schema $\mathcal{R}$, which consists of a collection of relation schemas $(R_1, \ldots, R_n)$. Each schema R_i is defined over a fixed set of attributes. For each attribute A, its domain is specified in R, denoted as $dom(A)$. To simplify the discussion we consider two domains: a countably infinite set $\mathbf{d}$ and a finite set $\mathbf{d}_f$ with at least two elements. We assume that $dom(A)$ is either infinite ($\mathbf{d}$) or finite ($\mathbf{d}_f$).

We say that an instance $D = (I_1, \ldots, I_n)$ of $\mathcal{R}$ is *contained in* another instance $D' = (I'_1, \ldots, I'_n)$ of $\mathcal{R}$, denoted by $D \subseteq D'$, if $I_j \subseteq I'_j$ for all $j \in [1, n]$.

Master data (*a.k.a.* reference data) is a closed-world database D_m, specified by a relational schema $\mathcal{R}_m$. As remarked earlier, an enterprise typically maintains master data that is assumed consistent and complete about certain information of the enterprise [11, 24, 21]. We do not impose any restriction on the relational schemas $\mathcal{R}$ and $\mathcal{R}_m$.

Containment constraints. Let $\mathcal{L}_C$ be a query language.

A *containment constraint* (CC) ϕ_v *in* $\mathcal{L}_C$ is of the form $q_v(\mathcal{R}) \subseteq p(\mathcal{R}_m)$, where q_v is a query in $\mathcal{L}_C$ defined over

schema $\mathcal{R}$, and p is a projection query over schema $\mathcal{R}_m$. An instance D of $\mathcal{R}$ and master data instance D_m of $\mathcal{R}_m$ *satisfy* ϕ_v, denoted by $(D, D_m) \models \phi_v$, if $q_v(D) \subseteq p(D_m)$.

We say that D and D_m *satisfy* a set V of CCs, denoted by $(D, D_m) \models V$, if for each $\phi_v \in V$, $(D, D_m) \models \phi_v$.

Intuitively, ϕ_v assures that D_m is an "upper bound" of the information extracted by $q_v(D)$. In other words, CWA is asserted for D_m that constrains the part of data identified by $q_v(D)$ from D. That is, while this part of D can be extended, the expansion cannot go beyond the information already in D_m. On the other hand, OWA is assumed for the part of D that is not constrained by ϕ_v.

We write $q_v(\mathcal{R}) \subseteq p(\mathcal{R}_m)$ as $q_v \subseteq p$ when $\mathcal{R}$ and $\mathcal{R}_m$ are clear in the context. We write $q_v \subseteq p$ as $q_v \subseteq \emptyset$ if p is a projection on an empty master relation.

Example 2.1: Recall Work_on from Example 1.1. We can write a CC $\phi_0 = q(\mathsf{Work\_on}) \subseteq \mathsf{Proj}(e, p)$ in the language of union of conjunctive queries, where

$$q(e, p) = \exists c\big(\bigvee\nolimits_{c \in C_0} \mathsf{Work\_on}(e, p, c)\big),$$

and C_0 is the set of clients described in Example 1.1.

Another CC ϕ_1 in conjunctive queries is $q \subseteq \emptyset$, where

$$q(p) = \exists e_1, \ldots, e_{k+1}\big(\textstyle\bigwedge_{i \in [1,k+1]} \mathsf{Work\_on}(e_i, p) \wedge \bigwedge_{i,j \in [1,k+1], i \neq j} (e_i \neq e_j)\big).$$

It asserts that at most k people can work on a project. □

A database D is referred to as a *partially closed* database *w.r.t.* (D_m, V) if $(D, D_m) \models V$.

Observe that a CC $q_v(\mathcal{R}) \subseteq p(\mathcal{R}_m)$ is an *inclusion dependency* (IND) when q_v is also a projection query. In the sequel we simply refer to such CCs as INDs.

Relative completeness. Let $\mathcal{L}_Q$ be a query language, not necessarily the same as $\mathcal{L}_C$. Let Q be a query in $\mathcal{L}_Q$.

Consider a partially closed database D *w.r.t.* master data D_m and a set V of CCs. We say that D is *complete for query Q relative to* (D_m, V) if for all instances D' of $\mathcal{R}$, if $D \subseteq D'$ and $(D', D_m) \models V$, then $Q(D) = Q(D')$.

That is, D is complete for Q relative to (D_m, V) if (a) D is partially closed *w.r.t.* (D_m, V), and (b) for any partially closed extension D' of D, $Q(D) = Q(D')$.

Observe that if D is complete for Q relative to (D_m, V) then no matter how D is expanded by including new tuples, as long as the extension does not violate V, the answer to query Q remains unchanged. That is, D has complete information for answering Q. The notion is *relative to* the master data D_m and CCs in V: the extensions of D should not violate the CCs in V, *i.e.,* $(D', D_m) \models V$. That is, CWA for D_m is observed.

Given D_m, V and a query Q in $\mathcal{L}_Q$, we define *the set of complete databases* for Q *w.r.t.* (D_m, V), denoted by $\mathsf{RCQ}(Q, D_m, V)$, to be the set of all complete databases for Q relative to (D_m, V).

When $\mathsf{RCQ}(Q, D_m, V)$ is nonempty, Q is called a *relatively complete query w.r.t.* (D_m, V).

Intuitively, Q is relatively complete if it is possible to find a database D such that the answer to Q in D is complete.

Example 2.2: Relation Work_on described in Example 1.1 is complete for query Q_1 *w.r.t.* Proj and the CC ϕ_0 of Example 2.1. Since Work_on is in $\mathsf{RCQ}(Q_1, \mathsf{Proj}, \phi_0)$, Q_1 is relatively complete *w.r.t.* (Proj, ϕ_0).

On the other hand, relation Manage of Example 1.1 is not complete for the CQ query Q_2. However, Q_2 is relatively complete *w.r.t.* Manage_m: one can make Manage complete for Q_2 by including the transitive closure of Manage_m. □

Decision problems. In this paper we study two decision problems. One is the *relatively complete database problem* for $\mathcal{L}_Q$ and $\mathcal{L}_C$, denoted by $\mathsf{RCDP}(\mathcal{L}_Q, \mathcal{L}_C)$ and stated as:

PROBLEM:	$\mathsf{RCDP}(\mathcal{L}_Q, \mathcal{L}_C)$
INPUT:	A query $Q \in \mathcal{L}_Q$, master data D_m, a set V of CCs in $\mathcal{L}_C$, and a partially closed database D *w.r.t.* (D_m, V).
QUESTION:	Is D in $\mathsf{RCQ}(Q, D_m, V)$? That is, is D complete for Q relative to (D_m, V)?

The other one is the *relatively complete query problem* for $\mathcal{L}_Q$ and $\mathcal{L}_C$, denoted by $\mathsf{RCQP}(\mathcal{L}_Q, \mathcal{L}_C)$ and stated as:

PROBLEM:	$\mathsf{RCQP}(\mathcal{L}_Q, \mathcal{L}_C)$
INPUT:	A query $Q \in \mathcal{L}_Q$, master data D_m, and a set V of CCs in $\mathcal{L}_C$.
QUESTION:	Is $\mathsf{RCQ}(Q, D_m, V)$ nonempty? That is, does there exist a database that is complete for Q relative to (D_m, V)?

Intuitively, RCDP is to decide whether a database has complete information to answer a query, and RCQP is to decide whether there exists a database complete for a query.

Query languages. We consider $\mathcal{L}_Q$ and $\mathcal{L}_C$ ranging over:

(a) conjunctive queries (CQ), built up from atomic formulas with constants and variables, *i.e.,* relation atoms in database schema $\mathcal{R}$, equality ($=$) and inequality ($\neq$), by closing under conjunction $\wedge$ and existential quantification $\exists$;

(b) union of conjunctive queries (UCQ) of the form $Q_1 \cup \cdots \cup Q_k$, where for each $i \in [1, k]$, Q_i is in CQ;

(c) positive existential FO queries ($\exists$FO$^+$), built from atomic formulas by closing under $\wedge$, *disjunction* $\vee$ and $\exists$;

(d) first-order logic queries (FO) built from atomic formulas using $\wedge$, $\vee$, *negation* $\neg$, $\exists$ and universal quantification $\forall$; and

(f) datalog queries (FP), defined as a collection of rules $p(\bar{x}) \leftarrow p_1(\bar{x}_1), \ldots, p_n(\bar{x}_n)$, where each p_i is either an atomic formula (a relation atom in $\mathcal{R}$, $=$, $\neq$) or an IDB predicate. In other words, FP is an extension of $\exists$FO$^+$with an inflational fixpoint operator. We refer the reader to, *e.g.,* [2], about the details of these languages.

2.2 Relative Completeness and Consistency

Real life data often contains errors, inconsistencies and conflicts [4]. To capture inconsistencies in the data, it is typical to use integrity constraints. That is, a set Σ of integrity constraints is imposed on a database D such that errors in D are detected as violations of one or more constraints in Σ.

Several classes of integrity constraints have been proposed for capturing inconsistencies in relational data (see, *e.g.,* [8, 13] for recent surveys). Below we review three classes recently studied for the consistency of data.

(a) Denial constraints [3, 8] are universally quantified FO sentences of the form:

$$\forall \bar{x}_1 \ldots \bar{x}_m\ \neg\big(R_1(\bar{x}_1) \wedge \cdots \wedge R_m(\bar{x}_m) \wedge \varphi(\bar{x}_1, \ldots, \bar{x}_m)\big),$$

where R_i is a relation atom for $i \in [1, m]$, and φ is a conjunction of built-in predicates $=$ and $\neq$.

(b) Conditional functional dependencies (CFDs) [14] are an extension of functional dependencies (FDs) of the form:

$$\forall \bar{x}_1 \bar{x}_2\ \bar{y}_1 \bar{y}_2\ \bar{z}_1 \bar{z}_2\ \big(R(\bar{x}_1, \bar{z}_1, \bar{y}_1) \wedge R(\bar{x}_2, \bar{z}_2, \bar{y}_2) \wedge \phi(\bar{x}_1) \wedge \phi(\bar{x}_2) \wedge \bar{x}_1 = \bar{x}_2 \rightarrow \bar{y}_1 = \bar{y}_2 \wedge \psi(\bar{y}_1) \wedge \psi(\bar{y}_2)\big)$$

where R is a relation atom, $\phi(\bar{x})$ is a conjunction of the form $x_{i_1} = c_1 \wedge \cdots \wedge x_{i_k} = c_k$; here $\{x_{i_j} \mid j \in [1, k]\}$ is a subset of $\bar{x}$, and c_j is a constant; $\psi(\bar{y})$ is defined similarly.

Intuitively, a CFD extends a traditional FD $X \rightarrow Y$ by incorporating patterns of semantically related values, where X and Y are the attributes denoted by $\bar{x}$ and $\bar{y}$ in $R(\bar{x}, \bar{z}, \bar{y})$ above, respectively. That is, for any R tuples t_1 and t_2, if $t_1[X] = t_2[X]$ and in addition, $t_1[X]$ and $t_2[X]$ have a certain constant pattern specified by $\phi(\bar{x})$, then $t_1[Y] = t_2[Y]$ and moreover, $t_1[Y]$ and $t_2[Y]$ have the constant pattern specified by $\psi(\bar{y})$. Observe that in the absence of $\phi(\bar{x})$ and $\psi(\bar{y})$, the CFD is a traditional FD.

As an example, recall from Example 1.1 that for any tuple (e, p, c) in Work_on, if project p is p_0, then the client c is AL. This can be expressed as a CFD that extends the FD proj $\rightarrow$ cli, such that if proj $= p_0$ then cli $=$ AL.

(c) Conditional inclusion dependencies (CINDs) [5] are an extension of inclusion dependencies (INDs) of the form:

$$\forall \bar{x}\ \bar{y}_1 \bar{z}_1\ \big(R_1(\bar{x}, \bar{y}_1, \bar{z}_1) \wedge \phi(\bar{y}_1) \rightarrow \exists \bar{y}_2 \bar{z}_2\ (R_2(\bar{x}, \bar{y}_2, \bar{z}_2) \wedge \psi(\bar{y}_2))\big)$$

where R_1, R_2 are relation atoms, $\phi(\bar{z})$ and $\psi(\bar{z})$ are defined as above. Similarly to CFDs, a CIND extends a traditional IND $R_1[X] \subseteq R_2[Y]$ by incorporating constant patterns specified by $\phi(\bar{y}_1)$ and $\psi(\bar{y}_2)$, for constraining R_1 tuples and R_2 tuples, respectively. Note that traditional INDs are a special case of CINDs, in the absence of $\phi(\bar{y}_1)$ and $\psi(\bar{y}_2)$.

It should be remarked that integrity constraints are posed on databases D regardless of master data D_m. In contrast, containment constraints are defined on (D, D_m).

From the proposition below it follows that by using containment constraints we can enforce both the relative completeness and the consistency of the data.

Proposition 2.1: (a) Denial constraints and CFDs can be expressed as containment constraints in CQ. (b) CINDs can be expressed as containment constraints in FO. □

PROOF. The proofs for denial constraints and CINDs are straightforward. Below we give the proof for CFDs. A CFD is equivalent to two sets of CCs in CQ. For each pair (y_1, y_2) of variables in $(\bar{y}_1, \bar{y}_2)$, the first set contains a CC: $q \subseteq \emptyset$, where $q(\bar{x}_1, \bar{z}_1, \bar{y}_1, \bar{x}_2, \bar{z}_2, \bar{y}_2)$ is

$$R(\bar{x}_1, \bar{z}_1, \bar{y}_1) \wedge R(\bar{x}_2, \bar{z}_2, \bar{y}_2) \wedge \phi(\bar{x}_1) \wedge \phi(\bar{x}_2) \wedge \bar{x}_1 = \bar{x}_2 \wedge y_1 \neq y_2,$$

assuring that the CFD is not violated by two distinct tuples.

The second set contains a CC of the form $q' \subseteq \emptyset$ for each variable y in $\bar{y}$ such that $y = c$ is in $\psi(\bar{y})$, where q' is

$$q'(\bar{x}, \bar{z}, \bar{y}) = R(\bar{x}, \bar{z}, \bar{y}) \wedge \phi(\bar{x}) \wedge y \neq c.$$

These CCs ensure that the CFD is not violated by a single tuple that does not observe the constant patterns. □

3. Deciding Relatively Complete Databases

In this section we focus on $\mathsf{RCDP}(\mathcal{L}_Q, \mathcal{L}_C)$, the relatively complete database problem. Given a query Q in $\mathcal{L}_Q$, master data D_m, a set V of containment constraints (CCs) in $\mathcal{L}_C$, and a partially closed database D *w.r.t.* (D_m, V), we want to determine whether or not D has complete information to answer Q, *i.e.,* whether or not $D \in \mathsf{RCQ}(Q, D_m, V)$.

We first establish matching lower and upper bounds for $\mathsf{RCDP}(\mathcal{L}_Q, \mathcal{L}_C)$ for all the query languages $\mathcal{L}_Q$ and $\mathcal{L}_C$ described in Section 2, as well as for a special case where CCs are INDs. We then present characterizations of databases in $\mathsf{RCQ}(Q, D_m, V)$ when CCs and queries are in CQ, to provide insight into what makes a database relatively complete. The characterizations readily extend to the settings where CCs are INDs or CQ, and queries are in CQ or UCQ.

3.1 Complexity Bounds for RCDP

We start with a negative result: when either $\mathcal{L}_Q$ or $\mathcal{L}_C$ is FO or FP, it is infeasible to determine whether a database D is relatively complete for a query Q *w.r.t.* (D_m, V). This tells us that both $\mathcal{L}_Q$ and $\mathcal{L}_C$ may impact the complexity of $\mathsf{RCDP}(\mathcal{L}_Q, \mathcal{L}_C)$. Worse still, the undecidability remains intact even when D_m and V are predefined and fixed.

Theorem 3.1: $\mathsf{RCDP}(\mathcal{L}_Q, \mathcal{L}_C)$ is undecidable when

1. $\mathcal{L}_Q$ is FO and $\mathcal{L}_C$ is CQ,
2. $\mathcal{L}_C$ is FO and $\mathcal{L}_Q$ is CQ,
3. $\mathcal{L}_Q$ is FP and $\mathcal{L}_C$ is CQ, or
4. $\mathcal{L}_C$ is FP and $\mathcal{L}_Q$ consists of a fixed query in FP.

If $\mathcal{L}_Q$ is FO or FP, the problem remains undecidable for fixed master data and fixed containment constraints. □

PROOF. (1) When $\mathcal{L}_Q$ is FO, the undecidability is proved by reduction from the satisfiability problem for FO queries, which is undecidable (cf. [2]). In the reduction the master data D_m and the set V of CCs are both fixed: D_m is an empty relation, and V is an empty set.

(2) When $\mathcal{L}_C$ is FO, it is also verified by reduction from the satisfiability problem for FO queries.

(3) When $\mathcal{L}_Q$ is FP, the proof is more involved. It is verified by reduction from the emptiness problem for deterministic finite 2-head automata (2-head DFA), which is undecidable [26]. Given a 2-head DFA $\mathcal{A}$, we use a database to encode a string, positions in the string and a successor function. A fixed set of CCs in CQ is used to ensure that these are well-formed. A query in FP is used to encode the non-emptiness of $\mathcal{A}$. The reduction only needs fixed D_m.

(4) When $\mathcal{L}_C$ is FP, the proof is again by reduction from the emptiness problem for 2-head DFA. In the reduction, the query Q is a fixed FP query, and V is a set of CCs in FP, which are not fixed.

When $\mathcal{L}_Q$ is FO or FP, it remains undecidable if D_m and V are fixed, since those are what the proofs use. □

In light of the undecidability results, below we focus on query languages that support neither negation nor recursion. We show that the absence of negation and recursion makes our lives easier: $\mathsf{RCDP}(\mathcal{L}_Q, \mathcal{L}_C)$ is in the polynomial hierarchy when $\mathcal{L}_Q$ and $\mathcal{L}_C$ are CQ, UCQ or $\exists$FO$^+$.

To simplify the discussion we assume in the rest of the section that $\mathcal{L}_Q$ and $\mathcal{L}_C$ are the same language, unless explicitly stated otherwise. This does not lose generality: if users are allowed to define CCs in a query language, there is no reason for not allowing them to issue queries in the same language. In addition, the proofs for the results below actually show that the complexity bounds remain unchanged when $\mathcal{L}_Q$ is CQ and $\mathcal{L}_C$ is $\exists$FO$^+$, or the other way around.

Nevertheless, we also consider a special case where CCs are INDs, *i.e.,* CCs of the form $q_v \subseteq p$ when both q_v and p are

projection queries, on D and D_m, respectively. We consider V consisting of INDs, as commonly found in practice, while the user queries are expressed in CQ, UCQ or $\exists$FO$^+$.

The next results tell us that the complexity bounds are rather robust: the problem is Π_2^p-complete when $\mathcal{L}_Q$ and $\mathcal{L}_C$ are $\exists$FO$^+$, and it remains Π_2^p-complete when $\mathcal{L}_Q$ is CQ and $\mathcal{L}_C$ is the class of INDs.

Theorem 3.2: $\mathsf{RCDP}(\mathcal{L}_Q, \mathcal{L}_C)$ is Π_2^p-complete when

1. $\mathcal{L}_C$ is the class of INDs and $\mathcal{L}_Q$ is CQ, UCQ or $\exists$FO$^+$,
2. $\mathcal{L}_Q$ and $\mathcal{L}_C$ are CQ,
3. $\mathcal{L}_Q$ and $\mathcal{L}_C$ are UCQ, or
4. $\mathcal{L}_Q$ and $\mathcal{L}_C$ are $\exists$FO$^+$. □

Proof. It suffices to show (1) and (2) below.

(1) $\mathsf{RCDP}(\mathcal{L}_Q, \mathcal{L}_C)$ is Π_2^p-hard when $\mathcal{L}_C$ is the class of INDs and $\mathcal{L}_Q$ is CQ. The lower bound is verified by reduction from $\forall^*\exists^*$3SAT-problem, which is Π_2^p-complete (cf. [23]). Given a $\forall^*\exists^*$3SAT sentence φ, we construct master data D_m consisting of six fixed relations, and a set V of CCs consisting of six fixed INDs, to encode "disjunction" and "negation" of propositional variables, and to ensure that truth assignments to variables in φ are valid. We then define a query Q in CQ and a database D to encode φ, such that φ is satisfiable iff D is complete for Q relative to (D_m, V).

(2) $\mathsf{RCDP}(\mathcal{L}_Q, \mathcal{L}_C)$ is in Π_2^p when $\mathcal{L}_C$ and $\mathcal{L}_Q$ are both $\exists$FO$^+$. The upper bound is shown by providing an algorithm that checks whether a database D is *not* complete for a given Q *w.r.t.* (D_m, V), by using a non-deterministic PTIME Turing machine with an NP oracle. This suffices. Indeed, since the algorithm is in Σ_2^p, the problem for deciding whether $D \in \mathsf{RCQ}(Q, D_m, V)$ is in Π_2^p.

The algorithm is based on a small model property: if D is not complete for Q, then there exists a set Δ of tuples such that (a) $D' = D \cup \Delta$ is partially closed *w.r.t.* (D_m, V), (b) there exists a tuple s such that $s \in Q(D')$ but $s \notin Q(D)$, by the monotonicity of $\exists$FO$^+$queries; (c) the size of Δ is bounded by the size $|Q|$ of Q; and (d) the active domain of Δ consists of the values in D and D_m, constants in Q or V, as well as a set New of distinct values such that its cardinality is bounded by $|Q|$ and V. Capitalizing on this property, the algorithm first guesses a set Δ of no more than $|Q|$ tuples, and then checks whether $(D', D_m) \models V$ and $Q(D) \neq Q(D')$ by using an NP oracle, where $D' = D \cup \Delta$. □

In practice, master data D_m and containment constraints V are often predefined and fixed, and only databases and user queries vary. One might be tempted to think that fixed D_m and V would lower the complexity bounds.

Unfortunately, the next result tells us that the lower bound of Theorem 3.2 remains unchanged when D_m and V are fixed, even when V is a fixed set of INDs.

Corollary 3.3: $\mathsf{RCDP}(\mathcal{L}_Q, \mathcal{L}_C)$ remains Π_2^p-complete when master data D_m and the set V of containment constraints are fixed, and when (a) $\mathcal{L}_C$ is the class of INDs and $\mathcal{L}_Q$ is CQ, UCQ or $\exists$FO$^+$; (b) $\mathcal{L}_Q$ and $\mathcal{L}_C$ are CQ; (c) $\mathcal{L}_Q$ and $\mathcal{L}_C$ are UCQ; or (d) $\mathcal{L}_Q$ and $\mathcal{L}_C$ are $\exists$FO$^+$. □

Proof. The result follows immediately from the proof for Theorem 3.2. The upper bound of Theorem 3.2 carries over to fixed D_m and V. For the lower bound, the proof for Theorem 3.2 shows that the problem is Π_2^p-hard when V is a fixed set of INDs, and when Q is a CQ query. □

We have also seen from Theorem 3.1 that the problem remains undecidable for queries in FO or FP when D_m and V are fixed. Putting these together, we can conclude that fixed D_m and V do not lower the complexity of $\mathsf{RCDP}(\mathcal{L}_Q, \mathcal{L}_C)$. In contrast, as will be seen in the next section, fixed D_m and V simplify the analysis of $\mathsf{RCQP}(\mathcal{L}_Q, \mathcal{L}_C)$, the problem for deciding whether a query is relatively complete.

3.2 Characterizations of Complete Databases

We next provide characterizations of relatively complete databases D for a query Q for certain decidable cases of $\mathsf{RCDP}(\mathcal{L}_Q, \mathcal{L}_C)$. That is, we identify sufficient and necessary conditions for D to be included in $\mathsf{RCQ}(Q, D_m, V)$. These conditions provide a guidance for what data should be collected by D in order to correctly answer query Q.

We first present characterizations when $\mathcal{L}_Q$ and $\mathcal{L}_C$ are CQ. We then adjust the conditions to characterize relatively complete databases when $\mathcal{L}_Q$ is CQ and $\mathcal{L}_C$ is the class of INDs, and when $\mathcal{L}_Q$ and $\mathcal{L}_C$ are UCQ. Along the same lines conditions can be developed when $\mathcal{L}_Q$ and $\mathcal{L}_C$ are $\exists$FO$^+$, which are not included due to the lack of space.

When $\mathcal{L}_Q$ and $\mathcal{L}_C$ are CQ. Consider a CQ query Q, master database D_m, a set V of CCs in CQ, and a partially closed database D *w.r.t.* (D_m, V). Assume *w.l.o.g.* that Q is satisfiable, since otherwise D is trivially complete for Q *w.r.t.* any D_m and V as long as $(D, D_m) \models V$.

We start with basic notations to express the conditions.

Tableau queries and valuations. To simplify the discussion, in the sequel we focus on CQ queries over a single relation. It does not lose generality since the characterization results also hold for CQ queries over databases involving more than one relation. Indeed, the following can be easily verified. For any relational schema $\mathcal{R}$, there exist a single relation schema R, a linear-time computable function f_D from instances of $\mathcal{R}$ to instances of R, and a linear-time computable function f_Q: CQ $\to$ CQ such that for any instance D of $\mathcal{R}$ and any CQ query Q over $\mathcal{R}$, $Q(D) = f_Q(Q)(f_D(D))$.

In light of this, we represent the given CQ query Q as a tableau query (T_Q, u_Q), where T_Q denotes formulas in Q and u_Q is the output summary (see *e.g.,* [2] for details). For each variable x in Q, we use $\mathsf{eq}(x)$ to denote the set of variables y in Q such that $x = y$ is induced from equality in Q. In T_Q, we represent atomic formula $x = y$ by assigning the same distinct variable to $\mathsf{eq}(x)$, and $x =$ 'c' by substituting constant 'c' for each occurrence of y in $\mathsf{eq}(x)$. This is well defined when Q is satisfiable. Note that the size of T_Q and the number of variables in T_Q are bounded by the size of Q.

We denote by Adom the set consisting of (a) all constants that appear in D, D_m, Q or V, and (b) a set New of distinct values not in D, D_m, Q and V, one for each variable that is in either T_Q or in the tableau representations of the queries in V; when there are more variables with finite domain than values in $\mathbf{d}_f$ (recall $\mathbf{d}_f$ from Section 2), $\mathbf{d}_f \subseteq \mathsf{Adom}$.

For each variable y in T_Q, we define its active domain, denoted by $adom(y)$. If y appears in some column A in T_Q such that $dom(A)$ is finite $\mathbf{d}_f$, then $adom(y)$ is $\mathbf{d}_f \cap \mathsf{Adom}$. Otherwise $adom(y)$ is Adom.

A valuation μ for variables in T_Q is said to be *valid* if (a) for each variable y in T_Q, $\mu(y)$ is a value from $adom(y)$, and (b) $Q(\mu(T_Q))$ is nonempty, *i.e.,* μ observes inequality formulas $x \neq y$ and $x \neq$ 'b' specified in Q.

Characterizations. To illustrate the conditions, let us first examine some examples of relatively complete databases.

Example 3.1: Recall the CC ϕ_1 from Example 2.1, which enforces any database to contain no more than k tuples. Consider a database D_1 of k tuples. Then adding any new tuple to D_1 violates ϕ_1. Thus D_1 is in $\mathsf{RCQ}(Q, D_m, \{\phi_1\})$.

As another example, consider a schema $R(A, B, C)$, on which an FD $A \to B, C$ imposed. By Proposition 2.1, we can express the FD as two CCs in CQ, denoted by Σ_2, using D_m that has an empty relation. Consider a CQ query Q_3 to find all tuples t with $t[A] =$ 'a'. Let D_2 be an instance of R that contains a tuple $t = (a, b, c)$. Then $Q_3(D_2) = \{t\}$ and D_2 is in $\mathsf{RCQ}(Q_3, D_m, \Sigma_2)$. Indeed, adding tuples to D_2 either violates Σ_2 or does not change the answer to Q_3. □

These examples tell us that there are intriguing interactions of Q, V and the data already in D. While it is hard to characterize the interactions syntactically, we provide sufficient and necessary conditions for D to be in $\mathsf{RCQ}(Q, D_m, V)$. These conditions are expressed in terms of a notion of bounded databases given as follows.

A database D is said to be *bounded by* (D_m, V) for Q if for any valid valuation μ for variables in T_Q, either $(D \cup \mu(T_Q), D_m) \not\models V$ or $\mu(u_Q) \in Q(D)$. More specifically,

- when $Q(D) = \emptyset$, then $(D \cup \mu(T_Q), D_m) \not\models V$;
- when $Q(D) \neq \emptyset$, if $(D \cup \mu(T_Q), D_m) \models V$, then $\mu(u_Q) \in Q(D)$.

That is, for any tuples Δ, if $Q(D) \neq Q(D \cup \Delta)$, then $(D \cup \Delta, D_m) \not\models V$. In other words, adding tuples to D either violates V or does not change $Q(D)$. While there may exist infinitely many Δ's, it suffices to inspect Δ constructed with values in Adom only (the small model property).

Proposition 3.4: For any query Q in CQ, master data D_m, any set V of containment constraints in CQ, and any partially closed D *w.r.t.* (D_m, V), D is in $\mathsf{RCQ}(Q, D_m, V)$ iff D is bounded by (D_m, V) for Q. □

PROOF. Suppose that $Q(D) \neq \emptyset$. If D is relatively complete, then it is easy to see that the condition holds. Conversely, assume that D is not relatively complete but by contradiction, the condition holds. Then there must exist an extension D' of D and a valuation μ of T_Q drawing values from D', such that $(D \cup \mu(T_Q), D_m) \models V$ and $\mu(u_Q) \notin Q(D)$, by the monotonicity of CQ queries. Define a valuation μ' such that for each variable x in T_Q, $\mu'(x)$ is a distinct value in New if $\mu(x)$ is not in $\mathsf{Adom} \setminus \mathsf{New}$, and $\mu'(x) = \mu(x)$ otherwise. Then it is easy to verify that μ' is a valid validation, $(D \cup \mu'(T_Q), D_m) \models V$ but $\mu(u_Q) \notin Q(D)$, which contradicts the assumption that the condition holds.

The proof for $Q(D) = \emptyset$ is similar. □

When $\mathcal{L}_C$ is the class of INDs. If V is a set of INDs, the notion of bounded databases is simpler. More specifically, in this setting a database D is said to be *bounded by* (D_m, V) for Q if for each valid valuation μ of T_Q, either $(\mu(T_Q), D_m) \not\models V$ or $\mu(u_Q) \in Q(D)$.

When V is a set of INDs, Proposition 3.4 holds using the revised notion of bounded databases.

When $\mathcal{L}_Q$ and $\mathcal{L}_C$ are UCQ. Consider a query in UCQ: $Q = Q_1 \cup \cdots \cup Q_k$, where Q_i is in CQ for each $i \in [1, k]$. We represent Q_i as a tableau query (T_i, u_i). Then *a valuation* μ *for* Q is $(\mu_1, \ldots, \mu_k)$ such that for each $i \in [1, k]$, μ_i is a valuation for variables in T_i and moreover, for each variable y in T_i, $\mu_i(y) \in adom(y)$. The valuation is *valid* if there exists some $j \in [1, k]$ such that $Q_j(\mu_i(T_j))$ is nonempty.

A database D is said to be *bounded by* (D_m, V) for Q if for each valid valuation $\mu = (\mu_1, \ldots, \mu_k)$ for Q, either $(D \cup \Delta, D_m) \not\models V$, or for each $i \in [1, k]$, $\mu(u_i) \in Q(D)$, where Δ denotes $\mu_1(T_1) \cup \cdots \cup \mu_k(T_k)$.

One can easily verify that Proposition 3.4 remains intact using this revised notion when $\mathcal{L}_Q$ and $\mathcal{L}_C$ are UCQ.

4. Determining Relatively Complete Queries

We next investigate $\mathsf{RCQP}(\mathcal{L}_Q, \mathcal{L}_C)$, the relatively complete query problem. Given a query Q in $\mathcal{L}_Q$, master data D_m and a set V of CCs in $\mathcal{L}_C$, we want to decide whether there exists a database D that is complete for Q relative to (D_m, V), *i.e.,* whether $\mathsf{RCQ}(Q, D_m, V)$ is nonempty.

We first provide lower and upper bounds, all matching, for $\mathsf{RCQP}(\mathcal{L}_Q, \mathcal{L}_C)$, when $\mathcal{L}_Q$ and $\mathcal{L}_C$ range over the query languages given in Section 2, and for a special case where CCs are INDs. Compared to their $\mathsf{RCDP}(\mathcal{L}_Q, \mathcal{L}_C)$ counterparts (Section 3), the complexity bounds of $\mathsf{RCQP}(\mathcal{L}_Q, \mathcal{L}_C)$ are relatively more diverse; furthermore, fixed master data and containment constraints simplify the analysis, to some extent. We then characterize relatively complete queries in CQ or UCQ, when $\mathcal{L}_C$ ranges from INDs to UCQ.

4.1 Complexity Bounds for RCQP

Recall from Theorem 3.1 that it is undecidable to determine whether a database is in $\mathsf{RCQ}(Q, D_m, V)$ when either $\mathcal{L}_Q$ or $\mathcal{L}_C$ is FO or FP. It is no better for $\mathsf{RCQP}(\mathcal{L}_Q, \mathcal{L}_C)$: in these settings $\mathsf{RCQP}(\mathcal{L}_Q, \mathcal{L}_C)$ is also undecidable. Moreover, the undecidability is rather robust: the problem is already beyond reach in practice when master data and containment constraints are predefined and fixed.

Theorem 4.1: $\mathsf{RCQP}(\mathcal{L}_Q, \mathcal{L}_C)$ is undecidable when

1. $\mathcal{L}_Q$ is FO and $\mathcal{L}_C$ consists of fixed queries in FO,
2. $\mathcal{L}_C$ is FO and $\mathcal{L}_Q$ is CQ,
3. $\mathcal{L}_Q$ is FP and $\mathcal{L}_C$ consists of fixed queries in FP, or
4. $\mathcal{L}_C$ is FP and $\mathcal{L}_Q$ is CQ.

When $\mathcal{L}_Q$ is FO or FP, it remains undecidable for fixed master data and fixed containment constraints. □

PROOF. (1) When $\mathcal{L}_Q$ is FO, the proof is not as simple as one might have expected. The undecidability is verified by reduction from the emptiness problem for 2-head DFA. Given a 2-head DFA $\mathcal{A}$, we construct a database schema $\mathcal{R}$ consisting of several relations, which intend to encode a string, positions in the string, a successor relation, transitions of $\mathcal{A}$, as well as the transitive closure of the transitions. We define a set V of CCs using *fixed* FO queries, to help assure that instances of $\mathcal{R}$ encode a valid run of $\mathcal{A}$. We also define an FO query to inspect whether a run is valid and whether there is a valid run of $\mathcal{A}$ that accepts a string. Fixed master data D_m is used. Using these, we show that $\mathcal{A}$ accepts some strings iff $\mathsf{RCQ}(Q, D_m, V)$ is nonempty.

(2) When $\mathcal{L}_C$ is FO, the proof is easier. The undecidability is verified by reduction from the satisfiability problem for FO queries. The reduction uses a set V of CCs in FO.

(3) When $\mathcal{L}_Q$ is FP, the proof is by reduction from the emptiness problem for 2-head DFA. In contrast to (1), recursion in FP is used to verify the existence of a valid run of a given 2-head DFA. The reduction uses a fixed set V of CCs in FP, and a query Q in FP. The master data D_m is also fixed.

(4) When $\mathcal{L}_C$ is FP, the proof is similar to (3), but the

reduction uses a fixed query Q in CQ and a set V of CCs in FP instead, where the CCs are not fixed, *i.e.,* they are defined based on the instance of 2-head DFA.

The proofs only use fixed (D_m, V) if $\mathcal{L}_Q$ is FO or FP, and thus verify the undecidability for fixed D_m and V. □

We have seen from Theorem 3.2 that the absence of negation and recursion in $\mathcal{L}_Q$ and $\mathcal{L}_C$ simplifies the analysis of $\mathsf{RCDP}(\mathcal{L}_Q, \mathcal{L}_C)$. Below we show that this is also the case for $\mathsf{RCQP}(\mathcal{L}_Q, \mathcal{L}_C)$, which is settled in positive in these settings. In contrast to Theorem 3.2, the complexity bounds for $\mathsf{RCQP}(\mathcal{L}_Q, \mathcal{L}_C)$ are no longer the same when $\mathcal{L}_C$ is $\exists$FO$^+$and when $\mathcal{L}_C$ is the class of INDs. In addition, when $\mathcal{L}_Q$ and $\mathcal{L}_C$ are CQ, $\mathsf{RCQP}(\mathcal{L}_Q, \mathcal{L}_C)$ becomes NEXPTIME-complete, *i.e.,* the analysis is harder than its $\mathsf{RCDP}(\mathcal{L}_Q, \mathcal{L}_C)$ counterpart. On the other hand, when $\mathcal{L}_Q$ is the class of INDs, the complexity is down to coNP-complete, better than its Π_2^p-complete counterpart for $\mathsf{RCDP}(\mathcal{L}_Q, \mathcal{L}_C)$.

Theorem 4.2: $\mathsf{RCQP}(\mathcal{L}_Q, \mathcal{L}_C)$ is

1. coNP-complete when $\mathcal{L}_C$ is the class of INDs and $\mathcal{L}_Q$ is CQ, UCQ or $\exists$FO$^+$; and
2. NEXPTIME-complete when
   (a) $\mathcal{L}_Q$ and $\mathcal{L}_C$ are CQ,
   (b) $\mathcal{L}_Q$ and $\mathcal{L}_C$ are UCQ, or
   (c) $\mathcal{L}_Q$ and $\mathcal{L}_C$ are $\exists$FO$^+$. □

PROOF. (1) *Lower bounds.* The coNP lower bound is verified by reduction from 3SAT to the complement of RCQP(CQ, INDs). Given a 3SAT instance ϕ, we define fixed master data D_m, a set V of fixed INDs and a CQ query Q such that ϕ is satisfiable iff $\mathsf{RCQ}(Q, D_m, V)$ is empty.

When $\mathcal{L}_C$ and $\mathcal{L}_Q$ are CQ, we verify the NEXPTIME lower bound by reduction from the tiling problem for $2^n \times 2^n$ square, which is NEXPTIME-complete (see, *e.g.,* [9]). Given an instance of the tiling problem, we construct a database schema $\mathcal{R}$ consisting relations $R_1, \ldots, R_n$ such that R_i encodes a $2^i \times 2^i$ square of tiles for $i \in [1, n]$. We define master data D_m and a set V of CCs in CQ to assure the vertical and horizontal compatibility of the tiling. Finally we define a CQ query Q such that $\mathsf{RCQ}(Q, D_m, V)$ is nonempty iff there exists a valid tiling of the given instance.

(2) *Upper bounds.* We show the upper bounds by giving constructive proofs. We develop sufficient and necessary conditions for $\mathsf{RCQ}(Q, D_m, V)$ to be nonempty.

Given master data D_m, a set V of INDs and a query Q in $\exists$FO$^+$, we show that it suffices to check whether for any set Δ of tuples, either $(\Delta, D_m) \not\models V$ or certain syntactic conditions on Q and V hold (see Section 4.2 for details). Better still, it suffices to inspect "bounded" Δ: its size is no larger than that of Q, and its active domain is determined by D_m, Q and V. Based on this, an NP algorithm is developed to check whether $\mathsf{RCQ}(Q, D_m, V)$ is empty. It guesses a set Δ of tuples, and then checks the conditions in PTIME.

When $\mathcal{L}_Q$ and $\mathcal{L}_C$ are $\exists$FO$^+$, we prove a small model property: given D_m, a query Q in $\exists$FO$^+$and a set V of CCs in $\exists$FO$^+$, $\mathsf{RCQ}(Q, D_m, V)$ is nonempty iff there exists a partially closed database D such that (a) the size of D is bounded by an exponential in the sizes of Q, V and D_m; (b) for any partial extension $D' = D \cup \Delta$ of D, either $Q(D') = Q(D)$ or $(D', D_m) \not\models V$; and furthermore, (c) it suffices to inspect increments Δ that are bounded as described above. The small model property is established by leveraging the monotonicity of $\exists$FO$^+$queries. Based on this property, an NEXPTIME algorithm can be readily developed for checking whether $\mathsf{RCQ}(Q, D_m, V)$ is nonempty. □

As remarked earlier, master data D_m and containment constraints V are often predefined and fixed in practice. Recall from Corollary 3.3 that fixed D_m and V have no impact on the complexity of $\mathsf{RCDP}(\mathcal{L}_Q, \mathcal{L}_C)$. In contrast, we show below that fixed D_m and V do make our lives easier, to some extent. (a) When $\mathcal{L}_Q$ and $\mathcal{L}_C$ are CQ, UCQ or $\exists$FO$^+$, $\mathsf{RCQP}(\mathcal{L}_Q, \mathcal{L}_C)$ becomes Σ_3^p-complete, down from NEXPTIME-complete. (b) On the other hand, when $\mathcal{L}_Q$ is CQ and $\mathcal{L}_C$ is the class of INDs, the problem remains coNP-complete. (c) Fixed D_m and V do not help when either $\mathcal{L}_Q$ or $\mathcal{L}_C$ is FO or FP, as we have seen in Theorem 4.1.

Corollary 4.3: When master data and containment constraints are fixed, $\mathsf{RCQP}(\mathcal{L}_Q, \mathcal{L}_C)$ is

1. coNP-complete $\mathcal{L}_C$ is the class of INDs and $\mathcal{L}_Q$ is CQ, UCQ or $\exists$FO$^+$, and
2. Σ_3^p-complete if $\mathcal{L}_Q$ and $\mathcal{L}_C$ are CQ, UCQ or $\exists$FO$^+$. □

PROOF. It suffices to show the following.

(1) $\mathsf{RCQP}(\mathcal{L}_Q, \mathcal{L}_C)$ is coNP-hard when $\mathcal{L}_Q$ is CQ and $\mathcal{L}_C$ is the class of INDs. This follows from the proof for the coNP lower bound of Theorem 4.2 (1), in which only fixed master data D_m and a set V of fixed INDs are used. The coNP upper bound carries over when D_m and V are fixed.

(2) We show that $\mathsf{RCQP}(\mathcal{L}_Q, \mathcal{L}_C)$ is Σ_3^p-hard when $\mathcal{L}_Q$ and $\mathcal{L}_C$ are CQ, by reduction from $\exists^*\forall^*\exists^*$3SAT-problem, which is Σ_3^p-complete (cf. [23]). Given an instance φ of the latter problem, we construct fixed master data D_m, a set V of fixed CCs in CQ, and a CQ query Q to encode φ and to ensure that truth assignments to variables in φ are well defined. We show φ is satisfiable iff $\mathsf{RCQ}(Q, D_m, V)$ is nonempty.

Recall the proof for $\mathsf{RCQP}(\exists\text{FO}^+, \exists\text{FO}^+)$ given for Theorem 4.2 (2). When V and D_m are fixed, it suffices to inspect "small models" D of a polynomial size. Based on this, we develop an algorithm to check whether $\mathsf{RCQ}(Q, D_m, V)$ is nonempty. The algorithm uses a non-deterministic PTIME Turing machine with a Π_2^p oracle to inspect whether the conditions given in the proof of Theorem 4.2 (2) hold. □

4.2 Characterizations of Complete Queries

We next characterize relatively complete queries for certain decidable cases of $\mathsf{RCQP}(\mathcal{L}_Q, \mathcal{L}_C)$. Given master data D_m, a set V of CCs and a query Q, we provide sufficient and necessary conditions for $\mathsf{RCQ}(Q, D_m, V)$ to be nonempty.

We first present conditions for Q and V in CQ. We then extend the conditions to characterize relatively complete Q when $\mathcal{L}_C$ consists of INDs, and when $\mathcal{L}_Q$ and $\mathcal{L}_C$ are UCQ. Similar conditions can also be developed when $\mathcal{L}_Q$ and $\mathcal{L}_C$ are $\exists$FO$^+$, which are not included due to the lack of space.

When $\mathcal{L}_Q$ and $\mathcal{L}_C$ are CQ. To get insight into the conditions, let us first look at some example complete queries.

Example 4.1: Consider a schema $R(A, B, C)$ on which an FD $A \to C$ imposed. The FD can be expressed as a CC ϕ_3 in CQ. Consider a query Q_4 in CQ that is to find all tuples t such that $t[A] =$ 'a' and $t[C] =$ 'c'. Let master data D_m be an empty relation. Then Q_4 is relatively complete. Indeed, a database D^- is complete for Q_4 relative to D_m and ϕ_3, where D^- consists of a single tuple $t_- = (a, b, d)$, $c \neq d$. Note that $Q_4(D^-) = \emptyset$, and D^- prevents any tuples Δ from being added to it as long as $Q_4(\Delta)$ is nonempty.

Now consider query Q_3 of Example 3.1, which is to find all tuples t with $t[A] =$ 'a'. Assume that $dom(B)$ is infinite. Then Q_3 is *not* complete *w.r.t.* D_m and ϕ_3. Indeed, no matter what database D we consider, we can always add a new tuple t' to D such that $t'[B]$ is a value not in D, and $Q_3(D) \neq Q_3(D \cup \{t'\})$. Contrast this with Example 3.1, which shows that if the FD is $A \to B, C$ (expressed as CCs Σ_2), then Q_3 is relatively complete. As we have seen there, a database complete for Q_3 is D^+, which consists of $t_+ = (a, b, c)$. Indeed, for any tuples Δ, if $Q_3(\Delta) \neq \emptyset$ and $(D^+ \cup \Delta, D_m) \models \Sigma_2$, then for any t' in Δ, D^+ enforces $t'[B]$ to take 'b' as its value. That is, the values of $t'[B]$ are bounded. □

As suggested by the example, a query Q is relatively complete iff one of the following two conditions holds. (a) There exists a set D^- of tuples such that $(D^-, D_m) \models V$, $Q(D^-) = \emptyset$ and moreover, D^- prevents those tuples Δ from being added to D^- if $Q(\Delta)$ is nonempty. That is, there exist no tuples Δ such that both $(D^- \cup \Delta, D_m) \models V$ and $Q(\Delta) \neq \emptyset$. (b) There exists a set D^+ of tuples such that $Q(D^+) \neq \emptyset$, $(D^+, D_m) \models V$, and moreover, D^+ "bounds" all those variables y in Q with an infinite domain, via D_m and V. That is, for any such y and any tuples Δ, if $Q(\Delta)$ is nonempty, then either $(D^+ \cup \Delta, D_m) \not\models V$ or $Q(\Delta) \subseteq Q(D^+)$, where y is instantiated to a value in D_m.

To formalize the intuition, we use the following notations.

(a) We revise the notion of Adom introduced in Section 3.2, such that it consists of constants in D_m, V, Q or New. Along the same lines as Section 3.2, we represent CQ query Q as a tableau query (T_Q, u_Q), and define valid valuations of T_Q.

For each variable y in T_Q, we say that the domain of y, denoted by $dom(y)$, is *finite* if y appears in some column A in T_Q such that $dom(A)$ is $\mathbf{d}_f$, and it is *infinite* otherwise.

(b) Assume that V consists of $q_i \subseteq p_i$ for $i \in [1, n]$, where q_i is a CQ query. We represent q_i as a tableau query (T_i, u_i). A *valuation* ν *of* V is $(\nu_1, \ldots, \nu_n)$ such that ν_i is a valuation for variables in T_i with values in Adom. We use D_ν to denote $\bigcup_{i\in[1,n]} \nu_i(T_i)$. For a set $\mathcal{V}$ of valuations of V, we use $D_\mathcal{V}$ to denote $\bigcup_{\nu\in\mathcal{V}} D_\nu$. In particular, when $\mathcal{V}$ is empty, so is $D_\mathcal{V}$.

Proposition 4.4: For any CQ query $Q = (T_Q, u_Q)$, master data D_m, and any set V of CCs in CQ, $\mathsf{RCQ}(Q, D_m, V)$ is nonempty iff either all variables in u_Q have a finite domain, or there exists a set $\mathcal{V}$ of valuations of V such that (a) $(D_\mathcal{V}, D_m) \models V$, and (b) for any valid valuation μ of T_Q, either $(D_\mathcal{V} \cup \mu(T_Q), D_m) \not\models V$ or $\mu(u_Q) \in Q(D_\mathcal{V})$. □

Proof. When all variables in u_Q have a finite domain, Q is trivially relatively complete *w.r.t.* (D_m, V). Below we assume that for some y in u_Q, $dom(y)$ is infinite.

Suppose that there exists a set $\mathcal{V}$ of valuations of V satisfying the conditions given in the proposition. Then one can verify the following. (a) If $Q(D_\mathcal{V}) = \emptyset$, then $D_\mathcal{V}$ is a set D^- of tuples with the properties mentioned above. (b) If $Q(D_\mathcal{V}) \neq \emptyset$, then there must exist a set D^+ of tuples as mentioned above. In both cases, $\mathsf{RCQ}(Q, D_m, V)$ is nonempty.

Conversely, suppose that $\mathsf{RCQ}(Q, D_m, V)$ contains a database D. We show the following. (a) If $Q(D) = \emptyset$, then there is a set $\mathcal{V}$ of valuations of V such that $(D_\mathcal{V}, D_m) \models V$ and for any valid valuation μ of T_Q, $(D_\mathcal{V} \cup \mu(T_Q), D_m) \not\models V$. (b) If $Q(D) \neq \emptyset$, then there exists a set $\mathcal{V}$ of valuations of V such that $(D_\mathcal{V}, D_m) \models V$, and for any valid valuation μ of T_Q, if $(D_\mathcal{V} \cup \mu(T_Q), D_m) \not\models V$ then $\mu(T_Q) \in Q(D_\mathcal{V})$. □

$\mathsf{RCDP}(\mathcal{L}_Q, \mathcal{L}_C)$	Complexity
(FO, CQ) (Th. 3.1(1))	undecidable
(CQ, FO) (Th. 3.1(2))	undecidable
(FP, CQ) (Th. 3.1(3))	undecidable
(fixed(FP), FP) (Th. 3.1(4))	undecidable
(CQ, INDs), ($\exists$FO$^+$, INDs) (Th. 3.2(1))	Π_2^p-complete
(CQ, CQ) (Th. 3.2(2))	Π_2^p-complete
(UCQ, UCQ) (Th. 3.2(3))	Π_2^p-complete
($\exists$FO$^+$, $\exists$FO$^+$) (Th. 3.2(4))	Π_2^p-complete

Table 1: Complexity of $\mathsf{RCDP}(\mathcal{L}_Q, \mathcal{L}_C)$

The conditions can be equivalently expressed as follows. For a database D and a variable y in u_Q, let $\mathsf{val}(D, y)$ denote the set of $\mu(y)$'s when μ ranges over all valuations of T_Q such that $Q(\mu(T_Q)) \neq \emptyset$ and $(D \cup \mu(T_Q), D_m) \models V$ (μ may draw values beyond Adom). Then $\mathsf{RCQ}(Q, D_m, V)$ is nonempty iff there is a set $\mathcal{V}$ of valuations of V such that $(D_\mathcal{V}, D_m) \models V$ and $\mathsf{val}(D_\mathcal{V}, y)$ is *finite* for all variables y in u_Q.

Observe that the size of any set $\mathcal{V}$ of valuations of V is at most exponential in the sizes of Q, V and D_m, and that each valid valuation μ of T_Q is no larger than Q. These yield the small model property used in the proof of Theorem 4.2.

When $\mathcal{L}_C$ is the class of INDs. In this setting, there is a syntactic characterization for relatively complete queries, which leads to the coNP upper bound of Theorem 4.2(1).

We say that (D_m, V) *bounds* a CQ query $Q = (T_Q, u_Q)$ if for all variables y in u_Q, either (a) $dom(y)$ is finite, or (b) there exists an IND $\pi_{(A,\ldots)} \subseteq p$ in V such that y appears in column A in T_Q, where π is the projection operator.

Proposition 4.5: For any CQ query $Q = (T_Q, u_Q)$, master data D_m, and any set V of INDs, $\mathsf{RCQ}(Q, D_m, V)$ is nonempty iff (D_m, V) bounds Q as long as there exists a valid valuation μ of T_Q such that $(\mu(T_Q), D_m) \models V$. □

Proof. If for all valuations μ of T_Q, $(\mu(T_Q), D_m) \not\models V$, then the empty database D is in $\mathsf{RCQ}(Q, D_m, V)$.

Suppose that there exists a valid valuation μ of T_Q such that $(\mu(T_Q), D_m) \models V$. If (D_m, V) bounds Q, then one can construct a complete database D^+ for Q relatively to (D_m, V). Conversely, if $\mathsf{RCQ}(Q, D_m, V)$ is nonempty, then one can prove by contradiction that (D_m, V) bounds Q. □

When $\mathcal{L}_Q$ and $\mathcal{L}_C$ are UCQ. A containment constraint in UCQ is of the form $(q_1 \cup \cdots \cup q_m) \subseteq p$, and is equivalent to a set of CCs in CQ, consisting of $q_j \subseteq p$ for each $j \in [1, m]$. Thus the notions of valuations of V and $D_\mathcal{V}$ for a set $\mathcal{V}$ of valuations of V are also well defined in this setting.

Consider a query $Q = Q_1 \cup \cdots \cup Q_k$ in UCQ, where Q_i is represented as a tableau query (T_i, u_i). Recall the notion of valid valuations of Q from Section 3.2.

Then Proposition 4.4 remains intact: $\mathsf{RCQ}(Q, D_m, V)$ is nonempty iff for all $i \in [1, n]$, either all variables in u_i have a finite domain, or there is a set $\mathcal{V}$ of valuations of V such that $(D_\mathcal{V}, D_m) \models V$, and for any valid valuation μ of Q, either $\mu(u_i) \in Q(D_\mathcal{V})$ or $(D_\mathcal{V} \cup \bigcup_{i\in[1,n]} \mu(T_i), D_m) \not\models V$.

5. Conclusions

We have proposed the notion of the relative completeness of information to capture incomplete information in emerging applications such as Master Data Management. We have also introduced and studied two important decision problems associated with this notion, namely, $\mathsf{RCDP}(\mathcal{L}_Q, \mathcal{L}_C)$

$\mathsf{RCQP}(\mathcal{L}_Q, \mathcal{L}_C)$	Complexity
(FO, fixed(FO)) (Th. 4.1(1))	undecidable
(CQ, FO) (Th. 4.1(2))	undecidable
(FP, fixed(FP)) (Th. 4.1(3))	undecidable
(CQ, FP) (Th. 4.1(4))	undecidable
(CQ, INDs) (Th. 4.2(1))	coNP-complete
($\exists$FO$^+$, INDs) (Th. 4.2(1))	coNP-complete
(CQ, CQ) (Th. 4.2(2.a))	NEXPTIME-complete
(UCQ, UCQ) (Th. 4.2(2.b))	NEXPTIME-complete
($\exists$FO$^+$, $\exists$FO$^+$) (Th. 4.2(2.c))	NEXPTIME-complete
When D_m and V are fixed	
(CQ, CQ) (Cor. 4.3(2))	Σ_3^p-complete
(UCQ, UCQ) (Cor. 4.3(2))	Σ_3^p-complete
($\exists$FO$^+$, $\exists$FO$^+$) (Cor. 4.3(2))	Σ_3^p-complete

Table 2: Complexity of $\mathsf{RCQP}(\mathcal{L}_Q, \mathcal{L}_C)$

and $\mathsf{RCQP}(\mathcal{L}_Q, \mathcal{L}_C)$. For a variety of query languages for expressing queries ($\mathcal{L}_Q$) and containment constraints ($\mathcal{L}_C$), we have provided a comprehensive picture of lower and upper bounds for these problems, all matching. We have also presented sufficient and necessary conditions for a database or a query to be relatively complete *w.r.t.* master data and containment constraints, when $\mathsf{RCDP}(\mathcal{L}_Q, \mathcal{L}_C)$ and $\mathsf{RCQP}(\mathcal{L}_Q, \mathcal{L}_C)$ are decidable. We expect that these results will help users determine whether a database has complete information to answer a query, whether a query can find a complete answer at all, and what data should be collected by a database in order to yield a complete answer to a query.

We summarize the main results for $\mathsf{RCDP}(\mathcal{L}_Q, \mathcal{L}_C)$ and $\mathsf{RCQP}(\mathcal{L}_Q, \mathcal{L}_C)$ in Tables 1 and 2, respectively, annotated with their corresponding theorems, where $\mathsf{fixed}(\mathcal{L})$ indicates a set of fixed queries in $\mathcal{L}$. When master data and containment constraints are fixed, we only show complexity bounds that differ from their counterparts in the general settings.

The study of relatively complete information is still preliminary. An open issue is about how to incorporate missing values, together with missing tuples, into the framework. To this end, we expect to use presentation systems for possible worlds (*e.g.,* v-tables, c-tables [16, 17]) instead of traditional relations. Another open issue concerns clean *syntactic* characterizations for relatively complete databases or queries, in certain cases. A third interesting topic is to identify tractable (PTIME) special cases for $\mathsf{RCDP}(\mathcal{L}_Q, \mathcal{L}_C)$ and $\mathsf{RCQP}(\mathcal{L}_Q, \mathcal{L}_C)$. Finally, although the containment constraints proposed in this work are fairly general, in certain applications one might want to formulate containment constraints not only from databases to master data, but also *from* the master data *to* the databases. We defer the treatment of this richer class of constraints to future work.

Acknowledgements. Fan and Geerts are supported in part by EPSRC EP/E029213/1. Fan is a Yangtze River Scholar at Harbin Institute of Technology.

6. References

[1] S. Abiteboul and O. M. Duschka. Complexity of answering queries using materialized views. In *PODS*, 1998.

[2] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[3] M. Arenas, L. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. In *PODS*, 1999.

[4] C. Batini and M. Scannapieco. *Data Quality: Concepts, Methodologies and Techniques*. Springer, 2006.

[5] L. Bravo, W. Fan, and S. Ma. Extending dependencies with conditions. In *VLDB*, 2007.

[6] A. Cali, D. Lembo, and R. Rosati. On the decidability and complexity of query answering over inconsistent and incomplete databases. In *PODS*, 2003.

[7] D. Calvanese, G. D. Giacomo, M. Lenzerini, and M. Y. Vardi. View-based query processing: On the relationship between rewriting, answering and losslessness. *TCS*, 371(3), 2007.

[8] J. Chomicki. Consistent query answering: Five easy pieces. In *ICDT*, 2007.

[9] E. Dantsin and A. Voronkov. Complexity of query answering in logic databases with complex values. In *LFCS*, 2007.

[10] A. Deutsch, B. Ludäscher, and A. Nash. Rewriting queries using views with access patterns under integrity constraints. *TCS*, 371(3), 2007.

[11] A. Dreibelbis, E. Hechler, B. Mathews, M. Oberhofer, and G. Sauter. Master data management architecture patterns. IBM, 2007.

[12] C. Elkan. Independence of logic database queries and updates. In *PODS*, 1990.

[13] W. Fan. Dependencies revisited for improving data quality. In *PODS*, 2008.

[14] W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Conditional functional dependencies for capturing data inconsistencies. *TODS*, 33(1), 2008.

[15] G. Gottlob and R. Zicari. Closed world databases opened through null values. In *VLDB*, 1988.

[16] G. Grahne. *The Problem of Incomplete Information in Relational Databases*. Springer, 1991.

[17] T. Imieliński and W. Lipski, Jr. Incomplete information in relational databases. *JACM*, 31(4), 1984.

[18] A. Y. Levy. Obtaining complete answers from incomplete databases. In *VLDB*, 1996.

[19] A. Y. Levy and Y. Sagiv. Queries independent of updates. In *VLDB*, 1993.

[20] C. Li. Computing complete answers to queries in the presence of limited access patterns. *VLDB J.*, 12(3), 2003.

[21] D. Loshin. *Master Data Management*. Knowledge Integrity, Inc., 2009.

[22] A. Motro. Integrity = validity + completeness. *TODS*, 14(4), 1989.

[23] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[24] J. Radcliffe and A. White. Key issues for master data management. Gartner, 2008.

[25] L. Segoufin and V. Vianu. Views and queries: determinacy and rewriting. In *PODS*, 2005.

[26] M. Spielmann. *Abstract state machines: Verification problems and complexity*. PhD thesis, RWTH Aachen, 2000.

[27] R. van der Meyden. Logical approaches to incomplete information: A survey. In J. Chomicki and G. Saake, editors, *Logics for Databases and Information Systems*. Kluwer, 1998.

[28] M. Vardi. On the integrity of databases with incomplete information. In *PODS*, 1986.