Theoretical Computer Science 296 (2003) 405–434

Theoretical Computer Science

www.elsevier.com/locate/tcs

Scalar aggregation in inconsistent databases[☆]

Marcelo Arenas[a], Leopoldo Bertossi[b], Jan Chomicki[c,*], Xin He[c], Vijay Raghavan[d], Jeremy Spinrad[d]

[a] *Department of Computer Science, University of Toronto, Toronto, Ont., Canada*
[b] *School of Computer Science, Carleton University, Ottawa, Ont., Canada*
[c] *Department of Computer Science and Engineering, 201 Bell Hall, University at Buffalo, Buffalo, NY 14260-2000, USA*
[d] *EECS Department, Vanderbilt University, USA*

Abstract

We consider here scalar aggregation queries in databases that may violate a given set of functional dependencies. We define consistent answers to such queries to be greatest-lowest/least-upper bounds on the value of the scalar function across all (minimal) repairs of the database. We show how to compute such answers. We provide a complete characterization of the computational complexity of this problem. We also show how tractability can be improved in several special cases (one involves a novel application of Boyce–Codd Normal Form) and present a practical hybrid query evaluation method.

1. Introduction

In this paper, we address the issue of obtaining *consistent* information from *inconsistent* databases—databases that violate the given integrity constraints. Our basic assumption departs from everyday practice of database management systems. Typically, a database management system checks the satisfaction of integrity constraints and backs out those updates that violate them. Therefore, databases seemingly never become inconsistent. However, we list below several practical scenarios in which inconsistent databases do occur.

[☆] An earlier version of this paper appeared in [5].
[*] Corresponding author. Tel.: +716-645-3180x103; fax: +716-645-3464.
E-mail addresses: marenas@cs.toronto.edu (M. Arenas), bertossi@scs.carleton.ca (L. Bertossi), chomicki@cse.buffalo.edu (J. Chomicki), xinhe@cse.buffalo.edu (Xin He), raghavan@vuse.vanderbilt.edu (V. Raghavan), spin@vuse.vanderbilt.edu (J. Spinrad).

0304-3975/03/$ - see front matter © 2002 Elsevier Science B.V. All rights reserved.
PII: S0304-3975(02)00737-5

Integration of autonomous data sources: The sources may separately satisfy the constraints but, when the sources are integrated together, the constraints may stop to hold. For instance, consider different, conflicting addresses for the same person in the taxpayer and the voter registration databases. Each of those databases separately satisfies the functional dependency that associates a single address with each person, yet together they violate this dependency. Moreover, since the sources are autonomous they cannot be simply fixed to satisfy the dependency by removing all but one of the conflicting tuples.

Unenforced integrity constraints: Even though integrity constraints capture an important part of the semantics of a given application, they may still fail to be enforced for a variety of reasons. A data source may be a legacy system that does not support the notion of integrity checking altogether. Integrity checking may be too costly (this is often the reason for dropping some integrity constraints from the database schema). Finally, the DBMS itself may support only a limited class of constraints. For example, SQL2 DBMS typically support only *key* functional dependencies (FDs), not arbitrary ones. Therefore, if the relations in a data warehouse are *denormalized* for efficiency reasons, some FDs may become unenforceable.

Temporary inconsistencies: It may often be the case that the database consistency is only temporarily violated and further updates or transactions are expected to restore it. This phenomenon is becoming more and more common, as databases are increasingly involved in a variety of long-running activities or *workflows*.

Conflict resolution: Removing tuples from a database to restore consistency leads to information loss, which may be undesirable. For example, one may want to keep multiple addresses for a person if it is not clear which is the correct one. In general, the process of conflict resolution may be complex, costly, and nondeterministic. In real-time decision-making applications, there may not be enough time to resolve all conflicts relevant to a query.

To formalize the notion of consistent information obtained from a (possibly inconsistent) database in response to a user query, we proposed in [3] the notion of a *consistent query answer*. A consistent answer is, intuitively, true regardless of the way the database is fixed to remove constraint violations. Thus, answer consistency serves as an indication of its reliability. The different ways of fixing an inconsistent database are formalized using the notion of *repair*: another database that is consistent and minimally differs from the original database.

For instance, in the case of multiple addresses of a single person, one can still consistently determine the addresses of those people who have only a single address in the integrated database. Or, more interestingly, if all tuples for the same person have the same birthdate, then the birthdate can be returned as a consistent answer, although there may be multiple conflicting addresses. Also, the different addresses may have a common part, e.g., the state name, that can be consistently returned and will suffice for some queries, e.g., those concerned with taxation. These examples show that simply discarding conflicting data will lead to information loss.

In [3], in addition to a formal definition of a consistent query answer, a computational mechanism for obtaining such answers was presented. However, the queries considered were just *first-order queries*. Here we address in the same context the issue of *aggregation queries*. Aggregation queries are important in OLAP and data warehousing—precisely the context in which inconsistent databases may occur (see above). We limit, however, ourselves to single relations that possibly violate a given set of FDs.

In defining consistent answers to aggregation queries we distinguish between queries with *scalar* and *aggregation* functions. The former return a single value for the entire relation. The latter perform grouping on an attribute (or a set of attributes) and return a single value for each group. Both kinds of queries use the same standard set of SQL-2 aggregate operators: `MIN`, `MAX`, `COUNT`, `SUM`, and `AVG`. In this paper, we address only *aggregation queries with scalar functions*.

Example 1. Consider the following example. Suppose the results of an election in which two candidates, Brown and Green are running, are kept in two relations: *BrownVotes* and *GreenVotes*.

BrownVotes

County	Date	Tally
A	11/07	541
A	11/11	560
B	11/07	302

GreenVotes

County	Date	Tally
A	11/07	653
A	11/11	730
B	11/07	101

Vote tallies in every county should be unique. Consequently, the functional dependency $County \rightarrow Tally$ should hold in both relations. On the other hand, we may want to keep multiple tallies corresponding to different counts (and recounts). Clearly, both relations will have two repairs each, depending on whether the first or the second count for county A is picked. Altogether, the original database has thus four repairs.

The total tally for Brown is 843 in one repair and 862 in the other. For Green, the corresponding figures are 754 and 831. It is clear that there is no single consistent answer to the aggregation query:

```
SELECT SUM(Tally)
FROM BrownVotes
```

and the same holds for the similar query involving the relation *GreenVotes*. Therefore, the notion of consistent query answer from [3] needs to be adapted in the context of aggregation queries. For such queries, we propose to return *ranges* of values: $[843, 862]$ for Brown and $[754, 831]$ for Green. Note that in this case we can safely say that Brown won the election, since the minimum vote for Brown is greater than the maximum vote for Green.

The plan of the paper is as follows. In Section 2, we provide a general definition of consistent answer to an aggregation query with a scalar function. We also define a graph-theoretical representation of database repairs, which is specifically geared towards FDs. In Section 3, we study data complexity of the problem of computing consistent answers to aggregation queries in inconsistent databases. In Section 4, we show how to reduce in practice the computational cost of computing such answers by decomposing the computation into two parts: one that involves standard relational query evaluation and one that computes the consistent answers in a smaller instance. In Section 5, we show that the complexity of computing consistent answers can be reduced by exploiting special properties of the given set of FDs or the given instances. In Section 6 we discuss related and further work.

2. Basic notions

In this paper we assume that we have a fixed database schema containing only one relation schema R with the set of attributes U. We will denote elements of U by $A, B, \ldots$, subsets of U by $X, Y, \ldots$, and the union of X and Y by XY. We also have two fixed, disjoint infinite database domains: D (uninterpreted constants) and N (rational numbers). We assume that elements of the domains with different names are different. The database instances can be seen as finite first-order structures that share the domains D and N. Every attribute in U is typed, thus all the instances of R can contain only elements either of D or of N in a single attribute. Since each instance is finite, it has a finite active domain which is a subset of $D \cup N$. As usual, we allow built-in predicates $(=, \neq, <, >, \leqslant, \geqslant)$ over N that have infinite, fixed extensions. There is also a set of integrity constraints F over R that captures the semantics of the database. E.g., it may express the property that an employee has only a single salary. The instances of the database do not have to satisfy F. A database that satisfies a given set of integrity constraints F, denoted by $r \models F$, is called *consistent*, otherwise *inconsistent*. In this paper we consider only integrity constraints that are functional dependencies (FDs).

2.1. Repairs

The following definitions are adapted from [3].

Definition 1. For the instances r, r', r'', $r' \leqslant_r r''$ if $r - r' \subseteq r - r''$.

Definition 2. Given a set of integrity constraints F and database instances r and r', we say that r' is a *repair* of r w.r.t. F if $r' \models F$ and r' is $\leqslant_r$-minimal in the class of database instances that satisfy F.

We denote by $\mathit{Repairs}_F(r)$ the set of repairs of r w.r.t. F. Examples 1 (earlier) and 2 (below) illustrate the notion of repair.

Because we consider only FDs here and for such constraints all the repairs of an instance are obtained by deleting tuples from it, the notion of repair from [3] can be simplified here. A repair is simply a maximal consistent subset of an instance. Clearly, there are only finitely many repairs, since the relations are finite. Also, in this case the union of all repairs of any instance r is equal to r. These properties are not necessarily shared by other classes of integrity constraints.

Definition 3. The *core* of r is defined as

$$Core_F(r) = \bigcap_{r' \in Repairs_F(r)} r'.$$

The *core* is a new database instance. If r consists of a single relation, then the core is the intersection of all the repairs of r. The core of r itself is not necessarily a repair of r.

Example 2. In Example 1, the relation *BrownVotes* has two repairs

r_1

County	Date	Tally
A	11/07	541
B	11/07	302

r_2

County	Date	Tally
A	11/11	560
B	11/07	302

The core of the relation *BrownVotes* consists of the single tuple

County	Date	Tally
B	11/07	302

and is not a repair. It satisfies the functional dependency $County \to Tally$ but is not a maximal consistent subset of the original instance.

2.2. *Consistent query answers*

2.2.1. *First-order queries*

Query answers for first-order queries are defined in the standard way.

Definition 4. A ground tuple $\bar{t}$ is an *answer* to a query $Q(\bar{x})$ in a database instance r if $r \models Q(\bar{t})$, i.e., the query $Q(\bar{x})$ is true of $\bar{t}$ in the instance r.

Consistent query answers were first defined in [3]. We present here a slightly modified but equivalent definition.

Definition 5. A ground tuple $\bar{t}$ is a *consistent answer* to a query $Q(\bar{x})$ with respect to a set of integrity constraints F in a database instance r if for every $r' \in Repairs_F(r)$, $r' \models Q(\bar{t})$. We denote the set of consistent answers to Q w.r.t. F in r by $Cqa_F^Q(r)$.

Example 3. The query

```
SELECT * FROM BrownVotes
```

has the following consistent answer in the instance of Example 1:

Brown	B	11/07	302

In the same instance the query

```
SELECT County FROM BrownVotes WHERE Tally > 400
```

has A as the only consistent answer. Notice that this answer cannot be obtained by evaluating the query in the original instance from which the conflicting tuples have been removed.

2.2.2. *Aggregation queries*

The aggregation queries we consider are queries of the form

```
SELECT f FROM R
```

where f is one of: `MIN(A)`, `MAX(A)`, `COUNT(A)`, `SUM(A)`, `AVG(A)`, or `COUNT(*)`, where `A` is an attribute of the schema R. These queries return single numerical values by applying the corresponding *scalar function*, i.e., for `MIN(A)` the minimum `A`-value in the given instance, etc. In general, f will also denote an aggregation query (or a scalar function itself). Thus, $f(r)$ will denote the result of applying f to the given instance r of R.

In contrast with first-order queries, there is no single intuitive notion of consistent query answer for aggregation queries. It is likely (see Example 5 below) that aggregation queries return different answers in different repairs, and thus there will be no single consistent answer in the sense of Definition 5. In order to obtain more informative answers even in such a case, we explore therefore several alternative definitions of consistent query answers.

Definition 6. Given a set of integrity constraints F, an aggregation query f and a database instance r, the set of possible answers $Poss_F^f(r)$ is defined as

$$Poss_F^f(r) = \{f(r') \mid r' \in Repairs_F(r)\}.$$

The *greatest-lower-bound* (glb)-*answer* $glb_F^f(r)$ to f w.r.t. F in r is defined as

$$glb_F^f(r) = glb\, Poss_F^f(r).$$

The *least-upper-bound* (lub)-*answer* $lub_F^f(r)$ to f w.r.t. F in r is defined as

$$lub_F^f(r) = lub\, Poss_F^f(r).$$

Example 4. In the instance of Example 1 and the query

```
SELECT SUM(Tally) FROM BrownVotes
```

the set of possible answers is $\{843, 862\}$, the glb-answer is 843 and the lub-answer is 862.

Based on Definition 6, one can envision several possible notions of consistent query answer for aggregation queries:
(1) the set of possible answers $Poss_F^f(r)$,
(2) the range of possible answers $[glb_F^f(r), lub_F^f(r)]$,
(3) some aggregate, for example average, of all possible answers, or
(4) some representation of the distribution of all possible answers.
We conjecture that each of those notions makes sense in the context of some application. In this paper, we study the second notion, that of the range of all possible answers $[glb_F^f(r), lub_F^f(r)]$, for the reasons outlined below.

Example 5. Consider the functional dependency $A \rightarrow B$ and the following family of relation instances r_n, $n > 0$:

A	B
1	0
1	1
2	0
2	2
…	
i	0
i	2^{i-1}
…	
n	0
n	2^{n-1}

We use this example to illustrate two points. First, the instance r_n has 2^n different repairs. Therefore, the approach to computing consistent query answers to any aggregation query (or any other query for that matter) by evaluating the query in every repair separately and then collecting the results is infeasible. Second, note that the aggregation query `SUM(B)` admits a different result in every repair. Actually, every integer in the answer range $[0, 2^n - 1]$ is the result of the query `SUM(B)` in some repair. In spite of that, glb- and lub-answers have polynomial size (since the bounds can be represented in binary). This is will not be the case if we represent all the possible values as a set, a distribution, or some form of disjunctive information e.g., an OR-object [22] or a C-table [21]. (An OR-object is a special domain value specified as a set of atomic values and interpreted as *one* of those values. A C-table is a table with null values that have to satisfy conditions associated with individual rows or the entire table. For a discussion of the relationship between tables with OR-objects and sets of all repairs, see Section 6.)

It is easy to see that glb- and lub-answers in our framework are always polynomially sized and thus exponentially more succinct than set-, distribution-, or disjunction-based representations. However, representing a set of values as a range may lead to information loss. For instance, while we guarantee that the value of the scalar function in every repair falls within the returned range, clearly not every value in this range will necessarily correspond to the value of the function obtained in some repair.

Further aggregating the values of an aggregation query over all repairs, e.g., taking the average, leads to further information loss. In fact, presented with such an answer the user can no longer say anything about the values the query has in the individual repairs.

We should note that regardless of whether a range- or a set-based representation is used, the obtained result is semantically not a standard relation, so it cannot directly serve as input to other SQL queries. In the first case, the obtained range $[a,b]$ can be represented as a pair but in fact should be interpreted as a *condition* $a \leqslant v \leqslant b$ on the repair-dependent value v of the scalar function. In the second case, the result is a set and thus requires going beyond First Normal Form. Moreover, the set needs to be interpreted as a condition too, in this case disjunctive. (The condition is $x = v_1 \vee \cdots \vee x = v_k$ where $\{v_1, \ldots, v_k\}$ is the set of possible values of the scalar function.)

We will also consider other auxiliary notions of query answer in inconsistent databases. *Core* answers are used for hybrid evaluation in Section 4 and *union* answers are defined for symmetry with core answers.

Definition 7. A number v is a *core answer* to f w.r.t. F in r if

$$v = f(Core_F(r)) = f\left(\bigcap_{r' \in Repairs_F(r)} r'\right).$$

A number v is a *union answer* to f w.r.t. F in r if

$$v = f\left(\bigcup_{r' \in Repairs_F(r)} r'\right).$$

However, union answers are trivial for FDs, as the union of all the repairs of r is r itself, so the union answer reduces to $f(r)$.

2.3. Graph representation

Given a set of FDs F and an instance r, all the repairs of r w.r.t. F can be succinctly represented as a graph.

Definition 8. The *conflict graph* $G_{F,r}$ is an undirected graph whose set of vertices is the set of tuples in r and whose set of edges consists of all the edges (t_1, t_2) such that $t_1 \in r$, $t_2 \in r$, and there is a dependency $X \rightarrow Y \in F$ for which $t_1[X] = t_2[X]$ and $t_1[Y] \neq t_2[Y]$.

Example 6. Consider a schema $R(AB)$, the set F of two FDs $A \rightarrow B$ and $B \rightarrow A$, and an instance $r = \{(a_1,b_1),(a_1,b_2),(a_2,b_2),(a_2,b_1)\}$ over this schema. The conflict graph $G_{F,r}$ looks as follows:

$$\begin{tikzcd}(a_1,b_1) \arrow[r, dash] \arrow[d, dash] & (a_1,b_2) \arrow[d, dash] \\ (a_2,b_1) \arrow[r, dash] & (a_2,b_2).\end{tikzcd}$$

Definition 9. An *independent set S* in an (undirected) graph $G = (V,E)$ is a subset of the set of vertices V of this graph, such that there is no edge in the set of edges E connecting two vertices in S. A *maximal* independent set is an independent set which is not a proper subset of any other independent set. A *maximum* independent set is an independent set of maximum cardinality.

Proposition 1. *Each repair in $Repairs_F(r)$ corresponds to a maximal independent set in $G_{F,r}$ and vice versa.*

Conflict graphs are geared specifically towards FDs. The repairs of other classes of constraints do not necessarily have similar representations.

We also note that, for a given set of FDs F over R, one can write an SQL2 query that for any instance r of R computes the edges of the conflict graph $G_{F,r}$.

2.4. Computational complexity

2.4.1. Data complexity

The data complexity notion [8,31] makes it possible to study the complexity of query processing as a function of the number of tuples in the database instance. We define separately the data complexity of checking repairs, the data complexity of computing consistent query answers to first-order queries, and that of computing consistent query answers to aggregation queries.

Definition 10. Given a class of databases $\mathscr{D}$ and a class of integrity constraints, the *data complexity* of checking repairs is defined to be the complexity of determining the membership of the sets

$$D_F = \{(r,r') \mid r \in \mathscr{D} \wedge r' \in Repairs_F(r)\}$$

for a fixed finite set F of integrity constraints. This problem is *C-data-hard* for a complexity class C if there is a finite set of integrity constraints F_0 such that D_{F_0} is C-hard.

Lemma 1. *For a given set F of FDs, the data complexity of checking whether an instance r' is a repair of r is in PTIME.*

Proof. Checking whether r' satisfies F is in PTIME. The repair r' has also to be $\leqslant_r$-minimal among those instances that satisfy F. For FDs, it means that r' has to be a maximal subset of r that satisfies F. Checking this property can be done as follows: try all the tuples $\bar{t}$ in $r - r'$, one by one. If $r' \cup \{\bar{t}\}$ satisfies F, then r' is not maximal. Otherwise, if for no such tuple $\bar{t}$, $r' \cup \{\bar{t}\}$ satisfies F, no superset of r' can satisfy F (violations of FDs cannot be removed by adding tuples) and r' is maximal. □

Definition 11. Given a class of databases $\mathscr{D}$, a class of first-order queries $\mathscr{L}$ and a class of integrity constraints, the *data complexity* of computing consistent query answers is defined to be the complexity of determining the membership of the sets

$$D_{F,\phi} = \{(r, \bar{t}) \mid r \in \mathscr{D} \wedge \bar{t} \in Cqa_F^{\phi}(r)\}$$

for a fixed $\phi \in \mathscr{L}$ and a fixed finite set F of integrity constraints. This problem is *C-data-hard* for a complexity class C if there is a query $\phi_0 \in \mathscr{L}$ and a finite set of integrity constraints F_0 such that D_{F_0,ϕ_0} is C-hard.

From Lemma 1, we can immediately obtain:

Corollary 1. *For any set of FDs F and first-order query Q, the data complexity of checking whether a tuple $\bar{t}$ is a consistent answer to Q is in* co-NP.

In Section 3, we will see that the above problem is in fact co-NP-hard (Corollary 2).

Definition 12. Given a class of databases $\mathscr{D}$, a class of aggregation queries $\mathscr{F}$ and a class of integrity constraints, the *data complexity* of computing the glb-answer (resp. lub-answer) is defined to be the complexity of determining the membership of the sets

$$D_{F,f} = \{(r, k) \mid r \in \mathscr{D} \wedge glb_F^f(r) \leqslant k\}$$

and

$$D_{F,f} = \{(r, k) \mid r \in \mathscr{D} \wedge lub_F^f(r) \geqslant k\},$$

respectively, for a fixed aggregation query $f \in \mathscr{F}$ and a fixed finite set F of integrity constraints. This problem is *C-data-hard* for a complexity class C if D_{F_0,f_0} is C-hard for some aggregation query $f_0 \in \mathscr{F}$ and a finite set of integrity constraints F_0.

In our case, each class of aggregation queries $\mathscr{F}$ contains only queries that use scalar functions of the same kind, e.g., `MIN(A)` for some attribute `A` of R.

Proposition 2. *For every class of aggregation queries F that contains only queries with scalar functions of the same kind, computing the glb- and lub-answer is in* NP.

Proof. Consider computing the glb-answer (the other case is symmetric). We have that $glb_F^f(r) \leqslant k$ if and only if there is a repair $r' \in Repairs_F(r)$ such that $f(r') \leqslant k$. The latter condition can be clearly checked in NP, in the view of Lemma 1. □

Our PTIME results will yield algorithms that compute the glb-answer $glb_F^f(r)$ (or $lub_F^f(r)$), which is clearly sufficient to determine the truth of the condition $glb_F^f(r) \leqslant k$ (resp. $lub_F^f(r) \geqslant k$).

3. Complexity of scalar aggregation

We have seen (Example 5) that there may be exponentially many repairs even in the case of one functional dependency. Therefore, it is computationally infeasible to evaluate a scalar aggregation query in every repair. In [3] and this paper, we have identified two ways of computing consistent answers by querying the given, possibly inconsistent database instance, without having to compute all the repairs. Query transformation modifies the original query, Q, into a new query, $T(Q)$, that returns only consistent answers. We have applied this approach in [3] to restricted first-order queries and universal integrity constraints. Except in some simple cases, this approach does not seem applicable to aggregation queries. For example, even when `MAX(A)` and `MIN(A)` queries can be written as first-order queries, their resulting syntax does not allow the application of the methodology developed in [3] to them. Moreover, as argued earlier in the paper, aggregation queries seem to require a different notion of consistent query answer than first-order queries. Therefore, we use instead the fact that for FDs, the set of all repairs of an instance can be compactly represented as the conflict graph. We develop techniques and algorithms geared specifically towards this representation.

We start by considering core answers—an easy case. Then we consider several aggregate operators—`MIN`, `MAX`, `SUM` and `COUNT(*)`—together. They share common properties: for each of them computing glb- and lub-answers is tractable only in the case of a single functional dependency and the proof of tractability uses the same technique of building an appropriate single repair. Subsequently, we consider the `AVG` operator which requires a much more involved tractability proof. Finally, we study `COUNT(A)`, for which even the single-dependency case is not tractable.

In the following r denotes an instance of the schema R. The input of the problem of computing consistent query answers will consist of r and a numerical parameter k (as required by Definition 12).

3.1. Core answers

For some aggregate operators, e.g., `COUNT` and `SUM` of nonnegative values, a core answer is a lower-bound-answer, but not necessarily the glb-answer. As we will see in Section 4, computing core answers to aggregation queries can be useful for computing consistent answers.

Theorem 1. *The data complexity of computing core answers for any scalar function is in PTIME.*

Proof. The core consists of all the isolated vertices in the conflict graph. □

We note that, for a given set of FDs F over R, one can write an SQL2 query that computes for any instance r of R the set of isolated vertices in the conflict graph $G_{F,r}$.

In general, computing glb- and lub-answers is considerably more involved than computing core answers.

3.2. *Aggregation using* `MIN`*,* `MAX`*,* `SUM` *and* `COUNT(*)`

3.2.1. *One functional dependency*

Consider `MAX(A)` (`MIN(A)` is symmetric). In this case computing the lub-answer in r w.r.t. an arbitrary set of FDs F consists of evaluating `MAX(A)` in r, thus it is clearly in PTIME. However, it is not obvious how to compute the glb-answer, namely the minimum of the set of maximums obtained by posing the query `MAX(A)` in every repair. Computing `MAX(A)` in $Core_F(r)$ gives us only a lower-bound-answer which does not have to be the glb-answer.

Theorem 2. *The data complexity of computing $glb_F^f(r)$ in r for a set of FDs F consisting of a single FD $X \rightarrow Y$ and $f \in \{$*`MAX(A),SUM(A),COUNT(*)`*$\}$ is in PTIME.*

Proof. The approach for all of the above scalar functions is essentially identical and consists of constructing a repair that minimizes the value of the scalar function. Call an (X,Y)-*cluster* a maximal set of tuples of r that have the same attribute values in X and Y. Clearly, in a single repair we can have only one (X,Y)-cluster for every given value of X. For every value of the attribute X we pick that (X,Y)-cluster that minimizes the scalar function and apply the scalar function to this cluster. Finally, we aggregate the obtained values across all values of X (and combine the (X,Y)-clusters if we want to obtain a repair minimizing f). This approach gives the minimum of the scalar function over all repairs. For `MAX(A)` it can be defined in SQL2 as the following sequence of views:

```
CREATE VIEW S(X,Y,C) AS
  SELECT X,Y,MAX(A) FROM R
  GROUP BY X,Y;

CREATE VIEW T(X,C) AS
  SELECT X, MIN(C) FROM S
  GROUP BY X;

SELECT MAX(C) FROM T;
```

For `SUM(A)`, we only have to replace `MAX` in the above by `SUM`. For `COUNT(*)`, we replace `MAX(A)` by `COUNT(*)` and `MAX(C)` by `SUM(C)`. Evaluating all those SQL2 queries can be done in PTIME. □

It is clear that there is a symmetric result to Theorem 2 for lub-answers to `MIN(A)`. Note that

$$lub_F^{\mathtt{MIN(A)}}(r) = -glb_F^{\mathtt{MAX(A)}}(r^-),$$

where r^- contains identical tuples to r except that their A-values are inverted (every A-value v is changed to $-v$).

We show now that Theorem 2 exhausts the tractable cases for the scalar functions in question.

3.2.2. Two FDs and `MAX(A)`

Theorem 3. *There is a set of* 2 *FDs* F_0 *for which deciding whether*

$$glb_{F_0}^{\mathtt{MAX(A)}}(r) \leqslant k$$

in r *is* NP-*data-hard.*

Proof. Reduction from 3SAT. Consider a propositional formula $\varphi = C_1 \wedge \cdots \wedge C_n$ in CNF. Let $p_1, \ldots p_m$ be the propositional variables in φ. Construct a relation r with the attributes A, B, C, D, and containing exactly the following tuples:
(1) $(p_i, 1, C_j, 1)$ if making p_i true makes C_j true,
(2) $(p_i, 0, C_j, 1)$ if making p_i false makes C_j true, and
(3) $(w, 2, C_j, 2)$, $1 \leqslant j \leqslant n$, where w is a new symbol.

The set of FDs F_0 consists of $A \rightarrow B$ (each propositional variable cannot have more than one truth value) and $C \rightarrow D$. Also, $k = 1$. We show that $glb_{F_0}^{\mathtt{MAX(D)}}(r) = 1$ iff φ is satisfiable.

Assume $glb_{F_0}^{\mathtt{MAX(D)}}(r) = 1$. Then there is a repair r_0 of r in which the attribute D assumes only the value 1. If for some j the repair r_0 does not contain any tuple of the form $(\_, \_, C_j, 1)$, then r_0 has to contain the tuple $(w, 2, C_j, 2)$ and `MAX(D)` returns 2 in this repair, a contradiction. From r_0 we can build a satisfying assignment for φ by reading off the values of the attributes A and B for each conjunct C_j. Notice that r_0 has to satisfy the FD $A \rightarrow B$ and thus each propositional variable receives in this way only a single value.

Assume now that φ is satisfiable. Then, given a satisfying assignment, we build a database instance r_1 in the following way: for every propositional variable p_i made true by the assignment and every conjunct C_j in which this variable occurs positively, we include the tuple $(p_i, 1, C_j, 1)$ in r_1. The variables made false by the assignment are treated symmetrically. Clearly, r_1 satisfies $A \rightarrow B$. Since the assignment satisfies φ, for every conjunct C_j there is a tuple in r_1 which has C_j as the value of the attribute C. Therefore, r_1 cannot contain any tuples of the third kind, and has to satisfy $C \rightarrow D$ as well. It is also maximal, and thus a repair. Since in every repair of r, `MAX(D)` returns a value greater or equal to 1, and `MAX(D)` returns 1 in r_1, then $glb_{F_0}^{\mathtt{MAX(D)}}(r) = 1$. □

The above reduction yields also a lower bound for checking consistent query answers for first-order queries.

Corollary 2. *There is a set of* 2 *FDs* F_0 *and a first-order query* Q *for which the problem of checking whether* $\bar{t}$ *is a consistent answer to* Q *is* co-NP-*data-hard.*

Proof. We use the same reduction and the same set of FDs F_0 as in Theorem 3. We note that the formula φ is unsatisfiable iff 2 is a consistent answer to the query

$$\exists x, y, z.\ R(x, y, z, w). \qquad \Box$$

Corollary 2 should be contrasted with the results of [3], which imply that in the presence of FDs the data complexity of computing consistent query answers for first-order queries consisting only of quantifier-free conjunctions of positive and negative literals is in PTIME. Thus Corollary 2 identifies the existential quantifier as a source of intractability.

3.2.3. *Two FDs and* `COUNT(*)`

We consider now `COUNT(*)`.

Lemma 2. *There is a set of* 2 *FDs* F_1 *for which the problem of determining the existence of a repair of* r *of size* $\geqslant k$ *is* NP-*data-hard.*

Proof. Reduction from 3-Colorability. Given a graph $G=(N,E)$, with $N=\{1,2,\ldots,n\}$, such that $(i,i)\notin E$ for each $i\in[1,n]$, and given colors w (white), b (blue) and r (red), we define the relation p with attributes A,B,C,D and the following tuples:

(1) for every $1\leqslant i\leqslant n$, $(i,w,i,w)\in p$, $(i,b,i,b)\in p$ and $(i,r,i,r)\in p$, and
(2) for every $(i,j)\in E$, $(i,w,j,b)\in p$, $(i,w,j,r)\in p$, $(i,b,j,w)\in p$, $(i,b,j,r)\in p$, $(i,r,j,w)\in p$ and $(i,r,j,b)\in p$.

We consider the set of FDs $F_1=\{A\rightarrow B,\ C\rightarrow D\}$. We will show that G is 3-colorable iff there is a repair p' of p with exactly $n+2\cdot|E|$ tuples (the maximum possible number of tuples in a repair). That property follows from Lemmas 3, 4 and 5. $\Box$

Lemma 3. *Assuming* p *is defined as in the proof of Lemma* 2, *every repair* p' *of* p *has at most* $n+2\cdot|E|$ *tuples.*

Proof. by induction on n. If n is equal to 1, then p is equal to

p

1	w	1	w
1	b	1	b
1	r	1	r

and, therefore, it has three repairs:

p_1

1	w	1	w

p_2

1	b	1	b

p_3

1	r	1	r

.

Thus, $|p_1|=|p_2|=|p_3|=1\leqslant n+2\cdot|E|$.

Suppose that the theorem is satisfied in every graph with n nodes. Let (N,E) be a graph containing $n+1$ nodes, p be a table constructed from (N,E) as we showed above and p' be a repair of p. Define $N^* = N - \{n+1\}$, $E^* = E \cap N^* \times N^*$, $p^* = p \cap N^* \times \{w,b,r\} \times N^* \times \{w,b,r\}$ and $(p^*)' = p' \cap N^* \times \{w,b,r\} \times N^* \times \{w,b,r\}$.

$(p^*)'$ satisfies the set of functional dependencies and it only contains tuples from table p^*. Then, there exists a repair $(p^*)''$ of p^* such that $(p^*)' \subseteq (p^*)''$. Thus, by induction hypothesis we conclude that $|(p^*)''| \leqslant n + 2 \cdot |E^*|$, and, therefore, $|(p^*)'| \leqslant n + 2 \cdot |E^*|$.

In order to know how many tuples p' could have, we need to know how many tuples $p' - (p^*)'$ could contain, which can be established by considering the following:

(I) This set could contains at most one of the following tuples: $(n+1,w,n+1,w)$, $(n+1,b,n+1,b)$, $(n+1,r,n+1,r)$.

(II) For each $(i,n+1) \in E$, this set could contains at most two tuples of the form $(i,color_1,n+1,color_2)$, $(n+1,color_3,i,color_4)$.

By (I) and (II) we conclude that

$$|p' - (p^*)'| \leqslant 1 + 2 \cdot |E - E^*|$$

and, therefore,

$$|p'| \leqslant n + 2 \cdot |E^*| + 1 + 2 \cdot |E - E^*| \leqslant n + 1 + 2 \cdot |E|. \quad \square$$

Lemma 4. *Assuming p is defined as in the proof of Lemma* 2, *if it is possible to color the graph* (N,E) *where* $N = \{1,2,\ldots,n\}$, *with colors* w, b and r, *then there exists a repair of p with* $n + 2 \cdot |E|$ *tuples.*

Proof. Suppose that C_i is the color assigned to the node i in the graph. Define p' as follows:

(1) for every $1 \leqslant i \leqslant n$, $(i,C_i,i,C_i) \in p'$, and

(2) for every $(i,j) \in E$, $(i,C_i,j,C_j) \in p'$ and $(j,C_j,i,C_i) \in p'$.

Clearly, p' satisfies the integrity constraints $A \to B$ and $C \to D$. But $|p'| = n + 2 \cdot |E|$ and, therefore, by the previous lemma we conclude that p' is a repair of p. $\square$

Lemma 5. *Assuming p is defined as in the proof of Lemma* 2, *if there is a repair* p' *of p with* $n + 2 \cdot |E|$ *tuples, then is possible to color the graph* (N,E) *by using colors* w, b *and* r.

Proof. Let $q = \{(i,x,i,x) \mid 1 \leqslant i \leqslant n$ and x is equal to w, b or $r\}$. For every $(i,j) \in E$, there are 12 tuples in p mentioning i and j:

(i,w,j,b), (i,r,j,w), (j,b,i,w),
(i,w,j,r), (i,r,j,b), (j,b,i,r),

(i,b,j,w), (j,w,i,b), (j,r,i,w),
(i,b,j,r), (j,w,i,r), (j,r,i,b).

A repair of p must have at most two tuples from this set and, therefore, $|p'-q| \leqslant 2 \cdot |E|$. Thus, $|p' \cap q|$ must be equal to n, since $|p'| = n + 2 \cdot |E|$. Hence, for every node i there exists a color C_i such that $(i, C_i, i, C_i) \in p'$. We will prove that if we choose color C_i for painting node i, then we have a coloring for the graph.

Let $(i,j) \in E$. There are at most two tuples in p' that mention i and j together. If we have zero or one of these kind of tuples, $|p'-q| < 2 \cdot |E|$ and, therefore, $|p'| < n+2 \cdot |E|$, a contradiction. Thus, we have exactly two tuples in p' mentioning i and j together. But these tuples together with (i, C_i, i, C_i) and (j, C_j, j, C_j) cannot violate the set of FDs, because p' is a repair. Then $(i, C_i, j, C_j) \in p$ and $(j, C_j, i, C_i) \in p$. By the definition of p, we conclude that $C_i \neq C_j$. □

Lemma 6. *There is a set of* 2 *FDs* F_2 *for which the problem of determining the existence of a repair of* r *of size* $\leqslant k$ *is* NP-*data-hard.*

Proof. Modification of the lower bound proof of Theorem 3. We build the instance by using the same tuples of the first and second kinds, as well as “sufficiently many tuples” of the third kind, each with a different new symbol w. It is enough to have $3n+1$ tuples of the third kind for each clause (where n is the number of clauses), thus the instance will have the total of $3n + n(3n+1)$ tuples. Every repair that contains a tuple of the third kind, has to contain at least $3n+1$ such tuples (by maximality). The formula φ is satisfiable iff there is a repair of size $\leqslant 3n$. □

Lemmas 2 and 6 imply the following theorems:

Theorem 4. *There is a set of two FDs* F_1 *for which determining whether*

$$lub_{F_1}^{\mathtt{COUNT(*)}}(r) \geqslant k$$

in r *is* NP-*data-hard.*

Theorem 5. *There is a set of two FDs* F_2 *for which determining whether*

$$glb_{F_2}^{\mathtt{COUNT(*)}}(r) \leqslant k$$

in r *is* NP-*data-hard.*

Analogous results to Theorems 4 and 5 can be obtained for `SUM(A)` and the proofs are easy modifications of the above proofs (`COUNT(*)` can be mimicked by `SUM` over an additional attribute that has the value 1 in each tuple).

3.3. Aggregation using AVG

3.3.1. One functional dependency

We reduce the problem of computing glb- and lub-answers to AVG(A) queries w.r.t. a single FD $X \rightarrow Y$ to the following problem of Maximum Average Weight (MAW):

> There are m bins, each containing weighted, colored objects. No two bins have objects of the same color, although any particular bin may contain more than one object of the same color. Choose exactly one color for each bin in such a way that the sum of the weights of all objects of the chosen colors divided by the total number of such objects (i.e., the average weight AVG of objects of the chosen color) is maximized.

In the reduction of the problem of computing lub-answers to MAW, bins correspond to different X-values, and objects of the same color have the same Y-values. Each object corresponds to a tuple in which the attribute A represents the weight of the object. Different objects of the same color can have different weight, since A does not have to be a member of Y or be functionally dependent on Y. For glb-answers, we use an *inverted* database, as in the remarks after the proof of Theorem 2.

To solve MAW, consider the well-known "2-OPT" strategy of starting with an arbitrary selection $\langle c_1, c_2, \ldots, c_m \rangle$ of one color each from each of the m bins. The 2-OPT strategy is simply to replace a color from one bin with a different color from the same bin if doing so increases the value of the average weight of objects of the colors in the selection.

More precisely, let $c = \langle c_1, c_2, \ldots, c_m \rangle$ be a selection of colors such that c_i is the color chosen from the ith bin. Let $AVG(c)$ be the average weight of objects with colors from c. Let OPT be the maximum value over all choices of c of $AVG(c)$. Then 2-OPT is the end result of the following strategy:

```
Let $c$ be any arbitrary selection of $m$ colors, one from each bin.
while there is a color $c'_i$ in bin $i$ : $AVG(\langle c_1, c_2, \ldots, c_{i-1}, c'_i, c_{i+1}, \ldots, c_m \rangle) > AVG(c)$ do
        $c := \langle c_1, c_2, \ldots, c_{i-1}, c'_i, c_{i+1}, \ldots, c_m \rangle$
endwhile
2-$OPT := AVG(c)$
```

We establish the proof of the main theorem through two intermediate lemmas.

Lemma 7. 2-$OPT = OPT$.

Proof. Let n_i denote the number of objects of color c_i and let w_i denote the total weight of objects of color c_i. For any color c'_i in bin i, it can be verified (after a little bit of arithmetic) that $AVG(\langle c_1, c_2, \ldots, c_{i-1}, c'_i, c_{i+1}, \ldots, c_m \rangle) > AVG(c)$ if and only if

one of the following holds (where n_i' is the number of objects of color c_i'):
(1) $n_i = n_i'$ and $w_i' > w_i$,
(2) $n_i < n_i'$ and $(w_i' - w_i)/(n_i' - n_i) > AVG(c)$, and
(3) $n_i > n_i'$ and $(w_i' - w_i)/(n_i' - n_i) < AVG(c)$.

Intuitively, the conditions above may be interpreted in the following way. Let the density of a set of objects be the sum of the weight of the objects divided by the number of objects. A swap is beneficial if and only if the net changes made correspond to adding objects with density greater than the current average density of the solution, or deleting objects with density smaller than the current average density of the solution.

Now if $2\text{-}OPT < OPT$, then there is some c for which none of the above 3 conditions holds for any choice of c_i' in any bin i and yet OPT is larger than $AVG(c)$. Specifically, let the coloring for OPT be $d = \langle d_1, d_2, \ldots, d_m \rangle$ where the total weight of objects of color d_i is u_i and the total number of such objects is m_i. We will show that for some i, the choice $c_i' = d_i$ will satisfy one of the above three conditions, yielding a contradiction.

First observe that, for any i such that $n_i = m_i$, having $w_i < u_i$ would immediately give a contradiction. Also, if $u_i < w_i$, then OPT can be improved by replacing d_i with c_i, again a contradiction. Therefore, if $n_i = m_i$ it must be that $u_i = w_i$ and we may as well assume that $d_i = c_i$ for all such i.

Next consider the colors which are different in d and c. We will use the elementary fact that if

$$\frac{p+A+B}{q+C+D} > \frac{p}{q} \quad \text{then either} \quad \frac{p+A}{q+C} > \frac{p}{q} \quad \text{or} \quad \frac{p+B}{q+D} > \frac{p}{q}.$$

In particular, let $q = \sum_i n_i$ and $p = \sum_i w_i$. Let $E = \sum_{i:d_i \neq c_i} u_i - w_i$ and $F = \sum_{i:d_i \neq c_i} m_i - n_i$. Observe that $(p+E)/(q+F) = OPT > 2\text{-}OPT = p/q$. If the sum in E runs over only one index i then $c_i' = d_i$ satisfies (2) or (3) above, a contradiction. Otherwise, E may be partitioned into two sums A and B and F into corresponding sums C and D such that the above fact guarantees that either $(p+A)/(q+C) > 2\text{-}OPT$ or $(p+B)/(q+D) > 2\text{-}OPT$. If the former is true, we replace E and F with A and C; otherwise we replace E and F with B and D. Repeated application of the above fact in this manner will eventually result in finding some i such that $c_i' = d_i$ satisfies (2) or (3), a contradiction. □

We have just shown that the simple $2\text{-}OPT$ strategy will converge to the value OPT. However, it does not necessarily follow that the number of iterations of the $2\text{-}OPT$ strategy is polynomial. For this, we need another idea.

Let c be any selection of colors with one color from each bin. We say that color c_i is *stable* if there exists no c_i' in the ith bin for which condition (1), (2), or (3) holds. Note that if color c_i can be replaced by any color in bin i to produce an increase in the value of AVG, then there exists a stable color with which it can be replaced, this simply being the color which results in the largest value of AVG obtained by maintaining the colors in all bins other than i fixed while trying different colors from the ith bin. Clearly, such a stable color can be found by simply cycling through the choices for the ith bin. This leads to the following "*Stable*-2-*OPT*" strategy.

```
Let $c$ be any arbitrary selection of $m$ colors, one from each bin.
while there is a color $c_i'$ in bin $i$ : $AVG(\langle c_1, c_2, \ldots, c_{i-1}, c_i', c_{i+1}, \ldots, c_m\rangle) > AVG(c)$ do
    Find a stable color $d_i$ for bin $i$.
    $c := \langle c_1, c_2, \ldots, c_{i-1}, d_i, c_{i+1}, \ldots, c_m\rangle$
endwhile
$Stable\text{-}2\text{-}OPT := AVG(c)$
```

From Lemma 7, it follows that $Stable\text{-}2\text{-}OPT = OPT$ also. In addition, we claim the following.

Lemma 8. *Any color c_i is chosen in the Stable-2-OPT strategy at most once as a stable color for the ith bin.*

Proof. Consider the situation when a color c_i is chosen as a stable color for the first time. At this point in time, none of conditions (1), (2), or (3) holds with respect to other colors c_i' in bin i. This means that none of the colors c_i' with $n_i < n_i'$ can ever take the ith position as a stable color since AVG increases monotonically in the run of the strategy and color c_i will always be preferred over such a color c_i'. Similarly, none of the colors c_i' with $n_i' = n_i$ can ever take the ith position as a stable color. Finally, if a color c_i' replaces c_i as a stable color, it must be because AVG has increased to such an extent that condition (3) now holds; subsequently, the monotonicity of AVG ensures that color c_i' will always be preferred to color c_i as a stable color for the ith bin and hence color c_i will never ever be chosen again. □

Each iteration of the while loop chooses a new stable color and by Lemma 8, a color is chosen at most once. It follows that the number of iterations of the while loop is at most the number of colors available. Therefore, $Stable\text{-}2\text{-}OPT$ finishes in polynomial time.

Now the main theorem follows from Lemmas 7 and 8.

Theorem 6. *If the set of FDs F consists of a single dependency $X \to Y$, with $X \cap Y = \emptyset$, then the data complexity of computing both $glb_F^{\mathtt{AVG(A)}}(r)$* and *$lub_F^{\mathtt{AVG(A)}}(r)$* in an instance *$r$* is in PTIME.

3.3.2. Two FDs

Theorem 7. *There is a set of two FDs F_3 for which determining whether*

$$glb_{F_3}^{\mathtt{AVG(A)}}(r) \leqslant k$$

in r is NP*-data-hard.*

Proof. We can use the same reduction from 3SAT as in Theorem 3. Given a set of clauses, there is a satisfying assignment if and only if there is a repair of the corresponding database r for which $\mathtt{AVG(D)} = 1$ (since otherwise the glb-answer is greater than 1). This is the case if and only if $glb_{F_3}^{\mathtt{AVG(D)}} r \leqslant 1$. □

Theorem 8. *There is a set of two FDs F_4 for which determining whether*

$$lub_{F_4}^{\texttt{AVG(A)}}(r) \geqslant k$$

in r is NP-*data-hard.*

Proof. We reduce 3SAT to our problem. Change the tuples of the instance in the proof of Theorem 3 as follows:

(3′) $(w, 2, C_j, d)$, $1 \leqslant j \leqslant n$, where w is a new symbol and $d < 1$.

Given a set of clauses, there is a satisfying assignment if and only if there is a repair of the corresponding database r for which $\texttt{AVG(D)} = 1$. This is the case if and only if $lub_{F_4}^{\texttt{AVG(D)}} r \geqslant 1$. □

3.4. Aggregation using COUNT(A)

We assume here that distinct values of A are counted (`COUNT (DISTINCT A)`).

Theorem 9. *There is a single FD $d_0 = B \rightarrow A$ for which determining whether*

$$glb_{d_0}^{\texttt{COUNT}(A)}(r) \leqslant k$$

in r is NP-*data-hard.*

Proof. To see that the lower bound holds, we will encode an instance of the HITTING SET problem in r (whose schema is $R(A, B)$). The HITTING SET problem [14] is formulated as follows: Given a collection $C = \{S_1, \ldots, S_n\}$ of sets, is there a set H (called a *hitting set*) with k or fewer elements that intersects all the members of C? For every set S_i in C and every element $x \in S_i$ we put the tuple (x, i) in r. There is in C a hitting set of size less than or equal to k if and only if there is a repair of r with at most k different values of the first attribute A. □

Theorem 10. *There is a single FD $d_1 = A \rightarrow B$ for which determining whether*

$$lub_{d_1}^{\texttt{COUNT}(C)}(r) \geqslant k$$

in r is NP-*data-hard.*

Proof. We reduce SAT to this problem. Let $\varphi = C_1 \wedge \cdots \wedge C_n$. Consider the functional dependency $A \rightarrow B$ and the database instance r over the schema ABC with the following tuples:
(1) $(p_i, 1, C_j)$ if making p_i true makes C_j true, and
(2) $(p_i, 0, C_j)$ if making p_i false makes C_j true.
Then, φ is satisfiable iff $lub_{d_1}^{\texttt{COUNT(C)}}(r) \geqslant n$. □

3.5. Summary of complexity results

The following is a tabular summary of the results presented in this section. The membership in NP is from Proposition 2.

	glb-answer		lub-answer	
	$\|F\|=1$	$\|F\|\geqslant 2$	$\|F\|=1$	$\|F\|\geqslant 2$
`MIN(A)`	PTIME	PTIME	PTIME	NP-complete
`MAX(A)`	PTIME	NP-complete	PTIME	PTIME
`COUNT(*)`	PTIME	NP-complete	PTIME	NP-complete
`COUNT(A)`	NP-complete	NP-complete	NP-complete	NP-complete
`SUM(A)`	PTIME	NP-complete	PTIME	NP-complete
`AVG(A)`	PTIME	NP-complete	PTIME	NP-complete

4. Hybrid computation

As we have seen, determining glb- and lub-answers is often computationally hard. However, it seems that hard instances of those problems are unlikely to occur in practice. We expect that in a typical instance a large majority of tuples are not involved in any conflicts. If this is the case, it is advantageous to break up the computation of the lub- (or the glb-answer) to f in r into three parts:

(1) the computation of f in the core of r,
(2) the computation of the lub-answer to f in the complement of the core of r (which should be small), and
(3) the combination of the results of the first two steps using an operator g (which depends on f).

The first step can be done using a DBMS because the core of r can be computed using a first-order query (Theorem 1).

Definition 13. The scalar function f *admits a g-decomposition* of its lub-answers (resp. glb-answers) w.r.t. a set of FDs F if for every instance r of R, the lub-answer (resp. glb-answer) v to f satisfies the condition

$$v = g(f(Core_F(r)), v'),$$

where $v' = lub_F^f(r - Core_F(r))$ (resp. $v' = glb_F^f(r - Core_F(r))$).

Theorem 11. *The following pairs describe g-decompositions admitted by scalar functions f:*
(1) $f=$`MIN(A)`, $g=\min$,
(2) $f=$`MAX(A)`, $g=\max$,

(3) $f = \texttt{COUNT(*)}$, $g = +$, and
(4) $f = \texttt{SUM(A)}$, $g = +$.

Proof. First, notice that every repair r' of r w.r.t. a set of FDs F is a union of $Core_F(r)$ and a repair of $r - Core_F(r)$. Now to see that the first decomposition holds for $f = \texttt{MIN(A)}$ consider:

$$lub\, Poss_F^f(r) = \min(f(Core_F(r)), lub\, Poss_F^f(r - Core_F(r)))$$

and similarly for glb-answers and other decompositions. □

5. Special cases

We consider here several cases when the conflict graph has a special form that could be used to reduce the complexity of computing answers to aggregation queries. We only consider lub-answers to `COUNT(*)` queries. It is an open question whether our approach will generalize to other classes of scalar aggregation queries.

5.1. BCNF

We show here that if the set of FDs F has two dependencies and the schema R is in Boyce–Codd Normal Form (BCNF), computing lub-answers to `COUNT(*)` queries can be done in PTIME. This should be contrasted with Theorem 4 which showed that two dependencies without the BCNF assumption are sufficient for NP-hardness.

Given a set of FDs F in a schema R, we say that the schema R is in *BCNF* if all the dependencies in F are of the form $X \rightarrow Y$ where X contains a key of R (w.r.t. F). This definition can be found in every database textbook. BCNF is often satisfied in practice, since schemas in BCNF are considered good, by the virtue of being free of redundancies and insertion/deletion/update anomalies. For example, the relation instance in the proof of Theorem 3 is not in BCNF, since neither A nor C is a key of this relation.

We pursue here two different approaches to BCNF schemas. The first [5] is based on the observation that for 2 FDs in BCNF the conflict graph is claw-free. For such graphs computing a maximum independent set (an independent set of maximum cardinality) can be done in PTIME. The second approach is direct and yields a subquadratic time complexity bound.

Definition 14. An FD $X \rightarrow Y$ is a *partition dependency* over R if $X \cup Y = U$ (where U is the set of all the attributes of R) and $X \cap Y = \emptyset$.

Lemma 9. *For any instance r of R and any partition dependency $d = X \rightarrow Y$ over R, the conflict graph $G_{d,r}$ is a union of disjoint cliques.*

Proof. Assume (t_1, t_2) and (t_2, t_3) are two edges in $G_{d,r}$ such that $t_1 \neq t_3$. Then $t_1[X] = t_2[X]$, $t_1[Y] \neq t_2[Y]$, $t_2[X] = t_3[X]$, and $t_2[Y] \neq t_3[Y]$. Therefore $t_1[X] = t_3[X]$. Also,

$t_1[Y] \neq t_3[Y]$ because otherwise t_1 and t_3 would be the same tuple. So (t_1, t_3) is an edge in $G_{d,r}$. □

Lemma 10. *If R is in BCNF and F is equivalent to a set of FDs with k dependencies, then F is equivalent to a set of FDs with at most k partition dependencies.*

Proof. We build a set of partition dependencies equivalent to F by replacing every nontrivial dependency $d = X \rightarrow Y$, $d \in F$, by the partition dependency $X \rightarrow U - X$. □

Therefore, in the case $|F| = 2$ we can assume that $F = \{d_1, d_2\}$ where d_1 and d_2 are different partition dependencies. (The case of $|F| = 1$ has already been shown to be in PTIME, even without the BCNF assumption.) Note that Lemma 10 does not have to hold for arbitrary FDs.

We consider now the first class of graphs for which maximum independent set can be computed in PTIME: *claw-free* graphs. Since repairs correspond to maximal independent sets in the conflict graph, the size of a maximum independent set provides the lub-answer to a `COUNT(*)` aggregation query.

Definition 15. A graph is *claw-free* if it does not contain an induced subgraph (V_0, E_0) where $V_0 = \{t_1, t_2, t_3, t_4\}$ and $E_0 = \{(t_2, t_1), (t_3, t_1), (t_4, t_1)\}$.

Lemma 11. *If R is in BCNF over $F = \{d_1, d_2\}$, then for every instance r of R, the conflict graph $G_{\{d_1,d_2\},r}$ is claw-free.*

Proof. Assume that the conflict graph contains a claw (V_0, E_0) where $V_0 = \{t_1, t_2, t_3, t_4\}$ and $E_0 = \{(t_2, t_1), (t_3, t_1), (t_4, t_1)\}$. Then two of the edges in E_0, say (t_2, t_1) and (t_3, t_1) come from one of $G_{d_1,r}$ or $G_{d_2,r}$. By Lemma 9, the edge (t_3, t_2) also belongs to that graph, and consequently to $G_{\{d_1,d_2\},r}$. Thus the subgraph induced by V_0 is not a claw. □

We note that it can also be shown that the conflict graph is perfect in this case [5]. (A graph is *perfect* if its chromatic number is equal to the size of its maximum clique.)

Theorem 12. *If the relational schema R is in BCNF and the given set of FDs F is equivalent to one with at most two dependencies, computing $lub_F^{\texttt{COUNT(*)}}(r)$ in any instance r of R can be done in PTIME.*

Proof. The theorem follows from Lemma 11 and the fact that a maximum independent set in a claw-free graph can be found in polynomial time [28,26]. □

We show now the second approach that directly yields an $O(n^{1.5})$ complexity bound.

Theorem 13. *If the relational schema R is in BCNF and the given set of FDs F is equivalent to one with at most two dependencies, computing $lub_F^{\texttt{COUNT(*)}}(r)$*

in any instance r of R can be done in $O(n^{1.5})$ *time where n is the number of tuples in r.*

Proof. Suppose that $G_{d_1,r}=(V,E_1)$ and $G_{d_2,r}=(V,E_2)$. Then $G_{\{d_1,d_2\},r}=(V,E_1\cup E_2)$. By Lemma 9, both $G_{d_1,r}=(V,E_1)$ and $G_{d_2,r}=(V,E_2)$ are unions of disjoint cliques. Let $U_1,U_2,\dots,U_{k_1}$ be the cliques in $G_{d_1,r}$. Let $W_1,W_2,\dots,W_{k_2}$ be the cliques in $G_{d_2,r}$.

In order to find a maximum independent set in $G_{\{d_1,d_2\},r}=(V,E_1\cup E_2)$, we construct a bipartite graph $H=(U\cup W,E_H)$ as follows: $U=\{u_1,u_2,\dots,u_{k_1}\}$ and $W=\{w_1,w_2,\dots,w_{k_2}\}$. For each vertex $v\in V$, v is in exactly one clique U_i and in exactly one clique W_j. We add an edge (u_i,w_j) into E_H. H contains only these edges.

A *matching* of H is a subset $M\subseteq E_H$ such that no two edges in M share a common end vertex. The crucial observation is that the independent sets in $G_{\{d_1,d_2\},r}=(V,E_1\cup E_2)$ one-to-one correspond to the matchings in H. To see this, first note that the vertices of $G_{\{d_1,d_2\},r}$ one-to-one correspond to the edges of H. Consider two vertices x,y in $G_{\{d_1,d_2\},r}$. Suppose that $e=(x,y)$ is an edge in $G_{\{d_1,d_2\},r}$. Without loss of generality, we may assume $e\in E_1$. Then both x and y are in the same clique U_i. Hence, the two edges in H corresponding to x and y share a common vertex u_i in H. Conversely, suppose that x and y are not adjacent in $G_{\{d_1,d_2\},r}$. Then x and y are in different cliques in $G_{\{d_1\},r}$, say they are in U_i and $U_{i'}$, where $i\neq i'$, respectively. Similarly, x and y are in different cliques in $G_{\{d_2\},r}$, say they are in W_j and $W_{j'}$, where $j\neq j'$, respectively. Thus, the edge (u_i,v_j) in H corresponding to x and the edge $(u_{i'},v_{j'})$ in H corresponding to y share no common end vertex in H. Therefore, a subset S of vertices in $G_{\{d_1,d_2\},r}$ is an independent set if and only its corresponding edge set is a matching in H. Hence, finding a maximum independent set in $G_{\{d_1,d_2\},r}$ is equivalent to finding a maximum matching in the bipartite graph H. This can be done in $O((|U|+|W|)^{1/2}|E_H|)$ time by using the algorithm in [20]. Since $|U|\leqslant n, |W|\leqslant n$ and $|E_H|=n$, the total time needed is $O(n^{1.5})$. □

We show now that more than two FDs, even in BCNF, push the problem of computing lub-answers beyond tractability.

Theorem 14. *If the relational schema R is in BCNF and the given set of FDs F is equivalent to one with three dependencies, the data complexity of computing $lub_F^{\mathtt{COUNT(*)}}(r)$ in an instance r of R is* NP-*hard.*

Proof. Let d_1,d_2,d_3 be three partition dependencies. As before, the graph $G_{d_i,r}=(V,E_i)$ $(1\leqslant i\leqslant 3)$ is a union of disjoint cliques. We also have $G_{\{d_1,d_2,d_3\},r}=(V,E_1\cup E_2\cup E_3)$. Our problem is equivalent to finding a maximum independent set in $G_{\{d_1,d_2,d_3\},r}$.

To show the problem is NP-hard, we reduce the 3-dimensional matching (3DM) problem [14] to it. The 3DM problem is defined as follows:

An instance of 3DM is a tuple (X,Y,Z,M), where X,Y,Z are three disjoint sets of the same cardinality, and $M\subseteq X\times Y\times Z$. A *matching* of the instance is a subset $M'\subseteq M$ such that no two elements in M' agree in any coordinate. The goal is to determine the existence of a (maximum) matching of size $|X|$.

Given an instance (X,Y,Z,M) of 3DM, we construct a graph $D=(V_D,E_D)$ as follows: $V_D=M$. Suppose $X=\{x_1,x_2,\ldots,x_t\}$. Partition M into $M_1,\ldots,M_t$ such that $M_i=\{(x_i,y,z)\in M\}$ (for $1\leqslant i\leqslant t$). For each i $(1\leqslant i\leqslant t)$, we add a clique into E_D whose vertices are exactly the triples in M_i. Denote the set of the edges added this way by E_X. Note that the graph (V_D,E_X) is a union of disjoint cliques. Similarly, we perform the same action for Y and Z, and let E_Y and E_Z be the sets of the edges added, respectively. We set $E_D=E_X\cup E_Y\cup E_Z$.

Note that the maximum matchings of the instance (X,Y,Z,M) one-to-one correspond to the maximum independent sets of the graph D. Also note that $D=G_{\{d_1,d_2,d_3\},r}$ for the instance $r=M$ and partition dependencies $d_1=A\rightarrow BC$, $d_2=B\rightarrow AC$, and $d_3=C\rightarrow AB$, where ABC is the schema of r. Thus, there is a maximum matching of size $|X|$ iff $lub^{\mathtt{COUNT(*)}}_{\{d_1,d_2,d_3\}}(r)=|X|$. This completes the reduction. □

5.2. Other tractable cases

There are other, simpler cases where the conflict graph has a structure that makes it possible to determine the cardinality of a maximum independent set in PTIME.

Theorem 15. *If an instance r is the disjoint union of two instances that separately satisfy F, the data complexity of computing $lub^{\mathtt{COUNT(*)}}_F(r)$ is in PTIME.*

Proof. In this case, the only conflicts are between the parts of r that come from different instances. Thus, the conflict graph is a bipartite graph. For bipartite graphs determining the cardinality m of a maximum independent set can be done in PTIME. This follows from the fact that $m=n-k$ where n is the number of vertices in the graph and k is the cardinality of the minimum vertex cover. The latter is equal to the cardinality of the maximum matching in the graph (König–Egervary Theorem [23]). □

Note that the assumption in Theorem 15 is satisfied when the instance r is obtained by merging together two consistent databases in the context of database integration.

Theorem 16. *If every tuple in an instance r is in conflict with at most two tuples in the same instance, the data complexity of computing $lub^{\mathtt{COUNT(*)}}_F(r)$ is in PTIME.*

Proof. In this case, each vertex in the conflict graph has degree at most 2, thus the conflict graph is a union of disjoint components each of which is an isolated vertex, a noncyclic path, or a single cycle. Finding the cardinality of a maximum independent set in such a graph can clearly be done in PTIME. □

6. Related and further work

We have provided a complete classification of the tractable/intractable cases of the problem of computing glb- and lub-answers to aggregation queries with scalar functions in the presence of functional dependencies. We have also shown how tractability can be obtained in several special cases and presented a practical hybrid computation method.

We only briefly survey the related work here. A more comprehensive discussion can be found in [3]. The need to accommodate violations of functional dependencies is one of the main motivations for considering disjunctive databases [22,30] and has led to various proposals in the context of data integration [2,6,13,25]. A purely proof-theoretic notion of consistent query answer comes from Bry [7]. This notion, described only in the propositional case, corresponds to our notion of core answer. None of the above approaches considers aggregation queries.

There seems to be an intriguing connection between relation repairs w.r.t. FDs and databases with disjunctive information [30]. For example, the set of repairs of the relation *BrownVotes* from Example 1 can be represented as a disjunctive database D consisting of the formulas

$$BrownVotes(A, 11/07, 541) \vee BrownVotes(A, 11/11, 560)$$

and

$$BrownVotes(B, 11/07, 302).$$

Each repair corresponds to a minimal model of D and vice versa. We conjecture that the set of all repairs of an instance w.r.t. a set of FDs can be represented as a disjunctive table (with rows that are disjunctions of atoms with the same relation symbol). This is not as obvious as it seems, as the repairs require an *exclusive* representation of disjunctions, which is forced through the minimal model semantics of disjunctive formulas. The relationship in the other direction does not hold. E.g., the set of minimal models of the formula

$$(p(a_1, b_1) \vee p(a_2, b_2)) \wedge p(a_3, b_3)$$

cannot be represented as a set of repairs of any set of FDs. However, we are not aware of any work on aggregation in general disjunctive databases (but see below).

The relationship between sets of repairs and databases with OR-objects [22,9] is more complicated.

Example 7. The set of repairs of the relation *BrownVotes* in Example 1 cannot be represented as a table with OR-objects. However, the set of repairs of the projection of *Brown Votes* on the first and third attributes:

County	Tally
A	541
A	560
B	302

can be represented as

County	Tally
A	$OR(541, 560)$
B	302

In the example above, the schema of the relation *BrownVotes* was not in BCNF. But even under BCNF, there is still a mismatch.

Example 8. Consider the following set of FDs $F = \{A \rightarrow B, A \rightarrow C\}$, which is in BCNF. The set of all repairs of the instance $\{(a_1, b_1, c_1), (a_1, b_2, c_2)\}$ cannot be represented as a table with OR-objects.

The relationship in the other direction, from tables with OR-objects to sets of repairs, also does not hold.

Example 9. Consider the following table with OR-objects:

$OR(a, b)$	c
a	$OR(c, d)$

It does not represent the set of all repairs of any instance under any set of FDs.

A correspondence between sets of repairs and tables with OR-objects occurs only in the very restricted case when a relation is binary, say $R(A, B)$, and there is one FD $B \rightarrow A$.

Several people [12,9] studied aggregation in databases with OR-objects. As in our case, the query results in this case are indefinite. The dissertation [12] suggests, like we do, to return ranges of values of the aggregate functions. On the other hand, the paper [9] proposes to return sets of all possible values of such functions. The second approach runs into the problem that the set of possible values may have exponential size, c.f., Example 5. The paper [9] discusses not only scalar aggregation but also aggregation functions (`GROUP BY` in SQL). Possibly, some of the techniques of that paper can be adapted if we extend the present results in that direction. Due to the above-mentioned lack of correspondence between sets of repairs and tables with OR-objects the results from our paper cannot be directly transferred to the context of [9], except in a very restricted case, and vice versa.

Incidentally, the paper [9] incorrectly claims that the greatest lower bound on the value of the aggregate function `COUNT(A)` can be computed in PTIME in tables with OR-objects. This is contradicted by our Theorem 9, which shows in an equivalent setting that checking whether the glb bound is less than or equal to k is an NP-complete problem. The paper [9] provides a greedy PTIME algorithm (Algorithm 3.1) for computing the glb of `COUNT(A)` but the algorithm is incorrect. To see this consider the set of OR-objects $S = \{OR(a,b), OR(a,c), OR(a,d), b, c, d\}$. The algorithm will compute 4 as the lower bound on the number of different values that cover all the OR-objects in S. However, this bound is actually $3 = |\{b,c,d\}|$.

There are several proposals for language constructs specifying nondeterministic queries that are related to our approach (*witness* [1], *choice* [15,16,18]). Essentially, the idea is to construct a maximal subset of a given relation that satisfies a given set of functional dependencies. Since there is usually more than one such subset, the approach yields nondeterministic queries in a natural way. Clearly, maximal consistent subsets (choice models [15]) correspond to repairs. Datalog with choice [15] is, in a sense, more general than our approach, since it combines enforcing functional dependencies with inference using Datalog rules. Answering queries in all choice models ($\forall G$-queries [18]) corresponds to our notion of computation of consistent query answers for first-order queries (Definition 5). However, in [18] the former problem is shown to be co-NP-complete and no tractable cases are identified. One of the sources of complexity in this case is the presence of Datalog rules, absent from our approach. Moreover, the procedure proposed in [18] runs in exponential time if there are exponentially many repairs, as in Example 5. Also, only conjunctions of literals are considered as queries in [18]. Arbitrary first-order or aggregation queries are not studied.

As mentioned earlier, the paper [3] contains a general method for transforming first-order queries in such a way that the transformed query computes the consistent answers to the original query. In that paper, soundness, completeness and termination of the transformation are studied, and some classes of constraints and queries for which consistent query answers can be computed in PTIME are identified. Representing repairs as stable models of logic programs with disjunction and classical negation has been proposed in [4,17]. Those papers consider computing consistent answers to first-order queries (but not to aggregation queries). No tractable cases beyond those of [3] are identified in [4,17], which is not surprising in view of Corollary 2.

Many further questions suggest themselves. First, is it possible to identify more tractable cases and to reduce the degree of the polynomial in those already identified? Second, is it possible to use approximation in the intractable cases? The INDEPENDENT SET problem is notoriously hard to approximate [19], but perhaps the special structure of the conflict graph may be helpful. Finally, it would be very interesting to see if our approach can be generalized to broader classes of queries and integrity constraints. In most implementations of SQL2, only functional dependencies in BCNF are supported (using `PRIMARY KEY` and `UNIQUE` constraints). Therefore, the approaches described in Section 5 may be applicable there. It is not obvious, however, how to generalize our approach to broader classes of queries. Is it possible to combine the approach of this paper with that of [3]?

There is some recent work done on rewriting aggregation queries in terms of *aggregation views* [29,10,11]. It would be interesting to explore how to take advantage of those results when computing consistent answers to aggregation queries. Another possible avenue is to consider *aggregation constraints* [24,27].

Finally, alternative definitions of repairs and consistent query answers that include, for example, preferences are left for future work. Also, one can apply further aggregation to the results of aggregation queries in different repairs, e.g., the average of all `MAX(A)` answers.

Acknowledgements

Work supported in part by FONDECYT Grant 1000593, NSF grant IIS-0119186, and NSF grant INT-9901877/CONICYT Grant 1998-02-083. The comments of the anonymous referees are gratefully acknowledged.

References

[1] S. Abiteboul, R. Hull, V. Vianu, Foundations of Databases, Addison-Wesley, Reading, MA, 1995.

[2] S. Agarwal, A.M. Keller, G. Wiederhold, K. Saraswat, Flexible relation: an approach for integrating data from multiple, possibly inconsistent databases, in: IEEE Internat. Conf. on Data Engineering, Taipei, Taiwan, 1995.

[3] M. Arenas, L. Bertossi, J. Chomicki, Consistent query answers in inconsistent databases, in: ACM Symp. on Principles of Database Systems, Philadelphia, PA, 1999, pp. 68–79.

[4] M. Arenas, L. Bertossi, J. Chomicki, Specifying and querying database repairs using logic programs with exceptions, in: Internat. Conf. on Flexible Query Answering Systems, Springer, Berlin, 2000, pp. 27–41.

[5] M. Arenas, L. Bertossi, J. Chomicki, Scalar aggregation in FD-inconsistent databases, in: Internat. Conf. on Database Theory, Lecture Notes in Computer Science, Vol. 1973, Springer, Berlin, 2001, pp. 39–53.

[6] C. Baral, S. Kraus, J. Minker, V.S. Subrahmanian, Combining knowledge bases consisting of first-order theories, Comput. Intell. 8 (1992) 45–71.

[7] F. Bry, Query answering in information systems with integrity constraints, in: IFIP WG 11.5 Working Conf. on Integrity and Control in Information Systems, Chapman & Hall, London, 1997.

[8] A.K. Chandra, D. Harel, Computable queries for relational databases, J. Comput. System Sci. 21 (1980) 156–178.

[9] A.L.P. Chen, J-S. Chiu, F.S.C. Tseng, Evaluating aggregate operations over imprecise data, IEEE Trans. Philadelphia, PA, Knowledge Data Engrg. 8 (2) (1996) 273–284.

[10] S. Cohen, W. Nutt, A. Serebrenik, Rewriting aggregate queries using views, in: Proc. ACM PODS'99, 1999, pp. 155–166.

[11] S. Cohen, W. Nutt, A. Serebrenik, Algorithms for rewriting aggregate queries using views, in: Proc. Symp. on Advances in Databases and Information Systems, ADBIS-DASFAA' 2000, Prague, September, 2000, pp. 65–78.

[12] L.G. De Michiel, Performing database operations over mismatched domains, Ph.D. Thesis, Stanford University, 1989.

[13] P.M. Dung, Integrating data from possibly inconsistent databases, in: Internat. Conf. on Cooperative Information Systems, Brussels, Belgium, 1996.

[14] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP Completeness, Freeman, New York, 1979.

[15] F. Giannotti, S. Greco, D. Sacca, C. Zaniolo, Programming with non-determinism in deductive databases, Ann. Math. Artificial Intell. 19 (3–4) (1997) 97–125.

[16] F. Giannotti, D. Pedreschi, Datalog with non-deterministic choice computes NDB-PTIME, J. Logic Programming 35 (1998) 75–101.

[17] G. Greco, S. Greco, E. Zumpano, A logic programming approach to the integration, repairing and querying of inconsistent databases, in: P. Codognet (Ed.), Internat. Conf. on Logic Programming, Lecture Notes in Computer Science, Vol. 2237, Springer, Berlin, 2001, pp. 348–364.

[18] S. Greco, D. Sacca, C. Zaniolo, Datalog queries with stratified negation and choice: from P to D^P, in: G. Gottlob, M.Y. Vardi (Eds.), Internat. Conf. on Database Theory, Springer, Berlin, 1995, pp. 82–96.

[19] D.S. Hochbaum, Approximating covering and packing problems: set cover, vertex cover, independent set, and related problems, in: D.S. Hochbaum (Ed.), Approximation Algorithms for NP-Hard Problems, PWS Publishing Co., MA, 1997.
[20] J.E. Hopcroft, R.M. Karp, An $O(n^{5/2})$ algorithm for maximum matchings in bipartite graphs, SIAM J. Comput. 2 (1973) 225–231.
[21] T. Imieliński, W. Lipski, Incomplete information in relational databases, J. ACM 31 (4) (1984) 761–791.
[22] T. Imieliński, S. Naqvi, K. Vadaparty, Incomplete objects—a data model for design and planning applications, in: J. Cliff, R. King (Eds.), ACM SIGMOD Internat. Conf. on Management of Data, Denver, Colorado, May 1991, pp. 288–297.
[23] E.L. Lawler, Combinatorial Optimization: Networks and Matroids, Holt, Rinehart and Winston, New York, 1976.
[24] A.Y. Levy, I.S. Mumick, Reasoning with aggregation constraints, in: Proc. EDBT, Avignon, France, 1996, pp. 514–534.
[25] J. Lin, A.O. Mendelzon, Merging databases under constraints, Internat. J. Cooperative Inform. Syst. 7 (1) (1996) 55–76.
[26] G.J. Minty, On maximal independent sets of vertices in claw-free graphs, J. Combin. Theory B 28 (1980) 284–304.
[27] K.A. Ross, D. Srivastava, P.J. Stuckey, S. Sudarshan, Foundations of aggregation constraints, Theoret. Comput. Sci. 193 (1–2) (1998) 149–179.
[28] M. Sbihi, Algorithme de Recherche d'un Stable de Cardinalité Maximum dans un Graphe sans Étoile, Discrete Math. 29 (1980) 53–76.
[29] D. Srivastava, S. Dar, H.V. Jagadish, A.Y. Levy, Answering queries with aggregation using views, in: Proc. VLDB'96, Bombay, India, 1996, pp. 318–329.
[30] R. van der Meyden, Logical approaches to incomplete information: a survey, in: J. Chomicki, G. Saake (Eds.), Logics for Databases and Information Systems, Chap. 10, Kluwer Academic Publishers, Boston, 1998.
[31] M.Y. Vardi, The complexity of relational query languages, in: ACM Symp. on Theory of Computing, San Francisco, CA, 1982, pp. 137–146.