On the tractability of certain answers for SQL nulls in relational algebra with inequalities

Etienne Toussaint

School of Informatics, University of Edinburgh

Abstract. Missing values in theoretical models of incomplete database are often represented with marked nulls, while in SQL databases missing values are all denoted by the same syntactic `NULL` object. Even practical algorithm to approximate certain answers (answers which are true regardless of how incomplete information is interpreted) are often developed in the model with marked nulls. However computing certain answers for marked nulls is co-NP complete even for the most simple queries when inequalities are allowed. In this short paper we study the tractability of certain answers for SQL nulls in a fragment of relational algebra where selection with inequalities is permitted. We define the fragment and present an algorithm to compute certain answers. We also show that if we add even small features to the fragment, computing certain answers becomes intractable. This study emphasises the necessity of a specific certain answers approximation scheme for SQL nulls and offers ideas to design it.

1 Introduction

The standard way of answering queries on incomplete databases is to compute certain answers: those that do not depend on the interpretation of unknown data. However, evaluating certain answers for core relational algebra is co-NP complete in data complexity [9]. As a consequence the community has developed sound tractable approximation schemes (which return a subset of the certain answers). Most of those schemes have been produced with the marked nulls model of incompleteness [3,7], and it is well known that even the simplest query with inequalities is intractable for marked nulls [1]. However nulls used in SQL databases are different. In this paper we study a fragment of relational algebra with inequalities for which computing certain answers for SQL nulls is tractable. This demonstrates the complexity gap between those two models of incompleteness and therefore emphasizes the need of a specific approximation scheme for SQL nulls.

We consider incomplete databases with nulls interpreted as missing information [5]. Below we recall definitions that are standard in the literature. Databases are populated by two types of elements: constants coming from countably infinite set denoted by *Const*, and the syntactic object `NULL`. The occurrences of `NULL` in an SQL database are typically interpreted as non-repeating elements of a set Null. That is, an SQL database can be seen as a Codd database where each occurrence of `NULL` is replaced by a fresh distinct marked null [4]. Therefore we denote by $Null(D) = \{\perp_1 \dots \perp_n\}$ the set of distinct marked nulls in the database D.

A valuation v on a database D is a map $v : Null(D) \to Const$ that assigns constant values to nulls occurring in the database. By $v(D)$ we denote the result of replacing each $\perp_i$ with $v(\perp_i)$ in D. A relational query Q of arity k takes a complete database D and returns a bag of k-tuples over $Const(D)$. If such a query Q is asked on an incomplete database D, to answer it we compute for each $\bar{t} \in (Const \cup Null(D))^k$ the bag of certain answers denoted $\Box(Q, D)$ which verify :

$$\#(\bar{t}, \Box(Q, D)) = \min_{v \text{ a valuation}} \#(v(\bar{t}), Q(v(D))).$$

Where

$$\#(\bar{t}, R) = \begin{cases} n \text{ if } \bar{t} \in^n R \\ 0 \text{ if } \bar{t} \notin R \end{cases}$$

and we use $\bar{t} \in^n R$ to say that $\bar{t}$ has a multiplicity n in the bag R [2]. If the multiplicity of a tuple in $\Box(Q, D)$ is equal to 0, this tuple does not belong to the certain answers.

In order to add boolean queries to relational algebra we add the operator $\pi_\emptyset$ such that for any complete database D and any query Q, $\pi_\emptyset(Q)(D) = \emptyset$ if $Q(D) = \emptyset$ and $\pi_\emptyset(Q)(D) = \{()\}$ otherwise.

2 Relational algebra fragment with efficient evaluation

Our goal is to find a fragment for which we can build an efficient algorithm to compute all certain answers. As a motivation we take inspiration from the hierarchical queries from probabilistic databases [8] to define the restricted *hierarchical* relational algebra defined below. We start with two subclasses of unions of *CQs* with inequalities.

$$\begin{aligned} Q_1 &:= Q_1 \cap Q_1 \mid Q_1 \cup Q_1 \mid \sigma_\theta(Q_1) \mid \pi_\alpha(Q_1) \mid R \\ Q_2 &:= Q_2 \cup Q_2 \mid \sigma_\theta(Q_2) \mid \pi_\alpha(Q_2) \mid R \end{aligned}$$

Note that the only difference between the two classes is that Q_1 allows intersection while Q_2 does not. We denote by $\mathbf{Q_1}$ resp. $\mathbf{Q_2}$ the set of query induced by the grammar Q_1 resp. Q_2. Based on them we define the class RA^H :

$$Q_0 := Q_0 \cup Q_0 \mid Q_1 \setminus Q_2 \mid \sigma_\theta(Q_0) \mid \pi_\alpha(Q_0) \mid \pi_\emptyset(Q_0) \mid R$$

A relational algebra query is called *non-repeating* if every relation symbol occurs at most once [8]. We denote $RA^{H,NR} = \{Q \mid Q \in RA^H \wedge Q \textit{ is non-repeating}\}$ the fragment of restricted hierarchical relational algebra where queries are non-repeating.

Our main result is :

Theorem 1. *For every query $Q \in RA^{H,NR}$ and every database D computing $\Box(Q, D)$ is tractable.*

We now outline the proof of Theorem 1. As the restricted hierarchical fragment of relational algebra imposes that every binary operation in selection conditions is between attributes of the same relation or constants, then for each query in the fragment we can build an equivalent query where selection occurs only on relation symbols.

Lemma 1 *For every $Q \in RA^H$ there exists $Q' \in RA^H$ such that $|Q'| = \mathcal{O}(|Q|)$ and for every complete database D, $\Box(Q, D) = \Box(Q', D)$ and every selection operator of Q' occurs on a relation symbol.*

Now we show that for $Q_1 \in \mathbf{Q_1}$ computing $\Box(Q_1, D)$ is tractable. One starts by applying the lemma 1 to push every selection operator. Its queries are now given by : $Q_1 := Q_1 \cap Q_1 \mid Q_1 \cup Q_1 \mid \sigma_\theta(R) \mid \pi_\alpha(Q_1) \mid R$. Then computation is done inductively by the following rules :

$$\begin{aligned}
\#(\bar{t}, \Box(Q \cap Q', D)) &= min(\#(\bar{t}, \Box(Q, D)), \#(\bar{t}, \Box(Q', D))) \\
\#(\bar{t}, \Box(Q \cup Q', D)) &= \#(\bar{t}, \Box(Q, D)) + \#(\bar{t}, \Box(Q', D)) \\
\#(\bar{t}, \Box(R, D)) &= \#(\bar{t}, R) \\
\#(\bar{t}, \Box(\pi_\alpha(Q), D) &= \sum_{\overline{u}, \pi_\alpha(\overline{u}) = \bar{t}} \#(\overline{u}, \Box(Q, D)) \\
\#(\bar{t}, \Box(\sigma_\theta(R), D)) &= \begin{cases} \#(\bar{t}, R) & \text{if } \forall v \ \textit{a valuation}, v(\bar{t}) = \bar{t} \wedge \theta(\bar{t}) \\ 1 & \text{if } \exists v \ \textit{a valuation}, v(\bar{t}) \neq \bar{t} \wedge \theta \ \textit{is a valid formulae} \\ 0 & \text{otherwise} \end{cases}
\end{aligned}$$

Therefore for every query $Q \in \mathbf{Q_1}$, the evaluation of $\Box(Q, D)$ is tractable. The computation rules above are sound and complete only because the relations hence the nulls can not repeat.

Then we show how to evaluate $\Box(Q_0, D)$. Informally in order to evaluate Q_0 one has to compute the possible answers of Q_2 which match an element of Q_1. But first notice that as intersection is not allowed in Q_2 we can push the projection operators :

Lemma 2 *For every $Q \in \mathbf{Q_2}$ there exists $Q' \in \mathbf{Q_2}$ such that $|Q'| = \mathcal{O}(|Q|)$ and for every complete database D, $Q(D) = Q'(D)$ and every projection operator of Q' occurs on a relation symbol, or on a selection operator over a relation symbol.*

Then from lemmas 1 and 2 we just have to consider queries of the form: $Q_0 = Q_1 \setminus \bigcup_{R \in Q_2} \pi_{\alpha_R}(\sigma_{\theta_R}(R))$ *with* $Q_1 \in \mathbf{Q_1} \wedge Q_2 \in \mathbf{Q_2}$

For every $R \in Q_2, \forall \bar{t} \in R$ we build a set:

$$\Downarrow_{Q_1, R} (\bar{t}) = \{\overline{u} \in \Box(Q_1, D) \mid \exists v \ \textit{a valuation}, \theta_R(v(\bar{t})) \wedge \pi_{\alpha_R}(v(\bar{t})) = v(\overline{u})\}$$

As the relations hence the nulls can not repeat, $\Box(Q_1, D)$ can be computed independently of Q_2 and is at most of the size of D, then for each R, the set

$\Downarrow_{Q_1,R}$ can be built in polynomial time.
Moreover for every $\overline{u} \in \Box(Q_1, D)$ we build a bag:

$$\forall \overline{t} \in (Const \cup Null)^*, \#(\overline{t}, \Uparrow_{Q_2}(\overline{u})) = \sum_{R \in \{R \in Q_2 | \overline{u} \in \Downarrow_{Q_1,R}(\overline{t})\}} \#(\overline{t}, R)$$

Here, $\Uparrow_{Q_2}(\overline{u})$ is the bag of elements in $\bigcup\limits_{R \in Q_2} R$ which unify with $\overline{u} \in \Box(Q_1, D)$.
Then a tuple $\overline{u}$ belongs to certain answers of Q_0 if and only if the multiplicity of $\overline{u}$ in $\Box(Q_1, D)$ is higher than the number of elements in $\Uparrow_{Q_2}(\overline{u})$.

Proposition 1 *Let $Q_0 = Q_1 \setminus \bigcup_{R \in Q_2} \pi_{\alpha_R}(\sigma_{\theta_R}(R))$ with $Q_1 \in \mathbf{Q_1} \wedge Q_2 \in \mathbf{Q_2}$. Then $\#(\overline{u}, \Box(Q_0, D)) = max(0, |\Uparrow_{Q_2}(\overline{u})| - \#(\overline{u}, \Box(Q_1, D)))$, and the evaluation of $\Box(Q_0, D)$ is tractable.*

However there exists a query Q_0 such that $\forall \overline{u}, \#(\overline{u}, \Box(Q_0, D)) = 0$ and $\Box(\pi_\emptyset(Q_0), D)) \neq \emptyset$. In order to compute $\Box(\pi_\emptyset(Q_0), D)$, we want to check if there exists a matching between $\Box(Q_1, D))$ and the elements that unify with it.

Proposition 2 *Let $Q_0 = Q_1 \setminus \bigcup_{R \in Q_2} \pi(\sigma_{\theta_R}(R))$ with $Q_1 \in \mathbf{Q_1} \wedge Q_2 \in \mathbf{Q_2}$. Then $\Box(\pi_\emptyset(Q_0), D) = \emptyset$ if and only if there exists an injective function $m : \Box(\pi_\emptyset(Q_1), D) \to \bigcup\limits_{R \in Q_2} R$ such that $\forall \overline{t} \in \Box(\pi_\emptyset(Q_1), D), m(\overline{t}) \in \Uparrow_{Q_2}(\overline{t})$.
m is a 2DM matching, and the evaluation of $\Box(\pi_\emptyset(Q_0), D)$ is tractable [6].*

3 Extending the fragment

In this section we discuss the difficulties of extending the fragment $R^{H,NR}$ to obtain tractable evaluation of certain answers.

Proposition 3 *For each of the following extensions of $R^{H,NR}$:*

- *allowing cross-product.*
- *allowing repetition of relation symbols.*
- *allowing intersection on the right-hand side of the $Q_1 \setminus Q_2$ operator.*
- *allowing difference on the right-hand side of the $Q_1 \setminus Q_2$ operator.*

the data complexity of evaluating certain answers is co-NP hard.

4 Conclusions

In this paper we have exhibited a fragment for which computing certain answers for SQL nulls is tractable. We have also shown that adding features to the fragment quickly lead to intractability. The next question that arises is toward maximality, in order to find a dichotomic property for certain answers with SQL nulls one would have to consider equivalently expressive classes of query. As soon as we fully understand what leads to intractability we will be able to design a more accurate approximation scheme for SQL nulls.

References

1. Serge Abiteboul, Paris Kanellakis, and Gösta Grahne. On the representation and querying of sets of possible worlds. *Theoretical Computer Science*, 78(1):159–187, 1991.
2. Marco Console, Paolo Guagliardo, and Leonid Libkin. On querying incomplete information in databases under bag semantics. IJCAI.
3. Paolo Guagliardo and Leonid Libkin. Correctness of SQL Queries on Databases with Nulls. *ACM SIGMOD Record*, 46(3):5–16, 2017.
4. Paolo Guagliardo and Leonid Libkin. On the Codd semantics of SQL nulls. *Alberto Mendelzon Workshop*, 36t, 2017.
5. Tomasz Imieliński and Witold Lipski Jr. Incomplete information in relational databases. *Journal of the ACM (JACM)*, 31(4):761–791, 1984.
6. Eugene L Lawler. *Combinatorial optimization: networks and matroids*. Courier Corporation, 1976.
7. Leonid Libkin. SQL's three-valued logic and certain answers. *ACM Transactions on Database Systems (TODS)*, 41(1):1, 2016.
8. Dan Suciu, Dan Olteanu, Christopher Ré, and Christoph Koch. Probabilistic databases. *Synthesis Lectures on Data Management*, 3(2):1–180, 2011.
9. Moshe Y Vardi. The complexity of relational query languages. In *Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pages 137–146. ACM, 1982.