Queries with Arithmetic on Incomplete Databases
Marco Console, Matthias Hofer, Leonid Libkin

To cite this version:

HAL Id: hal-03127717
https://inria.hal.science/hal-03127717
Submitted on 1 Feb 2021
ABSTRACT
The standard notion of query answering over incomplete database is that of certain answers, guaranteeing correctness regardless of how incomplete data is interpreted. In majority of real-life databases, relations have numerical columns and queries use arithmetic and comparisons. Even though the notion of certain answers still applies, we explain that it becomes much more problematic in situations when missing data occurs in numerical columns.

We propose a new general framework that allows us to assign a measure of certainty to query answers. We test it in the agnostic scenario where we do not have prior information about values of numerical attributes, similarly to the predominant approach in handling incomplete data which assumes that each null can be interpreted as an arbitrary value of the domain. The key technical challenge is the lack of a uniform distribution over the entire domain of numerical attributes, such as real numbers. We overcome this by associating the measure of certainty with the asymptotic behavior of volumes of some subsets of the Euclidean space. We show that this measure is well-defined, and describe approaches to computing and approximating it. While it can be computationally hard, or result in an irrational number, even for simple constraints, we produce polynomial-time randomized approximation schemes with multiplicative guarantees for conjunctive queries, and with additive guarantees for arbitrary first-order queries. We also describe a set of experimental results to confirm the feasibility of this approach.

1 INTRODUCTION
Handling incomplete or uncertain information is one of the central topics in databases, as well as in multiple applications of databases where uncertainty naturally arises. These applications include data integration [25], data exchange [5], ontology-based data access [7, 30], consistent query answering [6], probabilistic databases [31], and others. The standard approach to query answering in all of these is to find answers with some certainty guarantees [19]. Most commonly, one looks for certain answers that are true under all possible interpretations of missing data, although weaker guarantees are possible. In the probabilistic model, for example, we can talk about the probability of a tuple being an answer [31], and can even do this in the absence of explicit distributions [27].

There is one element common to all these approaches. They all start by the standard assumption prevalent in the database literature, which essentially says: assume that there is one domain of values $U$ from which all database elements are drawn. In reality however database columns are typed, and one would need to look hard for a database that does not have numerical columns and arithmetic operations in queries. For example, in the TPC-H benchmark [32], 7 out of 8 tables have numerical attributes, while over 85% of queries use at least order comparisons, and over 70% of queries use standard arithmetic operations such as addition and multiplication. Modeling extra types theoretically is done by viewing relational databases as many-sorted structures. Usually this is a mild assumption for theoretical work, unless there are specific interpreted operations on different types, such as for instance $+$, $\cdot$, $<$ on numerical types.

The expressive power and complexity of such languages was extensively studied in the 1990s; see [24] for a summary. For handling incomplete information, however, very little is known. We do know however that the problem becomes considerably harder. For example, under the single-domain assumption, unions of conjunctive queries on databases with nulls can be evaluated efficiently if we only compare values for equality [19], but already in the presence of order comparisons data complexity jumps to coNP, cf. [1]. With more complicated arithmetic, the complexity may jump even higher, all the way to undecidability (we shall give an example later).

At this point we should ask ourselves: what kind of query answers we would actually want in such scenarios. Look at a very simple example: a query $\sigma_{A>B}(R)$ on relation $R$ with attributes $A$ and $B$ and a single tuple $(\perp_1, \perp_2)$ with two nulls. Should the tuple be selected? If we know nothing about $\perp_1$ and $\perp_2$, it seems reasonable to say that with probability 1/2 the tuple will be in the answer. We now give an example showing the utility of such answers.
Example. A team of sale analysts is asked to predict the effectiveness of the forthcoming sales campaign using data from a database with three relations:

- Products(id, seg, rrp, dis) has product ids, their market segment, recommended retail price (rrp), and intended discount dis;
- Competition(id, seg, p) contains information about competing products on the market including their ids, market segment, and offered price p;
- Excluded(id, seg) has products excluded from the campaign.

As often happens in any business environment, information in these tables may be incomplete while the details of the campaign are being worked out. Furthermore, Competition is likely to be populated by an (automated) web extraction algorithm, leading to a high chance of incomplete data in that relation. A sales analyst may want to compile a list of market segments where the company will have a strong competitive advantage, i.e., the price, after discount, of all the products in that segment is below the competition. This is shown in the picture below. Excluded cases are intended to be worked out. Furthermore, Competition is likely to be populated by an (automated) web extraction algorithm, leading to a high chance of incomplete data in that relation. A sales analyst may want to compile a list of market segments where the company will have a strong competitive advantage, i.e., the price, after discount, of all the products in that segment is below the competition. The fraction depends on r, but as r increases, one can calculate that it tends to \( \approx 0.388 \) of the positive quadrant, which can be taken as our estimate. Putting a higher discount (e.g., changing 0.7 to 0.5) results in approximately half of the positive quadrant satisfying the constraint, thus increasing our confidence in s being an answer. □

Our goal is to provide a framework in which we can reason about answers in the same manner as in the example. Note that above we did not make any assumptions about probability distributions on data stored in numerical attributes. That is a deliberate assumption, for this initial study. We would like the framework to be general enough to accommodate additional information such as distributions or constraints on data. But for now we follow what most of the work on usual incomplete data in relational databases does. There, the standard assumption is that we have a domain of values, and every null can be interpreted as any of the elements of the domain, cf. [19]. We take it further by assuming that we now have a second numerical domain, and specifying that some columns take values in that numerical domain. This is similar to stating attribute types as int or float in create table statements. Then we let nulls in numerical columns be interpreted as arbitrary elements of the numerical domain.

Before outlining our contributions, we remark that dealing with missing numerical data is common in data analysis, and typically is dealt with via imputation, i.e., finding suitable replacements for missing data, cf. [14]. There are many variations of the approach that go from simple discarding of tuples with missing values to the inference of missing values through complex statistical models. Crucially though once the imputation is done, the data set is treated as complete, and if it resides in a database, it will be queries using standard techniques for querying complete data. Query answers then will have no way of referring to the fact that some data was initially missing. Our goal is different though: we want to query an incomplete database as given to us, with the full knowledge that it is incomplete, and provide the user with the additional information about confidence levels for potential query answers. Moreover, we want to do it even in the case when there is no prior information about data, such as probability distribution.

Our main contributions are as follows.

1. We define a model of relational databases with attributes of two distinct types – a base type and a numerical type – with nulls that may occur as values of attributes of both types. We use, as a query language, an extension of relational calculus (FO) with arithmetic operations and order comparisons. We define a measure \( \mu(q, D, \alpha) \) of certainty of a tuple \( \alpha \) as an answer to query \( q \) on database \( D \). It takes values in the interval \([0, 1]\). Value 1 means that the answer is almost surely
We now briefly recall the standard model of incomplete information, where incompleteness is represented by values. They may also come from a domain $\mathbb{N}$ from a domain $\mathbb{C}$, often viewed as the domain of constants, or known values. They may also come from a domain $\mathbb{N}$ of nulls, which are often viewed as currently unknown values. A database schema is a set of relation names and their arities. In a database $D$, a $k$-ary relation $R$ from the schema is interpreted as a $k$-ary relation over $\mathbb{C} \cup \mathbb{N}$, i.e., a finite set $R^D \subseteq (\mathbb{C} \cup \mathbb{N})^k$. Often $D$ is clear from the context, and we then write $R$ in place of $R^D$. We write $C(D)$ and $N(D)$ for the set of all elements of $C$ and $N$ that appear in relations of $D$.

Incomplete databases are interpreted by means of valuations. A valuation is a map $\nu : N(D) \rightarrow \mathbb{C}$ that interprets nulls by constants. Thus, an incomplete database $D$ represents complete (i.e., having no nulls) databases $\nu(D)$, where $\nu$ ranges over valuations, and $\nu(D)$ is the result of replacing each null $\perp \in N(D)$ with $\nu(\perp)$.

Suppose $q$ is a $n$-ary query that, on each complete database $D$ returns $q(D) \subseteq C(D)^n$. The standard notion of query answering is that of certain answers [19], i.e., answers independent of the valuation of nulls. For a database $D$ that now may have nulls, an $n$-tuple $\bar{a}$ over $C(D) \cup N(D)$ is a certain answer if $\nu(\bar{a}) \in q(\nu(D))$ for each valuation $\nu$.

This is the definition from [28]. One typically sees a restricted form of it when $\bar{a}$ has no nulls; then it says $\bar{a} \in q(\nu(D))$, [19]. We use the more permissive definition of [28], as it allows tuples with nulls to be in the answer. Its advantages are explained, for example, in [26]. As a simple example, if we have a relation $R$ with a single tuple $(1, \perp)$, and a query returning $R$, then the definition of [19] will return $\emptyset$, forgetting certain information that we have one tuple in $R$, and its first component is 1. The definition of [28] that we use here will, on the other hand, return $(1, \perp)$.

It is well-known that certainty comes at a computational price: while for conjunctive queries and their unions and some small extensions with restricted negation they can be computed efficiently [15, 16, 19], for first-order queries (or full relational algebra queries) finding certain answers is coNP-hard in data complexity. This cost however can be significantly reduced if we are only interested in almost certain answers, using a simple idea from [27].

Assume the domain of constants comes with some enumeration, $C = \{c_1, c_2, c_3, \ldots \}$. For a valuation $\nu : N(D) \rightarrow \mathbb{C}$, define $\|\nu\| = \max \{r \mid c_r \text{ is in the range of } \nu\}$.

Next, for a query $q$ and a tuple $\bar{a}$, define the $r$-support of a tuple $\bar{a}$ as an answer to $q$ as the set of valuations $\nu$ with $\|\nu\| \leq r$ such that $\nu(\bar{a}) \in q(\nu(D))$, i.e.,

$$\text{Supp}_r(q, D, \bar{a}) = \{\nu \mid \nu(\bar{a}) \in q(\nu(D)) \text{ and } \|\nu\| \leq r\}.$$  

We then take $\mu_r(q, D, \bar{a})$ to be the probability that a valuation $\nu$ with $\|\nu\| \leq r$ picked uniformly at random is in $\text{Supp}_r(q, D, \bar{a})$. As there are only $\|N(D)\|$ valuations $\nu$ with $\|\nu\| \leq r$ on a database $D$, this is well defined. Finally, to get rid of the dependence on $r$, one simply looks at the limit as $r$ grows and thus the set of valuations with $\|\nu\| \leq r$ becomes close to the set of all valuations; that is, $\mu(q, D, \bar{a}) = \lim_{r \rightarrow \infty} \mu_r(q, D, \bar{a})$.

Then, [27] established the following results:

- The value of $\mu(q, D, \bar{a})$ does not depend on a particular enumeration of $C$.
- For a large class of queries the value of $\mu(q, D, \bar{a})$ is either 0 or 1. These queries are generic, i.e., commuting with permutations of their domain. All queries in languages such as FO, relational algebra, datalog, etc., are generic.
We now move to a model where different columns in relations can where we allow for nulls as entries.

$C$ variable is typed (i.e., of base or numerical type). Its terms and formulae are defined inductively as follows.

$D$ of base type that occur in relation of type $R$ is a subset of the real numbers $\mathbb{R}$. The domain of base type is denoted by $C$ base, and the domain of the numerical type by $C$ num. Thus, what we previously viewed as the set of constants $C$ is now a disjoint union of $C$ base and $C$ num. We assume that the domain of the numerical type $C$ num is a subset of the real numbers $\mathbb{R}$.

A schema then, instead of simply specifying the arity of each relation, specifies types of columns. We assume that such specifications are of the form $R(\text{base}^k \text{num}^m)$ saying that the first $k$ columns of $R$ are of base type, and the remaining $m$ are of the numerical type. Such columns, as declared in any real-life DDL, can be interspersed, but here we assume, only for the simplicity of the notation, that all base type columns come first.

Each column, whether of base type or numerical, may contain null values. Using the same marked null model, we assume two sets of nulls: $N$ base = \{⊥, 1, . . .\} of nulls that occur in base type columns, and $N$ num = \{⊤, 1, 2, . . .\} of numerical type nulls. Then a relation of type $R(\text{base}^k \text{num}^m)$ in a database $D$ is interpreted as a finite set

$$R^D \subseteq (C\text{ base} \cup N\text{ base})^k \times (C\text{ num} \cup N\text{ num})^m.$$ 

That is, entries of each base type column come from $C$ base ∪ $N$ base and entries of each numerical type column come from $C$ num ∪ $N$ num. We use the same notations to denote sets of elements of each kind in a database $D$: for example, $C$ base$(D)$ is the set of all constants of base type that occur in $D$, and $N$ num$(D)$ is the set of all nulls in numerical columns that occur in $D$.

Query languages. As our basic query language, we consider two-sorted first-order logic with arithmetic, FO(+, <). In this logic, every variable is typed (i.e., of base or numerical type). Its terms and formulae are defined inductively as follows.

Terms • a variable of a base type is a base type term;
• a variable of a numerical type is a numerical type term;
• every element $a \in C$ num is a numerical type term;
• if $t$ and $t'$ are numerical type terms, then so are $t + t'$ and $t \cdot t'$.

Atomic formulae • for a relation $R(\text{base}^k \text{num}^m)$, if $\bar{x}$ is a $k$-tuple of base type variables and $\bar{f}$ is an $m$-tuple of numerical type terms, then $R(\bar{x}, \bar{f})$ is an atomic formula;
• if $x$ and $y$ are variables of the base type, then $x = y$ is an atomic formula;
• if $t$, $t'$ are numerical type terms, then $t < t'$ and $t = t'$ are atomic formulae.

Formulae are closed under Boolean connectives and quantifiers:
• if $\varphi, \psi$ are formulae then $\varphi \lor \psi, \varphi \land \psi$ and $\neg \varphi$ are formulae;
• if $\varphi$ is a formula then $\exists x \varphi$ and $\forall x \varphi$ are formulae.

We shall use the standard shortcuts, i.e., $x \leq y$ for $x < y \lor x = y$ or $2x$ and $x^2$ for $x + x$ and $x \cdot x$, etc. Since atomic formulae are of the form $t = t'$ or $t < t'$, we can assume that operations $–$ and $\div$ are allowed for building terms as well.

The notion of free variables and the semantics are standard. When queries are interpreted over complete databases $D$, quantifiers range over their domain. That is, $\exists x$ means that a witness is found among elements of $C$ base$(D)$ if $x$ is of base type, and among elements of $C$ num$(D)$ if $x$ is of numerical type. For example, $\exists y, z \in S(x, y, z)$ is a numerical type term, then $t = = t$ and $t < t'$ are allowed.

Notice that with +, we also have subtraction: for example, $t_1 - t_2 < t_3$ is $t_1 < t_2 + t_3$.

Sublanguages. If we restrict the use of arithmetic, we explicitly indicate so; for example, FO(+, <) means that arithmetic terms are linear functions, built with +, and comparisons $t < t'$ are allowed. Notice that with +, we have addition: for example, $t_1 - t_2 < t_3$ is $t_1 < t_2 + t_3$.

We shall refer to the $\exists, \land$-fragment of the language as conjunctive queries and write CQ() listing allowed operations in parentheses. For example, CQ(<) refers to conjunctive queries extended with order comparisons.

4 QUERY ANSWERS: MEASURE OF CERTAINTY

We now describe the approach to query answering in the presence of arithmetic and nulls. As the first step, we need to provide the semantics of incomplete databases over two types. Recall that under the single-type assumption, the semantics was given by valuations, which are functions from nulls to constants $C$. Now that the set of constants is two-sorted, such valuations on an incomplete database $D$ are pairs $v = (v$ base, $v$ num) where $v$ base : $N$ base$(D) \rightarrow C$ base and $v$ num : $N$ num$(D) \rightarrow C$ num. The database $v$($D$) is, as before, obtained by substituting each null $\bot \in N$ base$(D)$ with $v$ base$($$\bot$$)$ and each null $\top \in N$ num$(D)$ with $v$ num$(\top)$. Likewise, when we have a tuple $\bar{a}$, we write $v$($\bar{a}$) to denote a tuple obtained from $\bar{a}$ by replacing base type nulls according to $v$ base, numerical type nulls according to $v$ num, and leaving constants intact. For example, $v$($\bot, 1, 2$) = ($v$ base$(\bot), v$ num$(1), 2$).

Of course with this, we could define certain answers as before: a tuple $\bar{a}$ is a certain answer to query $q$ if $v$($\bar{a}$) $\in q$($v$($D$)) for each valuation $v$. However, it is well known that even adding simple order comparisons $< \alpha$ already leads to much higher complexity of query answering. For example, finding certain answers to conjunctive queries and their unions in the single-type scenario is well-known to be the same as query evaluation [19], i.e., $AC^0$ in data complexity and NP in combined complexity. However, with order comparisons, the complexity of the problem jumps coNP in data complexity and $\Pi_2^p$ in combined complexity [1, 21, 35]. Order comparisons lead to further complications when considered in typical scenarios.
involving certain answers such as integrating and exchanging data [3, 4].

When arithmetic operations are added, the situation is even worse; in fact we have the following easy observation.

**Proposition 4.1.** If $C_{num}$ is $\mathbb{Z}$, there is a CQ$(+, \cdot, <)$ query $q$ such that computing certain answers to $q$ is undecidable, even on a database of a fixed schema, with a single relation consisting of a single tuple.

**Proof Sketch.** Indeed, consider a polynomial $p \in \mathbb{Z}[x_1, \ldots, x_k]$, a relation $R(num^k)$ with a single tuple $(\top_1, \ldots, \top_k)$, and $q$ given by $\exists x R(x) \land p^2 > 0$. Then, a certain answer to this query is true iff $p$ has no integer roots. Since it is known that the problem of finding solutions to Diophantine equations is undecidable even for a fixed number of variable (in fact $k = 13$ [29]), the result follows. \square

Thus, it is even more justified in the case of numerical constraints to pass from absolute certainty to measures of certainty, as was done in [27]. This is what we do now.

**Measure of certainty.** Now consider a query $q(x, y)$, where $x$ are of base type, $y$ are of numerical type, and let $D$ be a database. Let $\bar{a}$ be a tuple of base type constants and nulls from $D$ of the same arity as $x$, and let $\bar{s}$ be a tuple of numerical type constants and nulls from $D$ of the same arity as $y$. Our goal is to define a measure $\mu(q, D, (\bar{a}, \bar{s}))$ of the likelihood of a tuple $(\bar{a}, \bar{s})$ being an answer to $q$ on $D$.

Given a database $D$, let its base type nulls be $N_{base}(D) = \{1, \ldots, m\}$ and its numerical type nulls be $N_{num}(D) = \{1, \ldots, r\}$. Then, for a valuation $v = (v_{num}, v_{base})$, we have

\[
v(N_{base}(D)) \in C_{base}^m \quad v(N_{num}(D)) \in C_{num}^r \subseteq \mathbb{R}^k\]

since we assume always that $C_{num} \subseteq \mathbb{R}$.

As we did in the previous section, assume that we have an enumeration of base type constants $C_{base}$ as $\{c_1, c_2, \ldots\}$ (we shall see soon that the choice of a particular enumeration does not affect the measure we define). When we had only one non-numerical domain we looked at the proportion of valuations $v$ with $\|v\| \leq r$ that witness $v(\bar{a}) \in q(v(D))$, and then analyzed its asymptotic behavior. Here we do essentially the same. However, notice that $v(N_{num}(D))$ is now a vector in $\mathbb{R}^k$, and thus there are infinitely many valuations $v$ with $\|v(N_{num}(D))\| \leq r$, where $\|\|$ refers to the usual (Euclidean) norm in $\mathbb{R}^k$. Thus, to measure the number of valuations, instead of cardinality we now look at their volume, denoted by $Vol$.

Thus, we mimic the definition from the previous section. Define

\[
\text{Supp}_r(q, D, (\bar{a}, \bar{s})) = \left\{ v \mid v(\bar{a}) \in q(v(D)) \land \|v_{base}\| \leq r \land \|v_{num}\| \leq r \right\} ,
\]

where of course $v = (v_{base}, v_{num})$, and where we write $\|v_{base}\|$ for $\|v_{base}(N_{base}(D))\|$ and likewise for $\|v_{num}\|$ when $D$ is clear from the context. Next, the idea is essentially the same as before: define the measure of certainty as the probability that a valuation picked uniformly at random belongs to $\text{Supp}_r(q, D, (\bar{a}, \bar{s}))$. There is a problem though: the set of valuations $v_{base}$ is discrete and there are infinitely many valuations $v_{base}$ with $\|v_{base}\| \leq r$, while for valuations $v_{num}$ we need to measure their volume. We thus need to combine the two in one measure somehow.

To do this, we relativize $\text{Supp}_r(q, D, (\bar{a}, \bar{s}))$ for each fixed valuation $v_{base}$ on base-type nulls. That is, we define $\text{Supp}_r(q, D, (\bar{a}, \bar{s}) \mid v_{base})$ as

\[
\{v_{num} \mid (v_{base}, v_{num}) \in \text{Supp}_r(q, D, (\bar{a}, \bar{s}))\} .
\]

Since we associate $v_{num}$ with $\text{Supp}_r(N_{num}(D)) \subseteq \mathbb{R}^k$, we have $\text{Supp}_r(q, D, (\bar{a}, \bar{s}) \mid v_{base}) \subseteq \mathbb{R}^k$ and thus, if it is measurable, we use its volume as its measure. Then we define

\[
\text{Vol}_r(q, D, (\bar{a}, \bar{s})) = \frac{\text{Vol}(\text{Supp}_r(q, D, (\bar{a}, \bar{s}) \mid v_{base}))}{\text{Vol}(B^k_r)}
\]

as the combined measure of $\text{Supp}_r(q, D, (\bar{a}, \bar{s}))$ taking into account the discrete nature of valuations on base type and continuous measure of valuations on the numerical type.

Of course as $r$ grows, so can $\text{Vol}_r(q, D, (\bar{a}, \bar{s}))$, since vectors $v_{num}(N_{num}(D))$ come from $B^k_r$, the $k$-dimensional ball of radius $r$. So we normalize this measure, to make it a number between 0 and 1 that determines the likelihood of $(\bar{a}, \bar{s})$ being an answer. For that, we divide by the largest possible value of $\text{Vol}_r(q, D, (\bar{a}, \bar{s}))$. We have at most $|r|^m$ valuations $v_{base}$ with $\|v_{base}\| \leq r$ (we take $|r|$ if $r$ is not an integer), and for each of them $\text{Supp}_r(q, D, (\bar{a}, \bar{s}) \mid v_{base}) \subseteq B^k_r$ and thus its volume is bounded by $\text{Vol}(B^k_r)$. Summing up, the maximum value of $\text{Vol}_r(q, D, (\bar{a}, \bar{s}))$ is $\text{Vol}(B^k_r) \cdot |r|^m$ and thus we define the likelihood as

\[
\mu_r(q, D, (\bar{a}, \bar{s})) = \frac{\text{Vol}_r(q, D, (\bar{a}, \bar{s}))}{\text{Vol}(B^k_r) \cdot |r|^m} \in [0, 1] \quad (2)
\]

to be this normalized value.

Finally, as before, we look at the asymptotic behavior of this measure:

\[
\mu(q, D, (\bar{a}, \bar{s})) = \lim_{r \to \infty} \mu_r(q, D, (\bar{a}, \bar{s})) . \quad (3)
\]

**Convention** For Boolean queries $q$, we shall write $\mu(q, D)$ instead of the more formal $\mu(q, D, (\))$, i.e., omitting the empty tuple of arguments.

**Remark** What if there are no numerical variables? In this case $\text{Supp}_r(q, D, (\bar{a}, \bar{s}) \mid v_{base})$ is a subset of $\mathbb{R}^k$, i.e., just a point. If we assume that $\text{Vol}(B^k_0) = 1$, then in this case $\mu(q, D, (\))$ can be easily seen to coincide with the measure defined in [27] and presented in the previous section. Thus we indeed provide a proper generalization of the framework of measuring certainty of answers to the case of numerical domains.

**5 THE MEASURE IS WELL DEFINED**

We provided a formal definition of the measure of certainty $\mu(q, D, (\bar{a}, \bar{s}))$ as a number in the interval $[0, 1]$, indicating how likely a tuple $(\bar{a}, \bar{s})$, consisting of elements of base and numerical types, is to be an answer to $q$ on an incomplete database $D$. The definition however took for granted the existence of certain quantities, and thus we need to prove that $\mu$ is actually well defined. Specifically, there are two elements of the definition of $\mu$ where well-definedness needs to be formally proved.

- We defined measure $\mu$ using volumes of sets $\text{Supp}_r(q, D, (\bar{a}, \bar{s}) \mid v_{base}) \subseteq \mathbb{R}^k$. But even though $\text{Supp}_r(q, D, (\bar{a}, \bar{s}) \mid v_{base})$ is a bounded set, as a subset of $B^k_r$, a priori it does not mean that it is measurable and the volume is defined.
• We then defined $\mu$ as the limit of some ratio (see equations (2) and (3)) as $r$ grows. However, it is not clear a priori that the limit always exists.

Our goal now is to prove that $\mu$ is well-defined for all queries in FO$(+,-,<)$. In the process of showing this, we also reduce the problem of computing $\mu(q, D, (\bar{a}, \bar{s}))$ to the problem of analyzing asymptotic behavior of formulae in the first-order theory of the reals. This connection will be essential for obtaining complexity results.

The existence of volumes. For this, we need to prove that sets $\text{Supp}_r(q, D, (\bar{a}, \bar{s}) \mid v_{\text{base}}) \subseteq \mathbb{R}^k$ are Lebesgue-measurable.

**Proposition 5.1.** For every query $q \in FO(+,-,<)$, database $D$, tuples $\bar{a}$ and $\bar{s}$, and valuation $v_{\text{base}}$ of base type nulls in $D$, the set $\text{Supp}_r(q, D, (\bar{a}, \bar{s}) \mid v_{\text{base}})$ is Lebesgue-measurable, i.e., its volume exists.

**Proof sketch.** To see this, simply put in the definition of each relation $R$ in $v_{\text{base}}(D)$ (as an explicit disjunction of tuples) in $q$ to see that $\text{Supp}_r(q, D, (\bar{a}, \bar{s}) \mid v_{\text{base}})$ is definable in FO theory of the real field $(\mathbb{R}, +, <)$. The latter, by cellular decomposition, is a finite union of cells, each of which is open and thus measurable, cf. [34].

The existence of limits. We prove this in two steps. First, we restate the definition of $\mu$ by fully eliminating its dependence on values over the base type. Then we show how to eliminate the database and reduce the question to the existence of limits of volumes of some definable sets over the real field $(\mathbb{R}, +, <)$, which is actually known.

**Eliminating base type.** We now show that to compute the measure, we can completely disregard values $v_{\text{base}}$. Recall that when there is no numerical type, $\mu(q, D, \bar{a}) \in \{0, 1\}$, and $\mu(q, D, \bar{a}) = 1$ iff $\bar{a}$ is returned by the naive evaluation of $q$ on $D$. This observation makes it possible to eliminate base type values from the consideration of the asymptotic behavior of $\mu_r$.

For $q, D$, and a tuple $\bar{a}$, we say that a valuation $v_{\text{base}}$ is bijective (with respect to $q, D, \bar{a}$) if $v_{\text{base}}$ is a bijection and its range is disjoint from $C_{\text{base}}(D)$. Then it easy to see that for sufficiently large $r$ and two bijective valuations $v_{\text{base}}$ and $v'_{\text{base}}$ we have

$$\text{Vol}(\text{Supp}_r(q, D, (\bar{a}, \bar{s}) \mid v_{\text{base}})) = \text{Vol}(\text{Supp}_r(q, D, (\bar{a}, \bar{s}) \mid v'_{\text{base}})).$$

Furthermore, as $r$ grows, almost all (asymptotically) valuations on base type become bijective. This easily leads to the following proposition.

**Proposition 5.2.** For a query $q$, a database $D$, and tuples $\bar{a}, \bar{s}$ of values of free variables of $q$, let $v_{\text{base}}$ be a bijective valuation with respect to $q, D, \bar{a}$. Then

$$\mu(q, D, (\bar{a}, \bar{s})) = \lim_{r \to \infty} \frac{\text{Vol}(\text{Supp}_r(q, D, (\bar{a}, \bar{s}) \mid v_{\text{base}}))}{\text{Vol}(B_r^k)}.$$  

**Proof sketch.** Using Proposition 5.2 and the observation that $\text{Supp}_r(q, D, (\bar{a}, \bar{s}) \mid v_{\text{base}}) = \text{Supp}_r(q, v_{\text{base}}(D), (v_{\text{base}}(\bar{a}), \bar{s}))$, we conclude that for computing $\mu(q, D, (\bar{a}, \bar{s}))$ we only need to look at databases $D$ without base type nulls. And once we have such databases, we can further reduce the problem to that for formulae that do not even mention database.

**Eliminating database relations.** We now look at databases that have only numerical nulls. Consider such a database $D$ with $N_{\text{num}}(D) = \{1, \ldots, k\}$, an FO$(+,-,<)$ query $q$, and tuples $\bar{a}$ over $C_{\text{base}}(D)$ and $\bar{s}$ over $C_{\text{num}}(D) \cup N_{\text{num}}(D)$ of the same length as $\bar{a}$ and $\bar{s}$ respectively. Let $R$ stand for the structure $(\mathbb{R}, +, <)$, i.e., the real closed field.

**Proposition 5.3.** Under the assumptions above, there exists, and can be constructed in time polynomial in the size of $D$, a quantifier-free formula $\varphi_{q, D, \bar{a}, \bar{s}}(z_1, \ldots, z_k)$ over $R$ such that $\forall \bar{z} \in \varphi_{q, D, \bar{a}, \bar{s}}(z_1, \ldots, z_k)$ iff $\varphi(q, \bar{a}, \bar{s}) \in q(\bar{v}(D))$, where $\bar{v}$ assigns $z_i$ to each $T_i$ for $i \leq k$.

**Proof sketch.** To see this, first associate with each numerical variable $\bar{z}$ a new variable $\bar{z}$. Then replace all existential quantifiers $\exists \bar{z}$ over base type variables by explicit disjunction $\bigvee v \in C_{\text{num}}(D)$ and quantifiers $\exists \bar{z}$ over numerical type variables by $\bigvee v \in C_{\text{num}}(D) \cup N_{\text{num}}(D)$. In the resulting formula all atomic relational formulae are of the form $R(\bar{e}, \bar{s})$, where $\bar{e}$ is a tuple of base type constants, and $\bar{s}$ is a tuple of numerical type constants and nulls of some length $p$. We then replace such formula by $\bigvee \bigwedge_{i=1}^{k} e_i = c_i$ if the disjunction is taken over all tuples $(\bar{e}, \bar{s})$ that occur in relation $R$ in $D$ and $c_i$ for $c_i \in \text{Num}$, while $\bigvee_{i=1}^{k} R_i(\bar{e}, \bar{s})$. Next, using standard arithmetic, we can assume that each atom $t < t'$ is given as a Boolean combinations of statements $p(\bar{e}) \{=, <, \neq\}$, where $p, p'$ are polynomials. We then again replace such formula by $p(u_1, \ldots, u_p)\{=, <, \neq\} p'(u_1, \ldots, u_p)$, resulting in the desired formula $\varphi(\bar{z})$. Verifying correctness of the translation is routine.

Thus, the set of all valuations $v$ witnessing $\varphi(\bar{a}, \bar{s}) \in q(\bar{v}(D))$ is defined by a formula over $R$. Now consider an arbitrary formula $\varphi(\bar{z})$ with $|\bar{z}| = k$ over $R$ and define

$$\nu(\varphi) = \lim_{r \to \infty} \frac{\text{Vol}(\varphi(\bar{z}) \cap B_r^k)}{\text{Vol}(B_r^k)},$$

where $\varphi(\mathbb{R}^k) = \{s \in \mathbb{R}^k \mid R \models \varphi(\bar{a})\}$. That is, the ratio $\text{Vol}(\varphi(\mathbb{R}^k) \cap B_r^k) / \text{Vol}(B_r^k)$ shows the proportion of the $k$-dimensional ball of radius $r$ occupied by vectors satisfying $\varphi$, and $\nu(\varphi)$ defines the asymptotic behavior of this ratio.

Next we use a known fact, namely that $\nu(\varphi)$ exists for every formula $\varphi$ over $R$. It was shown in [11], using results of [20, 22], how to definably approximate volumes of sets definable in $R$. Combining it with previous results, we obtain the main result of this section.

**Theorem 5.4.** For an FO$(+, -, <)$ query $q$, a database $D$, and tuples $\bar{a}, \bar{s}$ of values of free variables of $q$, the measure $\mu(q, D, (\bar{a}, \bar{s}))$ is well-defined and its value is a number in the interval $[0, 1]$. Moreover, one can construct in polynomial time in the size of $D$ a formula $\varphi(\bar{z})$ over $R$, where $|\bar{z}|$ is the number of numerical nulls in $D$, such that

$$\mu(q, D, (\bar{a}, \bar{s})) = \nu(\varphi),$$

where $\varphi$ is given
which shows that for $r > 8$, the ratio

$$\frac{\text{Vol}(\varphi(\mathbb{R}^2) \cap B_r^2)}{\text{Vol}(B_r^2)}$$

is in the interval

$$\left[\frac{\left(\frac{\pi}{2} - \arctan\left(\frac{10}{r}\right)\right)}{2\pi} - \frac{8}{\pi r}, \frac{\left(\frac{\pi}{2} - \arctan\left(\frac{10}{r}\right)\right)}{2\pi}\right]$$

which shows that $v(\varphi) = \left(\frac{\pi}{2} - \arctan\left(\frac{10}{r}\right)\right)/2\pi \approx 0.097$, or, if we are only interested in the proportion of the positive quadrant satisfying this constraint, it is four times that, i.e., $\approx 0.388$ as shown in the Introduction.

6 APPROACHES TO COMPUTING AND APPROXIMATING THE MEASURE

Now that we have the measure of certainty defined, the question is how to compute it. This would be an algorithm that computes a value of a function in the interval $[0, 1]$, rather than a decision problem.

Of course one can attempt to compute the function $\mu$. We shall see though that there are two obstacles along the way. First, the value of $\mu$ may be irrational and thus needs to be approximated. Second, computing the value of $\mu$, even if it is rational, may be computationally hard. Ideally, we would like it to be in FP, the class of functions computable in polynomial time. However, we will see that even for simple queries, the problem could be hard for FP$^P$, the class of functions computable in polynomial time with calls to a $^P$ and thus intractable oracle.

Starting with the first issue we show the following.

**Proposition 6.1.** Consider the Boolean query $q = \exists x, y R(x, y)$ and a database $D$ in which $R$ has a single tuple $\langle T, T' \rangle$. Then $\mu(q, D) \in \mathbb{Q}$ iff $\alpha = 0, \pm 1$.

**Proof idea.** Indeed, a simple calculation shows that $\mu(q, D) = \arctan(\alpha)/2\pi + 1/2$, and then one can show that it is irrational except for $\alpha = 0$ or $\alpha = \pm 1$. $\Box$

In some cases, we can guarantee rationality of $\mu(q, D)$ but then the complexity jumps.

**Proposition 6.2.** For queries in FO$(<)$, the value $\mu(q, D)$ is always rational. However, there are queries which computing $\mu(q, D)$ for queries from CQ$(<)$ is FP$^P$-hard in data complexity.

**Proof sketch.** Following [13], computing the rational number $\frac{n}{m} = \mu(q, D)$ is FP$^P$-hard if we can find a query $q$ and databases $D$ such that computing the denominator $m$ is done in polynomial time, and computing the numerator $n$ is $^P$-hard. We can show how to use a CQ$(<)$ query to compute the number of satisfying assignments to a 3DNF in $k$ variables which is known to be $^P$-hard [33]; the denominator in this case is $2^k$, the number of all assignments, and thus can be computed efficiently in binary. $\Box$

These propositions show that it is necessary to look for approximation schemes. Towards that, let us recall some basic definitions, cf. [36]. Given a problem of computing a function $f(x)$, we say that a randomized algorithm $A(x, \epsilon)$ is an FPRAS (fully polynomial-time randomized approximation scheme) for it if $A$ runs in time polynomial in the size of $x$ and $1/\epsilon$, and the value $\text{output}[A(x, \epsilon)]$ it produces satisfies

$$(1 - \epsilon)f(x) \leq \text{output}[A(x, \epsilon)] \leq (1 + \epsilon)f(x),$$

with probability at least $\frac{3}{4}$. The confidence level $\frac{3}{4}$ can be changed to any arbitrary value $1 - \delta$ for $\delta \in (0, 1)$ by running $A$ polynomially many times over the same input; alternatively $\delta$ can be set up as a parameter of the FPRAS and the algorithm $A$ then needs to run in time that is also polynomial in $\log(1/\delta)$.

However, even with order constraints, we cannot get an FPRAS for general FO queries, under a widely believed complexity-theoretic assumption.

**Theorem 6.3.** If $\text{NP} \not\subseteq \text{BPP}$, then there is no FPRAS for computing $\mu(q, D)$ for FO$(<)$ queries.

**Proof sketch.** In this case, we produce an FO query $q \in \text{FO}(<)$ and, for each 3CNF formula $\psi$ with $n$ variables, a database $D_\psi$ such that $\mu(q, D_\psi) \approx \frac{\#\psi}{2^n}$, where $\#\psi$ is the number of satisfying assignments to $\psi$. In turn, this proves that $\psi$ is satisfiable if and only if $\mu(q, D_\psi) > 0$, and therefore the associated decision problem is NP-hard. This observation together with a standard complexity theoretic argument allows us to conclude that, if an FPRAS for $\mu(q, D)$ exists, then $\text{NP} \not\subseteq \text{BPP}$. $\Box$

**Alternatives.** Negative results shown earlier provide us with clear limitations as to what our options could be:

- We could try to find an FPRAS for conjunctive queries, as nothing in our results precludes that; or
- We can try to find a weaker form of approximations for arbitrary FO($+, \cdot, <$) queries.

This is exactly what we do. In the next section we prove the existence of an FPRAS for CQ$(+, <)$, conjunctive queries with linear constraints. Then, for a weaker form of approximation, we note that the function $\mu$ has its values in the interval $[0, 1]$. Thus, unlike FPRAS that provides multiplicative error guarantees, for small values of $\epsilon$ we can also settle for a randomized algorithm $A(q, D, (\bar{a}, \bar{s}))$ that returns an answer with additive error guarantees, i.e.,

$$|\mu(q, D) - \text{output}[A(q, D, (\bar{a}, \bar{s}))]| < \epsilon.$$  

We provide such an algorithm for all FO($+, \cdot, <$) queries. The algorithm is polynomial in data complexity, i.e., it runs in time polynomial in the size of $D$ and $1/\epsilon$. We will then show experimentally that its performance is feasible not only in theory but also in practice.

7 FPRAS FOR CONJUNCTIVE QUERIES

While, in the general case, computing the likelihood of an answer is hard and cannot be efficiently approximated, in the important restricted case of conjunctive queries with linear constraints approximation is still possible. These queries are essentially joins of functions computable in polynomial time. However, even for simple queries, the problem could be hard for FP$^P$, the class of functions computable in polynomial time with calls to a #P and thus intractable oracle.

We can try to find a weaker form of approximations for FO($+, \cdot, <$) queries. The algorithm is polynomial in data complexity, i.e., it runs in time polynomial in the size of $D$ and $1/\epsilon$. We will then show experimentally that its performance is feasible not only in theory but also in practice.
Given a query \( q \in \text{CQ}(+, <) \), we define the function problem \( \mu_q(D, (\bar{a}, \bar{s})) \) that asks for the values of \( \mu(q, D, (\bar{a}, \bar{s})) \), with input \( D \) and \( (\bar{a}, \bar{s}) \). In other words, this reformulation corresponds to the function problem of computing \( \mu \) with respect to data complexity.

**Theorem 7.1.** Let \( q(\bar{x}, \bar{y}) \) be a query in \( \text{CQ}(+, <) \). There exists an FPRAS for \( \mu_q(D, (\bar{a}, \bar{s})) \), with input database \( D \), tuple \( (\bar{a}, \bar{s}) \), and \( \epsilon \in (0,1) \).

That is, there is a randomized algorithm \( A_\mu(D, \epsilon) \) that runs in time polynomial in \( D \) and \( 1/\epsilon \) and returns a value \( a \) such that the ratio \( a/\mu(q, D, (\bar{a}, \bar{s})) \) is in the interval \([1-\epsilon, 1+\epsilon]\).

**Proof Sketch.** To prove this, we take \( q(\bar{x}, \bar{y}) \), a database \( D \) with \( n \) numerical nulls, and tuples \( \bar{a}, \bar{s} \) and use Theorem 5.4 to produce a formula \( \psi(\bar{z}) \) with \( n \) free variables over \( R \) so that \( \mu(q, D, (\bar{a}, \bar{s})) = v(\psi) \). To get to the setting where we can define an FPRAS for \( \mu_q(D, (\bar{a}, \bar{s})) \), we need to analyze this translation in a bit more detail. Recall that first we had to apply a bijective valuation \( \nu_{\text{base}} \) to \( D \) to get base type nulls out of the way. Since this can be produced in linear time, we can just assume that such a valuation already was applied, and neither \( D \) nor \( \bar{a} \) have base type nulls.

Next, analyzing the construction of \( \varphi \) given in Proposition 5.3 and given the fact that it is applied to a conjunctive query where existential quantifiers translate into disjunctions and the rest of the formula is a conjunction of predicates, we observe the following about formula \( \varphi \):

- \( \varphi \) is a formula over \( \langle R, +, < \rangle \), i.e., it does not use multiplication;
- \( \varphi \) is of the form \( \bigvee_i \bigwedge_j \beta_{ij}(\bar{z}) \) where each \( \beta_{ij}(\bar{z}) \) is an atomic formula over \( \langle R, +, < \rangle \), i.e., of the form \( \bar{c} \cdot \bar{z} \in [c'] \), where \( \bar{c} \cdot \bar{z} \) is the scalar product, \( \bar{c} \in R^n \) and \( c' \in R \).

For such a formula \( \varphi \), let \( \bar{\varphi} \) denote its homogenized version, i.e., the formula obtained from \( \varphi \) by replacing each atomic formula of numerical type \( \bar{c} \cdot \bar{z} < c' \) by \( \bar{c} \cdot \bar{z} < 0 \). Then we know (see [11]) that

\[
v(\varphi) = \frac{\text{Vol}(\varphi(\mathbb{R}^n) \cap B_0^n)}{\text{Vol}(B_0^n)}.
\]

Next, using the special form of \( \varphi \), we see that \( \bar{\varphi}(\mathbb{R}^n) \) is a finite union of sets \( X_1, \ldots, X_m \), corresponding to the disjuncts of \( \varphi \), and each such set is convex, as the intersection of half-spaces defined by \( \bar{c} \cdot \bar{z} < 0 \). Furthermore, the intersection of each of these sets \( X_i \) with \( B_0^n \) is convex.

While the problem of computing the volume of a union of convex sets is intractable, in [9] it is shown that it admits an FPRAS under the assumptions that each of the sets is given by an individual membership oracle that, for a set \( X \subseteq \mathbb{R}^n \) and a point \( \bar{a} \in \mathbb{R}^n \), tests whether \( \bar{a} \in X \). Since sets are represented as conjunctions of linear constraints and constraints \( ||\bar{a}|| \leq r \), we have these polynomial-time oracles and can apply the FPRAS of [9] to approximate both \( \text{Vol}(\bar{\varphi}(\mathbb{R}^n) \cap B_0^n) \) and \( \text{Vol}(B_0^n) \). It is then an easy observation that we have an FPRAS for their ratio, i.e., \( v(\varphi) \), and thus for \( \mu_q(D, (\bar{a}, \bar{s})) \), completing the proof. \( \square \)

**8 FIRST-ORDER QUERIES AND ADDITIVE ERROR APPROXIMATIONS**

In Section 7, we proved the existence of an FPRAS for conjunctive queries with linear constraints. This algorithm has two drawbacks:

it is limited to conjunctive queries in order to reduce them to a formula of a specific shape, and then it uses that formula in combination with computational geometry algorithms that may not be best suited for implementation inside a DBMS. In contrast, we now prove that, if one settles for an additive error, it is possible to devise an efficient algorithm for all first-order queries with polynomial constraints. This algorithm has the advantage that it is natural to implement, as will be shown in Section 9 that presents an initial experimental evaluation of our approach.

First, we define what it means to have an additive fully polynomial-time randomized approximation scheme (AFPRAS) for a problem of computing a function \( f(\bar{x}, \epsilon) \). An AFPRAS is an algorithm \( A(\bar{x}, \epsilon) \) that produces a value \text{output}[A(\bar{x}, \epsilon)] \) in time polynomial in \( \bar{x} \) and \( 1/\epsilon \) such that

\[
f(\bar{x}) - \epsilon < \text{output}[A(\bar{x}, \epsilon)] < f(\bar{x}) + \epsilon
\]

holds with probability at least \( 3/4 \).

Once again, we are interested in data complexity, and thus the problem we analyze is that of computing the function \( \mu_q(D, (\bar{a}, \bar{s})) \), as in the previous section: for a fixed query \( q(\bar{x}, \bar{y}) \), the input that consists of a database \( D \) and values \( \bar{a} \) for base type variables \( \bar{x} \) and \( \bar{s} \) for numerical type variables \( \bar{y} \), compute \( \mu(q, D, (\bar{a}, \bar{s})) \).

**Theorem 8.1.** If \( q \in \text{FO}(+, -, <) \), then there is an AFPRAS for the function problem \( \mu_q(D, (\bar{a}, \bar{s})) \).

In the rest of the section we explain how to construct this algorithm. As in Section 7, we start with \( q(\bar{x}, \bar{y}) \), a database \( D \) with \( n \) numerical nulls, and tuples \( \bar{a}, \bar{s} \), and use Theorem 5.4 to produce a formula \( \psi(\bar{z}) \) with \( n \) free variables over \( R \) so that \( \mu(q, D, (\bar{a}, \bar{s})) = v(\psi) \). If \( \varphi \) only uses linear constraints, and is in conjunctive normal form, the existence of an AFPRAS follows from results of [11]. Now, however, the query \( q \) is in \( \text{FO}(+, -, <) \) and thus \( \varphi \) could be a formula over \( R \) of arbitrary shape that also uses multiplication, which complicates the search for an AFPRAS considerably.

The idea of an AFPRAS is to test the asymptotic behavior of \( \varphi \) by testing the truth value of \( \varphi \) far away enough from the origin, along different directions. To formalize this, for a formula \( \psi(\bar{z}) \) over \( R \) with \( n \) free variables, and for \( \bar{a} \in \mathbb{R}^n \), define

\[
f_{\psi, \bar{a}}(k) = \begin{cases} 1, & \text{if } R \models \psi(\bar{a} \cdot k) \\ 0, & \text{if } R \not\models \psi(\bar{a} \cdot k) \end{cases}
\]

where \( k \cdot \bar{a} \) is the result of multiplying each component of \( \bar{a} \) by \( k \). Intuitively \( f_{\psi, \bar{a}}(k) \) defines the behavior of \( \varphi \) along the direction \( \bar{a} \). Functions built this way exhibit a monotonic behavior as the following lemma shows.

**Lemma 8.2.** For each formula \( \psi(\bar{z}) \) over \( R \) with \( n \) free variables, and \( \bar{a} \in \mathbb{R}^n \),

\[
\lim_{k \to \infty} f_{\psi, \bar{a}}(k) \in \{0,1\}.
\]

**Proof Sketch.** To prove this, notice that the limit, if it exists, is either 0 or 1, as the function \( f_{\psi, \bar{a}}(k) \) assumes only these values. It remains to argue why \( \lim_{k \to \infty} f_{\psi, \bar{a}}(k) \) exists for all \( \bar{a} \in \mathbb{R}^n \). Each atomic formula of \( \varphi \) is given as a (multivariate) polynomial constraint. Each of these possesses, into each particular direction represented by \( \bar{a} \in \mathbb{R}^n \), finitely many zeros. As a consequence, there is a largest such zero, say \( k^n \), after which all constraints in \( \varphi \) are
We have implemented the additive error approximation algorithm from Section 8. Its main goal was not to produce a fully fledged system for generating a confidence level for queries, but rather to make sure that the proposed theoretical scheme, in addition to having nice theoretical properties, is also feasible in practice.

Specifically, we used this implementation to evaluate a set of join queries with numerical conditions that are fairly typical decision support queries. We evaluated the algorithm for various precision levels ranging from 0.1 to 0.01, essentially corresponding to getting one or two decimal digits correctly. The main parameter to measure was the time overhead required to run the algorithm. The key conclusion is that for common decision queries, and even fairly small $\varepsilon$ (i.e., high confidence level), getting approximate results is feasible at least for medium size databases.

**Implementation.** The implementation of the approximation algorithm itself is done in Python, using a standard library (NumPy) to perform the random sampling. It works on the output of a query, under naive evaluation, produced by the PostgreSQL database, to evaluate confidence level in tuples it generates. Our implementation follows very closely the theoretical algorithm presented in Section 8. Given a conjunctive query $q$ and a database $D$, we first use PostgreSQL to construct a compact representation of the formulae $q_{D, \alpha \delta}$ defined in Section 5. With it in place, we run the Monte-Carlo phase of our algorithm, with only one major difference.

In the theoretical algorithm, given a database $D$ with $n$ different numerical nulls and a query $q$, we sample a vector $\bar{z}$ uniformly at random from the $n$-dimensional unit ball and check the asymptotic behavior of $q$ along the direction of $\bar{z}$. To sample $\bar{z}$ uniformly, we use the standard technique of sampling $n$ independent and normally distributed random variables, and then scale the vector obtained this way to fit the unit ball, see [8].

While this technique is efficient in theory, in practice sampling normally distributed random variables requires the use of a relatively slow Python routine. When there are many nulls in the database, this inevitably leads to poor performances. In our implementation, instead of sampling the whole $\bar{z}$, we only sample as many many coordinates of $\bar{z}$ as needed to replace the nulls that affect the result of the input query. As the final scaling does not affect the direction represented by $\bar{z}$, we can safely use this partial vector for our asymptotic analysis. This optimization saves us a considerable amount of calls to the sampling routine and speeds up the computation substantially.

**Dataset and Queries.** To perform our experiments, we used synthetic data generated for a schema of a sales database resembling the one we used in the introduction. It has the following relations.

- **Product**$(id, \text{seg}, \text{rrp}, \text{dis})$ has product ids, their market segment (seg), recommended retail price (rrp), and the intended discount (dis).
- **Orders**$(id, \text{pr}, q, \text{dis})$ contains information about possible future orders, including the product id (pr), the quantity ordered (q), and the extra discount applied on the order (dis). In this case, the final discount depends on the quantity, and is calculated as $\text{dis}/\text{q}$.
- **Market**$(\text{seg}, \text{rrp}, \text{dis})$ stores the recommended retail price (rrp) and the forecast discount (dis) of the best competing product on the market.

To produce the data, we used DataFiller [10], a tool for the generation of synthetic data. We generated an SQL database of about 200K tuples, with null values. To use this data in our assessment, we replaced each SQL’s null with a different string so to obtain the
equivalent of a database with marked nulls. We then loaded the data into Postgres to perform our evaluation.

We used three typical decision support queries in our experiments. They ask for segments of the market where competitive advantage could be achieved, for having much better offers than the market, and for other companies offering unfair discounts. As is common in such decision support queries, we put a `limit` clause to present the analyst with an analyzable sample from the database. Below we give SQL code and brief descriptions of queries.

- **Competitive Advantage Query**: What are the market segments where the company will have a competitive advantage?
  ```sql
  SELECT P.seg
  FROM Products P, Market M
  WHERE P.seg = M.seg AND
  (P.rrp * P.dis * (O.q/O.dis) <= 0.5 * M.id * M.dis)
  LIMIT 25
  ```

- **Never Knowingly Undersold Query**: What are the products that will sell for less than half of the best price on the market?
  ```sql
  SELECT O.id
  FROM Products P, Orders O, Market M
  WHERE P.id = O.pr AND
  (P.rrp * P.dis * O.q) <= (0.5 * M.id * M.dis)
  LIMIT 25
  ```

- **Unfair Discount Query**: What are the orders that apply a discount at least 60% higher than the intended campaign discount?
  ```sql
  SELECT O.id
  FROM Products P, Orders O, Market M
  WHERE P.id = O.pr AND
  (P.rrp * P.dis * (O.q/O.dis) <= 0.5 * M.id * M.dis)
  LIMIT 25
  ```

**Experimental Setup and Results.** We ran our implementation for each of the above queries, using errors from 0.01 to 0.1 with a step of 0.005. This resulted in 19 different figures for each of the queries. All experiments ran on a machine with an Intel(R) Core(TM) i5-8500 CPU @ 3.00GHz processor, 16GB of RAM, 500GB hard-disk, running Xubuntu 19.04.

The results are shown in Figure 1. Along the x axis, we show error levels $\epsilon$, and along the y axis we show running time of the algorithm. Smaller values of $\epsilon$ correspond to higher precision and thus more sampling and longer running times. If we are interested only in the precision up to the first decimal digit ($\epsilon \in (0, 0.1)$), then our implementation, for all of the queries, runs in less than a second. This is a very small overhead that produces valuable information on the likelihood of tuples returned by the naive implementation of a query, and in particular tells the analyst whether results that may be based on incomplete information warrant further investigation. It is also reasonable to expect that such a level of precision would be acceptable. After all, it should not matter that much whether the likelihood of a tuple is 0.8 or 0.7. As the precision increases, and $\epsilon$ decreases, the performance degrades as expected. But even in the case when the second decimal digits matters ($\epsilon \in (0, 0.01)$), the overhead time required to perform our analysis stays below 10 seconds for every single query.

![Figure 1: Experimental results](image)

### 10 FUTURE WORK

The good theoretical bounds as well as promising results of the experimental evaluation suggest pushing this approach further in at least several directions.

The most important one is to move away from the totally agnostic approach where we say that each numerical null can be interpreted by an arbitrary element of the numerical domain. While this was motivated by the usual assumption in the literature on incomplete databases that nulls can denote arbitrary elements, and as such it was appropriate for this initial study, in real life with typed columns additional constraints are possible. Most commonly we have restrictions on ranges of numerical attributes. For example, price is expected to be positive, while discount is expected to be a number in $[0, 1]$. However the model we proposed here is very easily adaptable to such modifications. We can simply add such constraints in both the numerator and denominator of the ratio defining the measure of certainty. It then needs to be seen how the AFPRAS algorithm is to be adjusted. The model is also suitable for adding probability distributions associated with particular columns, which can simply replace uniform distributions over the $n$-dimensional ball.

So far we measured the likelihoods of answers by computing volumes in $\mathbb{R}^n$. In the case of integer type, we can provide an alternative measure by counting the number of integer lattice points. In general one may expect to obtain similar results as the $n$-dimensional
Acknowledgment

Work partially supported by EPSRC grants M025268, N023056, and S003800. Part of this work was done while the third author was at IRIF, Université de Paris, and DI-ENS, supported by a grant from the Foundation Sciences Mathématiques de Paris.

REFERENCES


