Algorithms for Provisioning Queries and Analytics*

Sepehr Assadi[†1], Sanjeev Khanna[†2], Yang Li[†3], and Val Tannen[‡4]

1 **University of Pennsylvania, Philadelphia, PA, USA**
sassadi@cis.upenn.edu
2 **University of Pennsylvania, Philadelphia, PA, USA**
sanjeev@cis.upenn.edu
3 **University of Pennsylvania, Philadelphia, PA, USA**
yangli2@cis.upenn.edu
4 **University of Pennsylvania, Philadelphia, PA, USA**
val@cis.upenn.edu

Abstract

Provisioning is a technique for avoiding repeated expensive computations in *what-if analysis*. Given a query, an analyst formulates k *hypotheticals*, each retaining some of the tuples of a database instance, *possibly overlapping*, and she wishes to answer the query under *scenarios*, where a scenario is defined by a subset of the hypotheticals that are "turned on". We say that a query admits *compact* provisioning if given any database instance and any k hypotheticals, one can create a poly-size (in k) *sketch* that can then be used to answer the query under any of the 2^k possible scenarios without accessing the original instance.

In this paper, we focus on provisioning complex queries that combine relational algebra (the logical component), grouping, and statistics/analytics (the numerical component). We first show that queries that compute quantiles or linear regression (as well as simpler queries that compute count and sum/average of positive values) can be compactly provisioned to provide (multiplicative) *approximate* answers to an arbitrary precision. In contrast, *exact* provisioning for each of these statistics requires the sketch size to be exponential in k. We then establish that for any complex query whose logical component is a *positive* relational algebra query, as long as the numerical component can be compactly provisioned, the complex query itself can be compactly provisioned. On the other hand, introducing negation or recursion in the logical component again requires the sketch size to be exponential in k. While our positive results use algorithms that do not access the original instance after a scenario is known, we prove our lower bounds even for the case when, knowing the scenario, limited access to the instance is allowed.

1998 ACM Subject Classification H.2.8 Database Applications

Keywords and phrases What-if Analysis, Provisioning, Data Compression, Approximate Query Answering

Digital Object Identifier 10.4230/LIPIcs.ICDT.2016.18

1 Introduction

"What if analysis" is a common technique for investigating the impact of decisions on outcomes in science or business. It almost always involves a data analytics computation. Nowadays

* The full version of the paper can be found at [3].
† Supported in part by National Science Foundation grants CCF-1116961, CCF-1552909, and IIS-1447470 and an Adobe research award.
‡ Supported in part by National Science Foundation grants IIS-1217798 and IIS-1302212.


© Sepehr Assadi, Sanjeev Khanna, Yang Li, and Val Tannen;
licensed under Creative Commons License CC-BY

such a computation typically processes very large amounts of data and thus may be expensive to perform, especially repeatedly. An analyst is interested in exploring the computational impact of multiple *scenarios* that assume modifications of the input to the analysis problem. Our general aim is to avoid repeating expensive computations for each scenario. For a given problem, and starting from a given set of potential scenarios, we wish to perform just *one* possibly expensive computation producing a small *sketch* (i.e., a compressed representation of the input) such that the answer for any of the given scenarios can be derived rapidly from the sketch, without accessing the original (typically very large) input. We say that the sketch is "provisioned" to deal with the problem under any of the scenarios and following [13], we call the whole approach *provisioning*. Again, the goal of provisioning is to allow an analyst to efficiently explore a multitude of scenarios, using only the sketch and thus avoiding expensive recomputations for each scenario.

In this paper, we apply the provisioning approach to queries that perform *in-database analytics* [25][1]. These are queries that combine *logical* components (relational algebra and Datalog), grouping, and *numerical* components (e.g., aggregates, quantiles and linear regression). Other analytics are discussed under further work.

Abstracting away any data integration/federation, we will assume that the inputs are relational instances and that the scenarios are defined by a set of *hypotheticals*. We further assume that each hypothetical indicates the fact that certain tuples of an input instance are *retained* (other semantics for hypotheticals are discussed under further work).

A scenario consists of turning on/off each of the hypotheticals. Applying a scenario to an input instance therefore means keeping only the tuples retained by at least one of the hypotheticals that are turned on. Thus, a trivial sketch can be obtained by applying each scenario to the input, solving the problem for each such modified input and collecting the answers into the sketch. However, with k hypotheticals, there are *exponentially* (in k) many scenarios. Hence, even with a moderate number of hypotheticals, the size of the sketch could be enormous. Therefore, as part of the statement of our problem we will aim to provision a query by an algorithm that maps each (large) input instance to a *compact* (essentially size $\text{poly}(k)$) sketch.

▶ **Example 1.** Suppose a large retailer has many and diverse sales venues (e.g., its own stores, its own web site, through multiple other stores, and through multiple other web retailers). An analyst working for the retailer is interested in learning, for each product in, say, "Electronics", a regression model for the way in which the *revenue* from the product depends on both a sales venue's *reputation* (assume a numerical score) and a sales venue *commission* (in %; 0% if own store). Moreover, the analyst wants to ignore products with small sales volume unless they have a large MSRP (manufacturer's suggested retail price). Usually there is a large (possibly distributed/federated) database that captures enough information to allow the computation of such an analytic query. For simplicity we assume in this example that the revenue for each product ID and each sales venue is in one table and thus we have the following query with a self-explanatory schema:

```
SELECT x.ProdID, LIN_REG(x.Revenue, z.Reputation, z.Commission) AS (B,A1,A2)
FROM  RevenueByProductAndVenue x
INNER JOIN Products y ON x.ProdID=y.ProdID
```

[1] In practice, the MADlib project [29] has been one of the pioneers for in-database analytics, primarily in collaboration with Greenplum DB [21]. By now, major RDBMS products such as IBM DB2, MS SQL Server, and Oracle DB already offer the ability to combine extensive analytics with SQL queries.

```
INNER JOIN SalesVenues z ON x.VenueID=z.VenueID
WHERE y.ProdCategory="Electronics" AND (x.Volume>100 OR y.MSRP>1000)
GROUP BY x.ProdID
```

The syntax for treating linear regression as a multiple-column-aggregate is simplified for illustration purposes in this example. Here the values under the attributes `B,A1,A2` denote, for each `ProdID`, the coefficients of the linear regression model that is learned, i.e., `Revenue = B + A1*Reputation + A2*Commission`.

A desirable what-if analysis for this query may involve hypotheticals such as retaining certain venue types, retaining certain venues with specific sales tax properties, retaining certain product types (within the specified category, e.g., tablets), and many others. Each of these hypotheticals can in fact be implemented as selections on one or more of the tables in the query (assuming that the schema includes the appropriate information). However, combining hypotheticals into scenarios is problematic. The hypotheticals overlap and thus cannot be separated. With 10 (say) hypotheticals there will be $2^{10} = 1024$ (in practice at least hundreds) of regression models of interest for each product. Performing a lengthy computation for each one of these models is in total very onerous. Instead, we can *provision* the what-if analysis of this query as the query in this example falls within the class covered by our positive results.

Our results. Our goal is to characterize the feasibility of provisioning with sketches of *compact* size (see Section 2 for a formal definition) for a practical class of *complex queries* that consist of a *logical* component (relational algebra or Datalog), followed by a *grouping* component, and then by a *numerical* component (aggregate/analytic) that is applied to each group (a more detailed definition is given in Section 5).

The main challenge that we address, and the part where our main contribution lies, is the design of compact provisioning schemes for numerical queries, specifically linear (ℓ_2) regression and quantiles. Together with the usual count, sum and average, these are defined in Section 4 as queries that take a set of numbers or of tuples as input and return a number or a tuple of constant width as output. It turns out that if we expect exact answers, then none of these queries can be compactly provisioned. However, we show that compact provisioning schemes indeed exist for all of them if we relax the objective to computing near-exact answers (see Section 2 for a formal definition). The following theorem summarizes our results for numerical queries (see Section 4):

▶ **Theorem 2** (Informal). *The* quantiles, linear (ℓ_2) regression, count, and sum/average (of positive numbers) queries *can be compactly provisioned to provide (multiplicative) approximate answers to an arbitrary precision, while their exact provisioning requires the sketch size to be exponential in the number of hypotheticals.*

Our results on provisioning numerical queries can then be used for complex queries as the following theorem summarizes (see Section 5):

▶ **Theorem 3** (Informal). *Any complex query whose logical component is a* positive relational algebra query *can be compactly provisioned to provide an approximate answer to an arbitrary precision as long as its numerical component can be compactly provisioned for the same precision. On the other hand, introducing* negation *or* recursion *in the logical component, requires the sketch size to be exponential in the number of hypotheticals.*

Our techniques. At a high-level, our approach for compact provisioning can be described as follows. We start by building a sub-sketch for each hypothetical by focusing solely on

the retained tuples of each hypothetical individually. We then examine these sub-sketches against each other and collect additional information from the original input to summarize the effect of appearance of other hypotheticals to each already computed sub-sketch. The first step usually involves using well-known (and properly adjusted) sampling or sketching techniques, while the second step, which is where we concentrate the bulk of our efforts, is responsible for gathering the information required for combining the sketches and specifically dealing with overlapping hypotheticals. Given a scenario, we answer the query by fetching the corresponding sub-sketches and merging them together; the result is a new sketch that act as sketch for the input consist of the *union* of the hypotheticals.

We prove our lower bounds by first identifying a central problem, i.e., the Coverage problem (see Problem 8), with provably large space requirement for any provisioning scheme, and then use reductions from this problem to establish the lower bound for other queries of interest. The space requirement of the Coverage problem itself is proven using simple tools from information theory.

Comparison with existing work. Our techniques for compact provisioning share some similarities with those used in data streaming and in the distributed computation model of [12, 35], and in particular *linear sketching*, which corresponds to applying a linear transformation to the input data to obtain the sketch. However, due to overlap in the input, our sketches are required to to be composable with the *union* operation (instead of the *addition* operation obtained by linear sketches) and hence linear sketching techniques are not directly applicable.

Dealing with duplicates in the input (similar to the overlapping hypotheticals) has also been considered in the streaming and distributed computation models (see, e.g., [10, 7]), which consider sketches that are "duplicate-resilient". Indeed, for simple queries like count, a direct application of these sketches is sufficient for compact provisioning (see Section 4.1). We also remark that the Count-Min sketch [9] can be applied to approximate quantiles even in the presence of duplication (see [7]), i.e., is duplicant-resilient. However, the approximation guarantee achieved by Count-Min sketch for quantiles is only *additive* (i.e., $\pm\varepsilon n$), in contrast to the stronger notion of *multiplicative* approximation (i.e., $(1 \pm \varepsilon)$) we seek in this paper. To the best of our knowledge, there is no similar result concerning duplicate-resilient sketches for multiplicative approximation of quantiles or the linear regression problem, and existing techniques do not seem to be applicable for our purpose. Indeed one of the primary technical contributions of this paper is designing provisioning schemes that can effectively deal with overlapping hypotheticals for quantiles and linear regression.

Further related work. *Provisioning*, in the sense used in this paper, originated in [13] together with a proposal for how to perform it taking advantage of provenance tracking. Answering queries under hypothetical updates is studied in [17, 4] but the focus there is on using a specialized warehouse to avoid transactional costs. We refer the interested reader to [13] for more related work.

Estimating the number of distinct elements (corresponding to the count query) has been studied extensively in data streams [16, 2, 5, 28] and distributed functional monitoring [11, 35]. For estimating quantiles in the data stream or the distributed model, [31, 18, 22, 9, 23, 36] achieve an additive error of εn for an input of length n, and [24, 8] achieve a (stronger guarantee of) $(1 \pm \varepsilon)$-approximation. Sampling and sketching techniques for ℓ_2-regression problem have also been studied in [14, 32, 15, 6] for either speeding up the computation or in data streams (see [30, 34] for excellent surveys on this topic).

2 Problem Statement

Hypotheticals. Fix a relational schema Σ. Our goal is to provision queries on Σ-instances. A *hypothetical* w.r.t. Σ is a computable function h that maps every Σ-instance I to a sub-instance $h(I) \subseteq I$. Formalisms for specifying hypotheticals are of course of interest (e.g., apply a selection predicate to each table in I) but we do not discuss them here because the results in this paper do not depend on them.

Scenarios. We will consider analyses (scenario explorations) that start from a finite set H of hypotheticals. A *scenario* is a non-empty set of hypotheticals $S \subseteq H$. The result of applying a scenario $S = \{h_1, \ldots, h_s\}$ to an instance I is defined as a sub-instance $I|_S = h_1(I) \cup \cdots \cup h_s(I)$. In other words, under S, if any $h \in S$ is said to be turned on (similarly, any $h \in H \setminus S$ is turned off), each turned on hypothetical h will retain the tuples $h(I)$ from I.

▶ **Definition 4** (Provisioning). Given a query Q, to *provision* Q means to design a pair of algorithms: (i) a **compression** algorithm that takes as input an instance I and a set H of hypotheticals, and outputs a data structure Γ called a **sketch**, and (ii) an **extraction** algorithm that for any scenario $S \subseteq H$, outputs $Q(I|_S)$ using only Γ (without access to I).

To be more specific, we assume the compression algorithm takes as input an instance I, and k hypotheticals $h_1, \ldots, h_k$ along with the sub-instances $h_1(I), \ldots, h_k(I)$ that they define. A hypothetical will be referred to by an index from $\{1, \ldots, k\}$, and the extraction algorithm will be given scenarios in the form of sets of such indices. Hence, we will refer to a scenario $S \subseteq H$ where $S = \{h_{i_1}, \ldots, h_{i_s}\}$ by abusing the notation as $S = \{i_1, \ldots, i_s\}$.

We call such a pair of compression and extraction algorithms a *provisioning scheme*. The compression algorithm runs only once; the extraction algorithm runs repeatedly for all the scenarios that an analyst wishes to explore. We refer to the time that the compression algorithm requires as the *compression time*, and the time that extraction algorithm requires for each scenario as the *extraction time*.

The definition above is not useful by itself for positive results because it allows for trivial space-inefficient solutions. For example, the definition is satisfied when the sketch Γ is defined to be a copy of I itself or, as mentioned earlier, a scenario-indexed collection of all the answers. Obtaining the answer for each scenario is immediate for either case, but such a sketch can be prohibitively large as the number of tuples in I could be enormous, and the number of scenarios is exponential in $|H|$.

This discussion leads us to consider complexity bounds on the size of the sketches.

▶ **Definition 5** (Compact provisioning). A query Q can be *compactly* provisioned if there exists a provisioning scheme for Q that given any input instance I and any set of hypotheticals H, constructs a sketch of size $\text{poly}(|H|, \log |I|)$ bits.

We make the following important remark about the restrictions made in Definitions 4 and 5.

▶ Remark. At first glance, the requirement that the input instance I cannot be examined *at all* during extraction may seem artificial, and the same might be said about the size of the sketch depending polynomially on $\log n$ rather than a more relaxed requirement. However, we show that our lower bound results hold *even if* a portion of size $o(n)$ of the input instance can be examined during extraction *after* the scenario is revealed and *even if* the space dependence of the sketch is only restricted to be $o(n)$ (instead of depending only polynomially on $\log n$). In spite of this, the positive results we obtain all use sketches with space that depend polynomially only on $\log n$ and does *not* require examining the original database

during the extraction. These calibration results further justify our design choices for compact provisioning.

Even though the definition of compact provisioning does not impose any restriction on either the compression time or the extraction time, all our positive results in this paper are supported by (efficient) polynomial time algorithm. Note that this is *data-scenario complexity*: we assume the size of the query (and the schema) to be a constant but we consider dependence on the size of the instance and the number of hypotheticals. Our negative results (lower bounds on the sketch size), on the other hand, hold even when the compression and the extraction algorithms are computationally unbounded.

Exact vs. approximate provisioning. Definition 5 focused on exact answers for the queries. While this is appropriate for, e.g., relational algebra queries, as we shall see, for queries that compute numerical answers such as aggregates and analytics, having the flexibility of answering queries approximately is essential for any interesting positive result.

▶ **Definition 6** (ε-provisioning)**.** For any $0 < \varepsilon < 1$, a query Q can be *ε-provisioned* if there exists a provisioning scheme for Q, whereby for each scenario S, the extraction algorithm outputs a $(1 \pm \varepsilon)$ approximation of $Q(I|_S)$, where I is the input instance.

We say a query Q can be *compactly* ε-provisioned if Q can be ε-provisioned by a provisioning scheme that, given any input instance I and any set of hypotheticals H, creates a sketch of size $\text{poly}(|H|, \log|I|, 1/\varepsilon)$ bits.

We emphasize that throughout this paper, we only consider the approximation guarantees which are *relative* (multiplicative) as opposed to the weaker notion of additive approximations. The precise definition of relative approximation guarantee will be provided for each query individually. Moreover, as expected, randomization will be put to good use in ε-provisioning. We therefore extend the definition to cover the provisioning schemes that use both randomization and approximation.

▶ **Definition 7.** For any $\varepsilon, \delta > 0$, an *(ε, δ)-provisioning scheme* for a query Q is a provisioning scheme where both compression and extraction algorithms are allowed to be randomized and the output *for every* scenario S is an $(1 \pm \varepsilon)$-approximation of $Q(I|_S)$ with probability $1 - \delta$. Moreover, the compression time of the scheme is $\text{poly}(|I|, |H|, 1/\varepsilon, \log(1/\delta))$ and extraction time is $\text{poly}(|\Gamma|)$.

An (ε, δ)-provisioning scheme is called *compact* iff it constructs sketches of size only $\text{poly}(|H|, \log|I|, 1/\varepsilon, \log(1/\delta))$ bits.

Note that in many applications, the size of the database is a very large number, and hence the exponent in the $\text{poly}(|I|)$-dependence of the compression time might become an issue. Therefore, we further define *(ε, δ)-linear provisioning scheme,* where the dependence of the compression time on $|I|$ is *essentially linear*, i.e., $O(|I|) \cdot \text{poly}(|H|, \log(|I|), 1/\varepsilon, \log(1/\delta))$. All our positive results for queries with numerical answers will be stated in terms of compact (ε, δ)-linear provisioning schemes, which ensure efficiency in both running time and sketch size.

Complex queries. Our main target consists of practical queries that combine logical, grouping, and numerical components. In Section 5, we focus on *complex queries* defined by a logical (relational algebra or Datalog) query that returns a set of tuples, followed by a group-by operation (on set of grouping attributes) and further followed by numerical query that is applied to each sets of tuples resulting from the grouping. This class of queries already covers

many practical examples. We observe that the output of such a complex query is a set of p tuples where p is *the number of distinct values taken by the grouping attributes*. Therefore, the size of any provisioning sketch must also depend on p. We show (in Theorem 19) that a sketch for a query that involves grouping can be obtained as a collection of p sketches. Hence, if each of the p sketches is of compact size (as in Definitions 5 and 7) and the value p itself is bounded by $\text{poly}(|H|, \log |I|)$, then the overall sketch for the complex query is also of compact size. Note that p is typically small for the kind of grouping used in practical analysis queries (e.g., number of products, number of departments, number of locations, etc.). Intuitively, an analyst would have trouble making sense of an output with a large number of tuples.

Notation. Throughout the paper, we denote by k the number of hypotheticals, and by n the size $|I|$ of the input instance. For any integer $m > 0$, $[m]$ denotes the set $\{1, 2, \ldots, m\}$. The $\tilde{O}(\cdot)$ notation suppresses $\log\log(n)$, $\log\log(1/\delta)$, $\log(1/\varepsilon)$, and $\log(k)$ factors. All logarithms are in base two unless stated otherwise.

3 Coverage: A "Hard" Problem for Provisioning

To establish our lower bounds in this paper, we introduce a "hard" problem called Coverage. Though not defined in the exact formalism of provisioning, the Coverage problem can be solved by many provisioning schemes using proper "reductions" and hence a lower bound for the Coverage problem can be used to establish similar lower bounds for provisioning various queries.

Informally speaking, in the Coverage problem, we are given a collection of k subsets of a universe $[n]$ and the goal is to "compress" this collection in order to answer to the questions in which indices of some subsets in the collection are provided and we need to figure out whether these subsets cover the universe $[n]$ or not. We are interested in compressing schemes for this problem that when answering each question, in addition to the already computed summary of the collection, also have a limited access to the original instance (see Remark 2 after Definition 5 for motivation of this modification). The Coverage problem is defined formally as follows.

▶ **Problem 8** (Coverage). *Suppose we are given a collection $\mathcal{S} = \{S_1, S_2, \ldots S_k\}$ of the subsets of $[n]$. The goal in the* Coverage *problem is to find a* compressing scheme *for $\mathcal{S}$, defined formally as a triple of algorithms:*

- *A* ***compression algorithm*** *which given the collection $\mathcal{S}$ creates a* data structure *D.*
- *An* ***examination algorithm*** *which given a subset of $[k]$, a* question*, $Q = \{i_1, \ldots, i_s\}$ and the data structure D, computes a set $J \subseteq [n]$ of indices and lookup for each $j \in J$ and each S_i ($i \in [k]$), whether $j \in S_i$ or not. The output of the examination algorithm is a tuple $S^J := (S_1^J, \ldots, S_k^J)$, where $S_i^J = S_i \cap J$.*
- *An* ***extraction algorithm*** *which given a question $\{i_1, \ldots, i_s\}$, the data structure D, and the tuple S^J, outputs "Yes", if $S_{i_1} \cup \ldots \cup S_{i_s} = [n]$ and "No" otherwise.*

We refer to the size of D, denoted by $|D|$, as the storage requirement of the compression scheme and to the size of J, denoted by $|J|$, as the examination requirement of the scheme. The above algorithms can all be randomized; in that case, we require that for *each* question Q, the final answer (of the extraction algorithm) to be correct with a probability at least 0.99. Note that this choice of constant is arbitrary and is used only to simplify the analysis; indeed, one can always amplify the probability of success by repeating the scheme constant number of times and return the majority answer.

While Coverage is not stated in the exact formalism of provisioning, the analogy between this problem and provisioning schemes should be clear. In particular, for our lower bound proofs for provisioning schemes, we can alter the Definition 4 to add an examination algorithm and allow a similar access to the original database to the provisioning scheme.

We establish the following lower bound on storage and examination requirement of any compressing scheme for the Coverage problem. The proof of this Theorem is deferred to the full version of the paper [3] (see Theorem 3.1).

▶ **Theorem 9.** *Any compressing scheme for the* Coverage *problem that answers each question correctly with probability at least* 0.99*, either has storage requirement or examination requirement of* $\min(2^{\Omega(k)}, \Omega(n))$ *bits.*

Allowing access to the original input in Theorem 9 makes the lower bound very robust. However, due to this property, the lower bound does not seem to follow from standard communication complexity lower bounds and hence we use an information-theoretic approach to prove this theorem directly, which may be of independent interest. We remark that since our other lower bounds are typically proven using a reduction from the Coverage problem, the properties in Theorem 9 (i.e., allowing randomization and $o(n)$ access to the database *after* being given the scenario) also hold for them and we do not mention this explicitly.

4 Numerical Queries

In this section, we study provisioning of numerical queries, i.e., queries that output some (rational) number(s) given a set of tuples. In particular, we investigate aggregation queries including count, sum, average, and quantiles (therefore min, max, median, rank, and percentile), and as a first step towards provisioning database-supported machine learning, linear (ℓ_2) regression. We assume that the relevant attribute values are rational numbers of the form a/b where both a, b are integers in range $[-W, W]$ for some $W > 0$.

4.1 The Count, Sum, and Average Queries

In this section, we study provisioning of the count, sum, and average queries, formally defined as follows. The answer to the count query is the number of tuples in the input instance. For the other two queries, we assume a relational schema with a binary relation containing two attributes: an *identifier (key)* and a *weight*. We say that a tuple x is smaller than the tuple y, if the weight of x is smaller than the weight of y. Given an instance I, the answer to the sum query (resp. the average query) is the *total weights* of the tuples (resp. the *average weight* of the tuples) in I.

We first show that none of the count, sum, average queries can be provisioned both compactly and exactly, which motivates the ε-provisioning approach, and then briefly describe how to build a compact (ε, δ)-linear provisioning scheme for each of them.

▶ **Theorem 10.** *Exact provisioning of the* count*,* sum*, or* average *queries requires sketches of size* $\min(2^{\Omega(k)}, \Omega(n))$ *bits.*

Proof Sketch. We prove the lower bound for the count using a reduction from the Coverage problem; the lower bound of the sum follows immediately by setting all weights to be 1. The reduction for the average query is slightly more involved and is deferred to the full version of the paper [3] (see Theorem 4.1).

Given $\{S_1, \ldots, S_k\}$, where each S_i is a subset of $[n]$, we solve Coverage using a provisioning scheme for the count query. Define an instance I of a relational schema with a unary

relation A, where $I = \{A(x)\}_{x\in[n]}$. Define a set H of k hypotheticals, where for any $i \in [k]$, $h_i(I) = \{A(x)\}_{x\in S_i}$. For any scenario $S = \{i_1, \ldots, i_s\}$, the count of $I|_S$ is n iff $S_{i_1} \cup \ldots \cup S_{i_s} = [n]$. Hence, any provisioning scheme for the count query solves the Coverage problem and the lower bound follows from Theorem 9. ◀

We further point out that if the weights can be both positive and negative, the sum (and average) cannot be compactly provisioned even approximately (see the full version [3], Theorem 4.2), and hence we will focus on ε-provisioning for *positive* weights.

We conclude this section by briefly explaining the ε-provisioning schemes for the count, sum, and average queries. These results are mostly direct application of known techniques and we present them here for completeness.

▶ **Theorem 11** (count, sum, average)**.** *For any $\varepsilon, \delta > 0$, there exist compact (ε, δ)-linear provisioning schemes for the* count *query and the* sum*/*average *queries (with positive weights), respectively.*

The count query can be provisioned by using *linear sketches* for estimating the ℓ_0-norm (see, e.g., [28]) as follows. Consider each hypothetical $h_i(I)$ as an n-dimensional boolean vector $\mathbf{x}_i$, where the j-th entry is 1 iff the j-th tuple in I belongs to $h_i(I)$. For each $\mathbf{x}_i$, create a linear sketch (using $\tilde{O}(\varepsilon^{-2}\log n)$ bits of space) that estimates the ℓ_0-norm [28]. Given any scenario S, combine (i.e., add together) the linear sketches of the hypotheticals in S and use the combined sketch to estimate the ℓ_0-norm (which is equal to the answer of count).

Note that we can directly use linear sketching for provisioning the count query since counting the duplicates once (as done by union) or multiple times (as done by addition) does not change the answer. However, this is *not* the case for other queries of interest like quantiles and regression and hence linear sketching is not directly applicable for them.

In the full version of the paper [3] (see Theorem 4.3), we describe a self-contained approach for ε-provisioning the count query with a slightly better dependence on the parameter n ($\log\log n$ instead of $\log n$). We name this sketch the CNT-Sketch, which will be used used later for provisioning other queries. In particular, provisioning the sum query using a CNT-Sketch is straightforward when the weights are positive: group the tuples by weight into $\Theta(\log n/\varepsilon)$ groups and construct a CNT-Sketch for each group to estimate the total sum. In the full version [3] (see Theorem 4.4), we describe this in more detail and point out how to further improve the sketch size. The provisioning scheme for average follows immediately from these results.

4.2 The Quantiles Query

We now study provisioning of the quantiles query. We again assume a relational schema with just one binary relation containing attributes identifier and weight. For any instance I and any tuple $x \in I$, we define the *rank* of x to be the number of tuples in I that are smaller than or equal to x (in terms of the weights). The output of a quantiles query with a given parameter $\phi \in (0, 1]$ on an instance I is the tuple with rank $\lceil \phi \cdot |I| \rceil$. Finally, we say a tuple x is a $(1 \pm \varepsilon)$-approximation of a quantiles query whose correct answer is y, iff the rank of x is a $(1 \pm \varepsilon)$-approximation of the rank of y.

Similar to the previous section, we first show that the quantiles query admits no compact provisioning scheme for exact answer and then provide a compact ε-provisioning scheme for this query.

▶ **Theorem 12.** *Exact provisioning of the* quantiles *query* even on disjoint hypotheticals *requires sketches of size* $\min(2^{\Omega(k)}, \Omega(n))$ *bits.*

In the quantiles query, the parameter ϕ may be given either already to the compression algorithm or only to the extraction algorithm. The latter yields an immediate lower bound of $\Omega(n)$, since by varying ϕ over $(0, 1]$, one can effectively "reconstruct" the original database. However, we achieve a more interesting lower bound for the case when ϕ is given at to the compression algorithm (i.e., a fixed ϕ for all scenarios, e.g., setting $\phi = 1/2$ to find the *median*). An important property of the lower bound for quantiles is that, in contrast to all other lower bounds for numerical queries in this paper, this lower bound holds even for disjoint hypotheticals[2]. The proof of Theorem 12 is deferred to the full version [3] (see Theorem 4.7).

We now turn to establish the main result of this section, which argue the existence of a compact scheme for ε-provisioning the quantiles. We emphasize that the approximation guarantee in the following theorem is *multiplicative*.

▶ **Theorem 13** (quantiles). *For any $\varepsilon, \delta > 0$, there exists a compact (ε, δ)-linear provisioning scheme for the* quantiles *query that creates a sketch of size $\tilde{O}(k\varepsilon^{-3} \log n \cdot (\log(n/\delta)+k)(\log W + k))$ bits.*

We should note that in this theorem the parameter ϕ is only provided in the extraction phase. Our starting point is the following simple lemma first introduced by [24].

▶ **Lemma 14** ([24]). *For any list of* unique *numbers $A = (a_1, \ldots, a_n)$ and parameters $\varepsilon, \delta > 0$, let $t = \lceil 12\varepsilon^{-2} \log(1/\delta) \rceil$; for any* target rank *$r > t$, if we independently sample each element with probability t/r, then with probability at least $1 - \delta$, the rank of the t-th smallest sampled element is a $(1 \pm \varepsilon)$-approximation of r.*

The proof of Lemma 14 is an standard application of the Chernoff bound and the main challenge for provisioning the quantiles query comes from the fact that hypotheticals overlap. We propose the following scheme which addresses this challenge.

Compression algorithm for the quantiles **query.** Given an instance I, a set H of hypotheticals, and two parameters $\varepsilon, \delta > 0$, let $\varepsilon' = \varepsilon/5$, $\delta' = \delta/3$, and $t = \lceil 12\varepsilon'^{-2}(\log(1/\delta') + 2k + \log(n)) \rceil$.

1. Create and **record** a CNT-Sketch for I and H with parameters ε' and δ'.
2. Let $\{r_j = (1+\varepsilon')^j\}_{j=0}^{\lceil \log_{(1+\varepsilon')} n \rceil}$. For each r_j, create the following sub-sketch individually.
3. If $r_j \leq t$, for each hypothetical h_i, **record** the r_j smallest chosen tuples in $h_i(I)$. If $r_j > t$, for each hypothetical h_i, choose each tuple in $h_i(I)$ with probability t/r_j, and **record** the $\lceil (1 + 3\varepsilon') \cdot t \rceil$ smallest tuples in a list $T_{i,j}$. For each tuple x in the resulting list $T_{i,j}$, **record** its *characteristics vector* for the set of the hypotheticals, which is a k-dimensional binary vector $(v_1, v_2, \ldots, v_k)$, with value 1 on v_l whenever $x \in h_l(I)$ and 0 elsewhere.

[2] All other numerical queries that we study in this paper can be compactly provisioned for exact answer, when the hypotheticals are *disjoint*.

Extraction algorithm for the quantiles query. Suppose we are given a scenario S and a parameter $\phi \in (0, 1]$. In the following, the rank of a tuple always refers to its rank in the sub-instance $I|_S$.

1. Denote by $\tilde{n}$ the output of the CNT-Sketch on S. Let $\tilde{r} = \phi \cdot \tilde{n}$, and find the index γ, such that $r_\gamma \leq \tilde{r} < r_{\gamma+1}$.
2. If $r_\gamma \leq t$, among all the hypotheticals turned on by S, take the union of the recorded tuples and output the r_γ-th smallest tuple in the union.
3. If $r_\gamma > t$, from each h_i turned on by S, and each tuple x recorded in $T_{i,\gamma}$ with a characteristic vector $(v_1, v_2, \ldots, v_k)$, *collect* x iff for any $l < i$, either $v_l = 0$ or $h_l \notin S$. In other words, a tuple x recorded by h_i is taken only when among the hypotheticals that are turned on by S, i is the smallest index s.t. $x \in h_i(I)$. We will refer to this procedure as the *deduplication*. Output the t-th smallest tuple among all the tuples that are collected.

We call a sketch created by the above compression algorithm a QTL-Sketch. We defer the analysis of this sketch and the proof of Theorem 13 to the full version of the paper [3] (see Theorem 4.8).

Extensions. By simple extensions of our scheme, many variations of the quantiles query can be answered, including outputting the rank of a tuple x, the percentiles (the rank of x divided by the size of the input), or the tuple whose rank is Δ larger than x, where $\Delta > 0$ is a given parameter. As an example, for finding the rank of a tuple x, we can find the tuples with ranks approximately $\{(1+\varepsilon)^l\}$, $l \in [\lceil \log_{(1+\varepsilon)} n \rceil]$, using the QTL-Sketch, and among the found tuples, output the rank of the tuple whose weight is the closest to the weight of x.

4.3 The Linear Regression Query

In this section, we study provisioning of the regression query (i.e., the ℓ_2-regression problem), where the input is a matrix $\mathbf{A}_{n\times d}$ and a vector $\mathbf{b}_{n\times 1}$, and the goal is to output a vector $\mathbf{x}$ that minimizes $\|\mathbf{Ax} - \mathbf{b}\|$ ($\|\cdot\|$ stands for the ℓ_2 norm). A $(1+\varepsilon)$-approximation of the regression query is a vector $\tilde{\mathbf{x}}$ such that $\|\mathbf{A}\tilde{\mathbf{x}} - \mathbf{b}\|$ is at most $(1+\varepsilon)\min_{\mathbf{x}} \|\mathbf{Ax} - \mathbf{b}\|$.

The input is specified using a relational schema Σ with a $(d+2)$-ary relation R. Given an instance I of Σ with n tuples, we interpret the projection of R onto its first d columns, the $(d+1)$-th column, and the $(d+2)$-column respectively as the matrix $\mathbf{A}$, the column vector $\mathbf{b}$, and the identifiers for the tuples in R. For simplicity, we denote $I = (\mathbf{A}, \mathbf{b})$, assume that the tuples are ordered, and use the terms the i-th tuple of I and the i-th row of $(\mathbf{A}, \mathbf{b})$ interchangeably.

Notation. For any matrix $\mathbf{M} \in \mathbb{R}^{n\times d}$, denote by $\mathbf{M}_{(i)}$ the i-th row of $\mathbf{M}$, and by $\mathbf{U}_M \in \mathbb{R}^{n\times\rho}$ (where ρ is the rank of $\mathbf{M}$) the orthonormal matrix of the column space of $\mathbf{M}$ (see [26] for more details). Given an instance $I = (\mathbf{A}, \mathbf{b})$, and k hypotheticals, we denote for each hypothetical h_i the sub-instance $h_i(I) = (\mathbf{A}_i, \mathbf{b}_i)$. For any integer i, $\mathbf{e}_i$ denotes the i-th standard basis; hence, the i-th row of $\mathbf{M}$ can be written as $\mathbf{e}_i^T\mathbf{M}$.

The following theorem shows that the regression query cannot be compactly provisioned for exact answers (see the full version [3], Theorem 4.10) and hence, we will focus on ε-provisioning.

▶ **Theorem 15.** *Exact provisioning of the* regression *query,* even when the dimension is $d = 1$, *requires sketches of size* $\min(2^{\Omega(k)}, \Omega(n))$ *bits*

Before continuing, we remark that if hypotheticals are disjoint, the regression query admits compact provisioning for exact answer (see the full version [3], Section 4.3), and hence the hardness of the problem again lies on the fact that hypotheticals overlap. We now turn to provide a provisioning scheme for the regression query and prove the following theorem.

▶ **Theorem 16** (regression)**.** *For any* $\varepsilon, \delta > 0$*, there exists a compact* (ε, δ)*-linear provisioning scheme for the* regression *query that creates a sketch of size* $\tilde{O}(\varepsilon^{-1}k^3 d \log(nW)(k + \log \frac{1}{\delta}))$ *bits.*

Overview. Our starting point is a non-uniform sampling based approach (originally used for speeding up the ℓ_2-regression computation [32]) which uses a small sample to accurately approximates the ℓ_2-regression problem. Since the probability of sampling a tuple (i.e., a row of the input) in this approach depends on its relative importance which can vary dramatically when input changes, this approach is not directly applicable to our setting.

Our contribution is a *two-phase sampling* based approach to achieve the desired sampling probability distribution for *any* scenario. At a high level, we first sample and record a small number of tuples from each hypothetical using the non-uniform sampling approach; then, given the scenario in the extraction phase, we re-sample from the recorded tuples of the hypotheticals presented in the scenario. Furthermore, to rescale the sampled tuples (as needed in the original approach), we obtain the exact sampling probabilities of the recorded tuples by recording their relative importance in each hypothetical. Our approach relies on a monotonicity property of the relative importance of a tuple when new tuples are added to the original input.

RowSample. We start by describing the non-uniform sampling algorithm. Let $\mathcal{P} = (p_1, p_2, \ldots, p_n)$ be a probability distribution, and $r > 0$ be an integer. Sample r tuples of $I = (\mathbf{A}, \mathbf{b})$ with replacement according to the probability distribution $\mathcal{P}$. For each sample, if the j-th row of $\mathbf{A}$ is sampled for some j, rescale the row with a factor $(1/\sqrt{rp_j})$ and store it in the sampling matrix $(\tilde{\mathbf{A}}, \tilde{\mathbf{b}})$. In other words, if the i-th sample is the j-th row of I, then $(\tilde{\mathbf{A}}_{(i)}, \tilde{\mathbf{b}}_{(i)}) = (\mathbf{A}_{(j)}, \mathbf{b}_{(j)})/\sqrt{rp_j}$. We denote this procedure by RowSample$(\mathbf{A}, \mathbf{b}, \mathcal{P}, r)$, and $(\tilde{\mathbf{A}}, \tilde{\mathbf{b}})$ is its output. The RowSample procedure has the following property [32] (see also [14, 34] for more details on introducing the parameter β).

▶ **Lemma 17** ([32])**.** *Suppose* $\boldsymbol{A} \in \mathbb{R}^{n \times d}$*,* $\boldsymbol{b} \in \mathbb{R}^n$*, and* $\beta \in (0, 1]$*;* $\mathcal{P} = (p_1, p_2, \ldots, p_n)$ *is a probability distribution on* $[n]$*, and* $r > 0$ *is an integer. Let* $(\tilde{\boldsymbol{A}}, \tilde{\boldsymbol{b}})$ *be an output of* RowSample$(\boldsymbol{A}, \boldsymbol{b}, \mathcal{P}, r)$*, and* $\tilde{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}} \|\tilde{\boldsymbol{A}}\boldsymbol{x} - \tilde{\boldsymbol{b}}\|$.

If for all $i \in [n]$*,* $p_i \geq \beta \frac{\|e_i^T \boldsymbol{U}_A\|^2}{\sum_{j=1}^n \|e_j^T \boldsymbol{U}_A\|^2}$ *, and* $r = \Theta(\frac{d \log d \log(1/\delta)}{\varepsilon \cdot \beta})$*, then with probability at least* $(1 - \delta)$*,* $\|\boldsymbol{A}\tilde{\boldsymbol{x}} - \boldsymbol{b}\| \leq (1 + \varepsilon) \min_{\boldsymbol{x}} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|$.

The value $\|e_i^T \mathbf{U}_A\|^2$, i.e the square norm of the i-th row of $\mathbf{U}_A$, is also called the *leverage score* of the i-th row of $\mathbf{A}$. One should view the leverage scores as the "relative importance" of a row for the ℓ_2-regression problem (see [30] for more details). Moreover, using the fact that columns of $\mathbf{U}_A$ are orthonormal, we have $\sum_j \|e_j^T \mathbf{U}_A\|^2 = \rho$, where ρ is the rank of $\mathbf{A}$.

We now define our provisioning scheme for the regression query, where the compression algorithm performs the first phase of sampling (samples rows from each hypothetical) and the extraction algorithm performs the second (samples from the recorded tuples).

Compression algorithm for the regression query. Suppose we are given an instance $I = (\mathbf{A}, \mathbf{b})$ and k hypotheticals with $h_i(I) = (\mathbf{A}_i, \mathbf{b}_i)$ $(i \in [k])$. Let $t = \Theta(\varepsilon^{-1} kd \log d \cdot (k + \log(1/\delta)))$, and define for each $i \in [k]$ a probability distribution $\mathcal{P}_i = (p_{i,1}, p_{i,2}, \ldots, p_{i,n})$ as follows. If the j-th row of $\mathbf{A}$ is the l-th row of $\mathbf{A}_i$ (they correspond to the same tuple), let $p_{i,j} = \|e_l^T \mathbf{U}_{A_i}\|^2/\rho$, where ρ is the rank of $\mathbf{A}_i$. If $\mathbf{A}_{(j)}$ does not belong to $\mathbf{A}_i$, let $p_{i,j} = 0$. Using the fact that for every $i \in [k]$, $\mathbf{U}_{A_i}$ is an orthonormal matrix, $\sum_{j=1}^{n} p_{i,j} = 1$. **Record** t independently chosen random permutations of $[k]$, and for each hypothetical h_i, create a sub-sketch as follows.

1. Sample t tuples of $h_i(I)$ with replacement, according to the probability distribution $\mathcal{P}_i$.
2. For each of the sampled tuples, assuming it is the j-th tuple of I, **record** the tuple along with its sampling rate in each hypothetical, i.e., $\{p_{i',j}\}_{i' \in [k]}$.

Extraction algorithm for the regression query. Given a scenario $S = \{i_1, \ldots, i_s\}$, we will recover from the sketch a matrix $\tilde{\mathbf{A}}_{t \times d}$ and a vector $\tilde{\mathbf{b}}_{t \times 1}$. For $l = 1$ to t:

1. Pick the l-th random permutation recorded in the sketch. Let γ be the first value in this permutation that appears in S.
2. If $(\mathbf{a}, b)$ is the l-th tuple sampled by the hypothetical h_γ, which is the j-th tuple of I, let $q_j = \sum_{i \in S} p_{i,j}/|S|$, using the recorded sampling rates.
3. Let $(\tilde{\mathbf{A}}_{(l)}, \tilde{\mathbf{b}}_{(l)}) = (\mathbf{a}, b)/\sqrt{tq_j}$. Return $\tilde{\mathbf{x}} = \arg\min_{\mathbf{x}} \|\tilde{\mathbf{A}}\mathbf{x} - \tilde{\mathbf{b}}\|$ (using any standard method for solving the ℓ_2-regression problem).

We call a sketch constructed above a REG-Sketch. In order to show the correctness of this scheme, we need the following lemma regarding the monotonicity of leverage scores (the proof is presented in the full version [3], Lemma 4.13).

▶ **Lemma 18** (Monotonicity of Leverage Scores). *Let $\boldsymbol{A} \in \mathbb{R}^{n \times d}$ and $\boldsymbol{B} \in \mathbb{R}^{m \times d}$ be any matrix. Define matrix $\boldsymbol{C} \in \mathbb{R}^{(n+m) \times d}$ by* appending *rows of $\boldsymbol{B}$ to $\boldsymbol{A}$, i.e., $\boldsymbol{C}^T = [\boldsymbol{A}^T, \boldsymbol{B}^T]$. For any $i \in [n]$, if L_i is the leverage score of $\boldsymbol{A}_{(i)}$ and $\hat{L}_i$ is the leverage score of $\boldsymbol{C}_{(i)}$, then $L_i \geq \hat{L}_i$.*

Lemma 18 claims that adding more rows to a matrix $\mathbf{A}$ can only reduce the importance of any point originally in $\mathbf{A}$. Note that this is true even when the matrix $\mathbf{C}$ is formed by arbitrarily combining rows of $\mathbf{B}$ and $\mathbf{A}$ (rather than appending at the end).

Proof of Theorem 16. Fix a scenario S and let $I|_S = (\hat{\mathbf{A}}, \hat{\mathbf{b}})$. It is straightforward to verify that, for any step $l \in [t]$, q_j (in line (2) of the extraction algorithm) is the probability that the j-th tuple of I is chosen, if the j-th tuple belongs to $I|_S$. Hence, assuming $\mathcal{P}'$ is the probability distribution defined by the q_js on rows of the $I|_S$, the extraction algorithm implements RowSample$(\hat{\mathbf{A}}, \hat{\mathbf{b}}, \mathcal{P}', t)$.

We will show that $q_j \geq \|e_l^T \mathbf{U}_{\hat{A}}\|^2/k\hat{\rho}$, where the l-th row of $\hat{\mathbf{A}}$ is the j-th row of $\mathbf{A}$, and $\hat{\rho}$ is the rank of $\hat{A}$. Then, by Lemma 17 with β set to $1/k$, with probability at least $1 - \frac{\delta}{2^k}$, $\|\hat{\mathbf{A}}\tilde{\mathbf{x}} - \hat{\mathbf{b}}\|$ is at most $(1+\varepsilon)\min_{\mathbf{x}} \|\hat{\mathbf{A}}\mathbf{x} - \hat{\mathbf{b}}\|$; hence, the returned vector $\tilde{\mathbf{x}}$ is a $(1+\varepsilon)$-approximation. Applying a union bound over all 2^k scenarios, with probability at least $(1-\delta)$, our scheme ε-provisions the regression query.

We now prove that $q_j \geq \|e_l^T \mathbf{U}_{\hat{A}}\|^2/k\hat{\rho}$. Denote by $L_{i,j}$ (resp. $L_{S,j}$) the leverage score of the j-th tuple of I in the matrix $\mathbf{A}_i$ (resp. $\hat{\mathbf{A}}$). Further, denote by ρ_i the rank of $\mathbf{A}_i$. Consequently, $p_{i,j} = L_{i,j}/\rho_i$, and our goal is to show $q_j \geq L_{S,j}/(k\hat{\rho})$. Pick any $i^* \in S$ where

$h_{i^*}(I)$ contains the j-th tuple of I, then:

$$q_j = \frac{\sum_{i \in S} p_{i,j}}{s} \geq \frac{p_{i^*,j}}{s} \geq \frac{L_{i^*,j}}{k\rho_i} \geq \frac{L_{S,j}}{k\hat{\rho}} \tag{1}$$

For the last inequality, since $\mathbf{A}_i$ is a sub-matrix of $\hat{\mathbf{A}}$, $\rho_i \leq \hat{\rho}$ and the leverage score decreases due to the monotonicity (Lemma 18).

To conclude, the probabilities will be stored with precision $1/n$ (hence stored using $O(\log n)$ bits each), and the size of the sketch is straightforward to verify. ◀

5 Complex Queries

We study the provisioning of queries that combine logical components (relational algebra and Datalog), with grouping and with the numerical queries that we studied in Section 4.

We start by defining a class of such queries and their semantics formally. For the purposes of this paper, a *complex query* is a triple $\langle Q_L; G_{\bar{A}}; Q_N \rangle$ where Q_L is a relational algebra or Datalog query that outputs some relation with attributes $\bar{A}\bar{B}$ for some $\bar{B}$, $G_{\bar{A}}$ is a *group-by* operation applied on the attributes $\bar{A}$, and Q_N is a numerical query that takes as input a relation with attributes $\bar{B}$. For any input I let $P = \Pi_{\bar{A}}(Q_L(I))$ be the $\bar{A}$-relation consisting of all the distinct values of the grouping attributes. For each tuple $\bar{u} \in P$, we define $\Gamma_{\bar{u}} = \{\bar{v} \mid \bar{u}\bar{v} \in Q_L(I)\}$. Then, the output of the complex query $\langle Q_L; G_{\bar{A}}; Q_N \rangle$ is a set of tuples $\{\langle \bar{u}, Q_N(\Gamma_{\bar{u}}) \rangle \mid \bar{u} \in P\}$.

In the following, we give positive results for the case where the logical component is a *positive relational algebra* (i.e., SPJU) query. It will be convenient to assume a different, but equivalent, formalism for these logical queries, namely that of *unions of conjunctive queries* (UCQs)[3]. We review quickly the definition of UCQs. A *conjunctive query* (CQ) over a relational schema Σ is of the form $ans(\overline{x}) \ :- \ R_1(\overline{x}_1), \ldots, R_b(\overline{x}_b)$, where atoms $R_1, \ldots, R_b \in \Sigma$, and the *size* of a CQ is defined to be the number of atoms in its body (i.e., b). A union of conjunctive query (UCQ) is a finite union of some CQs in which the head has the same schema.

In the following theorem, we show that for any complex query, where the logical component is a positive relational algebra query, compact provisioning of the numerical component implies compact provisioning of the complex query itself.

▶ **Theorem 19.** *For any complex query $\langle Q_L; G_{\bar{A}}; Q_N \rangle$ where Q_L is a* UCQ*, if the numerical component Q_N can be compactly provisioned (resp. compactly ε-provisioned), and if the number of groups is bounded by* $\mathrm{poly}(k, \log n)$*, then the query $\langle Q_L; G_{\bar{A}}; Q_N \rangle$ can also be compactly provisioned (resp. compactly ε-provisioned with the same parameter ε).*

Proof. Suppose Q_N can be compactly provisioned (the following proof also works when Q_N can be compactly ε-provisioned). Let b be the maximum size of the conjunctive queries in Q_L. Given an input instance I and a set H of k hypotheticals, we define a new instance $\hat{I} = Q_L(I)$ and a set $\hat{H}$ of $O(k^b)$ new hypotheticals as follows. For each subset $S \subseteq [k]$ of size at most b (i.e., $|S| \leq b$), define a hypothetical $\hat{h}_S(\hat{I}) = Q_L(I|_S)$ (though S is not a number, we still use it as an index to refer to the hypothetical $\hat{h}_S$). By our definition of the semantics

[3] Although the translation of an SPJU query to a UCQ may incur an exponential size blowup [1], in this paper, query (and schema) size are assumed to be constant. In fact, in practice, SQL queries often present with unions already at top level.

of complex queries, the group-by operation partitions $\hat{I}$ and each $\hat{h}_S$ into $p = |\Pi_{\bar{A}}(\hat{I})|$ groups. We treat each group individually, and create a sketch for each of them.

To simplify the notation, we still use $\hat{I}$ and $\hat{H}$ to denote respectively the portion of the new instance, and the portion of each new hypothetical that correspond to, without loss of generality, the first group. In the following, we show that a compact provisioning scheme for Q_N with input $\hat{I}$ and $\hat{H}$ can be adapted to compactly provision $\langle Q_L; G_{\bar{A}}; Q_N \rangle$ for the first group. Since the number of groups p is assumed to be $\text{poly}(\log |I|, |H|)$, the overall sketch size is still $\text{poly}(\log |I|, |H|)$, hence achieving compact provisioning for the complex query.

Create a sketch for Q_N with input $\hat{I}$ and $\hat{H}$. For any scenario $S \in [k]$ (over H), we can answer the numerical query Q_N using the scenario $\hat{S}$ (over $\hat{H}$) where $\hat{S} = \{S' \mid S' \subseteq S \text{ and } |S'| \leq b\}$. To see this, we only need to show that the input to Q_N remains the same, i.e., $Q_L(I|_S)$ is equal to $\hat{I}|_{\hat{S}}$. Each tuple t in $Q_L(I|_S)$ can be derived using (at most) b hypotheticals. Since any subset of S with at most b hypotheticals belongs to $\hat{S}$, the tuple t belongs to $\hat{I}|_{\hat{S}}$. On the other hand, each tuple t' in $\hat{I}|_{\hat{S}}$ belongs to some $\hat{h}_{S'}$ where $S' \in S$, and hence, by definition of $\hat{h}_{S'}$, the tuple t is also in $Q_L(I|_S)$. Hence, $Q_L(I|_S) = \hat{I}|_{\hat{S}}$.

Consequently, any compact provisioning scheme for Q_N can be adapted to a compact provisioning scheme for the query $\langle Q_L; G_{\bar{A}}; Q_N \rangle$. ◀

Theorem 19 further motivates our results in Section 4 for numerical queries as they can be extended to these quite practical queries. Additionally, as an immediate corollary of the proof of Theorem 19, we obtain that any boolean UCQ (i.e., any UCQ that outputs a boolean answer rather than a set of tuples) can be compactly provisioned.

▶ **Corollary 20.** *Any boolean UCQ can be compactly provisioned using sketches of size $O(k^b)$ bits, where b is the maximum size of each CQ.*

▶ Remark. [13] introduced query provisioning from a practical perspective and proposed *boolean provenance* [27, 20, 19, 33] as a way of building sketches. This technique can also be used for compactly provisioning boolean UCQs (see the full version [3], Remark 5.3).

We further point out that the exponential dependence of the sketch size on the query size (implicit) in Theorem 19 and Corollary 20 cannot be avoided even for CQs (the proof is in the full version [3]; see Theorem 5.5).

▶ **Theorem 21.** *There exists a boolean conjunctive query Q of size b such that provisioning Q requires sketches of size $\min(\Omega(k^{b-1}), \Omega(n))$ bits.*

More general queries. It is natural to ask (a) if Theorem 19 still holds when adding *negation* or *recursion* to the query Q_L (i.e. UCQ *with negation* and *recursive Datalog*, respectively), and (b) whether or not it is possible to provision queries in which logical operations are done *after* numerical ones. A typical example of a query in part (b) is a selection on aggregate values specified by a HAVING clause. Unfortunately, the answer to both questions is negative.

We first show that the answer to question (a) is negative.

▶ **Theorem 22.** *Exact provisioning of (i) boolean conjunctive queries with negation, or (ii) recursive Datalog (even without negation) queries requires sketches of size $\min(2^{\Omega(k)}, \Omega(n))$.*

Proof Sketch. We sketch the proof of each part separately; the complete proofs can be found in the full version [3] (see Theorem 5.6).

Part (i). Define the following boolean conjunctive query with negation over a schema with two unary relation symbols, A and B:

$$Q_{\texttt{NOTSUB}}() \text{ :- } A(x), \neg B(x)$$

This query returns true on I iff there exists some x such that $A(x) \in I$ and $B(x) \notin I$. Intuitively, if we view A and B as subsets of the active domain of I, it is querying whether or not "B is a subset of A". We use a reduction from the Coverage problem to prove the lower bound for $Q_{\texttt{NOTSUB}}$.

From an instance $\{S_1, \ldots, S_k\}$ ($S_i \subseteq [n]$) of Coverage, we create the following instance I for the schema $\Sigma = \{A, B\}$, where for any $x \in [n]$, $A(x), B(x) \in I$. Define the set of of hypotheticals $H = \{h_1, h_2, \ldots, h_{k+1}\}$, where for any $i \in [k]$, $h_i(I) = \{B(x) \mid x \in S_i\}$ and $h_{k+1}(I) = \{A(x) \mid x \in [n]\}$. It is easy to see that for any set $\hat{S} = \{i_1, \ldots, i_s\} \subseteq [k]$, under the scenario $S = \hat{S} \cup \{k+1\}$, $Q_{\texttt{NOTSUB}}(I|_S)$ is true iff there exists $x \in [n]$ s.t. $B(x) \notin I|_S$ which is equivalent to $[n] \not\subseteq S_{i_1} \cup \ldots \cup S_{i_s}$. Therefore any provisioning scheme of $Q_{\texttt{NOTSUB}}$ solves the Coverage problem and the result follows from Theorem 9.

Part (ii). Consider the following Datalog query, st-connectivity:

$$ans() \text{ :- } T(\texttt{t}) \quad (2)$$
$$T(y) \text{ :- } E(x, y), T(x) \quad (3)$$
$$T(\texttt{s}) \text{ :- } \quad (4)$$

This query returns true iff there is a path from the vertex `s` to the vertex `t` in the digraph defined by E. To prove a lower bound for the st-connectivity query we use again a reduction from the Coverage problem.

From an instance $\{S_1, \ldots, S_k\}$ ($S_i \subseteq [n]$) of Coverage, we create the following graph $G(V, E)$ with vertex set $V = \{(s =) v_0, v_1, v_2, \ldots, v_n (= t)\}$, and edges $E(v_{j-1}, v_j)$, for all $j \in [n]$. In G, there is only one path from s to t and that path uses all the n edges. The edge set E of the graph G is the input I to the provisioning scheme. Define the hypotheticals $H = \{h_1, h_2, \ldots, h_k\}$ where $h_i(I) = \{E(v_{j-1}, v_j)\}_{j \in S_i}$, for all $i \in [k]$. For any scenario $S = \{i_1, \ldots, i_s\} \subseteq [k]$, $T(t)$ is true (i.e., s is connected to t) in $I|_S$ iff all n edges are in $I|_S$, which is equivalent to $S_{i_1} \cup \ldots \cup S_{i_s} = [n]$. Therefore any provisioning scheme of st-connectivity solves the Coverage problem and the result follows from Theorem 9. ◀

Showing a negative answer to question (b) is very easy. As we already showed in Theorems 10 and 12, there are numerical queries that do not admit compact provisioning for *exact* answer. One can simply verify that each of those queries can act as a counter example for question (b) by considering HAVING clauses that test the equality of the answer to the numerical part against an exact answer (e.g. testing whether the answer to count is n or not).

6 Conclusions and Future Work

In this paper, we initiated a formal framework to study compact provisioning schemes for relational algebra queries, statistics/analytics including quantiles and linear regression, and complex queries. We considered provisioning for exact as well as approximate answers, and established upper and lower bounds on the sizes of the provisioning sketches.

The queries in our study include quantiles and linear regression queries from the list of in-database analytics highlighted in [25]. This is only a first step and the study of provisioning

for other core analytics problems, such as variance computation, k-means clustering, logistic regression, and support vector machines is of interest.

Another direction for future research is the study of queries in which *numerical* computations follow each other (e.g., when the linear regression training data is itself the result of aggregations). Yet another direction for future research is an extension of our model to allow other kinds of hypotheticals/scenarios as discussed in [13] that are also of practical interest. For example, an alternative natural interpretation of hypotheticals is that they represent tuples to be *deleted* rather than retained. Hence the application of a scenario $S \subseteq [k]$ to I becomes $I|_S = I \setminus (\bigcup_{i \in S} h_i(I))$. Using our lower bound techniques, one can easily show that even simple queries like count or sum cannot be approximated to within any multiplicative factor under this definition. Nevertheless, it will be interesting to identify query classes that admit compact provisioning in the delete model or alternative natural models.

References

1. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
2. Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. In *STOC*, pages 20–29. ACM, 1996.
3. Sepehr Assadi, Sanjeev Khanna, Yang Li, and Val Tannen. Algorithms for provisioning queries and analytics. *CoRR*, abs/1512.06143, 2015. URL: `http://arxiv.org/abs/1512.06143`.
4. Andrey Balmin, Thanos Papadimitriou, and Yannis Papakonstantinou. Hypothetical queries in an OLAP environment. In *VLDB*, pages 220–231, 2000. URL: `http://www.vldb.org/conf/2000/P220.pdf`.
5. Ziv Bar-Yossef, TS Jayram, Ravi Kumar, D Sivakumar, and Luca Trevisan. Counting distinct elements in a data stream. In *RANDOM*. Springer, 2002.
6. Kenneth L Clarkson and David P Woodruff. Numerical linear algebra in the streaming model. In *STOC*, pages 205–214. ACM, 2009.
7. G. Cormode, S. Muthukrishnan, and W. Zhuang. What's different: Distributed, continuous monitoring of duplicate-resilient aggregates on data streams. In *ICDE*, pages 20–31, 2006. `doi:10.1109/ICDE.2006.173`.
8. Graham Cormode, Flip Korn, S Muthukrishnan, and Divesh Srivastava. Space-and time-efficient deterministic algorithms for biased quantiles over data streams. In *PODS*, pages 263–272. ACM, 2006.
9. Graham Cormode and S Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1), 2005.
10. Graham Cormode and S. Muthukrishnan. Space efficient mining of multigraph streams. In *PODS*, pages 271–282, 2005.
11. Graham Cormode, S Muthukrishnan, and Ke Yi. Algorithms for distributed functional monitoring. *ACM Transactions on Algorithms (TALG)*, 7(2):21, 2011.
12. Graham Cormode, S Muthukrishnan, Ke Yi, and Qin Zhang. Optimal sampling from distributed streams. In *PODS*, pages 77–86. ACM, 2010.
13. Daniel Deutch, Zachary G Ives, Tova Milo, and Val Tannen. Caravan: Provisioning for what-if analysis. In *CIDR*, 2013.
14. Petros Drineas, Michael W Mahoney, and S Muthukrishnan. Sampling algorithms for ℓ_2 regression and applications. In *SODA*, pages 1127–1136. ACM, 2006.
15. Petros Drineas, Michael W Mahoney, S Muthukrishnan, and Tamás Sarlós. Faster least squares approximation. *Numerische Mathematik*, 117(2):219–249, 2011.
16. Philippe Flajolet and G Nigel Martin. Probabilistic counting algorithms for data base applications. *Journal of computer and system sciences*, 31(2):182–209, 1985.

17 Shahram Ghandeharizadeh, Richard Hull, and Dean Jacobs. Heraclitus: Elevating deltas to be first-class citizens in a database programming language. *ACM Trans. Database Syst.*, 21(3):370–426, 1996. doi:10.1145/232753.232801.

18 Anna C Gilbert, Yannis Kotidis, S Muthukrishnan, and Martin J Strauss. How to summarize the universe: Dynamic maintenance of quantiles. In *PVLDB*, pages 454–465. VLDB Endowment, 2002.

19 T.J. Green. Containment of conjunctive queries on annotated relations. *Theory Comput. Syst.*, 49(2), 2011.

20 T.J. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *PODS*, pages 31–40, 2007.

21 Greenplum DB (Pivotal). URL: http://pivotal.io/big-data/pivotal-greenplum-database.

22 Michael Greenwald and Sanjeev Khanna. Space-efficient online computation of quantile summaries. *ACM SIGMOD Record*, 30(2):58–66, 2001.

23 Michael B Greenwald and Sanjeev Khanna. Power-conserving computation of order-statistics over sensor networks. In *PODS*, pages 275–285. ACM, 2004.

24 Anupam Gupta and Francis X Zane. Counting inversions in lists. In *SODA*, pages 253–254. Society for Industrial and Applied Mathematics, 2003.

25 Joseph M. Hellerstein, Christopher Ré, Florian Schoppmann, Daisy Zhe Wang, Eugene Fratkin, Aleksander Gorajek, Kee Siong Ng, Caleb Welton, Xixuan Feng, Kun Li, and Arun Kumar. The madlib analytics library or MAD skills, the SQL. *PVLDB*, 5(12):1700–1711, 2012. URL: http://vldb.org/pvldb/vol5/p1700_joehellerstein_vldb2012.pdf.

26 Roger A Horn and Charles R Johnson. *Matrix analysis*. Cambridge university press, 2012.

27 T. Imielinski and W. Lipski. Incomplete information in relational databases. *J. ACM*, 31(4), 1984.

28 Daniel M Kane, Jelani Nelson, and David P Woodruff. An optimal algorithm for the distinct elements problem. In *PODS*, pages 41–52. ACM, 2010.

29 The MADlib Project. URL: http://madlib.net.

30 Michael W Mahoney. Randomized algorithms for matrices and data. *Foundations and Trends® in Machine Learning*, 3(2):123–224, 2011.

31 Gurmeet Singh Manku, Sridhar Rajagopalan, and Bruce G Lindsay. Approximate medians and other quantiles in one pass and with limited memory. *ACM SIGMOD Record*, 27(2):426–435, 1998.

32 Tamas Sarlos. Improved approximation algorithms for large matrices via random projections. In *FOCS*, pages 143–152. IEEE, 2006.

33 D. Suciu, D. Olteanu, C. Ré, and C. Koch. *Probabilistic Databases*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2011.

34 David P Woodruff. Sketching as a tool for numerical linear algebra. *arXiv:1411.4357*, 2014.

35 David P Woodruff and Qin Zhang. Tight bounds for distributed functional monitoring. In *STOC*, pages 941–960. ACM, 2012.

36 Ke Yi and Qin Zhang. Optimal tracking of distributed heavy hitters and quantiles. *Algorithmica*, 65(1):206–223, 2013.