***Abstract*—We present the data model, design choices, and performance of ProvSQL, a general and easy-to-deploy provenance tracking and probabilistic database system implemented as a PostgreSQL extension. ProvSQL's data and query models closely reflect that of a large core of SQL, including multiset semantics, the full relational algebra, and aggregation. A key part of its implementation relies on generic provenance circuits stored in memory-mapped files. We propose benchmarks to measure the overhead of provenance and probabilistic evaluation and demonstrate its scalability and competitiveness with respect to other state-of-the-art systems.**

***Index Terms*—data provenance, probabilistic database, semiring provenance, ProvSQL**

I. Introduction

As data is increasingly used for decision making and training machine learning models, it is crucial to account for two of its main inherent challenges. The first is that *sources matter*. Indeed, more and more importance is given to being able to trace final results back to the original data, to ensure that the results being presented to final users represent the truth. Secondly, in many cases data itself is *uncertain*, be it from the data collection process or as a result of data processing, such as using machine learning. To tackle these issues, it is important to have systems able to keep track of the *provenance* of data, and are also capable to *reason* about its uncertainty.

Data provenance – tracking data throughout a transformation process – has a long history in research, especially within relational data management. The main aims of data provenance are to keep track of *where* results come from, and to show *why* and *how* these results are computed. Early works on provenance [1], [2] reflected on the importance of determining the origin of query output, specifically in the form of *where* and *why* provenance. A related concept is that of data lineage [3], introduced in the Trio system for uncertain data management. A breakthrough was achieved through the *provenance semiring* framework [4] which concisely models many different forms of provenance, including why-provenance and Trio's lineage. In this framework, relations are *annotated* with elements of semirings and relational algebra operators are mapped to semiring operations. Extensions of this framework to more complex queries [5]–[7] have since been proposed.

Managing data uncertainty was another parallel line of research. Uncertainty was initially accounted for via concepts such as NULL values [8] and c-tables [9]. More recently, tuple-independent databases, where each tuple is independently annotated with a probability value, were introduced by Dalvi and Suciu [10], and extended to block-independent databases in [11], as a simple probabilistic model of uncertainty. Even in these simple frameworks, *probabilistic query evaluation*, i.e., computing the probability of a query result, is a #P-hard problem. Some tractable cases are however known: when queries have some specific properties (e.g., *safe queries* over tuple-independent databases [12]), or when data itself has a tree-like structure [13], captured by the notion of treewidth [14]. We are focusing in this work on the probabilistic database approach to modeling uncertainty, though we note that other approaches exist in the literature, e.g., [15], [16].

Though most of the works cited so far are theoretical in nature, various systems implement some form of provenance and probability evaluation or both: examples include Trio [3], Orchestra [17], Orion [18], MayBMS [19], Perm [20], and more recently GProM [21]. Except GProM, these systems are unfortunately unmaintained, have become hard to deploy or are even defunct. None of these systems provide a generic tool able to both keep track of the probability and provenance of data, for a wide variety of provenance frameworks.

In this article, we explain how the rich literature on provenance and probabilistic query evaluation is a basis for the implementation of ProvSQL, a plugin for the PostgreSQL database management system. ProvSQL aims to provide a generic, easy-to-deploy, and scalable solution to store and evaluate data provenance and probabilities. A proof-of-concept of ProvSQL was the subject of an early demonstration paper [22] in 2018; significant changes were brought to ProvSQL since: a complete architecture change (described in Section V-A), support for aggregation queries, computation of probability relying on tree decomposition techniques (see Section V-D), etc. ProvSQL was used as a provenance computation tool in several lines of research, by a variety of authors [23]–[28].

This paper is the first to provide a comprehensive presentation of its data model, query evaluation approach, and implementation aspects, as well as its real-world performance. Specifically, we provide the following contributions:

(i) We detail how the **theoretical results on provenance and probability are applied to real-world systems** using the SQL data model, using a generic representation of provenance and relying on a multiset semantics, and by evaluating the probability of Boolean provenance formulas for probabilistic query evaluation. We also discuss practical implementations of

extensions to the basic semiring model via the monus operator [5] and semimodules for aggregates [6].

(ii) We present **design choices in ProvSQL** for provenance and probability computation, such as rewriting queries for provenance tracking, memory-mapped storage of provenance circuits, and knowledge compilation for probability computation.

(iii) To enable fair comparison with other systems and to quantitatively assess the performance of ProvSQL in real-world scenarios, we **propose publicly available benchmarks**, inspired from the TPC-H relational query benchmark, for both provenance and probabilistic query evaluation.

(iv) We **evaluate the performance of ProvSQL** on these benchmarks, both by itself (by measuring the provenance and probability overhead ProvSQL adds to PostgreSQL) and by comparing it to other systems that provide some of the features of ProvSQL. Specifically, we compare to GProM for provenance management and MayBMS for probabilistic query evaluation.

Our findings show that ProvSQL handles provenance and probabilistic query evaluation at scale (on multi-gigabyte databases), with reasonable overhead. Performance varies by query, leaving room for optimization. Compared with similar systems, ProvSQL supports a larger subset of SQL (including multiset semantics and aggregation) and works with arbitrary semirings. Its compact provenance circuits let it scale better than GProM. For probability computation, MayBMS is competitive with ProvSQL *on those queries MayBMS support*; however, ProvSQL often performs similarly even though it does not use any query-based optimization as MayBMS does.

ProvSQL is available as open source[1]. A companion repository[2] complements this paper, including benchmark scripts, additional details, proofs of all results, and links to formal definitions and proofs for the Lean proof assistant (also identified and linked to by LEAN throughout the paper).

II. Related Systems

Trio [3] was an early system for managing uncertain data, focusing on the representation on different forms of uncertainty, including probabilities. It does not support computation of arbitrary marginals, which is at the core of modern probabilistic databases. Though it adopts a mediator approach, it is tied to specific and obsolete versions of PostgreSQL (8.2 or 8.3).

Obsolescence is a common problem for systems of this era: the provenance tracking system Perm [20] and the probabilistic datbase system Orion [18] are respectively unmaintained forks of PostgreSQL 8.3 and 8.4; the distributed Orchestra system [17], which provides an early implementation of provenance semirings, cannot be compiled because some of its dependencies are on servers that have disappeared; MystiQ [29], implementing safe query plan evaluation, requires Java 5.0, which reached end-of-service in 2009.

MayBMS [19][3] is a probabilistic database system implementing the general and compact model of U-relations [30]. Its query evaluator, SPROUT [31] was designed for efficiency and is in particular able to exploit the structure of *safe* queries [12]. It was developed as a fork of PostgreSQL 8.3, which is obsolete and hard to deploy. Some effort has been made in keeping the system compilable, however, though it requires using a virtual machine running an older operating system. It does not support provenance semirings. We use MayBMS in our experiments as a state-of-the-art system for probabilistic query evaluation.

GProM [21][4] is a middleware layer that adds provenance support to various database backends (in particular, Oracle and PostgreSQL). It translates declarative queries with provenance requirements into SQL code, which is subsequently executed by the backend database system. GProM supports the capture of provenance for SQL queries, as well as some Datalog queries. Some features not present in ProvSQL are support for transactions and PROV-JSON serialization [32]. On the other hand, in contrast with ProvSQL, it does not support probabilistic query evaluation, arbitrary semiring provenance, or provenance of non-monotone queries. As it is modern, actively maintained, and feature-rich, we use it as a comparison point in Section VI.

ProbLog [33][5] is an actively maintained and easy-to-use probabilistic programming system, inspired from Prolog and Datalog. SQL queries and probabilistic databases can be encoded into ProbLog in a straightforward way. Data can also be stored in a SQLite database and ProbLog uses knowledge compilation to compute probabilities of program outputs. However, our experiments showed it does not scale, as even though the data is stored in a database, it is loaded in main memory when needed by the query evaluator, and none of the usual query optimization infrastructure is used. Indeed, on the simplest query of our benchmark (query 1 of $\mathcal{Q}^{\text{cust}}$, see Section VI), ProbLog ran out-of-memory on the smallest database scale factor used in our experiments (1 GB). On a database 10 times smaller, it did not complete after 5 hours.

III. Extended Relational Algebra

We now present the query language underlying ProvSQL; it is an extension of the relational algebra with multiset (or *bag*) semantics, allowing for aggregation. We first give basic definitions and notation about multisets.

Definition 1. *A* multiset *m over a set V is a function $V \to \mathbb{N}^*$. It is is* finite *if it has finite domain. For $v \in V$, we note $v \in m$ if $m(v) > 0$ and $v \notin m$ otherwise. Given two multisets m_1 and m_2 over V, we define the* multiset union *of m_1 and m_2 as $m_1 \uplus m_2 : x \mapsto m_1(x) + m_2(x)$ and the* cross product *of m_1 and m_2 as the multiset over V^2 defined by $m_1 \times m_2 : (x_1, x_2) \mapsto m_1(x_1) \times m_2(x_2)$. Any set V can be seen as the multiset over V where $m(v) = 1$ for all $v \in V$.*

A finite multiset with domain $\{a_1, \ldots, a_n\}$ can be written in the form $\left\{\!\!\left\{ \underbrace{a_1, \ldots, a_1}_{m(a_1)\text{ times}}, \ldots, \underbrace{a_n, \ldots, a_n}_{m(a_n)\text{ times}} \right\}\!\!\right\}$. Finally, we also use the notation $\{\!\{ f(x) \mid x \in m \}\!\}$ or $\biguplus_{x \in m} \{\!\{ f(x) \}\!\}$ for a finite multiset m with domain

$\{a_1, \ldots, a_n\}$ *and some function* f *over* V *to mean the multiset* $\left\{\!\!\left\{ \underbrace{f(a_1), \ldots, f(a_1)}_{m(a_1)\ times}, \ldots, \underbrace{f(a_n), \ldots, f(a_n)}_{m(a_n)\ times} \right\}\!\!\right\}$.

We define an algebra for queries over relational databases under the multiset semantics which is as close as possible to a simple subset of SQL as dealt with by standard DBMSs. Our algebra is based on the relational algebra with set semantics typically used in database theory [34] but includes operators to explicitly control duplicate elimination as in [35], [36] and to express aggregation [37].

To simplify the presentation, we adopt an ordered, unnamed, and untyped perspective: attributes are referred to by their positions within the relation, do not have a name, and are assumed to be from a universal domain $\mathcal{V}$. In practical SQL implementations, attributes do have names and types, but it is straightforward to propagate them throughout query execution.

Let us fix a set $\mathcal{L}$ of relation labels and a set $\mathcal{V}$ of values. A database schema $\mathcal{D} : \mathcal{L} \to \mathbb{N}$ assigns arities to a *finite* set of relation labels. A *relation of arity* $k \in \mathbb{N}$ is a *finite multiset* of tuples over $\mathcal{V}^k$. An instance of a database schema $\mathcal{D}$, or a database over $\mathcal{D}$ for short, is a function mapping each relation name R in the domain of $\mathcal{D}$ to a relation of arity $\mathcal{D}(R)$.

A *positional index* is an expression of the form "#i" for $i \in \mathbb{N}^*$. A *term* is any expression involving positional indices and values from $\mathcal{V}$, combined using arbitrary operators and functions over values in $\mathcal{V}$. The *max-index* of a term is the maximum of all i such that "#i" appears in the term (or 0 if none appear). Given a tuple $u = (u_1, \ldots, u_k)$ and a term t of max-index $\leqslant k$, we define $t(u)$ as the value of $\mathcal{V}$ obtained by replacing in t every positional index #i with u_i and evaluating the whole expression.

Given a database schema $\mathcal{D}$, we recursively define the language RA_k of relational algebra queries of arity $k \in \mathbb{N}$ as follows [⊔∀∧]:

relation for any relation R in the domain of $\mathcal{D}$, $R \in \mathsf{RA}_{\mathcal{D}(R)}$;
projection for $k \in \mathbb{N}$, $q \in \mathsf{RA}_k$, $t_1, \ldots, t_n$ terms of max-index $\leqslant k$, $\Pi_{t_1,\ldots,t_n}(q) \in \mathsf{RA}_n$;
selection for $k \in \mathbb{N}$, $q \in \mathsf{RA}_k$, and φ a Boolean combination of (in)equality comparisons between terms of max-index $\leqslant k$, $\sigma_\varphi(q) \in \mathsf{RA}_k$;
cross product for $k_1, k_2 \in \mathbb{N}$, $q_1 \in \mathsf{RA}_{k_1}$, $q_2 \in \mathsf{RA}_{k_2}$, $q_1 \times q_2 \in \mathsf{RA}_{k_1+k_2}$;
multiset sum for $k \in \mathbb{N}$, $q_1, q_2 \in \mathsf{RA}_k$, $q_1 \uplus q_2 \in \mathsf{RA}_k$;
duplicate elimination for $k \in \mathbb{N}$, $q \in \mathsf{RA}_k$, $\varepsilon(q) \in \mathsf{RA}_k$;
multiset difference for $k \in \mathbb{N}$, $q_1, q_2 \in \mathsf{RA}_k$, $q_1 - q_2 \in \mathsf{RA}_k$;
aggregation for $k \in \mathbb{N}$, $q \in \mathsf{RA}_k$, distinct $(i_j)_{1 \leqslant j \leqslant k}$, terms $t_1, \ldots t_n$ of max-index $\leqslant k$, functions $f_1, \ldots, f_n$ from finite multisets of values to values, $\gamma_{i_1,\ldots,i_m}[t_1 : f_1, \ldots, t_n : f_n](q) \in \mathsf{RA}_{m+n}$.

Some other operators can be seen as syntactic sugar: **join** $q_1 \bowtie_\varphi q_2 \stackrel{\text{def}}{=} \sigma_\varphi(q_1 \times q_2)$; **set union** $q_1 \cup q_2 \stackrel{\text{def}}{=} \varepsilon(q_1 \uplus q_2)$.

The semantics $[\![\cdot]\!]^I$ of queries on an instance I over $\mathcal{D}$ is defined as expected (see companion repository for details) for most operators. Note, however, that the definition of the multiset difference is not the usual one, and also not the one corresponding to the `EXCEPT ALL` operator of SQL, though it does correspond to the important `NOT IN` operator of SQL. This is mainly for practicality: the standard definition of multiset difference where $[\![q_1 - q_2]\!]^I(t) = \max(0, [\![q_1]\!]^I(t) - [\![q_2]\!]^I(t))$ leads to intractability of the provenance computation. The difference disappears in the set semantics: the semantics of $\varepsilon(q_1 - q_2)$ matches with that of `EXCEPT`.

Importantly, in this work, we assume that the aggregation operator, if it is used, is applied last (at the top-level operator), similarly to what is done in Section 3 of [6]; nested aggregation queries, discussed in Section 4 of [6], are implemented in ProvSQL, but as there is currently no clear semantics for semiring provenance for arbitrary semirings over nested aggregation queries, we leave them out-of-scope of this work.

Table I: The *Personnel* relation (derived from [38])

id	name	position	city	
1	Juma	Director	Nairobi	t_1
2	Paul	Janitor	Nairobi	t_2
3	David	Analyst	Paris	t_3
4	Ellen	Field agent	Beijing	t_4
5	Aaheli	Double agent	Paris	t_5
6	Nancy	HR	Paris	t_6
7	Jing	Analyst	Beijing	t_7

Example 2. *Consider the instance I of the relation Personnel given in Table I, containing names of people, their position and their city (remember attribute names are not part of the model, they are just given here for ease of reading). Ignoring for now the t_i's, the relation has arity 4. Consider the following query:* "What are the cities where at least two persons are working?". *This can be expressed (without using aggregation) as:* $q_{\text{city}} \stackrel{def}{=} \varepsilon(\Pi_{\#4}(\textit{Personnel} \bowtie_{\#4=\#8 \wedge \#1<\#5} \textit{Personnel}))$. *Note the need for duplicate elimination ε due to the multiset semantics. Then* $[\![q_{\text{city}}]\!]^I = \{\!\{ (\text{Nairobi}), (\text{Paris}), (\text{Beijing}) \}\!\}$.

IV. Querying Annotated Relations

Now that our query language is fixed, we introduce the notion of annotated relations (Section IV-A) and the semantics of queries over such annotated relations in Section IV-B. This is based on annotated relations defined for provenance semirings as in [4], but with two key differences: we go beyond semirings for annotation, namely using the m-semirings from [5] and the δ-semirings of [6], and, as in SQL, relations are multisets and not sets. We then show in Section IV-C how the semantics of provenance over annotated relations can be captured by query rewriting. Finally, we define probabilistic query evaluation and show how it can be performed in an intensional way relying on provenance (Section IV-D).

A. Semirings and Beyond

Given some algebraic structure $\mathbb{K}$ for *annotations* (typically, a semiring or a generalization thereof) and some set $\mathcal{V}_\mathbb{K} \supseteq \mathcal{V}$ of *values* (typically, either $\mathcal{V}$ itself or an extension of $\mathcal{V}$ including $\mathbb{K}$-semimodules), a $\mathbb{K}$-*relation of arity* $k \in \mathbb{N}$ over $\mathcal{V}_\mathbb{K}$ is a

multiset of tuples over $(\mathcal{V}_{\mathbb{K}})^k \times \mathbb{K}$, often written (u, α) with $u \in (\mathcal{V}_{\mathbb{K}})^k$ and $\alpha \in \mathbb{K}$. A $\mathbb{K}$-instance of a database schema $\mathcal{D}$, or $\mathbb{K}$-database over $\mathcal{D}$ for short, is a function mapping each relation name R in the domain of $\mathcal{D}$ to a $\mathbb{K}$-relation of arity $\mathcal{D}(R)$. ∃∀ℕ Such $\mathbb{K}$-instances are typically denoted $\hat{I}$, and when we use such a notation I then means the relation projected on the first k columns, without the annotation.

We define in particular:

Definition 3. *A* semiring $(\mathbb{K}, \oplus, \otimes, \mathbb{0}, \mathbb{1})$ *is a set* $\mathbb{K}$ *of elements along with two binary operators over* $\mathbb{K}$ *(*$\oplus$ *and* $\otimes$*) and two distinguished elements of* $\mathbb{K}$ *(*$\mathbb{0}$ *and* $\mathbb{1}$*), such that:*

(i) $(\mathbb{K}, \oplus, \mathbb{0})$ *is a commutative monoid;*

(ii) $(\mathbb{K}, \otimes, \mathbb{1})$ *is a (non-necessarily commutative) monoid;*

(iii) $\otimes$ *distributes over* $\oplus$*:* $\forall a, b, c \in \mathbb{K}$, $a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$ *and* $(a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c)$*;*

(iv) $\mathbb{0}$ *is annihilator for* $\otimes$*:* $\forall a\ \mathbb{0} \otimes a = a \otimes \mathbb{0} = \mathbb{0}$.

Example 4. *The following structures are semirings:*

Counting semiring $(\mathbb{N}, +, \times, 0, 1)$

Boolean function semiring *for a finite set* X, $(\mathcal{B}[X], \hat{\vee}, \hat{\wedge}, \hat{\bot}, \hat{\top})$ *where* $\mathcal{B}[X]$ *is the set of functions mapping valuations over* X *(i.e., functions from* X *to* $\{\bot, \top\}$*) to either* $\bot$ *(false) or* $\top$ *(true),* $f \hat{\vee} g$ *(resp.,* $f \hat{\wedge} g$*) is the function that maps a valuation* ν *to* $f(\nu) \vee g(\nu)$ *(resp.,* $f(\nu) \wedge g(\nu)$*), and* $\hat{\bot}$ *(resp.,* $\hat{\top}$*) is the constant function returning always* $\bot$ *(resp.,* $\top$*)*

Why-provenance [2] *for a finite set* X, $(2^{2^X}, \varnothing, \{emptyset\}, \cup, \Cup)$ *where* $\Cup$ *is defined by* $A \Cup B \stackrel{def}{=} \{a \cup b \mid a \in A, b \in B\}$

Definition 5. ∃∀ℕ *A* semiring with monus $\mathbb{K}$*, or* m-semiring *is a semiring along with an extra binary operation* $\ominus$ *such that for all* $a, b, c \in \mathbb{K}$*: (i)* $a \oplus (b \ominus a) = b \oplus (a \ominus b)$*; (ii)* $(a \ominus b) \ominus c = a \ominus (b \oplus c)$*; (iii)* $a \ominus a = \mathbb{0} \ominus a = \mathbb{0}$.

The semirings from Example 4 can be extended with a monus operator to form an m-semiring: this was noted in [5] for $\mathbb{N}$ (where $\ominus$ is the truncated difference ($a \ominus b \stackrel{\text{def}}{=} \max(a-b, 0)$) and in $\mathcal{B}[X]$ (where it is $(a, b) \mapsto a \wedge \neg b$). For Why-provenance:

Proposition 6. ∃∀ℕ *For a set* X, $(2^{2^X}, \varnothing, \{\varnothing\}, \cup, \Cup, \setminus)$ *is an m-semiring.*

For computing the provenance of aggregate queries, [6] introduces an extra operator: a *δ-semiring* is a semiring along with a unary operation δ such that: (i) $\delta(\mathbb{0}) = \mathbb{0}$; (ii) $\delta(\mathbb{1} \oplus \cdots \oplus \mathbb{1}) = \mathbb{1}$ whatever the number of $\mathbb{1}$s as input to δ. For a discussion on choosing such a δ function, see [6]. A simple choice we will use for all semirings is the function that maps $\mathbb{0}$ to $\mathbb{0}$ and anything else to $\mathbb{1}$.

Finally, also to capture the provenance of aggregate queries, we will need to restrict which aggregate functions can be used:

Definition 7. *A* monoid aggregate function f *over values in a set* V *is a monoid homomorphism from the monoid of finite multisets of values of* V *(with multiset union as monoid operation) to some monoid over* V*. In other words,* $f : (V \to \mathbb{N}^*) \to V$ *is a monoid aggregate function if there exists a monoid* $(V, \cdot, e)$ *such that* $f(\{\!\{\ \}\!\}) = e$ *and* $f(S \uplus T) = f(S) \cdot f(T)$.

Example 8. *The function* count*, that turns a multiset* m *into the sum of all* $m(v)$ *for* v *in the domain of* m *is a monoid aggregate function from multisets of arbitrary values to* $(\mathbb{N}, +)$*. Similarly,* sum *is a monoid aggregate function from multisets over, say,* $\mathbb{Q}$ *to* $(\mathbb{Q}, +)$*, and* min *from multisets over* $\mathbb{Q}$ *to* $(\mathbb{Q}, \min)$.

B. Algebra over Annotated Relations

We define the semantics $\langle\!\langle \cdot \rangle\!\rangle^{\hat{I}}$ of extended relational algebra queries on a $\mathbb{K}$-instance $\hat{I}$ over $\mathcal{D}$ by induction; for the definition to be meaningful, $\mathbb{K}$ needs to be a semiring; a m-semiring for the multiset difference operator; and a δ-semiring for the aggregation operator. (We say that $\mathbb{K}$ is *appropriate* for q if this is the case.) ∃∀ℕ

relation for any relation label R in the domain of $\mathcal{D}$, $\langle\!\langle R \rangle\!\rangle^{\hat{I}} \stackrel{\text{def}}{=} \hat{I}(R)$;

projection for $k \in \mathbb{N}$, $q \in \mathsf{RA}_k$, $t_1, \ldots, t_n$ terms of max-index $\leqslant k$, $\langle\!\langle \Pi_{t_1,\ldots,t_n}(q) \rangle\!\rangle^{\hat{I}} \stackrel{\text{def}}{=} \left\{\!\!\left\{ (t_1(u), \ldots, t_n(u), \alpha) \mid (u, \alpha) \in \langle\!\langle q \rangle\!\rangle^{\hat{I}} \right\}\!\!\right\}$;

selection for $k \in \mathbb{N}$, $q \in \mathsf{RA}_k$, and φ a Boolean combination of (in)equality comparisons between terms of max-index $\leqslant k$, $\langle\!\langle \sigma_\varphi(q) \rangle\!\rangle^{\hat{I}} \stackrel{\text{def}}{=} \left\{\!\!\left\{ (u, \alpha) \mid (u, \alpha) \in \langle\!\langle q \rangle\!\rangle^{\hat{I}}, \varphi(u) \right\}\!\!\right\}$;

cross product for $k_1, k_2 \in \mathbb{N}$, $q_1 \in \mathsf{RA}_{k_1}$, $q_2 \in \mathsf{RA}_{k_2}$, $\langle\!\langle q_1 \times q_2 \rangle\!\rangle^{\hat{I}} \stackrel{\text{def}}{=} \left\{\!\!\left\{ (u, v, \alpha_1 \otimes \alpha_2) \mid (u, \alpha_1, v, \alpha_2) \in \langle\!\langle q_1 \rangle\!\rangle^{\hat{I}} \times \langle\!\langle q_2 \rangle\!\rangle^{\hat{I}} \right\}\!\!\right\}$;

multiset sum for $k \in \mathbb{N}$, $q_1, q_2 \in \mathsf{RA}_k$, $\langle\!\langle q_1 \uplus q_2 \rangle\!\rangle^{\hat{I}} \stackrel{\text{def}}{=} \langle\!\langle q_1 \rangle\!\rangle^{\hat{I}} \uplus \langle\!\langle q_2 \rangle\!\rangle^{\hat{I}}$;

duplicate elimination for $k \in \mathbb{N}$, $q \in \mathsf{RA}_k$, $\langle\!\langle \varepsilon(q) \rangle\!\rangle^{\hat{I}} \stackrel{\text{def}}{=} \bigcup_{u \mid \exists \alpha (u,\alpha) \in \langle\!\langle q \rangle\!\rangle^{\hat{I}}} \left\{ \left(u, \bigoplus_{\alpha \mid (u,\alpha) \in \langle\!\langle q \rangle\!\rangle^{\hat{I}}} \alpha \right) \right\}$;

multiset difference for $k \in \mathbb{N}$, $q_1, q_2 \in \mathsf{RA}_k$, $\langle\!\langle q_1 - q_2 \rangle\!\rangle^{\hat{I}} \stackrel{\text{def}}{=} \left\{\!\!\left\{ \left(u, \alpha \ominus \bigoplus_{\beta \mid (u,\beta) \in \langle\!\langle q_2 \rangle\!\rangle^{\hat{I}}} \beta \right) \,\middle|\, (u, \alpha) \in \langle\!\langle q_1 \rangle\!\rangle^{\hat{I}} \right\}\!\!\right\}$;

aggregation for $k \in \mathbb{N}$, $q \in \mathsf{RA}_k$, distinct $(i_j)_{1 \leqslant j \leqslant k}$, terms $t_1, \ldots t_n$ of max-index $\leqslant k$, monoid aggregate functions $f_1, \ldots, f_n$,

$\langle\!\langle \gamma_{i_1,\ldots,i_m}[t_1 : f_1, \ldots, t_n : f_n](q) \rangle\!\rangle^{\hat{I}} \stackrel{\text{def}}{=}$

$\Big\{ (\nu_1,\ldots,\nu_m, \hat{f}_1\big(\{\!\{ t_1(u) * \alpha \mid (u,\alpha) \in \langle\!\langle q \rangle\!\rangle^{\hat{I}}, (u_{i_1},\ldots,u_{i_m}) = (\nu_1,\ldots,\nu_m) \}\!\}\big), \ldots, \hat{f}_n\big(\{\!\{ t_n(u) * \alpha \mid (u,\alpha) \in \langle\!\langle q \rangle\!\rangle^{\hat{I}}, (u_{i_1},\ldots,u_{i_m}) = (\nu_1,\ldots,\nu_m) \}\!\}\big), \delta(\beta)) \,\Big|\, (\nu_1,\ldots,\nu_m,\beta) \in \langle\!\langle \varepsilon(\Pi_{\#i_1,\ldots,\#i_m}(q)) \rangle\!\rangle^{\hat{I}} \Big\}$ where:

- "$*$" denotes a tensor product used to combine the data values of $\mathcal{V}$ with the δ-semiring annotations from $\mathbb{K}$ in a semimodule $\mathcal{V}_{\mathbb{K}}$;
- for any f_i, $\hat{f}_i$ is a new aggregate function lifted from values in $\mathcal{V}$ to values in $\mathcal{V}_{\mathbb{K}}$

See [6] for details about provenance semimodules.

Example 9. *We return to Example 2 and to the instance of the Personnel table of Table I. The* t_i*'s are now interpreted as tuple annotations in some semiring* $\mathbb{K}$*, resulting in a* $\mathbb{K}$*-relation* $\hat{I}$*. Following the semantics of* q_{city} *over annotated relations, we compute* $\langle\!\langle q_{\text{city}} \rangle\!\rangle^{\hat{I}}$ *as:*

Nairobi	$t_1 \otimes t_2$
Paris	$(t_3 \otimes t_5) \oplus (t_5 \otimes t_6) \oplus (t_3 \otimes t_6)$
Beijing	$t_4 \otimes t_7$

If we choose for $\mathbb{K}$ *the semiring* $\mathcal{B}[X]$ *where* $X = \{t_1, \ldots, t_7\}$*, the annotation for* Paris *is the Boolean function given by the formula* $(t_3 \wedge t_5) \vee (t_5 \wedge t_6) \vee (t_3 \wedge t_6)$*. This is a form of* Boolean provenance *[38]:* Paris *is in the result iff either both tuples representing* David *and* Aaheli*, or* Aaheli *and* Nancy*, or* David *and* Nancy*, are present.*

C. Query Rewriting

The key way ProvSQL implements the semantics of the extended algebra over annotated relations is by *rewriting the query* over annotated relations to include in the query the necessary operations on provenance operations. The query can then be evaluated using the standard query evaluator of PostgreSQL. We employ the following 5 rewriting rules, which are applied inductively on an extended relational algebra formula: ∃∀N

(R1) for $k \in \mathbb{N}$, $q \in \mathsf{RA}_k$, $t_1, \ldots, t_n$ terms of max-index $\leqslant k$, $\Pi_{t_1,\ldots,t_n}(q)$ is rewritten to $\Pi_{t_1,\ldots,t_n,\#(k+1)}(q)$

(R2) For $k_1, k_2 \in \mathbb{N}$, $q_1 \in \mathsf{RA}_{k_1}$, $q_2 \in \mathsf{RA}_{k_2}$, $q_1 \times q_2$ is rewritten to: $\Pi_{\#1,\ldots,\#k_1,\#(k_1+2),\ldots,\#(k_1+k_2+1),\#(k_1+1)\otimes\#(k_1+k_2+2)}(q_1 \times q_2)$.

(R3) For $k \in \mathbb{N}$, $q \in \mathsf{RA}_k$, $\varepsilon(q)$ is rewritten to: $\gamma_{1,\ldots,k}[\#(k+1) : \bigoplus](q)$.

(R4) For $k \in \mathbb{N}, q_1, q_2 \in \mathsf{RA}_k$, $q_1 - q_2$ is rewritten to:

$$\Pi_{\#1,\ldots,\#(k+1)}(q_1 \bowtie_{\#1=\#(k+2)\wedge\cdots\wedge\#k=\#(2k+1)} \varepsilon(\Pi_{\#1,\ldots,\#k}(q_1) - \Pi_{\#1,\ldots,\#k}(q_2))) \uplus \Pi_{\#1,\ldots,\#k,\#(k+1)\ominus\#(2k+2)}(q_1 \bowtie_{\#1=\#(k+2)\wedge\cdots\wedge\#k=\#(2k+1)} \gamma_{\#1,\ldots,\#k}[\#(k+1):\oplus](q_2))$$

(R5) For $k \in \mathbb{N}$, $q \in \mathsf{RA}_k$, distinct $(i_j)_{1\leqslant j\leqslant k}$, terms $t_1, \ldots, t_n$ of max-index $\leqslant k$, monoid aggregate functions $f_1, \ldots, f_n$, $\gamma_{i_1,\ldots,i_m}[t_1 : f_1, \ldots, t_n : f_n](q)$ is rewritten to: $\gamma_{i_1,\ldots,i_m}[t_1 * \#(k+1) : \hat{f}_1, \ldots, t_n * \#(k+1) : \hat{f}_n, \#(k+1) : \delta(\bigoplus)](q)$.

Note that relation names, selection, and multiset sum operators are not rewritten. Using query rewriting to perform provenance evaluation has been proposed in previous works (see, in particular, Amsterdamer et al. [6], Fink and Olteanu [39] and Perm [20]). The two distinguishing aspects of this work is that we natively support the multiset semantics that SQL engines natively implement (requiring adaptation of rewriting rules for projection and explicit duplicate elimination) and that we support difference.

We show that these rewriting rules allow us to recover the exact semantics of the extended relational algebra over annotated relations:

Theorem 10. ∃∀N *Let* $\mathcal{D}$ *be a database schema,* q *any extended relational algebra query over* $\mathcal{D}$*,* $\mathbb{K}$ *an appropriate algebraic structure, and* $\hat{I}$ *a* $\mathbb{K}$*-instance over* $\mathcal{D}$*. Let* $\hat{q}$ *be the query rewritten from* q *by applying the rewriting rules (R1)–(R5) recursively bottom up. Then* $\langle\!\langle q \rangle\!\rangle^{\hat{I}} = [\![\hat{q}]\!]^{\hat{I}}$.

Example 11. *Returning to query* q_{city} *from Example 2, let us trace the rewriting rules applied by ProvSQL bottom up (for space reasons, Personnel is abbreviated to P):*

$$\begin{aligned} q_{\text{city}} &= \varepsilon(\Pi_{\#4}(P \bowtie_{\#4=\#8\wedge\#1<\#5} P)) \\ &= \varepsilon(\Pi_{\#4}(\sigma_{\#4=\#8\wedge\#1<\#5}(P \times P))) \\ &\xrightarrow{(R2)} \varepsilon(\Pi_{\#4}(\sigma_{\#4=\#8\wedge\#1<\#5}(\Pi_{\#1,\ldots,\#4,\#6,\ldots\#9,\#5\otimes\#10}(P \times P)))) \\ &\xrightarrow{(R1)} \varepsilon(\Pi_{\#4,\#9}(\sigma_{\#4=\#8\wedge\#1<\#5}(\Pi_{\#1,\ldots,\#4,\#6,\ldots\#9,\#5\otimes\#10}(P \times P)))) \\ &\xrightarrow{(R3)} \gamma_1[\#2 : \bigoplus](\\ &\qquad \Pi_{\#4,\#9}(\sigma_{\#4=\#8\wedge\#1<\#5}(\Pi_{\#1,\ldots,\#4,\#6,\ldots\#9,\#5\otimes\#10}(P \times P)))) \end{aligned}$$

If $\hat{q}_{\text{city}}$ *is this rewritten query, one can check that* $[\![\hat{q}_{\text{city}}]\!]^{\hat{I}} = \langle\!\langle q_{\text{city}} \rangle\!\rangle^{\hat{I}}$.

D. Probabilistic Query Evaluation

Annotated relations can be used for probabilistic query evaluation using the so-called intensional approach [40]. First, assume a finite set X of variables, each variable $x \in X$ being assigned a probability $\Pr(x)$. Pr can be extended to a probability distribution over valuations over X, assuming independence of variables: $\Pr(\nu) = \prod_{x|\nu(x)=\top} \nu(x) \times \prod_{x|\nu(x)=\bot}(1-\nu(x))$. This in turns extends to a probability distribution over Boolean functions: if $f \in \mathcal{B}[X]$ is a Boolean function over variables in X, then $\Pr(f) = \sum_{\nu|f(\nu)=\top} \Pr(\nu)$.

Consider a $\mathcal{B}[X]$-relation $\hat{I}$ of arity k. For any subinstance $\hat{J}$ of $\hat{I}$, the characteristic Boolean function of $\hat{J}$ within $\hat{I}$ is: $\Phi_{\hat{I}}(\hat{J}) \stackrel{\text{def}}{=} \bigwedge_{(u,\alpha)\in\hat{J}} \alpha \wedge \bigwedge_{(u,\alpha)\in\hat{I},(u,\alpha)\notin J} \neg\alpha$. The probability of $\hat{J}$ is then $\Pr(\Phi_{\hat{I}}(\hat{J}))$. For a query q over $\hat{I}$ in RA_k, we define the marginal probability that a tuple t of arity k appears in the output of q as $\Pr(t \in q(\hat{I})) \stackrel{\text{def}}{=} \sum_{\hat{J}\subseteq\hat{I},t\in[\![q]\!]^J} \Pr(\hat{J})$.

Our semantics for extended relational algebra over $\mathbb{K}$-relations is compatible with probabilistic query evaluation:

Theorem 12. *For any finite set of variables* X*, probability distribution* Pr *over* X*,* $\mathcal{B}[X]$*-relation* $\hat{I}$ *and relational algebra query* q *without aggregation, for any tuple* t *with same arity as* q*,* $\Pr(t \in q(\hat{I})) = \Pr\left(\bigvee_{(t,\alpha)\in\langle\!\langle q\rangle\!\rangle^{\hat{I}}} \alpha\right)$.

The reason why this theorem is about queries without aggregation is that the result of aggregate queries are tuples whose values include annotations; for a similar result, we would need to talk about the distribution of data values, or some summary thereof (such as the expected value). This is out of scope of this work.

Combining Theorems 10 and 12, we obtain:

Corollary 13. *For any finite set of variables* X*, probability distribution* Pr *over* X*,* $\mathcal{B}[X]$*-relation* $\hat{I}$ *and relational algebra query* q *without aggregation, for any tuple* t *with same arity as* q*, if* $\hat{q}$ *is the query rewritten from* q *by applying the rewritten rules (R1)–(R5) then* $\Pr(t \in q(\hat{I})) = \Pr\left(\bigvee_{(t,\alpha)\in[\![\hat{q}]\!]^{\hat{I}}} \alpha\right)$.

Example 14. *Returning again to Examples 2 and 9, assume we independently assign to each* t_i *in our Personnel* $\mathcal{B}[\{t_1, \ldots, t_7\}]$*-instance a probability* $\Pr(t_i)$ *with* $\Pr(t_1) = 0.5$

and $\Pr(t_2) = 0.7$. *We can then compute the probability of* q_{city} *results as, for instance for Nairobi:* $\Pr(t_1 \wedge t_2) = 0.5 \cdot 0.7 = 0.35$.

V. Implementation in ProvSQL

Theorem 10 and Corollary 13 pave the way for a practical implementation of provenance management and probabilistic query evaluation in a SQL DBMS. Note that the data model follows as closely as possible the practical behavior of SQL engines, instead of the theoretical model of annotated relations used in most of the literature: multiset semantics, explicit duplicate elimination using `DISTINCT` or `GROUP BY`, ordered attributes within relations.

We now explain how ProvSQL is implemented. ProvSQL uses PostgreSQL's extension mechanism to change the behavior of the DBMS. This extension mechanism allows to provide user-defined functions (UDFs), which can be written either in PL/pgSQL, the application language of PostgreSQL, or in C, to implement provenance- and probability-related functionalities in C. The extension mechanism also allows to run *hooks* through a C interface at different key phases of query evaluation. ProvSQL uses a combination of SQL, PL/pgSQL, C, and C++ code interfacing with the C code, to implement its behavior.

A. Annotated relations

As in Section IV-A, ProvSQL adds to every relation for which provenance needs to be tracked (which can be specified with a UDF) an extra *provsql* attribute to store the annotation. Instead of choosing a specific algebraic structure for the annotation ahead of time, this annotation is a generic *universally unique identifier (UUID)* [41]. We explain further on how these generic annotations can be specialized to specific (m)-semirings. Annotations of *base tuples* are randomly generated using the UUIDv4 standard and can be interpreted as abstract tuple identifiers; the 128-bit address space makes the collision probability vanishingly small. Annotations are built from these base annotations by using the (m-)semiring operations; to store resulting annotations in the same attribute, they are also UUIDs, this time computed following UUIDv5 by computing a SHA-1 hash formed from a fixed namespace, and a normalized description of the operator and its UUID operands. This guarantees, in particular, that annotations are generated in a deterministic manner: the same query over the same data will result in the exact same result, annotations included.

Retaining opaque UUIDs is not enough: one also needs to be able to efficiently determine how they relate to each other. This is done by storing a *provenance circuit* (a compact DAG representation of provenance expressions, first introduced in [42]) where each UUIDv4 points to a leaf gate representing inputs of the circuit and each UUIDv5 to an inner gate labeled by the operator and with children the operand UUIDs. Similarly, data values that are results of aggregate queries are represented as gates in the same circuit (with UUIDv5 representation) encoding their semimodule construction.

Storage of this provenance circuit in the DBMS is a challenge:

(i) It is an append-only data structure; except in cases where one wants to perform some clean-up, there is never the need of removing gates from the circuit, and a gate never needs to be updated.

(ii) It is a data structure which is updated at the time a query is evaluated, following the query rewriting approach of Section IV-C.

(iii) The content of this circuit needs to be stored in a persistent manner: UUIDs become uninterpretable without the circuit.

(iv) The circuit can become very large, as it contains one gate per tuple, and one gate per operation performed in a query.

(v) Access to the circuit needs to be as fast as possible.

Initial implementations of ProvSQL stored this circuit as an extra table within the database, which solved (iii)–(iv). But PostgreSQL tables are not optimized for append-only operations (i) and, most importantly, (ii) caused concurrency control issues. We then experimented with shared-memory storage of the circuit, which solved most issues but did not scale (iv) and made persistence (iii) an issue in case of failure.

Our final implementation provides a satisfactory solution to (i)–(v): The circuit is stored outside of the database, in *append-only files* that are properly indexed and *memory-mapped* so that the operating system can keep most often used fragments in RAM buffers. To avoid concurrency issues, access to these files is managed by a *single PostgreSQL worker process*, communicating with other backends through inter-process communication (with *pipes*). Finally, each backend process has a *small local cache* of most recently accessed circuit information.

B. Query rewriting

The main way ProvSQL is keeping track of the provenance of data is by the query rewriting approach discussed in Section IV-C. To implement this within PostgreSQL, ProvSQL defines a custom hook at *query planning time* before the query is sent to PostgreSQL's planner. If the query involves annotated relations (i.e., tables with the special *provsql* attribute of UUID type), the query is analyzed to determine if it is part of the supported language (the extended relational algebra described in Section III, as represented in SQL). If not, an error message is displayed instead of performing inadequate provenance computation. If so, the query is rewritten as described in Section IV-C and then sent to the regular planner for continued processing and evaluation.

C. Specialization to specific (m-)semirings

Annotations computed by ProvSQL are abstract UUIDs, with description in the circuit on how these are computed from base tuple UUIDs. One important point is that representation can be specialized to any arbitrary (m-)semiring and semimodule by applying homomorphisms: ProvSQL essentially works in the *universal* semiring (the how-semiring of integer polynomials, see [4]) and in the *universal* m-semiring (the free m-semiring, see [5]), from which there exist unique homomorphisms to any application (m-)semiring. Each algebraic structure of interest

is compiled into a C++ class whose methods describe the different operations ($\oplus$, $\otimes$, possibly $\ominus$ and δ) of the algebraic structure. At a user's request, given a *provenance mapping* from base UUIDs to values in that algebraic structure (e.g., integers in the counting semiring), the relevant part of the circuit is retrieved from its memory mapped representation, and ProvSQL evaluates the circuit in a bottom-up manner to determine the actual semiring annotation of every tuple in the result of a query, using the methods provided by the class.

D. Probability evaluation

Probabilistic query evaluation in ProvSQL relies on Corollary 13: provenance is computed in the same way as for provenance tracking, using query rewriting, and when a marginal probability computation is requested, we specialize the provenance to $\mathcal{B}[X]$ (given a mapping of base UUIDs to some Boolean function, e.g., one distinct variable per UUIDs to obtain a tuple-independent database) and then compute the probability of the corresponding Boolean circuit. Note that the computation of Boolean provenance, as it is crucial for probabilistic query evaluation, is further optimized from the regular retrieval of subcircuits from the the memory-mapped circuit.

ProvSQL implements different ways of computing the probability of a Boolean circuit (e.g., naïve enumeration of possible worlds, Monte-Carlo sampling, or some other approximation techniques such as WeightMC [43]). We focus here on its default computation method, which works in three steps:

(1) We determine whether the Boolean circuit is read-once [44], i.e., if every variable occurs only once; if so, its probability can be computed in linear-time.

(2) Otherwise, we use a heuristic algorithm [14] to quickly obtain a tree-decomposition of small width ($\lesssim 10$) of the circuit, if possible; if so, we use an algorithm from [45] to extract a deterministic and decomposable circuit from the Boolean circuit, on which the probability is computed in linear time.

(3) Otherwise, we encode the circuit in conjunctive normal form using Tseitin's transformation [46] and use an external knowledge compiler (by default d4 [47] but we also support c2d [48] and DSHARP [49]) as a black box to compile it into a deterministic and decomposable circuit, on which the probability is computed in linear time (but this new circuit may be exponential in size).

Recall that probabilistic query evaluation is intractable (#P-hard): the potential exponential blow-up of the last step is unavoidable.

E. Other features

At the time of writing, ProvSQL also implements other features, which are outside of the scope of this paper: it supports provenance for non-terminal aggregate queries, following Section 4 of [6]; Shapley value computation and expected Shapley value computation as described in [50]; provenance of update operations [51] as proposed in [7]; expected value computation for `COUNT`, `MIN`, `MAX`, `SUM`, following the algorithms in [52]; and where-provenance [2], [22], which is a form of provenance not captured by semirings [53].

VI. Experimental Evaluation

VII. Conclusion

ProvSQL's data model, architecture, and generic design makes it suitable to compute a large range of provenance annotations and to provide a generic solution for query evaluation in probabilistic databases. ProvSQL supports a larger subset of SQL than other existing systems and is actively maintained. Its performance is suitable for multi-GB databases; overhead of provenance computation is often constant (with the notable exception of aggregate queries) and, despite the #P-hardness of the problem, it is often possible to compute probabilities of query outputs in acceptable time.

Many perspectives for improvement exist. First, augmenting the support of SQL is a necessity for general applicability; this includes relatively simple cases such as subqueries in `WHERE` clauses or `OUTER JOIN` queries, nested aggregations, and much more complex cases such as `WITH RECURSIVE` queries which requires changes to the query executor. Second, when evaluating safe queries over tuple-independent databases, the intensional approach to probabilistic query evaluation is suboptimal. It is unknown whether the safe plan algorithm [12] can be turned into an intensional approach, but some results in this direction [54] are a first step toward safe query plans.

Acknowledgements

AI-Generated Content Acknowledgement

We used generative AI tools, like ChatGPT, to draft an initial version of some automation scripts for benchmarks and figures; these scripts were subsequently reviewed and fully adjusted by hand. Some of the Lean formal proofs were also aided by the use of generative AI tools. We did not use any AI system to develop any of the algorithms, theorems, human-readable proofs, or any other technical content of the paper; we did not use any AI system for the implementation of ProvSQL itself.

References

[1] P. Buneman, S. Khanna, and W. C. Tan, "Data provenance: Some basic issues," in *Foundations of Software Technology and Theoretical Computer Science, 20th Conference, FST TCS 2000 New Delhi, India, December 13-15, 2000, Proceedings*, ser. Lecture Notes in Computer Science, S. Kapoor and S. Prasad, Eds., vol. 1974. Springer, 2000, pp. 87–93. [Online]. Available: https://doi.org/10.1007/3-540-44450-5_6

[2] ——, "Why and where: A characterization of data provenance," in *Database Theory - ICDT 2001, 8th International Conference, London, UK, January 4-6, 2001, Proceedings*, ser. Lecture Notes in Computer Science, J. V. den Bussche and V. Vianu, Eds., vol. 1973. Springer, 2001, pp. 316–330. [Online]. Available: https://doi.org/10.1007/3-540-44503-X_20

[3] P. Agrawal, O. Benjelloun, A. D. Sarma, C. Hayworth, S. U. Nabar, T. Sugihara, and J. Widom, "Trio: A system for data, uncertainty, and lineage," in *Proceedings of the 32nd International Conference on Very Large Data Bases, Seoul, Korea, September 12-15, 2006*, U. Dayal, K. Whang, D. B. Lomet, G. Alonso, G. M. Lohman, M. L. Kersten, S. K. Cha, and Y. Kim, Eds. ACM, 2006, pp. 1151–1154. [Online]. Available: http://dl.acm.org/citation.cfm?id=1164231

[4] T. J. Green, G. Karvounarakis, and V. Tannen, "Provenance semirings," in *Proceedings of the Twenty-Sixth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 11-13, 2007, Beijing, China*, L. Libkin, Ed. ACM, 2007, pp. 31–40. [Online]. Available: https://doi.org/10.1145/1265530.1265535

[5] F. Geerts and A. Poggi, "On database query languages for K-relations," *J. Appl. Log.*, vol. 8, no. 2, pp. 173–185, 2010. [Online]. Available: https://doi.org/10.1016/j.jal.2009.09.001

[6] Y. Amsterdamer, D. Deutch, and V. Tannen, "Provenance for aggregate queries," in *Proceedings of the 30th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2011, June 12-16, 2011, Athens, Greece*, M. Lenzerini and T. Schwentick, Eds. ACM, 2011, pp. 153–164. [Online]. Available: https://doi.org/10.1145/1989284.1989302

[7] P. Bourhis, D. Deutch, and Y. Moskovitch, "Equivalence-invariant algebraic provenance for hyperplane update queries," in *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*, D. Maier, R. Pottinger, A. Doan, W. Tan, A. Alawini, and H. Q. Ngo, Eds. ACM, 2020, pp. 415–429. [Online]. Available: https://doi.org/10.1145/3318464.3380578

[8] E. F. Codd, "Understanding relations (installment #7)," *FDT Bull. ACM SIGFIDET SIGMOD*, vol. 7, no. 3, pp. 23–28, 1975.

[9] T. Imielinski and W. L. Jr., "Incomplete information in relational databases," *J. ACM*, vol. 31, no. 4, pp. 761–791, 1984. [Online]. Available: https://doi.org/10.1145/1634.1886

[10] N. N. Dalvi and D. Suciu, "Efficient query evaluation on probabilistic databases," *VLDB J.*, vol. 16, no. 4, pp. 523–544, 2007. [Online]. Available: https://doi.org/10.1007/s00778-006-0004-3

[11] N. N. Dalvi, C. Ré, and D. Suciu, "Queries and materialized views on probabilistic databases," *J. Comput. Syst. Sci.*, vol. 77, no. 3, pp. 473–490, 2011. [Online]. Available: https://doi.org/10.1016/j.jcss.2010.04.006

[12] N. N. Dalvi and D. Suciu, "The dichotomy of probabilistic inference for unions of conjunctive queries," *J. ACM*, vol. 59, no. 6, pp. 30:1–30:87, 2012. [Online]. Available: https://doi.org/10.1145/2395116.2395119

[13] A. Amarilli, P. Bourhis, and P. Senellart, "Provenance circuits for trees and treelike instances," in *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II*, ser. Lecture Notes in Computer Science, M. M. Halldórsson, K. Iwama, N. Kobayashi, and B. Speckmann, Eds., vol. 9135. Springer, 2015, pp. 56–68. [Online]. Available: https://doi.org/10.1007/978-3-662-47666-6_5

[14] S. Maniu, P. Senellart, and S. Jog, "An experimental study of the treewidth of real-world graph data," in *22nd International Conference on Database Theory, ICDT 2019, March 26-28, 2019, Lisbon, Portugal*, ser. LIPIcs, P. Barceló and M. Calautti, Eds., vol. 127. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, pp. 12:1–12:18. [Online]. Available: https://doi.org/10.4230/LIPIcs.ICDT.2019.12

[15] M. Umano and S. Fukami, "Fuzzy relational algebra for possibility-distribution-fuzzy-relational model of fuzzy data," *Journal of Intelligent Information Systems*, vol. 3, pp. 7–27, 1994.

[16] A. Abelló and J. Cheney, "Eris: Measuring discord among multidimensional data sources," *The VLDB Journal*, vol. 33, pp. 399–423, 2024.

[17] T. J. Green, G. Karvounarakis, N. E. Taylor, O. Biton, Z. G. Ives, and V. Tannen, "ORCHESTRA: facilitating collaborative data sharing," in *Proceedings of the ACM SIGMOD International Conference on Management of Data, Beijing, China, June 12-14, 2007*, C. Y. Chan, B. C. Ooi, and A. Zhou, Eds. ACM, 2007, pp. 1131–1133. [Online]. Available: https://doi.org/10.1145/1247480.1247631

[18] S. Singh, C. Mayfield, S. Mittal, S. Prabhakar, S. E. Hambrusch, and R. Shah, "Orion 2.0: native support for uncertain data," in *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008*, J. T. Wang, Ed. ACM, 2008, pp. 1239–1242. [Online]. Available: https://doi.org/10.1145/1376616.1376744

[19] J. Huang, L. Antova, C. Koch, and D. Olteanu, "MayBMS: a probabilistic database management system," in *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2009, Providence, Rhode Island, USA, June 29 - July 2, 2009*, U. Çetintemel, S. B. Zdonik, D. Kossmann, and N. Tatbul, Eds. ACM, 2009, pp. 1071–1074. [Online]. Available: https://doi.org/10.1145/1559845.1559984

[20] B. Glavic and G. Alonso, "Perm: Processing provenance and data on the same data model through query rewriting," in *Proceedings of the 25th International Conference on Data Engineering, ICDE 2009, March 29 2009 - April 2 2009, Shanghai, China*, Y. E. Ioannidis, D. L. Lee, and R. T. Ng, Eds. IEEE Computer Society, 2009, pp. 174–185. [Online]. Available: https://doi.org/10.1109/ICDE.2009.15

[21] B. S. Arab, S. Feng, B. Glavic, S. Lee, X. Niu, and Q. Zeng, "GProM - A swiss army knife for your provenance needs," *IEEE Data Eng. Bull.*, vol. 41, no. 1, pp. 51–62, 2018. [Online]. Available: http://sites.computer.org/debull/A18mar/p51.pdf

[22] P. Senellart, L. Jachiet, S. Maniu, and Y. Ramusat, "ProvSQL: Provenance and probability management in PostgreSQL," *Proc. VLDB Endow.*, vol. 11, no. 12, pp. 2034–2037, 2018. [Online]. Available: http://www.vldb.org/pvldb/vol11/p2034-senellart.pdf

[23] D. Calvanese, D. Lanti, A. Ozaki, R. P. aloza, and G. Xiao, "Enriching ontology-based data access with provenance," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, S. Kraus, Ed. ijcai.org, 2019, pp. 1616–1623. [Online]. Available: https://doi.org/10.24963/ijcai.2019/224

[24] S. B. Davidson, D. Deutch, N. Frost, B. Kimelfeld, O. Koren, and M. Monet, "Shapgraph: An holistic view of explanations through provenance graphs and shapley values," in *SIGMOD '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022*, Z. G. Ives, A. Bonifati, and A. E. Abbadi, Eds. ACM, 2022, pp. 2373–2376. [Online]. Available: https://doi.org/10.1145/3514221.3520172

[25] M. Leybovich and O. Shmueli, "ML based lineage in databases," *CoRR*, vol. abs/2109.06339, 2021. [Online]. Available: https://arxiv.org/abs/2109.06339

[26] P. S. Pintor, R. L. C. Costa, and J. M. Moreira, "Provenance in spatial queries," in *IDEAS'22: International Database Engineered Applications Symposium, Budapest, Hungary, August 22 - 24, 2022*, B. C. Desai and P. Z. Revesz, Eds. ACM, 2022, pp. 55–62. [Online]. Available: https://doi.org/10.1145/3548785.3548802

[27] S. Borgwardt, S. Breuer, and A. Kovtunova, "Computing abox justifications for query answers via datalog rewriting," in *Proceedings of the 36th International Workshop on Description Logics (DL 2023) co-located with the 20th International Conference on Principles of Knowledge Representation and Reasoning and the 21st International Workshop on Non-Monotonic Reasoning (KR 2023 and NMR 2023)., Rhodes, Greece, September 2-4, 2023*, ser. CEUR Workshop Proceedings,

O. Kutz, C. Lutz, and A. Ozaki, Eds., vol. 3515. CEUR-WS.org, 2023. [Online]. Available: https://ceur-ws.org/Vol-3515/paper-8.pdf

[28] F. Yunus, P. Karmakar, P. Senellart, T. Abdessalem, and S. Bressan, "Using A probabilistic database in an image retrieval application," in *Proceedings 28th International Conference on Extending Database Technology, EDBT 2025, Barcelona, Spain, March 25-28, 2025*, A. Simitsis, B. Kemme, A. Queralt, O. Romero, and P. Jovanovic, Eds. OpenProceedings.org, 2025, pp. 1106–1109. [Online]. Available: https://doi.org/10.48786/edbt.2025.100

[29] J. Boulos, N. N. Dalvi, B. Mandhani, S. Mathur, C. Ré, and D. Suciu, "MYSTIQ: a system for finding more answers by using probabilities," in *Proceedings of the ACM SIGMOD International Conference on Management of Data, Baltimore, Maryland, USA, June 14-16, 2005*, F. Özcan, Ed. ACM, 2005, pp. 891–893. [Online]. Available: https://doi.org/10.1145/1066157.1066277

[30] L. Antova, T. Jansen, C. Koch, and D. Olteanu, "Fast and simple relational processing of uncertain data," in *Proceedings of the 24th International Conference on Data Engineering, ICDE 2008, April 7-12, 2008, Cancún, Mexico*, G. Alonso, J. A. Blakeley, and A. L. P. Chen, Eds. IEEE Computer Society, 2008, pp. 983–992. [Online]. Available: https://doi.org/10.1109/ICDE.2008.4497507

[31] D. Olteanu, J. Huang, and C. Koch, "SPROUT: lazy vs. eager query plans for tuple-independent probabilistic databases," in *Proceedings of the 25th International Conference on Data Engineering, ICDE 2009, March 29 2009 - April 2 2009, Shanghai, China*, Y. E. Ioannidis, D. L. Lee, and R. T. Ng, Eds. IEEE Computer Society, 2009, pp. 640–651. [Online]. Available: https://doi.org/10.1109/ICDE.2009.123

[32] M. O. Jewell, A. S. Keshavarz, D. T. Michaelides, H. Yang, and L. Moreau, "The PROV-JSON serialization," https://openprovenance.org/prov-json/, 2014, W3C member submission.

[33] D. Fierens, G. V. den Broeck, J. Renkens, D. S. Shterionov, B. Gutmann, I. Thon, G. Janssens, and L. D. Raedt, "Inference and learning in probabilistic logic programs using weighted boolean formulas," *Theory Pract. Log. Program.*, vol. 15, no. 3, pp. 358–401, 2015. [Online]. Available: https://doi.org/10.1017/S1471068414000076

[34] S. Abiteboul, R. Hull, and V. Vianu, *Foundations of Databases*. Addison-Wesley, 1995. [Online]. Available: http://webdam.inria.fr/Alice/

[35] U. Dayal, N. Goodman, and R. H. Katz, "An extended relational algebra with control over duplicate elimination," in *Proceedings of the ACM Symposium on Principles of Database Systems, March 29-31, 1982, Los Angeles, California, USA*, J. D. Ullman and A. V. Aho, Eds. ACM, 1982, pp. 117–123. [Online]. Available: https://doi.org/10.1145/588111.588132

[36] S. Grumbach and T. Milo, "An algebra for pomsets," *Inf. Comput.*, vol. 150, no. 2, pp. 268–306, 1999. [Online]. Available: https://doi.org/10.1006/inco.1998.2777

[37] L. Libkin, "Expressive power of SQL," *Theor. Comput. Sci.*, vol. 296, no. 3, pp. 379–404, 2003. [Online]. Available: https://doi.org/10.1016/S0304-3975(02)00736-3

[38] P. Senellart, "Provenance and probabilities in relational databases," *SIGMOD Rec.*, vol. 46, no. 4, pp. 5–15, 2017. [Online]. Available: https://doi.org/10.1145/3186549.3186551

[39] R. Fink, L. Han, and D. Olteanu, "Aggregation in probabilistic databases via knowledge compilation," *Proceedings of the VLDB Endowment*, vol. 5, no. 5, pp. 490–501, 2012.

[40] D. Suciu, D. Olteanu, C. Ré, and C. Koch, *Probabilistic Databases*, ser. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2011. [Online]. Available: https://doi.org/10.2200/S00362ED1V01Y201105DTM016

[41] K. R. Davis, B. G. Peabody, and P. J. Leach, "Universally unique identifiers (UUIDs)," *RFC*, vol. 9562, pp. 1–46, 2024. [Online]. Available: https://doi.org/10.17487/RFC9562

[42] D. Deutch, T. Milo, S. Roy, and V. Tannen, "Circuits for Datalog provenance," in *Proc. 17th International Conference on Database Theory (ICDT), Athens, Greece, March 24-28, 2014*, N. Schweikardt, V. Christophides, and V. Leroy, Eds. OpenProceedings.org, 2014, pp. 201–212. [Online]. Available: https://doi.org/10.5441/002/icdt.2014.22

[43] S. Chakraborty, D. J. Fremont, K. S. Meel, S. A. Seshia, and M. Y. Vardi, "Distribution-aware sampling and weighted model counting for SAT," in *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada*, C. E. Brodley and P. Stone, Eds. AAAI Press, 2014, pp. 1722–1730. [Online]. Available: https://doi.org/10.1609/aaai.v28i1.8990

[44] P. Sen, A. Deshpande, and L. Getoor, "Read-once functions and query evaluation in probabilistic databases," *Proc. VLDB Endow.*, vol. 3, no. 1, pp. 1068–1079, 2010. [Online]. Available: http://www.vldb.org/pvldb/vldb2010/pvldb_vol3/R95.pdf

[45] A. Amarilli, F. Capelli, M. Monet, and P. Senellart, "Connecting knowledge compilation classes and width parameters," *Theory Comput. Syst.*, vol. 64, no. 5, pp. 861–914, 2020. [Online]. Available: https://doi.org/10.1007/s00224-019-09930-2

[46] G. S. Tseitin, "On the complexity of derivation in propositional calculus," in *Automation of Reasoning: 2: Classical Papers on Computational Logic 1967–1970*, J. H. Siekmann and G. Wrightson, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1983, pp. 466–483. [Online]. Available: https://doi.org/10.1007/978-3-642-81955-1_28

[47] J. Lagniez and P. Marquis, "An improved decision-DNNF compiler," in *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, C. Sierra, Ed. ijcai.org, 2017, pp. 667–673. [Online]. Available: https://doi.org/10.24963/ijcai.2017/93

[48] A. Darwiche, "New advances in compiling CNF into decomposable negation normal form," in *Proceedings of the 16th Eureopean Conference on Artificial Intelligence, ECAI'2004, including Prestigious Applicants of Intelligent Systems, PAIS 2004, Valencia, Spain, August 22-27, 2004*, R. L. de Mántaras and L. Saitta, Eds. IOS Press, 2004, pp. 328–332.

[49] C. J. Muise, S. A. McIlraith, J. C. Beck, and E. I. Hsu, "DSHARP: Fast d-DNNF compilation with sharpSAT," in *Advances in Artificial Intelligence - 25th Canadian Conference on Artificial Intelligence, Canadian AI 2012, Toronto, ON, Canada, May 28-30, 2012. Proceedings*, ser. Lecture Notes in Computer Science, L. Kosseim and D. Inkpen, Eds., vol. 7310. Springer, 2012, pp. 356–361. [Online]. Available: https://doi.org/10.1007/978-3-642-30353-1_36

[50] P. Karmakar, M. Monet, P. Senellart, and S. Bressan, "Expected Shapley-like scores of Boolean functions: Complexity and applications to probabilistic databases," *Proc. ACM Manag. Data*, vol. 2, no. 2, p. 92, 2024. [Online]. Available: https://doi.org/10.1145/3651593

[51] A. A. Widiaatmaja, B. Djeffal, A. Dandekar, and P. Senellart, "Demonstration of ProvSQL Update Provenance through Temporal Databases," in *Proc. PW*, Berlin, Germany, 06 2025, demonstration.

[52] S. Abiteboul, T. H. Chan, E. Kharlamov, W. Nutt, and P. Senellart, "Capturing continuous data and answering aggregate queries in probabilistic XML," *ACM Trans. Database Syst.*, vol. 36, no. 4, pp. 25:1–25:45, 2011. [Online]. Available: https://doi.org/10.1145/2043652.2043658

[53] J. Cheney, L. Chiticariu, and W. C. Tan, "Provenance in databases: Why, how, and where," *Found. Trends Databases*, vol. 1, no. 4, pp. 379–474, 2009. [Online]. Available: https://doi.org/10.1561/1900000006

[54] M. Monet, "Solving a special case of the intensional vs extensional conjecture in probabilistic databases," in *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2020, Portland, OR, USA, June 14-19, 2020*, D. Suciu, Y. Tao, and Z. Wei, Eds. ACM, 2020, pp. 149–163. [Online]. Available: https://doi.org/10.1145/3375395.3387642