UNIVERSITÉ
PARIS SCIENCES
& LETTRES

Provenance of HAVING Queries in Semirings with Monus

Aryak Sen Pratik Karmakar Silviu Maniu Angelo Saadeh *Pierre Senellart*

NTU, Singapore, 12 August 2026

Outline

- Data Provenance
- The Semiring Framework and ProvSQL
- HAVING: The Missing Operator
- A Possible-World Semantics
- Properties of the Semantics
- Algorithms
- In ProvSQL: Implementation and Experiments
- Wrap-Up

UNIVERSITÉ PARIS SCIENCES & LETTRES

Data Provenance

A motivating pipeline

1. Run an **object detector** (YOLO) over a few hundred thousand images
2. Store one row per detected object, keeping the detector's **confidence score**
3. **Query** the result: *"find me the images with at least two refrigerators"*

A motivating pipeline

1. Run an **object detector** (YOLO) over a few hundred thousand images
2. Store one row per detected object, keeping the detector's **confidence score**
3. **Query** the result: *"find me the images with at least two refrigerators"*

dataset

id	img	obj	conf
1	i_1	fridge	0.9
2	i_1	fridge	0.5
3	i_1	fridge	0.4
4	i_1	oven	0.8
5	i_2	fridge	0.7
6	i_2	fridge	0.3

COCO 2017 [Lin et al., 2014] + YOLOv5 [Ultralytics, 2021], as in [Yunus et al., 2025]

The awkward question

The rows are **not facts**: each is a guess with a confidence. So *how confident should we be in each query answer*?

Beyond query evaluation

We often want **something more** than the answer to a query:

- **Where** does this answer come from?
- **Why** was it returned?
- **How** was it produced?
- **How many times** can it be derived?
- What is its **probability**?
- What **clearance** do I need to see it?
- What is the **cheapest** derivation?
- How does it change if data is **deleted**?

Beyond query evaluation

We often want **something more** than the answer to a query:

- **Where** does this answer come from?
- **Why** was it returned?
- **How** was it produced?
- **How many times** can it be derived?
- What is its **probability**?
- What **clearance** do I need to see it?
- What is the **cheapest** derivation?
- How does it change if data is **deleted**?

Provenance management **[Buneman et al., 2001]**

Alongside query evaluation, record **bookkeeping information** rich enough to answer all of the above – computed **once**, reused **many times**

Beyond query evaluation

We often want **something more** than the answer to a query:

- **Where** does this answer come from?
- **Why** was it returned?
- **How** was it produced?
- **How many times** can it be derived?
- What is its **probability**?
- What **clearance** do I need to see it?
- What is the **cheapest** derivation?
- How does it change if data is **deleted**?

Provenance management **[Buneman et al., 2001]**

Alongside query evaluation, record **bookkeeping information** rich enough to answer all of the above – computed **once**, reused **many times**

This talk: what happens when the query says `GROUP BY ... HAVING ...`

Data model: annotated relations

Every tuple carries one extra column, its **provenance annotation** – think of it, for now, as a tuple identifier

dataset

id	img	obj	conf
1	i_1	fridge	0.9
2	i_1	fridge	0.5
3	i_1	fridge	0.4
4	i_1	oven	0.8
5	i_2	fridge	0.7
6	i_2	fridge	0.3

Data model: annotated relations

Every tuple carries one extra column, its **provenance annotation** – think of it, for now, as a tuple identifier

dataset

id	img	obj	conf	prov
1	i_1	fridge	0.9	x_1
2	i_1	fridge	0.5	x_2
3	i_1	fridge	0.4	x_3
4	i_1	oven	0.8	y
5	i_2	fridge	0.7	z_1
6	i_2	fridge	0.3	z_2

Data model: annotated relations

Every tuple carries one extra column, its **provenance annotation** – think of it, for now, as a tuple identifier

dataset

id	img	obj	conf	prov
1	i_1	fridge	0.9	x_1
2	i_1	fridge	0.5	x_2
3	i_1	fridge	0.4	x_3
4	i_1	oven	0.8	y
5	i_2	fridge	0.7	z_1
6	i_2	fridge	0.3	z_2

- A **query** is an ordinary query: it neither reads nor writes annotations
- We give a **semantics** for the annotations of the **output**, as a function of those of the input
- Different **choices of annotation domain** answer different questions

Boolean provenance

[Imieliński and Lipski Jr., 1984; Senellart, 2017]

- Fix a finite set $X = \{x_1, \ldots, x_n\}$ of **Boolean events**
- Annotations are **Boolean functions** over X

Boolean provenance

[Imieliński and Lipski Jr., 1984; Senellart, 2017]

- Fix a finite set $X = \{x_1, \ldots, x_n\}$ of **Boolean events**
- Annotations are **Boolean functions** over X
- **Meaning:** a valuation $\nu : X \to \{\bot, \top\}$ is a **possible world** – the sub-database $\nu(D)$ obtained by dropping the tuples whose annotation is false

Boolean provenance

[Imieliński and Lipski Jr., 1984; Senellart, 2017]

- Fix a finite set $X = \{x_1, \ldots, x_n\}$ of **Boolean events**
- Annotations are **Boolean functions** over X
- **Meaning:** a valuation $\nu : X \to \{\bot, \top\}$ is a **possible world** – the sub-database $\nu(D)$ obtained by dropping the tuples whose annotation is false
- The provenance of an answer t is then the Boolean function

$$\nu \longmapsto \left[t \in q(\nu(D)) \right]$$

"for which sub-databases is t still an answer?"

Boolean provenance

[Imieliński and Lipski Jr., 1984; Senellart, 2017]

- Fix a finite set $X = \{x_1, \ldots, x_n\}$ of **Boolean events**
- Annotations are **Boolean functions** over X
- **Meaning:** a valuation $\nu : X \to \{\bot, \top\}$ is a **possible world** – the sub-database $\nu(D)$ obtained by dropping the tuples whose annotation is false
- The provenance of an answer t is then the Boolean function

$$\nu \longmapsto \left[t \in q(\nu(D)) \right]$$

"for which sub-databases is t still an answer?"

Annotating every input tuple with a **distinct variable** makes the possible worlds **all 2^n sub-databases**

Boolean provenance, on our example

Query: *which images contain a refrigerator?*

```
SELECT DISTINCT img FROM dataset WHERE obj = 'fridge'
```

id	img	obj	prov
1	i_1	fridge	x_1
2	i_1	fridge	x_2
3	i_1	fridge	x_3
4	i_1	oven	y
5	i_2	fridge	z_1
6	i_2	fridge	z_2

Boolean provenance, on our example

Query: *which images contain a refrigerator?*

```
SELECT DISTINCT img FROM dataset WHERE obj = 'fridge'
```

id	img	obj	prov
1	i_1	fridge	x_1
2	i_1	fridge	x_2
3	i_1	fridge	x_3
4	i_1	oven	y
5	i_2	fridge	z_1
6	i_2	fridge	z_2

img	prov
i_1	$x_1 \vee x_2 \vee x_3$
i_2	$z_1 \vee z_2$

Boolean provenance, on our example

Query: *which images contain a refrigerator?*

```
SELECT DISTINCT img FROM dataset WHERE obj = 'fridge'
```

id	img	obj	prov
1	i_1	fridge	x_1
2	i_1	fridge	x_2
3	i_1	fridge	x_3
4	i_1	oven	y
5	i_2	fridge	z_1
6	i_2	fridge	z_2

img	prov
i_1	$x_1 \vee x_2 \vee x_3$
i_2	$z_1 \vee z_2$

Selection and projection **keep** annotations; here the `DISTINCT` **merges** the occurrences of each image, so their annotations are $\vee$**-ed**

From provenance to probabilities

- Read the detector confidences as **independent** probabilities $\Pr(x)$ on the events – a **tuple-independent** database [Suciu et al., 2011]

From provenance to probabilities

- Read the detector confidences as **independent** probabilities $\Pr(x)$ on the events – a **tuple-independent** database [Suciu et al., 2011]
- Then, for every answer t,

$$\Pr\big(t \in q(D)\big) \;=\; \Pr\big(\text{provenance of } t\big)$$

the probability that its **Boolean provenance** evaluates to true

From provenance to probabilities

- Read the detector confidences as **independent** probabilities $\Pr(x)$ on the events – a **tuple-independent** database [Suciu et al., 2011]
- Then, for every answer t,

$$\Pr\big(t \in q(D)\big) \;=\; \Pr\big(\text{provenance of } t\big)$$

  the probability that its **Boolean provenance** evaluates to true
- On our example:

$$\Pr(x_1 \vee x_2 \vee x_3) = 1 - 0.1 \cdot 0.5 \cdot 0.6 = 0.97, \qquad \Pr(z_1 \vee z_2) = 0.79$$

From provenance to probabilities

- Read the detector confidences as **independent** probabilities $\Pr(x)$ on the events – a **tuple-independent** database [Suciu et al., 2011]
- Then, for every answer t,

$$\Pr\big(t \in q(D)\big) \;=\; \Pr\big(\text{provenance of } t\big)$$

  the probability that its **Boolean provenance** evaluates to true
- On our example:

$$\Pr(x_1 \vee x_2 \vee x_3) = 1 - 0.1 \cdot 0.5 \cdot 0.6 = 0.97, \qquad \Pr(z_1 \vee z_2) = 0.79$$

Provenance turns **probabilistic query evaluation** into **one query** + **one Boolean-function probability** computation – #**P-hard** in general

The Semiring Framework and ProvSQL

One framework, many kinds of provenance [Green et al., 2007]

A **commutative semiring** $(\mathbb{K}, \oplus, \otimes, \mathbb{0}, \mathbb{1})$:

- $\oplus$, $\otimes$ associative, commutative
- neutral elements $\mathbb{0}$, $\mathbb{1}$
- $\otimes$ **distributes** over $\oplus$
- $\mathbb{0}$ **annihilates**: $a \otimes \mathbb{0} = \mathbb{0}$

One framework, many kinds of provenance [Green et al., 2007]

A **commutative semiring** $(\mathbb{K}, \oplus, \otimes, \mathbb{0}, \mathbb{1})$:

- $\oplus$, $\otimes$ associative, commutative
- neutral elements $\mathbb{0}$, $\mathbb{1}$
- $\otimes$ **distributes** over $\oplus$
- $\mathbb{0}$ **annihilates**: $a \otimes \mathbb{0} = \mathbb{0}$

Relations are **multisets**: duplicates stay **distinct occurrences**. One rule per operator:

selection, projection, renaming, union	annotations **unchanged**
duplicate elimination ε	annotations of merged tuples $\oplus$**-ed**
product, join	annotations of combined tuples $\otimes$**-ed**

One framework, many kinds of provenance [Green et al., 2007]

A **commutative semiring** $(\mathbb{K}, \oplus, \otimes, \mathbb{0}, \mathbb{1})$:

- $\oplus$, $\otimes$ associative, commutative
- neutral elements $\mathbb{0}$, $\mathbb{1}$
- $\otimes$ **distributes** over $\oplus$
- $\mathbb{0}$ **annihilates**: $a \otimes \mathbb{0} = \mathbb{0}$

Relations are **multisets**: duplicates stay **distinct occurrences**. One rule per operator:

selection, projection, renaming, union	annotations **unchanged**
duplicate elimination ε	annotations of merged tuples $\oplus$**-ed**
product, join	annotations of combined tuples $\otimes$**-ed**

Cost: a **PTIME** overhead over plain query evaluation

The semiring determines the form of provenance

Semiring	$(\oplus, \otimes)$	What the annotation tells you
$\mathbb{B} = \{\bot, \top\}$	$(\vee, \wedge)$	does the answer survive in this sub-database?
$\mathbb{N}$	$(+, \times)$	how many derivations – multiset semantics
$\mathbb{B}[X]$, Boolean functions	$(\vee, \wedge)$	**Boolean provenance $\Rightarrow$ probabilities**
Security	$(\min, \max)$	minimum clearance level needed
Tropical $\mathbb{N} \cup \{\infty\}$	$(\min, +)$	cheapest derivation (think shortest path)
Why$[X]$	$(\cup, \Cup)$	which sets of tuples suffice
$\mathbb{N}[X]$, polynomials	$(+, \times)$	**universal**: how, with multiplicities

The semiring determines the form of provenance

Semiring	$(\oplus, \otimes)$	What the annotation tells you
$\mathbb{B} = \{\bot, \top\}$	$(\vee, \wedge)$	does the answer survive in this sub-database?
$\mathbb{N}$	$(+, \times)$	how many derivations – multiset semantics
$\mathbb{B}[X]$, Boolean functions	$(\vee, \wedge)$	**Boolean provenance $\Rightarrow$ probabilities**
Security	$(\min, \max)$	minimum clearance level needed
Tropical $\mathbb{N} \cup \{\infty\}$	$(\min, +)$	cheapest derivation (think shortest path)
Why[X]	$(\cup, \Cup)$	which sets of tuples suffice
$\mathbb{N}[X]$, polynomials	$(+, \times)$	**universal**: how, with multiplicities

Compile once, evaluate many [Green et al., 2007]

Semiring **homomorphisms commute** with provenance computation, and $\mathbb{N}[X]$ is **universal**: compute provenance **once** there, then push it to any other semiring by a homomorphism – **no re-execution** of the query

Negation: semirings with monus [Geerts and Poggi, 2010]

So far, positive relational algebra only: for `EXCEPT` we must **subtract** annotations.

- Define $a \leqslant b$ as $\exists c,\ b = a \oplus c$; if this is an **order**, the semiring is **naturally ordered**
- An **m-semiring** adds a $\ominus$ satisfying the **residuation law** $x \ominus y \leqslant z \iff x \leqslant y \oplus z$ – which **determines** $\ominus$ uniquely

Negation: semirings with monus [Geerts and Poggi, 2010]

So far, positive relational algebra only: for `EXCEPT` we must **subtract** annotations.

- Define $a \leqslant b$ as $\exists c,\ b = a \oplus c$; if this is an **order**, the semiring is **naturally ordered**
- An **m-semiring** adds a $\ominus$ satisfying the **residuation law** $x \ominus y \leqslant z \iff x \leqslant y \oplus z$ – which **determines** $\ominus$ uniquely
- Examples: $\mathbb{B}[X]$ with $\wedge\neg$; $\mathrm{Why}[X]$ with $\setminus$; $\mathbb{N}$ with truncated difference

Negation: semirings with monus [Geerts and Poggi, 2010]

So far, positive relational algebra only: for `EXCEPT` we must **subtract** annotations.

- Define $a \leqslant b$ as $\exists c,\ b = a \oplus c$; if this is an **order**, the semiring is **naturally ordered**
- An **m-semiring** adds a $\ominus$ satisfying the **residuation law** $x \ominus y \leqslant z \iff x \leqslant y \oplus z$ – which **determines** $\ominus$ uniquely
- Examples: $\mathbb{B}[X]$ with $\wedge\neg$; $\mathrm{Why}[X]$ with $\setminus$; $\mathbb{N}$ with truncated difference
- **Almost all** provenance semirings admit a $\ominus$ – but **not all** [Amarilli and Monet, 2016]

Negation: semirings with monus [Geerts and Poggi, 2010]

So far, positive relational algebra only: for `EXCEPT` we must **subtract** annotations.

- Define $a \leqslant b$ as $\exists c,\ b = a \oplus c$; if this is an **order**, the semiring is **naturally ordered**
- An **m-semiring** adds a $\ominus$ satisfying the **residuation law** $x \ominus y \leqslant z \iff x \leqslant y \oplus z$ – which **determines** $\ominus$ uniquely
- Examples: $\mathbb{B}[X]$ with $\wedge\neg$; $\mathrm{Why}[X]$ with $\setminus$; $\mathbb{N}$ with truncated difference
- **Almost all** provenance semirings admit a $\ominus$ – but **not all** [Amarilli and Monet, 2016]

m-semirings support the **full relational algebra**, still in **PTIME** – they underlie Codd's theorem over semirings [Badia et al., 2025] and `EXCEPT` in ProvSQL [Sen et al., 2026]

Not every expected property holds

An m-semiring may or may not satisfy:

Idempotence	$a \oplus a = a$
Absorptivity	$\mathbb{1} \oplus a = \mathbb{1}$
Distributivity of $\otimes$ over $\ominus$	$a \otimes (b \ominus c) = (a \otimes b) \ominus (a \otimes c)$
Characteristic 0	$\bigoplus_{i=1}^{n} \mathbb{1} \neq \mathbb{0}$

Not every expected property holds

An m-semiring may or may not satisfy:

Idempotence	$a \oplus a = a$
Absorptivity	$\mathbb{1} \oplus a = \mathbb{1}$
Distributivity of $\otimes$ over $\ominus$	$a \otimes (b \ominus c) = (a \otimes b) \ominus (a \otimes c)$
Characteristic 0	$\bigoplus_{i=1}^{n} \mathbb{1} \neq \mathbb{0}$

- These are **exactly** the properties our results will turn on – **and the literature got some of them wrong**

Not every expected property holds

An m-semiring may or may not satisfy:

Idempotence	$a \oplus a = a$
Absorptivity	$\mathbb{1} \oplus a = \mathbb{1}$
Distributivity of $\otimes$ over $\ominus$	$a \otimes (b \ominus c) = (a \otimes b) \ominus (a \otimes c)$
Characteristic 0	$\bigoplus_{i=1}^{n} \mathbb{1} \neq \mathbb{0}$

- These are **exactly** the properties our results will turn on – **and the literature got some of them wrong**
- [Geerts and Poggi, 2010] claimed the tropical semiring over $\mathbb{R}$ admits no $\ominus$ (it does); [Amsterdamer et al., 2011a] claimed $\mathbb{N}[X]$ and Why$[X]$ satisfy $\otimes$-over-$\ominus$ distributivity (they do not)

Not every expected property holds

An m-semiring may or may not satisfy:

Idempotence	$a \oplus a = a$
Absorptivity	$\mathbb{1} \oplus a = \mathbb{1}$
Distributivity of $\otimes$ over $\ominus$	$a \otimes (b \ominus c) = (a \otimes b) \ominus (a \otimes c)$
Characteristic 0	$\bigoplus_{i=1}^{n} \mathbb{1} \neq \mathbb{0}$

- These are **exactly** the properties our results will turn on – **and the literature got some of them wrong**
- [Geerts and Poggi, 2010] claimed the tropical semiring over $\mathbb{R}$ admits no $\ominus$ (it does); [Amsterdamer et al., 2011a] claimed $\mathbb{N}[X]$ and Why$[X]$ satisfy $\otimes$-over-$\ominus$ distributivity (they do not)
- Easy mistakes to make – so we **machine-checked the whole table in Lean**

Common commutative m-semirings, formally verified

M-semiring	$a \ominus b$	Idempot.	Absorpt.	$\otimes$ / $\ominus$	Char. 0
$\mathbb{B}$	$a \wedge \neg b$	✓	✓	✓	✓
$\mathbb{B}[X]$	$\nu \mapsto a(\nu) \wedge \neg b(\nu)$	✓	✓	✓	✓
Why$[X]$	$a \setminus b$	✓	✗	✗	✓
Which$[X]$	$\begin{cases} \bot & \text{if } a = \bot \text{ or } a \subseteq b \\ a & \text{if } a \neq \bot \text{ and } b = \bot \\ a \setminus b & \text{otherwise} \end{cases}$	✓	✗	✗	✓
Temporal (interval unions)	$a \setminus b$	✓	✓	✓	✓
Tropical over $\mathbb{N}$	$a \geqslant b \ ? \ \infty : a$	✓	✓	✓	✓
Tropical over $\mathbb{R}$	$a \geqslant b \ ? \ \infty : a$	✓	✗	✓	✓
Viterbi	$a \leqslant b \ ? \ 0 : a$	✓	✓	✓	✓
Security	$a \geqslant b \ ? \ \top : a$	✓	✓	✗	✓
$\mathbb{N}$	$\max(a - b, 0)$	✗	✗	✓	✓
$\mathbb{N}[X]$	coefficient-wise truncated $-$	✗	✗	✗	✓
Łukasiewicz	$a \leqslant b \ ? \ 0 : a$	✓	✓	✓	✓

Common commutative m-semirings, formally verified

M-semiring	$a \ominus b$	Idempot.	Absorpt.	$\otimes$ / $\ominus$	Char. 0
$\mathbb{B}$	$a \wedge \neg b$	✓	✓	✓	✓
$\mathbb{B}[X]$	$\nu \mapsto a(\nu) \wedge \neg b(\nu)$	✓	✓	✓	✓
Why[X]	$a \setminus b$	✓	✗	✗	✓
Which[X]	$\begin{cases} \bot & \text{if } a = \bot \text{ or } a \subseteq b \\ a & \text{if } a \neq \bot \text{ and } b = \bot \\ a \setminus b & \text{otherwise} \end{cases}$	✓	✗	✗	✓
Temporal (interval unions)	$a \setminus b$	✓	✓	✓	✓
Tropical over $\mathbb{N}$	$a \geqslant b \ ? \ \infty : a$	✓	✓	✓	✓
Tropical over $\mathbb{R}$	$a \geqslant b \ ? \ \infty : a$	✓	✗	✓	✓
Viterbi	$a \leqslant b \ ? \ 0 : a$	✓	✓	✓	✓
Security	$a \geqslant b \ ? \ \top : a$	✓	✓	✗	✓
$\mathbb{N}$	$\max(a - b, 0)$	✗	✗	✓	✓
$\mathbb{N}[X]$	coefficient-wise truncated $-$	✗	✗	✗	✓
Łukasiewicz	$a \leqslant b \ ? \ 0 : a$	✓	✓	✓	✓

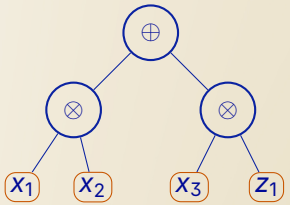
Highlighted: absorptive *and* $\otimes$ distributive over $\ominus$ – remember these

ProvSQL: our provenance layer inside PostgreSQL

- A **lightweight extension** of PostgreSQL [Sen et al., 2026]
- Queries over provenance-tracked tables are **transparently rewritten** so as to also build annotations
- Annotations are not formulas but gates of a shared **provenance circuit**: a DAG of $\oplus$/ $\otimes$ /$\ominus$ over the input tuples

ProvSQL: our provenance layer inside PostgreSQL

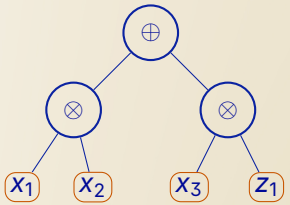
- A **lightweight extension** of PostgreSQL [Sen et al., 2026]
- Queries over provenance-tracked tables are **transparently rewritten** so as to also build annotations
- Annotations are not formulas but gates of a shared **provenance circuit**: a DAG of $\oplus$/ $\otimes$ /$\ominus$ over the input tuples
- One circuit, **many uses**: `sr_formula`, `probability_evaluate`, `shapley`...



a provenance circuit, shared by all answers

ProvSQL: our provenance layer inside PostgreSQL

- A **lightweight extension** of PostgreSQL [Sen et al., 2026]
- Queries over provenance-tracked tables are **transparently rewritten** so as to also build annotations
- Annotations are not formulas but gates of a shared **provenance circuit**: a DAG of $\oplus$/ $\otimes$ /$\ominus$ over the input tuples
- One circuit, **many uses**: `sr_formula`, `probability_evaluate`, `shapley`...



a provenance circuit, shared by all answers

Open source and runnable in your browser: `https://provsql.org/`

HAVING: The Missing Operator

A `HAVING` query over uncertain data

*Which images contain **at least two** refrigerators?*

```
SELECT img FROM dataset WHERE obj = 'fridge'
GROUP BY img HAVING COUNT(*) >= 2
```

dataset

img	obj	conf	prov
i_1	fridge	0.9	x_1
i_1	fridge	0.5	x_2
i_1	fridge	0.4	x_3
i_1	oven	0.8	y
i_2	fridge	0.7	z_1
i_2	fridge	0.3	z_2

A `HAVING` query over uncertain data

*Which images contain **at least two** refrigerators?*

```
SELECT img FROM dataset WHERE obj = 'fridge'
GROUP BY img HAVING COUNT(*) >= 2
```

dataset

img	obj	conf	prov
i_1	fridge	0.9	x_1
i_1	fridge	0.5	x_2
i_1	fridge	0.4	x_3
i_1	oven	0.8	y
i_2	fridge	0.7	z_1
i_2	fridge	0.3	z_2

Plain SQL answers $\{i_1, i_2\}$. But what should the **output relation** look like?

img	prov	Pr
i_1	$(x_1 \wedge x_2) \vee (x_1 \wedge x_3) \vee (x_2 \wedge x_3)$	0.65
i_2	$z_1 \wedge z_2$	0.21

i_1's annotation is exactly *"at least two of x_1, x_2, x_3 hold"*

A `HAVING` query over uncertain data

*Which images contain **at least two** refrigerators?*

```
SELECT img FROM dataset WHERE obj = 'fridge'
GROUP BY img HAVING COUNT(*) >= 2
```

dataset

img	obj	conf	prov
i_1	fridge	0.9	x_1
i_1	fridge	0.5	x_2
i_1	fridge	0.4	x_3
i_1	oven	0.8	y
i_2	fridge	0.7	z_1
i_2	fridge	0.3	z_2

Plain SQL answers $\{i_1, i_2\}$. But what should the **output relation** look like?

img	prov	Pr
i_1	$(x_1 \wedge x_2) \vee (x_1 \wedge x_3) \vee (x_2 \wedge x_3)$	0.65
i_2	$z_1 \wedge z_2$	0.21

i_1's annotation is exactly *"at least two of x_1, x_2, x_3 hold"*

Boolean provenance suffices here – we want it in **any** m-semiring.

Aggregation in the semiring framework [Amsterdamer et al., 2011b]

- Aggregate **values** become elements of a **semimodule**: formal sums $\bigoplus_i v_i \otimes \alpha_i$ of a data value and an annotation
- A unary δ operator records the **existence** of a group

Aggregation in the semiring framework [Amsterdamer et al., 2011b]

- Aggregate **values** become elements of a **semimodule**: formal sums $\bigoplus_i v_i \otimes \alpha_i$ of a data value and an annotation
- A unary δ operator records the **existence** of a group
- And, for `HAVING`, a brand-new **comparison operator** turning an inequality between two semimodule elements into an element of $\mathbb{K}$:

$$\left[\texttt{COUNT}(*) \geqslant 2\right] \in \mathbb{K} \,?$$

Aggregation in the semiring framework [Amsterdamer et al., 2011b]

- Aggregate **values** become elements of a **semimodule**: formal sums $\bigoplus_i v_i \otimes \alpha_i$ of a data value and an annotation
- A unary δ operator records the **existence** of a group
- And, for `HAVING`, a brand-new **comparison operator** turning an inequality between two semimodule elements into an element of $\mathbb{K}$:

$$\left[\texttt{COUNT}(*) \geqslant 2\right] \in \mathbb{K} \,?$$

Elegant on paper, but...

- the comparison operator needs an **ad-hoc definition for every pair** (semiring, aggregation monoid);
- aggregates must be **associative and commutative** – no `AVG`, no `ARRAY_AGG`.

Consequences for provenance systems

- **Our own ProvSQL** [Senellart et al., 2018; Sen et al., 2026], until now: only **terminal aggregation** – aggregate last, never compare

Consequences for provenance systems

- **Our own ProvSQL** [Senellart et al., 2018; Sen et al., 2026], until now: only **terminal aggregation** – aggregate last, never compare
- **GProM** [Arab et al., 2018]: `HAVING` treated as a **downstream selection**; it does not record how the provenance depends on whether the comparison holds – no probabilistic engine

Consequences for provenance systems

- **Our own ProvSQL** [Senellart et al., 2018; Sen et al., 2026], until now: only **terminal aggregation** – aggregate last, never compare
- **GProM** [Arab et al., 2018]: `HAVING` treated as a **downstream selection**; it does not record how the provenance depends on whether the comparison holds – no probabilistic engine
- [Pintor et al., 2025]: builds the **abstract formulas** with the comparison operator, as strings, **never evaluated** in a concrete semiring

Consequences for provenance systems

- **Our own ProvSQL** [Senellart et al., 2018; Sen et al., 2026], until now: only **terminal aggregation** – aggregate last, never compare
- **GProM** [Arab et al., 2018]: `HAVING` treated as a **downstream selection**; it does not record how the provenance depends on whether the comparison holds – no probabilistic engine
- [Pintor et al., 2025]: builds the **abstract formulas** with the comparison operator, as strings, **never evaluated** in a concrete semiring
- In [Yunus et al., 2025] we needed exactly this query – and had to **hand-rewrite it as a chain of self-joins**

Consequences for provenance systems

- **Our own ProvSQL** [Senellart et al., 2018; Sen et al., 2026], until now: only **terminal aggregation** – aggregate last, never compare
- **GProM** [Arab et al., 2018]: `HAVING` treated as a **downstream selection**; it does not record how the provenance depends on whether the comparison holds – no probabilistic engine
- [Pintor et al., 2025]: builds the **abstract formulas** with the comparison operator, as strings, **never evaluated** in a concrete semiring
- In [Yunus et al., 2025] we needed exactly this query – and had to **hand-rewrite it as a chain of self-joins**

This work

A semantics for `HAVING` in **any commutative m-semiring**, with **no new operator**, for **any** aggregate function – plus algorithms, an implementation in ProvSQL, and Lean proofs

UNIVERSITÉ PARIS SCIENCES & LETTRES

A Possible-World Semantics

The idea

- A `HAVING` condition is a statement about a **group**

The idea

- A `HAVING` condition is a statement about a **group**
- In a sub-database, only **some** members of the group survive – and whether `COUNT(*) >= 2` holds depends on **which**

The idea

- A `HAVING` condition is a statement about a **group**
- In a sub-database, only **some** members of the group survive – and whether `COUNT(*) >= 2` holds depends on **which**
- So: **enumerate the possible worlds of the group**, keep those where the condition holds, and $\oplus$ their annotations

The idea

- A `HAVING` condition is a statement about a **group**
- In a sub-database, only **some** members of the group survive – and whether `COUNT(*) >= 2` holds depends on **which**
- So: **enumerate the possible worlds of the group**, keep those where the condition holds, and $\oplus$ their annotations

Back to **possible-world semantics** – the very idea behind Boolean provenance – but applied **locally, inside one group**, and expressed with $\oplus$, $\otimes$, $\ominus$ **only**

The idea

- A `HAVING` condition is a statement about a **group**
- In a sub-database, only **some** members of the group survive – and whether `COUNT(*) >= 2` holds depends on **which**
- So: **enumerate the possible worlds of the group**, keep those where the condition holds, and $\oplus$ their annotations

Back to **possible-world semantics** – the very idea behind Boolean provenance – but applied **locally, inside one group**, and expressed with $\oplus$, $\otimes$, $\ominus$ **only**

No comparison operator. No semimodule. No monoid.

The semantics

Let U be the sequence of **group members**, with their annotations. A **possible world** $W \sqsubseteq U$ is a subsequence; its **annotation** is

$$\mathrm{ann}_U(W) := \underbrace{\left(\bigotimes_{(u,\alpha)\in W} \alpha \right)}_{\text{kept}} \otimes \underbrace{\left(\mathbb{1} \ominus \bigoplus_{(u,\alpha)\in U\setminus W} \alpha \right)}_{\text{discarded}}$$

The semantics

Let U be the sequence of **group members**, with their annotations. A **possible world** $W \sqsubseteq U$ is a subsequence; its **annotation** is

$$\mathrm{ann}_U(W) := \underbrace{\Big(\bigotimes_{(u,\alpha)\in W} \alpha\Big)}_{\text{kept}} \otimes \underbrace{\Big(\mathbb{1} \ominus \bigoplus_{(u,\alpha)\in U\setminus W} \alpha\Big)}_{\text{discarded}}$$

and its **aggregate value** is $\mathrm{agg}_{t,f}(W) = f\big((t(u))_{(u,\alpha)\in W}\big)$.

The semantics

Let U be the sequence of **group members**, with their annotations. A **possible world** $W \sqsubseteq U$ is a subsequence; its **annotation** is

$$\mathrm{ann}_U(W) := \underbrace{\Big(\bigotimes_{(u,\alpha)\in W} \alpha\Big)}_{\text{kept}} \otimes \underbrace{\Big(\mathbb{1} \ominus \bigoplus_{(u,\alpha)\in U\setminus W} \alpha\Big)}_{\text{discarded}}$$

and its **aggregate value** is $\mathrm{agg}_{t,f}(W) = f\big((t(u))_{(u,\alpha)\in W}\big)$. The provenance of the group under the condition $\psi = \big(f(t) \star C\big)$ is then

$$[\![\psi]\!] := \bigoplus_{\emptyset \neq W \sqsubseteq U} \mathrm{ann}_U(W) \otimes \chi_\star\big(\mathrm{agg}_{t,f}(W),\, C\big)$$

The semantics

Let U be the sequence of **group members**, with their annotations. A **possible world** $W \sqsubseteq U$ is a subsequence; its **annotation** is

$$\mathrm{ann}_U(W) := \underbrace{\Big(\bigotimes_{(u,\alpha)\in W} \alpha\Big)}_{\text{kept}} \otimes \underbrace{\Big(\mathbb{1} \ominus \bigoplus_{(u,\alpha)\in U\setminus W} \alpha\Big)}_{\text{discarded}}$$

and its **aggregate value** is $\mathrm{agg}_{t,f}(W) = f\big((t(u))_{(u,\alpha)\in W}\big)$. The provenance of the group under the condition $\psi = (f(t) \star C)$ is then

$$[\![\psi]\!] := \bigoplus_{\emptyset \neq W \sqsubseteq U} \mathrm{ann}_U(W) \otimes \chi_\star\big(\mathrm{agg}_{t,f}(W),\, C\big)$$

with $\chi_\star(a,b) = \mathbb{1}$ if $a \star b$ holds and $\mathbb{0}$ otherwise – and only **non-empty** worlds, which already says "the group exists"

Worked example: `COUNT(*) >= 2`

Group i_1, three members annotated x_1, x_2, x_3 in $\mathbb{B}[X]$; write a world as the set of members it **keeps**. The valid non-empty worlds are the three pairs and the triple:

$$\mathrm{ann}_U(\{x_1, x_2\}) = x_1 \wedge x_2 \wedge \neg x_3, \quad \mathrm{ann}_U(\{x_1, x_3\}) = x_1 \wedge x_3 \wedge \neg x_2,$$
$$\mathrm{ann}_U(\{x_2, x_3\}) = x_2 \wedge x_3 \wedge \neg x_1, \quad \mathrm{ann}_U(\{x_1, x_2, x_3\}) = x_1 \wedge x_2 \wedge x_3.$$

Worked example: `COUNT(*) >= 2`

Group i_1, three members annotated x_1, x_2, x_3 in $\mathbb{B}[X]$; write a world as the set of members it **keeps**. The valid non-empty worlds are the three pairs and the triple:

$$\mathrm{ann}_U(\{x_1, x_2\}) = x_1 \wedge x_2 \wedge \neg x_3, \quad \mathrm{ann}_U(\{x_1, x_3\}) = x_1 \wedge x_3 \wedge \neg x_2,$$
$$\mathrm{ann}_U(\{x_2, x_3\}) = x_2 \wedge x_3 \wedge \neg x_1, \quad \mathrm{ann}_U(\{x_1, x_2, x_3\}) = x_1 \wedge x_2 \wedge x_3.$$

Their $\oplus$-sum:

$$(x_1 \wedge x_2) \vee (x_1 \wedge x_3) \vee (x_2 \wedge x_3)$$

Worked example: `COUNT(*) >= 2`

Group i_1, three members annotated x_1, x_2, x_3 in $\mathbb{B}[X]$; write a world as the set of members it **keeps**. The valid non-empty worlds are the three pairs and the triple:

$$\mathrm{ann}_U(\{x_1, x_2\}) = x_1 \wedge x_2 \wedge \neg x_3, \quad \mathrm{ann}_U(\{x_1, x_3\}) = x_1 \wedge x_3 \wedge \neg x_2,$$
$$\mathrm{ann}_U(\{x_2, x_3\}) = x_2 \wedge x_3 \wedge \neg x_1, \quad \mathrm{ann}_U(\{x_1, x_2, x_3\}) = x_1 \wedge x_2 \wedge x_3.$$

Their $\oplus$-sum:

$$(x_1 \wedge x_2) \vee (x_1 \wedge x_3) \vee (x_2 \wedge x_3)$$

Exactly "at least two of x_1, x_2, x_3" – and the **negative factors cancelled**, because $\mathbb{B}[X]$ is **absorptive**: the **minimal** valid worlds already carry the whole provenance

Worked example: `COUNT(*) >= 2`

Group i_1, three members annotated x_1, x_2, x_3 in $\mathbb{B}[X]$; write a world as the set of members it **keeps**. The valid non-empty worlds are the three pairs and the triple:

$$\mathrm{ann}_U(\{x_1, x_2\}) = x_1 \wedge x_2 \wedge \neg x_3, \quad \mathrm{ann}_U(\{x_1, x_3\}) = x_1 \wedge x_3 \wedge \neg x_2,$$
$$\mathrm{ann}_U(\{x_2, x_3\}) = x_2 \wedge x_3 \wedge \neg x_1, \quad \mathrm{ann}_U(\{x_1, x_2, x_3\}) = x_1 \wedge x_2 \wedge x_3.$$

Their $\oplus$-sum:

$$(x_1 \wedge x_2) \vee (x_1 \wedge x_3) \vee (x_2 \wedge x_3)$$

Exactly "at least two of x_1, x_2, x_3" – and the **negative factors cancelled**, because $\mathbb{B}[X]$ is **absorptive**: the **minimal** valid worlds already carry the whole provenance

In a non-absorptive m-semiring they do not cancel, and the semantics really uses $\ominus$

Arbitrary aggregate functions

- An aggregate function is just $f : \mathrm{Seq}(\mathcal{V}) \to \mathcal{V}$ – a function on **sequences** of values

Arbitrary aggregate functions

- An aggregate function is just $f : \mathrm{Seq}(\mathcal{V}) \to \mathcal{V}$ – a function on **sequences** of values
- No monoid, no associativity, no commutativity required:
  - `SUM`, `COUNT`, `MIN`, `MAX`: commutative, the order is irrelevant
  - `AVG`: not associative, but perfectly fine here
  - `ARRAY_AGG`, `LISTAGG`, `PICKFIRST`: **order-sensitive**

Arbitrary aggregate functions

- An aggregate function is just $f : \mathrm{Seq}(\mathcal{V}) \to \mathcal{V}$ – a function on **sequences** of values
- No monoid, no associativity, no commutativity required:
  - `SUM`, `COUNT`, `MIN`, `MAX`: commutative, the order is irrelevant
  - `AVG`: not associative, but perfectly fine here
  - `ARRAY_AGG`, `LISTAGG`, `PICKFIRST`: **order-sensitive**
- For order-sensitive aggregates we carry the order along, as SQL already does: `AGG(...) WITHIN GROUP (ORDER BY ...)`

Arbitrary aggregate functions

- An aggregate function is just $f : \mathrm{Seq}(\mathcal{V}) \to \mathcal{V}$ – a function on **sequences** of values
- No monoid, no associativity, no commutativity required:
  - `SUM`, `COUNT`, `MIN`, `MAX`: commutative, the order is irrelevant
  - `AVG`: not associative, but perfectly fine here
  - `ARRAY_AGG`, `LISTAGG`, `PICKFIRST`: **order-sensitive**
- For order-sensitive aggregates we carry the order along, as SQL already does: `AGG(...) WITHIN GROUP (ORDER BY ...)`

Limitation

Grouping and ordering are only meaningful on **regular** values: queries that **group or order by an aggregate value** are rejected. Order and uncertainty interact in complicated ways.

Comparisons further downstream

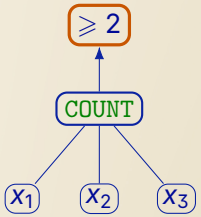
- An aggregate value may be **carried downstream** – through projections, joins, further selections – before being compared

Comparisons further downstream

- An aggregate value may be **carried downstream** – through projections, joins, further selections – before being compared
- ProvSQL materializes it as an **aggregate token**: a gate labelled by the aggregate function, whose **inputs are the group's occurrences** – nothing more to store

Comparisons further downstream

- An aggregate value may be **carried downstream** – through projections, joins, further selections – before being compared
- ProvSQL materializes it as an **aggregate token**: a gate labelled by the aggregate function, whose **inputs are the group's occurrences** – nothing more to store
- The comparison becomes a **comparison gate**, evaluated **lazily** at semiring-evaluation time – over the possible worlds of the **originating group**



one **comparison gate**, *N* inputs

UNIVERSITÉ
PARIS
SCIENCES
& LETTRES

Properties of the Semantics

A consistency check: rewriting `HAVING` away

Any SQL programmer can express the `HAVING` query $Q_1 =$ `COUNT(*) >= C` without `HAVING`, by joining C **copies** of the table – giving Q_2:

```
SELECT DISTINCT d1.img FROM dataset d1
  JOIN dataset d2 ON d2.img = d1.img AND d2.obj = 'fridge' AND d2.id > d1.id
  ...                                                  -- C-2 further self-joins
WHERE d1.obj = 'fridge'
```

A consistency check: rewriting `HAVING` away

Any SQL programmer can express the `HAVING` query $Q_1 =$ `COUNT(*) >= C` without `HAVING`, by joining C **copies** of the table – giving Q_2:

```
SELECT DISTINCT d1.img FROM dataset d1
  JOIN dataset d2 ON d2.img = d1.img AND d2.obj = 'fridge' AND d2.id > d1.id
  ...                                                  -- C-2 further self-joins
WHERE d1.obj = 'fridge'
```

And `COUNT(*) <= C` as "$\geqslant 1$" `EXCEPT` "$\geqslant C+1$" – non-monotone, hence the need for $\ominus$.

A consistency check: rewriting `HAVING` away

Any SQL programmer can express the `HAVING` query $Q_1 =$ `COUNT(*) >= C` without `HAVING`, by joining C **copies** of the table – giving Q_2:

```
SELECT DISTINCT d1.img FROM dataset d1
  JOIN dataset d2 ON d2.img = d1.img AND d2.obj = 'fridge' AND d2.id > d1.id
  ...                                                  -- C-2 further self-joins
WHERE d1.obj = 'fridge'
```

And `COUNT(*) <= C` as "$\geqslant 1$" `EXCEPT` "$\geqslant C+1$" – non-monotone, hence the need for $\ominus$.

The question

Q_2 needs no new semantics: it is aggregate-free, so the **standard m-semiring rules** already annotate it. Does our `HAVING` semantics give Q_1 the **same annotation**?

Agreement for well-behaved m-semirings

Theorem

For $\star \in \{\leqslant, =, \geqslant\}$ and any $C \geqslant 1$:

(i) In every commutative m-semiring that is **absorptive** and where $\otimes$ **distributes over** $\ominus$, Q_1 and Q_2 get the **same annotation** on every instance.

Agreement for well-behaved m-semirings

Theorem

For $\star \in \{\leqslant, =, \geqslant\}$ and any $C \geqslant 1$:

(i) In every commutative m-semiring that is **absorptive** and where $\otimes$ **distributes over** $\ominus$, Q_1 and Q_2 get the **same annotation** on every instance.

(ii) **Distributivity is needed:** a five-element chain, **absorptive** but not $\otimes$-over-$\ominus$ distributive, where they **differ**.

(iii) **Absorptivity is needed:** tropical over $\mathbb{Z}$, **distributive** but not absorptive, where they **differ**.

Agreement for well-behaved m-semirings

Theorem

For $\star \in \{\leqslant, =, \geqslant\}$ and any $C \geqslant 1$:

(i) In every commutative m-semiring that is **absorptive** and where $\otimes$ **distributes over** $\ominus$, Q_1 and Q_2 get the **same annotation** on every instance.

(ii) **Distributivity is needed:** a five-element chain, **absorptive** but not $\otimes$-over-$\ominus$ distributive, where they **differ**.

(iii) **Absorptivity is needed:** tropical over $\mathbb{Z}$, **distributive** but not absorptive, where they **differ**.

- $\mathbb{B}[X]$ satisfies both conditions $\Rightarrow$ **we are provably right where it matters** (Boolean provenance, probabilities)

Agreement for well-behaved m-semirings

Theorem
For $\star \in \{\leqslant, =, \geqslant\}$ and any $C \geqslant 1$:

(i) In every commutative m-semiring that is **absorptive** and where $\otimes$ **distributes over** $\ominus$, Q_1 and Q_2 get the **same annotation** on every instance.

(ii) **Distributivity is needed:** a five-element chain, **absorptive** but not $\otimes$-over-$\ominus$ distributive, where they **differ**.

(iii) **Absorptivity is needed:** tropical over $\mathbb{Z}$, **distributive** but not absorptive, where they **differ**.

- $\mathbb{B}[X]$ satisfies both conditions $\Rightarrow$ **we are provably right where it matters** (Boolean provenance, probabilities)
- Not a defect: outside Boolean provenance, semiring provenance depends on **how a query is written**, not only on what it computes

Both properties hold in common m-semirings

Absorptive $+$ $\otimes$ distr. over $\ominus$	**Not both**
$\mathbb{B}$, $\mathbb{B}[X]$	Why$[X]$, Which$[X]$
Temporal (unions of intervals)	Tropical over $\mathbb{R}$
Tropical over $\mathbb{N}$, Viterbi	Security
Łukasiewicz	$\mathbb{N}$, $\mathbb{N}[X]$

Both properties hold in common m-semirings

Absorptive $+$ $\otimes$ distr. over $\ominus$	**Not both**
$\mathbb{B}$, $\mathbb{B}[X]$	Why$[X]$, Which$[X]$
Temporal (unions of intervals)	Tropical over $\mathbb{R}$
Tropical over $\mathbb{N}$, Viterbi	Security
Łukasiewicz	$\mathbb{N}$, $\mathbb{N}[X]$

- The theorem is a **proper** generalization of the Boolean case: there is **no homomorphism** from $\mathbb{B}[X]$ to Tropical $\mathbb{N}$, Viterbi or Łukasiewicz sending variables to arbitrary values

Both properties hold in common m-semirings

Absorptive $+$ $\otimes$ distr. over $\ominus$	**Not both**
$\mathbb{B}$, $\mathbb{B}[X]$	Why$[X]$, Which$[X]$
Temporal (unions of intervals)	Tropical over $\mathbb{R}$
Tropical over $\mathbb{N}$, Viterbi	Security
Łukasiewicz	$\mathbb{N}$, $\mathbb{N}[X]$

- The theorem is a **proper** generalization of the Boolean case: there is **no homomorphism** from $\mathbb{B}[X]$ to Tropical $\mathbb{N}$, Viterbi or Łukasiewicz sending variables to arbitrary values
- For the semirings that do not qualify, we simply **adopt** our semantics as *the* implementable semantics of `HAVING`

Further properties of the semantics

Homomorphisms still commute

For every m-semiring homomorphism $h : \mathbb{K} \to \mathbb{K}'$, every `HAVING` query Q and every instance $\hat{I}$: $\langle\!\langle Q \rangle\!\rangle^{h(\hat{I})} = h(\langle\!\langle Q \rangle\!\rangle^{\hat{I}})$

$\Rightarrow$ **"compile once, evaluate many" still applies** – one circuit, any semiring

Further properties of the semantics

Homomorphisms still commute

For every m-semiring homomorphism $h : \mathbb{K} \to \mathbb{K}'$, every `HAVING` query Q and every instance $\hat{I}$: $\langle\!\langle Q \rangle\!\rangle^{h(\hat{I})} = h(\langle\!\langle Q \rangle\!\rangle^{\hat{I}})$

$\Rightarrow$ **"compile once, evaluate many" still applies** – one circuit, any semiring

Probabilistic query evaluation is correct

For a Boolean `HAVING` query over a tuple-independent instance, $\Pr(I \models Q) = \Pr(\langle\!\langle Q \rangle\!\rangle^{\hat{I}})$: the query probability **is** the probability of our annotation, with $\hat{I}$ annotating tuples by independent $\mathbb{B}[X]$ variables

Further properties of the semantics

Homomorphisms still commute

For every m-semiring homomorphism $h : \mathbb{K} \to \mathbb{K}'$, every `HAVING` query Q and every instance $\hat{I}$: $\langle\!\langle Q \rangle\!\rangle^{h(\hat{I})} = h(\langle\!\langle Q \rangle\!\rangle^{\hat{I}})$

$\Rightarrow$ **"compile once, evaluate many" still applies** – one circuit, any semiring

Probabilistic query evaluation is correct

For a Boolean `HAVING` query over a tuple-independent instance, $\Pr(I \models Q) = \Pr(\langle\!\langle Q \rangle\!\rangle^{\hat{I}})$: the query probability **is** the probability of our annotation, with $\hat{I}$ annotating tuples by independent $\mathbb{B}[X]$ variables

Unsurprisingly, evaluation is hard

For $\mathbb{K} \in \{\mathbb{B}[X], \mathbb{N}[X]\}$, deciding whether the provenance of a `HAVING` query is $\neq \mathbb{0}$ is **NP-complete** – and NP-hard already in **data** complexity, for a single fixed query

Machine-checked proofs

- Every statement that can be formalized comes with a **Lean 4 proof**, hyperlinked from the paper
- The exceptions are the explicit **running-time bounds** – out of scope without a formal cost model

Machine-checked proofs

- Every statement that can be formalized comes with a **Lean 4 proof**, hyperlinked from the paper
- The exceptions are the explicit **running-time bounds** – out of scope without a formal cost model
- This is not decoration: it **caught real errors** in the m-semiring literature that had stood for fifteen years

Machine-checked proofs

- Every statement that can be formalized comes with a **Lean 4 proof**, hyperlinked from the paper
- The exceptions are the explicit **running-time bounds** – out of scope without a formal cost model
- This is not decoration: it **caught real errors** in the m-semiring literature that had stood for fifteen years
- These building blocks have **counter-intuitive** properties; clean, verified statements are a contribution in themselves

Machine-checked proofs

- Every statement that can be formalized comes with a **Lean 4 proof**, hyperlinked from the paper
- The exceptions are the explicit **running-time bounds** – out of scope without a formal cost model
- This is not decoration: it **caught real errors** in the m-semiring literature that had stood for fifteen years
- These building blocks have **counter-intuitive** properties; clean, verified statements are a contribution in themselves
- Even the **NP-completeness** result is machine-checked, via **DescriptiveComplexity** – my Lean library where hardness is a **first-order reduction**, needing no machine model

Machine-checked proofs

- Every statement that can be formalized comes with a **Lean 4 proof**, hyperlinked from the paper
- The exceptions are the explicit **running-time bounds** – out of scope without a formal cost model
- This is not decoration: it **caught real errors** in the m-semiring literature that had stood for fifteen years
- These building blocks have **counter-intuitive** properties; clean, verified statements are a contribution in themselves
- Even the **NP-completeness** result is machine-checked, via **DescriptiveComplexity** – my Lean library where hardness is a **first-order reduction**, needing no machine model

The development: `https://provsql.org/lean-docs/`

Complexity library: `https://github.com/PierreSenellart/descriptive-complexity`

UNIVERSITÉ
PARIS
SCIENCES
& LETTRES

Algorithms

Two enumeration procedures

ValidCount: $\texttt{COUNT} \star C$

The count only depends on **how many** members survive: recursively decide in/out for each occurrence, enumerating subsets of the **admissible sizes**, then union the ranges selected by $\star$

ValidSum: $\texttt{SUM}(t) \star C$

A **subset-sum dynamic program**: $dp[j]$ holds the worlds realizing sum j; each new tuple either leaves a world alone or extends it. Read off the dp cells selected by $\star$

Two enumeration procedures

ValidCount: $\texttt{COUNT} \star C$

The count only depends on **how many** members survive: recursively decide in/out for each occurrence, enumerating subsets of the **admissible sizes**, then union the ranges selected by $\star$

ValidSum: $\texttt{SUM}(t) \star C$

A **subset-sum dynamic program**: $dp[j]$ holds the worlds realizing sum j; each new tuple either leaves a world alone or extends it. Read off the dp cells selected by $\star$

- Both are **proved correct** for our semantics
- Any other aggregate: **brute-force** enumeration of the non-empty worlds always works; for aggregate **vs. aggregate**, evaluate one side to a constant first

Two enumeration procedures

ValidCount: $\texttt{COUNT} \star C$

The count only depends on **how many** members survive: recursively decide in/out for each occurrence, enumerating subsets of the **admissible sizes**, then union the ranges selected by $\star$

ValidSum: $\texttt{SUM}(t) \star C$

A **subset-sum dynamic program**: $dp[j]$ holds the worlds realizing sum j; each new tuple either leaves a world alone or extends it. Read off the dp cells selected by $\star$

- Both are **proved correct** for our semantics
- Any other aggregate: **brute-force** enumeration of the non-empty worlds always works; for aggregate **vs. aggregate**, evaluate one side to a constant first

But enumeration is exponential. When can we do better?

The absorptive collapse: only *minimal* worlds matter

Three occurrences annotated x_1, x_2, x_3 in an arbitrary m-semiring, carrying attribute values $a = 3, 2, 2$ respectively, under `SUM(a) >= 5`. The worlds whose sum reaches 5 are $\{x_1, x_2\}$, $\{x_1, x_3\}$ and $\{x_1, x_2, x_3\}$, so the provenance is

$$(x_1 \otimes x_2) \otimes (\mathbb{1} \ominus x_3) \;\oplus\; (x_1 \otimes x_3) \otimes (\mathbb{1} \ominus x_2) \;\oplus\; x_1 \otimes x_2 \otimes x_3$$

The absorptive collapse: only *minimal* worlds matter

Three occurrences annotated x_1, x_2, x_3 in an arbitrary m-semiring, carrying attribute values $a = 3, 2, 2$ respectively, under `SUM(a) >= 5`. The worlds whose sum reaches 5 are $\{x_1, x_2\}$, $\{x_1, x_3\}$ and $\{x_1, x_2, x_3\}$, so the provenance is

$$(x_1 \otimes x_2) \otimes (\mathbb{1} \ominus x_3) \;\oplus\; (x_1 \otimes x_3) \otimes (\mathbb{1} \ominus x_2) \;\oplus\; x_1 \otimes x_2 \otimes x_3$$

This family of worlds is **upward-closed**; in an **absorptive** m-semiring the $\ominus$-factors are then redundant, so it collapses to the $\oplus$-sum over the **minimal** valid worlds alone:

$$(x_1 \otimes x_2) \oplus (x_1 \otimes x_3) \;=\; x_1 \otimes (x_2 \oplus x_3)$$

The absorptive collapse: only *minimal* worlds matter

Three occurrences annotated x_1, x_2, x_3 in an arbitrary m-semiring, carrying attribute values $a = 3, 2, 2$ respectively, under `SUM(a) >= 5`. The worlds whose sum reaches 5 are $\{x_1, x_2\}$, $\{x_1, x_3\}$ and $\{x_1, x_2, x_3\}$, so the provenance is

$$(x_1 \otimes x_2) \otimes (\mathbb{1} \ominus x_3) \;\oplus\; (x_1 \otimes x_3) \otimes (\mathbb{1} \ominus x_2) \;\oplus\; x_1 \otimes x_2 \otimes x_3$$

This family of worlds is **upward-closed**; in an **absorptive** m-semiring the $\ominus$-factors are then redundant, so it collapses to the $\oplus$-sum over the **minimal** valid worlds alone:

$$(x_1 \otimes x_2) \oplus (x_1 \otimes x_3) \;=\; x_1 \otimes (x_2 \oplus x_3)$$

Exponentially many valid worlds, but only **polynomially many minimal** ones when the threshold is fixed $\Rightarrow$ **polynomial time**

Tractable cases (data complexity)

Condition	Requirement on the m-semiring
`MIN(a)` $\star$ c, `MAX(a)` $\star$ c, `PICKFIRST(a)` $\star$ c for *any* $\star \in \{<, \leqslant, =, \geqslant, >, \neq\}$	absorptive $+$ $\otimes$ distr. over $\ominus$
`COUNT(*) >=` k and `COUNT(*) >` k, k fixed	absorptive
`SUM(a) >=` c and `SUM(a) >` c over $\mathbb{N}$ with $c/a \leqslant k$ for every nonzero value a	absorptive
`COUNT(*) =` k, `<=` k, `<` k, k fixed	**any** m-semiring

Tractable cases (data complexity)

Condition	Requirement on the m-semiring
`MIN(a)` $\star$ c, `MAX(a)` $\star$ c, `PICKFIRST(a)` $\star$ c for *any* $\star \in \{<, \leqslant, =, \geqslant, >, \neq\}$	absorptive $+$ $\otimes$ distr. over $\ominus$
`COUNT(*)` >= k and `COUNT(*)` > k, k fixed	absorptive
`SUM(a)` >= c and `SUM(a)` > c over $\mathbb{N}$ with $c/a \leqslant k$ for every nonzero value a	absorptive
`COUNT(*)` = k, <= k, < k, k fixed	**any** m-semiring

- The **monotone** `COUNT`/`SUM` cases rely on the minimal-world collapse, hence on absorptivity
- The **bounded** cases ($= k$, $\leqslant k$, $< k$) need no absorptivity: there are only polynomially many worlds to begin with

Probabilities without enumerating worlds

When a group's contributors are **independent**, we do not need the worlds at all – just a **convolution DP** over $\rho_n(j)$, the probability that exactly j of the first n contributors survive (a **Poisson-binomial** distribution):

$$\rho_n(j) = (1 - p_n)\,\rho_{n-1}(j) + p_n\,\rho_{n-1}(j-1)$$

Probabilities without enumerating worlds

When a group's contributors are **independent**, we do not need the worlds at all – just a **convolution DP** over $\rho_n(j)$, the probability that exactly j of the first n contributors survive (a **Poisson-binomial** distribution):

$$\rho_n(j) = (1 - p_n)\,\rho_{n-1}(j) + p_n\,\rho_{n-1}(j-1)$$

With $p_1 = 0.9$, $p_2 = 0.5$, $p_3 = 0.4$:

	j=0	j=1	j=2	j=3
ρ_0	1			
ρ_1	.1	.9		
ρ_2	.05	.50	.45	
ρ_3	.03	.32	.47	.18

$\Pr[\texttt{COUNT}(*) \geqslant 2] = .47 + .18 = 0.65$

Probabilities without enumerating worlds

When a group's contributors are **independent**, we do not need the worlds at all – just a **convolution DP** over $\rho_n(j)$, the probability that exactly j of the first n contributors survive (a **Poisson-binomial** distribution):

$$\rho_n(j) = (1 - p_n)\,\rho_{n-1}(j) + p_n\,\rho_{n-1}(j-1)$$

With $p_1 = 0.9$, $p_2 = 0.5$, $p_3 = 0.4$:

	$j{=}0$	$j{=}1$	$j{=}2$	$j{=}3$
ρ_0	1			
ρ_1	.1	.9		
ρ_2	.05	.50	.45	
ρ_3	.03	.32	.47	.18

$\Pr[\texttt{COUNT}(*) \geqslant 2] = .47 + .18 = 0.65$

Exact, for *every* comparison operator

`COUNT(*)`	$O(N \cdot \min(C, N - C))$
`SUM(`t`)`	$O(N \cdot S)$, $S = \sum_i t(u_i)$
`MIN`/`MAX`	$O(N)$

Probabilities without enumerating worlds

When a group's contributors are **independent**, we do not need the worlds at all – just a **convolution DP** over $\rho_n(j)$, the probability that exactly j of the first n contributors survive (a **Poisson-binomial** distribution):

$$\rho_n(j) = (1 - p_n)\,\rho_{n-1}(j) + p_n\,\rho_{n-1}(j-1)$$

With $p_1 = 0.9$, $p_2 = 0.5$, $p_3 = 0.4$:

	$j{=}0$	$j{=}1$	$j{=}2$	$j{=}3$
ρ_0	1			
ρ_1	.1	.9		
ρ_2	.05	.50	.45	
ρ_3	.03	.32	.47	.18

$\Pr[\texttt{COUNT}(*) \geqslant 2] = .47 + .18 = 0.65$

Exact, for *every* comparison operator

`COUNT(*)`	$O(N \cdot \min(C, N - C))$
`SUM`(t)	$O(N \cdot S)$, $S = \sum_i t(u_i)$
`MIN`/`MAX`	$O(N)$

Including the **non-monotone** $\leqslant, <, =, \neq$, which the symbolic results could not reach – and **no knowledge compiler**

In ProvSQL: Implementation and Experiments

Implementation

In **C++**, on top of the circuit machinery ProvSQL already had:

- Query rewriting emits an **aggregate token** per group and a **comparison gate** per `HAVING` atom; the possible-world sum is triggered **lazily**, at semiring-evaluation time

Implementation

In **C++**, on top of the circuit machinery ProvSQL already had:

- Query rewriting emits an **aggregate token** per group and a **comparison gate** per `HAVING` atom; the possible-world sum is triggered **lazily**, at semiring-evaluation time
- **Generic range check:** compute the **minimum and maximum** the aggregate can reach over all worlds; return $\mathbb{0}$ if the condition is **unsatisfiable**, or the $\oplus$ of all worlds if it is **necessarily true**

Implementation

In **C++**, on top of the circuit machinery ProvSQL already had:

- Query rewriting emits an **aggregate token** per group and a **comparison gate** per `HAVING` atom; the possible-world sum is triggered **lazily**, at semiring-evaluation time
- **Generic range check:** compute the **minimum and maximum** the aggregate can reach over all worlds; return $\mathbb{0}$ if the condition is **unsatisfiable**, or the $\oplus$ of all worlds if it is **necessarily true**
- **DP tightening:** bound the maximum tracked sum J rather than running all the way to C; skip the sums not yet reachable

Implementation

In **C++**, on top of the circuit machinery ProvSQL already had:

- Query rewriting emits an **aggregate token** per group and a **comparison gate** per `HAVING` atom; the possible-world sum is triggered **lazily**, at semiring-evaluation time
- **Generic range check:** compute the **minimum and maximum** the aggregate can reach over all worlds; return $\mathbb{0}$ if the condition is **unsatisfiable**, or the $\oplus$ of all worlds if it is **necessarily true**
- **DP tightening:** bound the maximum tracked sum J rather than running all the way to C; skip the sums not yet reachable
- **Absorptivity** exploited whenever the semiring allows it; **semiring identities folded** on the fly ($\otimes\mathbb{1}$, $\oplus\mathbb{0}\ldots$)

Implementation

In **C++**, on top of the circuit machinery ProvSQL already had:

- Query rewriting emits an **aggregate token** per group and a **comparison gate** per `HAVING` atom; the possible-world sum is triggered **lazily**, at semiring-evaluation time
- **Generic range check:** compute the **minimum and maximum** the aggregate can reach over all worlds; return $\mathbb{0}$ if the condition is **unsatisfiable**, or the $\oplus$ of all worlds if it is **necessarily true**
- **DP tightening:** bound the maximum tracked sum J rather than running all the way to C; skip the sums not yet reachable
- **Absorptivity** exploited whenever the semiring allows it; **semiring identities folded** on the fly ($\otimes\mathbb{1}$, $\oplus\mathbb{0}\ldots$)
- The **Poisson-binomial** path for probabilistic `COUNT`/`SUM`/`MIN`/`MAX` comparisons

Experimental setup

Data

- COCO 2017 [Lin et al., 2014] images, objects detected by YOLOv5 [Ultralytics, 2021]
- One tuple per detection, **probability = confidence**, independent
- **239 377** images, **1 570 565** object annotations
- Exactly the setting of [Yunus et al., 2025]

Queries

- `COUNT(*) >=` k and `COUNT(*) <=` k on class 72 (*refrigerator*), $k = 0 \ldots 15$
- Against the equivalent **self-**`JOIN` rewriting
- ProvSQL 1.8.0, PostgreSQL 18, 3 runs each

Experimental setup

Data

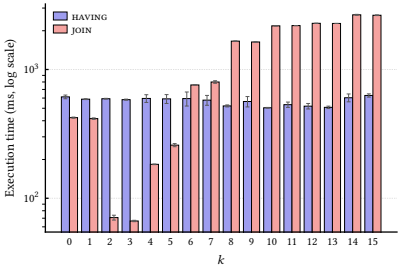
- COCO 2017 [Lin et al., 2014] images, objects detected by YOLOv5 [Ultralytics, 2021]
- One tuple per detection, **probability = confidence**, independent
- **239 377** images, **1 570 565** object annotations
- Exactly the setting of [Yunus et al., 2025]

Queries

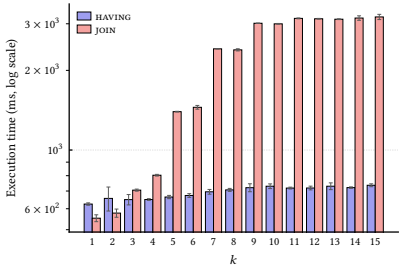
- `COUNT(*) >=` k and `COUNT(*) <=` k on class 72 (*refrigerator*), $k = 0 \ldots 15$
- Against the equivalent **self-`JOIN`** rewriting
- ProvSQL 1.8.0, PostgreSQL 18, 3 runs each

At most 13 refrigerators in any image, so $k \geqslant 14$ is **vacuous**

Provenance computation: the runtime is *flat* in *k*

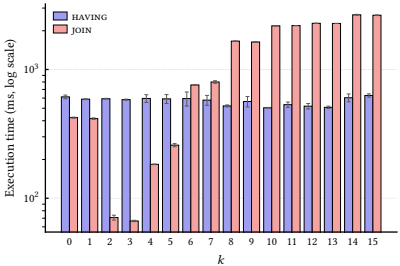


`COUNT(*) >= k`

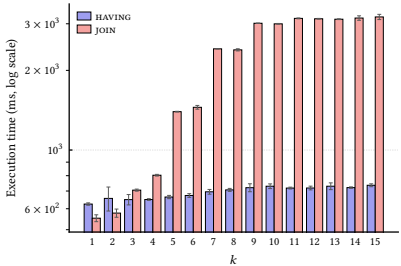


`COUNT(*) <= k`

Provenance computation: the runtime is *flat* in *k*



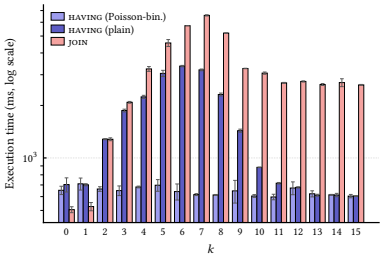
`COUNT(*) >= k`



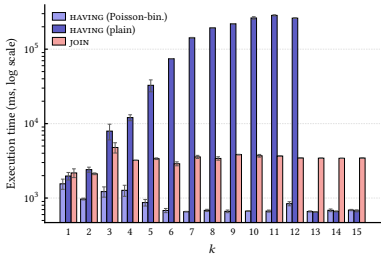
`COUNT(*) <= k`

- `JOIN` wins for **very small** *k*, then degrades: *k* self-joins is a lot of joins
- Ours is **stable in** *k*: one comparison gate, whatever the threshold – and `<=` *k* drags a non-monotone `EXCEPT` into the `JOIN` rewriting

Probability evaluation: the gap widens

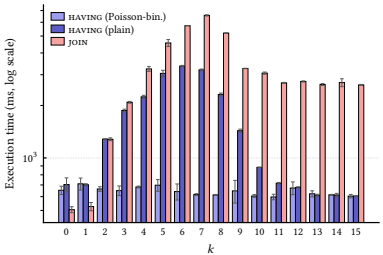


`COUNT(*) >= k`

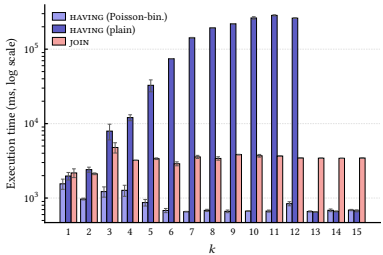


`COUNT(*) <= k` – note the scale

Probability evaluation: the gap widens



`COUNT(*) >= k`



`COUNT(*) <= k` – note the scale

- **Poisson-binomial**: fast for **every** k, both directions
- **Plain** `HAVING`: beats `JOIN` on the monotone $\geqslant k$, loses on $\leqslant k$, where non-monotone circuits are hard on the knowledge compiler
- **Both** of ours exploit that the query is **vacuous** beyond $k = 13$; the `JOIN` rewriting cannot

UNIVERSITÉ
PARIS
SCIENCES
& LETTRES

Wrap-Up

Takeaways

- `HAVING` was the **missing operator** of the semiring provenance framework – and the blocker was a semantics, not engineering

Takeaways

- `HAVING` was the **missing operator** of the semiring provenance framework – and the blocker was a semantics, not engineering
- We give a **possible-world semantics** inside each group that needs **no new operator**: $\oplus$, $\otimes$, $\ominus$ suffice, in **any** commutative m-semiring, for **any** aggregate function

Takeaways

- `HAVING` was the **missing operator** of the semiring provenance framework – and the blocker was a semantics, not engineering
- We give a **possible-world semantics** inside each group that needs **no new operator**: $\oplus$, $\otimes$, $\ominus$ suffice, in **any** commutative m-semiring, for **any** aggregate function
- It **agrees with the** `JOIN` **rewriting** whenever the m-semiring is absorptive with $\otimes$ distributive over $\ominus$ (both needed) – so for $\mathbb{B}[X]$, **the probabilities are right**; and homomorphisms still commute

Takeaways

- `HAVING` was the **missing operator** of the semiring provenance framework – and the blocker was a semantics, not engineering
- We give a **possible-world semantics** inside each group that needs **no new operator**: $\oplus$, $\otimes$, $\ominus$ suffice, in **any** commutative m-semiring, for **any** aggregate function
- It **agrees with the** `JOIN` **rewriting** whenever the m-semiring is absorptive with $\otimes$ distributive over $\ominus$ (both needed) – so for $\mathbb{B}[X]$, **the probabilities are right**; and homomorphisms still commute
- Hard in general, but **tractable** for the conditions people actually write, and probabilities come out of a **Poisson-binomial DP**

Takeaways

- `HAVING` was the **missing operator** of the semiring provenance framework – and the blocker was a semantics, not engineering
- We give a **possible-world semantics** inside each group that needs **no new operator**: $\oplus$, $\otimes$, $\ominus$ suffice, in **any** commutative m-semiring, for **any** aggregate function
- It **agrees with the** `JOIN` **rewriting** whenever the m-semiring is absorptive with $\otimes$ distributive over $\ominus$ (both needed) – so for $\mathbb{B}[X]$, **the probabilities are right**; and homomorphisms still commute
- Hard in general, but **tractable** for the conditions people actually write, and probabilities come out of a **Poisson-binomial DP**
- **Shipped in ProvSQL**: the first end-to-end provenance-aware implementation of selection on aggregate values – and **everything formalizable is proved in Lean**

Ongoing and future work

- Combine with the **query-shape trichotomy** of [Ré and Suciu, 2009], to *guarantee* polynomial-time exact or approximate probabilistic evaluation of safe `HAVING` queries

Ongoing and future work

- Combine with the **query-shape trichotomy** of [Ré and Suciu, 2009], to *guarantee* polynomial-time exact or approximate probabilistic evaluation of safe `HAVING` queries
- Aggregate-vs-aggregate comparisons: do better than evaluating one side to a constant first

Ongoing and future work

- Combine with the **query-shape trichotomy** of [Ré and Suciu, 2009], to *guarantee* polynomial-time exact or approximate probabilistic evaluation of safe `HAVING` queries
- Aggregate-vs-aggregate comparisons: do better than evaluating one side to a constant first
- Grouping or ordering **by** an aggregate value – order and uncertainty interact in subtle ways

Ongoing and future work

- Combine with the **query-shape trichotomy** of [Ré and Suciu, 2009], to *guarantee* polynomial-time exact or approximate probabilistic evaluation of safe `HAVING` queries
- Aggregate-vs-aggregate comparisons: do better than evaluating one side to a constant first
- Grouping or ordering **by** an aggregate value – order and uncertainty interact in subtle ways
- More aggregate functions with dedicated polynomial-time paths

Ongoing and future work

- Combine with the **query-shape trichotomy** of [Ré and Suciu, 2009], to *guarantee* polynomial-time exact or approximate probabilistic evaluation of safe `HAVING` queries
- Aggregate-vs-aggregate comparisons: do better than evaluating one side to a constant first
- Grouping or ordering **by** an aggregate value – order and uncertainty interact in subtle ways
- More aggregate functions with dedicated polynomial-time paths

Open source – try it: `https://provsql.org/`

In your browser, no install: `https://provsql.org/playground/`

The Lean development: `https://provsql.org/lean-docs/`

UNIVERSITÉ
PARIS
SCIENCES
& LETTRES

Thank you!

References I

Antoine Amarilli and Mikaël Monet. Example of a naturally ordered semiring which is not an m-semiring. http://math.stackexchange.com/questions/1966858, 2016.

Yael Amsterdamer, Daniel Deutch, and Val Tannen. On the limitations of provenance for queries with difference. In Peter Buneman and Juliana Freire, editors, *3rd Workshop on the Theory and Practice of Provenance, TaPP'11, Heraklion, Crete, Greece, June 20-21, 2011*. USENIX Association, 2011a. URL https://www.usenix.org/conference/tapp11/limitations-provenance-queries-difference.

Yael Amsterdamer, Daniel Deutch, and Val Tannen. Provenance for aggregate queries. In *Proceedings of the thirtieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 153–164, 2011b.

Bahareh Sadat Arab, Su Feng, Boris Glavic, Seokki Lee, Xing Niu, and Qitian Zeng. GProM - A swiss army knife for your provenance needs. *IEEE Data Eng. Bull.*, 41(1):51–62, 2018. URL http://sites.computer.org/debull/A18mar/p51.pdf.

References II

Guillermo Badia, Phokion G. Kolaitis, and Carles Noguera. Codd's theorem for databases over semirings. *Proc. ACM Manag. Data*, 3(5):277:1–277:26, 2025. doi: 10.1145/3767713. URL https://doi.org/10.1145/3767713.

Peter Buneman, Sanjeev Khanna, and Wang Chiew Tan. Why and where: A characterization of data provenance. In Jan Van den Bussche and Victor Vianu, editors, *Database Theory - ICDT 2001, 8th International Conference, London, UK, January 4-6, 2001, Proceedings*, volume 1973 of *Lecture Notes in Computer Science*, pages 316–330. Springer, 2001. doi: 10.1007/3-540-44503-X_20. URL https://doi.org/10.1007/3-540-44503-X_20.

Floris Geerts and Antonella Poggi. On database query languages for K-relations. *J. Appl. Log.*, 8(2):173–185, 2010. doi: 10.1016/J.JAL.2009.09.001. URL https://doi.org/10.1016/j.jal.2009.09.001.

Todd J. Green, Gregory Karvounarakis, and Val Tannen. Provenance semirings. In Leonid Libkin, editor, *Proceedings of the Twenty-Sixth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 11-13, 2007, Beijing, China*, pages 31–40. ACM, 2007. doi: 10.1145/1265530.1265535. URL https://doi.org/10.1145/1265530.1265535.

References III

Tomasz Imieliński and Witold Lipski Jr. Incomplete information in relational databases. *J. ACM*, 31(4):761–791, 1984. doi: 10.1145/1634.1886. URL https://doi.org/10.1145/1634.1886.

Tsung-Yi Lin, Michael Maire, Serge J. Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. In David J. Fleet, Tomás Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision - ECCV 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V*, volume 8693 of *Lecture Notes in Computer Science*, pages 740–755. Springer, 2014. doi: 10.1007/978-3-319-10602-1_48. URL https://doi.org/10.1007/978-3-319-10602-1_48.

Paulo Pintor, Rogério Costa, and José Moreira. A dbms-independent approach for capturing provenance polynomials through query rewriting. *CoRR*, abs/2508.14608, 2025. doi: 10.48550/ARXIV.2508.14608. URL https://doi.org/10.48550/arXiv.2508.14608.

Christopher Ré and Dan Suciu. The trichotomy of HAVING queries on a probabilistic database. *VLDB J.*, 18(5):1091–1116, 2009. doi: 10.1007/S00778-009-0151-4. URL https://doi.org/10.1007/s00778-009-0151-4.

References IV

Aryak Sen, Silviu Maniu, and Pierre Senellart. Provsql: A general system for keeping track of the provenance and probability of data. In *Proc. ICDE*, Montréal, Canada, May 2026.

Pierre Senellart. Provenance and probabilities in relational databases. *SIGMOD Rec.*, 46(4): 5–15, 2017. doi: 10.1145/3186549.3186551. URL https://doi.org/10.1145/3186549.3186551.

Pierre Senellart, Louis Jachiet, Silviu Maniu, and Yann Ramusat. ProvSQL: Provenance and probability management in PostgreSQL. *Proc. VLDB Endow.*, 11(12):2034–2037, 2018. doi: 10.14778/3229863.3236253. URL http://www.vldb.org/pvldb/vol11/p2034-senellart.pdf.

Dan Suciu, Dan Olteanu, Christopher Ré, and Christoph Koch. *Probabilistic Databases*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2011. ISBN 978-3-031-00751-4. doi: 10.2200/S00362ED1V01Y201105DTM016. URL https://doi.org/10.2200/S00362ED1V01Y201105DTM016.

Ultralytics. YOLOv5: A state-of-the-art real-time object detection system. https://docs.ultralytics.com, 2021.

References V

Fajrian Yunus, Pratik Karmakar, Pierre Senellart, Talel Abdessalem, and Stéphane Bressan. Using A probabilistic database in an image retrieval application. In Alkis Simitsis, Bettina Kemme, Anna Queralt, Oscar Romero, and Petar Jovanovic, editors, *Proceedings 28th International Conference on Extending Database Technology, EDBT 2025, Barcelona, Spain, March 25-28, 2025*, pages 1106–1109. OpenProceedings.org, 2025. doi: 10.48786/EDBT.2025.100. URL https://doi.org/10.48786/edbt.2025.100.