Subset Sampling over Joins

Aryan Esmailpour[1] Xiao Hu[2] Jinchao Huang[3] Stavros Sintos[1]

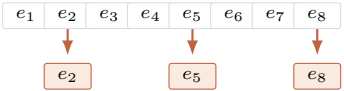
[1]University of Illinois Chicago [2]University of Waterloo [3]The Chinese University of Hong Kong

PODS 2026 June 2026 ▪ Bengaluru, India

A Familiar Primitive, but in the Relational Setting

Explicit subset sampling

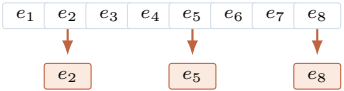
Given items $e_1, \ldots, e_n$, include each item independently with probability $\mathrm{p}(e_i)$.



A Familiar Primitive, but in the Relational Setting

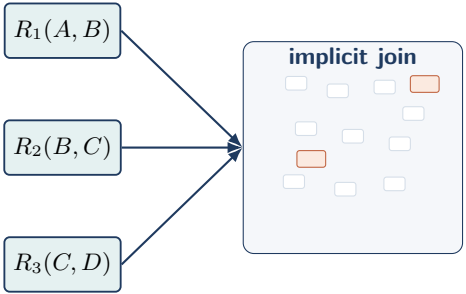
Explicit subset sampling

Given items $e_1, \ldots, e_n$, include each item independently with probability $\mathrm{p}(e_i)$.



Relational subset sampling

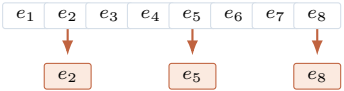
The items are not stored explicitly: they are all tuples in $Join(Q)$.



A Familiar Primitive, but in the Relational Setting

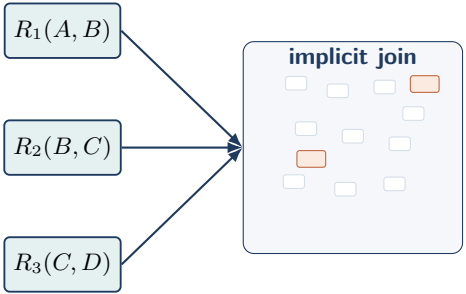
Explicit subset sampling

Given items $e_1, \ldots, e_n$, include each item independently with probability $\mathrm{p}(e_i)$.



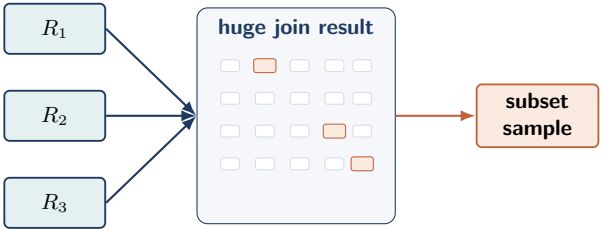
Relational subset sampling

The items are not stored explicitly: they are all tuples in $Join(Q)$.



Goal: generate exactly the subset sample, without materializing the join.

Why Subset Sampling over Joins?



$|Join(Q)|$ **can be much larger than** N

Many tasks do not need the full join

- approximate analytics and visualization
- optimizer statistics and selectivity estimation
- relational ML, coresets, and data condensation

Importance-aware sampling

Subset sampling selects join results by considering the "importance" of constituent tuples.

More motivation: A concurrent independent SIGMOD paper, *Poisson Sampling over Acyclic Joins* [BNPV'26], gives systems-oriented examples.

Problem Definition

Input. Acyclic join $Q = \{R_1, \ldots, R_k\}$; per-tuple weights $\mathrm{p}_j : R_j \to [0, 1]$.

Join-result weight:

$$\mathrm{p}(\mathbf{u}) \;=\; \mathcal{F}\big(\mathrm{p}_1(\mathbf{u}[\mathit{schema}(R_1)]), \ldots, \mathrm{p}_k(\mathbf{u}[\mathit{schema}(R_k)])\big), \quad \mathcal{F} \in \{\text{PRODUCT, MIN, MAX, SUM}\}.$$

Relational Subset Sampling

Return $\mathbf{X} \subseteq \mathit{Join}(Q)$ where each $\mathbf{u} \in \mathit{Join}(Q)$ is in $\mathbf{X}$ *independently* w.p. $\mathrm{p}(\mathbf{u})$.

Problem Definition

Input. Acyclic join $Q = \{R_1, \ldots, R_k\}$; per-tuple weights $\mathrm{p}_j : R_j \to [0, 1]$.

Join-result weight:

$$\mathrm{p}(\mathbf{u}) \;=\; \mathcal{F}\big(\mathrm{p}_1(\mathbf{u}[\mathit{schema}(R_1)]), \ldots, \mathrm{p}_k(\mathbf{u}[\mathit{schema}(R_k)])\big), \quad \mathcal{F} \in \{\text{PRODUCT, MIN, MAX, SUM}\}.$$

Relational Subset Sampling

Return $\mathbf{X} \subseteq \mathit{Join}(Q)$ where each $\mathbf{u} \in \mathit{Join}(Q)$ is in $\mathbf{X}$ *independently* w.p. $\mathrm{p}(\mathbf{u})$.

Three settings:

Static Index	**One-shot**	**Dynamic (insertions)**
Sample from data structure	One sample from scratch	Base tuples are streaming in

Key quantities (data complexity).

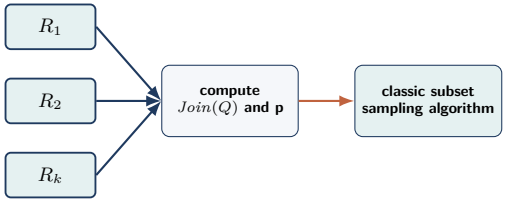
$N = \sum_j |R_j|$ = input size; $\mu_\Psi = \mathbf{E}[|\mathbf{X}|] = \sum_{\mathbf{u}} \mathrm{p}(\mathbf{u})$ = expected sample size; $|\mathit{Join}(Q)|$ = join size.

Want runtime in N and μ_Ψ — *not* in $|\mathit{Join}(Q)|$.

Baseline: Materialize the Join, Then Sample

Reduction to the classic problem

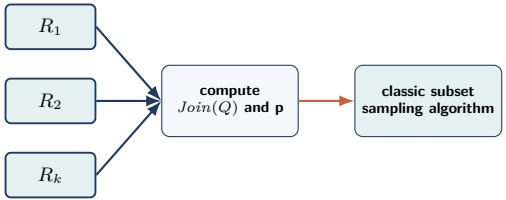
Build the explicit instance $\Psi = \langle Join(Q), \mathrm{p} \rangle$: compute $\mathbf{u} \in Join(Q)$ and their probabilities from the base-tuple weights.



Baseline: Materialize the Join, Then Sample

Reduction to the classic problem

Build the explicit instance $\Psi = \langle Join(Q), \mathrm{p} \rangle$: compute $\mathbf{u} \in Join(Q)$ and their probabilities from the base-tuple weights.



Setting	Preproc.	Sampling	Space
Static index	$O(N + \lvert Join(Q) \rvert)$	$O(1 + \mu_\Psi)$	$O(\lvert Join(Q) \rvert)$
One-shot	—	$O(N + \lvert Join(Q) \rvert)$	$O(\lvert Join(Q) \rvert)$

Materialization bottleneck

AGM gives $|Join(Q)| = \Theta(N^{\rho^*})$ in the worst case. The explicit baseline pays for the whole join even when $\mu_\Psi \ll |Join(Q)|$.

Our Results at a Glance (PRODUCT, acyclic joins)

Setting	Method	Preproc. / Update	Sampling Time	Space
Static Index	Baseline	$O(N + \lvert Join(Q)\rvert)$	$O(1 + \mu_\Psi)$	$O(\lvert Join(Q)\rvert)$
	Ours	$O(N \log N \log\log N)$	$O(1 + \mu_\Psi \log N)$	$O(N \log N)$
One-shot	Baseline	—	$O(N + \lvert Join(Q)\rvert)$	$O(\lvert Join(Q)\rvert)$
	Ours	—	$O(N \log^2 N + \mu_\Psi)$	$O(N \log^2 N + \mu_\Psi)$
Dynamic (Index)	**Basic**	$O(\log^k N)$	$O(1 + \mu_\Psi \log N)$	$O(N \log^{k-1} N)$
	Optimized	$O(\log^3 N \log\log N)$	$O(1 + \mu_\Psi \log N)$	$O(N \log N)$
Dynamic (One-shot)	**Ours**	—	$O(N \log^3 N \log\log N + \mu_\Psi \log N)$	$O(N \log N)$

- **Near-optimal** in N and μ_Ψ up to polylog factors.
- Cyclic joins: replace N by N^w via tree decomposition.
- Extends to MIN, MAX, SUM with an extra $\log N$ factor (Appendix).

Building Blocks: Sampling by DirectAccess

Uniform Probability

For a perfect-uniform subset sampling instance $\Psi = \langle S, p \rangle$ with DirectAccess oracle, each subset sampling query can be answered in $O(1 + \mu_\Psi)$ expected time.

$i \leftarrow i + 1 + \mathsf{Geo}(p)$, then output $\mathsf{DirectAccess}(i)$

Building Blocks: Sampling by DirectAccess

Uniform Probability

For a perfect-uniform subset sampling instance $\Psi = \langle S, p \rangle$ with DirectAccess oracle, each subset sampling query can be answered in $O(1 + \mu_\Psi)$ expected time.

$i \leftarrow i + 1 + \text{Geo}(p)$, then output DirectAccess(i)

Non-uniform Probability

For a subset sampling instance $\Psi = \langle S, \mathrm{p} \rangle$ with DirectAccess oracle and $\max \mathrm{p} \le p^+$, each subset sampling query can be answered in $O(1 + |S| \cdot p^+)$ expected time.

near-uniform: $\max \mathrm{p} \le \beta \min \mathrm{p}$, cost $O(1 + \beta\mu_\Psi)$

light: $\max \mathrm{p} \le 1/|S|$, cost $O(1)$

Divide and Conquer

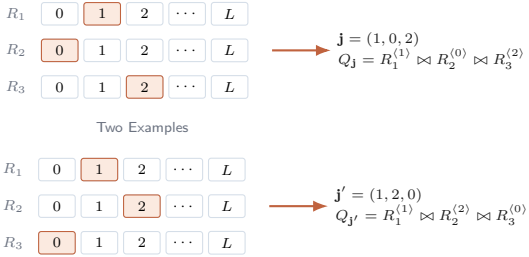
Partition a general problem instance into multiple subproblems with nice property.
If a subproblem is near-uniform or light *and* supports $|S|$ plus DirectAccess, we can sample efficiently from it.

First Attempt: Partition the Input Relations

Let $L = \lceil 2\rho^* \log N \rceil$, so $2^L \geq |Join(Q)|$. For every relation,

$$R_i^{\langle j \rangle} = \{\mathbf{u} \in R_i : 2^{-j-1} < \mathrm{p}_i(\mathbf{u}) \leq 2^{-j}\}\ (j < L), \qquad R_i^{\langle L \rangle} = \{\mathbf{u} : \mathrm{p}_i(\mathbf{u}) \leq 2^{-L}\}.$$

A bucket vector $\mathbf{j} = (j_1, \ldots, j_k)$ defines $Q_{\mathbf{j}} = \{R_1^{\langle j_1 \rangle}, \ldots, R_k^{\langle j_k \rangle}\}$, and $Join(Q) = \bigsqcup_{\mathbf{j} \in \{0,\ldots,L\}^k} Join(Q_{\mathbf{j}})$.

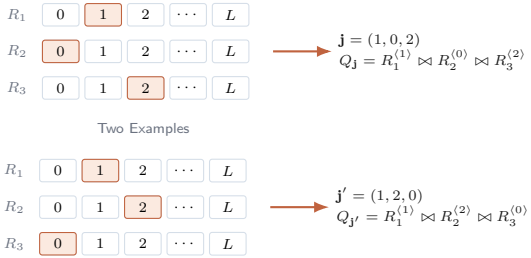


First Attempt: Partition the Input Relations

Let $L = \lceil 2\rho^* \log N \rceil$, so $2^L \geq |Join(Q)|$. For every relation,

$$R_i^{\langle j \rangle} = \{\mathbf{u} \in R_i : 2^{-j-1} < \mathrm{p}_i(\mathbf{u}) \leq 2^{-j}\}\ (j < L), \qquad R_i^{\langle L \rangle} = \{\mathbf{u} : \mathrm{p}_i(\mathbf{u}) \leq 2^{-L}\}.$$

A bucket vector $\mathbf{j} = (j_1, \ldots, j_k)$ defines $Q_{\mathbf{j}} = \{R_1^{\langle j_1 \rangle}, \ldots, R_k^{\langle j_k \rangle}\}$, and $Join(Q) = \bigsqcup_{\mathbf{j} \in \{0,\ldots,L\}^k} Join(Q_{\mathbf{j}})$.



Each sub-join has nice property

For $\mathbf{u} \in Join(Q_{\mathbf{j}})$:

$$2^{-\sum_i j_i - k} < \mathrm{p}(\mathbf{u}) \leq 2^{-\sum_i j_i}.$$

- if $\sum_i j_i < L$: $Q_{\mathbf{j}}$ is 2^k-**uniform**;
- if $\sum_i j_i \geq L$: $Q_{\mathbf{j}}$ is **light**.

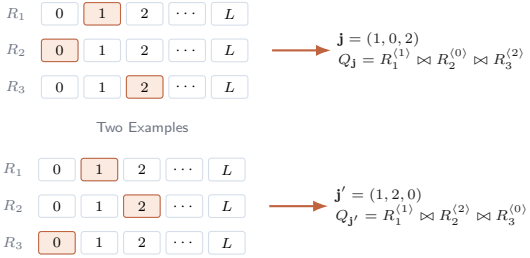
For each sub-join, we build index to support DirectAccess [Zhao et al.'18, Carmeli et al.'20].

First Attempt: Partition the Input Relations

Let $L = \lceil 2\rho^* \log N \rceil$, so $2^L \geq |Join(Q)|$. For every relation,

$$R_i^{\langle j \rangle} = \{\mathbf{u} \in R_i : 2^{-j-1} < \mathrm{p}_i(\mathbf{u}) \leq 2^{-j}\}\ (j < L), \qquad R_i^{\langle L \rangle} = \{\mathbf{u} : \mathrm{p}_i(\mathbf{u}) \leq 2^{-L}\}.$$

A bucket vector $\mathbf{j} = (j_1, \ldots, j_k)$ defines $Q_{\mathbf{j}} = \{R_1^{\langle j_1 \rangle}, \ldots, R_k^{\langle j_k \rangle}\}$, and $Join(Q) = \bigsqcup_{\mathbf{j} \in \{0,\ldots,L\}^k} Join(Q_{\mathbf{j}})$.



Each sub-join has nice property

For $\mathbf{u} \in Join(Q_{\mathbf{j}})$:

$$2^{-\sum_i j_i - k} < \mathrm{p}(\mathbf{u}) \leq 2^{-\sum_i j_i}.$$

- if $\sum_i j_i < L$: $Q_{\mathbf{j}}$ is 2^k-**uniform**;
- if $\sum_i j_i \geq L$: $Q_{\mathbf{j}}$ is **light**.

For each sub-join, we build index to support DirectAccess [Zhao et al.'18, Carmeli et al.'20].

Feasible, but still expensive: $(L+1)^k$ sub-joins; each input tuple appears in $(L+1)^{k-1}$ sub-indexes.

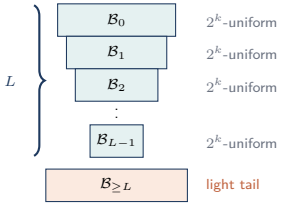
Optimization: Partition the Conceptual Join Results

The first attempt classifies $Q_{\mathbf{j}}$ using only the total $\sum_i j_i$, not the whole vector $\mathbf{j}$.

Per-tuple score. For $\mathbf{u} \in R_i$, let $\phi(\mathbf{u}) = \lfloor -\log_2 \mathrm{p}_i(\mathbf{u}) \rfloor$.

Join-result score. $\bar{\phi}(\mathbf{u}) = \sum_{i=1}^{k} \phi(\mathbf{u}[schema(R_i)])$. **Merged bucket.** $\mathcal{B}_\ell = \{\mathbf{u} \in Join(Q) : \bar{\phi}(\mathbf{u}) = \ell\}$.

For $\ell < L$, $\mathcal{B}_\ell$ merges all $Join(Q_{\mathbf{j}})$ with $\sum_i j_i = \ell$; for $\mathbf{u} \in \mathcal{B}_\ell$, $2^{-\ell-k} < \mathrm{p}(\mathbf{u}) \leq 2^{-\ell}$.



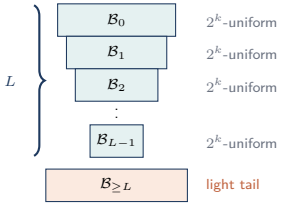
Optimization: Partition the Conceptual Join Results

The first attempt classifies $Q_{\mathbf{j}}$ using only the total $\sum_i j_i$, not the whole vector $\mathbf{j}$.

Per-tuple score. For $\mathbf{u} \in R_i$, let $\phi(\mathbf{u}) = \lfloor -\log_2 \mathrm{p}_i(\mathbf{u}) \rfloor$.

Join-result score. $\bar{\phi}(\mathbf{u}) = \sum_{i=1}^{k} \phi(\mathbf{u}[schema(R_i)])$. **Merged bucket.** $\mathcal{B}_\ell = \{\mathbf{u} \in Join(Q) : \bar{\phi}(\mathbf{u}) = \ell\}$.

For $\ell < L$, $\mathcal{B}_\ell$ merges all $Join(Q_{\mathbf{j}})$ with $\sum_i j_i = \ell$; for $\mathbf{u} \in \mathcal{B}_\ell$, $2^{-\ell-k} < \mathrm{p}(\mathbf{u}) \leq 2^{-\ell}$.



Each bucket has nice property

- if $\ell < L$: $\mathcal{B}_\ell$ is **2^k-uniform**;
- if $\ell \geq L$: $\mathcal{B}_\ell$ is **light**.

No known results for supporting DirectAccess in these buckets.

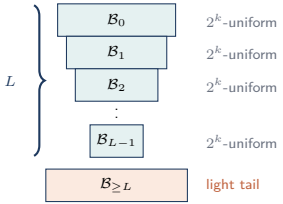
Optimization: Partition the Conceptual Join Results

The first attempt classifies $Q_{\mathbf{j}}$ using only the total $\sum_i j_i$, not the whole vector $\mathbf{j}$.

Per-tuple score. For $\mathbf{u} \in R_i$, let $\phi(\mathbf{u}) = \lfloor -\log_2 \mathrm{p}_i(\mathbf{u}) \rfloor$.

Join-result score. $\bar{\phi}(\mathbf{u}) = \sum_{i=1}^{k} \phi(\mathbf{u}[schema(R_i)])$. **Merged bucket.** $\mathcal{B}_\ell = \{\mathbf{u} \in Join(Q) : \bar{\phi}(\mathbf{u}) = \ell\}$.

For $\ell < L$, $\mathcal{B}_\ell$ merges all $Join(Q_{\mathbf{j}})$ with $\sum_i j_i = \ell$; for $\mathbf{u} \in \mathcal{B}_\ell$, $2^{-\ell-k} < \mathrm{p}(\mathbf{u}) \leq 2^{-\ell}$.



Each bucket has nice property

- if $\ell < L$: $\mathcal{B}_\ell$ is **2^k-uniform**;
- if $\ell \geq L$: $\mathcal{B}_\ell$ is **light**.

No known results for supporting DirectAccess in these buckets.

Main remaining question: support DirectAccess in each **conceptual** bucket $\mathcal{B}_\ell$.

(What we are familiar with: only one bucket.)

Preprocessing: Compute Weights of L Buckets by DP

Fix a join tree $\mathcal{T}$ of Q. For node R_i, child $R_j \in \mathcal{C}_i$, tuple $\mathbf{u} \in R_i$, score ℓ:

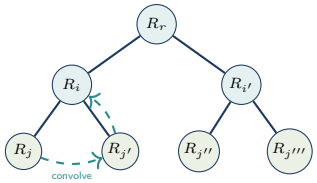
$$W^j_{i,\mathbf{u}}(\ell) = \#\{\text{tuples in } \bowtie_{h \in \mathcal{T}_i^j} R_h \text{ joinable with } \mathbf{u}, \text{ score } \ell\}; \quad M_{i,\mathbf{v}}(\ell) = \sum_{\mathbf{u} \in R_i \ltimes \mathbf{v}} W^{\emptyset}_{i,\mathbf{u}}(\ell).$$

Recursive structure ($R_{\mathsf{next}(j)}$ = next sibling of R_j):

$$W^j_{i,\mathbf{u}}(\ell) = \sum_{\ell_1+\ell_2=\ell-\phi(\mathbf{u})} M_{j,\mathbf{u}[\mathsf{key}(j)]}(\ell_1) \cdot W^{\mathsf{next}(j)}_{i,\mathbf{u}}(\ell_2).$$

This is a convolution $\Rightarrow$ use FFT

For each $(R_i, \mathbf{u}, R_j)$, all L values $W^j_{i,\mathbf{u}}(\cdot)$ in $O(L \log L)$ — not $O(L^2)$.



Bottom-up: $W^j_{i,\mathbf{u}}$ over children in decreasing order; $M_{i,\mathbf{v}}$ aggregates siblings.

Preprocessing Cost

Index built in $O(N \log N \log\log N)$ time, $O(N \log N)$ space.

Direct Access: Retrieve the τ-th Tuple in $\mathcal{B}_\ell$ in $O(\log N)$

Goal. Given $\ell < L$ and rank $\tau \in [|\mathcal{B}_\ell|]$, return the τ-th tuple in $\mathcal{B}_\ell$ *without listing* $\mathcal{B}_\ell$.

RecursiveAccess$(i, j, \mathbf{v}, \ell, \tau)$ — τ-th tuple in $\mathcal{T}_i^j \ltimes \mathbf{v}$, score ℓ

1. **Locate** $\mathbf{u} \in R_i \ltimes \mathbf{v}$: binary search on prefix sums of $W^j_{i,\mathbf{u}'}(\ell)$. $O(\log N)$
2. **Locate score pair** (ℓ_1, ℓ_2) with $\ell_1 + \ell_2 = \ell - \phi(\mathbf{u})$ absorbing τ. $O(\log N)$
3. **Compute sub-ranks** $\tau_1 = \lceil \tau / W^{\mathsf{next}(j)}_{i,\mathbf{u}}(\ell_2) \rceil$, $\tau_2 = ((\tau - 1) \bmod \cdots) + 1$.
4. **Recurse** on $\mathcal{T}_j$ with (ℓ_1, τ_1) and $\mathcal{T}_i^{\mathsf{next}(j)}$ with (ℓ_2, τ_2); join the answers.

Complexity. $k^2 = O(1)$ recursive calls, each $O(\log N)$ work $\Rightarrow O(\log N)$ total. Distinct calls return distinct tuples (induced lex order on $\mathbf{u}$ and (ℓ_1, ℓ_2)).

Theorem (Static Index)

For acyclic Q with PRODUCT weights: index of size $O(N \log N)$ built in $O(N \log N \log\log N)$; each subset-sampling query in expected $O(1 + \mu_\Psi \log N)$.

One-Shot Subset Sampling — Batching DirectAccess Queries

Baseline (static-index + one query): $O(N \log N \log\log N + \mu_\Psi \log N)$. The $\mu_\Psi \log N$ hurts when $\mu_\Psi \gg N$.

Where the $\log N$ comes from. Two binary searches inside RecursiveAccess: locating $\mathbf{u} \in R_i \ltimes \mathbf{v}$, and locating the score pair (ℓ_1, ℓ_2).

Batch all rank queries together — BatchRecursiveAccess$(i, j, \mathcal{P}_{i,j})$

1. Precompute prefix arrays $X_{i,j,\mathbf{v},\ell}$ and score-pair arrays $Y_{i,j,\mathbf{u},\ell}$.
2. **Radix-sort** the rank set $\mathcal{P}_{i,j}$ by τ; group by $(i, j, \mathbf{v}, \ell)$.
3. *Sweep once* through each array — no per-query binary search.
4. Recurse on the two children with the propagated rank sets.

Theorem (One-shot)

One subset sample in expected time $O\big(\min\{N \log N \log\log N + \mu_\Psi \log N,\ N \log^2 N + \mu_\Psi\}\big)$. Faster in the high-output regime where the $\mu_\Psi \log N$ term dominates.

Supporting Insertion Stream

Why exact maintenance is hopeless. Even maintaining the join *size* on a 3-chain join takes $\Omega(\sqrt{N})$ per insertion — far too slow.

Idea: maintain power-of-2 aggregate upper bounds

- Store $\hat{M}_{i,\mathbf{v}}(\ell) = \sum_{\mathbf{u}} \tilde{W}^{\emptyset}_{i,\mathbf{u}}(\ell)$, then $\tilde{M}_{i,\mathbf{v}}(\ell) = 2^{\lceil \log \hat{M}_{i,\mathbf{v}}(\ell) \rceil}$.
- Insertion-time recursion: only re-convolve at $R_{\mathsf{parent}(i)}$ when $\tilde{M}$ *crosses* a power of 2.
- $\tilde{M}_{i,\mathbf{v}}(\ell)$ doubles at most $O(\log N)$ times — charges only logarithmically.

Querying with approximate stats. The implicit array contains *dummy* tuples. Rejection sampling on the fly: constant-factor approximation $\Rightarrow$ each draw is real with probability $\geq 2^{-c}$.

Theorem (Dynamic, optimized)

Under tuple insertions, an index on acyclic Q is maintained with

$$\text{amortized update } O(\log^3 N \log\log N), \qquad \text{sample } O(1 + \mu_\Psi \log N), \qquad \text{space } O(N \log N).$$

Stream one-shot: $O(N \log^3 N \log\log N + \mu_\Psi \log N)$. *Full dynamics:* non-hierarchical joins have OMv lower bound $\Omega(N^{0.5-\varepsilon})$ for Boolean maintenance.

Takeaways & Open Directions

What we showed

Formulate **subset sampling over joins** and design an efficient framework. Near-optimal in N and μ_Ψ:

- Static: $O(N \log N \log\log N)$ build, $O(1 + \mu_\Psi \log N)$ query.
- One-shot: $O\big(\min\{N \log N \log\log N + \mu_\Psi \log N,\ N \log^2 N + \mu_\Psi\}\big)$.
- Dynamic: $O(\log^3 N \log\log N)$ update.

Reusable techniques

- **Database Partitioning** — make each subproblem easy to handle.
- **DP + FFT** — exploit convolutional structure.
- **Batching DirectAccess** — share computation among DirectAccess queries.
- **Power-of-2 approximations** — charge only when a counter doubles.

Open problems. Non-decomposable / holistic aggregations; practical engineering [BNPV'26].

Thank you — questions?