# Extending Extended Statistics to Joins

Alexandra Wang, EDB
pgconf.dev 2026

Link to the slides:
https://github.com/l-wang/pgconfdev-2026/blob/main/ext-stats.pdf

©EDB 2026 — ALL RIGHTS RESERVED.

# Agenda

- The Problem
- The Ongoing Discussion for Join Statistics
- Current Work
  - Design choices
  - Benchmark Results
- Alternative, Next Steps, and Open Questions


©EDB 2026 — ALL RIGHTS RESERVED.

# The Problem


©EDB 2026 — ALL RIGHTS RESERVED.

# The Independence Assumption - Multiple Columns

**P(A AND B) = P(A) × P(B)** **The planner assumes independence until told otherwise.**

*SELECT * FROM customers WHERE city = '**Vancouver**' AND country = '**Canada**';*

**P(city = 'Vancouver')** = 0.056 ← from pg_stats MCV

**P(country = 'Canada')** = 0.05 ← from pg_stats MCV

**P(Vancouver AND Canada)** = 0.056 × 0.05 = 0.0028

Estimated rows: 0.0028 × 200,000 = 560

Actual rows: 5,000 ← **9x off**

**The majority of the Vancouver rows are in Canada — they are functionally dependent.**

** There is a Vancouver city in the Washington state in the US*

EDB POSTGRES AI

©EDB 2026 — ALL RIGHTS RESERVED.

# Extended Statistics

**Defines data correlation between multiple columns**

Example:

```
-- Tells planner: city and zip_code are correlated
CREATE STATISTICS s1 (ndistinct, mcv) ON city, country FROM customers;
```

- Introduced PostgreSQL 10 (Tomas Vondra, 2017)
- Captures correlations **WITHIN a single table**
- Stat kinds: **ndistinct, functional dependencies, MCV lists**
- Catalog: ***pg_statistic_ext (definition)*** + ***pg_statistic_ext_data (payload)***

EDB POSTGRES AI

©EDB 2026 — ALL RIGHTS RESERVED.

# Extended Statistics - Current Limits

**Single-table only.** Cannot help with joins.

What happens when the correlation **crosses a join boundary**?

```
SELECT *
FROM A JOIN B
 ON (B.id = A.foreign_key)
WHERE B.filter_col = 'popular-filter-value';
```

- Filter on dimension table (B)
- Join key correlation in fact table (A)
- Stats stored per relation
- Cross-table correlation invisible



©EDB 2026 — ALL RIGHTS RESERVED.

A 2015 VLDB paper: https://www.vldb.org/pvldb/vol9/p204-leis.pdf

# The Join Order Benchmark (JOB)

- Uses the IMDB data set, 113 queries
- Queries have between 3 and 16 joins, with an average of 8 joins per query

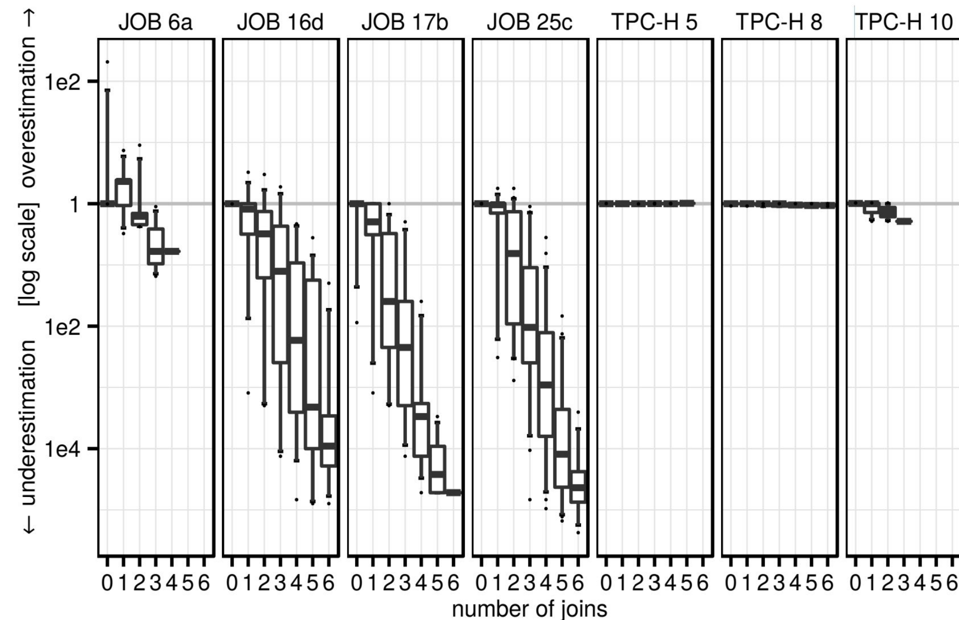



**Figure 4: PostgreSQL cardinality estimates for 4 JOB queries and 3 TPC-H queries**

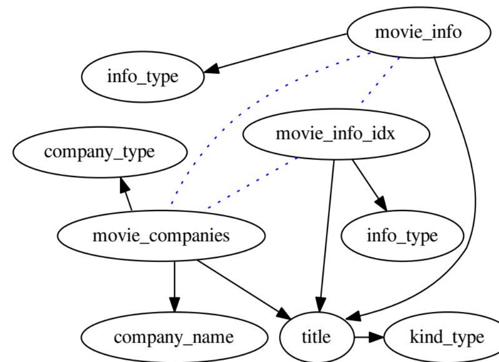



**Figure 2: Typical query graph of our workload**

## How Good Are Query Optimizers, Really?


Viktor Leis
TUM
leis@in.tum.de

Andrey Gubichev
TUM
gubichev@in.tum.de

Atanas Mirchev
TUM
mirchev@in.tum.de

Peter Boncz
CWI
p.boncz@cwi.nl

Alfons Kemper
TUM
kemper@in.tum.de

Thomas Neumann
TUM
neumann@in.tum.de



### ABSTRACT

Finding a good join order is crucial for query performance. In this paper, we introduce the Join Order Benchmark (JOB) and experimentally revisit the main components in the classic query optimizer architecture using a complex, real-world data set and realistic multi-join queries. We investigate the quality of industrial-strength cardinality estimators and find that all estimators routinely produce large errors. We further show that while estimates are essential for finding a good join order, query performance is unsatisfactory if the query engine relies too heavily on these estimates. Using another set of experiments that measure the impact of the cost model, we find that it has much less influence on query performance than ...rdinality estimates. Finally, we investigate plan enumera- ...chniques comparing exhaustive dynamic programming with ...tic algorithms and find that exhaustive enumeration improves ...mance despite the sub-optimal cardinality estimates.


### INTRODUCTION

...problem of finding a good join order is one of the most stud- ...oblems in the database field. Figure 1 illustrates the classical, ...ased approach, which dates back to System R [36]. To obtain ...cient query plan, the query optimizer enumerates some subset ...valid join orders, for example using dynamic programming. ...cardinality estimates as its principal input, the cost model ...hooses the cheapest alternative from semantically equivalent ...lternatives.

...oretically, as long as the cardinality estimations and the cost ...are accurate, this architecture obtains the optimal query plan. ...lity, cardinality estimates are usually computed based on sim- ...g assumptions like uniformity and independence. In real- ...data sets, these assumptions are *frequently* wrong, which ...ead to sub-optimal and sometimes disastrous plans.

...his *experiments and analyses* paper we investigate the three components of the classical query optimization architecture ...er to answer the following questions:

- How good are cardinality estimators and when do bad estimates lead to slow queries?


This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-nd/4.0/. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org.



*Proceedings of the VLDB Endowment,* Vol. 9, No. 3



Copyright 2015 VLDB Endowment 2150-8097/15/11.

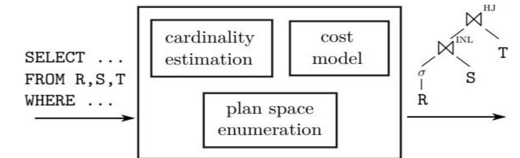




**Figure 1: Traditional query optimizer architecture**

- How important is an accurate cost model for the overall query optimization process?
- How large does the enumerated plan space need to be?

To answer these questions, we use a novel methodology that allows us to isolate the influence of the individual optimizer components on query performance. Our experiments are conducted using a real-world data set and 113 multi-join queries that provide a challenging, diverse, and realistic workload. Another novel aspect of this paper is that it focuses on the increasingly common main-memory scenario, where all data fits into RAM.

The main contributions of this paper are listed in the following:

- We design a challenging workload named *Join Order Benchmark* (JOB), which is based on the IMDB data set. The benchmark is publicly available to facilitate further research.
- To the best of our knowledge, this paper presents the first end-to-end study of the join ordering problem using a real-world data set and realistic queries.
- By quantifying the contributions of cardinality estimation, the cost model, and the plan enumeration algorithm on query performance, we provide guidelines for the complete design of a query optimizer. We also show that many disastrous plans can easily be avoided.

The rest of this paper is organized as follows: We first discuss important background and our new benchmark in Section 2. Section 3 shows that the cardinality estimators of the major relational database systems produce bad estimates for many realistic queries, in particular for multi-join queries. The conditions under which these bad estimates cause slow performance are analyzed in Section 4. We show that it very much depends on how much the query engine relies on these estimates and on how complex the physical database design is, i.e., the number of indexes available. Query engines that mainly rely on hash joins and full table scans,


EDB POSTGRES AI

# The Join Order Benchmark (JOB) - PostgreSQL in 2026

Setup:

- On my Macbook: ./configure 'CFLAGS=-g -O3'
- Cold runs: restarted database & cleared system cache
- Warm runs: repeat run of the entire test suite
- Average of 3 runs per test


©EDB 2026 — ALL RIGHTS RESERVED.

# JOB Results - Total Execution Time (pg19devel)

| Configuration | Cold Runs (s) | Warm Runs (s) |
|---|---|---|
| Default settings | 140.75 ± 2.1 | 117.72 ± 0.8 |
| SET enable_nestloop = off | 87.16 ± 0.2 | 86.68 ± 0.4 |

When Nest Loop Joins are disabled:

Cold run: **38% faster than default**

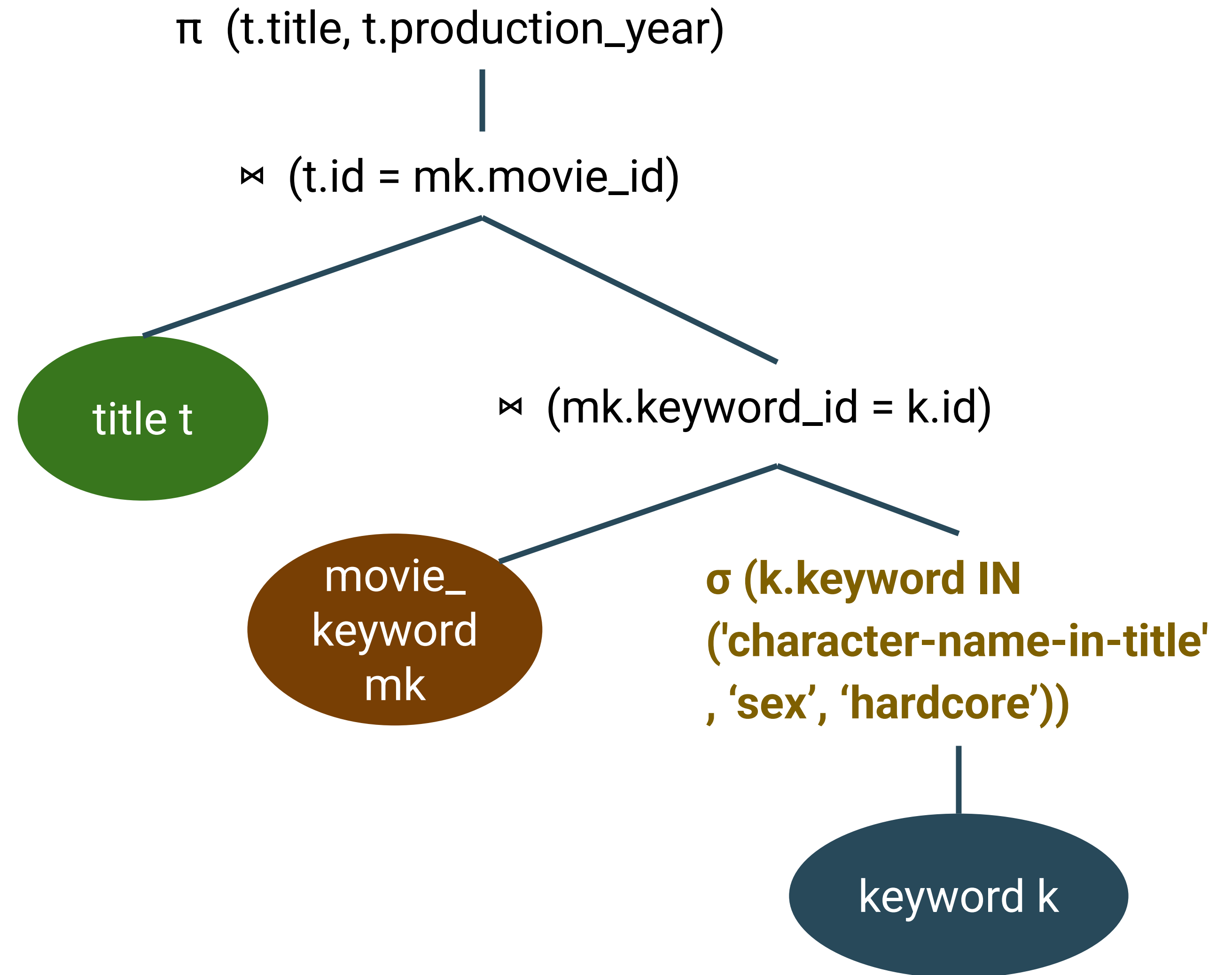
Warm run: **26% faster than default**

EDB POSTGRES AI

©EDB 2026 — ALL RIGHTS RESERVED.

# A Minimal Example

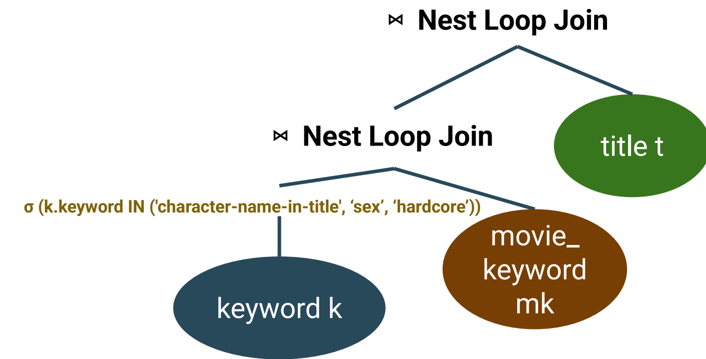
```
EXPLAIN (ANALYZE)
SELECT t.title, t.production_year
FROM keyword k
JOIN movie_keyword mk ON mk.keyword_id = k.id
JOIN title t ON t.id = mk.movie_id
WHERE k.keyword IN ('character-name-in-title',
                    'sex',
                    'hardcore');
```






©EDB 2026 — ALL RIGHTS RESERVED.

# EXPLAIN ANALYZE - Default Setting




```
Nested Loop  (rows=101) (actual … rows=172784.00 loops=1)
  -> Nested Loop  (rows=101) (actual …rows=172784.00 loops=1)
       -> Seq Scan on keyword k (rows=3) (actual … rows=3)
            Filter: (keyword = ANY ('{character-name-in-title,sex,hardcore}'::text[]))
       -> Bitmap Heap Scan on movie_keyword mk (rows=307) (actual … rows=57594.67 loops=3)
            -> Bitmap Index Scan on keyword_id_movie_keyword
                 Index Cond: (keyword_id = k.id)
  -> Index Scan using title_pkey on title t  (rows=1) (actual … rows=1.00 loops=172784)
       Index Cond: (id = mk.movie_id)
Planning Time: 11.314 ms
Execution Time: 1340.905 ms
```

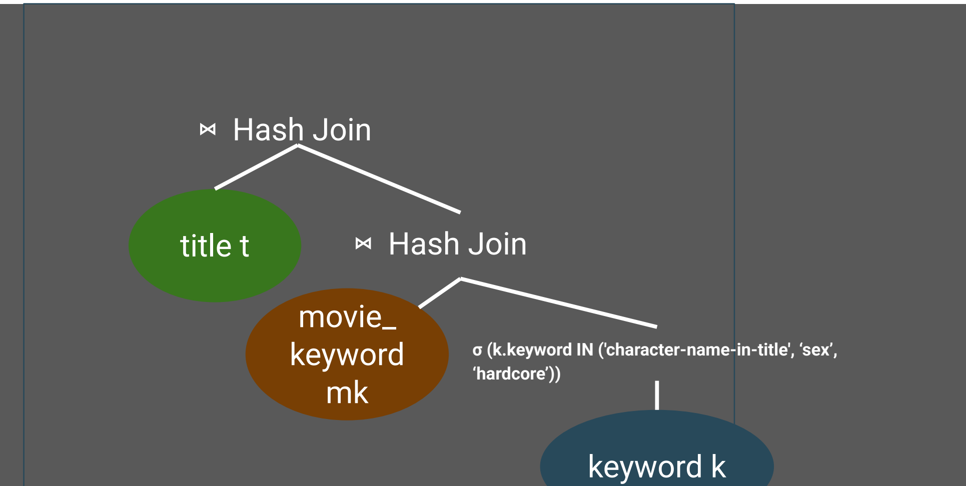
EDB POSTGRES AI

*Omitted irrelevant details*

©EDB 2026 — ALL RIGHTS RESERVED.

# EXPLAIN ANALYZE - Set enable_nestloop = off

**Hash Join** (rows=101) (actual rows=172784.00)
  Hash Cond: (t.id = mk.movie_id)
  **-> Seq Scan** on title t
  **-> Hash** (rows=101) (actual rows=172784.00)
    **-> Hash Join** (**rows=101)** (**actual rows=172784.00**)
      Hash Cond: (mk.keyword_id = k.id)
      **-> Seq Scan** on movie_keyword mk
      **-> Hash**
        -> Seq Scan on keyword k (rows=3) (actual rows=3.00 loops=1)
          Filter: (keyword = ANY ('{character-name-in-title,sex,hardcore}'::text[]))
Planning Time: 0.435 ms
**Execution Time: 581.273 ms**




**Omitted irrelevant details*
**Also SET max_parallel_workers_per_gather = 0; to make plan more readable and easier to compare*

EDB POSTGRES AI

©EDB 2026 — ALL RIGHTS RESERVED.

# Execution Time

| | Planner's Estimate | Reality |
|---|---|---|
| Rows | 307 | 172784 |
| Plan Chosen | 2 Nest Loop Joins | 2 Hash Joins |
| Executions Time | **1,341 ms** | **581 ms**<br>**(252 ms when parallel)** |


©EDB 2026 — ALL RIGHTS RESERVED.

# The Discussion


©EDB 2026 — ALL RIGHTS RESERVED.

# An Ongoing Discussion on Join Statistics

**Is there value in having optimizer stats for joins/foreignkeys?**


©EDB 2026 — ALL RIGHTS RESERVED.

# Current Consensus and Open Questions

**Emerging direction (not final):**

- Extending the **CREATE STATISTICS** syntax to joins
- **User-declared, not automatic** — no auto-creation for all FKs
- **Efficient sampling** (Tomas Vondra, citing Leis et al. 2017)
- **N-way join** support in catalog design, even if v1 is 2-way only (Tomas Vondra)
- Narrow v1 scope, extensible later

**What is NOT yet settled:**

- **N-way utility: multiple 2-way stats sufficient**? Or 3+ table correlations common?
- **VIEW-based stats as alternative syntax** (Andrei Lepikhov and Corey Huinker proposed)
- **Stat types beyond MCV**: ndistinct/functional dependency, and even histograms can also be useful for aggregates on join results, but probably not must have for v1


©EDB 2026 — ALL RIGHTS RESERVED.

# Current Patch - Do What Works

https://commitfest.postgresql.org/patch/6724/



©EDB 2026 — ALL RIGHTS RESERVED.

# Proposed Syntax for MCV Stats

```
CREATE STATISTICS [ [ IF NOT EXISTS ] statistics_name ]
    [ ( mcv ) ]
    ON { table_name1.column_name1 [, { table_name1.column_name2 } [, ...]]
    FROM table_name1 JOIN table_name2 ON table_name1.column_name3 = table_name2.column_name4
```

**Example:**

```
-- Create cross-table MCV statistics on a single filter column (keyword)
CREATE STATISTICS movie_keywords2_keyword_stats (mcv)
ON k.keyword
FROM movie_keywords2 mk JOIN keywords2 k ON (mk.keyword_id = k.id);
ANALYZE movie_keywords2;
-- Create cross-table MCV statistics on multiple filter columns (keyword + phonetic_code)
CREATE STATISTICS movie_keywords2_multi_stats (mcv)
ON k.keyword, k.phonetic_code
FROM movie_keywords2 mk JOIN keywords2 k ON (mk.keyword_id = k.id);
ANALYZE movie_keywords2;
```


©EDB 2026 — ALL RIGHTS RESERVED.

# Catalog Design - N-way Ready

**Relevant columns in pg_statistic_ext**

| Name | Type | Description |
|---|---|---|
| stxrelid (existing) | oid | **The first table ("anchor" table)** in CREATE STATISTICS |
| stxkeys (existing) | int2vector | The attnums of the statistics objects |
| … | | |
| ***stxjoinrels (new)*** | oidvector | The "non-anchor" tables who participate in joins |
| ***stxkeyrefs (new)*** | int2vector | Which *stxkeys* belong to which *stxjoinrels* relation; same size as *stxkeys* |
| ***stxjoinconds (new)*** | pg_node_tree | Join conditions as List of OpExpr<br>• varnos are synthetic: 1 = stxrelid, 2 = stxjoinrels[0]<br>• Example "ON (mk.keyword_id = k.id)" stored as "OpExpr(varno=1, attnum=2) = (varno-2, attnum=1)" |

**Existing pg_statistic_ext_data Column**

| Name | Type | Description |
|---|---|---|
| `stxdmcv` | `bytea` | MCV payload (existing field) |

EDB POSTGRES AI

©EDB 2026 — ALL RIGHTS RESERVED.

# Index-base Sampling

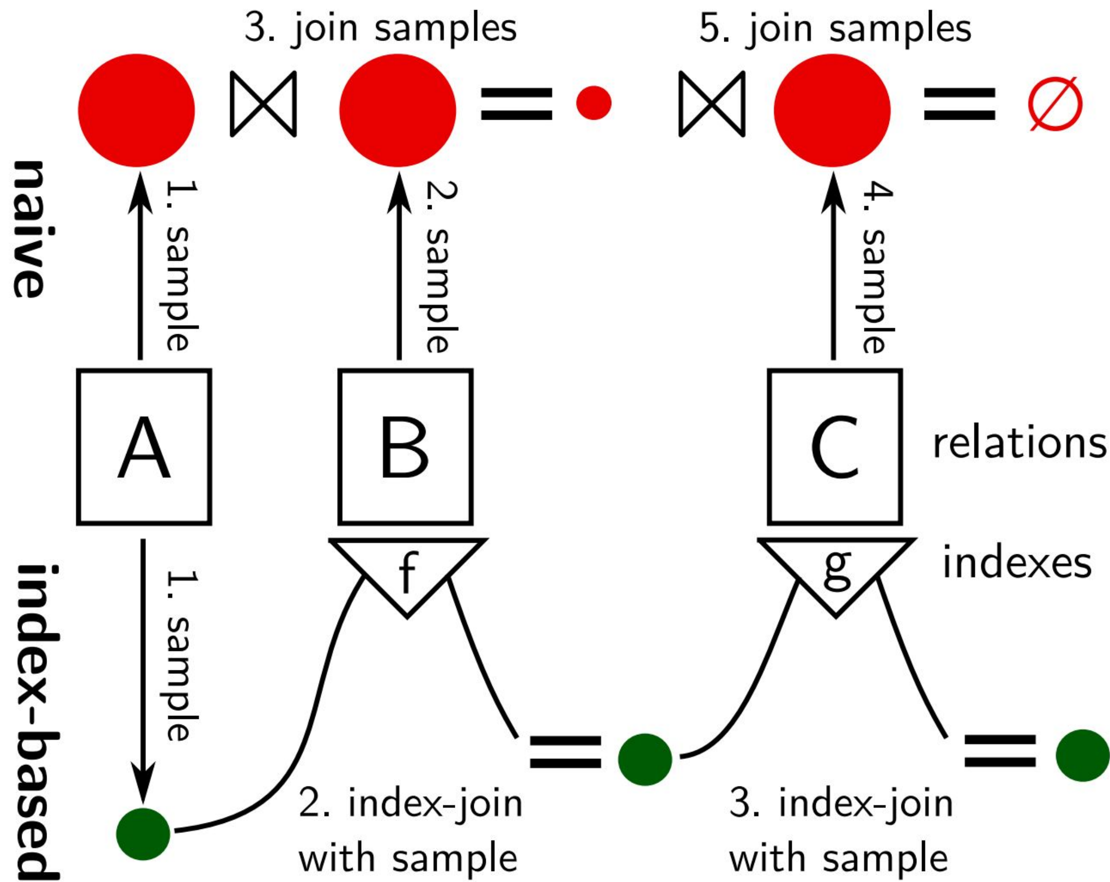



**Figure 1: Naive (top) vs. index-based (bottom) join sampling**

**Cardinality Estimation Done Right: Index-Based Join Sampling**


Viktor Leis, Bernhard Radke, Andrey Gubichev[†]*, Alfons Kemper, Thomas Neumann
Technische Universität München
{leis,radke,kemper,neumann}@in.tum.de
Google[†]
gubichev@google.com[†]

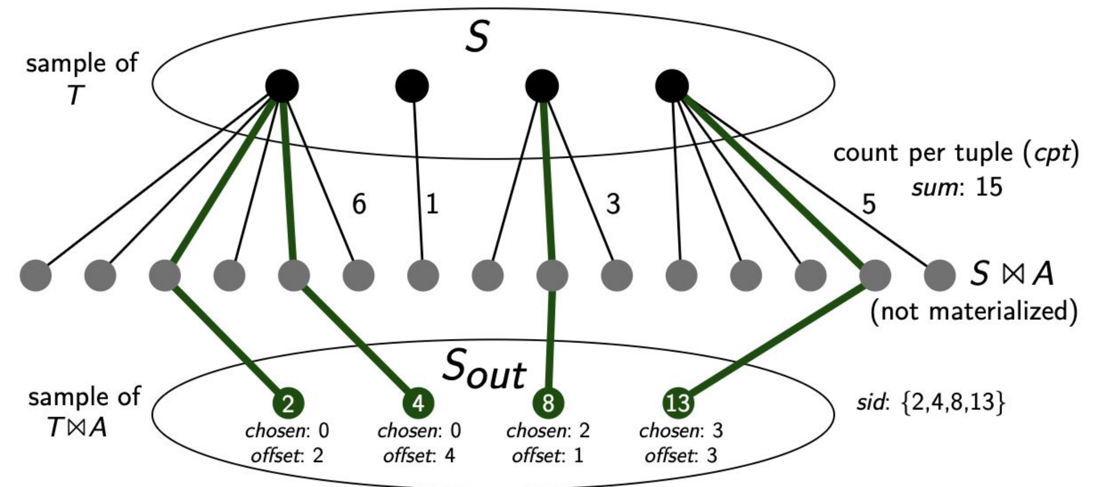




**Figure 2: Illustration of Algorithm 1. Using a sample $S$, which consists of 4 tuples, and a suitable index, the algorithm first counts the number of join partners for each tuple in $S$ (6, 1, 3, and 5). To compute the final sample, it then draws 4 random tuples (without replacement) from these 15 candidates**


©EDB 2026 — ALL RIGHTS RESERVED.

# Index-base Sampling

- Currently supports ***2-way Only***
- Currently triggered by ***ANALYZE the “anchor” table***
- **Enforce Index** at statistics creation time
- Currently does ***not support composite index*** on multiple columns


©EDB 2026 — ALL RIGHTS RESERVED.

# Stats Matching

Entry point:

- *get_foreign_key_join_selectivity()*
- *statext_join_mcv_clauselist_selectivity()*

Bare minimum matching:

- **Equality inner join** only
- **Single-condition** between two tables only, **can extend to multiple conditions per table pair**, need to support composite index probing first.
- Only ***"col = const"*** and ***"col IN (...)"*** filter predicates recognized.


©EDB 2026 — ALL RIGHTS RESERVED.

# Using the Matched Stat (MCV List)

```
// For each MCV item (value, frequency):
IF match(value, WHERE_filters):
  frequency_sum += item.freq
matched_sel = frequency_sum
non_mcv_sel = (1 - mcv_total_freq) / (ndistinct - mcv_nvalues)

raw_sel = matched_sel + (unmatched * non_mcv_sel)


adjusted_sel = raw_sel / (anchor_sel * other_tuples * other_sel)
Divides out filters already applied to rel rows to avoid double-counting.
```

**Safety Check:** ***If join MCV estimate < standard eqjoinsel, fall back to the default estimate.***

EDB POSTGRES AI

©EDB 2026 — ALL RIGHTS RESERVED.

# Performance


©EDB 2026 — ALL RIGHTS RESERVED.

# The Join Order Benchmark (JOB)

**Setup:**

- On my Macbook: ./configure 'CFLAGS=-g -O3'
- Cold runs: restarted database & cleared system cache
- Warm runs: repeat run of the entire test suite
- Average of 3 runs per test
- Add a single join MCV statistics object:

```
CREATE STATISTICS movie_keyword_keyword_stats (mcv)
ON k.keyword
FROM movie_keyword mk
JOIN keyword k ON (mk.keyword_id = k.id);
```

- **To avoid sampling noises, use the same database for the "with stats" vs "without stats" runs**:
  - CREATE STATISTICS + ANALYZE once, benchmark with stats, then DROP STATISTICS and benchmark without.


©EDB 2026 — ALL RIGHTS RESERVED.

# Cardinality Estimates

32 queries where the *keyword/movie_keyword* join appears in both plans with the same actual row count.

***q-error = max(estimated/actual, actual/estimated). Lower is better.***

| | Without Stats | With Stats |
|---|---|---|
| Geometric mean | 103.4x | 4.2x |
| Median | 131.7x | 2.4x |
| 90th percentile | **1,230.6x** | **29.6x** |

- 16 improved, 0 regressed, 16 unchanged
- Unchanged queries have filters the MCV doesn't cover:
  LIKE patterns, no filter on keyword, or rare values outside MCV

EDB POSTGRES AI

©EDB 2026 — ALL RIGHTS RESERVED.

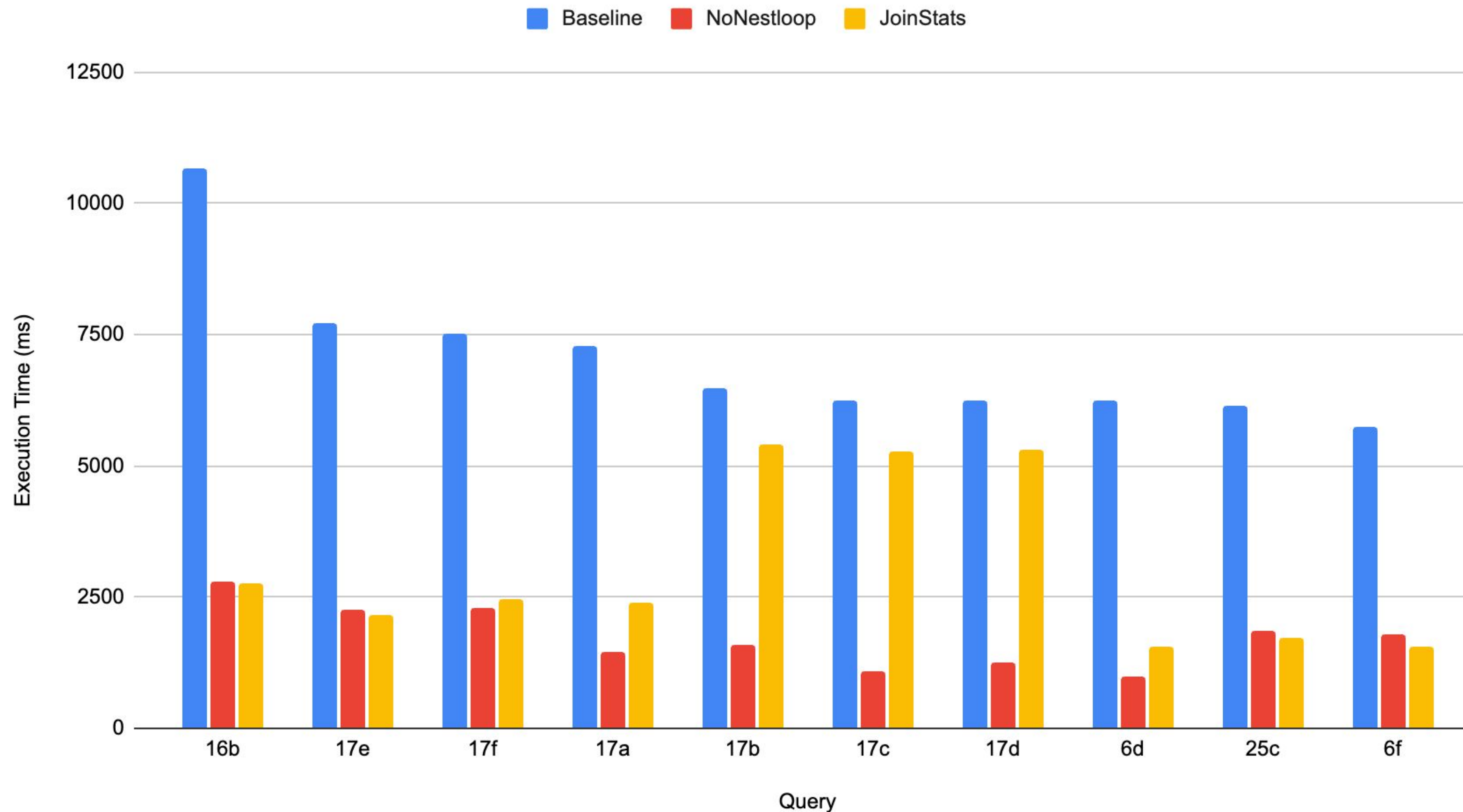
# Total Execution Time

## With a Single Join Statistic Object:

| Configuration | Without Stats | With Stats | **Speedup** |
|---|---|---|---|
| Warm | 114.1s | 74.0s | **1.54x** |
| Cold | 141.7s | 103.7s | **1.37x** |


©EDB 2026 — ALL RIGHTS RESERVED.

# Per-query Execution Times (from an older run)




EDB POSTGRES

©EDB 2026 — ALL RIGHTS RESERVED.

# ANALYZE Costs

| Configuration | ANALYZE time | Overhead |
| --- | --- | --- |
| Baseline (No Join Stats) | 266ms | - |
| 1 join stat (mk -> (keyword, 135K rows)) | 487ms | +221ms (83%) |

EDB POSTGRES AI

©EDB 2026 — ALL RIGHTS RESERVED.

# Alternatives, Next Steps, and Open Questions


©EDB 2026 — ALL RIGHTS RESERVED.

# Three Independent Design Challenges for Join Statistics

### Part 1: Syntax & Storage

How to **DEFINE** and **STORE**?

### Part 2: Collection

How and when to **COLLECT**?

### Part 3: Query Exploitation

How does the planner **FIND** and **USE** the right stat at plan time?


©EDB 2026 — ALL RIGHTS RESERVED.

# Syntax & Storage



©EDB 2026 — ALL RIGHTS RESERVED.

# Statistical Views (DB2 Model)

```
-- SQL Server:
CREATE VIEW sales_product_v AS
  SELECT s.sale_id, s.amount, p.category, p.price
  FROM sales s JOIN products p ON (s.product_id = p.product_id)
  WHERE s.status = 'completed';

-- Mark the view as statistical view
ALTER VIEW sales_product_v ENABLE QUERY OPTIMIZATION;

-- Collect Statistics
RUNSTATS ON TABLE schema.sales_product_v WITH DISTRIBUTION;
```

[DB2 Statistical Views]



©EDB 2026 — ALL RIGHTS RESERVED.

# Catalog Design

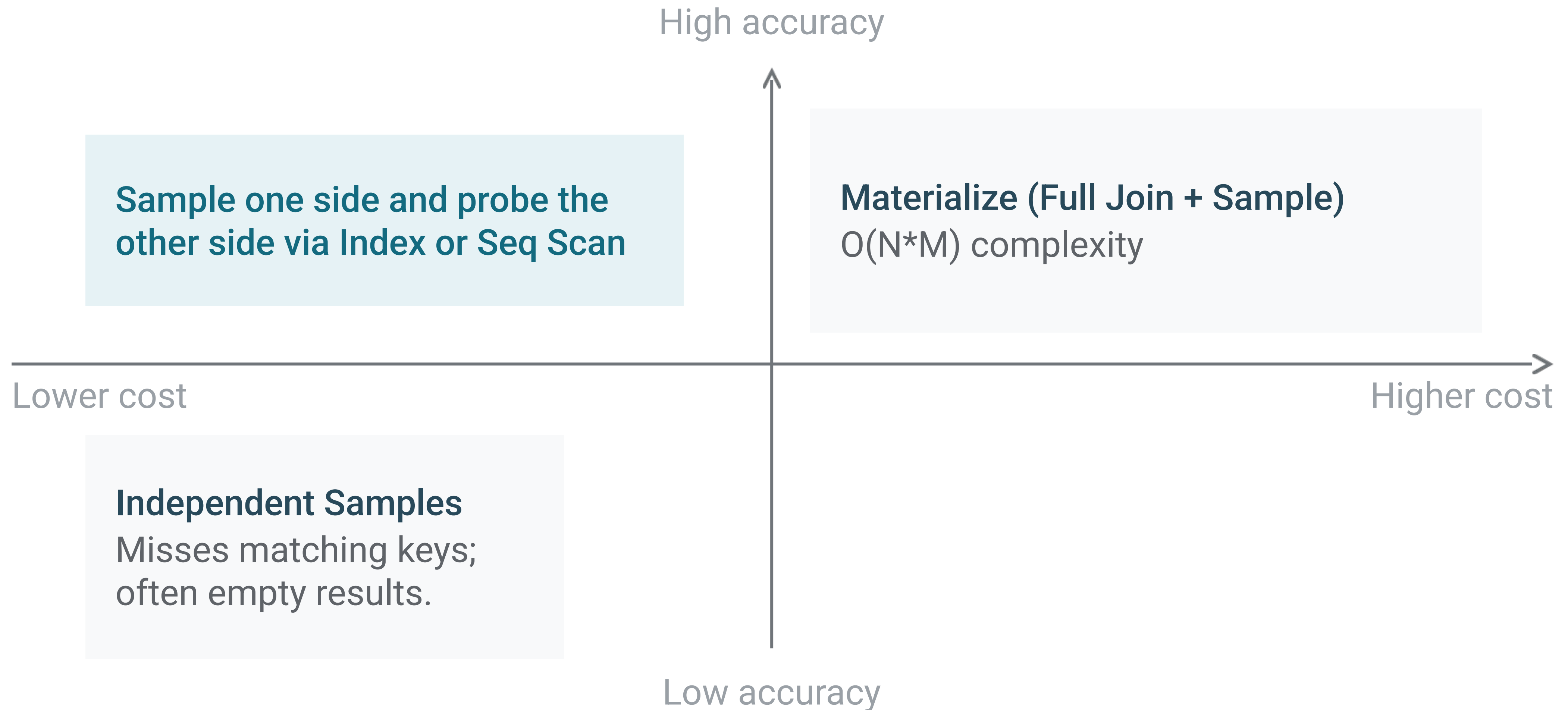
- Stats stored in ***pg_statistic*** where *starelid* is the OID for the VIEW
- Add a boolean field in ***pg_class*** for the RELATION who is associated with an STATISTICAL VIEW.


©EDB 2026 — ALL RIGHTS RESERVED.

# Statistics Collection


©EDB 2026 — ALL RIGHTS RESERVED.

# Ways to Collect Join Statistics




EDB POSTGRES AI

©EDB 2026 — ALL RIGHTS RESERVED.

# No Index? Join the Sample Directly (Not yet implemented)

- Take sample **S**, join it against the full other relation **A**
- Hash the small side (**S is always tiny** → fits in memory, hash is free)
- Scan **A** sequentially, probe hash for matches
- Cost: **O(|A|)** sequential I/O
- No index required — works for any equijoin
- Produces: same unbiased uniform sample

EDB POSTGRES AI

©EDB 2026 — ALL RIGHTS RESERVED.

# N-Way Joins (Not implemented yet)

Enumerate

- Stores intermediate results
- Results grow exponentially
- Set time or lookup budget

Don’t Enumerate

- Use current query order
- Follow Constraint/Index graph


©EDB 2026 — ALL RIGHTS RESERVED.

# Query Exploitation



©EDB 2026 — ALL RIGHTS RESERVED.

# Subexpression Matching

**Finding the right stat for a query is a restricted form of view matching.**

$Wq$ = query's full WHERE clause (including join clause)
$Wv$ = stat's defining predicate

where $W$ is decomposed into three logical parts:

$W = PE \cap PR \cap PU$

- Simpler if conservative syntactic matching only
- More complex if supports self-joins (NP-complete)
- More complex if supports Aggregates

**Row Containment Tests:**

| Test | Description |
|---|---|
| **Equijoin Subsumption ($PE$)** | Every column equality in the statistic must also hold in the query.<br>e.g., **R.x = S.y** |
| **Range Subsumption ($PR$)** | The stat's range predicates must be at least as wide as the query's.<br>e.g., **R.price > 100** |
| **Residual Subsumption ($PU$)** | Every remaining conjunct in the stat matches one in the query.<br>e.g., **R.name LIKE '%abc%',**<br>**R.a + S.b > 10** |

EDB POSTGRES AI

©EDB 2026 — ALL RIGHTS RESERVED.

# Deployed Expression Matching Solutions

| System | Feature | Scope |
|---|---|---|
| **SQL Server** | Index views | Inner equi-joins, no self-joins |
| **Snowflake** | Materialized views | Single-table only |
| **Redshift** | Materialized views | Inner equi-joins, limited agg |
| **DB2** | Statistical views | Unclear |

A good starting point: **Inner Equality Join**

EDB POSTGRES AI

©EDB 2026 — ALL RIGHTS RESERVED.

# What Else Can Benefit from Expression Matching

- Materialized View Rewrite
- Query Plan Caching
- Query Result Caching
- Training an AI model for Statistics Estimate or Query Plan?


©EDB 2026 — ALL RIGHTS RESERVED.

# “Island of Accuracy” - Accurate Locally, Worse Globally?

Consider a 6-way join query:

A —— B —— C —— D —— E —— F

We have accurate stats HERE
(sel = 0.05 vs default 0.001)

└—— {A,B} is now 50X bigger than default estimate

But then:
C joins {A,B}: C has even stronger correlation with A, but we didn’t define that join stats

Will the more accurate selectivity for {A,B} help or hurt the final plan?


©EDB 2026 — ALL RIGHTS RESERVED.

# References

- Hacker’s Discussion: Is there value in having optimizer stats for joins/foreignkeys?
- Comitfest Entry: Add join MCV statistics for selectivity estimation
- Join Order Benchmark Repo: l-wang/join-order-benchmark
- Leis et al., "How Good Are Query Optimizers, Really?" PVLDB 2015
- Leis et al., "Cardinality Estimation Done Right: Index-Based Join Sampling" (CIDR 2017)
- Goldstein & Larson, “Optimizing queries using materialized views: a practical, scalable solution”, SIGMOD, 2001
- DB2 Statistical Views
- SQL Server Index Views
- Snowflake Materialized Views
- Redshift Materialized Views
- Relavent Talks:
  - Corey Huinker’s Talk on Join Statistics
  - Helping Planner Help You

EDB POSTGRES AI

©EDB 2026 — ALL RIGHTS RESERVED.

# Thank You


©EDB 2026 — ALL RIGHTS RESERVED.