What Works and What Does Not: A Winning Strategy for Join Query Execution

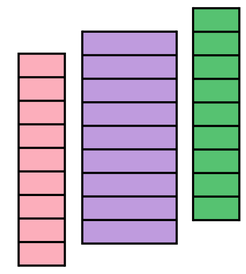
Altan Birler, Tobias Schmidt, Stefan Lehner, Florian Drescher, Maximilian Rieger, Simon Ellmann, Maximilian Reif, Adrian Riedl
Team (No) SortMergeJoins

Goal
Execute a join query on columnar data as fast as possible.

Challenge
- α-acyclic equi-joins, PostgreSQL query plans
- Every query processes fresh input data, no statistics
- Bespoke input format
- Variable length text data
- IMDB dataset small for modern machines, overheads matter

```
R1.X = R2.X
R2.y = R4.y
R3.Z = R5.Z
R4.K = R5.K
```

Query
predicates & joins



Columnar Layout
integer & text

Approach

Keep it simple, keep it fast

Planning
- Statistics based on index-based join sampling [Leis et al. 2017]
- DP based join ordering [Moerkotte & Neumann 2006]
- Incremental query planing [Neumann & Galindo-Legaria 2013]

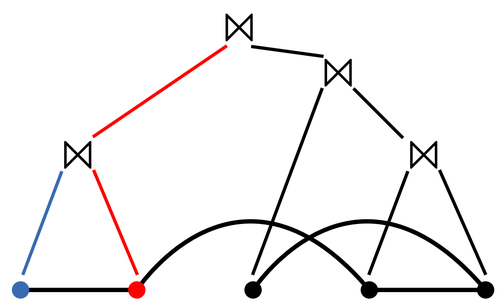
Table scan
- Fast scans and filtering using bitmaps and vectorization
- Bloom filters [Birler et al. 2024, Schmidt et al. 2021]

Join pipeline
- (Pre-)Compiled join pipelines [Neumann 2011]
- Chaining hash table with partitioned loads [Birler et al. 2024]
- Eager aggregation of duplicates [Birler et al. 2024]

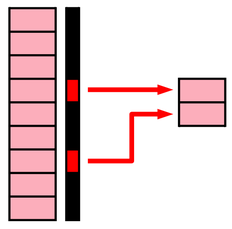
Infrastructure
- Bump memory allocator
- Efficient scheduling of small and large tasks
- Continuous profiling with Perfetto
- Random test query generation for robustness

Source Code:
github.com/umbra-db/contest-sigmod2025

(1) Build Query Graph
predicates as a graph, relations as nodes



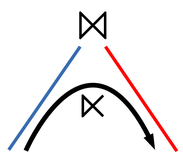
(2) Optimize Join Plan
pick the next cheapest pipeline



(3) Vectorized Table Scan
filter early, read and process less

```
for tuple in table:
    if not tuple.key in probeHt:
        continue
    for partner in probeHt[tuple.key]:
        targetHt.insert(tuple + partner)
```

(4) Run Join Pipeline
probe joins and build next hash table

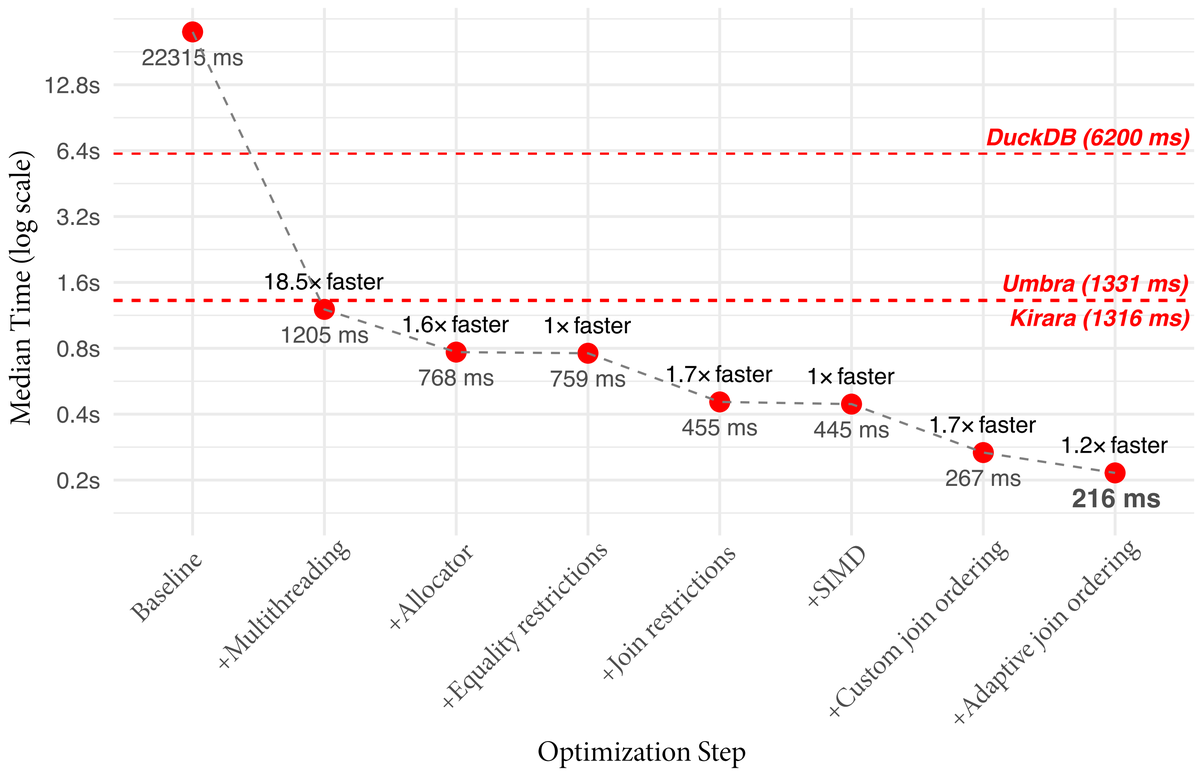


(5) Build Join Filters
prepare for future scans

(6) Repeat
update statistics & continue with (1)

Evaluation

Execution Time Improvement with Optimizations
Competitor performance shown as reference lines
Each optimization step shows speedup vs. previous



3 repeated executions of JOB queries on prefiltered base tables measured on AMD EPYC 9454P (Linux 6.11.0-26).

Findings
- The general-purpose database system Umbra is roughly as fast as the second-best solution, even when query-compilation and data-decompression times are included.
- A simple yet efficient **hash table** combined with **compiled join pipelines** already provides a strong baseline.
- The execution is memory bound. **Early filtering** helps as reading less data makes us faster. SIMD does not make a difference as we are not compute bound.
- PostgreSQL query plans are okay but not great. **Adaptive query optimization** improves runtime by 2×.