

DuckDB Coordinate Lattice

Applying the tagma coordinate-lattice principle to multi-dimensional point and range queries in DuckDB

Taeho Lee*
SSCCS Initiative
ssccs.org

August, 2026

1 Abstract

DuckDB multi-dimensional point and range queries over bounded integer dimensions run as full table scans. The scan path restricts index scans to one filter and a single-column ART, so a query such as $d1 = x$ AND $d2 = y$ AND $d3 = z$ always scans the table. The tagma coordinate lattice addresses the matching subspace directly: for bounded dimensions the table is a regular lattice, and the physical rows of a point or range query are a closed form of the coordinates.

The work followed the measured discipline: a profile gate before implementation, a prototype against the scan and a standard index, then the verdict. The gate passes, the lattice prototype is implemented and verified on the fork branch 54-duckdb-lattice, and the verdict is a measured win at point-query volume.

2 The measured pattern

Precedent	Mechanism	Result	Relevance to DuckDB
CERN ROOT-Coord	Per-read index traversal replaced by the closed form	69 to 181x	Many small reads with per-item metadata
ExaVerif structural	Full-space enumeration replaced by valid subspace enumeration	about 50x	Bounded dimensions address the matching subspace directly
vLLM and llama.cpp KV	Block-table and cell-index indirection removal	parity	The indirection was a negligible share of the compute

The DuckDB claim is the ExaVerif pattern applied to queries: for a table with bounded integer dimensions, a point or range query addresses the matching rows as a lattice subspace instead of scanning. The comparison included the standard index, not only the scan.

*Founder & Architect; Corresponding author: lee@ssccs.org

3 The outcome

The profile gate confirmed the gap: every multi-filter query scans all rows even with indexes on all three dimensions, and the ART index never activates on the benchmark lattice under the default threshold. The lattice prototype, the `lattice_scan` table function, addresses the matching subspace in closed form and is verified against the regular scan on point, range, full-lattice, and partial-dimension queries.

The A/B measured the scaling law with the release build:

- The lattice per-query point cost is flat in the table size, 0.31 to 0.36 ms from 9.9M to 600M rows.
- The scan grows linearly with the table size, so the ratio grows with N: 1.54x at 9.9M rows to 11.3x at 600M rows.
- The verdict is a measured win at point-query volume. Wide queries are bounded by the table function row-emission cost.

The comparison baseline is the zonemap-pruned scan on the lattice-sorted table (2 of 4,876 row groups for the point query at 600M), and the win mechanism is skipping the row-group metadata traversal. The cold-cache I/O case is measured and refuted for point queries (2.7x cold against 16x warm at 600M). The detailed plan, the code anchors, the charts, and the corrected insights live in the companion development plan page </works/duckdb/lattice/>. The work lives on the fork branches `54-duckdb-lattice` and `55-join-probe` in [ssccsorg/duckdb](https://github.com/ssccsorg/duckdb).

4 Follow-up tracks

The track produced two follow-up results on the same fork.

The join layer ([#55](https://github.com/ssccsorg/syntaxma)) generalized the perfect hash join to D equality conditions with a mixed-radix fold on the branch `55-join-probe`. The implementation is correct (the PHJ suite and `verify_phj.sh` pass), and the measured advantage is limited: 1.33x on the probe-dominated 3D workload, 0.8x loss on the build-dominated dense self join. The branch is merged through pull request #2 in [ssccsorg/duckdb](https://github.com/ssccsorg/duckdb).

The multi-filter index scan ([#56](https://github.com/ssccsorg/syntaxma)) implemented the `TableScanInitGlobal` FIXME on the branch `56-duckdb-multi-filter-index`: scan one single-column ART per filtered column and intersect the row-ID sets. Measured on a 100M-row random-order table where the scan cannot prune: 3.7x warm count, 4.9x warm payload, about 4.5x cold (first cold sample 8.6x). The same feature is implemented more completely by upstream [duckdb/duckdb](https://github.com/duckdb/duckdb) pull request #24942, so the fork branch stands as the measurement record, not an upstream contribution. The implementation also fixed a latent binding bug in `TryScanIndex` that prevented index scans on non-first columns.

5 Status

Phase 1 (profile gate), Phase 2 (lattice prototype), and Phase 3 (verdict) are complete, with the follow-up tracks #55 and #56 recorded above. The measured win at point-query volume is recorded, with the scaling law confirmed across four table sizes against the zonemap-pruned scan baseline, the cold-cache I/O case is measured and refuted for point queries, and the multi-filter index scan measured 3.7 to 4.9x on an unprunable table. The strategic direction is the unprunable and traversal-dominated domains, with a profile gate before any new target. The syntaxma core upgrade remains re-scoped to general dimensions, wide-result row emission, and virtual lattice columns.