CSCI 543 Foundations of Modern Data Management
Project - 2 Proposal

Learned Indexes in DuckDB

Group Info

- Siddharth Raj Dash (srdash@usc.edu)
- Mayur Prasanna (mayurpra@usc.edu)
- Venkataraman Venkatasubramaniam (vv70613@usc.edu)

Project Idea

DuckDB currently supports two indexing mechanisms: the Min-Max Index and the Adaptive Radix Tree (ART). The Min-Max Index enables zone-map pruning by first dividing a table into segments and then storing per-segment minimum and maximum values, allowing chunks that fall out of this range to be skipped during scans. It is lightweight and efficient for analytical workloads, but for large segments and highly selective predicates (like equality or narrow range filters), the scan still performs large vectorized scans on remaining data.

The Adaptive Radix Tree (ART) provides a secondary index for direct lookups, which is automatically built for PRIMARY KEY, UNIQUE or FOREIGN KEY columns, or created manually via CREATE INDEX. ART supports point and prefix queries but incurs high pointer-chasing overhead and memory cost, making it less suitable for large, ordered analytical tables.

Between coarse-grained Min-Max pruning and fine-grained ART traversal, there exists a gap for workloads over sorted, clustered, or locally monotonic columns where tuple positions can be predicted rather than searched. To bridge this, we propose integrating a Recursive Model Index (RMI) as a lightweight, learned index that uses a statistical model to map key values to physical positions. Instead of navigating a tree, RMI predicts where a value lies, turning index lookups into arithmetic operations. This design reduces memory overhead, cache misses, and traversal latency, extending DuckDB's indexing system with a third, model-driven alternative optimized for large, ordered analytical workloads.

Proposed Solution

We will integrate the Recursive Model Index as a new index type in DuckDB, accessible via

```
CREATE INDEX idx_name on table_name(col_name) USING rmi
```

The RMI learns a compact statistical model that predicts the approximate physical position of a key in a column's sorted order. Within this predicted confidence window, a short correction probe is made ensuring accuracy while keeping lookup time nearly constant.

The following is a list of non-exhaustive changes we have identified across the DuckDB components to achieve the integration of this learned index.

- **New Index Type:** Add an RMIIndex class within DuckDB's index framework.
- **Model Building:** Train the model at index creation using sorted column values.

- **Storage Integration:** Store model parameters and error bounds as metadata.
- **Optimizer Support:** Extend cost estimation so the planner can choose RMI when cheaper than ART or scans.
- **Execution Path:** Modify index-scan operator to invoke RMI for position prediction.

We plan to implement and evaluate the learned indexes across the following three lightweight models:

- Linear Regression Model (single model per indexed column)
- Two-level RMI (a root-level router with child-level linear models)
- Piecewise Linear Spline

Proposed Evaluation

Since the RMI system uses predictions to find ranges of queries, we will need to evaluate the system on two fronts.

1. Index range prediction evaluation
    a. We plan to test how well the learned models predict tuple positions on sorted data. This phase is exclusively to test the performance of the model in giving us accurate tuples within a range.
    b. Datasets: We plan to curate synthetic workloads or utilize benchmarking datasets (IMDB/JOB) for validating the model accuracy and to test the performance in a larger workload.
    c. Metrics: Prediction accuracy, fallback rate, index build time (model learning time).

2. Database Performance Evaluation
    a. We test the database performance as a whole with and without the RMI index for read-only analytical workloads.
    b. Datasets: Subsets of TPC-H and TPC-DS queries featuring selective ranges on sorted columns and point lookup read-only queries (OLAP workloads).
    c. Metrics: Query latency, rows scanned, index/scan time, memory footprint.

Timeline

- **Week 1** - Design
    - Set up DuckDB development environment, understand DuckDB index and planner internals, finalize RMI design, model choices and evaluation plan.
- **Week 2** - Index Integration
    - Implement RMI-Index class, enabling the create-index command, build model-training flow on sorted columns and store model metadata.
- **Week 3** - Query Integration
    - Implement lookup path (prediction, bounded search, fallback), extend index-scan operators and integrate RMI into optimizer cost-model.
- **Week 4** - Evaluation
    - Run model-level and end-to-end performance tests using IMDB/JOB and TPC-H/DS subsets; measure accuracy, latency, and resource usage.
- **Week 5** – Verification
    - Refine implementation (based on experiments), analyze and summarize the tests, and compile a final report with structured results and performance comparisons.