BLAWK_KDB MASTER BLUEPRINT

Generated on: 2026-02-17T08:54:41.224754 UTC

This document provides a complete architectural, performance, execution, migration, and implementation blueprint for building a kdb-like high-performance columnar engine named blawk_kdb.

1. Vision & Design Philosophy

blawk_kdb is a columnar, vectorized, mmap-capable analytical engine designed for time-series and macro-financial workloads. It combines the pipeline ergonomics of blawk with the execution discipline of kdb+/q.

Core goals:

- Typed columnar storage (no generic matrix abstraction).

- Fused vector execution (single-pass kernels).

- Minimal allocation strategy.

- mmap CSV ingestion for large datasets.

- Rolling window and Ft-measurable time-series semantics.

- Minimal q-like query layer.

2. Repository Layout

```
blawk_kdb/
  src/
    main.rs
    table/
      mod.rs
      table.rs
      col.rs
      index.rs
    io/
      csv_read.rs
      csv_write.rs
    expr/
      ast.rs
      types.rs
      ir.rs
      lower.rs
    exec/
      plan.rs
      kernels.rs
      runtime.rs
    builtins/
      math.rs
```

```
      rolling.rs
      combinatorics.rs
  tests/
  benches/
```

3. Core Data Model

Table Structure:

```
struct Table {
    schema: Schema,
    columns: Vec<Column>,
    row_count: usize,
}
```

Column Enum:

```
enum Column {
    F64(Vec<f64>, Option<Vec<u8>>),   // data + optional validity bitmap
    I64(Vec<i64>, Option<Vec<u8>>),
    Bool(Vec<u8>, Option<Vec<u8>>),
    Sym(Vec<u32>, Option<Vec<u8>>),
}
```

Null Handling Strategy:

- Initial phase: sentinel -99999 for backward compatibility.

- Final phase: validity bitmap (Arrow-style).

- Kernels must branch once per chunk, not per element when possible.

4. CSV Ingestion via mmap

Steps:

1. mmap file.

2. Split by newline boundaries.

3. Parse header.

4. Allocate column vectors with pre-estimated capacity.

5. Multi-thread chunk parsing.

6. Finalize into Table.

```
let mmap = unsafe { Mmap::map(&file)? };
let chunks = split_into_chunks(&mmap);
parallel_parse(chunks);
```

5. Execution Model

Pipeline:

AST -> Type Inference -> IR -> Logical Plan -> Physical Plan -> Execution Kernels

IR Node Examples:

```
enum IRNode {
    MapUnary { op: UnaryOp, input: ColRef },
    MapBinary { op: BinaryOp, left: ColRef, right: ColRef },
    Shift { input: ColRef, lag: usize },
    Rolling { op: RollingOp, input: ColRef, window: usize },
    Filter { mask: ColRef },
    Agg { op: AggOp, input: ColRef },
}
```

6. Kernel Fusion Strategy

Consecutive element-wise operations must be fused into a single tight loop.

```
for i in 1..n {
    let l1 = x[i].ln();
    let l0 = x[i-1].ln();
    out[i] = l1 - l0;
}
```

Avoid intermediate Vec allocations.

Fuse MapUnary + MapBinary patterns automatically during planning.

7. Rolling Window Implementation

All rolling windows must be Ft-measurable (use past data only).

```
for i in window..n {
    let slice = &x[i-window..i];
    out[i] = mean(slice);
}
```

Optimized rolling mean uses cumulative sums.

Rolling std uses Welford or two-pass cumulative formula.

All rolling kernels must support step replication logic.

8. Groupby & Aggregation

Use hash-based grouping for Sym and I64 keys.

Aggregation operators: sum, avg, min, max, count.

```
let mut groups: HashMap<Key, AggState> = HashMap::new();
```

9. CLI Interface

Pipeline Mode:

```
blawk_kdb read file.csv | dlog px 1 | wavg 20 | write out.csv
```

Query Mode:

```
blawk_kdb -q 't:read["file.csv"]; select sym,avg:px by sym from t'
```

10. Performance Engineering Rules

- Always column-major contiguous memory.

- Avoid heap allocation inside tight loops.

- SIMD-friendly loops (future AVX-512 optimization).

- Parallelize by column for map ops.

- Parallelize by row for groupby.

- Avoid materializing intermediate Tables.

11. Benchmark Targets

Baseline tests:

- 4.5M element log workload.

- 1000 columns x 10k rows rolling stats.

- Groupby on 5M rows.

blawk_kdb must exceed legacy blawk performance via fusion and typed columns.

12. Migration Plan

Phase 1: Implement typed Table and CSV reader.

Phase 2: Port math and rolling kernels.

Phase 3: Introduce IR and fusion.

Phase 4: Implement groupby and select/update parser.

Phase 5: Add benchmark harness and profiling.

13. Definition of Done

blawk_kdb is production-ready when:

- All legacy numeric ops match golden tests.

- Fusion eliminates intermediate allocations.

- Rolling stats are Ft-measurable and verified.

- CLI supports both pipeline and q-lite mode.

- Benchmarks demonstrate performance gains.