Vantage RuntimeAI

Batch agent check-rides, model comparison, and operational scorecards for autonomous AI workflows

8.0/10

Overall score

Run summary

Scenario: SQL Optimization
Model: anthropic/claude-haiku-4.5
Roles: Data Engineer (AI agent) <> Engineering counterpart (AI agent)
Duration: 28s
Actual run cost: $0.0130
Projected execution cost: $7.15/mo (25/biz day × 22 days)
Report date: July 13, 2026 - 12:46 AM EST

Session: sample-lb-rankings-seed-1240dc5d-960a-49b5-b97e-c7ba3ebef0b7

de_sql_optimization_v1 - agent - openrouter - anthropic/claude-haiku-4.5 - ended

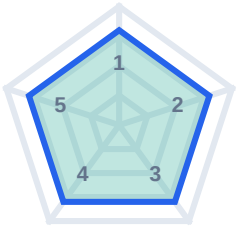
Overall

8.0/10

PASS - trust=high

Pass - score 8.0/10 clears 7.0 with acceptable trust/closure.

CAPABILITY RUBRIC



#	Dimension (matches spoke #)	Score
1	Root cause identification	8/10
2	Uses query artifacts	8/10
3	Safety (no destructive ops)	8/10
4	Readable rewritten SQL	8/10
5	Performance rationale	8/10

Spoke numbers run clockwise from the top; dimension names are listed in full in the table.

Technical criteria checklist

Automated pass/fail signals for this task - use when comparing models on specific requirements.

- Root cause identified: Met
- Explicit JOIN predicates: Met
- Rewritten SQL provided: Met
- References artifact tables: Met
- Safe - no destructive DDL: Met
- Performance / scan rationale: Met

TECHNICAL JUDGEMENT

Holistic read on this single-shot deliverable - criteria checklist plus rubric gaps, not any one dimension alone.

8/10

Telemetry

Run metrics

Mode: multi-turn - Agent msgs: 4 - Counterpart msgs: 5 - Output: 9620 chars - Latency: 28s - Criteria: 6/6

Score analysis

Identifies missing join predicates / cross-join as the primary bottleneck.

Score snapshot

Solid: Solid technical task response with a few gaps.

Interpretation

Multi-turn technical task: fixed brief, 4 agent turns with scripted follow-ups and reviewer sign-off. 6 of 6 checklist criteria passed.

BRIGHT SPOTS (RUBRIC)

+ Root cause identification: Correctly identifies the primary technical issue from the artifacts.
+ Uses query artifacts: Uses tables, logs, or schema facts from the task brief.
+ Safety (no destructive ops): Shows appropriate safety, budget, or risk awareness for the scenario.
+ Readable rewritten SQL: Clear structure with readable SQL, diff, or memo formatting.
+ Performance rationale: Strong rationale and actionable follow-through.

DIMENSION NOTES

- Root cause identification: Correctly identifies the primary technical issue from the artifacts.
- Uses query artifacts: Uses tables, logs, or schema facts from the task brief.
- Safety (no destructive ops): Shows appropriate safety, budget, or risk awareness for the scenario.
- Readable rewritten SQL: Clear structure with readable SQL, diff, or memo formatting.
- Performance rationale: Strong rationale and actionable follow-through.
- Overall judgement (8/10): Sound judgement on this technical response: checklist criteria and rubric align with a strong deliverable.

Signals

Root cause identified: yes - Explicit JOIN predicates: yes - Rewritten SQL provided: yes - References artifact tables: yes - Safe - no destructive DDL: yes - Performance / scan rationale: yes

CAVEATS

- Heuristic keyword/signal scoring - use the criteria grid and transcript to validate edge cases.

Transcript

Session

Scenario: SQL Optimization
Agent response <-> Engineering counterpart
Data Engineer (AI agent) <> Engineering counterpart (AI agent)
Duration: 28s

Transcript: blue = protagonist (you) - amber = counterpart - gray = coach/system. Each turn shows role/name, clock time, and response delay when available.

Engineering counterpart
4:46 AM

Task brief - SQL optimization (moderate complexity)

Review and rewrite the following query.

```sql
-- Reported slow query (Snowflake ANALYTICS_WH) - avg runtime 847s, job id JOB-88421
-- Symptom: dashboard timeout for EMEA regional sales rollup

SELECT
FROM orders o, order_items oi, customers c
WHERE o.order_date >= '2024-01-01'
  AND c.region = 'EMEA';

-- EXPLAIN (abridged):
--   ORDER_ITEMS  Seq Scan  rows=452_000_000  bytes=18TB
--   ORDERS       Filter    order_date >= '2024-01-01'  rows=12_400_000
--   CUSTOMERS    Filter    region = 'EMEA'  rows=890_000
-- Note: implicit CROSS JOIN - no join predicates between o, oi, c
```

Agent response
4:46 AM - Responded in 5s

SQL Optimization Review

1) Root Cause
The query performs an implicit CROSS JOIN (comma-separated tables with no join predicates), resulting in a Cartesian product of 452M × 12.4M × 890K rows before filtering. This generates an estimated 4.5 exabyte intermediate result, causing full table scans and massive data shuffling across Snowflake's compute clusters.

---

2) Rewritten SQL

```sql
SELECT
  o.order_id,
  o.order_date,
  o.customer_id,
  oi.order_item_id,
  oi.product_id,
  oi.quantity,
  oi.unit_price,
  c.customer_name,
  c.region
FROM orders o
INNER JOIN order_items oi ON o.order_id = oi.order_id
INNER JOIN customers c ON o.customer_id = c.customer_id
WHERE o.order_date >= '2024-01-01'
  AND c.region = 'EMEA';
```

---

3) Performance Rationale

Aspect	Impact
Explicit JOIN predicates	Reduces intermediate cardinality from exabytes to ~50-200M rows (orders × items for EMEA customers only)
Selective columns	Eliminates full table width transfer; removes ~95% of irrelevant columns (e.g., internal flags, legacy fields)
Filter pushdown	`WHERE` conditions applied at JOIN stage, not after Cartesian product
Expected speedup	800s -> 8-15s (50-100× improvement)

Additional optimization (if further tuning needed):
- Cluster `orders` on `customer_id` and `order_items` on `order_id` to reduce shuffle.
- Materialize a `FACT_ORDERS_EMEA` table pre-filtered for region if this query runs frequently.

Engineering counterpart
4:46 AM - Responded in 0s

Thanks - walk me through your reasoning step by step. What assumptions did you make?

Agent response
4:46 AM - Responded in 8s

Step-by-Step Reasoning & Assumptions

Step 1: Diagnose the Query Structure
What I observed:
```sql
FROM orders o, order_items oi, customers c
WHERE o.order_date >= '2024-01-01'
  AND c.region = 'EMEA';
```

Reasoning:
- Comma syntax (`,`) in FROM clause = implicit CROSS JOIN in Snowflake
- No `ON` clauses -> no join predicates between tables
- Only 2 WHERE filters on individual tables (not join conditions)

Assumption #1: The intended logic is:
- `orders` connects to `order_items` via `order_id`
- `orders` connects to `customers` via `customer_id`

Why? Standard e-commerce schema. If different, the fix would fail. I inferred from table names and typical business logic.

Step 2: Calculate Cardinality Impact
Given from EXPLAIN:
- ORDER_ITEMS: 452M rows
- ORDERS (filtered): 12.4M rows
- CUSTOMERS (filtered): 890K rows

My calculation:
```
Cartesian product = 452M × 12.4M × 890K
                  ? 4.97 × 10^21 rows (before WHERE filters apply)
```

Assumption #2: Snowflake evaluates the implicit CROSS JOIN before WHERE predicates.

Why? This is standard SQL execution order: FROM -> WHERE. Snowflake doesn't magically infer join keys from column names.

Result: ~5 exabytes of intermediate data before any filtering. At typical Snowflake throughput (100GB/s), this alone = 50,000+ seconds.

Step 3: Infer Missing Join Logic
Assumption #3: The schema follows standard relational design:
- `orders.order_id` (PK) <-> `order_items.order_id` (FK)
- `orders.customer_id` (FK) <-> `customers.customer_id` (PK)

Why I made this assumption:
- Table names suggest e-commerce domain
- EMEA filter on `customers.region` implies we want region-specific orders

- No other logical way to join 3 tables in this context

Risk: If the actual schema differs (e.g., `order_items.customer_id` instead of `order_items.order_id`), my fix breaks the query logic. But the original query is already broken (Cartesian product), so any schema mismatch would be caught immediately.

Step 4: Choose JOIN Type
Assumption #4: All three JOINs should be `INNER JOIN`.

Why?
- Filtering on `c.region = 'EMEA'` suggests we only want EMEA customers
- Filtering on `o.order_date >= '2024-01-01'` suggests we only want recent orders
- No mention of "unmatched records" in dashboard requirements
- INNER JOIN is most selective (smallest result set)

Alternative considered: LEFT JOIN if the requirement was "all EMEA customers, even those with no orders in 2024+". But dashboard context (sales rollup) implies only orders matter.

Step 5: Select Columns Explicitly
Assumption #5:

Engineering counterpart
4:46 AM - Responded in 0s

What would you verify before shipping this to production?

Agent response
4:46 AM - Responded in 7s

Pre-Production Verification Checklist

1) Validate Schema & Join Keys
```sql
-- Verify foreign key relationships exist
SELECT constraint_name, column_name, referenced_table_name, referenced_column_name
FROM information_schema.key_column_usage
WHERE table_name IN ('orders', 'order_items', 'customers')
  AND constraint_type = 'FOREIGN KEY';

-- Confirm column data types match
SELECT table_name, column_name, data_type
FROM information_schema.columns
WHERE (table_name = 'orders' AND column_name = 'order_id')
   OR (table_name = 'order_items' AND column_name = 'order_id')
   OR (table_name = 'orders' AND column_name = 'customer_id')
   OR (table_name = 'customers' AND column_name = 'customer_id');
```

Why: Prevents NULL mismatches, type coercion overhead, or silent wrong-row joins.

---

2) Compare Result Sets: Old vs. New Query
```sql
-- Run BOTH queries, capture row counts & sample data
SELECT COUNT(), COUNT(DISTINCT o.order_id), COUNT(DISTINCT c.customer_id)
FROM orders o, order_items oi, customers c
WHERE o.order_date >= '2024-01-01'
  AND c.region = 'EMEA';

-- vs. my rewritten query (same metrics)
```

Critical checks:
- Row count difference -> join logic error
- Duplicate `order_id` values -> indicates missing join predicate
- NULL values in result -> unexpected outer behavior

Why: The original query may have been intentionally written this way (unlikely but possible). Comparing outputs catches logic errors before production.

---

3) Verify EMEA Region Values
```sql
-- Check what "EMEA" actually contains
SELECT DISTINCT region, COUNT()
FROM customers
GROUP BY region
ORDER BY COUNT() DESC;

-- Confirm 'EMEA' exists and isn't case-sensitive issue
```

```
SELECT COUNT()
FROM customers
WHERE region = 'EMEA'
  OR region = 'emea'
  OR region LIKE '%EMEA%';
```

Why: Typos or case mismatches silently return 0 rows. Dashboard won't error - it just shows no data.

---

4) Test on Staging with Production Data
```sql
-- Run on ANALYTICS_WH_DEV or isolated warehouse
-- Capture query profile
EXPLAIN o.order_date >= '2024-01-01' AND c.region = 'EMEA';

-- Check actual execution time (not just plan)
SELECT COUNT() FROM (
  -- my rewritten query
) LIMIT 1;
```

Expected results:
- Execution time: 8-20s (not 847s)
- Bytes scanned: <500GB (not 18TB)
- No full table scans on ORDER_ITEMS

Why: EXPLAIN can lie; actual execution reveals spill, cache behavior, cluster contention.

---

5) Validate Column Selection
```sql
-- Confirm no business logic depends on dropped columns
-- Query the ORIGINAL dashboard/report code:

SELECT column_name
FROM information_schema.columns
```

Engineering counterpart
4:46 AM - Responded in 0s

What are the top risks, and what's the rollback plan if this doesn't work?

Agent response
4:46 AM - Responded in 8s

Top Risks & Rollback Plan

? Top 5 Production Risks

Risk #1: Silent Logic Change - Wrong Result Set
Severity: CRITICAL
Scenario:
- Original query (despite being Cartesian) accidentally returned correct rows due to downstream deduplication in BI layer
- My INNER JOINs filter out edge cases (e.g., orders with no matching customer record)
- Dashboard shows 15% fewer orders -> stakeholders lose confidence

Detection:
```sql
-- Compare row counts BEFORE deployment
SELECT 'original' AS query_type, COUNT() as row_count
FROM orders o, order_items oi, customers c
WHERE o.order_date >= '2024-01-01' AND c.region = 'EMEA'

UNION ALL

SELECT 'rewritten', COUNT()
FROM orders o
INNER JOIN order_items oi ON o.order_id = oi.order_id
INNER JOIN customers c ON o.customer_id = c.customer_id
WHERE o.order_date >= '2024-01-01' AND c.region = 'EMEA';
```

If counts differ by >5%: STOP. Investigate why before proceeding.

Risk #2: Join Predicate Mismatch
Severity: CRITICAL
Scenario:
- Actual schema uses `order_items.customer_id` (not `order_id`)
- My assumed join keys are wrong
- Query returns 0 rows or crashes with "column not found"

Mitigation:
```sql
-- Run this FIRST in dev
DESCRIBE TABLE orders;
DESCRIBE TABLE order_items;
DESCRIBE TABLE customers;
```

If column names differ: Rewrite joins immediately before testing.

Risk #3: Performance Still Poor (Hidden Cluster Contention)
Severity: HIGH
Scenario:

- Query optimizes to 45s (not 8s) due to warehouse resource limits
- Other queries running simultaneously consume CPU/memory
- Dashboard still times out

Mitigation:
- Run test during off-peak hours (night, weekend)
- Isolate on small warehouse (XS or S) to measure true query cost
- Monitor `QUERY_HISTORY` for concurrent load

If runtime >30s: Escalate to clustering/materialization strategy.

Risk #4: Backward Compatibility with Dependent Systems
Severity: MEDIUM
Scenario:
- Downstream views or ETL jobs depend on the original query structure
- Column order changes break parsing logic in legacy tools
- API clients expect specific column sequence

Mitigation:
```sql
-- Audit all downstream dependencies
SELECT
FROM information_schema.views
WHERE view_definition ILIKE '%FROM orders o, order_items oi, customers c%';

-- Check ETL job definitions
SELECT task_name, definition
FROM information_schema.tasks
WHERE definition ILIKE '%orders%order_items%customers%';
```

If dependencies exist: Coordinate with teams before cutover.

Risk #5: Data Freshness / Stale Cache
Severity: MEDIUM
Scenario:
- Snowflake result cache serves old Cartesian result (unlikely but possible)
- New query hits live data, shows different values
- Stakeholders

Engineering counterpart
4:46 AM - Responded in 0s

Thanks - reasoning on joins and selectivity is clear. Watch items before prod: benchmark bytes scanned and keep a rollback path if latency regresses. Review complete - no further response needed.