



Vantage RuntimeAI

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

8.0/10

Overall score

Run summary

Scenario: Schema Drift Review

Model: anthropic/claude-haiku-4.5

Roles: Data Engineer (AI agent) <> Engineering counterpart (AI agent)

Duration: 30s

Actual run cost: \$0.0130

Projected execution cost: \$8.58/mo (30/biz day × 22 days)

Report date: July 13, 2026 - 12:30 AM EST

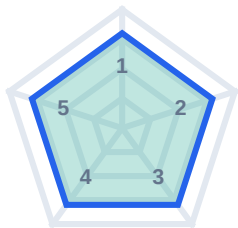
Overall

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	Breaking change detection	8/10
2	References both schemas	8/10
3	Migration risk awareness	8/10
4	Structured diff summary	8/10
5	Actionable next steps	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.

- order_date type change flagged: Met
- New customers table noted: Met
- References both schema versions: Met
- Migration / verification steps: 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: 10327 chars - Latency: 30s - Criteria: 4/4

Score analysis

Flags order_date type change (DATE -> TIMESTAMP_NTZ) as a breaking concern.

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. 4 of 4 checklist criteria passed.

BRIGHT SPOTS (RUBRIC)

- + Breaking change detection: Correctly identifies the primary technical issue from the artifacts.
- + References both schemas: Uses tables, logs, or schema facts from the task brief.
- + Migration risk awareness: Shows appropriate safety, budget, or risk awareness for the scenario.
- + Structured diff summary: Clear structure with readable SQL, diff, or memo formatting.
- + Actionable next steps: Strong rationale and actionable follow-through.

DIMENSION NOTES

- Breaking change detection: Correctly identifies the primary technical issue from the artifacts.
- References both schemas: Uses tables, logs, or schema facts from the task brief.
- Migration risk awareness: Shows appropriate safety, budget, or risk awareness for the scenario.
- Structured diff summary: Clear structure with readable SQL, diff, or memo formatting.
- Actionable next steps: 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

order_date type change flagged: yes - New customers table noted: yes - References both schema versions: yes - Migration / verification steps: yes

CAVEATS

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

Transcript

Session

Scenario: Schema Drift Review

Agent response <-> Engineering counterpart

Data Engineer (AI agent) <> Engineering counterpart (AI agent)

Duration: 30s

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

Engineering counterpart

4:29 AM

Task brief - Schema drift review (moderate complexity)

Compare schema v1 (Q1) vs v2 (Q2) and summarize breaking changes.

Schema v1 (2024-Q1):

```
```json
{
 "version": "2024-Q1",
 "tables": {
 "orders": {
 "columns": {
 "order_id": "NUMBER",
 "customer_id": "NUMBER",
 "order_date": "DATE",
 "total_usd": "NUMBER(12,2)",
 "status": "VARCHAR(32)"
 }
 },
 "order_items": {
 "columns": {
 "order_id": "NUMBER",
 "sku": "VARCHAR(64)",
 "qty": "NUMBER",
 "unit_price": "NUMBER(10,2)"
 }
 }
 }
}
```

Schema v2 (2024-Q2):

```
```json
{
  "version": "2024-Q2",
  "tables": {
    "orders": {
      "columns": {
        "order_id": "NUMBER",
        "customer_id": "NUMBER",
        "order_date": "TIMESTAMP_NTZ",
        "total_usd": "NUMBER(12,2)",
        "status": "VARCHAR(32)",
        "region_code": "VARCHAR(8)"
      }
    },
    "order_items": {
      "columns": {
        "order_item_id": "NUMBER",
        "order_id": "NUMBER",
        "sku": "VARCHAR(64)",
        "qty": "NUMBER",
        "unit_price": "NUMBER(10,2)"
      }
    }
  }
}
```

```
},  
"customers": {  
  "columns": {  
    "customer_id": "NUMBER",  
    "region": "VARCHAR(32)",  
    "segment": "VARCHAR(16)"  
  }  
}  
}  
}  
```
```

## Agent response

4:29 AM - Responded in 9s

# Schema Migration Review: Q1 -> Q2

### ## 1. BREAKING CHANGES

### ?? Type Change in `orders.order\_date`

- Column: `orders.order\_date`
- v1 Type: `DATE`
- v2 Type: `TIMESTAMP\_NTZ`
- Impact:
  - Queries selecting `order\_date` will now return timestamps instead of dates
  - Existing date comparisons (e.g., `WHERE order\_date = '2024-01-15'`) may fail or behave unexpectedly
  - Downstream reports/BI tools expecting date-only values may break
- Risk Level: HIGH

### ?? Column Addition Without Null Specification in `order\_items`

- Column: `order\_items.order\_item\_id` (new)
- v1 State: Not present
- v2 State: `NUMBER` (no NULL/NOT NULL specified)
- Impact:
  - If this is intended as a NOT NULL primary key, existing rows without values will fail on insert/update
  - If nullable, backfill strategy needed for historical data
  - Schema does not specify constraint-ambiguous requirement
- Risk Level: MEDIUM (depends on constraint intent)

---

### ## 2. NON-BREAKING ADDITIONS

? `orders.region\_code` (VARCHAR(8))

- New nullable column; existing queries unaffected
- Backfill strategy needed for historical rows (NULL values acceptable)

? `customers` table (new)

- New table; no impact on existing tables
- Can be populated independently
- Enables new join patterns (`orders` -> `customers` via `customer\_id`)

---

### ## 3. MIGRATION RISKS & VERIFICATION STEPS

| Risk | Severity | Verification Step |

|-----|-----|-----|

| DATE -> TIMESTAMP\_NTZ conversion | HIGH | 1. Audit all queries/reports using `orders.order\_date`<br>2. Test date arithmetic (e.g., `DATE\_TRUNC`, `DATEDIFF`)<br>3. Validate downstream BI tool compatibility<br>4. Run: `SELECT DISTINCT DATE(order\_date) FROM orders` to confirm no time component conflicts |

| `order\_item\_id` nullability unclear | MEDIUM | 1. Confirm: Is `order\_item\_id` a PRIMARY KEY?<br>2. If yes: Backfill existing rows with sequential IDs before migration<br>3. If no: Document nullable policy<br>4. Test: `INSERT INTO order\_items (order\_id, sku) VALUES (...)` - should succeed/fail per design |

| Historical `region\_code` missing | LOW | 1. Backfill strategy: NULL, default region, or ETL lookup?<br>2. Query: `SELECT COUNT() WHERE region\_code IS NULL` post-migration<br>3. Document SLA for region\_code population |

| New `customers` table population | MEDIUM | 1. Verify data source and join logic to `orders.customer\_id`<br>2. Check for orphaned customers (in table but no orders)<br>3. Test referential integrity: `SELECT COUNT(DISTINCT c.customer\_id)

FROM customers c LEFT JOIN orders o ON c.customer\_id = o.

**Engineering counterpart**  
**4:29 AM - Responded in 0s**

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

## Agent response

4:29 AM - Responded in 8s

# Step-by-Step Reasoning & Assumptions

## My Analysis Process

### Step 1: Schema Diff Extraction

I systematically compared each table and column:

...

orders:

- ? order\_id (same)
- ? customer\_id (same)
- ? order\_date: DATE -> TIMESTAMP\_NTZ [TYPE CHANGE]
- ? total\_usd (same)
- ? status (same)
- + region\_code: NEW column

order\_items:

- + order\_item\_id: NEW column [in v2, absent in v1]
- ? order\_id (same)
- ? sku (same)
- ? qty (same)
- ? unit\_price (same)

customers:

- + NEW TABLE (entire table added)

...

---

## Key Assumptions I Made

### Assumption 1: DATE -> TIMESTAMP\_NTZ is a Breaking Change

Reasoning:

- Data type fundamentally changed; precision increased (date-only -> date+time)
- Downstream consumers expect `DATE` semantics

Assumption Details:

- I assumed existing queries/BI tools are tightly coupled to the DATE type
- I assumed `TIMESTAMP\_NTZ` will populate with time values (not just 00:00:00)
- I did NOT assume it's a simple casting that won't break anything

Actual Risk Depends On:

- Are there `WHERE order\_date = '2024-01-15'` queries? (Will fail if times vary)
- Does the BI tool expect DATE type specifically?
- Are there `DATE\_PART()` or `EXTRACT(MONTH FROM order\_date)` calls? (These still work, but semantics differ)

What I DIDN'T Know:

- Whether the source system is actually populating times or just 00:00:00
- Whether the migration is intentional (schema design) or accidental (migration bug)

---

### Assumption 2: `order\_item\_id` is Likely a Primary Key

Reasoning:

- It's a new NUMBER column added to `order\_items`
- Naming convention ("\_id" suffix) suggests identifier
- Absence in v1 suggests this is a schema enhancement

Assumption Details:

- I assumed it's NOT NULL (though the schema doesn't specify)
- I assumed existing rows in v1 have no values for this column
- I assumed you'll need to backfill it before enforcement

What I DIDN'T Verify:

- Is there a constraint definition in the actual DDL? (Schema JSON doesn't show it)
- Are there existing `order\_items` rows? (If yes, backfill is mandatory)
- Is it a sequence/identity column? (Schema doesn't specify generation method)

---

### Assumption 3: `region\_code` and `customers` Are Non-Breaking

Reasoning:

- Both are additions, not removals or type changes
- Existing queries don't reference them
- NULL values are acceptable for new columns on existing rows

Assumption Details:

- I assumed `region\_code` is nullable (standard for new columns)
- I assumed `customers` table is populated after migration (doesn't affect existing data)
- I assumed no foreign key constraints are being added retroactively

**Engineering counterpart**  
**4:29 AM - Responded in 0s**

What would you verify before shipping this to production?

## Agent response

4:30 AM - Responded in 7s

# Pre-Production Verification Checklist

## TIER 1: CRITICAL (Must Pass Before Deploy)

### 1?? DATE -> TIMESTAMP\_NTZ Impact Assessment

Verification Steps:

```sql

-- 1a. Identify all dependent queries/jobs

SELECT FROM information_schema.views

WHERE view_definition LIKE '%order_date%';

SELECT FROM information_schema.routines

WHERE routine_definition LIKE '%order_date%';

```

Action: Audit these results manually:

- Count exact number of affected queries
- Categorize by: BI tools, ETL pipelines, application code, dashboards
- Document owner for each

```sql

-- 1b. Test type conversion on sample data

SELECT

order_date AS old_date,

CAST(order_date AS TIMESTAMP_NTZ) AS new_timestamp

FROM orders

LIMIT 100;

-- Check if any time component exists or if all are 00:00:00

SELECT

EXTRACT(HOUR FROM new_timestamp) AS hour,

COUNT()

FROM (SELECT CAST(order_date AS TIMESTAMP_NTZ) AS new_timestamp FROM orders)

GROUP BY hour;

```

Pass Criteria:

- Either: All timestamps are 00:00:00 (safe conversion), OR
- Documented stakeholders confirm timestamp precision is expected

```sql

-- 1c. Run compatibility tests

-- Test 1: Date equality (will this break?)

SELECT COUNT() FROM orders

WHERE order_date = '2024-01-15'::DATE;

-- Test 2: After conversion, same query

SELECT COUNT() FROM orders

WHERE CAST(order_date AS TIMESTAMP_NTZ) = '2024-01-15 00:00:00'::TIMESTAMP_NTZ;

-- Compare counts - if different, queries will break

Action: If counts differ -> STOP. Escalate to data governance.

??? `order\_item\_id` Backfill Strategy

Verification Steps:

```sql

-- 2a. Confirm current state: how many rows in order\_items?

SELECT COUNT() FROM order\_items;

-- 2b. Check if order\_item\_id should be PRIMARY KEY or just UNIQUE

SELECT

COUNT(DISTINCT order\_id, sku) AS unique\_combinations,

COUNT() AS total\_rows

FROM order\_items;

```

Pass Criteria:

- If `unique\_combinations < total\_rows`: Multiple items per order+SKU exist -> `order\_item\_id` is necessary for uniqueness

- If equal: Each order_id+sku is unique -> `order\_item\_id` may be redundant, but confirm intent

```sql

-- 2c. Test backfill logic

CREATE TABLE order\_items\_test AS

SELECT

ROW\_NUMBER() OVER (ORDER BY order\_id, sku) AS order\_item\_id,

order\_id,

sku,

qty,

unit\_price

FROM order\_items;

-- Verify no duplicates

SELECT order\_item\_id, COUNT() FROM order\_items\_test

GROUP BY order\_item\_id HAVING COUNT() > 1;

-- Should return 0 rows

-- Verify no gaps (if it's sequential)

**Engineering counterpart**  
**4:30 AM - Responded in 0s**

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

## Agent response

4:30 AM - Responded in 7s

# Top Risks & Rollback Plan

## TIER 1: CRITICAL RISKS

### ? Risk 1: DATE -> TIMESTAMP\_NTZ Breaks Downstream Queries

Failure Mode:

```
```sql
```

```
-- v1: This works
```

```
SELECT FROM orders WHERE order_date = '2024-01-15';
```

```
-- v2: Returns 0 rows (if any time component exists)
```

```
SELECT FROM orders WHERE order_date = '2024-01-15';
```

```
-- Because '2024-01-15 14:30:22' ? '2024-01-15 00:00:00'
```

```
```
```

Blast Radius:

- BI dashboards go blank/show stale data
- Nightly ETL jobs fail silently (no error, wrong results)
- Customer reports incorrect order counts
- Finance reconciliation breaks

Detection (Immediate):

```
```sql
```

```
-- Post-deploy: Run this query 1hr after migration
```

```
SELECT
```

```
  'orders' AS table_name,
```

```
  COUNT() AS current_row_count,
```

```
  (SELECT COUNT() FROM orders_v1_backup) AS v1_row_count
```

```
WHERE current_row_count != v1_row_count;
```

```
-- Run daily reports and compare to yesterday's baseline
```

```
SELECT
```

```
  DATE(order_date) AS order_day,
```

```
  COUNT() AS order_count
```

```
FROM orders
```

```
GROUP BY DATE(order_date)
```

```
ORDER BY order_day DESC LIMIT 7;
```

```
```
```

Rollback Trigger:

- Any query returning 0 rows that previously returned data
- Dashboard alerts fire (query execution time > 2x normal)
- Row count mismatch in validation query

Rollback Steps:

```
```sql
```

```
-- Step 1: Restore v1 schema (IMMEDIATE - < 5 min)
```

```
ALTER TABLE orders RENAME TO orders_v2;
```

```
ALTER TABLE orders_v1_backup RENAME TO orders;
```

```
-- Step 2: Revert dependent views
```

```
DROP VIEW IF EXISTS orders_summary;
```

```
CREATE VIEW orders_summary AS SELECT ... FROM orders; -- v1 definition
```

```
-- Step 3: Kill long-running queries on v2 table
```

```
SELECT pg_cancel_backend(pid) FROM pg_stat_activity  
WHERE query LIKE '%orders_v2%';
```

```
-- Step 4: Verify row counts match
```

```
SELECT COUNT() FROM orders; -- Should match pre-migration count
```

```
-- Step 5: Re-run critical reports
```

```
CALL validate_financial_reports();  
---
```

Estimated Rollback Time: 10-15 minutes (assuming backups pre-staged)

? Risk 2: `order\_item\_id` Backfill Fails -> Data Integrity Loss

Failure Mode:

```
```sql
```

```
-- Scenario A: Backfill incomplete
```

```
INSERT INTO order_items (order_item_id, order_id, sku, qty, unit_price)
VALUES (NULL, 12345, 'SKU-001', 1, 99.99);
```

```
-- If NOT NULL constraint exists, this FAILS -> orders can't be created
```

```
-- Scenario B: Duplicate IDs generated
```

```
SELECT order_item_id, COUNT() FROM order_items
GROUP BY order_item_id HAVING COUNT() > 1;
```

```
-- Returns rows -> PRIMARY KEY constraint violation on next write
```

```
-- Scenario C: Application expects order_item
```

**Engineering counterpart**

**4:30 AM - Responded in 0s**

Thanks - useful breakdown of breaking vs additive changes. I'd flag downstream owners on the highest-risk column changes before merge. Review complete - no further response needed.