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: qwen/qwen3-235b-a22b-2507
Roles: Data Engineer (AI agent) <> Engineering counterpart (AI agent)
Duration: 3m 11s
Actual run cost: $0.001734
Projected execution cost: $1.14/mo (30/biz day × 22 days)
Report date: July 14, 2026 - 3:24 AM EST

Session: sample-lb-rankings-seed-1b8b3251-b146-46b2-9014-80aea2a806de

de_schema_drift_v1 - agent - openrouter - qwen/qwen3-235b-a22b-2507 - ended

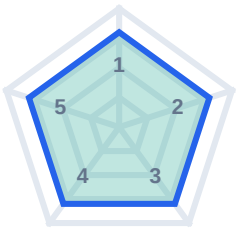
Overall

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	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: 12683 chars - Latency: 191s - 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: 3m 11s

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

Engineering counterpart
7:21 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
7:22 AM - Responded in 1m 8s

? Schema Drift Review: v1 (2024-Q1) -> v2 (2024-Q2)

---

1. Breaking Changes

These changes may break existing queries, ETL pipelines, or applications relying on the old schema.

? Data Type Change in `orders.order_date`
- Before: `DATE`
- After: `TIMESTAMP_NTZ`
- Impact: Queries that expect a `DATE` (e.g., using date arithmetic, filtering with `DATE = '2024-01-01'`) may behave unexpectedly or fail if implicit casting is not supported. Functions like `DATE()` may now require explicit casting.

? Added Required Context: `region_code` in `orders`
- Added: `region_code` -> `VARCHAR(8)`
- Impact: While not explicitly marked as `NOT NULL`, the addition of a new column without a default in a table may break `INSERT` operations in downstream processes if they do not account for it (though less likely in read-heavy analytic systems). However, since it's a new non-nullable field in a critical fact table, ingestion pipelines may break unless `region_code` is supplied.

> ?? Note: In absence of `NOT NULL` specification, assume nullable unless known otherwise - but the presence of this field in v2 suggests new business logic dependency.

? Primary Key Change in `order_items`
- Before: Composite key likely on `(order_id, sku)` or `(order_id, qty)`
- After: New column `order_item_id` introduced - implies shift to surrogate key
- Impact:
  - Existing joins or lookups on `order_items` using only `order_id` may miss granularity.
  - Any downstream logic assuming no unique row identifier in `order_items` will need revision.
  - Removal of implicit uniqueness on `(order_id, sku)` unless enforced via constraint.

? Missing Table Reference in v1 -> v2 Transition
- `customers` table is new in v2, but `orders.region_code` suggests referential logic (e.g., region now lives in both `orders` and `customers`). This could indicate denormalization, which must be validated.

---

2. Non-Breaking Additions

These additions are safe for existing consumers but expand capabilities.

? New Column: `orders.region_code`
- `VARCHAR(8)` - optional addition (unless `NOT NULL`, which is not specified).
- Enables regional analysis at order level.
- Safe for existing queries that don't reference it.

? New Surrogate Key: `order_items.order_item_id`
- `NUMBER` - allows for easier deduplication, CDC, and row-level tracking.
- Does not break existing queries using `SELECT `, but adds a new field.
- Enhances integrity for slowly changing dimensions or incremental loads.

? New Table: `customers`

- Contains `customer_id`, `region`, `segment`
- Enables customer segmentation analytics.
- Since it doesn't replace or remove any v1 functionality, it's non-breaking.
- May suggest deprecation of `orders.region_code` in future (if region is sourced from `customers`), but not currently conflicting.

---

3. Migration Risks & Recommended Verification Steps

Risk	Description	Mitigation
Timestamp vs Date mismatch	`order_date` changed from `DATE` to `TIMESTAMP_NTZ`. Queries filtering by date may return unexpected results due to time component.	Update queries to use `DATE(order_date)` or `order_date::DATE`. Validate report logic (e.g., daily

Engineering counterpart
7:22 AM - Responded in 0s

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

Agent response
7:23 AM - Responded in 56s

Absolutely - let's walk through the step-by-step reasoning behind the schema drift analysis, and clearly surface the assumptions made along the way.

? Step 1: Understand the Task
We are comparing two JSON-based schema snapshots:
- v1 (2024-Q1) -> Baseline
- v2 (2024-Q2) -> New version

Goal: Identify breaking changes - i.e., changes that could break existing code, ETL pipelines, reports, or assumptions made by downstream systems.

Also to detect non-breaking additions (safe extensions), and assess migration risks.

? Step 2: Compare Tables and Columns

We go table-by-table, column-by-column.

? Tables Present in Both: `orders`, `order_items`
- These exist in both versions -> possible structural changes.

? New Table in v2: `customers`
- Not present in v1.
- This is a net addition -> likely non-breaking, since no existing logic depended on it.

> ? Assumption: Downstream systems do not require all tables to be present unless explicitly referenced. Adding a new table doesn't break existing queries on `orders` or `order_items`.

? Step 3: Analyze `orders` Table

Column	v1 Type	v2 Type	Change?
`order_id`	`NUMBER`	`NUMBER`	? Same
`customer_id`	`NUMBER`	`NUMBER`	? Same
`order_date`	`DATE`	`TIMESTAMP_NTZ`	?? Type change
`total_usd`	`NUMBER(12,2)`	`NUMBER(12,2)`	? Same
`status`	`VARCHAR(32)`	`VARCHAR(32)`	? Same
(new) `region_code`	-	`VARCHAR(8)`	? Added

? Change: `order_date` from `DATE` -> `TIMESTAMP_NTZ`

- Breaking? YES, potentially.
- Why?
  - `DATE` stores only date (e.g., `2024-01-15`)
  - `TIMESTAMP_NTZ` stores date + time (e.g., `2024-01-15 14:30:00`)
  - A query like:
    ```sql
    WHERE order_date = '2024-01-15'
    ```

```

may now exclude rows unless the time component is midnight, or explicit casting is used.

> ? Assumption: Consumers rely on date-only filtering. Many BI tools and legacy queries assume `order_date` is `DATE`. The type change increases precision, but introduces potential filtering mismatches.

- Also affects:
  - Partitioning strategies (if partitioned by date)
  - Materialized views
  - ETL jobs that validate data types

> ? Assumption: There is no automatic/implicit cast from `TIMESTAMP_NTZ` to `DATE` in all consumer contexts. Some tools (e.g., older JDBC drivers) may fail or behave unexpectedly.

##### ? New Column: `region_code` added

- Not in v1 -> new field.
- Likely non-breaking for reads, but risky for writes.
- However, if downstream processes perform `INSERT INTO orders (...)` without listing columns, they may fail when trying to insert into v2 if `region_code` is `NOT NULL`.

> ?? Assumption: We don't see
```

Engineering counterpart
7:23 AM - Responded in 0s

What would you verify before shipping this to production?

Agent response
7:24 AM - Responded in 49s

Excellent question. Before shipping the schema v2 (2024-Q2) to production - especially after identifying the changes and risks - you must verify several key areas to prevent data pipeline failures, reporting inaccuracies, or application outages.

Here's a structured verification checklist, grouped by area, with clear what to verify, how, and why:

? Pre-Production Verification Checklist

1. Validate `order_date` Type Change: `DATE` -> `TIMESTAMP_NTZ`

What to Verify	How to Verify	Why It Matters
Downstream queries still work as expected	Run existing reports and dashboards using `order_date` (e.g., daily sales) against a staging copy of v2. Check results.	Queries like `WHERE order_date = '2024-01-15'` may return zero results if time components exist.
Date-only logic is preserved	Confirm use of `order_date::DATE`, `DATE(order_date)`, or `CAST(order_date AS DATE)` in all queries.	Ensures filters, joins, and aggregations by day still function correctly.
ETL jobs handle timestamp correctly	Review incremental load logic: e.g., does `last_ingested_date` now compare timestamps? Could cause duplicate/missed records.	Prevents data loss or reprocessing loops.

> ? Action: Add regression tests for date-based aggregations and filters.

---

2. Confirm Behavior of New `orders.region_code` Column

What to Verify	How to Verify	Why It Matters
Is `region_code` nullable?	Check DDL: is it defined as `NULL` or `NOT NULL`?	If `NOT NULL`, any process inserting into `orders` without this field will fail.
Where is `region_code` sourced from?	Trace lineage: is it derived from `customer_id` via `customers`? Is it filled at order time?	Risk of inconsistency if `orders.region_code` ? `customers.region`.
Does it duplicate data?	Compare `orders.region_code` vs `customers.region` for same `customer_id`.	Denormalization is acceptable, but must be consistent.

> ? Action: Ensure data pipeline populates `region_code` correctly at insert time.

---

3. Verify `order_items.order_item_id` Surrogate Key Impact

What to Verify	How to Verify	Why It Matters
Is `order_item_id` populated in all existing and new data?	Check backfill logic: was it added with `AUTO_INCREMENT` or generated from source?	Gaps break downstream joins or change data capture.
Do any processes rely on `(order_id, sku)` as a unique key?	Audit codebase for uniqueness assumptions.	If duplicates now exist in `(order_id, sku)` due to new key, logic may break.
Are incremental loads using `order_item_id` for deduplication?	Review merge/upsert logic (e.g., `MERGE INTO` on `order_item_id`).	Without using new key, risk of duplicates increases.

> ? Action: Update SCD (slowly changing dimension) logic to use `order_item_id` as row identifier.

---

4. Test the New `customers` Table Integration

What to Verify	How to Verify	Why It Matters
Is `customers` populated completely and timely?	Validate completeness: do all `customer_id`s in `orders` exist in `customers`?

Engineering counterpart
7:24 AM - Responded in 0s

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

Agent response
7:24 AM - Responded in 18s

Excellent - now that we've reviewed the changes and verification steps, let's focus on crisis readiness: what could go wrong in production, and how do we respond?

Here are the top risks, ranked by impact and likelihood, followed by a clear, actionable rollback plan.

?? Top Risks (High Impact / High Likelihood)

1. `order_date` Type Change Breaks Critical Reports
- Risk: Reports that filter by `order_date = '2024-03-01'` return zero results because the column is now `TIMESTAMP_NTZ` with time components (e.g., `2024-03-01 14:22:10`).
- Impact: High - daily sales, dashboards, and KPIs go dark or show zero.
- Likelihood: High - most BI tools and SQL scripts assume date filtering works as before.
- Symptoms:
  - Sudden drop in reported daily volume.
  - Filters stop working.
  - Data teams get paged.

> ? Critical Path Risk: This change silently breaks logic without throwing errors in many SQL engines.

---

2. ETL Jobs Fail Due to Missing `region_code` in Inserts
- Risk: If `orders.region_code` is `NOT NULL` and ingestion jobs don't supply it, `INSERT` statements fail.
- Impact: High - order data stops loading. Pipeline halts.
- Likelihood: Medium to High - especially if `region_code` depends on joined data not yet available.
- Symptoms:
  - Load jobs fail with `NULL value not allowed for NOT NULL column`.
  - Backfill stalls.

> ? Operational Outage Risk: Halts data freshness.

---

3. `order_items` Surrogate Key Causes Joins or Duplicates
- Risk: Downstream logic assumes `(order_id, sku)` is unique, but now multiple rows with same `(order_id, sku)` exist (e.g., two line items for same product in one order).
- Impact: Medium - aggregations double-count revenue or quantity.
- Likelihood: Medium - depends on source system behavior.
- Symptoms:
  - Revenue overstatement in item-level reports.
  - Reconciliation failures vs source system.

> ? Data Integrity Risk: Wrong numbers, hard to detect quickly.

---

4. `customers` Table Incomplete or Stale
- Risk: `customers` table is new, but not fully backfilled or refreshed. Queries joining `orders` -> `customers` miss data.
- Impact: Medium - segmentation reports (e.g., by region or segment) show incomplete results.
- Likelihood: Medium - common in early rollout of new dimension tables.
- Symptoms:

- "Unknown" regions in dashboards.
- Join cardinality issues (e.g., 1:N instead of 1:1).

> ? Analytical Accuracy Risk: Misleads business decisions.

?? Rollback Plan (If Things Go Wrong)

If critical issues are detected within 24-48 hours of deployment, execute rollback in this order:

? Step 1: Stop Ingestion & Freeze Changes
- Pause all batch and streaming writes to affected tables (`orders`, `order_items`).
- Prevent further corruption or inconsistency.

> ? Tooling: Use pipeline orchestrator (e.g., Airflow, Dagster) to pause DAGs.

? Step 2: Restore Tables to v1 Schema
Decide based on severity:

? Option A: Full Schema Rollback (Recommended

Engineering counterpart
7:24 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.