



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/qwen-2.5-7b-instruct

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

Duration: 33s

Actual run cost: \$0.000880

Projected execution cost: \$0.5808/mo (30/biz day × 22 days)

Report date: July 14, 2026 - 3:21 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: 12280 chars - Latency: 33s - 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: 33s

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

Engineering counterpart

7:20 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

7:20 AM - Responded in 6s

### ### Breaking Changes

#### 1. Type Changes:

- `orders` table:
  - `order\_date` was changed from `DATE` to `TIMESTAMP\_NTZ`.

#### 2. Removed Columns:

- `orders` table:
  - The column `customer\_id` was not removed but is now redundant in the `customers` table.

#### 3. Removed Tables:

- `customers` table was added in v2 but was not present in v1.

### ### Non-breaking Additions

#### 1. New Column:

- `orders` table:
  - `region\_code` was added.
- `order\_items` table:
  - `order\_item\_id` was added.

### ### Migration Risks and Recommended Verification Steps

#### 1. Type Change Risks:

- Risk: Data truncation or loss due to the change from `DATE` to `TIMESTAMP\_NTZ`.
- Verification Steps:
  - Validate the `order\_date` values to ensure they are not truncated.
  - Confirm the granularity of the date/time values to ensure no loss of information.
  - Run SELECT queries to inspect the new `TIMESTAMP\_NTZ` values.

#### 2. New Column Risks:

- Risk: Missing values in `region\_code` column if the data was not previously available.
- Verification Steps:
  - Check for NULLs in the `region\_code` column.
  - Populate the `region\_code` column if necessary, especially for new orders.

#### 3. Added Table Risks:

- Risk: Missing customer metadata if the `customers` table data was not previously available.
- Verification Steps:
  - Ensure all `customer\_id` references in `orders` and `order\_items` have corresponding entries in the `customers` table.
  - Populate the `customers` table if necessary.

#### 4. Redundant Fields:

- Risk: Redundancy due to `customer\_id` in `orders` table being present in the new `customers` table.
- Verification Steps:
  - Validate that the `customer\_id` column in `orders` table is not used in any business logic that relies on it.
  - Ensure that any queries or views that depend on `customer\_id` in `orders` are adjusted if necessary.

### ### Summary

- Breaking Changes: Type change of `order\_date` to `TIMESTAMP\_NTZ`, removal of `customer\_id` column (now redundant), and removal of the `orders` table (replaced by `customers` table).
- Non-breaking Additions: Addition of `region\_code` to `orders` table and `order\_item\_id` to `order\_items` table.
- Migration Risks: Ensure no data loss during type conversions, handle missing values in new columns, and address redundancy in fields.

By following these verification steps, you can ensure a smooth migration with minimal risk.

**Engineering counterpart**  
**7:20 AM - Responded in 0s**

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

## Agent response

7:21 AM - Responded in 8s

Certainly! Let's break down the reasoning step by step, including the assumptions made during the analysis.

### ### Step-by-Step Reasoning

#### #### 1. Identify Schema Differences

First, we need to compare the two schemas and identify the differences:

- Schema v1 (2024-Q1):

- `version: "2024-Q1"`
- `tables:`
  - `orders`
    - `customer\_id` (NUMBER)
    - `order\_date` (DATE)
    - `total\_usd` (NUMBER(12,2))
    - `status` (VARCHAR(32))
  - `order\_items`
    - `order\_id` (NUMBER)
    - `sku` (VARCHAR(64))
    - `qty` (NUMBER)
    - `unit\_price` (NUMBER(10,2))

- Schema v2 (2024-Q2):

- `version: "2024-Q2"`
- `tables:`
  - `orders`
    - `customer\_id` (NUMBER)
    - `order\_date` (TIMESTAMP\_NTZ)
    - `total\_usd` (NUMBER(12,2))
    - `status` (VARCHAR(32))
    - `region\_code` (VARCHAR(8))
  - `order\_items`
    - `order\_item\_id` (NUMBER)
    - `order\_id` (NUMBER)
    - `sku` (VARCHAR(64))
    - `qty` (NUMBER)
    - `unit\_price` (NUMBER(10,2))
  - `customers`
    - `customer\_id` (NUMBER)
    - `region` (VARCHAR(32))
    - `segment` (VARCHAR(16))

#### #### 2. Identify Breaking Changes

Breaking changes are significant changes that may cause issues during the migration.

- Type Changes:

- `orders.order\_date` in v1 was a `DATE`, but in v2 it is a `TIMESTAMP\_NTZ`. This is a breaking change because the data type has changed, which might affect how the date values are stored and processed.

- Removed Columns:

- The `customer\_id` column in `orders` table does not exist in v2. However, it is present in the `customers` table, making it redundant.

- Removed Tables:

- The `customers` table was added in v2 but did not exist in v1. This means that the data from `customer\_id` in `orders`

table now resides in the `customers` table.

### #### 3. Identify Non-Breaking Additions

Non-breaking additions are new columns or tables that do not change the existing schema fundamentally.

#### - New Column:

- `orders.region\_code`: This is a new column added to the `orders` table in v2.

#### - New Table:

- `customers`: A new table was added in v2, which contains customer metadata like `customer\_id`, `region`, and `segment`.

### ### Assumptions Made

#### 1. Data Integrity:

- Assumption: The `customer\_id` in `orders` table in v1 must have corresponding entries in the new `customers` table.
- Reasoning: If `customer\_id` in `orders` does not have corresponding entries in `customers`, the migration could lead to missing customer metadata.

#### 2. Data Conversion:

- Assumption: The `order\_date` values in `orders` table (v1) should be compatible with the new `TIMESTAMP\_NTZ` type in `orders` table (v2).
- Reasoning: `DATE` can be converted to `TIMESTAMP\_NTZ` without loss of information, but we need to validate this

**Engineering counterpart**  
**7:21 AM - Responded in 0s**

What would you verify before shipping this to production?

## Agent response

7:21 AM - Responded in 12s

Before shipping the schema migration to production, it's crucial to verify several aspects to ensure the migration is smooth and won't introduce any issues. Here's a detailed plan of verification steps:

### ### Verification Steps

#### #### 1. Data Type Validation

##### - Validate Date Conversion:

- Step: Run a query to check if the `order\_date` values in the `orders` table are compatible and correctly converted to `TIMESTAMP\_NTZ`.

##### - Example Query:

```
```sql
SELECT order_id, order_date, CAST(order_date AS TIMESTAMP_NTZ) AS converted_date
FROM orders;
```
```

- Expected Outcome: Ensure that all `order\_date` values are correctly converted to `TIMESTAMP\_NTZ` without any truncation or loss of information.

##### - Ensure No Missing Values:

- Step: Verify that no `order\_date` values are missing after the conversion.

##### - Example Query:

```
```sql
SELECT COUNT() AS count
FROM orders
WHERE order_date IS NULL OR CAST(order_date AS TIMESTAMP_NTZ) IS NULL;
```
```

- Expected Outcome: No rows should be returned indicating all `order\_date` values are correctly converted.

#### #### 2. Schema Verification

##### - Tables and Columns: Ensure that all tables and columns are correctly mapped.

- Step: Verify that the `orders` table now contains `region\_code` and does not contain `customer\_id`.

- Step: Verify that the `order\_items` table now contains `order\_item\_id`.

- Step: Verify that the `customers` table exists and contains the expected columns (`customer\_id`, `region`, `segment`).

##### - Example Queries:

##### - Orders Table:

```
```sql
DESCRIBE TABLE orders;
```
```

- Expected Outcome: The `region\_code` column should be present.

- Step: Ensure `customer\_id` is no longer present.

##### - Order Items Table:

```
```sql
DESCRIBE TABLE order_items;
```
```

- Expected Outcome: The `order\_item\_id` column should be present.

##### - Customers Table:

```
```sql
DESCRIBE TABLE customers;
```
```

- Expected Outcome: The `customers` table should exist with the expected columns.

### #### 3. Data Integrity Verification

- Check for Missing Customer Metadata:

- Step: Ensure that all `customer\_id` values in the `orders` table have corresponding entries in the `customers` table.

- Example Query:

```
```sql
SELECT o.customer_id
FROM orders o
LEFT JOIN customers c ON o.customer_id = c.customer_id
WHERE c.customer_id IS NULL;
```
```

- Expected Outcome: No rows should be returned indicating all `customer\_id` values have corresponding entries in the `customers` table.

- Check for Redundant Fields:

- Step: Ensure that the `customer\_id` column in the `orders` table is not used in any business logic.

- Example Query:

```
```sql
DESCRIBE TABLE orders;
```
```

- Expected Outcome: Confirm that `customer\_id` is no longer part of the `orders` table schema.

### #### 4. Business Logic Verification

- Test Queries:

- Step: Ensure that all queries and business logic that rely on the old schema still work correctly.

- Example Query:

```
```sql
SELECT customer_id, COUNT() AS order_count
FROM orders
GROUP BY customer_id
ORDER BY order_count DESC;
```
```

- Expected Outcome: The same results should be returned as before, with the correct mapping of `customer\_id` to `customers` table.

- Test Reports

**Engineering counterpart**  
**7:21 AM - Responded in 0s**

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

## Agent response

7:21 AM - Responded in 7s

### ### Top Risks and Rollback Plan

#### #### Top Risks

##### 1. Data Loss Due to Type Conversion:

- Risk: The `order\_date` values might be truncated or lost when converting from `DATE` to `TIMESTAMP\_NTZ`.
- Mitigation: Validate the data conversion and ensure no data loss.

##### 2. Missing Customer Metadata:

- Risk: There might be `customer\_id` values in the `orders` table that do not have corresponding entries in the `customers` table.
- Mitigation: Verify the integrity of the `customers` table and ensure all `customer\_id` values have corresponding entries.

##### 3. Redundant Fields:

- Risk: Query logic that relies on the `customer\_id` in the `orders` table might break.
- Mitigation: Ensure that all query logic has been adjusted for the new schema.

##### 4. Business Logic Failures:

- Risk: Business logic and reports might produce incorrect results due to schema changes.
- Mitigation: Test all business logic and reports to ensure they still work correctly.

### ### Rollback Plan

If any of these issues arise during the migration and the changes need to be rolled back, here is a detailed rollback plan:

#### #### Rollback Steps

##### 1. Backup Current Database:

- Step: Ensure a full backup of the current database is taken.
- Example Command:

```
```sql
BACKUP DATABASE current_database;
```
```

##### 2. Restore to Pre-Migration State:

- Step: Restore the database to the state before the schema migration.
- Example Command:

```
```sql
RESTORE DATABASE current_database FROM BACKUP;
```
```

##### 3. Revert Schema Changes:

- Step: Revert the schema changes to v1.
- Script Example:

```
```sql
-- Revert orders table schema
ALTER TABLE orders RENAME COLUMN customer_id TO pre_customer_id;
ALTER TABLE orders ADD COLUMN customer_id NUMBER;
ALTER TABLE orders MODIFY COLUMN order_date DATE;
-- Revert order_items table schema
ALTER TABLE order_items RENAME COLUMN order_id TO pre_order_id;
ALTER TABLE order_items ADD COLUMN order_id NUMBER;
-- Revert customers table schema
```

```
DROP TABLE customers;
'''
```

4. Repopulate Tables:

- Step: Repopulate the `customers` table from the backup if necessary.
- Example Command:

```
```sql
INSERT INTO customers (customer_id, region, segment)
SELECT customer_id, region, segment
FROM backup_customers;
'''
```

#### 5. Update Business Logic:

- Step: Ensure that all business logic and queries are updated to the pre-migration state.
- Example Script:

```
```sql
-- Update orders table queries
SELECT pre_customer_id, order_date, total_usd, status
FROM orders;

-- Update order_items table queries
SELECT pre_order_id, order_id, sku, qty, unit_price
FROM order_items;
'''
```

6. Test Post-Rollback:

- Step: Thoroughly test the system to ensure all functionalities work as expected.
- Example Tests:

```
```sql
-- Test orders table
SELECT COUNT() AS count
FROM orders
WHERE order_date IS NOT NULL;

-- Test order_items table
SELECT COUNT() AS count
FROM order_items
WHERE order_id IS NOT NULL;

-- Test customers table
SELECT COUNT() AS count
FROM customers
WHERE customer_id IS NOT NULL;
'''
```

#### 7. Communicate with Stakeholders:

- Step: Inform all stakeholders

**Engineering counterpart**

**7:21 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.