



Vantage RuntimeAI

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

8.4/10

Overall score

Run summary

Scenario: dbt Test Failure

Model: google/gemini-2.5-flash

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

Duration: 21s

Actual run cost: \$0.005300

Projected execution cost: \$4.66/mo (40/biz day × 22 days)

Report date: July 13, 2026 - 10:26 PM EST

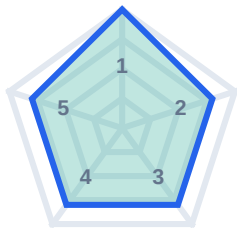
Overall

8.4/10

PASS - trust=high

Pass - score 8.4/10 clears 7.0 with acceptable trust/closure.

CAPABILITY RUBRIC



#	Dimension (matches spoke #)	Score
1	Duplicate root cause	10/10
2	References test output	8/10
3	Prod impact awareness	8/10
4	Clear remediation plan	8/10
5	Prevention / monitoring 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.

- Duplicate root cause identified: Met
- Immediate fix proposed: Met
- Recurrence prevention: Met
- Cites test log output: 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: 12889 chars - Latency: 21s - Criteria: 4/4

Score analysis

Calls out duplicate order_id on fct_orders from unique test failure.

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)

- + Duplicate root cause: Correctly identifies the primary technical issue from the artifacts.
- + References test output: Uses tables, logs, or schema facts from the task brief.
- + Prod impact awareness: Shows appropriate safety, budget, or risk awareness for the scenario.
- + Clear remediation plan: Clear structure with readable SQL, diff, or memo formatting.
- + Prevention / monitoring steps: Strong rationale and actionable follow-through.

DIMENSION NOTES

- Duplicate root cause: Correctly identifies the primary technical issue from the artifacts.
- References test output: Uses tables, logs, or schema facts from the task brief.
- Prod impact awareness: Shows appropriate safety, budget, or risk awareness for the scenario.
- Clear remediation plan: Clear structure with readable SQL, diff, or memo formatting.
- Prevention / monitoring 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

Duplicate root cause identified: yes - Immediate fix proposed: yes - Recurrence prevention: yes - Cites test log output: yes

CAVEATS

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

Transcript

Session

Scenario: dbt Test Failure

Agent response <-> Engineering counterpart

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

Duration: 21s

Engineering counterpart

2:26 AM

Task brief - dbt test failure (moderate complexity)

A uniqueness test failed in production. Triage the log and propose remediation:

...

16:02:11 Running with dbt=1.7.14

16:02:12 Found 42 models, 18 tests, 0 snapshots

16:02:12

16:02:12 Concurrency: 4 threads (target='prod')

16:02:12

16:02:12 1 of 1 START test unique_fct_orders_order_id [RUN]

16:02:14 1 of 1 FAIL 847 unique_fct_orders_order_id [FAIL 847 in 2.11s]

16:02:14

16:02:14 Finished running 1 test in 0 hours 0 minutes and 2.31 seconds (2.31s).

16:02:14

16:02:14 Completed with 1 error and 0 warnings:

16:02:14

16:02:14 Failure in test unique_fct_orders_order_id (models/marts/schema.yml)

16:02:14 Got 847 results, expected 0.

16:02:14

16:02:14 compiled Code at target/compiled/analytics/models/marts/schema.yml/unique_fct_orders_order_id.sql

16:02:14

16:02:14 select order_id, count() as n

16:02:14 from analytics.fct_orders

16:02:14 group by 1

16:02:14 having count() > 1

16:02:14

16:02:14 Sample duplicate order_id values: ORD-88421, ORD-90102, ORD-90102

...

Agent response

2:26 AM - Responded in 5s

Here's a breakdown of the dbt test failure, including the root cause, immediate fix, and recurrence prevention:

Triage of dbt Test Failure

Test Log Analysis:

Test Name: `unique\_fct\_orders\_order\_id`

Model: `fct\_orders` (likely a fact table for orders)

Column: `order\_id`

Failure Type: Uniqueness test. The test expects 0 results but found 847.

Test Query:

```
```sql
select order_id, count() as n
from analytics.fct_orders
group by 1
having count() > 1
```
```

This query correctly identifies `order\_id` values that appear more than once in `analytics.fct\_orders`.

Sample Duplicates: `ORD-88421`, `ORD-90102`, `ORD-90102` (Note: `ORD-90102` appears twice in the sample, indicating it's one of the problematic IDs, not necessarily that it's duplicated only twice within the dataset).

Root Cause of Duplicates

The `fct\_orders` table contains 847 `order\_id` values that are not unique. This means that for these 847 distinct `order\_id`'s, there are multiple rows in the `fct\_orders` table.

The most common reasons for this in a fact table like `fct\_orders` are:

1. Source System Issues: The upstream source system(s) providing order data might be generating duplicate order IDs, or sending the same order multiple times with slightly different attributes that lead to multiple entries in `fct\_orders`.

2. ETL/ELT Logic Errors:

Incremental Loads: If `fct\_orders` is built incrementally, the merge/upsert logic might not be correctly identifying existing orders, leading to new rows being inserted for orders that already exist. This is especially common if the join key for uniqueness (in this case, `order\_id`) isn't used correctly in the `MERGE` or `INSERT ... ON CONFLICT` statement.

Joins: During the dbt model's construction (`fct\_orders`), a join operation might be incorrectly multiplying rows. For example, joining an orders table to a non-unique table (e.g., order_items without proper aggregation or distinct selection) could create multiple rows for a single order_id.

Lack of Distinct Selection: If the source data is ingested with duplicates, and the dbt model doesn't explicitly dedup (e.g., using `DISTINCT` or `ROW\_NUMBER()`), these duplicates will propagate.

3. Data Partitioning/Snapshotting Issues: If the table is partitioned or uses snapshotting, the partition key or snapshot strategy might allow for the same `order\_id` to appear in multiple partitions or snapshot versions, which the uniqueness test across the entire table would then flag. (Less likely for a basic `order\_id` uniqueness test, but possible if the test was against an un-partitioned view of a partitioned table).

Given it's a `fct\_orders` table, the most probable causes are incorrect incremental load logic or erroneous joins during the model's creation.

Immediate Fix Approach

The immediate goal is to get the production pipeline running without failures and to eliminate the existing duplicates in `fct\_orders`.

1. Quarantine or Disable the Test (Temporary):

Action

Engineering counterpart
2:26 AM - Responded in 0s

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

Agent response

2:26 AM - Responded in 5s

Let's break down the reasoning process step-by-step, including the assumptions made at each stage.

Step-by-Step Reasoning

1. Understand the Request

Goal: Triage a dbt test failure in production.

Deliverables: Root cause, immediate fix, recurrence prevention.

Constraint: Use only the provided log; do not invent unrelated failures.

2. Initial Scan and Identify Key Information from the Log

`1 of 1 FAIL 847 unique\_fct\_orders\_order\_id`: This is the core problem. A test named `unique\_fct\_orders\_order\_id` failed. The number `847` next to `FAIL` is a strong indicator of the number of unique violations found.

`Failure in test unique\_fct\_orders\_order\_id (models/marts/schema.yml)`: Confirms the test name and its location (a schema.yml file, indicating a schema-level uniqueness test). It also tells us it's likely a `marts` model, implying it's a cleaned, transformed, and aggregated table for business users.

`Got 847 results, expected 0.`: This is the most direct statement of the failure. The test found 847 things that violated the uniqueness constraint.

`select order\_id, count() as n from analytics.fct\_orders group by 1 having count() > 1`: This is the actual SQL query dbt ran to perform the test. This is incredibly valuable.

`Sample duplicate order\_id values: ORD-88421, ORD-90102, ORD-90102`: Concrete examples of the data problem.

3. Deconstruct the Test Query and its Implications

Query: `select order\_id, count() as n from analytics.fct\_orders group by 1 having count() > 1`

What it does: It finds all `order\_id` values in the `analytics.fct\_orders` table that appear more than once.

`Got 847 results`: This means the query returned 847 rows. Each row represents a distinct `order\_id` that has duplicates.

Conclusion: The `order\_id` column in `analytics.fct\_orders` is not unique, and there are 847 distinct `order\_id` values that have duplicate entries in the table.

4. Determine the "Root Cause" Category

Based on the test query and model name (`fct\_orders`):

`fct\_orders`: This naming convention (common in dbt) implies a "fact table" - a central table in a star schema, usually containing measures and foreign keys to dimension tables. Fact tables are typically built from transactional data.

`order\_id`: This is almost universally expected to be the primary key or a unique identifier for an individual order.

Uniqueness failure on a primary key candidate: This immediately points to data integrity issues. How did duplicate `order\_id`s get into a table where they should be unique?

Assumptions made here:

`fct\_orders` follows common Kimball dimensional modeling principles, where `order\_id` would ideally be unique for each order event.

The `order\_id` is intended to be a business-level unique identifier for an order.

5. Brainstorm Potential Root Causes (General

What would you verify before shipping this to production?

Agent response

2:26 AM - Responded in 5s

Before shipping the proposed fix (or any significant change) to production, I would follow a rigorous verification process. The goal is to ensure the fix correctly addresses the problem, doesn't introduce new issues, and is robust for the long term.

Here's a detailed list of what I would verify:

Verification Checklist Before Production Deployment

1. Code Review (for the dbt model changes)

Logic Correctness: Does the `ROW_NUMBER()` or `DISTINCT` logic correctly identify and eliminate undesired duplicates? Is the partitioning and ordering correct (`PARTITION BY order_id ORDER BY <some_timestamp_column> DESC`) to retain the "latest" or "most correct" record?

Performance: Will the added `ROW_NUMBER()` or `DISTINCT` significantly impact the model's build time, especially on large datasets? Are there any obvious performance bottlenecks (e.g., full table scans without indexes on large tables)?

Readability & Maintainability: Is the code clear, well-commented, and easy for other team members to understand and maintain?

Best Practices: Does it adhere to dbt and team-specific coding standards?

2. Local Development Environment Testing

Run `dbt compile`: Ensure there are no syntax errors and the SQL compiles correctly.

Run `dbt run --select fct_orders`: Build the model locally (using a small, representative dataset if possible).

Run `dbt test --select unique_fct_orders_order_id`: Verify the uniqueness test now passes on the local data.

Ad-hoc Query for Duplicates: Run the failing test query manually on the locally built `fct_orders` to confirm it returns 0 rows.

Data Integrity Check (Spot Checks): Manually inspect a few `order_id`'s that were previously duplicated in the source/staging. Confirm that only one record remains in the local `fct_orders` and that the correct record (e.g., latest, most complete) was retained.

3. Staging/UAT Environment Testing (Most Critical Step)

Data Volume & Characteristics: The staging environment should ideally mirror production data volume and characteristics as closely as possible, even if it's a recent snapshot.

Full `dbt run`: Deploy the changes to staging and perform a full `dbt run` (or at least `dbt run --select fct_orders` to build `fct_orders` and its downstream dependencies).

`dbt test` Suite: Run all dbt tests relevant to `fct_orders` and its downstream models.

Uniqueness Test: Crucially, `dbt test --select unique_fct_orders_order_id` must pass.

Other Tests on `fct_orders`: Ensure no other tests (e.g., `not_null`, `relationships`, custom data quality tests) on `fct_orders` or its columns have been inadvertently broken.

Tests on Downstream Models: Verify that models depending on `fct_orders` (e.g., aggregated dashboards, other fact tables) are not negatively impacted. Look for new test failures or unexpected data changes.

Data Validation (Quantitative):

Row Counts: Compare the row count of the new `fct_orders` in staging with the row count of the old `fct_orders` (if available, e.g., in a previous version or backup).

Engineering counterpart
2:26 AM - Responded in 0s

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

Agent response

2:26 AM - Responded in 5s

Okay, let's address the critical aspects of risks and rollback for this production change.

Top Risks

When implementing this fix, especially the ``ROW_NUMBER()`` or ``DISTINCT`` approach, the primary risks revolve around unintended data loss or corruption, and performance degradation.

1. Data Loss/Incorrect Data Retention (HIGH RISK):

Description: The most significant risk is that the deduplication logic might inadvertently remove legitimate data or retain the wrong version of a duplicated record. If the ``ORDER BY`` clause in ``ROW_NUMBER()`` is incorrect or insufficient (e.g., ``ORDER BY updated_at DESC`` but ``updated_at`` isn't always reliable), we might keep an older, less complete, or incorrect version of the order.

Impact: Reports become inaccurate, business decisions are made on faulty data, potential financial discrepancies, loss of user trust.

Mitigation:

Thorough testing in staging with production-like data, focusing on manual inspection of retained records for problematic ``order_id``'s.

Clear understanding of business rules for "which duplicate to keep."

Robust ``ORDER BY`` clause that unambiguously identifies the correct record (e.g., ``ORDER BY updated_at DESC, version_id DESC, created_at DESC``).

2. Performance Degradation (MEDIUM RISK):

Description: Applying ``ROW_NUMBER()`` or ``DISTINCT`` over a large fact table can be computationally intensive, especially if the underlying data warehouse struggles with window functions or large sorts. This could significantly increase the model's build time.

Impact: Delayed data availability, increased compute costs, potential for pipeline timeouts, and downstream dashboards/reports not refreshing on time.

Mitigation:

Performance testing in staging with production-scale data.

Consider adding appropriate clustering/partitioning or indexes on the ``order_id`` and the ``ORDER BY`` columns if supported by the data warehouse.

Monitoring build times before and after deployment.

Optimizing the SQL query (e.g., ensuring filtering happens before deduplication if possible).

3. Introduction of New Bugs/Test Failures (MEDIUM RISK):

Description: Any code change, even one intended to fix a problem, can introduce new, unforeseen issues. Modifying ``fct_orders`` could break downstream models or reports that rely on specific (even if previously incorrect) characteristics of the data.

Impact: Cascading failures in the data pipeline, broken reports, loss of user trust.

Mitigation:

Comprehensive test suite: Run all dbt tests (``dbt test``) on the modified model and its downstream dependencies.

Thorough UAT with business users on critical dashboards/reports.

Data diffing tools to compare output of the new model vs. old model for key metrics.

4. Incomplete Fix / Recurrence (LOW-MEDIUM RISK, if source not addressed):

Description: If the root cause is upstream (e.g., source system generating duplicates, or an issue in a preceding dbt model), simply deduplicating ``fct_orders`` might only be a band-aid. New duplicates could flow in again in the future if the source isn't cleaned up.

Impact: The problem recurs, requiring repeated manual intervention, loss of credibility for the data team.

Mitigation: This is why the "Prevent Recurrence" section is crucial. Even with the immediate fix, the investigation into the ultimate source of

Engineering counterpart**2:26 AM - Responded in 0s**

Thanks - duplicate root cause and fix direction are clear. Ship the immediate dedupe/uniqueness fix; schedule the longer guardrail separately. Review complete - no further response needed.