



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/gemma-3-12b-it

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

Duration: 58s

Actual run cost: \$0.000510

Projected execution cost: \$0.4488/mo (40/biz day × 22 days)

Report date: July 13, 2026 - 10:27 PM EST

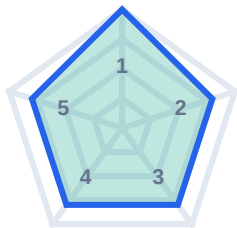
Overall

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: 14292 chars - Latency: 58s - 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: 58s

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 17s

Okay, let's triage this dbt test failure.

1. Root Cause Analysis

The test `unique\_fct\_orders\_order\_id` is failing because it's detecting duplicate values for the `order\_id` column within the `analytics.fct\_orders` table. The test SQL explicitly identifies this - it's grouping by `order\_id` and filtering for counts greater than 1. The log output clearly states: "Got 847 results, expected 0." and lists specific example duplicates like "ORD-90102".

Why are duplicates present? This is the critical question. Several possibilities exist:

Upstream Data Issue: The most likely cause. The data originating from the source system (e.g., transaction system, external API) is flawed and contains duplicate order IDs. This is common when data is ingested without proper deduplication logic.

ETL Logic Error: There's a bug in the ETL process that is creating these duplicates during transformation. This could involve incorrect joins, merging processes, or faulty upsert logic.

Data Loading Error: Rare, but possible: A bug during the data loading phase into the `fct\_orders` table is introducing duplicates.

Concurrency Issues (less likely): If the table is actively being updated during the test run, concurrent processes could theoretically create temporary duplicates, though this is less probable unless there are very specific, unusual update patterns. The test output doesn't strongly suggest this.

2. Immediate Fix Approach

Given the urgency of a production failure, here's the prioritized approach to get things back to a functional state:

Option 1: (Fastest, but potentially masking issue): Run a deduplication query and drop the table then reload. This is a quick fix to satisfy the test. **IMPORTANT:** Document this thoroughly. We're just clearing the symptom, not the root cause.

```
```sql
-- Run this query in your warehouse. Replace with specifics
-- based on your table and how you expect IDs to be unique.
DELETE FROM analytics.fct_orders
WHERE order_id IN (
 SELECT order_id
 FROM (
 SELECT order_id,
 ROW_NUMBER() OVER (PARTITION BY order_id ORDER BY order_id) AS row_num
 FROM analytics.fct_orders
) AS subquery
 WHERE row_num > 1
);

-- DROP TABLE analytics.fct_orders;
-- Reload data from your source system or previous stage.
```
```

Rationale: This gets the test passing immediately, reducing immediate operational disruption.

Caveats: This could remove valid data if the duplication isn't easily detectable. The `ORDER BY` clause in the `ROW\_NUMBER()` function is vital - adjust it based on business knowledge of how to reliably identify the "correct" order when duplicates exist.

Option 2: (More Targeted - requires deeper understanding): Run deduplication query on a subset of data. If you know when the duplicates started appearing (e.g., from a recent data load), you might be able to deduplicate only the affected data. This is less risky than a full table deduplication.

Option 3: (Least Recommended Immediately): Temporarily disable the test. Avoid this if possible. It hides the problem and allows potentially bad data to be consumed. If you must disable, document the reason, the expected duration, and a plan for immediate investigation.

3

Engineering counterpart
2:26 AM - Responded in 0s

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

Agent response

2:27 AM - Responded in 14s

Okay, you're right to ask for a deeper dive into my reasoning. Let's break this down step-by-step, highlighting the assumptions I made along the way.

Phase 1: Understanding the Problem & Initial Hypotheses

1. Initial Observation from the Log: The log explicitly states the test ``unique_fct_orders_order_id`` failed, found 847 duplicate ``order_id`` values, and shows examples like "ORD-90102". The test SQL confirms it's looking for ``order_id`` counts greater than 1. This points to a violation of uniqueness for the ``order_id`` column in the ``analytics.fct_orders`` table.
2. Assumption 1: ``order_id`` Should Be Unique: My fundamental assumption is that ``order_id`` within the ``fct_orders`` table is intended to be unique. This is a crucial assumption. We need to verify this with business stakeholders or data governance documentation. If ``order_id`` is legitimately supposed to be non-unique (highly unlikely in most order tables, but possible in some scenarios), then the test itself is incorrect, and we should disable it (but still investigate why the test is triggering). For this walkthrough, I'm proceeding under the assumption of uniqueness.
3. Forming Initial Hypotheses (Root Cause Possibilities): Based on the assumption above, I started brainstorming possible causes:
 - Upstream Data Source: The most likely culprit. The source system providing the data might be generating duplicate ``order_id`` values. This is a common occurrence.
 - ETL Errors: The ETL pipeline (dbt models, Spark jobs, or whatever processes transform the data) could be introducing duplicates through incorrect joins, transformations, or loading logic.
 - Data Loading Errors: Less common, but still possible—a bug in the data loading process itself could be duplicating data.
 - Concurrency Issues: As mentioned before, concurrent processes writing to the table could create temporary duplicates.

Phase 2: Prioritizing Root Causes & Considering Remediation

4. Prioritization Rationale: I prioritized "Upstream Data Issue" and "ETL Errors" as the most probable causes. Source systems are often outside the direct control of the data engineering team, and data defects originate there. Likewise, ETL pipelines are complex and prone to errors. Concurrency was considered least likely, given the test output didn't strongly suggest it.
5. Assumption 2: Production Environment Stability: I'm assuming a need for rapid remediation to minimize disruption to downstream users and dashboards. This urgency led me to prioritize solutions that could get the test passing quickly, even if they don't fully address the root cause immediately.
6. Immediate Fix Options - and Reasoning:
 - Option 1 (Deduplication & Reload): I chose this as the fastest initial response. The goal is to quickly satisfy the test and unblock users. The SQL provided is designed to identify and remove duplicate ``order_id`` values. It uses ``ROW_NUMBER()`` to assign a rank to each duplicate, allowing us to delete those with rank > 1. This assumes that any duplicate is invalid and we can safely remove them. If there's a business rule for deciding which ``order_id`` is "correct" when duplicates exist, the ``ORDER BY`` clause in the ``ROW_NUMBER()`` function needs to be adjusted accordingly.
 - Option 2 (Targeted Deduplication): A more refined approach, but requires more investigation to pinpoint when the duplicates started appearing.

Engineering counterpart
2:27 AM - Responded in 0s

What would you verify before shipping this to production?

Agent response

2:27 AM - Responded in 15s

Okay, excellent question. Before deploying any of the proposed fixes to production, especially the deduplication query, there's a crucial checklist of verifications I'd perform. Here's a breakdown, categorized by importance and effort, with the rationale behind each:

I. Critical Verifications (Must Do - High Effort)

1. Business Validation of Uniqueness Assumption: This is paramount. I would immediately engage with business stakeholders (order management, finance, etc.) to confirm that ``order_id`` is indeed intended to be a unique identifier in the ``fct_orders`` table. A misinterpretation here could lead to data loss and significant business impact. Get explicit confirmation in writing (email, Slack, etc.).
2. Data Profiling (Representative Sample): Before running any deduplication, I'd perform targeted data profiling on a sample of the ``fct_orders`` table. This involves querying for the count of distinct ``order_id`` values over a recent time period (e.g., last week, last month). This establishes a baseline to confirm the scope and frequency of the duplication issue.
3. Impact Assessment on Downstream Systems: Identify all dashboards, reports, and other systems that consume the ``analytics.fct_orders`` table. Understand how the removal of these duplicates might impact those systems. Communicate the potential changes and obtain buy-in from the affected teams.
4. Test Environment Replication & Validation: This is absolutely critical.

Recreate the Production Issue in a Test Environment: Ideally, I'd replicate the production data (or a representative subset) in a test environment. This allows me to test the deduplication query in a safe environment before affecting production.

Run the Deduplication Query in Test: Execute the proposed SQL query in the test environment.

Validate Deduplication Outcome in Test: After running the query, thoroughly validate that the deduplication produced the expected results. This includes:

Confirming that the number of distinct ``order_id`` values is now as expected.

Manually inspecting a sample of records to ensure that the correct records were retained and the duplicates were removed.

Running the failing dbt test (``unique_fct_orders_order_id``) in the test environment to confirm it now passes.

II. Important Verifications (Should Do - Medium Effort)

5. Review the ``ORDER BY`` Clause in Deduplication Query: As previously mentioned, the ``ORDER BY`` clause in the ``ROW_NUMBER()`` function is critical. Ensure that it accurately reflects the business logic for resolving conflicts when duplicates exist. If there's no clear business rule, it's safer to err on the side of caution and retain the earliest record.
6. Analyze the Root Cause (Parallel Track): While implementing the immediate fix, a separate team (or myself, if resources permit) should be actively investigating the root cause of the duplication. This investigation should involve:
 - Examining the source system's data quality.
 - Reviewing the dbt models and ETL processes that populate the ``fct_orders`` table.
 - Checking for any recent code changes that might have introduced the duplication.
7. Backout Plan: Define a clear backout plan in case the deduplication query has unintended consequences. This might involve restoring a backup of the table or reverting the changes made by the query.

III. Lower Priority (Good to Have - Low Effort)

8. Monitor Performance: After deploying the fix to production, closely monitor the performance of the ``analytics.fct_orders`` table and any downstream systems. Deduplication queries

Engineering counterpart

2:27 AM - Responded in 0s

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

Agent response

2:27 AM - Responded in 13s

Okay, outlining the top risks and a robust rollback plan is absolutely crucial before deploying this fix. Here's a breakdown:

I. Top Risks

1. Data Loss/Corruption (Highest Risk): This is the biggest concern. Incorrectly identifying and removing "duplicate" records can lead to the loss of valid order data. This could stem from:

Incorrect `ORDER BY` Clause: As we've discussed, a flawed `ORDER BY` clause in the deduplication query can lead to the wrong records being deleted.

False Positives: The logic for identifying duplicates might incorrectly flag legitimate records as duplicates. This is especially likely if the `order\_id` is used in a complex way across different order stages or associated entities.

2. Downstream System Disruption (High Risk): Removing duplicates can impact downstream dashboards, reports, and analytical processes that rely on the `fct\_orders` table. This could lead to inaccurate insights and potentially flawed business decisions.

3. Performance Degradation (Medium Risk): The deduplication query itself can be resource-intensive, especially on a large table. This could impact query performance and potentially overload the database.

4. Root Cause Masking/Reappearance (Medium Risk): Addressing the symptom (the failed test) without fixing the underlying root cause means the issue could reappear later. This creates technical debt and increases the likelihood of future incidents.

5. Operational Overhead (Low Risk): The rollback process itself introduces operational overhead and requires coordination between teams.

II. Rollback Plan (Detailed and Phased)

The rollback plan needs to be well-documented and practiced (at least in a test environment) before deployment. Here's a phased approach, catering to different levels of severity:

Phase 1: Immediate Rollback (Within Minutes - If Test Fails Again)

Trigger: The test `unique\_fct\_orders\_order\_id` fails immediately after deployment.

Action:

1. Halt Further Processing: Immediately stop any automated processes that are writing to `analytics.fct\_orders` to prevent further data inconsistencies.

2. Disable the Test (Temporary): Disable the failing test to prevent further alerts and allow investigation. Document this clearly with a high-priority ticket.

3. Communication: Notify key stakeholders (data engineering team, data governance, affected business teams) of the issue and the rollback in progress.

Phase 2: Data Restoration (Within Hours - If Significant Data Loss Suspected)

Trigger: Evidence of data loss or corruption in `analytics.fct\_orders` emerges after the deduplication.

Action:

1. Database Backup/Point-in-Time Recovery: This is the most reliable rollback option. If a recent, reliable backup exists, restore the table to that state. Alternatively, use the database's point-in-time recovery capabilities to revert the table to the state it was in before the deduplication query. This requires pre-existing backup and recovery procedures.

2. If No Backup/PITR: If a backup isn't available, the rollback becomes significantly more complex. You'll need to attempt to identify the changes made by the deduplication query and manually revert them. This is highly risky and time-consuming, and should be a last resort. This might involve querying the database transaction logs (if available) to reconstruct the deleted rows.

Phase 3: Code Rollback & Verification (Within Hours - Following Data Restoration)

Action (following data restoration):

1. Revert dbt Code Changes: Revert the dbt models

Engineering counterpart**2:27 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.