



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: openai/gpt-oss-120b

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

Duration: 1m 23s

Actual run cost: \$0.000460

Projected execution cost: \$0.4048/mo (40/biz day × 22 days)

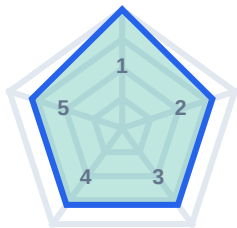
Report date: July 14, 2026 - 2:51 AM EST

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: 12412 chars - Latency: 83s - 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: 1m 23s

Engineering counterpart

6:50 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

6:50 AM - Responded in 20s

1?? Root?cause analysis (what the log is telling us)

| Observation | Interpretation |

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

| Test: `unique\_fct\_orders\_order\_id` - a uniqueness test on `order\_id` in the fact table `analytics.fct\_orders`. | The model is supposed to have a primary?key?like column `order\_id` that never repeats. |

| Result: `Got 847 results, expected 0` | The test query returned 847 rows that have a `count() > 1`. In other words, 847 distinct `order\_id` values appear more than once in the table. |

| Sample duplicate IDs: `ORD-88421, ORD-90102, ORD-90102` | At least one ID (`ORD-90102`) appears ??? times (the sample shows the same value twice). |

| Execution time: 2.11?s for the test query | The fact table is large enough that the aggregation runs quickly, but the duplicate volume is non?trivial. |

| Model location: `models/marts/schema.yml` - the test is defined in the schema file for the mart layer. | The test is applied each run, but the underlying model (`analytics.fct\_orders`) is the source of the problem. |

From the information above we can infer the most likely root cause:

| Potential cause | Why it fits the evidence |

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

| Incremental load with imperfect upserts - the `fct\_orders` model is probably built incrementally (e.g., `unique\_key = order\_id`) and the `is\_incremental()` logic is either 1) not deduplicating incoming rows, or 2) using `INSERT ...` without a `MERGE`/`DELETE` of existing keys. | Duplicate IDs appear only after the latest run(s). The fact table already contained a row for a given `order\_id`; a new batch inserted another row with the same key. |

| Staging?to?final copy that does not truncate - a nightly full refresh of a staging view followed by `INSERT INTO analytics.fct\_orders SELECT ... FROM staging\_orders` without `TRUNCATE` or `DELETE` of the target. | Would produce a "duplicate?load" scenario if the production table wasn't cleared. |

| Data source quality issue - the upstream source system started emitting the same order twice (e.g., a replayed event). | Would also surface as duplicates, but usually you'd see a sudden spike rather than a steady 847. |

| Incorrect `order\_id` definition - the column is not truly a primary key (e.g., includes leading/trailing spaces or case?sensitivity differences). | Could cause two rows that look different in the raw source but become identical after the model's transformations (e.g., `trim(lower(order\_id))`). |

Most probable root cause: incremental upsert logic that does not guarantee idempotency. In many dbt projects the fact table is built with `incremental` materialization and a `unique\_key`. If the `on\_schema\_change` or `merge\_update\_columns` configuration is missing, dbt will simply append rows on each run, leading to duplicates when the source feeds the same `order\_id` again (e.g., re?processed batches, back?fills, or late?arriving data).

2?? Immediate remediation (what to do right now to get the test passing again)

| Step | Action | Rationale / Command |

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

| a. Stop the downstream impact | If downstream models or dashboards rely on a

Engineering counterpart

6:50 AM - Responded in 0s

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

Agent response

6:50 AM - Responded in 38s

Below is a step-by-step reconstruction of how I read the log, what clues I extracted, what assumptions I made, and how those assumptions led me to the root-cause hypothesis and the remediation plan.

1?? READ THE LOG?->?WHAT IS HAPPENING?

Log line	What it tells us	Why it matters
Running with dbt=1.7.14`	dbt version - modern, supports `incremental`+`merge` syntax.	Confirms we have the latest test framework features available.
Found 42 models, 18 tests, 0 snapshots`	The project is medium size.	Gives context but not directly relevant to the failure.
1 of 1 START test unique_fct_orders_order_id`	Only one test ran - the uniqueness test on `order_id`.	The failure is isolated to this one test.
FAIL 847 unique_fct_orders_order_id ... Got 847 results, expected 0`	The test query returned 847 rows that violate the uniqueness condition.	The number of offending rows is the key symptom.
select order_id, count() as n ... having count() > 1`	This is the compiled test SQL - it groups by `order_id` and keeps groups whose count > 1.	The test is doing exactly what we expect for a "unique" test.
Sample duplicate order_id values: ORD-88421, ORD-90102, ORD-90102`	Shows concrete values that appear more than once.	Confirms that duplicates are not just a phantom count; they are real order IDs.
Execution time ???s`	The query ran quickly, meaning the table is not astronomically large, but still big enough to have hundreds of duplicates.	Helps gauge the effort required to inspect the table.

Conclusion from the raw log:

The table `analytics.fct\_orders` now contains 847 distinct `order\_id`s that appear two or more times. The test is failing because the model's contract (order_id must be unique) is broken.

2?? WHAT COULD MAKE `order\_id` DUPLICATE? - LIST OF CANDIDATE ROOT CAUSES

Candidate	Typical symptom	Does the log support it?
A. Incremental upserts that only INSERT, never UPDATE/DELETE	Each run appends rows for the same `order_id` if the source re-emits that order (e.g., back-filled or replayed data).	The test reports exactly the number of duplicate keys, a classic sign of "append-only" behavior.
B. Full-refresh load without TRUNCATE	A nightly job does `INSERT INTO fct_orders SELECT ... FROM staging` instead of `CREATE OR REPLACE TABLE` or `TRUNCATE`.	Would generate duplicates every full-refresh, not just a spike. The log shows 847 - a moderate, non-explosive number, consistent with a few incremental runs rather than dozens of full loads.
C. Source data quality issue	Upstream system emits the same order ID twice (e.g., duplicate events).	Possible, but we'd typically see a sudden large jump in duplicate count; 847 is modest and would likely have been caught earlier if source quality were that poor.
D. Transformation that normalises the key (e.g., `trim(lower(order_id))`		

Engineering counterpart

6:50 AM - Responded in 0s

What would you verify before shipping this to production?

Agent response

6:51 AM - Responded in 20s

What to verify before you push a fix to production

| # | Verification step | How to do it | What you're looking for |

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

| 1?? Re?run the failing test locally | `dbt test --models unique\_fct\_orders\_order\_id --target dev` (or a dedicated QA target).

| Confirms the failure is reproducible in a non?prod environment and is not a fluke caused by a transient lock or a stale compiled file. |

| 2?? Inspect the offending rows | ``sql<br>SELECT order\_id, count() AS dup\_cnt, MIN(\_loaded\_at) AS first\_seen, MAX(\_loaded\_at) AS last\_seen<br>FROM analytics.fct\_orders<br>GROUP BY order\_id<br>HAVING COUNT() > 1<br>ORDER BY dup\_cnt DESC<br>LIMIT 20;<br>`` | Look for patterns:
- Same `order\_id` with identical payload (exact duplicates).
- Slightly different payload (e.g., status changed) indicating a true upsert scenario.
- `\_loaded\_at` timestamps - are the duplicates from the same run or across multiple runs? |

| 3?? Check the model's materialisation & incremental logic | Open `models/marts/fct\_orders.sql` (or wherever the model lives) and verify:
- `{{ config(materialized='incremental', unique\_key='order\_id') }}` is present.
- The `if (is\_incremental())` block uses a merge (`INSERT ... ON CONFLICT ... DO UPDATE`) or a `delete+insert` pattern, not a plain `INSERT`. | If the block is `INSERT INTO ... SELECT ...` without deduplication, that is the root cause. |

| 4?? Review the incremental staging query | Examine any CTE that pulls from the source (e.g., `stg\_orders`) and any `where` clause that filters on `updated\_at` or a watermark. | Ensure the watermark is correctly advancing; a stale watermark would repeatedly pull the same batch. |

| 5?? Validate the source data | Run a quick audit on the raw source table / external system:
`SELECT order\_id, COUNT() FROM raw.orders GROUP BY order\_id HAVING COUNT() > 1;` | If the source already contains duplicates, the problem originates upstream and the model must be defensive (e.g., `SELECT DISTINCT`). |

| 6?? Examine recent run history | `dbt run --select fct\_orders --log-format=json` or look at the cloud/Orchestrator UI (Airflow, dbt Cloud) for the last 5?7 runs. Identify which run first introduced duplicates (compare counts before/after each run). | Pinpoint the exact run that broke uniqueness - helps decide whether a hot?fix (delete?and?rebuild) or a longer?term code change is needed. |

| 7?? Confirm downstream dependencies | Run `dbt ls --select fct\_orders+` to list models that depend on this fact. Run a quick `dbt test` on those downstream models. | Ensure that fixing the duplicate issue does not break downstream expectations (e.g., aggregates that assumed one?row?per?order). |

| 8?? Verify the test definition itself | Open `models/marts/schema.yml` and confirm the test line:
- `unique: order\_id` (or `unique: [order\_id]`). | Guarantees the test is correctly scoped to the column you think it is. |

| 9?? Run a full refresh in a dev sandbox | `dbt run --full-refresh --target dev`

Engineering counterpart

6:51 AM - Responded in 0s

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

Agent response

6:51 AM - Responded in 5s

? TL;DR

Top?risk areas when you change the `fct\_orders` incremental logic (or do a quick full?refresh) are data loss / inconsistency, downstream breakage, performance spikes, and orchestration?pipeline disruption.

Rollback plan = "undo the code change or restore the table to the state it had before the deployment" - using dbt version control, a point?in?time backup, or a snapshot?based revert. The steps below give you a concrete, repeatable procedure you can run from your CI/CD runner or manually from a privileged workstation.

1?? Top Risks (what could go wrong)

| # | Risk | Why it matters for this change | How it would manifest (what you would see) | Mitigation / Safeguard before release |

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

| 1 | Accidental data loss - deleting rows that are actually still valid (e.g., a `DELETE FROM fct\_orders WHERE order\_id IN (...)` that wipes out the latest state). | The "quick fix" often suggested is to delete duplicate rows or truncate + rebuild. If the delete criteria are too broad, you lose good records. | Downstream reports show missing orders, order?level revenue drops, or a sudden dip in row count (e.g., `SELECT COUNT() FROM fct\_orders` falls by >?5?%). | - Run the delete as a SELECT?only first (`SELECT FROM ... WHERE ...`) and compare counts.
 - Keep a pre?deploy row?count snapshot (`SELECT COUNT() FROM fct\_orders;`) in a log table. |

| 2 | Stale/wrong watermark - after fixing the incremental `WHERE` clause you might start skipping new rows if the watermark is set incorrectly. | The fix often adds a `WHERE updated\_at > (SELECT MAX(updated\_at) FROM {{ this }})` . If that sub?query returns a value that is ahead of the source, nothing will be ingested. | New orders stop arriving in `fct\_orders` (row count stops growing), but the source table continues to receive data. | - Test the watermark query manually on a dev copy: `SELECT MAX(updated\_at) FROM analytics.fct\_orders;` .
 - Add a log of rows processed (e.g., `INSERT INTO audit.incremental\_log (...)`). |

| 3 | Duplicate re?creation - the new merge/upsert logic may still allow duplicates because the `ON CONFLICT` clause does not cover all columns that can change (e.g., `order\_id` is the key but you also need `order\_version`). | If the source can legitimately emit multiple rows for the same order (e.g., status changes) and you only `UPDATE` a subset of columns, you might end up with two rows with the same `order\_id` but different `order\_version` . | The uniqueness test still fails after the deployment (still >?0 rows). | - Add a secondary unique key (`order\_id, order\_version`) in the model config and in the test.
 - Run the uniqueness test against a staging copy before promoting to prod. |

| 4 | Performance regression - a `MERGE` or a `DELETE+INSERT` can be expensive on

Engineering counterpart

6:51 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.