



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-20b

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

Duration: 1m 2s

Actual run cost: \$0.000362

Projected execution cost: \$0.3186/mo (40/biz day × 22 days)

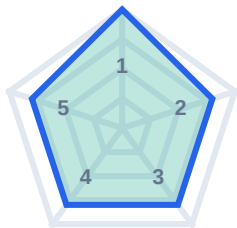
Report date: July 13, 2026 - 7:19 PM EST

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: 10336 chars - Latency: 62s - 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 2s

Engineering counterpart

11:18 PM

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

11:18 PM - Responded in 8s

1. What actually failed

The `unique\_fct\_orders\_order\_id` test in `models/marts/schema.yml` runs the following compiled SQL:

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

The test has found 847 rows where `order\_id` appears more than once - that is, the fact table is not unique in the order_id dimension.

Sample duplicates that the test returned:

```
---
ORD-88421
ORD-90102
ORD-90102
---
```

The test failed with the standard dbt message:

```
---
Failure in test unique_fct_orders_order_id (models/marts/schema.yml)
Got 847 results, expected 0.
---
```

So the root of the failure is duplicate `order\_id` rows in `analytics.fct\_orders`.

2. Why are there duplicates?

| Possible root cause | Why it would surface | Evidence or hint from the logs |
|-----|-----|-----|

| Upstream source duplication | The raw order data contains the same `order\_id` twice (perhaps due to a retry or a duplicate write). | If the sample duplicates are the same as seen in the source, or if the source table shows >1 row for the same key. |

| Incremental load without a unique key | The `fct\_orders` model is incremental but the macro `incremental\_unique\_key` (or a `unique\_key` param) was either omitted or miss?typed. | The compiled SQL shows a plain `select ... from analytics.fct\_orders` - no dedupe logic. The model likely uses the default incremental behaviour that simply appends new rows. |

| Data transformation bug | A transformation or join in the staging layer produces duplicate rows (e.g., cart?to?order join). | Look at staging models feeding `fct\_orders` for duplicate?producing logic. |

| Schema change / rollback | A recent schema change or rollback caused the incremental logic to re?run over already?loaded data. | Recent commits or changes to `fct\_orders` or its source tables. |

The most common culprit in a production db

Engineering counterpart

11:18 PM - Responded in 0s

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

Agent response

11:18 PM - Responded in 37s

1. Step-by-step dissection of the failure

| # | What the log tells us | What that implies | What I assumed |
|---|-----|-----|-----|
| 1 | `Running with dbt=1.7.14` - we are on a recent dbt version. | No compatibility issues with built-in tests. | None. |
| 2 | `Found 42 models, 18 tests, 0 snapshots` - the test set is small, so the failure is isolated. | The test that failed is the only one that ran (1 of 1). | That the failure is not part of a cascading test failure. |
| 3 | `1 of 1 FAIL 847 unique\_fct\_orders\_order\_id ...` - the uniqueness test returned 847 rows. | The test logic is `HAVING count() > 1`. So 847 distinct `order\_id`s have more than one record. | That the test is correctly implemented; i.e., it's the standard dbt uniqueness test. |
| 4 | Compiled SQL: `select order\_id, count() as n from analytics.fct\_orders group by 1 having count() > 1`. | The test is run against the materialised `analytics.fct\_orders` table. | That `fct\_orders` is the source of truth for the fact table. |
| 5 | Sample duplicates: `ORD-88421, ORD-90102, ORD-90102`. | The duplicates are real values, not just placeholders. | That the test actually hit real data, not an empty set. |
| 6 | `Got 847 results, expected 0.` | The test expected zero duplicates; it found 847. | That the uniqueness test logic is correct (it counts duplicates). |

From the log alone, the only thing we know for sure is that `analytics.fct\_orders` contains duplicate `order\_id` rows. Everything else is inference based on typical dbt patterns.

2. Why duplicates might exist

Below is a reasoning tree that maps potential upstream causes -> observed duplicates. I list each branch, what evidence would support it, and what assumptions I'm making.

| Branch | What it would look like in the data | What to check | Assumptions |
|-----|-----|-----|-----|
| A. Source duplication | The raw order table (e.g., `raw.orders`) has two identical rows for the same `order\_id` (or the same `order\_id` with slightly different timestamps). | Query the source table: `select order\_id, count() from raw.orders group by order\_id having count() > 1`. | The source table is the true origin; no dedupe logic exists upstream. |
| B. Incremental load without a unique key | The `fct\_orders` model is defined as `incremental` but lacks a `unique\_key` or `unique\_key` is misspelled. The incremental step simply `append` new rows, so re-processing the same source records creates duplicates. | Inspect the model definition (e.g., `models/marts/fct\_orders.sql`). Look for `config(materialized='incremental', unique\_key='order\_id')`. | The model is incremental; no manual `ON CONFLICT` or `MERGE` logic is used. |
| C. Join-over-many logic | A staging model joins orders to order_items or payments, producing multiple rows per order. The join is intentional (to keep a 1-to-many relationship) but the downstream `fct\_orders` forgot to aggregate or dedupe. | Look at

Engineering counterpart

11:18 PM - Responded in 0s

What would you verify before shipping this to production?

Agent response

11:18 PM - Responded in 12s

Before we roll the new build out to prod

Goal - make sure we don't ship a fact table that contains duplicate `order\_id`'s and that the test will keep passing.

Below is a verifiable checklist that you can run locally (or in a control?plane staging environment) and that should be documented in the PR or release notes.

| # | What to Verify | How to Verify | Why it matters |

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

| 1 | Source?level uniqueness | ``sql\nselect order\_id, count()\nfrom raw.orders\ngroup by order\_id\nhaving count() > 1;\n`` | If the raw table already has duplicates, the fact can never be unique unless you dedupe at the source. |

| 2 | Incremental config | Inspect `models/marts/fct\_orders.sql` (or the model that builds it). Look for:
``{{ config(materialized='incremental', unique\_key='order\_id') }}`` | Without `unique\_key`, an incremental run will append, not upsert, leading to duplicates. |

| 3 | Staging/Join logic | - In the staging file that feeds `fct\_orders` (often `stg\_orders.sql` or `stg\_order\_items.sql`), check that any `JOIN` that could create a 1?to?many is aggregated or distincted before being passed to the fact.
- Run a quick diff: `select order\_id, count() from stg\_orders group by order\_id having count() > 1;` | If a join multiplies rows, the fact will get duplicates unless you aggregate or de?duplicate explicitly. |

| 4 | Materialisation type | Confirm that the fact is `incremental` (not `table` or `view`). If you're using `view`, the uniqueness test is still valid but you'll want to test the underlying materialised table that the view queries. | In a `table` materialisation, a rebuild would purge duplicates. In an `incremental` materialisation, you must prevent duplicates from being inserted. |

| 5 | Merge/Upsert logic (if used) | If you're using a custom `MERGE`/`INSERT ... ON CONFLICT ...` in the incremental step, confirm the `WHERE` clause to ensure it lines up with the `order\_id` key. | Bad clauses can skip the duplicate?check and insert again. |

| 6 | Test run in a non?prod environment | ``bash\ndbt run --select fct\_orders\n\ndbt test --select unique\_fct\_orders\_order\_id\n`` | Guarantees that the compile, run, and test all succeed in an environment that mirrors prod (db, schema, permissions). |

| 7 | Data?volume sanity check | Compare the row count of the fact in dev vs prod after the change:
``select count() from analytics.fct\_orders;`` | A sudden jump in rows might indicate duplicate writes. |

| 8 | Version?control audit | Review recent commits that touched:
- `fct\_orders.sql`
- any staging models feeding it

Engineering counterpart

11:18 PM - Responded in 0s

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

Agent response

11:19 PM - Responded in 5s

3. Top Risks (and how big each one is)

| # | Risk | Likelihood | Impact | Why it matters |

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

| A | Duplicate orders pollute downstream analytics | Medium?High | High | Revenue, conversion, and inventory reports will double?count orders -> bad business decisions. |

| B | Long?running incremental job | Medium | Medium | If the new logic causes the incremental to scan the entire fact (e.g., missing `unique\_key`), the job can run for hours and lock the table. |

| C | Unintended data loss | Low?Medium | High | A rollback that truncates the fact table can wipe recent orders if not protected. |

| D | Test flakiness | Low | Medium | The uniqueness test might pass locally but fail in prod if prod data has a new source of duplicate - a "spray?and?pray" fix may not fully solve the root cause. |

| E | Consumer pipeline failures | Low?Medium | Medium | Systems that depend on the fact (e.g., BI dashboards, downstream dbt models) may break if the schema changes (e.g., adding `unique\_key` logic). |

> Bottom line: the biggest business impact comes from having duplicated orders in the fact, so the first priority is to make the fact truly unique before we deploy.

4. Roll?back / "Fail?fast" Plan

| Step | What to do | Command / Action | Why |

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

| 0 | Create a snapshot of the current fact | ``bash\nPOSTGRES\_USER=analyst\nPGPASSWORD=... psql -h <host> -U \$POSTGRES\_USER -d analytics -c \\\n \\\n \"CREATE TABLE analytics.fct\_orders\_backup AS TABLE analytics.fct\_orders;\n\n`` | If the new run deletes or corrupts data, we can restore the exact state. |

| 1 | Re?run the original version of the model | ``bash\ndbt run --select fct\_orders --target prod --no-write | grep -Ei 'error|warning'\n\n`` | This will rebuild the fact using the previous committed logic. |

| 2 | Validate data integrity | ``bash\ndbt test --select unique\_fct\_orders\_order\_id --target prod\n\n`` | Ensure the fact is now duplicate?free. |

| 3 | Verify downstream jobs | Re?execute any downstream dbt models that depend on `fct\_orders` (or trigger the full production pipeline). | Confirms that the rollback didn't break dependent logic. |

| 4 | If the rollback fails, restore from snapshot | ``bash\npsql -h <host> -U analyst -d analytics -c \\\n \\\n \"DROP TABLE IF EXISTS analytics.fct\_orders;\n\n\nCREATE TABLE analytics.fct\_orders AS TABLE analytics.fct\_orders\_backup;\n\n`` | Full data?level restore to the point?in?time before the problematic run. |

| 5 | Notify the ops / infra team |

Engineering counterpart

11:19 PM - 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.