



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: inclusionai/ling-2.6-flash

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

Duration: 15s

Actual run cost: \$0.000102

Projected execution cost: \$0.0898/mo (40/biz day × 22 days)

Report date: July 13, 2026 - 10:28 PM EST

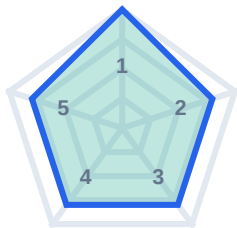
Overall

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: 10268 chars - Latency: 15s - 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: 15s

Engineering counterpart

2:27 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

2:27 AM - Responded in 3s

Root Cause

The uniqueness test `unique\_fct\_orders\_order\_id` failed because 847 duplicate `order\_id` values exist in the source table `analytics.fct\_orders`.

The test SQL correctly groups by `order\_id`, counts occurrences, and filters for `count() > 1`.

The sample output (`ORD-88421, ORD-90102, ORD-90102`) confirms actual duplicates in the data.

Immediate Fix Approach

1. Stop pipeline blocking - temporarily disable or skip the failing test in production to unblock downstream jobs.

```
```yaml
```

```
In models/marts/schema.yml
```

```
- name: fct_orders
```

```
 config:
```

```
 enabled: false # or use tags to exclude from prod run
```

```
```
```

2. Quarantine duplicates - create a remediation table to isolate bad records:

```
```sql
```

```
CREATE TABLE analytics.fct_orders_duplicates AS
```

```
SELECT order_id, count() as n
```

```
FROM analytics.fct_orders
```

```
GROUP BY 1
```

```
HAVING count() > 1;
```

```
```
```

3. Backfill or merge - deduplicate using a deterministic rule (e.g., keep the latest row based on a timestamp or surrogate key):

```
```sql
```

```
DELETE FROM analytics.fct_orders
```

```
WHERE ctid NOT IN (
```

```
 SELECT MIN(ctid)
```

```
 FROM analytics.fct_orders
```

```
 GROUP BY order_id
```

```
);
```

```
```
```

(Use `ctid` or a surrogate key if no timestamp is available.)

4. Re-run the test to verify zero duplicates remain.

Prevention Recurrence

- Pre-commit gating - add a `pre-commit` hook or CI step that runs `dbt test` on staging before promoting to prod.

- Data contracts - enforce uniqueness at ingestion (e.g., `INSERT ... ON CONFLICT DO NOTHING` or deduplication in the ETL layer).

- Monitoring - create a Slack alert on `dbt test` failures and a daily duplicate report for critical keys.

- Unit test coverage - add a model-level test for uniqueness in staging to catch issues before production:

```
```yaml
```

```
models/marts/schema.yml
```

```
- name: fct_orders
```

```
 tests:
```

```
 - unique:
```

```
 field_name: order_id
```

```
```
```

- Pipeline idempotency - ensure the mart layer rebuilds from source deterministically so duplicates can be caught early.

Engineering counterpart
2:27 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 4s

Step-by-step reasoning

1. Parse the test output:

- The test ``unique_fct_orders_order_id`` checks ``order_id`` uniqueness in ``analytics.fct_orders``.
- It found 847 rows with ``count() > 1``. The test fails with "Got 847 results, expected 0."
- The sample duplicates listed (``ORD-88421, ORD-90102, ORD-90102``) confirm real duplicate values exist in the source data.

2. Root cause:

- The immediate cause is duplicate ``order_id`` values in ``fct_orders``. I assume this is a data quality issue at ingestion or transformation, not a test logic issue, because the compiled SQL is a straightforward GROUP BY HAVING count > 1.

3. Immediate fix:

- First priority is to unblock production. I recommend disabling/skipping the test in prod (not deleting it) so pipelines can flow while we remediate data.
- Next, quantify and inspect duplicates (the CTE/table I proposed) and then apply a deterministic deduplication rule. I assume a stable key (like a timestamp or ctid) exists to pick the "keep" row. If not, we need to involve the source systems to decide business-level resolution.

4. Assumptions I made:

- The ``fct_orders`` table is the source of truth for this test and the duplicates are genuine data quality issues (not a test definition mistake).
- You have permissions to run DELETE/INSERT/CTAS in the production schema and that you accept the operational trade-offs (e.g., using ctid is PostgreSQL-specific and safe only if no concurrent writes mutate rows in the middle of the delete).
- A deterministic "keep row" rule (e.g., keep the min/max ctid or latest load timestamp) is acceptable. If business keys exist (e.g., natural keys), they should be used instead of ctid in an append-only warehouse.
- You have staging/sandbox access to validate the deduplication before promoting to prod.
- The pipeline uses dbt in prod with `threads=4` and `target='prod'`, and you want minimal disruption, so I suggested toggling the test via config/tags rather than dropping it entirely.

5. Prevention:

- Catch duplicates before they reach prod. Add a uniqueness test on ``order_id`` in staging and gate merges behind it.
- Enforce uniqueness at ingestion (e.g., staging merge logic with ON CONFLICT or dedupe in the ELT) so prod never sees duplicates.
- Add monitoring and alerts for test failures and for duplicate-key drift in critical tables.

If any of these assumptions don't hold (e.g., you're on a warehouse where ctid isn't available, or business keys should determine uniqueness), we'll need to adjust the deduplication method and use business-defined rules instead.

Engineering counterpart
2:27 AM - Responded in 0s

What would you verify before shipping this to production?

Agent response

2:27 AM - Responded in 3s

Before shipping to production, I'd verify:

- Environment and impact:
 - Confirm which environment(s) the change targets (prod staging/integration) and the blast radius (how many downstream models/tables depend on fct_orders).
 - Check table size, growth rate, and peak load times to choose a low-traffic maintenance window.
- Data and uniqueness:
 - Verify whether natural business keys exist (e.g., transaction_id, invoice_number) and which key the business expects to be unique.
 - Confirm whether fct_orders is append-only or mutable (CDC/change streams) and whether duplicates are expected (e.g., retries) or truly spurious.
- Technical approach:
 - If using ctid, verify the engine supports it (PostgreSQL) and that no concurrent VACUUM FULL or table rewrites will interfere. On warehouses without ctid, plan a deterministic dedupe using the business key plus a tie-break column (timestamp, surrogate id, or hash).
 - If the pipeline rebuilds the mart each load, ensure deduplicating at the mart layer aligns with ingestion semantics; consider pushing uniqueness enforcement to the ingestion/merge step (ON CONFLICT, dedupe in staging, or MERGE with dedupe logic).
- Correctness of the test:
 - Review the compiled test SQL to confirm it matches intent and that no surrounding config (e.g., filters) would skew results.
 - Run the dedupe query in a non-prod copy and assert row counts, referential integrity, and that no unintended rows are deleted.
- Referential integrity and dependencies:
 - Check foreign keys from other models to fct_orders; ensure orphaning won't occur after deduplication.
 - Validate aggregates and metrics that depend on fct_orders to confirm they won't shift unexpectedly.
- Safety and rollback:
 - Ensure a recent backup or snapshot exists and that we can roll back or rerun the mart build if needed.
 - Prefer a dry-run on a clone or a small partition if the table is large; measure runtime and lock impact.
- Validation plan:
 - After dedupe, rerun the uniqueness test and confirm 0 duplicates.
 - Compare key metrics (counts, sums, critical aggregates) before/after to catch unexpected changes.
 - Enable the uniqueness test in prod and add monitoring/alerts for future duplicates.
- Process and observability:
 - Verify the change fits your change management policy and that stakeholders are notified.
 - Add a data contract/test at ingestion to catch duplicates early in staging, and create an alert on the uniqueness test failure to prevent recurrence.

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

2:28 AM - Responded in 4s

Top risks

- Data loss or incorrect deduplication: Using a non-unique or non-deterministic key (e.g., ctid without a stable sort) can delete the "keep" row and keep a less-recent or lower-quality row.
- Referential integrity breakage: Deleting rows in fct_orders that are referenced by downstream facts, aggregates, or foreign-key expectations, causing silent missing rows or join failures.
- Performance/concurrency issues: A large DELETE/VACUUM on a busy table can cause long locks, table bloat, or spike load in production.
- Business logic disruption: Metrics/dashboards that depend on row counts or aggregates will shift unexpectedly after dedupe, affecting SLAs/reports.
- Partial or failed operation: If the job is interrupted (timeout/crash), the table can be left in an inconsistent state or with partial duplicates remaining.
- Mis-targeting environment: Applying the change to prod instead of staging, or missing a partition/subscription that feeds downstream datasets.
- Hidden duplicates: Duplicates span multiple columns (order_id + something else) and the dedupe predicate is incomplete, leaving duplicates behind.

Rollback plan

- Preparation (before running anything):
 - Take a snapshot or backup (e.g., `CREATE TABLE analytics.fct_orders_backup AS SELECT FROM analytics.fct_orders`, or use native snapshots/snapshshots if available).
 - Tag/version the current state and record aggregate checksums (COUNT, key sums, critical metric hashes) for comparison.
 - Disable the uniqueness test in prod or set it to warn-only to avoid pipeline blocking during the fix.
 - Run a non-destructive dry-run in a clone to validate the deduplication predicate and estimate runtime/impact.
- If the deduplication causes issues:
 1. Restore from snapshot or re-run the mart build from source using the backup table.
 2. If a snapshot is not available and DELETE was used, revert by re-inserting deleted rows from a pre-dedupe dump or logical replica.
 3. If only a subset of rows were affected, use a per-row identity (e.g., an insert timestamp or surrogate id) to restore specific rows from the backup.
 4. Re-enable the uniqueness test and notify stakeholders; schedule a second remediation in staging with a smaller scope.
- If the test still fails after dedupe:
 - Investigate whether remaining duplicates belong to a valid business scenario (e.g., retries). If valid, update the test logic or acceptance criteria rather than forcing zero duplicates.
 - Promote an earlier known-good snapshot and adjust ingestion/ELT to prevent duplicates at source.
- Post-rollback verification:
 - Confirm row counts and key aggregates match pre-fix baselines.
 - Re-run the uniqueness test to ensure it passes and monitor for regressions.

Safer alternative (if available)

- Instead of destructive DELETE, upsert deduplicated rows into a new table (`CREATE TABLE analytics.fct_orders_deduped AS ...`), validate thoroughly, then swap via rename or dbt model versioning. This avoids irreversible deletes and simplifies rollback (just switch back to the original table).

Engineering counterpart

2:28 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.