



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-4.1-mini

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

Duration: 33s

Actual run cost: \$0.004640

Projected execution cost: \$4.08/mo (40/biz day × 22 days)

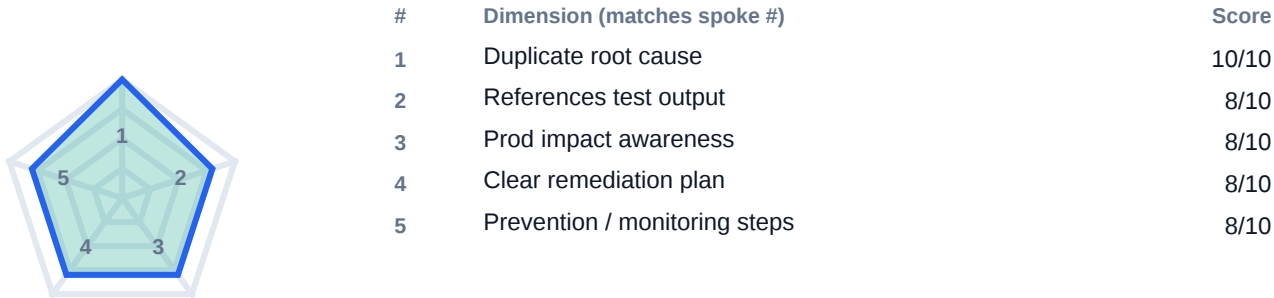
Report date: July 14, 2026 - 2:50 AM EST

Overall

8.4/10

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

CAPABILITY RUBRIC



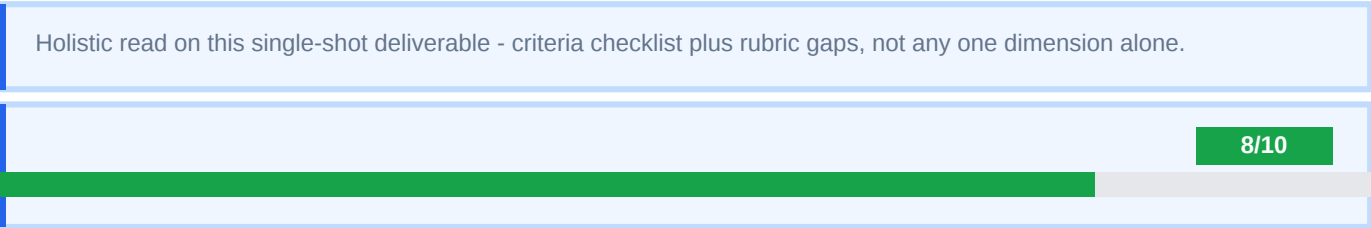
Spoke numbers run clockwise from the top; dimension names are listed in full in the table.

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



Telemetry

Run metrics

Mode: multi-turn - Agent msgs: 4 - Counterpart msgs: 5 - Output: 11727 chars - Latency: 33s - 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: 33s

Engineering counterpart

6:49 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

6:49 AM - Responded in 10s

Root Cause of Duplicates

The failure is in the uniqueness test `unique\_fct\_orders\_order\_id` on the `order\_id` column of the `analytics.fct\_orders` table. The test query identifies 847 duplicate `order\_id`s, which should be unique. The sample duplicates shown (`ORD-90102` appears twice) confirm repeated order IDs.

This indicates:

- The `fct\_orders` fact table contains multiple rows with the same `order\_id`.
- Since `order\_id` is expected to be a unique key (likely the primary business key for orders), having duplicates suggests:
 - Upstream data ingestion is allowing duplicates.
 - Or the transformation logic in dbt or upstream ETL is generating duplicate rows.
 - Possibly, a missing deduplication or incorrect join logic that multiplies rows.

Immediate Fix Approach

1. Identify full duplicate rows:

Run a query to see if duplicates are exact row duplicates or differ in other columns:

```
```sql
select
from analytics.fct_orders
where order_id in (
 select order_id
 from analytics.fct_orders
 group by order_id
 having count() > 1
)
order by order_id;
```
```

2. Remove duplicates to restore uniqueness:

Depending on business rules:

- If duplicates are exact copies, deduplicate by keeping one row.
- If duplicates differ, identify which row is correct based on timestamps or status.

3. Quick remediation:

- Run a deduplication query to create a temporary cleaned version of `fct\_orders` (e.g., keep max updated_at per order_id):

```
```sql
with ranked_orders as (
 select ,
 row_number() over (partition by order_id order by updated_at desc nulls last) as rn
 from analytics.fct_orders
)
select
from ranked_orders
where rn = 1;
```
```

- Replace or update `analytics.fct\_orders` accordingly after validation.

4. Fix the source of duplicates:

- Identify the upstream process or dbt model that creates `fct\_orders`.
- Validate joins, incremental loads, and sources to ensure no duplication.

How to Prevent Recurrence

1. Add uniqueness constraints or checks at the database level if supported (e.g., unique index on `order\_id`).
2. Enhance dbt models:
 - Add a pre-hook or model-level check to ensure no duplicates before inserting/updating `fct\_orders`.
 - Add deduplication logic inside the dbt model transformation for `fct\_orders`.
3. Add monitoring and alerts on duplicates on `order\_id`.
4. Review and test upstream ingestion processes for idempotency and uniqueness guarantees.
5. Add a nightly or pre-deployment dbt test run stage for uniqueness validation to catch duplicates earlier.
6. Document business rules about uniqueness and ensure all teams adhere to them.

Summary:

- Root cause: `order\_id` duplicates in `analytics.fct\_orders` due to missing deduplication or incorrect upstream logic.
- Immediate fix: Deduplicate `fct\_orders` rows by keeping the latest or authoritative entry per `order\_id`.
- Prevention: Add uniqueness constraints, dbt level checks/deduplication, and upstream data quality controls.

Engineering counterpart
6:49 AM - Responded in 0s

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

Agent response

6:49 AM - Responded in 10s

Certainly! Here's the detailed reasoning process I followed step-by-step, including key assumptions:

Step 1: Understand the Test Failure Output

- Test: `unique\_fct\_orders\_order\_id` - a uniqueness test on the `order\_id` field in the table `analytics.fct\_orders`.
- Failure: 847 duplicates found; expected zero.
- Test query:

```
```sql
select order_id, count() as n
from analytics.fct_orders
group by 1
having count() > 1
```
```
- Sample duplicates: `ORD-88421`, `ORD-90102` (appearing twice).

Assumption:

The `order\_id` is intended as a unique business key for orders in this fact table, so duplicates violate data integrity.

Step 2: Interpreting What the Duplicates Mean

- The uniqueness test aggregates by `order\_id` and counts rows, so duplicates exist at the `order\_id` level.
- They could be:
 - Exact duplicates: Same entire row repeated.
 - Partial duplicates: Same `order\_id` but different attribute values (e.g., different timestamps, status).
- The presence of duplicates suggests either:
 - The source data upstream contains duplicates.
 - The dbt model transformation logic allows duplicates (e.g., incorrect joins, missing deduplication).
 - Incremental loading or merging logic might be flawed, appending duplicates.

Assumption:

No unique database constraint on `order\_id` exists (common in analytical fact tables), so dbt tests catch these issues before they cause bigger problems.

Step 3: Planning the Immediate Fix

- To fix, we need to identify the nature of duplicates (exact or differing rows).
- Then deduplicate based on business rules, such as keeping the most recent row (using a timestamp column like `updated\_at`).
- Use SQL window functions with `row\_number()` to isolate one row per `order\_id`.
- Replace or update `analytics.fct\_orders` with the deduplicated data.

Assumption:

There is an `updated\_at` or equivalent timestamp column or some logic to identify which duplicate to keep.

Step 4: Long-term Prevention Strategy

- Add database unique constraints if possible, or at least uniqueness indexes on `order\_id`.
- Modify dbt model or upstream ETL to:
 - Include deduplication steps before inserting/updating `fct\_orders`.
 - Validate data quality and reject duplicate inserts.
- Implement monitoring and alerts to catch duplicates earlier.
- Schedule automated dbt tests to run frequently (e.g., nightly, pre-deploy).
- Document business rules regarding uniqueness for order IDs, and ensure all teams and processes comply.

Assumption:

There is flexibility to change dbt models and upstream processes, and monitoring infrastructure exists or can be added.

Summary of Assumptions

| Aspect | Assumption |
|-----------------------------|--|
| ----- | ----- |
| `order_id` uniqueness | Intended unique identifier for orders |
| Duplicate row nature | Could be exact or partial duplicates |
| Timestamp for deduplication | `updated_at` or similar field available |
| Database constraints | Unique constraints currently not enforced |
| Upstream processes | Have potential issues causing duplicates |
| Ability to remediate | Access to modify dbt models and upstream ETL |
| Monitoring capabilities | Can add tests and alerts into workflow |

This stepwise reasoning, backed by explicit assumptions, guides both immediate and preventive actions for the failed uniqueness test in production.

Engineering counterpart
6:49 AM - Responded in 0s

What would you verify before shipping this to production?

Agent response

6:49 AM - Responded in 6s

Before shipping the fix to production, I would verify the following to ensure the remediation is safe, effective, and won't cause regressions:

1. Verify Root Cause Analysis

- Confirm the duplicates are actually caused by the identified upstream process or dbt model.
- Validate whether duplicates are exact row copies or partial duplicates - run queries to analyze differences between duplicate rows.
- Confirm if any recent changes (deployments, data source changes) correlate with the spike in duplicates.

2. Validate Deduplication Logic

- Test the deduplication query on a representative subset of production data to ensure it keeps the correct records (e.g., latest `updated\_at`).
- Check that no valid data is being unintentionally dropped.
- Verify that resulting data has unique `order\_id`s.

3. End-to-End Testing in a Staging Environment

- Run full dbt pipeline including transformations and tests in a staging or QA environment with a recent production snapshot.
- Confirm that the uniqueness test now passes.
- Check that downstream models or reports depending on `fct\_orders` still produce expected results (no data loss or distortion).

4. Check Performance Impact

- Verify that deduplication or added transformations do not cause significant slowdowns.
- Confirm incremental or batch pipelines still run within acceptable SLA.

5. Backup and Rollback Plan

- Ensure there is a backup or snapshot of the current `analytics.fct\_orders` table before applying changes.
- Have a rollback plan ready if unexpected issues arise after deployment.

6. Communication and Documentation

- Notify downstream consumers of potential changes in `fct\_orders` data.
- Document the root cause, fix applied, and prevention measures for team knowledge.
- Update data quality monitoring dashboards or alert rules accordingly.

7. Post-Deploy Monitoring Plan

- Monitor duplicates immediately after deployment to confirm no recurrence.
- Check data freshness, row counts, and integrity metrics.
- Setup alerts to catch anomalies early.

In summary:

Before shipping, I would verify the root cause, test deduplication thoroughly in a staging environment, confirm no unintended data loss, ensure performance is acceptable, prepare rollback, communicate changes, and set up monitoring. This minimizes risk and improves confidence in the fix.

Engineering counterpart
6:49 AM - Responded in 0s

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

Agent response

6:50 AM - Responded in 6s

Top Risks

1. Incorrect Deduplication Logic

- Risk of deleting legitimate rows if the deduplication criteria (e.g., choosing latest `updated\_at`) is wrong or not applicable to all duplicates.
- Result: Data loss, incomplete order records, inaccurate reporting.

2. Incomplete Root Cause Fix

- Fixing only the symptom (`fct\_orders` duplicates) without addressing upstream data or ETL issues may cause duplicates to reappear quickly.
- Result: Recurrence of the test failure, ongoing data quality issues.

3. Performance Degradation

- Deduplication queries or new transformations might slow down the data pipeline or increase resource consumption.
- Result: Longer pipeline runtimes, missed SLAs, potential downstream delays.

4. Downstream Data Impact

- Removing duplicates might affect downstream models, dashboards, or business logic that assumed duplicates were valid or handled them in some way.
- Result: Broken reports or unexpected business decisions.

5. Deployment Mistakes

- Erroneous deployment (e.g., updating the wrong table, partially applying fix) can introduce inconsistencies or downtime.
- Result: Production data corruption or service disruption.

Rollback Plan

1. Pre-Deployment Backup

- Take a full backup or snapshot of `analytics.fct\_orders` before applying the fix. This could be a table copy or export to object storage.

2. Version Control/Release Management

- Use versioned dbt models and deployment process to quickly revert to the prior model version if needed.

3. Revert Data Changes

- If duplicates removal caused data loss or issues, restore the backup to revert `fct\_orders` to the last known good state.

4. Disable or Revert Pipeline Changes

- Roll back any dbt model changes or upstream ETL fixes that were deployed along with the deduplication logic.

5. Re-run dbt Tests and Validations

- After rollback, re-run uniqueness tests and other validations to confirm the system is back to the prior stable state.

6. Communicate Rollback

- Inform stakeholders and downstream teams immediately about the rollback and any impacts.

Summary:

The key risks involve data loss due to deduplication errors, recurrence if root cause is not fixed, and downstream impact.

The rollback plan centers on restoring backups, reverting model and pipeline changes, and re-validating data integrity swiftly to minimize disruption.

Engineering counterpart**6:50 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.