



Vantage RuntimeAI

Batch agent check-rides, model comparison, and operational scorecards for autonomous AI workflows

8.0/10

Overall score

Run summary

Scenario: dbt Test Failure

Model: mistralai/mistral-nemo

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

Duration: 50s

Actual run cost: \$0.000156

Projected execution cost: \$0.1373/mo (40/biz day × 22 days)

Report date: July 13, 2026 - 10:29 PM EST

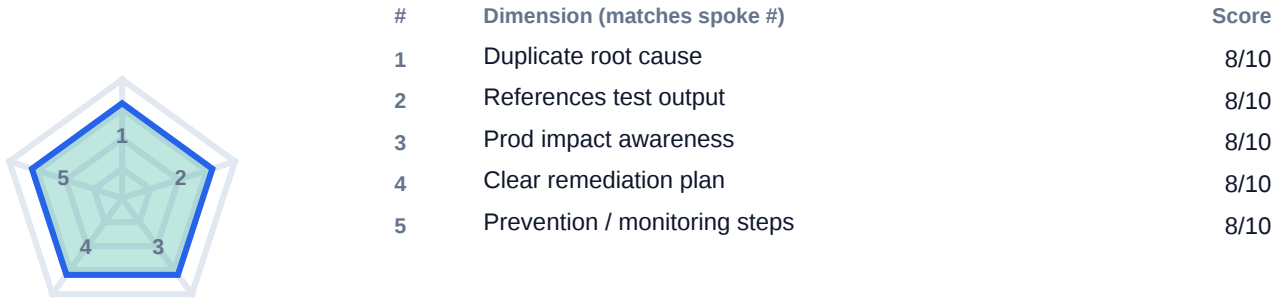
Overall

8.0/10

PASS - trust=high

Pass - score 8.0/10 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

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: 12008 chars - Latency: 50s - 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: 50s

Engineering counterpart

2:29 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

2:29 AM - Responded in 17s

Root cause:

The uniqueness test `unique\_fct\_orders\_order\_id` failed because the `order\_id` column in the `fct\_orders` table is not unique, as evidenced by the sample duplicate values provided in the test output (e.g., `ORD-90102` appears twice).

Immediate fix approach:

1. Identify the data source(s) contributing to the duplicate `order\_id` values.
2. Temporarily disable the failing test using the `failures` config in the test's definition to prevent it from blocking further runs (e.g., in your `models/marts/schema.yml` file):

```
```yaml
unique_fct_orders_order_id:
 enabled: false
 tests:
 - unique
```
```

3. Investigate and address the data quality issue at the source by either:
 - a. Correcting the data by removing the duplicate rows (ensuring that this doesn't lead to data loss or inconsistencies elsewhere).
 - b. Updating the `order\_id` generation logic to ensure uniqueness moving forward.
4. Manually run the failing test again to ensure the issue has been resolved.
5. Re-enable the test by setting `enabled: true`.

Prevent recurrence:

1. Data validation at source: Implement data validation rules at the source system to ensure `order\_id` uniqueness before data is inserted or updated in the `fct\_orders` table.
2. Unique constraint on the database: Add a unique constraint on the `order\_id` column in the `fct\_orders` table to enforce uniqueness at the database level.
3. Data profiling and monitoring: Regularly profile and monitor data quality in the `fct\_orders` table to quickly identify and address any potential data quality issues, including duplicates.
4. dbt tests and CI/CD: Ensure dbt tests, including the uniqueness test, are run as part of the CI/CD pipeline to catch data quality issues early and prevent them from reaching production.
5. Root cause analysis (RCA): Conduct an RCA to understand why duplicates occurred in the first place, and implement controls to prevent recurrence based on the findings.
6. Documentation and training: Document the data quality rules and ensure that relevant stakeholders are adequately trained on data quality expectations and the impacts of not adhering to them.

Engineering counterpart

2:29 AM - Responded in 0s

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

Agent response

2:29 AM - Responded in 11s

I'm glad you found the initial response helpful. Let's walk through the reasoning behind the root cause, immediate fix approach, and prevention steps, along with the assumptions made:

1. Root cause:

a. Assumption: The ``order_id`` column in the ``fct_orders`` table should be unique, as it's a common practice to make identifiers in fact tables unique.

b. Reasoning: The uniqueness test failed because there are duplicate values in the ``order_id`` column. This is evident from the test output, which states: "Got 847 results, expected 0" and provides sample duplicate values (``ORD-88421``, ``ORD-90102``, etc.).

c. Conclusion: The root cause is the presence of duplicate values in the ``order_id`` column of the ``fct_orders`` table.

2. Immediate fix approach:

a. Assumption: The data contributing to the duplicates can be identified and corrected or removed without causing data loss or inconsistencies elsewhere.

b. Reasoning:

- To temporarily disable the failing test, we use the ``failures`` config in the test's definition, as mentioned in the dbt documentation (<https://docs.getdbt.com/docs/build/test-configuration>).

- Addressing the data quality issue at the source involves either correcting the data or updating the ``order_id`` generation logic. The specific approach will depend on factors such as the data source, the ETL process, and the reason for the duplicates.

c. Conclusion: Temporarily disable the test, investigate and resolve the data quality issue at the source, manually run the test again, and re-enable it after confirming the issue is resolved.

3. Prevent recurrence:

a. Assumption: Implementing appropriate data validation, monitoring, and control measures can help prevent duplicates from occurring again.

b. Reasoning:

- Data validation at the source can help ensure ``order_id`` uniqueness before data is inserted or updated in the ``fct_orders`` table.

- Adding a unique constraint on the ``order_id`` column at the database level can enforce uniqueness and help prevent duplicates.

- Regularly profiling and monitoring data quality can help quickly identify and address potential data quality issues.

- Including dbt tests in the CI/CD pipeline can catch data quality issues early and prevent them from reaching production.

- Conducting an RCA can help understand the root cause of the duplicates and implement controls to prevent recurrence.

- Documenting data quality rules and training relevant stakeholders can ensure everyone is on the same page regarding data quality expectations.

c. Conclusion: Implementing a combination of data validation, data monitoring, and data control measures can help prevent duplicates from recurring in the ``order_id`` column.

Throughout this reasoning process, I've made assumptions based on common data warehousing practices and the information provided in the test output. Some assumptions might not hold true for your specific use case, so it's essential to adapt the suggestions to fit your specific environment and requirements.

What would you verify before shipping this to production?

Agent response

2:29 AM - Responded in 9s

Before shipping the data quality fix and the updated tests to production, verify the following to ensure a smooth deployment and minimize the risk of introducing new issues:

1. Data validation:

Ensure that the data correction or cleanup process has been successfully applied to the production environment.

Verify that all duplicate `order\_id` values have been resolved or removed without causing data loss or inconsistencies elsewhere.

Confirm that the updated data meets the business and data quality expectations.

2. dbt tests:

Manually run the re-enabled uniqueness test (`unique\_fct\_orders\_order\_id`) in the production environment to confirm that it passes without any errors or warnings.

Ensure that the test runs within an acceptable timeframe, as slow-running tests can impact the overall deployment process.

Validate that the test's configuration and settings are appropriate for the production environment (e.g., concurrency, resources, etc.).

3. Database schema changes:

If you've added a unique constraint on the `order\_id` column, verify that it has been successfully applied in the production environment.

Ensure that the unique constraint does not cause any data loading or ETL process issues due to existing duplicate values (if any).

Test the impact of the unique constraint on the data loading process, query performance, and any relevant business processes.

4. End-to-end data validation:

Validate that the data transformation, loading, and testing processes work correctly end-to-end in the production environment.

Verify that all downstream systems, reports, and dashboards that depend on the `fct\_orders` table continue to function correctly with the updated data and tests.

5. Performance testing:

Monitor the performance of the dbt tests and the overall data pipeline in the production environment to ensure they run within acceptable timeframes.

Verify that the data cleanup process and any related schema changes have not negatively impacted query performance or data loading times.

6. Documentation and communication:

Update relevant documentation, including data quality rules, test configurations, and any changes to the data pipeline or ETL processes.

Communicate the changes, their purpose, and the expected impact to all relevant stakeholders, such as data engineers, data analysts, and business users.

7. Rollback plan:

Have a rollback plan in place, in case the deployment introduces new issues or negatively impacts the production environment.

Ensure that you can quickly revert the changes if needed, and that the data and tests can be restored to their previous state.

By verifying these aspects, you can help ensure a successful deployment of the data quality fix and the updated tests to production, minimizing the risk of introducing new issues or negatively impacting the data pipeline and downstream systems.

Engineering counterpart
2:29 AM - Responded in 0s

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

Agent response

2:29 AM - Responded in 13s

Top risks and potential issues:

1. Data loss or inconsistency:

- Incorrectly identifying and removing duplicate rows may lead to data loss.
- Applying the wrong correction or update logic might result in data inconsistencies.

2. Data validation and test failures:

- Incomplete or incorrect data cleanup might cause other tests or downstream systems to fail.
- Changes to data or tests might introduce new failures in the data pipeline.

3. Performance degradation:

- Adding a unique constraint to the `order\_id` column may impact query performance or data loading times, especially if the column is frequently updated or searched.
- Slow-running tests or data cleanup processes might cause delays in the data pipeline.

4. Downtime and service disruption:

- Deploying changes during peak hours or without proper planning might result in downtime or service disruption for downstream systems or users.

5. Lack of stakeholder communication:

- Inadequate communication about the changes, their purpose, and expected impact might lead to confusion, concern, or resistance among stakeholders.

6. Incomplete or incorrect RCA:

- An inadequate or incorrect root cause analysis might result in recurring data quality issues or the introduction of new problems.

7. Unanticipated interactions with other systems or processes:

- Changes to the data pipeline or tests might introduce unforeseen issues when interacting with other systems, such as ETL tools, databases, or downstream applications.

Rollback plan:

In case the deployment introduces new issues, causes significant data quality problems, or negatively impacts production, have a rollback plan in place to quickly revert the changes and restore the previous state. Here's a rollback plan for the described scenario:

1. Data cleanup:

- If data has been incorrectly cleaned or removed, have a backup of the original data or a previously working state.
- Restore the original data or the backup to revert the data cleanup changes.
- Ensure that the restore process does not introduce new duplicates or data inconsistencies.

2. dbt tests:

- Revert the changes to the dbt tests by:
 - + Removing the unique constraint from the `order\_id` column (if added).
 - + Disabling or deleting the uniqueness test that was causing failures (`unique\_fct\_orders\_order\_id`).
 - + Re-enabling previously disabled tests.
- Restore the previous test configuration and settings.

3. Database schema changes:

- If a unique constraint on the `order\_id` column caused issues, remove it from the production environment.
- Ensure that the removal of the constraint does not reintroduce data quality problems.

4. Data pipeline and ETL processes:

- Temporarily revert the changes to the data pipeline or ETL processes if they are causing issues.
- Restore the previous version of the data pipeline or ETL processes, if available.

5. Communication and documentation:

- Communicate the rollback process to all relevant stakeholders.
- Update the documentation to reflect the rollback and any changes made during the process.

6. Monitoring and validation:

- Monitor the data pipeline and tests closely to ensure they have been successfully rolled back.
- Validate that the rollback has restored the data and tests to their previous working state.

To minimize the need for a rollback, thoroughly test and validate the changes in a development or staging environment before deploying them to production. Engage with stakeholders throughout the process to address any concerns and ensure that the changes meet their expectations and requirements. Having a well-defined rollback plan can help mitigate risks and provide a safety net in case things don't go as expected in production.

Engineering counterpart**2:29 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.