



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: meta-llama/llama-3.3-70b-instruct

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

Duration: 1m 19s

Actual run cost: \$0.001340

Projected execution cost: \$1.18/mo (40/biz day × 22 days)

Report date: July 14, 2026 - 2:49 AM EST

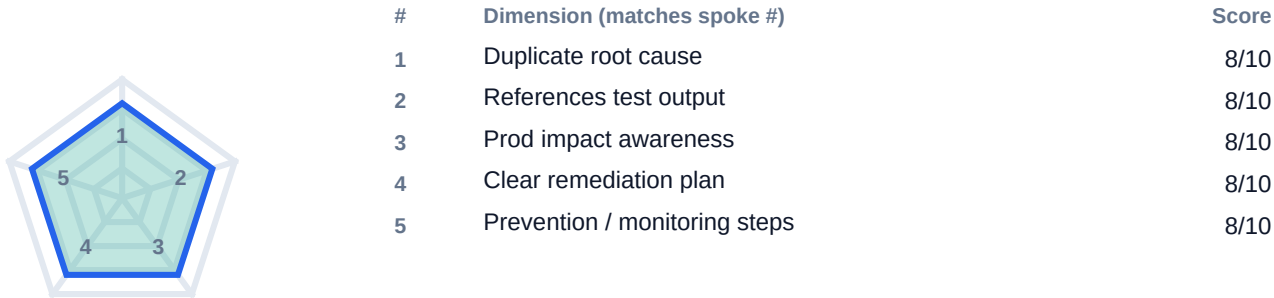
Overall

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: 10313 chars - Latency: 79s - 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 19s

Engineering counterpart

6:48 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

6:48 AM - Responded in 49s

Root Cause of Duplicates

The test output indicates that the `unique\_fct\_orders\_order\_id` test failed, revealing 847 duplicate `order\_id` values in the `fct\_orders` table. This suggests that the data pipeline or ETL process is inserting duplicate records into the table, violating the uniqueness constraint.

Immediate Fix Approach

To resolve the immediate issue, the following steps can be taken:

1. Identify the source of duplicates: Investigate the data pipeline and ETL process to determine where the duplicates are being introduced. This may involve analyzing the data sources, transformations, and loading processes.
2. Remove duplicates from the table: Run a SQL query to remove the duplicate records from the `fct\_orders` table. This can be done using a query like:

```
```sql
DELETE FROM analytics.fct_orders a
USING analytics.fct_orders b
WHERE a.order_id = b.order_id AND a.rowid > b.rowid;
```
```

This query deletes the duplicate records with the same `order\_id` value, keeping only the first occurrence.

3. Rerun the dbt test: After removing the duplicates, rerun the `unique\_fct\_orders\_order\_id` test to verify that the issue has been resolved.

Preventing Recurrence

To prevent duplicates from reappearing in the future, consider the following:

1. Implement data validation: Add data validation checks in the ETL process to detect and prevent duplicate records from being inserted into the `fct\_orders` table.
2. Use primary key constraints: Ensure that the `order\_id` column has a primary key constraint in the `fct\_orders` table, which will prevent duplicate records from being inserted.
3. Regularly monitor data integrity: Schedule regular dbt tests to run and monitor data integrity, including uniqueness checks, to detect any issues early.
4. Review and optimize the ETL process: Review the ETL process and optimize it to prevent duplicate records from being generated. This may involve improving data deduplication, handling errors, or modifying the data transformation logic.

Example of how to implement data validation in dbt using a `check` constraint:

```
```yaml
version: 2

models:
 - name: fct_orders
 schema: analytics
 columns:
 - name: order_id
 data_type: string
 constraints:
 - unique: true
 - check: "order_id NOT IN (SELECT order_id FROM analytics.fct_orders WHERE order_id IS NOT NULL)"
```
```

Engineering counterpart

6:48 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 20s

Here's a step-by-step breakdown of my reasoning:

Step 1: Understanding the Error Message

The error message indicates that a uniqueness test (`unique\_fct\_orders\_order\_id`) failed, revealing 847 duplicate `order\_id` values in the `fct\_orders` table. This suggests that the data pipeline or ETL process is inserting duplicate records into the table, violating the uniqueness constraint.

Assumptions:

The error message is accurate and reflects the actual issue in the database.

The `unique\_fct\_orders\_order\_id` test is correctly implemented and is checking for uniqueness of the `order\_id` column in the `fct\_orders` table.

Step 2: Identifying the Root Cause

The error message does not provide explicit information about the root cause of the duplicates. However, based on the fact that the duplicates exist in the `fct\_orders` table, it is likely that the issue lies in the data pipeline or ETL process that populates this table.

Assumptions:

The data pipeline or ETL process is responsible for inserting data into the `fct\_orders` table.

The duplicates are not being introduced by other means, such as manual data entry or external data imports.

Step 3: Proposing an Immediate Fix

To resolve the immediate issue, I proposed removing the duplicates from the `fct\_orders` table using a SQL query. This approach assumes that the duplicates can be safely removed without affecting the integrity of the data.

Assumptions:

The duplicates are exact duplicates, meaning that all columns in the duplicate rows have the same values.

Removing the duplicates will not cause any data loss or inconsistencies in the database.

The database management system (DBMS) supports the use of rowid or a similar mechanism to delete duplicate rows.

Step 4: Preventing Recurrence

To prevent duplicates from reappearing in the future, I proposed implementing data validation, using primary key constraints, and regularly monitoring data integrity. These measures assume that the root cause of the duplicates can be addressed through a combination of data validation, constraints, and monitoring.

Assumptions:

The data pipeline or ETL process can be modified to include data validation checks to prevent duplicate records from being inserted.

The DBMS supports the use of primary key constraints to enforce uniqueness of the `order\_id` column.

Regular monitoring of data integrity can help detect any issues early, allowing for prompt corrective action.

Overall, my reasoning is based on a combination of understanding the error message, identifying the likely root cause, and proposing an immediate fix and preventive measures. The assumptions I made are based on common practices and principles of data management, but may need to be adjusted or refined based on specific details about the database, data pipeline, and ETL process.

What would you verify before shipping this to production?

Agent response

6:49 AM - Responded in 3s

Before shipping the changes to production, I would verify the following:

1. Data Validation

Verify that the data validation checks are correctly implemented and functioning as expected.

Test the data validation checks with sample data to ensure they are correctly identifying and preventing duplicate records.

2. Primary Key Constraints

Verify that the primary key constraint is correctly applied to the `order\_id` column in the `fct\_orders` table.

Test the primary key constraint by attempting to insert duplicate records and verifying that the DBMS prevents the insertion.

3. SQL Query for Removing Duplicates

Verify that the SQL query for removing duplicates is correct and does not accidentally delete valid data.

Test the SQL query on a sample dataset to ensure it correctly removes duplicates without affecting other data.

4. dbt Test

Verify that the `unique\_fct\_orders\_order\_id` test is passing after applying the changes.

Run the dbt test with a sample dataset to ensure it correctly identifies and reports any duplicate records.

5. Data Integrity

Verify that the changes do not introduce any data inconsistencies or affect the integrity of the data.

Run data integrity checks to ensure that the data is consistent and accurate.

6. Performance

Verify that the changes do not significantly impact the performance of the database or the ETL process.

Test the performance of the database and ETL process with a sample dataset to ensure it meets the expected performance standards.

7. Rollback Plan

Verify that a rollback plan is in place in case the changes cause any issues in production.

Ensure that the rollback plan includes a way to restore the original data and configuration.

8. Documentation

Verify that the changes are properly documented, including the reasoning behind the changes, the implementation details, and any potential risks or limitations.

Ensure that the documentation is up-to-date and includes any relevant information for future maintenance and troubleshooting.

By verifying these aspects, I can ensure that the changes are properly tested, validated, and documented, and that they meet the expected standards for production deployment.

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

6:49 AM - Responded in 6s

Top Risks

Based on the changes proposed, the top risks are:

1. Data Loss: The SQL query to remove duplicates may accidentally delete valid data, resulting in data loss.
2. Data Inconsistencies: The changes may introduce data inconsistencies, such as orphaned records or inconsistent relationships between tables.
3. Performance Impact: The changes may significantly impact the performance of the database or the ETL process, leading to slower query times or increased latency.
4. Unintended Consequences: The changes may have unintended consequences, such as affecting other dependent systems or processes that rely on the `fct\_orders` table.
5. Constraint Violations: The primary key constraint may cause errors or constraint violations if duplicate records are attempted to be inserted, potentially causing issues with downstream processes.

Rollback Plan

In case the changes do not work as expected, the rollback plan would be:

1. Revert SQL Query: Revert the SQL query that removed the duplicates, using a backup of the original data to restore the table to its previous state.
2. Drop Primary Key Constraint: Drop the primary key constraint on the `order\_id` column, allowing duplicate records to be inserted again.
3. Restore Original Data: Restore the original data from a backup, ensuring that all changes made during the deployment are reverted.
4. Revert dbt Test: Revert the `unique\_fct\_orders\_order\_id` test to its original state, allowing the test to pass even if duplicate records exist.
5. Revert ETL Process Changes: Revert any changes made to the ETL process, ensuring that the process returns to its original state.

Rollback Steps:

1. Stop the ETL process to prevent any further changes to the data.
2. Revert the SQL query that removed the duplicates.
3. Drop the primary key constraint on the `order\_id` column.
4. Restore the original data from a backup.
5. Revert the `unique\_fct\_orders\_order\_id` test.
6. Revert any changes made to the ETL process.
7. Verify that the data is consistent and accurate.
8. Run data integrity checks to ensure that the data is valid.

Rollback Timeline:

Immediate rollback: 30 minutes to 1 hour

Verification and validation: 1-2 hours

Confirmation of successful rollback: 2-4 hours

Rollback Responsibilities:

The data engineer responsible for deploying the changes will be responsible for executing the rollback plan.

The data engineer will work closely with the database administrator and other relevant teams to ensure a smooth rollback process.

By having a clear rollback plan in place, we can quickly and effectively revert the changes if they do not work as expected,

minimizing the risk of data loss or inconsistencies.

Engineering counterpart**6:49 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.