



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: amazon/nova-micro-v1

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

Duration: 17s

Actual run cost: \$0.000406

Projected execution cost: \$0.3573/mo (40/biz day × 22 days)

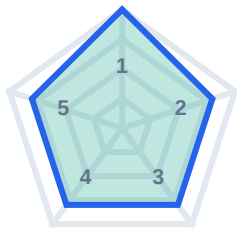
Report date: July 13, 2026 - 7:11 PM EST

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: 12820 chars - Latency: 17s - 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: 17s

Engineering counterpart

11:11 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

11:11 PM - Responded in 4s

Root Cause Analysis

The uniqueness test failed because the `analytics.fct\_orders` table contains multiple records with the same `order\_id`, violating the unique constraint enforced in the test. The log indicates that there are 847 duplicate `order\_id` entries in the table.

Immediate Fix Approach

To address this failure swiftly, we need to clean the data to ensure that each `order\_id` is unique.

1. Identify and flag the duplicates:

The identified duplicate `order\_id` values (`ORD-88421` and `ORD-90102`) need to be investigated further to determine the correct records.

2. Data Clean Up:

Once duplicates that should be removed are identified (e.g., based on additional data in the table like timestamps, user IDs, etc.), we can use a process such as:

```
```sql
CREATE TABLE analytics.fct_orders_clean AS
SELECT
 order_id,
 MAX(order_timestamp) AS order_timestamp,
 ... -- other required columns
FROM analytics.fct_orders
GROUP BY order_id
HAVING COUNT() = 1 OR MAX(order_timestamp) = MIN(order_timestamp);
```
```

3. Update References:

If there are any dependent records in other tables that reference `fct\_orders.order\_id`, those should be updated or cleaned to reflect the changes.

4. Test Rerun:

Once the clean-up is complete, rerun the `dbt test`:

```
```bash
dbt test
```
```

Preventative Measures

To avoid recurrences:

1. Add Data Quality Checks:

Add more robust data quality checks in your ETL pipeline before loading into `fct\_orders`. This can include:

```
```sql
SELECT
 order_id,
 COUNT()
FROM analytics.fct_orders
GROUP BY order_id
HAVING COUNT() > 1;
```
```

Failing the data load when duplicates are detected.

2. Implement Constraint Enforcements:

Enforce unique constraints directly in the database schema to prevent duplicate entries from being inserted:

```
```sql
```

```
ALTER TABLE analytics.fct_orders ADD CONSTRAINT unique_order_id UNIQUE (order_id);
```

```
```
```

3. Regular Audits:

Schedule regular data quality checks to identify and fix duplicates before they can create issues downstream.

4. Enhance ETL Logic:

Update your ETL processes to handle edge cases where orders might be duplicated. For example, consider merging records with the most recent timestamps or highest version.

5. Logging and Alerts:

Implement logging and alerting mechanisms for when the data quality tests fail, so the data engineering team can quickly respond to any anomalies.

By taking these immediate and long-term steps, you can ensure that data integrity is maintained and similar failures are avoided in the future.

Engineering counterpart

11:11 PM - Responded in 0s

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

Agent response

11:11 PM - Responded in 5s

Sure, let's walk through the reasoning step by step and identify the key assumptions made in our approach to resolving and preventing the dbt test failure.

Root Cause Analysis

1. Identifying Test Failure Source:

- Log Analysis:

The log indicates a failure from a uniqueness test (`unique\_fct\_orders\_order\_id`).

```
```text
```

Failure in test unique\_fct\_orders\_order\_id (models/marts/schema.yml)

Got 847 results, expected 0.

```
```
```

The SQL query used in the test is:

```
```sql
```

select order\_id, count() as n

from analytics.fct\_orders

group by 1

having count() > 1

```
```
```

This indicates that there are 847 `order\_id` values that appear more than once, causing the test to fail.

2. Assumption:

- The immediate interpretation is that the `analytics.fct\_orders` table has records that violate the unique constraint on `order\_id`.

Immediate Fix Approach

1. Identify and Flag Duplicates:

- Sample Output:

The log also provides some sample duplicate values, such as `ORD-90102`. This aids in identifying specific cases.

```
```text
```

Sample duplicate order\_id values: ORD-88421, ORD-90102, ORD-90102

```
```
```

- Assumption:

Duplicated entries are present and need to be identified based on some criteria like timestamps, additional context within the data, or a composite key approach.

2. Data Clean-Up:

- Cleaning Table with SQL:

Example SQL to create a cleaned table ensuring each `order\_id` is unique:

```
```sql
```

CREATE TABLE analytics.fct\_orders\_clean AS

SELECT

order\_id,

MAX(order\_timestamp) AS order\_timestamp,

... -- other required columns

FROM analytics.fct\_orders

GROUP BY order\_id

HAVING COUNT() = 1 OR MAX(order\_timestamp) = MIN(order\_timestamp);

```
```
```

- Assumption:

The most recent or latest entry will be used if there are multiple records for the same order. This may not always be the right approach, hence additional context or rules must be applied to decide which record to keep.

Preventative Measures

1. Adding Data Quality Checks:

- ETL Process Check:

Implement checks during the ETL process to identify duplicates before insertion:

```
```sql
SELECT
 order_id,
 COUNT()
FROM analytics.fct_orders
GROUP BY order_id
HAVING COUNT() > 1;
```
```

- Assumption:

These checks will fail the ETL process, preventing duplicate entries from being loaded.

2. Enforcing Constraints:

- DDL Constraints:

Adding a unique constraint to prevent new duplicates from being inserted:

```
```sql
ALTER TABLE analytics.fct_orders ADD CONSTRAINT unique_order_id UNIQUE (order_id);
```
```

- Assumption:

This constraint enforcement will prevent any new duplicate `order\_id` entries from being inserted.

3. Regular Audits:

- Scheduled Audits:

Setting up regular audits to identify and fix duplicates before they cause downstream failures.

- Assumption:

Automated regular audits will catch duplicates early enough to prevent data integrity issues.

4. Enhanced ETL Logic:

- ETL

Engineering counterpart

11:11 PM - Responded in 0s

What would you verify before shipping this to production?

Agent response

11:11 PM - Responded in 4s

Before shipping the fix and preventive measures to production, thorough verification steps are crucial to ensure correctness, data integrity, and avoid any unintended disruptions. Here are the steps and considerations:

Pre-Production Verification Steps

1. Local Environment Testing:

- Test Data Replication:

Set up a local replica of the production environment with a subset of production data to test the logic and identify any issues without impacting the live environment.

- Run Tests:

Implement the cleaning logic and run the same dbt tests in this environment to confirm that the duplicates are removed without losing any legitimate records.

2. Unit Tests for Clean-Up Logic:

- SQL Unit Tests:

Write and run unit tests to validate that the SQL used for cleaning up the data correctly identifies and handles duplicates as intended.

- Assumption Validation:

Ensure that the logic correctly identifies what constitutes a duplicate based on the necessary assumptions (e.g., timestamp criteria).

3. Data Integrity Checks:

- Row Count Checks:

Compare the total number of rows in `analytics.fct\_orders` before and after cleaning to ensure no legitimate data is lost.

- Order Consistency:

Double-check that important attributes related to each `order\_id` are preserved and correctly migrated.

4. Backup Production Data:

- Full Backup Before Change:

Ensure a full backup of the production data is taken before applying any changes. This allows for recovery if something goes wrong.

5. Impact Analysis:

- Dependent Systems Check:

Review any downstream systems that depend on the `analytics.fct\_orders` table to ensure they won't be adversely affected by the changes.

- Data Dependencies:

Confirm that data consumers like BI tools or reports that rely on this table will function correctly post-cleanup.

6. Performance Testing:

- Performance Impact:

Evaluate the performance of the cleaned table to ensure it doesn't adversely impact query performance. This step is particularly important if you are handling very large datasets.

7. Automated Checks:

- Pre-deployment Checks:

Integrate automated checks and tests into the deployment pipeline to ensure all criteria are met before the changes are applied to production.

Production Cutover:

1. Schedule Downtime Wisely:

- Maintenance Window:

Schedule the deployment during a maintenance window to minimize impact on users.

2. Deploy in Stages:

- Phased Rollout:

Consider a phased rollout, perhaps deploying first to a smaller subset or using feature toggles to control the exposure of the change.

3. Monitoring and Rollback Plan:

- Continuous Monitoring:

Set up alerting and monitoring systems to watch for anomalies post-deployment.

- Rollback Plan:

Have a clear and executable rollback plan in case something goes wrong.

By following these steps, you can ensure that both the immediate issue of removing duplicates and the preventative measures are thoroughly tested and validated, thereby reducing the risk of further issues when deploying to production.

Engineering counterpart

11:11 PM - Responded in 0s

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

Agent response

11:11 PM - Responded in 5s

When making significant changes to production data and processes, the greatest risks often involve data loss, system downtime, query performance degradation, and unforeseen dependencies being broken. Here's a closer look at the top risks and a detailed rollback plan:

Top Risks

1. Data Loss or Integrity Issues:

- Risk: Accidental removal of legitimate records during clean-up can result in significant data loss.
- Mitigation: Comprehensive testing, including diffing data before and after the clean-up in a controlled environment.

2. System Downtime or Unavailability:

- Risk: The clean-up process might take longer than expected, causing the system to be unavailable.
- Mitigation: Schedule the clean-up during maintenance windows. Communicate potential disruptions preemptively.

3. Performance Degradation:

- Risk: Post-clean-up, if the cleaned table has fewer rows but larger average row sizes, query performance could degrade.
- Mitigation: Performance testing before and after the change, including realistic load tests.

4. Breakage of Dependent Systems:

- Risk: Changes might break dependent systems or processes that rely on the specifics of the original data.
- Mitigation: Rigorous testing in staging environments that mirror production dependencies.

5. Unidentified Duplicate Criteria:

- Risk: The criteria used for identifying duplicates may not accurately reflect business logic, leading to wrong entries being removed or incorrect duplicates being kept.
- Mitigation: Validate the criteria against business requirements and apply them rigorously in testing.

Rollback Plan

Immediate Pre-Deployment Checks:

1. Verify Backups:

- Ensure that a complete, recent backup of the production environment has been taken.

2. Test Thoroughly in Staging:

- Carry out the full process (including clean-up and potential new constraints) in a staging environment that mirrors production data and schema.

During Deployment:

3. Utilize a Shadow Mode:

- Consider deploying the cleaned table in a shadow mode where it is temporarily used in parallel with the original table. This allows switching back quickly if issues arise.

4. Feature Toggle:

- Implement a toggle that controls whether queries interact with the cleaned table or the original one. This provides a safety switch to revert easily.

Post-Deployment Rollback:

5. Monitor and Revert:

- Once in production, monitor for issues closely and have automated alerts in place. If issues are identified, immediately

engage the rollback plan.

6. Data Restoration:

- Quickly restore from the most recent backup if data integrity issues arise.

7. Switch Back to Original Table:

- If issues occur, use the feature toggle to switch back to the original table until the issue is resolved. The shadow table can also be brought up to date with the backups if necessary.

8. Remediate Performance Issues:

- If performance degrades, analyze the query plans and consider optimizing queries or adjusting database configurations as needed before reintroducing the cleaned data.

Post-Rollback Analysis:

9. Gather Feedback:

- Collect feedback from system users and data consumers to understand the impact.

10. Review and Adjust:

- Based on the feedback and observed issues, review and adjust either the clean-up logic or the rollback approach.

By implementing robust testing, thorough monitoring during and after deployment, and having a clear, executable rollback plan, you mitigate the risks significantly when deploying changes to a production environment.

Engineering counterpart

11:11 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.