



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-4o-mini

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

Duration: 32s

Actual run cost: \$0.001740

Projected execution cost: \$1.53/mo (40/biz day × 22 days)

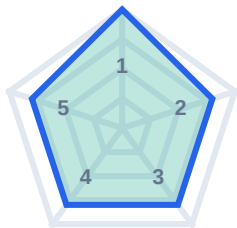
Report date: July 13, 2026 - 10:30 PM EST

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: 13667 chars - Latency: 32s - 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: 32s

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

2:30 AM - Responded in 8s

Root Cause of Duplicates

The uniqueness test, `unique\_fct\_orders\_order\_id`, failed because there are 847 instances of duplicate `order\_id` values in the `analytics.fct\_orders` table. The SQL query in the test highlights that multiple records have the same `order\_id`, which violates the expected uniqueness constraint for this column. The sample duplicates shown in the output, such as `ORD-90102`, confirm that the same `order\_id` appears more than once in the dataset.

Immediate Fix Approach

To remediate the immediate issue of duplicates, follow these steps:

1. Identify Duplicates: Run the provided SQL query to understand the scope of the duplicates. This query groups by `order\_id` and counts the occurrences:

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

2. Analyze Causes: Investigate the source of the data populating `fct\_orders` to determine why duplicates are being inserted. This could be due to issues in upstream ETL processes, such as:

- Missing deduplication logic.
- Issues with data integration from different sources (e.g., multiple systems inserting the same order).

3. Immediate Data Clean-up: As a temporary fix, you can remove duplicates from the `fct\_orders` table. Depending on the business rules, this could involve:

- Retaining the latest entry based on a timestamp or some other logic.
- Aggregating records if they have the same `order\_id` but differ in other attributes.

An example SQL command to remove duplicates while retaining one could look like this (this may need to be adapted based on your specific schema):

```
```sql
DELETE FROM analytics.fct_orders
WHERE ctid NOT IN (
 SELECT MIN(ctid)
 FROM analytics.fct_orders
 GROUP BY order_id
);
```
```

4. Run the Test Again: After cleanup, rerun the dbt test to confirm the uniqueness constraint is satisfied.

Prevent Recurrence

To prevent similar issues in the future, implement the following:

1. Change Data Capture (CDC): If you are working with a source that regularly updates or inserts records, implement CDC to ensure that only new records are added without duplications.

2. Add Deduplication Logic: In your ETL pipelines, ensure there is logic to either check for existing records or merge records based on your business logic before inserting into `fct\_orders`.

3. Implement Constraints: If your database supports it, consider adding unique constraints directly to the `fct\_orders` table

on the `order\_id` column to prevent future insertions of duplicate values.

4. Regular Data Quality Checks: Schedule regular audits and dbt tests to catch these issues before they reach production. This could include running the uniqueness test on a daily or weekly basis.

5. Logging and Monitoring: Enhance your logging to capture instances of data insertion that might lead to duplicates, allowing for quicker resolution when issues arise.

By addressing both the immediate fixing of data and the underlying processes, you can ensure a more robust data pipeline and minimize the risk of duplicates in the future.

Engineering counterpart
2:30 AM - Responded in 0s

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

Agent response

2:30 AM - Responded in 8s

Certainly! Here's a step-by-step breakdown of my reasoning, along with the assumptions I made at each step:

Step-by-Step Reasoning

1. Understanding the Test Failures:

- The output indicates a dbt test failure for the uniqueness of `order\_id` in the `fct\_orders` table, which is critical for maintaining data integrity.
- The test expected zero duplicates but found 847, which highlights a significant issue that needs addressing.

Assumption: The failure is solely due to data integrity issues within the `analytics.fct\_orders` table.

2. Analyzing the SQL Query Provided:

- The SQL query in the error log confirms that the test is counting occurrences of each `order\_id` in the `fct\_orders` table.
- The presence of duplicates is confirmed with the sample values provided in the log (e.g., `ORD-90102`).

Assumption: Duplicates are present in the data and have not been correctly managed in the ETL process that populates this table.

3. Identifying Immediate Fix Steps:

- I proposed running the SQL query provided to further understand the degree of duplication. This is a common troubleshooting step to gather data for analysis.
- A clean-up strategy was necessary to handle the immediate issue by deduplicating the `fct\_orders` table.

Assumption: Duplicates can be safely removed or consolidated without significant business impact. This assumes there's clarity on which records to retain based on existing business logic.

4. Investigating Underlying Causes:

- I suggested analyzing the upstream ETL processes for potential issues leading to duplicate entries.
- This step is crucial because it recognizes that fixing the data is only part of the solution; the processes that created the duplicates need to be reviewed.

Assumption: There is a systematic issue in the data ingestion process that led to duplicates, and that the team has access to these upstream processes for analysis.

5. Proposing Long-term Preventative Measures:

- I emphasized implementing deduplication logic in ETL processes and possibly adding unique constraints in the database.
- These measures aim to establish a robust framework that prohibits duplicates from being created in the future.

Assumption: The data engineering team has the ability and authority to modify ETL processes and database schemas to enforce data integrity.

6. Consideration of Regular Monitoring:

- I recommended regular data quality checks and monitoring to catch future duplication issues proactively.
- This is crucial for maintaining data integrity over time.

Assumption: There are resources available (time, personnel) to implement monitoring practices and that stakeholders recognize the importance of ongoing data quality.

Summary of Assumptions

- The duplicates stem from issues in the data ingestion or ETL processes.
- Clean-up of duplicates can be performed without significant harm to business operations.
- The data team has the capability to modify ETL processes and database constraints.

- Regular monitoring practices can be established to ensure ongoing data quality.

By following this structured approach and making these assumptions, I aimed to provide a thorough analysis and actionable remediation steps while ensuring that the core issue (data integrity) is addressed comprehensively.

Engineering counterpart
2:30 AM - Responded in 0s

What would you verify before shipping this to production?

Agent response

2:30 AM - Responded in 7s

Before shipping the immediate fix to production, several verification steps should be undertaken to ensure that the changes are safe, effective, and do not introduce new issues. Here's a list of what to verify:

1. Review Data Cleanup Logic

- Verify Clean-Up Query: Ensure that the SQL query designed to remove duplicates has been thoroughly reviewed and tested in a development or staging environment. This includes:
 - Confirming that the logic retains the correct records based on business rules (e.g., keeping the most recent entry).
 - Ensuring it doesn't inadvertently remove necessary data.
- Test with Sample Data: Use sample data that mirrors the production dataset to validate the effectiveness of the deduplication logic without affecting real data.

2. Check Upstream ETL Processes

- Review Source Data: Validate the upstream processes that populate the `fct\_orders` table to ensure they are robust. This may involve:
 - Confirming that deduplication logic is added to the ETL processes.
 - Checking for any changes in the source systems that might lead to duplicates.
- Simulate Data Ingestion: Run a test ingestion with the expected data flow to see if duplicates are generated again.

3. Rerun dbt Tests

- Validate Tests Locally: Before deploying to production, run all relevant dbt tests in a development environment. This includes:
 - The uniqueness test for `order\_id`.
 - Any other related tests that might be affected by the changes to ensure no new issues arise.
- Check for Other Errors: Ensure that there are no other known issues in the data that could impact production.

4. Assess Impact on Other Systems

- Determine Dependencies: Check if other systems rely on the `fct\_orders` table and whether the changes could affect them. This may involve:
 - Communicating with stakeholders from other teams who may rely on this data.
 - Reviewing any reporting or downstream processes that utilize this table.

5. Establish Monitoring and Rollback Plans

- Set Up Monitoring: Implement monitoring for the `fct\_orders` table post-deployment to quickly identify any new duplicates or anomalies.
- Prepare a Rollback Plan: Develop a plan to revert changes if unexpected issues arise after deployment. This could involve restoring from a backup or rolling back the deduplication changes.

6. Coordinate with Stakeholders

- Communicate Fixes: Inform relevant stakeholders across teams about the planned changes and how they may affect their processes.
- Schedule Deployment: Coordinate the deployment at a time that minimizes disruption, ensuring that relevant teams are aware and available to address any immediate concerns.

7. Documentation

- Update Documentation: Ensure that any changes to the ETL processes, data models, or business logic are documented for future reference and can help with onboarding or maintenance.

By going through these verification steps, you can minimize the risk of introducing new issues into production and ensure that the immediate fix for duplicates is both effective and safe.

Engineering counterpart
2:30 AM - Responded in 0s

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

Agent response

2:30 AM - Responded in 9s

When addressing the issue of duplicates in the `fct\_orders` table and deploying the necessary changes to production, several risks need to be considered. Below are the top risks associated with the changes, along with a detailed rollback plan should the deployment not work as intended.

Top Risks

1. Data Loss:

- Risk: The deduplication process may inadvertently remove valid records, leading to loss of critical data.
- Mitigation: Thoroughly test the deduplication logic in a non-production environment before applying it to production.

2. New Duplicates:

- Risk: Changes made to the ETL processes to prevent duplicates may not be effective, leading to new duplicates being introduced.
- Mitigation: Implement logging and monitoring to track duplicate entries and validate the changes post-deployment.

3. Performance Issues:

- Risk: The cleanup operation might be resource-intensive, impacting database performance or causing downtime during execution.
- Mitigation: Schedule the cleanup during off-peak hours and test the performance impact in a staging environment.

4. Upstream System Interference:

- Risk: Changes in the deduplication logic or ETL processes can disrupt the data ingestion from upstream systems.
- Mitigation: Ensure rigorous testing of the entire data pipeline and communicate with teams managing upstream systems to understand their workflows.

5. Compliance and Reporting Issues:

- Risk: If reporting or compliance relies on the completeness of the `fct\_orders` data, removing records may lead to regulatory concerns or inaccurate reporting.
- Mitigation: Ensure that all stakeholders are informed and adjustments are made in reporting systems as needed.

Rollback Plan

If the changes do not work as intended, the following rollback plan can be enacted to minimize disruption and restore the system to its previous state:

1. Backup Data:

- Before Deployment: Ensure a full backup of the `fct\_orders` table (and potentially related tables) is created prior to making any changes. This allows for a restore point to revert to if necessary.

2. Revert Changes:

- If Data Loss Occurs: If the deduplication process results in data loss, restore the `fct\_orders` table from the backup using the following SQL command:

```
```sql
DELETE FROM analytics.fct_orders;
INSERT INTO analytics.fct_orders (SELECT FROM backup.fct_orders);
```
```

3. Monitoring Logs:

- Check Logs: Review logs for any errors during the deduplication and ETL processes. If errors are detected indicating that new duplicates were created, capture relevant information for troubleshooting.

4. Notify Stakeholders:

- Immediate Communication: Inform stakeholders across affected teams that the deployment did not yield the desired results and that a rollback is being initiated. Provide details on the impact and expected timeline for resolution.

5. Disable New ETL Logic:

- Temporarily Disable: If the new ETL processes introduced changes to prevent duplicates, revert those changes in the ETL logic to ensure that normal operations can resume without the risk of new duplicates being introduced.

6. Post-Rollback Review:

- Conduct a Post-Mortem: After the rollback, conduct a review of what went wrong, focusing on identifying weaknesses in the testing or deployment process to prevent recurrence of similar issues in the future.

7. Reassess Approach:

- Iterate on Solution: Based on findings from the rollback and review, iterate on the deduplication logic, ETL process changes, and testing strategies before attempting a new deployment.

By having these risks and a rollback plan in place, you can be better prepared to handle potential failures and ensure that your data integrity processes remain

Engineering counterpart

2:30 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.