



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: ibm-granite/granite-4.0-h-micro

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

Duration: 1m 41s

Actual run cost: \$0.000259

Projected execution cost: \$0.2277/mo (40/biz day × 22 days)

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

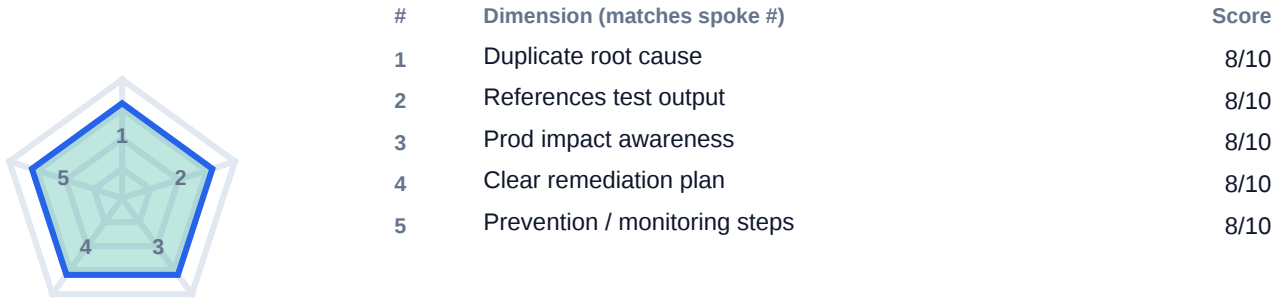
Overall

8.0/10

PASS - trust=high

Pass - score 8.0/10 clears 7.0 with acceptable trust/closure.

CAPABILITY RUBRIC



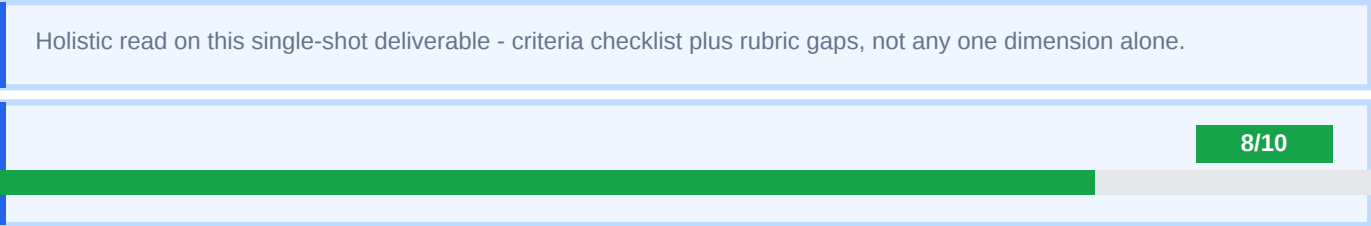
Spoke numbers run clockwise from the top; dimension names are listed in full in the table.

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



Telemetry

Run metrics

Mode: multi-turn - Agent msgs: 4 - Counterpart msgs: 5 - Output: 15012 chars - Latency: 101s - 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 41s

Engineering counterpart

2:27 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

2:27 AM - Responded in 25s

Root Cause Analysis

The uniqueness test `unique\_fct\_orders\_order\_id` failed because the query executed against the `analytics.fct\_orders` table returned 847 duplicate `order\_id` values, whereas the test expected zero duplicates. This indicates that the `order\_id` column in the `fct\_orders` fact table is not unique, which is a critical data quality issue.

Key Observations from the Log:

- Query Executed:

```
```sql
select order_id, count() as n
from analytics.fct_orders
group by 1
having count() > 1
```
```

This query correctly identifies rows where `order\_id` appears more than once.

- Duplicate Values Found: The log explicitly states that the duplicate `order\_id` values include `ORD-88421`, `ORD-90102`, and `ORD-90102` (showing that `ORD-90102` is duplicated).

- Test Expectation: The test was expecting 0 duplicates, which is a standard requirement for a unique constraint on `order\_id`.

Immediate Fix Approach

1. Identify and Address the Data Issue:

- Data Source Inspection: Investigate the source of the `fct\_orders` table. This could be from an ETL process, a data feed, or a direct database load. The source might be inserting duplicate `order\_id` values.

- ETL Process Review: Check the ETL or data pipeline that loads `fct\_orders`. Ensure that the process is designed to handle and prevent duplicate `order\_id` entries. This might involve:

- Adding a unique constraint on `order\_id` in the source system.

- Implementing a deduplication step in the ETL process to ensure that only unique `order\_id`s are loaded into the data warehouse.

- Data Validation: Implement or enhance data validation checks in the ETL process to catch duplicate `order\_id`s before they reach the data warehouse.

2. Immediate Remediation:

- Drop and Re-create the Table: As a temporary measure, you could drop the `fct\_orders` table and re-load it with a deduplication step. This is a drastic measure and should be used only if the issue cannot be resolved quickly.

- Update the Test: Once the data issue is resolved, update the uniqueness test to reflect the new expected outcome (i.e., 0 duplicates). This will ensure future runs pass once the data quality issue is fixed.

3. Rollback or Workaround:

- If the data issue cannot be resolved immediately, consider temporarily disabling the uniqueness test until the issue is fixed. This should be a last resort and clearly documented to avoid confusion.

Preventing Recurrence

To prevent this issue from recurring, implement the following preventive measures:

1. Data Quality Checks in ETL:

- Unique Constraint Enforcement: Ensure that the `order\_id` column in the source system is defined with a unique constraint. This prevents duplicate entries from being inserted in the first place.

- Duplicate Detection: Implement a deduplication step in the ETL process. This could involve:

- Running a deduplication script that checks for duplicate `order\_id`s before loading data into the warehouse.

- Using dbt's `dedupe` or similar functions to identify and handle duplicates during the transformation stage.

2. Automated Data Validation:

- Schema Tests: Continue using dbt tests to validate data quality. In addition to the uniqueness test, add other relevant tests such as:

- Presence of Unique Constraint: Use dbt to check if the `order\_id` column has a unique constraint defined in the schema.

- Data Integrity Checks: Implement tests to ensure that `order\_id` values are not null or contain unexpected characters.

3.

Engineering counterpart
2:27 AM - Responded in 0s

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

Agent response

2:28 AM - Responded in 25s

Certainly! Let's break down the reasoning process step by step, including the assumptions made during the analysis and remediation of the dbt test failure related to duplicate `order\_id` values in the `fct\_orders` table.

Step-by-Step Reasoning

1. Understanding the Test Failure

- Observation: The uniqueness test `unique\_fct\_orders\_order\_id` failed with the message "Got 847 results, expected 0."
- Implication: This indicates that the `order\_id` column in the `fct\_orders` table contains duplicate values, as the query executed by the test (`select order\_id, count() as n from analytics.fct\_orders group by 1 having count() > 1`) returned 847 rows where `order\_id` appeared more than once.

2. Identifying the Root Cause

- Assumption: The `order\_id` column is intended to be unique in the business context (e.g., each order should have a unique identifier).
- Root Cause Analysis:
 - Data Source Issue: The most likely cause is that the source system (where `fct\_orders` is populated) is not enforcing the uniqueness constraint on `order\_id`. This could be due to:
 - ETL Process: The ETL pipeline loading data into `fct\_orders` might not be checking for duplicates before inserting them into the data warehouse.
 - Source System: The source system itself might be inserting duplicate `order\_id`s due to a bug or lack of unique constraint enforcement.
 - Data Validation: There might be a lack of data validation steps in the ETL process to catch and handle duplicates before they reach the data warehouse.

3. Immediate Fix Approach

- Assumption: The immediate fix should address the data quality issue without disrupting ongoing operations.
- Immediate Remediation Steps:
 - Data Source Inspection:
 - Action: Investigate the source system (e.g., database, application, or external feed) that populates `fct\_orders`. Determine how `order\_id`s are generated and inserted.
 - Action: Ensure that the source system enforces a unique constraint on `order\_id`. This might involve:
 - Adding a unique constraint in the source database schema.
 - Implementing a deduplication step in the ETL process to filter out duplicate `order\_id`s before loading them into the warehouse.
 - ETL Process Review:
 - Action: Review the ETL pipeline responsible for loading `fct\_orders`. Ensure that it includes a step to detect and handle duplicates.
 - Action: Implement a deduplication mechanism within the ETL process. This could involve:
 - Running a deduplication script that checks for duplicate `order\_id`s and either removes duplicates or flags them for review.
 - Using dbt's `dedupe` function or similar to identify and handle duplicates during the transformation stage.
 - Immediate Fix:
 - Action: As a temporary measure, you could drop the `fct\_orders` table and re-load it with the deduplication step in place. This is a drastic measure and should be used only if the issue cannot be resolved quickly.
 - Action: Update the uniqueness test to reflect the new expected outcome (i.e., 0 duplicates). This ensures that future runs of the test will pass once the data quality issue is fixed.

4. Preventing Recurrence

- Assumption: To prevent similar issues in the future, it's crucial to embed data quality checks into the ETL process and schema design.
- Preventive Measures:
 - Data Quality Checks in ETL:
 - Unique Constraint Enforcement: Ensure that the `order\_id` column in the source system is defined with

Engineering counterpart
2:28 AM - Responded in 0s

What would you verify before shipping this to production?

Agent response

2:28 AM - Responded in 26s

Before shipping the fix to production, it's crucial to verify several aspects to ensure that the issue has been resolved correctly and that no new problems have been introduced. Here's a comprehensive checklist of what to verify:

1. Data Quality Verification

- Uniqueness of `order\_id`:
 - Action: Run the uniqueness test `unique\_fct\_orders\_order\_id` again to confirm that it now passes (i.e., returns 0 duplicates).
 - Verification: Ensure that the `order\_id` column in the `fct\_orders` table is indeed unique in the production database.
- Data Integrity:
 - Action: Verify that the deduplication process (if implemented) has correctly removed or flagged duplicates without losing any critical data.
 - Verification: Check a sample of the data to ensure that no important `order\_id` values were inadvertently removed or altered.

2. ETL Process Verification

- Deduplication Logic:
 - Action: Review the ETL pipeline code to ensure that the deduplication logic is correctly implemented and functioning as expected.
 - Verification: Test the ETL process with a small batch of data to confirm that duplicates are handled correctly.
- Data Validation Steps:
 - Action: Ensure that additional data validation steps (e.g., checks for null values, data type consistency, etc.) are in place and functioning correctly.
 - Verification: Run the ETL process with a variety of test datasets to ensure that all validation checks pass without errors.

3. dbt Tests and Schema Verification

- dbt Test Suite:
 - Action: Run the full dbt test suite to ensure that all other tests pass successfully.
 - Verification: Confirm that the test suite reports no failures or warnings.
- Schema Changes:
 - Action: Verify that the schema changes (e.g., adding a unique constraint on `order\_id`) have been correctly applied to the production database.
 - Verification: Use dbt's `dbt docs` or `dbt docs generate` commands to generate and review the updated schema documentation.
- Schema Validation:
 - Action: Ensure that the unique constraint on `order\_id` is enforced in the production database.
 - Verification: Run a query to check the constraints on the `fct\_orders` table to confirm the unique constraint is present.

4. Performance and Load Testing

- Action: Conduct performance testing to ensure that the ETL process and dbt tests run efficiently in production.
- Verification: Monitor the execution time and resource usage of the ETL process and dbt tests. Ensure that they meet the performance benchmarks set for production.
- Load Testing: If applicable, perform load testing to simulate high-volume data loads and ensure that the system handles the expected data volume without performance degradation.

5. Rollback and Recovery Plan

- Action: Prepare a rollback plan in case the fix introduces unforeseen issues.
- Verification: Ensure that the rollback plan includes steps to revert the schema changes and re-enable any previously disabled tests or processes.
- Verification: Test the rollback plan in a staging environment to confirm that it works as expected.

6. Stakeholder Review and Approval

- Action: Conduct a review meeting with relevant stakeholders (e.g., data engineers, business analysts, and data consumers) to ensure that everyone is aware of the changes and approves them.

- Verification: Document the changes, the verification steps taken, and obtain sign-off from stakeholders before proceeding to production.

7. Monitoring and Alerting

- Action: Implement monitoring and alerting for the ETL process and dbt tests to detect any issues post-deployment.

Engineering counterpart
2:28 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 25s

Top Risks

1. Data Loss or Corruption

- Risk: The deduplication process might inadvertently remove or alter critical data, leading to data loss or corruption.
- Impact: This could affect downstream analytics, reporting, and business decisions that rely on accurate `order\_id` data.

2. ETL Process Failure

- Risk: The ETL pipeline might fail to load the corrected data, causing a disruption in data availability.
- Impact: Business operations that depend on timely data loads could be impacted, leading to delays in reporting and decision-making.

3. Schema Changes Not Applied Correctly

- Risk: The unique constraint on `order\_id` might not be correctly applied to the production database, leaving the data vulnerable to duplicates.
- Impact: This could lead to future test failures and data quality issues, undermining the integrity of the data warehouse.

4. Performance Degradation

- Risk: The deduplication process or additional data validation steps might introduce performance bottlenecks, slowing down the ETL process or dbt tests.
- Impact: This could lead to longer runtimes, increased resource consumption, and potential timeouts or failures during data processing.

5. Rollback Complexity

- Risk: If the rollback plan is not well-tested or executed correctly, it could lead to further complications, such as data inconsistencies or incomplete rollbacks.
- Impact: This could result in a prolonged period of instability or require additional troubleshooting efforts.

Rollback Plan

In case the fix does not work as expected, having a well-defined rollback plan is crucial to quickly revert to the previous state and minimize the impact on production operations. Here's a step-by-step rollback plan:

1. Immediate Actions

- Stop the ETL Process: Immediately halt the ETL pipeline to prevent further data loading or processing.
- Pause dbt Tests: Pause or disable the dbt tests that are currently running to prevent them from executing against the potentially problematic data.

2. Data Recovery

- Restore from Backup: If a backup of the `fct\_orders` table exists, restore it to the production database. This step should be performed as quickly as possible to minimize data loss.
- Action: Use your database backup and restore tools (e.g., `pg\_restore` for PostgreSQL, `mysqldump` for MySQL) to restore the backup.
- Verification: Verify that the restored data matches the pre-fix state, ensuring that the `order\_id` column is unique and free of duplicates.
- Revert Schema Changes: If schema changes were made (e.g., adding a unique constraint), revert these changes to the previous state.
- Action: Use SQL commands to drop the unique constraint on `order\_id` if it was added as part of the fix.

```
```sql
ALTER TABLE analytics.fct_orders DROP CONSTRAINT IF EXISTS unique_order_id;
```
```

3. ETL Process Recovery

- Resume ETL Process: Once the data has been restored and schema changes reverted, resume the ETL process to

load the latest data into the data warehouse.

- Verification: Monitor the ETL logs to ensure that the process completes successfully without errors.

4. Testing and Validation

- Run dbt Tests: Re-run the dbt test suite to verify that all tests pass successfully, confirming that the data quality issues have been resolved.

- Action: Execute the dbt tests using the command:

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

- Verification: Ensure that all tests pass without any failures or warnings.
- Data Validation Checks: Perform additional data validation checks to ensure that the data integrity is maintained.
- Action: Run data validation scripts or queries to confirm

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.