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: qwen/qwen3-32b
Roles: Data Engineer (AI agent) <> Engineering counterpart (AI agent)
Duration: 1m 41s
Actual run cost: $0.000872
Projected execution cost: $0.7674/mo (40/biz day × 22 days)
Report date: July 14, 2026 - 2:53 AM EST

Session: sample-lb-rankings-seed-c0cf7c20-715c-4f8f-a44a-f9bd1af0fc65

de_dbt_test_failure_v1 - agent - openrouter - qwen/qwen3-32b - ended

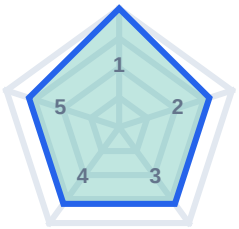
Overall

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: 7900 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

Transcript: blue = protagonist (you) - amber = counterpart - gray = coach/system. Each turn shows role/name, clock time, and response delay when available.

Engineering counterpart
6:51 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
6:51 AM - Responded in 14s

Root Cause of Duplicates

Engineering counterpart
6:51 AM - Responded in 0s

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

Agent response
6:52 AM - Responded in 59s

Step-by-Step Reasoning & Root Cause Analysis

1. Root Cause of Duplicates
The uniqueness test `unique_fct_orders_order_id` failed because duplicate `order_id` values exist in the `fct_orders` table. The test SQL:
```sql
SELECT order_id, COUNT() AS n
FROM analytics.fct_orders
GROUP BY 1
HAVING COUNT() > 1
```
indicates that the same `order_id` (e.g., `ORD-90102`) appears multiple times in the table. This violates the assumption that `order_id` should be unique, likely because:

- The DAG is re-ingesting historical data without filtering (e.g., pulling all orders from the source instead of just new ones). For example, a full-refresh DAG running daily would overwrite data and potentially reintroduce duplicates if no date range is applied.
- The transformation logic failed to deduplicate upstream data. If `fct_orders` aggregates from a source table (e.g., `stg_orders`) that contains multiple rows per order (e.g., due to joins with other tables like `order_items`), and the model does not `GROUP BY` or `DISTINCT`, duplicates could propagate.
- The model is materialized as a table without enforcing uniqueness at the database level (e.g., no primary key constraint). This allows accidental insertion of duplicate rows during reprocessing.

Assumptions made:
- `order_id` is a business key that should uniquely identify a transaction in the source system (e.g., an order ID should never be reused or duplicated).
- The issue is recent (triggered by a DAG run or model rebuild) rather than historical corruption, given the production context.
- No upstream table explicitly allows multiple `order_id` entries per order (e.g., `fct_orders` is grain: 1 row per order).

---

2. Immediate Fix Approach
Objective: Remove existing duplicates from `fct_orders` and resume normal operations.

Steps:
1. Identify and Prioritize Data:
   - Query the duplicates to determine which rows are the latest/up-to-date entries (e.g., using a timestamp column like `ORDER_CREATED_AT` or `EFFECTIVE_DATE`).
  - Example SQL to delete duplicates:
    ```sql
    DELETE FROM analytics.fct_orders
    WHERE (order_id, ORDER_CREATED_AT) IN (
      SELECT order_id, MIN(ORDER_CREATED_AT)
      FROM analytics.fct_orders
      GROUP BY 1
      HAVING COUNT() > 1
    )
    ```
    This keeps the most recent version of each order.

2. Pause or Fix the Ingestion Logic:
  - Temporary workaround: Block the DAG from re-ingesting historical data temporarily to prevent further duplicates.

- Permanent fix: Update the DAG or model to only process new/updated orders (e.g., filter by `created_at >= {{ last_run_time }}`).

3. Resume Operations:
- After deduplication, re-enable the DAG/model. Ensure the fix doesn't disrupt downstream dependencies.

---

3. Prevent Recurrence
Root Cause Mitigation:
- Materialization Fix: Switch `fct_orders` to an incremental model with a primary key constraint on `order_id`. This ensures:
  ```yml
  # models/fct_orders.sql
  {{ config(
    materialized='incremental',
    unique_key='order_id',
    tags=['orders']
  ) }}
  ```
  - `unique_key='order_id'` tells dbt to only insert new rows (not overwrite).
  - Add a database-level primary key (if possible) to enforce uniqueness at the schema level:
    ```sql
    ALTER
    ```

Engineering counterpart
6:52 AM - Responded in 0s
What would you verify before shipping this to production?

Agent response
6:52 AM - Responded in 12s
Verification Steps Before Shipping to Production

Before deploying the fix to production, I'd validate the following to ensure the solution is robust, backward-compatible, and won't cause new issues:

---

1. Verify Deduplication Logic
- Check data integrity: Query `fct_orders` post-fix to confirm:
  - Duplicate `order_id`s are removed (e.g., `SELECT order_id, COUNT() FROM fct_orders GROUP BY 1 HAVING COUNT() > 1` should return 0 rows).
  - Critical fields (e.g., `total_amount`, `order_date`) remain unaffected for retained rows (e.g., use `LIMIT 100` to sample and compare pre- and post-fix data).
- Audit deletion strategy: Confirm the deduplication script (e.g., `DELETE` with `MIN(ORDER_CREATED_AT)`) prioritizes the correct version of the data (assuming `ORDER_CREATED_AT` is the source of truth).

---

2. Confirm Ingestion Logic Changes
- Validate upstream filtering: For the DAG/model that populates `fct_orders`, ensure:
  - The ingestion query now filters for new/updated orders only (e.g., `created_at > {{ last_run_time }}`

Engineering counterpart
6:52 AM - Responded in 0s
What are the top risks, and what's the rollback plan if this doesn't work?

Agent response
6:53 AM - Responded in 16s

Top Risks of the Proposed Fix

1. Data Loss from Deduplication
   - Risk: Mistakenly deleting the "correct" version of an order (e.g., assuming `MIN(created_at)` identifies the latest data when it actually represents the earliest).
   - Mitigation: Ensure the deduplication logic uses the right timestamp/basis for determining valid rows (e.g., `MAX(updated_at)` if updates exist). Validate with a sample of data where you know the correct outcome.

2. Ingestion Logic Breaking New Data
   - Risk: Filtering by `created_at >= {{ last_run_time }}` might exclude valid new orders if the source data uses a different time zone or if `last_run_time` is incorrectly calculated.
   - Mitigation:
   - Test the DAG/model in a staging environment with the same logic.
   - Ensure `last_run_time` is derived from a reliable source (e.g., `last_modified` in the source system or a dbt metadata table like `dbt_log`).

3. Database Schema Violations
   - Risk: Adding a primary key constraint to `order_id` will fail if duplicates still exist in the table.
   - Mitigation: Confirm the table is fully deduplicated before applying the constraint.

4. Downstream Dependency Failures
   - Risk: Existing models or dashboards might rely on the current duplicated data structure, causing errors after deduplication.
   - Mitigation:
   - Review all downstream models to ensure they don't rely on counts of duplicates (e.g., `JOIN fct_orders ON order_id` for unique counts).
   - Run a full validation suite for downstream consumers of `fct_orders`.

Rollback Plan if Fix Fails

If the fix introduces unexpected issues (e.g., data corruption, downtime), execute the following step-by-step rollback:

1. Revert the Deduplication DELETE
   - Action: Immediately restore the table from a pre-deduplication backup (if one exists).
   - Details:
   - If no backup exists, re-run the original DAG/model (not the deduplication script) to regenerate the table from source data.
   - Use a `INSERT INTO` statement with a restored backup of duplicates to recover data:
     ```sql
     INSERT INTO analytics.fct_orders (order_id, ...)
     SELECT  FROM analytics.fct_orders_backup
     ```

2. Revert Ingestion Logic
   - Action: Roll back the DAG/model code to the previous stable version (before the unique key/created_at filter).
   - Details:
   - Use source control (e.g., Git) to revert changes to the DAG/model code.
   - Ensure the DAG uses the full-refresh mode or a safe incremental strategy with a fallback for duplicate handling (e.g., `GROUP BY` in the model SQL).

3. Revert Model Materialization

- Action: If switching to incremental materialization caused issues (e.g., missing data), revert to the original materialization (likely a table).
- Details:
  - Update the model config to use `materialized='table'` again.
  - Ensure the model includes a `GROUP BY` or `DISTINCT` clause to maintain uniqueness until the root cause is fully resolved.

4. Remove Primary Key Constraint
- Action: If applying the primary key constraint caused schema errors, drop the constraint:
  ```sql
  ALTER TABLE analytics.fct_orders DROP
  ```

Engineering counterpart
6:53 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.