



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/qwen3-235b-a22b-2507

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

Duration: 2m 53s

Actual run cost: \$0.001734

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

Report date: July 14, 2026 - 2:54 AM EST

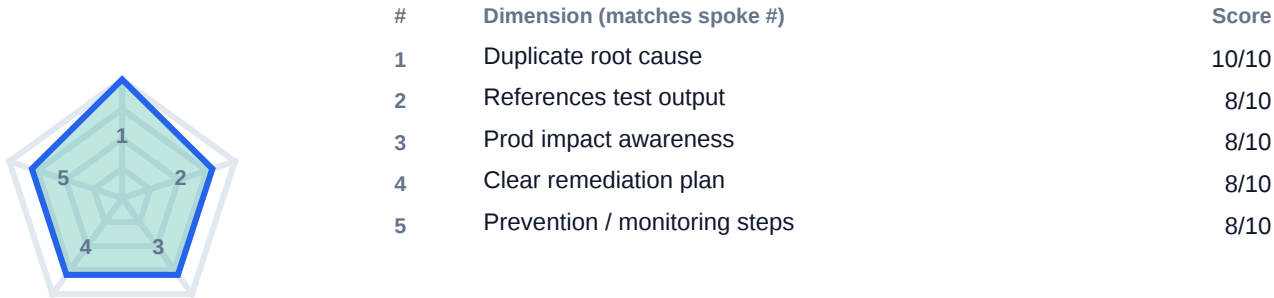
Overall

8.4/10

PASS - trust=high

Pass - score 8.4/10 clears 7.0 with acceptable trust/closure.

CAPABILITY RUBRIC



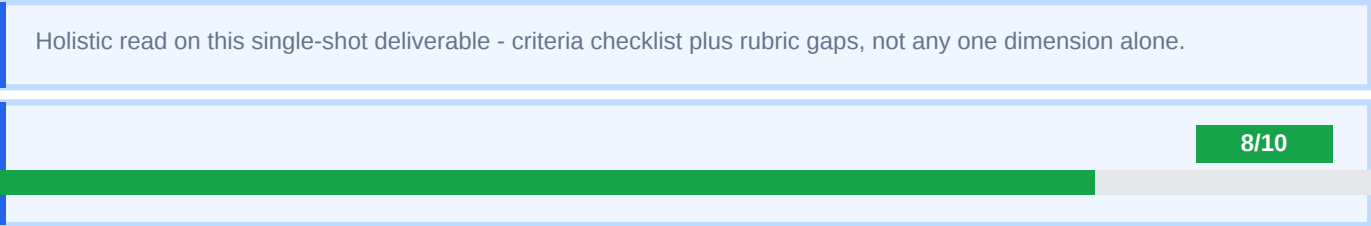
Spoke numbers run clockwise from the top; dimension names are listed in full in the table.

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: 12838 chars - Latency: 173s - 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: 2m 53s

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

6:52 AM - Responded in 56s

Root Cause of Duplicates

The uniqueness test `unique\_fct\_orders\_order\_id` on the `fct\_orders` table failed because 847 `order\_id` values appear more than once, violating the expectation that `order\_id` should be unique.

From the sample provided in the test output (`ORD-88421`, `ORD-90102`, `ORD-90102`), we observe actual duplicate occurrences. The test query:

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

confirmed duplicates in the materialized `fct\_orders` fact table.

Immediate Fix Approach

1. Investigate Source of Duplication

Run diagnostic queries to determine whether:

- Duplicates originated from the source system (e.g., `stg\_orders`).
- Duplicates were introduced during transformation (e.g., incorrect joins, missing deduplication logic).
- The model logic lacks a deduplication step on `order\_id`.

```
```sql
-- Check for duplicates in source/staging
select order_id, count() as n
from analytics.stg_orders
group by order_id
having count() > 1
limit 5;
```
```

If no duplicates exist in staging, the issue likely stems from a join or union in the `fct\_orders` model that inadvertently created duplicate rows.

2. Temporarily Flag and Log Duplicates

Add a temporary logging mechanism or alert for any future duplicates, while preserving data integrity.

3. Rebuild `fct\_orders` with Deduplication Logic

Modify the `fct\_orders` model to deduplicate on `order\_id`. Choose a strategy based on business logic:

- Most recent record (by timestamp):

```
```sql
select
from (
 select ,
 row_number() over (partition by order_id order by updated_at desc) as rn
 from analytics.stg_orders
) ranked
where rn = 1
```
```

- First occurrence (by load timestamp) or based on business key validity.

4. Re-run the Model and Test

After applying deduplication:

```
```bash
dbt run -m fct_orders
dbt test -m fct_orders
```
```

5. Backfill or Replace Table

If using incremental model, consider a full refresh to eliminate historical duplicates:

```
```bash
dbt run -m fct_orders --full-refresh
```
```

How to Prevent Recurrence

1. Enforce Uniqueness in Model Logic

- Ensure all models producing `fct\_orders` deduplicate on `order\_id` if it is intended to be a unique key.
- Implement `row\_number()` or `qualify` logic to retain only one record per `order\_id`.

2. Add Data Quality Monitoring

- Keep the `unique` test in place.
- Add a warning test with threshold if zero tolerance isn't always feasible:

```
```yaml
- test:
 name: unique_fct_orders_order_id
 config:
 error_if: '> 0'
 warn_if: '> 0'
```
```

3. Implement Upstream Controls

- If duplicates come from raw ingestion, apply deduplication at the staging layer (`stg\_orders`) with proper versioning or merging logic.

4. Document Assumptions and Ownership

- Clarify in model documentation whether `order\_id` is expected to be unique and under what conditions (e.g., "unique per source system").
- Tag owners in `schema.yml` for alerting.

5. Add Alerting Pipeline

Integrate

Engineering counterpart
6:52 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 15s

Absolutely - let's walk through the reasoning step by step, focusing strictly on the evidence in the test output, and clearly calling out what was observed, what can be inferred, and what assumptions were made.

? Step 1: Read and Understand the Test Output

We are given this key piece of information:

1 of 1 FAIL 847 unique_fct_orders_order_id [FAIL 847 in 2.11s]

...

Got 847 results, expected 0.

...

compiled Code at .../unique_fct_orders_order_id.sql

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

Sample duplicate order_id values: ORD-88421, ORD-90102, ORD-90102

? Observations:

1. A `unique` test on `fct\_orders.order\_id` failed.
2. The test returned 847 rows, each representing an `order\_id` that appears more than once.
3. The test query groups by `order\_id` and filters for counts > 1.
4. A sample shows two instances of `ORD-90102`, confirming actual duplication (not just cardinality issue).
5. The model is called `fct\_orders`, which is a fact table (standard data modeling convention).

? Step 2: What Does a Uniqueness Test Failure Mean?

In dbt, a `unique` test asserts that every value in the specified column is distinct.

So when we test `unique(fct\_orders.order\_id)`, we are asserting:

> "Each order_id should appear exactly once in the fct_orders table."

? Failure means:

There are duplicate `order\_id` values in the materialized `fct\_orders` table - 847 `order\_id`s are repeated (each may be duplicated 2+ times).

This indicates a data integrity issue, either:

- In the source,
- In the transformation logic,
- Or in how the table was built or updated.

> There is no evidence of a syntax error, connection failure, or schema drift - just a data anomaly.

? Step 3: Identify Where the Problem Lives

We know:

- The test runs against `analytics.fct\_orders`.
- It uses `order\_id` as the tested column.
- The failure occurred in production (`target='prod'`).

We don't see:

- The SQL for the `fct\_orders` model.
- Any logs from upstream models or ingestion steps.
- Details about whether it's incremental or table materialization.

But we can infer:

- The table currently contains duplicates.
- The duplicates were either:
 - Already present in source/staging,
 - Introduced during transformation (e.g., bad join causing duplication),
 - Or reintroduced during a refresh (e.g., not deduplicating on load).

? Root Cause (Based Only on Evidence)

> The `fct\_orders` table contains multiple rows with the same `order\_id`. This violates the uniqueness constraint assumed by business logic and enforced by the test.

While we can't know for sure why the duplicates exist without looking upstream, the most common root causes are:

1. No deduplication during ETL - if source data has multiple versions of the same order, they're all being loaded.
2. Incorrect joins - e.g., joining orders to line items without aggregating, causing one order to fan out into many rows, all with the same `order\_id`.
3. Incremental logic issues - e.g., missing a uniqueness constraint in `unique`

Engineering counterpart
6:52 AM - Responded in 0s

What would you verify before shipping this to production?

Agent response

6:52 AM - Responded in 16s

Great follow-up. Before shipping any fix for the `unique\_fct\_orders\_order\_id` failure to production, we must verify several critical items to avoid unintended data loss, incorrect business logic, or downstream disruptions.

Here's a structured checklist of what to verify, why, and how, grounded in the context of the observed failure:

? 1. Verify the Nature of Duplicates

Why: Not all duplicates are errors. We must determine whether they are:

- True data corruption (e.g., same row ingested twice),
- Multiple valid updates to the same order (e.g., order status changes),
- Or unintentional fan-out from a join (e.g., one-to-many expansion without aggregation).

How to verify:

```
```sql
-- Sample a few duplicate order_id records
SELECT
 order_id,
 COUNT() AS row_count,
 MIN(created_at) AS first_seen,
 MAX(created_at) AS last_seen,
 ARRAY_AGG(updated_at ORDER BY updated_at) AS update_history,
 COUNT(DISTINCT order_status) AS status_changes
FROM analytics.fct_orders
WHERE order_id IN ('ORD-88421', 'ORD-90102')
GROUP BY order_id;
```
```

? If `order\_id` has multiple distinct `updated\_at`, `order\_status`, or `amount` values, this suggests slowly changing dimensions - not duplicates, but versions.

? If all columns are identical, likely true duplicates from ingestion.

? 2. Check the Source: Is `order\_id` Supposed to Be Unique?

Why: Confirm the business contract. Is `order\_id` intended to be a unique identifier across the system?

How to verify:

- Interview stakeholders (e.g., product, order management team).
- Review source system documentation:
Is `order\_id` a primary key in the source database (e.g., `orders` table in PostgreSQL)?

- Query raw layer:

```
```sql
SELECT order_id, COUNT()
FROM raw.orders
GROUP BY order_id
HAVING COUNT() > 1
LIMIT 5;
```
```

-> If duplicates exist in `raw`, deduplication must happen in transformation.

? 3. Review the `fct\_orders` Model Logic

Why: The model may inadvertently create duplicates via:

- Joining to a table with multiple matches (e.g., line items -> orders),
- Missing `GROUP BY` or `DISTINCT`,
- Using `LEFT JOIN` without deduplication on the right side.

How to verify:

- Open `models/marts/fct\_orders.sql`.
- Look for:
 - Joins that could fan out (e.g., orders -> line items -> shipments),
 - Absence of `ROW\_NUMBER()` or `QUALIFY` for deduplication,
 - No `WHERE` clause filtering out stale/overwritten records.

Example red flag:

```
```sql
SELECT o.order_id, o.amount, i.item_name
FROM stg_orders o
LEFT JOIN stg_order_items i ON o.order_id = i.order_id -- Creates 1:N
```
```

-> This will produce one row per item -> multiple rows per `order\_id`.

? 4. Validate the Fix on a Subset or Dev Environment

Why: Never apply deduplication logic blindly - you might drop the wrong version of an order (e.g., keep the oldest, not the latest).

How to verify:

1. Run the proposed deduplication logic in dev:

```
```sql
SELECT ,
 ROW_NUMBER() OVER (
 PARTITION BY order_id
 ORDER BY updated_at DESC, _etl_loaded_at DESC
) AS rn
FROM analytics.stg_orders
```

**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

6:54 AM - Responded in 1m 25s

Excellent question. When addressing a data integrity issue like duplicate `order\_id`'s in a production fact table, the stakes are high - incorrect fixes can corrupt business metrics (e.g., revenue, order counts), break downstream dashboards, or violate audit requirements.

Here's a clear breakdown of top risks, followed by a robust rollback plan.

---

### ## ?? Top Risks of Fixing the Duplicate Issue

#### ### 1. Incorrect Version Retention

- Risk: Deduplicating by `updated\_at DESC` may keep the latest version - but what if it's a partial update (e.g., only status changed, amount is invalid)?
- Impact: We preserve corrupted or incomplete data, leading to inaccurate reporting.
- Likelihood: Medium (depends on source data quality).

#### ### 2. Over-Deduplication / Data Loss

- Risk: Aggressively filtering or grouping may drop legitimate variations (e.g., split shipments, order corrections).
- Impact: Orders disappear or are misrepresented in KPIs (e.g., GMV drops unexpectedly).
- Example: Two rows with same `order\_id` but different `order\_value` - deduplication keeps one arbitrarily.

#### ### 3. Breaking Downstream Dependencies

- Risk: Models, dashboards, or apps assume `order\_id` is unique and join directly to `fct\_orders`.
- If we change row count drastically (e.g., from 1M -> 200K rows), those models may break or misreport.
- Impact: Key reports show zero sales, or incremental models fail due to missing expected keys.

#### ### 4. Full Refresh Side Effects

- Risk: Running `--full-refresh` on `fct\_orders` might:
  - Lock the table for minutes/hours.
  - Trigger cascading large refreshes in downstream incremental models.
  - Exceed warehouse compute limits or SLAs.
- Impact: BI dashboards time out; pipeline alerts fire.

#### ### 5. Misdiagnosed Root Cause

- Risk: Duplicates were introduced upstream (e.g., in `stg\_orders`), and we fix `fct\_orders` only to have duplicates reappear in next run.
- Impact: Test passes temporarily -> fails again later -> loss of trust in data.

---

### ## ?? Rollback Plan (If Fix Fails)

We treat this like a production deployment: prep, monitor, and plan for reversal.

#### ### ? Pre-Rollback Preparation (Before Applying Fix)

##### 1. Snapshot Current State

```
```sql
```

```
CREATE TABLE analytics.fct_orders_pre_dedup AS
SELECT FROM analytics.fct_orders;
```

```
```
```

- Keep for 7+ days.
- Include metadata (e.g., `\_etl\_run\_id`, `dbt\_version`).

## 2. Tag the Deployment

- Use `dbt --vars 'deployment\_id: "fix-order-dups-v1"'` or annotate in DAG tool (e.g., Airflow run ID).
- Helps trace impact.

## 3. Pause Downstream Schedules (Optional for High Risk)

- Pause non-critical downstream models during rollout.
- Resume only after verification.

---

### ### ? Rollback Steps (If Something Goes Wrong)

#### #### ? Option 1: Restore from Snapshot (Fastest, Safest)

```
``sql
```

```
-- Revert table directly (if using table materialization)
```

```
DROP TABLE analytics.fct_orders;
```

```
CREATE TABLE analytics.fct_orders AS
```

```
SELECT FROM analytics.fct_orders_pre_dedup;
```

```
``
```

> ?? Only works if no upstream dependencies have been rebuilt.

Then:

```
``bash
```

```
Re-run test to confirm original state
```

```
dbt test -m fct_orders
```

```
``
```

#### #### ? Option

**Engineering counterpart**

**6:54 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.