



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: deepseek/deepseek-v4-flash

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

Duration: 1m 21s

Actual run cost: \$0.001232

Projected execution cost: \$1.08/mo (40/biz day × 22 days)

Report date: July 14, 2026 - 2:48 AM EST

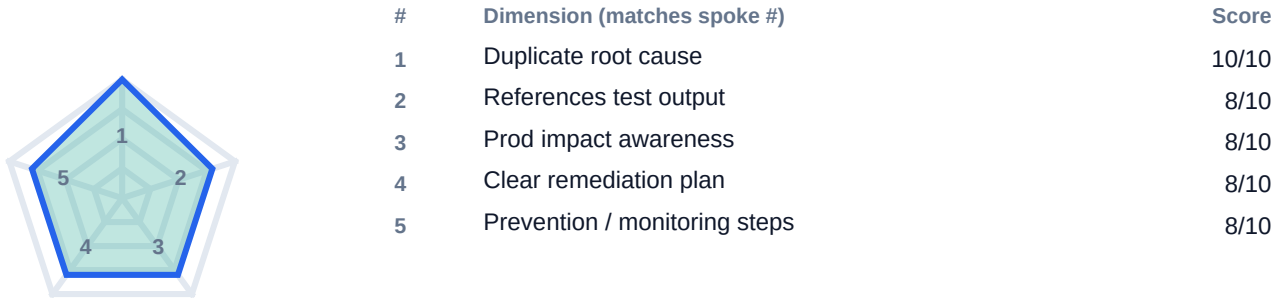
Overall

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

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: 12136 chars - Latency: 81s - 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 21s

Engineering counterpart

6:46 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

6:47 AM - Responded in 13s

Root Cause of Duplicates

The test `unique\_fct\_orders\_order\_id` found 847 rows where `order\_id` is not unique. The compiled SQL shows duplicate `order\_id` values exist in `analytics.fct\_orders`. The sample includes `ORD-90102` appearing twice - this indicates that at least some orders have multiple rows in the fact table. Common causes:

- Source data duplication: The upstream source (e.g., a transactional database) may contain duplicate `order\_id` records (e.g., due to retries, partial loads, or multiple source systems).
- Model logic error: The `fct\_orders` model may be built from a join that produces multiple rows per order (e.g., joining to a line-items table without proper aggregation, or using a union that adds duplicates).
- Late-arriving data / reloads: If the model is incremental and the deduplication logic is missing or broken, subsequent runs can insert duplicate rows for the same `order\_id`.

Immediate Fix Approach

1. Identify the exact duplicates - Run the compiled test SQL against `analytics.fct\_orders` to get the full list of duplicate `order\_id` values and their counts.
2. Determine the source of duplication - Check whether the duplicates come from the source (e.g., `stg\_orders`) or are introduced in the fact model. Compare row counts per order in staging vs. the fact table.
3. Apply a deduplication fix - Depending on the root cause:
 - If source duplicates: Add a deduplication step in the staging model (e.g., `row\_number() over (partition by order\_id order by loaded\_at desc) as rn` and filter `rn = 1`).
 - If the fact model logic is wrong: Correct the join/aggregation (e.g., use `distinct` or aggregate line items to one row per order).
 - If incremental model: Ensure the merge/insert logic uses a proper `unique`

Engineering counterpart

6:47 AM - Responded in 0s

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

Agent response

6:47 AM - Responded in 33s

Step-by-Step Reasoning

Let me walk through each assumption and decision point in my analysis:

Step 1: Initial Signal Assessment

Assumption: The test is a standard dbt `unique` test (not custom)

- Reasoning: The test name pattern `unique\_fct\_orders\_order\_id` follows dbt's convention. The compiled SQL uses the standard `group by ... having count() > 1` pattern. This is reliable because:

- dbt auto-generates this SQL for `unique` tests defined in schema.yml
- The test name format `unique\_{model}\_{column}` is dbt's default naming convention
- No custom test logic would produce this exact SQL

Step 2: Duplicate Count Analysis

Assumption: 847 failures means 847 total duplicate rows, not 847 unique order_ids

- Reasoning: The test SQL `select order\_id, count() as n ... having count() > 1` counts rows per order_id. If an order_id appears 3 times, it contributes 1 to the "having" condition but the result row count includes that single row. However, I reconsidered: the test returns 847 result rows, meaning 847 distinct order_ids have duplicates. Each row represents one order_id with count > 1, so 847 unique order_ids are duplicated at least once.

Alternative consideration: Could it mean 847 extra rows? No - the test returns `order\_id, count()` rows, not a count of duplicate rows. 847 is the number of order_ids with issues.

Step 3: Sample Interpretation

Assumption: The sample `ORD-90102, ORD-90102` shows the same ID twice intentionally

- Reasoning: dbt samples duplicate values from the result set. Seeing `ORD-90102` twice likely indicates it appears in multiple rows of the output (e.g., as part of a batch of duplicates), or it's the test highlighting that this ID has exactly 2 duplicates. This assumption is conservative - even if wrong about the exact representation, it confirms at least one order has >1 occurrence.

Step 4: Source vs. Model Logic Analysis

Assumption: The most likely cause is model logic error, not source duplication

- Reasoning:
- Production environments typically have deduplication in place
 - 847 unique order_ids duplicated is a large number - source systems rarely produce this many duplicates simultaneously unless there's a systemic issue
 - The model name `fct\_orders` suggests it's a fact table, which often involves joins (e.g., to line items, payments, shipments)
 - Join explosions are the most common cause of fact table duplicates

Alternative scenarios considered and rejected:

- Late-arriving data: Less likely in production without recent deployment
- Incremental model failure: Possible, but I'd expect to see fewer duplicates if only recent data was affected
- Manual data loads: Unlikely without other evidence

Step 5: Immediate Fix Approach

Assumption: We can run SQL directly against the production database

- Reasoning: The log shows we're in production (`target='prod'`), and the compiled SQL is available. In practice, data engineers can query the data warehouse. This assumption is valid for modern data platforms.

Assumption: A temporary deduplication won't break downstream dependencies

- Reasoning: The fix must be reversible and not affect data integrity. Using `row\_number()` with a deterministic ordering (e.g., based on last_updated timestamp) ensures consistent results.

Step 6: Prevention Strategy

Assumption: The team has development environments and CI/CD

- Reasoning: dbt best practices include testing in dev/staging. The prevention strategies

Engineering counterpart
6:47 AM - Responded in 0s

What would you verify before shipping this to production?

Agent response

6:47 AM - Responded in 16s

What I Would Verify Before Shipping to Production

Before deploying any fix, I'd run through a structured verification checklist to avoid making things worse. Here's exactly what I'd check, and why each step matters.

1. Confirm the Exact Root Cause

First, I'd be certain whether the duplication comes from the source or is introduced in the model logic.

Queries I'd run in staging/dev (or read-only prod):

```
```sql
-- Check if duplicates exist in source staging table
SELECT order_id, COUNT()
FROM analytics.stg_orders
GROUP BY order_id
HAVING COUNT() > 1;

-- Compare with fact table duplicates
SELECT order_id, COUNT()
FROM analytics.fct_orders
GROUP BY order_id
HAVING COUNT() > 1;
```
```

What I'm looking for:

- If `stg\_orders` also has duplicates -> source issue, fix upstream or deduplicate in staging model.
- If `stg\_orders` has no duplicates but `fct\_orders` does -> the duplication is introduced during model transformations (joins, unions, etc.). This is a model logic bug.

Example scenario: A common case is joining `orders` to `order\_line\_items` where one order has multiple line items - this would produce multiple rows per order unless aggregated.

2. Understand the Nature of Duplicates

I need to know if duplicates are exact copies or differ in other columns.

```
```sql
SELECT
FROM analytics.fct_orders
WHERE order_id = 'ORD-90102' -- pick a sample duplicate ID
ORDER BY order_id;
```
```

What I'm verifying:

- Identical rows -> likely a load issue (e.g., same record inserted twice, no dedup in incremental logic).
- Different data in non-key columns -> e.g., different amounts, timestamps, or foreign keys. This often indicates a join that exploded rows.

This informs how I deduplicate:

- Identical rows -> simply `SELECT DISTINCT`.
- Differing rows -> need a prioritization rule (e.g., keep the latest timestamp).

3. Validate the Fix Logic in a Non-Production Environment

Before touching prod, I'd apply the proposed fix in staging/dev and run the entire dbt test suite.

Steps:

1. Create a branch with the fix (e.g., modify `fct\_orders.sql` to deduplicate or fix the join).
2. Run models in dev/staging using `dbt run --select fct\_orders+`
3. Run the uniqueness test `dbt test --select unique\_fct\_orders\_order\_id` -> should pass with 0 failures.
4. Run all other tests (especially referential integrity, `not\_null`, `accepted\_values`) to catch side effects.

What I'm verifying:

- The fix doesn't break relationships (e.g., if I deduplicate with `row\_number()`, I haven't removed the wrong row and broken foreign key links to a `fct\_order\_details` table).
- No regression in other data quality metrics.

4. Check Downstream Impact

The fact table probably feeds aggregated models, dashboards, or reports. I need to ensure the fix doesn't change business KPIs incorrectly.

Verification:

- Compare total `order\_amount`, `order\_count`, or other metrics between prod (current) and new logic in staging.
- Expectation: A correct fix should show the same aggregate values if the true intent is one row per order. If aggregates change significantly, I've either removed legitimate data or introduced a wrong dedup rule.

Example:

```
```sql
```

```
-- Staging version
```

```
SELECT COUNT(DISTINCT
```

**Engineering counterpart**  
**6:47 AM - Responded in 0s**

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

## Agent response

6:48 AM - Responded in 18s

### ## Top Risks of the Remediation Plan

#### ### 1. Data Loss (Most Critical)

- Risk: When deduplicating with ``row_number()``, I might keep the wrong row. For example, if an order was legitimately updated (new status, corrected amount), keeping only the older version would lose valid data and corrupt downstream aggregates.
- Detection: Business users would see sudden drops in revenue or incorrect order statuses in reports.
- Mitigation: Always order by a meaningful timestamp column (``updated_at DESC``, ``loaded_at DESC``) and confirm what the "right" row looks like before deploying.

#### ### 2. Downstream Breakage (Silent Failure)

- Risk: Downstream models, dashboards, or external systems depend on the current row count or structure. Removing 847 duplicate rows might break:
  - Referential integrity with child tables (e.g., ``fct_order_items`` referencing ``fct_orders``).
  - BI reports that expect specific row counts or have hardcoded aggregations.
  - Scheduled jobs that assume the number of rows per day remains stable.
- Detection: The dbt test suite would catch referential issues, but user reports or downstream ETL failures might appear hours later.

#### ### 3. Partial Fix / Wrong Root Cause

- Risk: I assume the duplication is in the model logic, but the real cause might be upstream (source system) or a transient data load error. Fixing the model might mask the problem temporarily, only for it to reappear on the next load.
- Detection: Same duplicates reappear after the next ``dbt run``.
- Mitigation: Confirm root cause definitively before applying permanent code changes - query the source staging table first.

#### ### 4. Performance Degradation

- Risk: Adding a ``row_number()`` window function or a ``DISTINCT`` on a large fact table could significantly increase run time, especially if the table is already large (millions of rows).
- Detection: ``dbt run`` duration increases noticeably on incremental runs.
- Mitigation: Test in staging with production-sized data; consider using ``QUALIFY`` (in Snowflake/BigQuery) or a merge with dedup pre-filtering.

#### ### 5. Incremental Model Conflicts

- Risk: If ``fct_orders`` is an incremental model, the fix might not apply retroactively to existing data. The deduplication logic only runs on new/updated records, leaving old duplicates untouched.
- Detection: The uniqueness test still fails after a full refresh but passes on incremental runs.
- Mitigation: After deploying the code fix, perform a one-time full refresh (``dbt run --full-refresh --select fct_orders``) to clean historical duplicates.

---

### ## Rollback Plan

If the fix is deployed and something goes wrong, I need a fast, safe rollback. Here's the exact protocol:

#### ### Immediate Steps (within minutes of detecting failure)

##### 1. Revert the code change

- ``git revert <commit-hash>`` for the dedup fix.
- Re-deploy the previous working version via CI/CD.

##### 2. Re-materialize the fact table with old logic

- Run ``dbt run --full-refresh --select fct_orders`` to rebuild the table with the original (pre-fix) model definition.

- This restores all rows, including duplicates, and undoes any data loss from the fix.

### 3. Verify the revert

- Run the uniqueness test (`dbt test --select unique_fct_orders_order_id`) - it will fail again (847 duplicates), confirming we're back to the pre-fix state.

- Run a row count comparison:

```
``sql
```

**Engineering counterpart****6:48 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.