



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-chat

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

Duration: 1m 30s

Actual run cost: \$0.002985

Projected execution cost: \$2.63/mo (40/biz day × 22 days)

Report date: July 13, 2026 - 10:26 PM EST

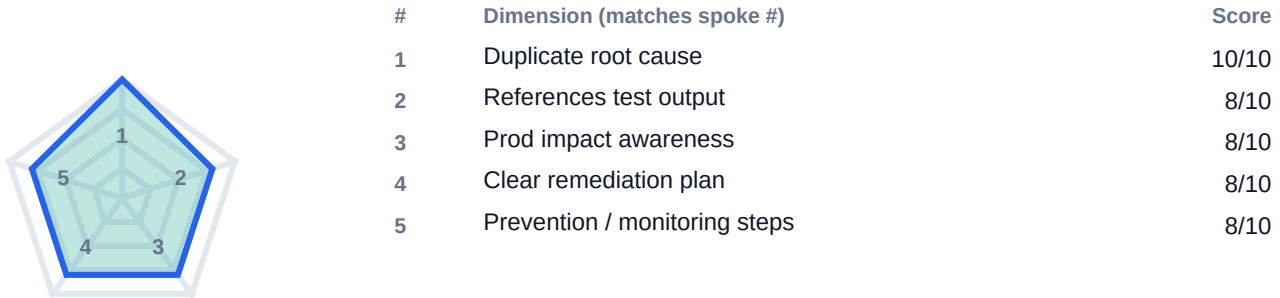
Overall

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: 10930 chars - Latency: 90s - 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 30s

Engineering counterpart

2:24 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

2:25 AM - Responded in 15s

Root Cause Analysis

1. Test Failure: The uniqueness test on `fct\_orders.order\_id` found 847 duplicate order IDs (e.g., `ORD-90102` appears twice in the sample).
2. Likely Causes:
 - Data Pipeline Issue: Duplicate records ingested from source (e.g., API/webhook retries, CDC misconfiguration).
 - ETL Logic Bug: Transformation logic (e.g., joins/grouping) in the `fct\_orders` model incorrectly duplicates records.
 - Late-Arriving Data: Historical data reloads without deduplication checks.

Immediate Fix Approach

1. Diagnose the Source:

```
```sql
-- Identify duplicate records (not just IDs)
SELECT
FROM analytics.fct_orders
WHERE order_id IN ('ORD-88421', 'ORD-90102')
ORDER BY order_id, created_at;
```
```

- Check timestamps, source system IDs, or metadata columns to determine if duplicates are legitimate (e.g., updated records) or erroneous.

2. Hotfix (if safe):

```
```sql
-- Example: Keep only the latest record per order_id
DELETE FROM analytics.fct_orders
WHERE (order_id, created_at) NOT IN (
 SELECT order_id, MAX(created_at)
 FROM analytics.fct_orders
 GROUP BY order_id
);
```
```

Note: Backup first and validate business logic before deleting.

3. Patch the dbt Model:

Add deduplication logic to the `fct\_orders` model:

```
```sql
-- models/marts/fct_orders.sql
WITH raw_orders AS (
 SELECT ,
 ROW_NUMBER() OVER (PARTITION BY order_id ORDER BY created_at DESC) as rn
 FROM {{ ref('stg_orders') }}
)
SELECT EXCEPT(rn)
FROM raw_orders
WHERE rn = 1
```
```

Prevention Plan

1. Upstream Safeguards:

- Add uniqueness tests to staging models (`stg\_orders`).
- Implement CDC checks in ingestion pipelines (e.g., `\_hash\_key` comparisons).

2. Monitoring:

- Add a `dbt test --store-failures` job to track duplicate trends.

- Set up alerts for sudden spikes in test failures.

3. Process Improvement:

- Document data contracts with source systems (expected uniqueness keys).
- Add data quality checks in the ingestion layer (e.g., Airflow checks before dbt runs).

Priority: Fix the model logic first (prevents recurrence), then clean existing duplicates during low-traffic hours.

Engineering counterpart
2:25 AM - Responded in 0s

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

Agent response

2:25 AM - Responded in 33s

Step-by-Step Reasoning

1. Understand the Test Failure

- Test Output: The uniqueness test `unique\_fct\_orders\_order\_id` failed with 847 duplicates for `order\_id`.
- Sample Duplicates: `ORD-88421`, `ORD-90102` (the latter appears twice).
- Compiled SQL:

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

This query identifies duplicate `order\_id` values.

Assumption: The test is correctly implemented, and duplicates exist in the `fct\_orders` table.

2. Diagnose the Root Cause

Possible Causes:

- Data Ingestion:
 - Duplicate records could be introduced during data ingestion (e.g., API/webhook retries, CDC issues).
 - Example: If `order\_id` is not the primary key in the source system, duplicates might be loaded.
- ETL Logic:
 - The `fct\_orders` model might incorrectly duplicate records (e.g., incorrect joins, lack of deduplication).
 - Example: A left join could create duplicates if the right table has multiple matches.
- Late-Arriving Data:
 - Historical data reloads might reintroduce duplicates if deduplication logic is missing.

Assumption: The issue lies either in the ingestion pipeline or the `fct\_orders` model logic.

3. Identify Duplicate Records

To confirm, inspect full records for duplicate `order\_id` values:

```
```sql
SELECT
FROM analytics.fct_orders
WHERE order_id IN ('ORD-88421', 'ORD-90102')
ORDER BY order_id, created_at;
```
```

Assumption: Metadata columns like `created\_at` or `updated\_at` exist to help identify duplicates.

4. Immediate Fix

If duplicates are erroneous:

```
```sql
-- Keep only the latest record per order_id
```



```
DELETE FROM analytics.fct_orders
WHERE (order_id, created_at) NOT IN (
 SELECT order_id, MAX(created_at)
 FROM analytics.fct_orders
 GROUP BY order_id
);

```

Assumption: The latest record (`MAX(created\_at)`) is the correct one. This might not hold for all use cases (e.g., updates).

---

#### #### 5. Patch the dbt Model

Prevent duplicates by adding deduplication logic to `fct\_orders`:

```
```sql
WITH raw_orders AS (
  SELECT ,
    ROW_NUMBER() OVER (PARTITION BY order_id ORDER BY created_at DESC) as rn
  FROM {{ ref('stg_orders') }}
)
SELECT EXCEPT(rn)
FROM raw_orders
WHERE rn = 1
---
```

Assumption: The `stg\_orders` table contains `created\_at` and doesn't already deduplicate records.

6. Prevent Recurrence

Upstream Safeguards:

- Add uniqueness tests to `stg\_orders`.
- Validate data contracts with source systems.

Monitoring:

- Use `dbt test --store-failures` to track duplicates over time.
- Set up alerts for sudden spikes in test failures.

Process Improvement:

- Document expected uniqueness keys for each source system.
- Add data quality checks in the ingestion layer.

Assumption: The team has control over the ingestion pipeline and can implement safeguards.

Key Assumptions

1. The `fct\_orders` table has metadata columns like `created\_at` to deduplicate records.
2. The latest record (`MAX(created\_at)`) is the correct one to keep.
3. The `stg_

Engineering counterpart
2:25 AM - Responded in 0s

What would you verify before shipping this to production?

Agent response

2:26 AM - Responded in 23s

Before shipping any fixes to production, I'd verify the following key areas to ensure the solution is robust and won't introduce new issues:

1. Verify the Root Cause

- Confirm duplicates are real and not a test artifact
 - Run the test query manually in production to validate the duplicates (``SELECT order_id, COUNT() FROM fct_orders GROUP BY 1 HAVING COUNT() > 1``).
 - Check if the duplicates are legitimate (e.g., business logic allows multiple records per ``order_id`` for updates).

- Inspect sample duplicates

```
```sql
SELECT
FROM fct_orders
WHERE order_id IN ('ORD-88421', 'ORD-90102')
ORDER BY order_id, created_at;
```
```

- Look for differences in metadata (``created_at``, ``updated_at``, ``source_system_id``).
- Check if duplicates are identical (indicating an ingestion bug) or variations (e.g., updates).

Assumption: If all columns except ``order_id`` are identical, it's likely a pipeline issue. If some fields differ, it might be intentional (e.g., order updates).

2. Validate the Fix Logic

- Test deduplication logic in staging
 - Apply the ``ROW_NUMBER()`` fix to a subset of data in a dev environment and verify:

```
```sql
WITH deduped AS (
 SELECT ,
 ROW_NUMBER() OVER (PARTITION BY order_id ORDER BY created_at DESC) as rn
 FROM fct_orders
 WHERE order_id IN ('ORD-88421', 'ORD-90102')
)
SELECT FROM deduped WHERE rn = 1;
```
```

- Ensure the correct record is kept (e.g., latest ``created_at``).

- Check edge cases
 - Null ``order_id`` values.
 - Records with identical ``order_id`` and ``created_at`` (add a tiebreaker like ``updated_at`` or ``id``).

Assumption: ``created_at`` is reliable for determining the "correct" record. If not, adjust the ``ORDER BY`` clause (e.g., ``ORDER BY created_at DESC, updated_at DESC``).

3. Backfill Safely

- Dry-run the DELETE/UPDATE
 - For the hotfix, test the deletion logic in a transaction first:

```
```sql
```

```

BEGIN;
-- Preview records to delete
SELECT order_id, created_at
FROM fct_orders
WHERE (order_id, created_at) NOT IN (
 SELECT order_id, MAX(created_at)
 FROM fct_orders
 GROUP BY order_id
);
-- ROLLBACK; -- Verify before committing
-- COMMIT; -- Only if the preview looks correct

- Ensure no unintended records are removed.

```

Assumption: The business can tolerate deleting duplicates. If not, consider soft-deleting or flagging them instead.

---

#### ### 4. Test Downstream Impact

- Check dependent models/reports
  - Run downstream models (e.g., `metrics\_orders`, `customer\_lifetime\_value`) with the fixed data to ensure no breaks.
  - Verify key dashboards (e.g., order volume, revenue) still match expectations.
- Compare row counts before/after
  - Example:
 

```

```sql
-- Before fix
SELECT COUNT() FROM fct_orders; -- 1,000,000
-- After fix
SELECT COUNT() FROM fct_orders; -- 999,153 (847 duplicates removed)
```

```
  - Ensure the count delta matches

**Engineering counterpart**  
**2:26 AM - Responded in 0s**

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

## Agent response

2:26 AM - Responded in 20s

### ### Top Risks & Mitigations

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

- Risk: The `DELETE` or deduplication logic accidentally removes valid records.
- Mitigation:
  - Backup the `fct\_orders` table before making changes:

```
```sql
CREATE TABLE backup.fct_orders_YYYYMMDD AS
SELECT * FROM analytics.fct_orders;
```
```
  - Use a transaction with `ROLLBACK` for dry-run validation.

#### #### 2. Downstream Breakage

- Risk: Reports or models relying on `fct\_orders` break due to missing/updated data.
- Mitigation:
  - Test downstream models in staging with the fixed data.
  - Notify stakeholders of potential metric shifts (e.g., "Order counts may drop by 847 due to deduplication").

#### #### 3. Performance Degradation

- Risk: The `ROW\_NUMBER()` deduplication slows down model execution.
- Mitigation:
  - Test the model's runtime in staging with production-like data volume.
  - Add an index on `order\_id` if missing:

```
```sql
CREATE INDEX IF NOT EXISTS idx_fct_orders_order_id ON analytics.fct_orders (order_id);
```
```

#### #### 4. Reintroduction of Duplicates

- Risk: The root cause (e.g., ingestion pipeline bug) isn't fixed, and duplicates return.
- Mitigation:
  - Add a `dbt test` to `stg\_orders` to catch duplicates earlier.
  - Monitor `fct\_orders` for duplicates daily.

---

### ### Rollback Plan

If the fix causes issues:

#### #### 1. Revert the Data

```
```sql
-- Restore from backup (if DELETE was used)
TRUNCATE analytics.fct_orders;
INSERT INTO analytics.fct_orders
SELECT * FROM backup.fct_orders_YYYYMMDD;
```
```

#### #### 2. Revert the dbt Model

- Rollback the `fct\_orders.sql` change in Git and redeploy:

```
```bash
git revert <commit-hash>
dbt run --models fct_orders
```
```

#### #### 3. Communicate

- Alert stakeholders:

- > "Rollback initiated: Deduplication caused unexpected issues. Investigating root cause."

#### #### 4. Post-Mortem

- Document:

- Why the fix failed (e.g., incorrect `ORDER BY` logic, missing edge cases).
  - Steps to improve testing for future changes.

---

#### ### Key Takeaways

1. Always backup data before bulk `DELETE`/`UPDATE`.
2. Test in staging with production-like data.
3. Monitor aggressively post-deploy for anomalies.
4. Have a rollback playbook-know how to revert code and data quickly.

Would you like to walk through the rollback steps in a dry-run?

**Engineering counterpart**

**2:26 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.