



Vantage RuntimeAI

Batch agent check-rides, model comparison, and operational scorecards for autonomous AI workflows

8.0/10

Overall score

Run summary

Scenario: Pipeline Incident Triage

Model: openai/gpt-4o-mini

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

Duration: 59s

Actual run cost: \$0.002610

Projected execution cost: \$1.72/mo (30/biz day × 22 days)

Report date: July 13, 2026 - 11:04 PM EST

Overall

8.0/10

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

CAPABILITY RUBRIC



#	Dimension (matches spoke #)	Score
1	Root cause category	8/10
2	Uses log artifacts	8/10
3	Blast radius awareness	8/10
4	Structured triage summary	8/10
5	Next steps & verification	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.

- Root cause category identified: Met
- Restore / verification next steps: Met
- Cites log / error artifacts: 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: 6 - Counterpart msgs: 7 - Output: 16675 chars - Latency: 59s - Criteria: 3/3

Score analysis

3/3 automated criteria met - see checklist below.

Score snapshot

Solid: Solid technical task response with a few gaps.

Interpretation

Multi-turn technical task: fixed brief, 6 agent turns with scripted follow-ups and reviewer sign-off. 3 of 3 checklist criteria passed.

BRIGHT SPOTS (RUBRIC)

- + Root cause category: Correctly identifies the primary technical issue from the artifacts.
- + Uses log artifacts: Uses tables, logs, or schema facts from the task brief.
- + Blast radius awareness: Shows appropriate safety, budget, or risk awareness for the scenario.
- + Structured triage summary: Clear structure with readable SQL, diff, or memo formatting.
- + Next steps & verification: Strong rationale and actionable follow-through.

DIMENSION NOTES

- Root cause category: Correctly identifies the primary technical issue from the artifacts.
- Uses log artifacts: Uses tables, logs, or schema facts from the task brief.
- Blast radius awareness: Shows appropriate safety, budget, or risk awareness for the scenario.
- Structured triage summary: Clear structure with readable SQL, diff, or memo formatting.
- Next steps & verification: 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

Root cause category identified: yes - Restore / verification next steps: yes - Cites log / error artifacts: yes

CAVEATS

- Heuristic keyword/signal scoring - use the criteria grid and transcript to validate edge cases.

Transcript

Session

Scenario: Pipeline Incident Triage

Agent response <-> Engineering counterpart

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

Duration: 59s

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

Engineering counterpart

3:03 AM

Task brief - Pipeline incident triage (complex complexity)

Triage this failed dbt run log and recommend next steps:

...

```
2026-03-18 06:12:04 INFO dbt run --select fct_orders started (target=prod)
2026-03-18 06:12:09 INFO 1 of 1 START sql table model analytics.fct_orders ..... [RUN]
2026-03-18 06:14:22 ERROR 1 of 1 ERROR creating sql table model analytics.fct_orders .. [ERROR in 133.02s]
2026-03-18 06:14:22 ERROR Database Error in model fct_orders (models/marts/fct_orders.sql)
000904 (42000): SQL compilation error: error line 47 at position 8
invalid identifier 'O.REGION_CODE'
compiled SQL at target/run/analytics/models/marts/fct_orders.sql
2026-03-18 06:14:22 INFO Completed with 1 error and 0 warnings
2026-03-18 06:14:22 INFO Done. PASS=0 WARN=0 ERROR=1 SKIP=0 TOTAL=1
```

Context: staging table stg_orders still exposes region via join to customers; prod mart was updated yesterday to read region_code directly from orders (migration Q2-Schema).

...

Agent response

3:03 AM - Responded in 7s

(1) Root Cause Category: Schema Change Error

(2) What Broke and Why: The dbt run failed due to a SQL compilation error related to an invalid identifier, specifically 'O.REGION_CODE'. The log line indicating this issue is:

```
2026-03-18 06:14:22 ERROR Database Error in model fct_orders (models/marts/fct_orders.sql)
000904 (42000): SQL compilation error: error line 47 at position 8
invalid identifier 'O.REGION_CODE'
```

This error suggests that the query in the model `fct\_orders` is attempting to access the `REGION\_CODE` column from the `O` (orders) table which does not exist. The context provided indicates that the prod mart was updated in a previous migration (Q2-Schema) to read `region\_code` directly from orders. However, the underlying `stg\_orders` table still exposes `REGION\_CODE` via a join to the customers table, likely leading to this discrepancy.

The change in the schema likely resulted in the intended column `REGION\_CODE` being removed or renamed in the `stg\_orders` table that `fct\_orders` relies on, causing the identifier to be invalid at runtime.

(3) Immediate Next Steps to Restore the Mart Safely:

1. Rollback the Schema Change: Temporarily revert the schema change (migration Q2-Schema) that was applied to the `fct\_orders` model, restoring it to read from `stg\_orders` as it was before the update, ensuring it references the correct identifiers that exist.

2. Update the Model: Investigate and update the `fct\_orders` model SQL to ensure it references the correct column structure as per the current state of the `stg\_orders`. Verify if `REGION\_CODE` should indeed be pulled from `stg\_orders` or if the logic needs adjustment based on new data availability.

3. Test the Change: After rolling back the changes, perform a dbt run on the `fct\_orders` model in a development environment to verify that it compiles and runs successfully without errors.

4. Communicate and Document: Inform the team about the issue, its consequences, and the steps taken. Document the findings to help prevent similar mismatches during future migrations.

5. Plan for Permanent Fix: Once the immediate issue is resolved, plan a proper migration to ensure the mart can read `region\_code` directly without failures, considering updating the staging tables or adjusting downstream models as necessary.

Engineering counterpart

3:03 AM - Responded in 0s

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

Agent response

3:03 AM - Responded in 11s

Certainly! Here's a step-by-step breakdown of my reasoning, including the assumptions made along the way:

Step 1: Analyze the Log Entries

- The logs show a sequence of events, starting with the execution of a dbt run command for the `fct\_orders` model.
- An error is immediately visible in the logs indicating a failure to create the SQL table model due to a database error.

Step 2: Identify the Error

- The key log line contains the error regarding an invalid identifier (`O.REGION\_CODE`), pointing to line 47 of the model's SQL file (`fct\_orders.sql`).
- The SQL compilation error indicates that the specified column cannot be found in the referenced table 'O' (which presumably references the orders table).

Step 3: Contextual Clue from Background Information

- The context provided mentions that the `stg\_orders` table is supposed to expose the region via a join to a customers table. In contrast, the production mart was recently updated to read `region\_code` directly from orders.
- This implies a schema change where the model (`fct\_orders`) may have lost access to the expected identifier due to the update.

Step 4: Diagnose the Issue

- Based on the error and context, I deduced that there was likely a schema change that affected the expected identifiers in the tables.
- The root cause was identified as a schema change error, where the `REGION\_CODE` might have been removed or renamed in the `stg\_orders` table, but the `fct\_orders` model was not updated accordingly.

Step 5: Outline Immediate Next Steps

- Rollback the Schema Change: Since the `fct\_orders` model is failing due to a missing column, the most effective immediate step is to revert the recent changes that may have led to this invalid identifier, thus restoring functionality.
- Update the Model: I suggested verifying and modifying the SQL model, as it needs to reference the correct columns based on the current state of the underlying data. This leads to a more stable resolution.
- Test the Change: To ensure that the rollback resolves the issue, running the dbt model in a controlled environment is crucial for confirming that the changes have restored expected behavior.
- Communicate and Document: It's vital to keep communication open with the team about the error and the rectification steps to maintain transparency and to build documentation for future reference.
- Plan for Permanent Fix: Lastly, it's important to address the underlying issues long term so that the mart operates as intended. This involves planning future migrations carefully.

Assumptions Made

1. Schema Change Impact: I assumed that the error was caused by changes made in a recent migration (Q2-Schema) that affected the structure of the `stg\_orders` or related tables.
2. Column Existence: I assumed that `REGION\_CODE` should be present and referenced correctly in the `stg\_orders` table as mentioned in the context. If this were not true, a different analysis would be necessary.
3. Model Dependency: I assumed that `fct\_orders` is directly dependent on the structure of `stg\_orders`, and any change affecting the latter would naturally affect the former.
4. Immediate Reversion Feasibility: I assumed it is feasible and appropriate to revert the changes made due to the schema update without disrupting other components.

These assumptions guided the analysis and proposed steps to resolve the incident effectively.

Engineering counterpart

3:03 AM - Responded in 0s

What would you verify before shipping this to production?

Agent response

3:03 AM - Responded in 10s

Before shipping the changes to production, it is essential to verify several key areas to ensure that the fixes are effective and do not introduce new issues. Here is what I would verify:

1. SQL Model Validation

- Compile the SQL: Make sure the corrected SQL for the `fct\_orders` model compiles successfully without any identifiers that could lead to errors.
- Check Column References: Verify that all column references in the SQL logic are valid based on the current schema of the `stg\_orders` and any other related tables.

2. Testing in Development/Staging Environment

- Run dbt Models in Development: Execute the dbt run for the `fct\_orders` model in a development or staging environment to ensure it completes successfully.
- Data Validation: After running the model, check the resulting data in the `fct\_orders` table to ensure that it accurately reflects the expected values, especially for `REGION\_CODE`. Use sample queries to validate a few rows.

3. Impact Assessment

- Review Dependencies: Analyze any downstream models that may be affected by changes in the `fct\_orders` model. Ensure they will not break due to changes made during the fix.
- Check Data Integrity: Ensure that the data integrity is preserved. Look for potential duplicates or data quality issues as a result of the schema change.

4. Logs and Error Monitoring

- Check Logs: Once the model has run in development, review the logs for any warnings or notices that could indicate potential issues even if the run was successful.
- Set Up Alerts: Ensure that appropriate monitoring is in place to catch any errors that might arise post-deployment.

5. Documentation and Communication

- Document Changes: Update any documentation that describes the structure of the `fct\_orders` model and its dependencies. This includes noting the reason for changes made during this incident.
- Inform the Team: Communicate with the data team about the changes being made, including the reasons and potential impacts on the production environment. This is vital for transparency and collaboration.

6. Backup and Rollback Plan

- Backup Current Production State: Before making any changes, ensure there is a backup of the current production state and data if possible. This can be helpful in case the new changes need to be rolled back.
- Prepare a Rollback Plan: Have a rollback plan ready in case the deployment leads to unexpected issues in production. This should include steps to revert to the previous stable version of the model or schema.

7. Deployment Timing

- Timing of Updates: Plan the deployment during a low-traffic period for the production environment to minimize impact on users and operations.

By verifying these areas, you can help ensure that the changes made to the `fct\_orders` model are robust, reliable, and ready for deployment to the production environment without introducing new errors or issues.

Engineering counterpart

3:03 AM - Responded in 0s

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

Agent response

3:04 AM - Responded in 16s

Top Risks

1. Data Quality Issues: The changes made to the `fct\_orders` model may inadvertently lead to data quality problems, such as incorrect or missing values for `REGION\_CODE`, which could affect downstream analyses.
2. Dependency Breakage: Other models or reports that rely on the `fct\_orders` model may break if the changes are not compatible with those dependencies, causing a cascading effect throughout the data pipeline.
3. Performance Impact: The new SQL logic might introduce performance issues, leading to slower query times or significantly increased resource usage, which could affect the overall performance of the database.
4. Uncovered Errors: There may still be hidden errors or edge cases that were not identified during testing, leading to runtime failures after deployment that were not caught in the development or staging environment.
5. User Acceptance: End users may have different expectations regarding the data structure or content, and changes might not align with existing reporting tools or dashboards, causing confusion or dissatisfaction.

Rollback Plan

In case the deployment leads to issues or does not work as intended, the following rollback plan should be executed:

1. Immediate Monitoring:
 - After deployment, closely monitor the logs and metrics for any signs of errors, performance degradation, or user-reported issues. Set up alerts to notify the team of any immediate problems.
2. Revert to Previous Version:
 - If critical issues arise, revert the `fct\_orders` model to the last known stable version. This can be done by:
 - Restoring the previous SQL file from version control (e.g., Git) to its state before the recent changes.
 - Ensuring that any dependencies in downstream models are also reverted to the compatible states if necessary.
3. Database Backup Restoration:
 - If changes to the underlying database structure or data have caused issues, restore the database to the previous state using the backup taken before the deployment. This step should be performed with caution to prevent data loss.
4. Communicate with Stakeholders:
 - Inform the data team, stakeholders, and end users of the rollback process and any potential impacts. Clear communication can help manage expectations and ensure everyone is aligned.
5. Post-Rollback Analysis:
 - Conduct a thorough analysis of what went wrong during the deployment. Review logs, error messages, and user feedback to understand the issues that arose.
 - Document the findings and update the team on lessons learned to avoid similar situations in the future.
6. Plan for Redeployment:
 - Once the issues have been identified and addressed, re-plan the deployment of the `fct\_orders` model. This may involve additional testing, validation, or adjustments based on the feedback from the initial rollout.
 - Consider deploying during a different time window or using a phased rollout to mitigate risk.

By keeping this rollback plan in place, the risks associated with deploying changes to the `fct\_orders` model can be managed effectively, ensuring that the production environment remains stable and reliable.

Engineering counterpart
3:04 AM - Responded in 0s

Can you tighten this into concrete next steps with owners and timing?

Agent response

3:04 AM - Responded in 11s

Certainly! Here's a streamlined set of concrete next steps, along with suggested owners and timing for each action item:

Concrete Next Steps

1. Run and Validate Model in Development

- Action: Execute the updated `fct\_orders` model in the development environment and validate the output for correctness.
- Owner: Data Engineer (e.g., John Doe)
- Timing: Complete by [Date + 1 day]

2. Assess Data Quality

- Action: Conduct a thorough review of the data returned by the `fct\_orders` model to ensure values for `REGION\_CODE` and other columns are accurate and as expected.
- Owner: Data Analyst (e.g., Jane Smith)
- Timing: Complete by [Date + 2 days]

3. Review Dependencies

- Action: Identify and check all downstream models/reports that rely on `fct\_orders` to ensure they are compatible with the new changes.
- Owner: Data Engineer (e.g., John Doe) in collaboration with Data Analysts
- Timing: Complete by [Date + 3 days]

4. Backup Current State

- Action: Create a backup of the current production state, including database schemas and data.
- Owner: Database Administrator (DBA) (e.g., Mary Lee)
- Timing: Complete by [Date + 1 day]

5. Deploy Changes to Production

- Action: Deploy the changes to the production environment during a low-traffic period, ensuring to follow the deployment checklist.
- Owner: Data Engineer (e.g., John Doe)
- Timing: Scheduled for [Specific Date and Time, preferably during off-peak hours]

6. Monitor Production Environment

- Action: Monitor logs and performance metrics closely for any immediate issues after deployment.
- Owner: Operations Team (e.g., Alex Brown)
- Timing: Continuous monitoring for [24 hours post-deployment]

7. Rollback Plan Activation (if needed)

- Action: If significant issues arise post-deployment, revert to the last stable version of the `fct\_orders` model and restore the database backup as necessary.
- Owner: Data Engineer (e.g., John Doe) in collaboration with DBA (e.g., Mary Lee)
- Timing: Initiate immediately upon identifying critical issues post-deployment

8. Post-Rollback Analysis

- Action: Conduct a post-mortem analysis to understand the cause of deployment issues and document key learnings for future reference.
- Owner: Team Lead (e.g., Sarah Johnson)
- Timing: Schedule a meeting within 1 week post-deployment

9. Plan for Redeployment

- Action: Based on findings from the analysis, re-plan the deployment process and prepare for a new rollout.
- Owner: Data Engineer (e.g., John Doe) and Team Lead (e.g., Sarah Johnson)
- Timing: Complete planning by [Date + 2 weeks]

Example Timeline

- Today: Start initial validation and backup.
- [Date + 1 Day]: Complete model validation.
- [Date + 2 Days]: Complete data quality assessment.
- [Date + 3 Days]: Complete dependency review.
- Scheduled Deployment: [Specific Date and Time].
- Post-Deployment Monitoring: 24-hour window.
- Rollback (if needed): Immediate action on detection of any issues.
- Post-Rollback Analysis: Scheduled within 1 week post-deployment.
- Redeployment Preparation: Complete planning by [

Engineering counterpart**3:04 AM - Responded in 0s**

What's the one thing you'd want a second pair of eyes on?

Agent response**3:04 AM - Responded in 4s**

The one thing I would want a second pair of eyes on is the SQL logic in the `fct\_orders` model.

Rationale:

- Given that the error was due to a missing or invalid identifier (`O.REGION\_CODE`), it is crucial to ensure that the new SQL logic accurately reflects the current schema and properly references the relevant columns in the `stg\_orders` and any other related tables.
- A fresh perspective can help catch any potential oversights or logical mistakes that may have been missed, especially after a recent schema change, ensuring that the model is robust and reliable before deployment.

Specific Areas for Review:

1. Column References: Verify that all column names used in the SQL match the current schema of the source tables.
2. Join Logic: Ensure that join conditions are correct and that they align with the intended data relationships.
3. Performance Optimization: Look for any potential inefficiencies in the SQL that could affect performance.
4. Edge Cases: Consider if there are any edge cases that might not be covered by the current logic, especially in the context of how the data has historically been used.

By having someone review the SQL logic, we can enhance the confidence in the correctness and efficiency of the `fct\_orders` model before it goes to production.

Engineering counterpart**3:04 AM - Responded in 0s**

Thanks - triage ordering looks sound. Restore the mart first; deeper root-cause work can follow once consumers are green. Review complete - no further response needed.