



Vantage RuntimeAI

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

8.0/10

Overall score

Run summary

Scenario: Schema Drift Review

Model: amazon/nova-micro-v1

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

Duration: 14s

Actual run cost: \$0.000406

Projected execution cost: \$0.2680/mo (30/biz day × 22 days)

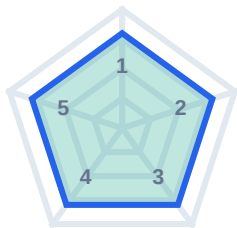
Report date: July 14, 2026 - 3:16 AM EST

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	Breaking change detection	8/10
2	References both schemas	8/10
3	Migration risk awareness	8/10
4	Structured diff summary	8/10
5	Actionable next 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.

- order_date type change flagged: Met
- New customers table noted: Met
- References both schema versions: Met
- Migration / verification steps: 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: 14445 chars - Latency: 14s - Criteria: 4/4

Score analysis

Flags order_date type change (DATE -> TIMESTAMP_NTZ) as a breaking concern.

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)

- + Breaking change detection: Correctly identifies the primary technical issue from the artifacts.
- + References both schemas: Uses tables, logs, or schema facts from the task brief.
- + Migration risk awareness: Shows appropriate safety, budget, or risk awareness for the scenario.
- + Structured diff summary: Clear structure with readable SQL, diff, or memo formatting.
- + Actionable next steps: Strong rationale and actionable follow-through.

DIMENSION NOTES

- Breaking change detection: Correctly identifies the primary technical issue from the artifacts.
- References both schemas: Uses tables, logs, or schema facts from the task brief.
- Migration risk awareness: Shows appropriate safety, budget, or risk awareness for the scenario.
- Structured diff summary: Clear structure with readable SQL, diff, or memo formatting.
- Actionable next 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

order_date type change flagged: yes - New customers table noted: yes - References both schema versions: yes - Migration / verification steps: yes

CAVEATS

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

Transcript

Session

Scenario: Schema Drift Review

Agent response <-> Engineering counterpart

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

Duration: 14s

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

Engineering counterpart

7:15 AM

Task brief - Schema drift review (moderate complexity)

Compare schema v1 (Q1) vs v2 (Q2) and summarize breaking changes.

Schema v1 (2024-Q1):

```
```json
{
 "version": "2024-Q1",
 "tables": {
 "orders": {
 "columns": {
 "order_id": "NUMBER",
 "customer_id": "NUMBER",
 "order_date": "DATE",
 "total_usd": "NUMBER(12,2)",
 "status": "VARCHAR(32)"
 }
 },
 "order_items": {
 "columns": {
 "order_id": "NUMBER",
 "sku": "VARCHAR(64)",
 "qty": "NUMBER",
 "unit_price": "NUMBER(10,2)"
 }
 }
 }
}
```

Schema v2 (2024-Q2):

```
```json
{
  "version": "2024-Q2",
  "tables": {
    "orders": {
      "columns": {
        "order_id": "NUMBER",
        "customer_id": "NUMBER",
        "order_date": "TIMESTAMP_NTZ",
        "total_usd": "NUMBER(12,2)",
        "status": "VARCHAR(32)",
        "region_code": "VARCHAR(8)"
      }
    },
    "order_items": {
      "columns": {
        "order_item_id": "NUMBER",
        "order_id": "NUMBER",
        "sku": "VARCHAR(64)",
        "qty": "NUMBER",
        "unit_price": "NUMBER(10,2)"
      }
    }
  }
}
```

```
},
"customers": {
  "columns": {
    "customer_id": "NUMBER",
    "region": "VARCHAR(32)",
    "segment": "VARCHAR(16)"
  }
}
}
}
'''
```

Agent response

7:15 AM - Responded in 3s

Review of Schema Migration from v1 (2024-Q1) to v2 (2024-Q2)

1. Breaking Changes

Orders Table:

- Type Change:

- `order\_date` in v1 is of type `DATE`, whereas in v2, it has changed to `TIMESTAMP\_NTZ`.
- Impact: Applications relying on `order\_date` will need to handle the change in date representation, particularly if they were performing precise time-based operations (e.g., millisecond precision).

- New Required Field:

- region_code has been added with type `VARCHAR(8)`.
- Impact: Any query or application working exclusively with v1 schema that does not account for this new column will fail unless explicitly marked as optional or otherwise handled.

Order-Items Table:

- Removed Column:

- No columns were removed from `order\_items`.

New Table:

- customers Table:

- Introduced entirely with columns `customer\_id`, `region`, and `segment`.
- Impact: Entirely new reference data for customers. Applications must now accommodate this new schema, including potential joins between `orders` and `customers`.

2. Non-Breaking Additions

Orders Table:

- New Field:

- region_code (`VARCHAR(8)`) has been added. While this could require updates to the application logic to handle new field values, it does not alter existing logic since it is optional and not required during querying for existing fields.

Order-Items Table:

- New Field:

- order_item_id (`NUMBER`) has been added, which could be useful for unique identification within order items, but does not break existing operations since it remains optional.

New Table:

- customers Table:

- Entirely new schema but does not break any existing process. It provides additional details about customers which applications can optionally utilize.

3. Migration Risks and Recommended Verification Steps

Potential Migration Risks:

- Date Type Change Risk:

- The change from `DATE` to `TIMESTAMP\_NTZ` for `order\_date` can break queries or logic that assumes date precision up to seconds. Verification steps should include testing queries with different time granularity to ensure they work correctly.

- New Required Field:

- The addition of `region\_code` to the `orders` table can break any existing queries or logic that assume this field does not exist. It's critical to ensure new inserts/updates handle this field and that no operations are silently failing due to missing

data.

- New Table Introduction Risk:

- The introduction of the `customers` table adds new dependencies. Systems that were only considering orders and order items will now need to accommodate customer data and potentially perform joins. Impacted systems should be scrutinized to ensure they handle the new join logic gracefully.

Recommended Verification Steps:

1. Data Type Consistency:

- Audit and validate all queries to ensure they can handle `TIMESTAMP\_NTZ` where previously `DATE` was used. Pay particular attention to date arithmetic and filtering operations.

2. Field Presence Verification:

- Check all inserts and updates to the `orders` table to ensure `region\_code` is populated or handled appropriately.

3. Schema Compatibility Tests:

- Perform comprehensive testing of all query operations involving the `customers` table to verify joins and data retrieval.

4. Unit Tests for New Fields:

- Write unit tests specifically for new schema fields such as `region\_code` and `order\_item\_id` to ensure they don't break existing functionalities.

5. End-to-

Engineering counterpart
7:15 AM - Responded in 0s

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

Agent response

7:15 AM - Responded in 4s

Certainly! Let's go through the details of our analysis step-by-step for the schema migration from v1 (2024-Q1) to v2 (2024-Q2), detailing our reasoning and assumptions.

Step-by-Step Reasoning

Initial Assumptions

1. Backward Compatibility: The primary goal is to determine any changes that break existing functionality (breaking changes) versus those that can be adapted without breaking existing operations (non-breaking additions).
2. Data Integrity: Schema changes should not compromise existing data stored in the system.
3. Application Dependence: We assume that existing applications rely on certain fields and their data types.

Breaking Changes (v1 to v2)

1. Orders Table:

Field: `order\_date`

- Type Change: In v1, `order\_date` is of type `DATE`. In v2, it has changed to `TIMESTAMP\_NTZ`.
 - Reasoning: This is a breaking change because the `TIMESTAMP\_NTZ` type includes time-related information whereas `DATE` does not. Therefore, any logic specifically relying on the `DATE` type could break.
 - Assumption: Applications might rely on `order\_date` purely for date information, but not on time, which is not stored in `DATE`.

Field: `region\_code`

- New Field: Added in v2.
 - Reasoning: This is a breaking change because it is required to store an entry in the `orders` table. Any inserts or updates to orders without a `region\_code` value will fail unless explicitly handled.
 - Assumption: The field did not exist in v1, so applications will require changes to handle this new field.

2. Orders Table: No Columns Removed

3. New Table: customers

Fields: `customer\_id`, `region`, `segment`

- Reasoning: This table is entirely new, providing additional customer details. It does not change existing functionality but introduces new data sources.
- Assumption: Existing applications won't have references to, or dependencies on this table.

Non-Breaking Additions (v1 to v2)

1. Orders Table:

Field: `region\_code`

- Reasoning: While it is required, it does not inherently break existing functionality. While not previously existing, it can be populated or handled separately.

2. Order-Items Table:

Field: `order\_item\_id`

- Reasoning: While new, it augments functionality and does not break existing operations. It's purely additive.

Migration Risks and Considerations

1. Date Type Change Risk:

- Reasoning: The change from `DATE` to `TIMESTAMP\_NTZ` for `order\_date` needs thorough verification because date operations can vary substantially between types.

- Verification Step: Test date operations, filters, and arithmetic to ensure they function as intended.

2. New Required Field:

- Reasoning: `region\_code` in the `orders` table requires handling. Applications must accommodate this new field.

- Verification Step: Validate insert/update operations to check for proper handling.

3. New Table Introduction Risk:

- Reasoning: The `customers` table can introduce new joins. Existing queries or logic might not handle these new joins gracefully.

- Verification Step: Ensure that existing queries and any new joins on `customers` table work correctly and no existing functionality is hindered.

Assumptions Recap

- Backward Compatibility: We assume that most existing operations will continue to work but need some verification and updates to handle the new schema changes

Engineering counterpart
7:15 AM - Responded in 0s

What would you verify before shipping this to production?

Agent response

7:16 AM - Responded in 4s

Before shipping this schema change to production, it's crucial to verify several components to ensure both data integrity and application compatibility. Here's a detailed breakdown of what should be verified:

1. Schema Consistency and Data Type Compatibility

Orders Table:

- `order_date`: Specifically test date-based operations, filters, and calculations to ensure they work seamlessly with ``TIMESTAMP_NTZ``. Ensure that all date-related queries account for the timestamp component.
- Test Case: Add tests that use date functions to check if results remain consistent.

Unit Tests:

- Tests for Data Types: Write unit tests focused on data types being manipulated in different operations. Specifically, ensure date operations are compatible with ``TIMESTAMP_NTZ``.

Application Logic:

- Insert and Update Operations: Verify that ``region_code`` is handled correctly in all new or updated ``orders`` records. Ensure proper default values are managed if no value exists initially.
- Test Case: Create unit tests that attempt to insert and update ``orders`` records with and without ``region_code``.

2. Database Integrity

Data Migration Scripts:

- Validation: Ensure that any scripts converting old ``DATE`` to ``TIMESTAMP_NTZ`` in ``order_date`` are correct. Specifically, check if time information is maintained or set to zero.
- Test Case: Manually test migration script on a small set of data before wide deployment.

Addition of New Table:

- `customers` Table: Ensure the new table ``customers`` doesn't interfere with existing data and that the referential integrity in any future joins is intact.
- Test Case: Insert test data in ``customers`` and perform joins with ``orders``, verify correctness.

3. Application Level Integration

APIs and Queries:

- Data Retrieval: Verify that all APIs and query endpoints that use ``order_date`` or ``orders`` table now correctly handle the new schema.
- Test Case: Test end-to-end data retrieval operations, looking for any discrepancies or failures.

Application Logic:

- Handling of New Fields: Ensure any business logic that depends on ``region_code``, ``order_item_id``, or ``customers`` table can manage these new fields without breaking.
- Test Case: Write integration tests that mimic real business scenarios involving these fields.

4. Load and Performance Testing

- Throughput: Test if the schema changes affect throughput of the system under load. Ensure the transition doesn't cause any performance degradation.
- Test Case: Simulate production-like load and monitor system performance.

5. Backup and Recovery

- Data Backups: Ensure that backup processes account for new schema changes and test backups can be correctly restored.

- Test Case: Perform backups pre-deployment, restore to a testing environment, and verify data integrity.

6. User Acceptance Testing (UAT)

- End User: Conduct UAT sessions to receive feedback from users interacting with the system post schema changes.
- Test Case: Create test cases based on common user actions to ensure none breaks and the new fields are appropriately used.

7. Monitoring and Alerts

- Post-Deployment Monitoring: Set up monitoring and alerting to track any anomalies or performance issues immediately after deployment.
- Test Case: Monitor critical metrics post-deployment to quickly address any issues.

In summary, conducting thorough unit and integration tests, load tests, data integrity checks, and user acceptance testing while ensuring backup and recovery processes account for the schema changes will give confidence in shipping this update to production.

Engineering counterpart
7:16 AM - Responded in 0s

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

Agent response

7:16 AM - Responded in 3s

When deploying schema changes, it's essential to identify the top risks and prepare a robust rollback plan. Here, I'll outline the top risks for the schema migration from v1 to v2, followed by a rollback strategy.

Top Risks:

1. Data Integrity Risks:

- Migration Script Failures: Errors in the transformation scripts converting `DATE` to `TIMESTAMP\_NTZ`.
- Incomplete Data Migration: Ensuring all related data is correctly migrated to the new schema without loss or duplication.

2. Operational Disruption:

- Application Downtime: Potential disruption to applications and services due to the schema change.
- Integration Issues: New table `customers` and field `region\_code` requiring application logic changes might lead to integration issues.

3. Performance Impacts:

- Performance Degradation: Schema changes, especially adding new columns or tables and altering data types, may affect system performance.

4. Security Concerns:

- Data Exposure: If not properly managed, schema changes can inadvertently expose more data than before, particularly if new fields are not adequately secured.

5. Compliance Risks:

- Regulatory Compliance: Any changes in data storage or types might affect compliance with regional regulations depending on the nature and sensitivity of the data.

Rollback Plan:

A rollback plan should be meticulously detailed to quickly revert to the previous schema (v1) if the change does not work out as planned.

Preparatory Steps Before Roll Back:

1. Backup Current Schema:

- Take a backup of the current live schema (v2) before starting the rollback process. This ensures all changes are preserved for thorough investigation.

2. Communication:

- Inform all stakeholders about the potential need for a rollback and the planned procedure.
- Set up a monitoring system to track performance, data integrity, and user interactions post-deployment.

Steps for Rollback:

1. Disable New Schema Features:

- Temporarily disable integration with the new `customers` table and the `region\_code` field on `orders`.

2. Revert Data Types and Fields:

- order_date:
 - Reversion involves converting all `TIMESTAMP\_NTZ` to `DATE` where necessary. This requires carefully crafted scripts that handle this conversion without data loss.
- region_code:
 - Simply remove the `region\_code` field from the `orders` table.

3. Database Rollback:

- If feasible, revert the database to the previous schema version by restoring from the latest backup taken before the schema change (v1).
- Use a database versioning tool if available to revert schema changes.

4. Update Application Code:

- Modify any application logic that was written to handle the new schema in v2 back to the old v1 logic.

5. Test Rollback:

- Verify that all reverted changes have been correctly applied and that systems operate as expected under the new schema.

6. Monitor Post-Rollback:

- Closely monitor performance, data integrity, and application behavior for any anomalies after the rollback.

7. Post-mortem Analysis:

- After stabilizing the system, conduct a thorough analysis to understand what went wrong in the first place, focusing on migration script failures, performance issues, etc.

This plan is designed to swiftly address unexpected issues, ensuring minimal disruption to business operations and maintaining data integrity. It's also crucial to communicate these plans well in advance with all stakeholders involved.

Engineering counterpart

7:16 AM - Responded in 0s

Thanks - useful breakdown of breaking vs additive changes. I'd flag downstream owners on the highest-risk column changes before merge. Review complete - no further response needed.