



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: openai/gpt-4.1-mini

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

Duration: 52s

Actual run cost: \$0.004640

Projected execution cost: \$3.06/mo (30/biz day × 22 days)

Report date: July 14, 2026 - 3:19 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: 12108 chars - Latency: 52s - 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: 52s

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

Engineering counterpart

7:18 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

7:18 AM - Responded in 12s

1) Breaking Changes

- `orders.order\_date` type change:
 - From `DATE` in v1 to `TIMESTAMP\_NTZ` in v2.
 - This may affect queries, ETL jobs, or applications expecting a plain date without time.
- `order\_items` table column change:
 - Removed `order\_items` primary key assumption on `(order\_id, sku)`?
 - Added new column `order\_item\_id` (of type `NUMBER`) in v2, no longer a pure key on `(order\_id, sku)`.
 - This is a schema structural change that may impact joins or key constraints relying on previous composite keys.

2) Non-breaking Additions

- New column in `orders`:
 - `region\_code` (`VARCHAR(8)`) added in v2.
 - New optional metadata for geolocation or segmentation.
- New table `customers` added:
 - Contains `customer\_id`, `region`, and `segment`.
 - Enhances customer dimension but does not break existing tables.

3) Migration Risks and Recommended Verification Steps

- Data type conversion for `orders.order\_date`:
 - Risk: Downstream consumers expecting `DATE` may fail or behave incorrectly with `TIMESTAMP\_NTZ`.
 - Verify: Validate ETL pipelines, reports, and application code to handle timestamp values correctly without time zone.
- Handling `order\_items` new primary key:
 - Risk: Introducing `order\_item\_id` may change uniqueness assumptions and foreign key relationships; if `(order\_id, sku)` was previously a composite key, this may break referential integrity or data deduplication logic.
 - Verify: Confirm new primary/unique keys. Update all joins or FK constraints referencing `order\_items`. Test queries and data integrity.
- `orders.region\_code` nullability and usage:
 - Risk: If made required later, existing data loads might fail; also verify default handling if missing in historical data.
 - Verify: Check nullability, backfill missing data, and confirm queries handle optional column.
- New `customers` table integration:
 - Risk: Potential confusion or duplication if `customer\_id` references are inconsistent with `orders.customer\_id`.
 - Verify: Validate referential integrity between `orders.customer\_id` and `customers.customer\_id`. Confirm ETL pipelines populate and maintain `customers` accurately.

Summary

Change Area	Type	Impact	Recommendations	
----- ----- ----- -----				
`orders.order_date`	`DATE` -> `TIMESTAMP_NTZ`	Breaking (type change)	Validate downstream consumers for timestamp	
`order_items` columns	Added `order_item_id`	Breaking (PK structural)	Confirm PK/FK, test queries, update code	
`orders.region_code`	New column	Non-breaking	Verify nullability, backfill if needed	

| New table `customers` | Added | Non-breaking | Validate FK, populate and maintain data |

Engineering counterpart
7:18 AM - Responded in 0s

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

Agent response

7:18 AM - Responded in 18s

Certainly! Here is my detailed reasoning and assumptions when reviewing the schema differences between v1 (2024-Q1) and v2 (2024-Q2):

Step 1: Understand the baseline schema (v1)

- Two tables: `orders` and `order\_items`.
- `orders` columns:
 - `order\_id` (NUMBER): Presumably the primary key.
 - `customer\_id` (NUMBER): Foreign key to customers (though no explicit `customers` table exists here).
 - `order\_date` (DATE): stores order date without time.
 - `total\_usd` (NUMBER(12,2)): total order amount.
 - `status` (VARCHAR(32)): order status.
- `order\_items` columns:
 - `order\_id` (NUMBER): foreign key to orders.
 - `sku` (VARCHAR(64)): product identifier.
 - `qty` (NUMBER): quantity ordered.
 - `unit\_price` (NUMBER(10,2)): price per unit.
- No explicit primary keys defined in v1, but common practice suggests `orders.order\_id` is PK, and `order\_items` might use `(order\_id, sku)` as a composite key.

Step 2: Review differences found in v2 (2024-Q2)

- orders table:
 - `order\_date` type changed from `DATE` to `TIMESTAMP\_NTZ`.
 - New column `region\_code` (VARCHAR(8)) added.
- order_items table:
 - New column `order\_item\_id` (NUMBER) added.
 - Existing columns remain the same.
- New customers table:
 - `customer\_id` (NUMBER), `region` (VARCHAR(32)), `segment` (VARCHAR(16)).

Step 3: Identify breaking changes

- Type change of orders.order_date:
 - Changing from `DATE` to `TIMESTAMP\_NTZ` is breaking because:
 - `DATE` generally stores only year-month-day (no hours/minutes).
 - `TIMESTAMP\_NTZ` stores date and time without timezone.
 - Downstream code, ETL, or reports expecting only a date may break or produce incorrect results.
 - Assumption: Downstream applications expect the original DATE type.
- order_items primary key assumptions:
 - v1 likely uses `(order\_id, sku)` as a compound key (common pattern for order line items).
 - v2 adds `order\_item\_id` which strongly suggests a surrogate primary key replacing or complementing the composite key.
 - This is a breaking structural change because:
 - Any code assuming `(order\_id, sku)` uniquely identifies an item will fail.
 - Foreign keys referencing `order\_items` by `(order\_id, sku)` would need adjustment.
 - Assumption: The addition of `order\_item\_id` is intended as a unique ID replacing composite keys.
 - No tables or columns are removed, so no direct drop-related breakage.

Step 4: Identify non-breaking additions

- Adding new optional columns or tables without removing or changing existing ones is typically non-breaking:
 - `orders.region\_code`: new optional column, no impact if handled as nullable.
 - New `customers` table: additive dimension table, adds data model richness but does not invalidate existing tables or columns.

Step 5: Assess migration risks and verification steps

- orders.order_date:
 - Risk: apps/ETL breaking due to new timestamp behavior.
 - Verify: test downstream consumers, validate data loads and reports.
- order_items order_item_id:
 - Risk: data integrity changes, join queries failing.
 - Verify: review keys and constraints, update foreign keys and queries, test data consistency.
- orders.region_code:
 - Risk: null or missing data if not

Engineering counterpart
7:18 AM - Responded in 0s

What would you verify before shipping this to production?

Agent response

7:18 AM - Responded in 10s

Before deploying the 2024-Q2 schema changes to production, I would verify the following critical areas to mitigate risks and ensure smooth migration:

1) Data Type and Application Compatibility

- orders.order_date (DATE -> TIMESTAMP_NTZ)

- Validate that all ETL pipelines ingesting `order\_date` accept and correctly process timestamps without time zone.
- Confirm downstream applications, reports, and queries handle timestamps properly, especially any logic relying on date-only (e.g., date truncation, comparisons).
- Run regression tests comparing v1 and v2 outputs for typical date-related queries.
- Check for any timezone assumptions or conversions made outside the database and adjust accordingly.

2) Primary Key and Foreign Key Integrity

- order_items new `order\_item\_id`

- Verify whether `order\_item\_id` is the new primary key or a surrogate key supplementing `(order\_id, sku)`.
- Confirm all foreign keys referencing `order\_items` are updated to use `order\_item\_id` if applicable.
- Ensure uniqueness constraints / indexes exist to avoid duplicates.
- Test all queries, joins, and ETL logic that use `order\_items`-especially those that previously relied on `(order\_id, sku)` as unique identifiers-to confirm they work correctly with the new key.
- Validate data migration or backfill for populating `order\_item\_id` for existing rows.

3) New Columns and Tables Functionality

- orders.region_code

- Check nullability and default handling (is it optional or required?).
- Backfill historical data if needed to prevent null-related issues.
- Validate downstream usage-reports, filters, aggregations involving `region\_code`.

- customers table

- Verify consistent and accurate population of `customers` data.
- Confirm referential integrity between `orders.customer\_id` and `customers.customer\_id`.
- Test that joins across these tables return expected results.
- Validate new downstream consumers using `customers` table.

4) Data Consistency and Integrity Tests

- Perform data validation comparing v1 vs. v2 for critical metrics:

- Total counts of orders and order items.
 - Amounts aggregated per order.
 - Customer segmentation correctness.
- Run end-to-end pipelines with sample production data or a staging environment dataset.
 - Check for any data loss or corruption after migration.

5) Performance and Indexing

- Ensure appropriate indexes exist for new columns and keys (`order\_item\_id`, `region\_code`).
- Monitor query performance pre- and post-migration to identify regressions due to schema changes.

6) Communication and Rollback Plans

- Communicate schema changes and impacts to all downstream consumers and stakeholders.
- Prepare a rollback plan to revert to the previous schema if critical issues are detected after deployment.

By thoroughly verifying these areas, you minimize risk and ensure the schema migration is safe and reliable before shipping to production.

Engineering counterpart
7:18 AM - Responded in 0s

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

Agent response

7:19 AM - Responded in 12s

Top Risks

1. `orders.order\_date` type change impacts downstream systems
 - Downstream ETL, reports, or applications expecting a `DATE` type may malfunction with `TIMESTAMP\_NTZ` (including timezone mishandling, querying errors, or incorrect aggregations).
2. `order\_items` primary key and uniqueness assumptions broken
 - Addition of `order\_item\_id` may invalidate existing logic relying on `(order\_id, sku)` as a composite key. This can cause join failures, duplicate data issues, or broken foreign key references.
3. Data integrity and referential integrity issues
 - Missing or inconsistent `order\_item\_id` values for existing data.
 - Mismatch or missing `customers` data leading to orphaned `orders.customer\_id`.
 - Null or incomplete `region\_code` in `orders` affecting reports or filters.
4. Performance degradation
 - New keys or columns not indexed properly leading to slower queries.
 - Additional join complexity between `orders` and new `customers` table.

Rollback Plan

1. Maintain a backup snapshot of the current production schema and data (v1) before deployment.
 - Snapshot or export metadata for all impacted tables.
 - Backup data for `orders`, `order\_items`, and any new `customers` data loaded.
2. Deploy changes in a controlled staged environment first
 - Verify completeness and correctness before production rollout.
3. If issues arise post-deployment in production:
 - Step 1: Temporarily disable or redirect workflows relying on the new schema changes (e.g., stop ETL jobs, pause report generation).
 - Step 2: Restore the schema to v1 definition:
 - Revert `orders.order\_date` from `TIMESTAMP\_NTZ` back to `DATE`.
 - Remove `order\_item\_id` column from `order\_items` and revert to prior primary key structure.
 - Drop or ignore `customers` table if not yet fully integrated.
 - Remove `region\_code` column from `orders`.
 - Step 3: Reload data from backups if necessary, ensuring data consistency matches the v1 schema.
 - Step 4: Validate that all downstream pipelines and applications operate correctly on the restored schema and data.
 - Step 5: Communicate the rollback and plan next steps (fix issues, retest, reschedule migration).
4. Post-rollback:
 - Conduct root cause analysis to identify the failure points.
 - Adjust migration scripts, code, or data modeling based on findings.
 - Plan a carefully phased redeployment with additional safeguards (e.g., feature flags, backward compatibility layers).

Having this rollback plan and clearly understanding the top risks will help minimize downtime and business impact if the migration encounters issues in production.

Engineering counterpart

7:19 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.