Schema Drift Detection Strategy

1. Implementing Schema Drift Detection in a Live Data Pipeline

To prevent silent schema changes from impacting downstream systems, schema drift detection should occur as an integrated step within the ingestion workflow. Immediately after extracting data from the source system, the pipeline should infer the structure of the incoming dataset and validate it against a stored reference schema (schema registry).

Any deviations should be classified based on impact:

- **Non-breaking changes:**
  Examples include additional nullable columns. These should allow ingestion to continue but must be logged and flagged for review.
- **Breaking changes:**
  Examples include removed fields, renamed fields, or incompatible data type changes. These should trigger alerts and, depending on system policy, pause ingestion or reroute the batch to a quarantine zone.

Embedding this detection step ensures that schema changes are captured within minutes rather than remaining invisible until dashboards or pipelines fail downstream.

---

2. Tracking Field-Level Lineage and Schema Version History

Maintaining schema history is essential for both traceability and impact analysis. A schema registry should be used to store metadata for each dataset, including:

- Schema version
- Field names and data types
- Nullable constraints
- Change timestamp
- Author or source of change
- Reason for change (optional)

Each time a schema drift is validated and approved, a new version should be recorded.

For lineage, field-level mapping should track how each source field propagates across staged tables, curated data layers, and ultimately business dashboards. This allows teams to answer questions such as:

“If the partner removes industry_group, which models and reports break?”

Having lineage tightly coupled with schema history significantly reduces the time to analyze and resolve drift incidents.

---

3. Logging, Alerting, and Visualization

When a schema drift is detected, the system should generate a structured event log containing:

- Source dataset name
- Expected schema vs detected schema
- Type of drift (new column, removed field, type mismatch, etc.)
- Severity level
- Pipeline run ID
- Timestamp

Alerts should follow a tiered approach:

- **High severity:** Type mismatches or removed/renamed fields should trigger immediate Slack, MS Teams, or PagerDuty notifications.
- **Medium severity:** Added fields with no downstream impact should be recorded but do not require escalation.

A monitoring dashboard should visualize:

- Trend of schema changes over time
- Active and unresolved schema violations
- Impacted downstream assets via lineage overlay

This provides engineers with actionable context to evaluate, approve, or remediate drift quickly.

---

4. Proof of Concept Approach Summary

A proof-of-concept implementation compares two versions of a dataset by inferring field names and data types. It identifies:

- Newly added fields
- Fields that no longer exist
- Type mismatches between versions

The output is a concise structured drift report and can optionally be stored locally or in a metadata repository for traceability.

This approach validates the feasibility of automating schema comparison and provides the foundation for integrating schema drift governance into a production data pipeline.
