**Databricks Processing & Standardization Issues with Mitigations**

**Databricks Processing & Standardization Issues Comparison Table with Mitigations**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | Processing & Standardization Issue | One-Liner Description | Where It Typically Arises | Processing Areas Most Affected | Mitigation Strategies |
| 1 | Inconsistent Ingestion Formats | Incoming data arrives in varying formats and encodings, complicating processing. | Raw ingestion pipelines | Bronze Layer, Auto Loader | Define standard schemas and enforce file format validation in ingestion workflows. |
| 2 | Schema Drift | Source systems evolve without notice, introducing unexpected fields or dropping columns. | Streaming and batch ingestion | Bronze and Silver Layers | Use schema evolution with Auto Loader; quarantine incompatible records for review. |
| 3 | Data Type Inconsistencies | Columns are stored with inconsistent types across datasets, causing transformation failures. | Data consolidation, joins | Silver Layer, Enrichment Jobs | Standardize data types during ingestion; validate with schema enforcement and casting logic. |
| 4 | Lack of Standard Naming | Columns and tables have inconsistent naming conventions, reducing clarity and maintainability. | Schema design, data modeling | Bronze, Silver, Gold Layers | Establish and enforce naming conventions for all processing pipelines and outputs. |
| 5 | Duplicate Records | Data arrives more than once or joins create duplicates, inflating downstream metrics. | Multi-source ingestion, deduplication workflows | Silver Layer, Aggregations | Use deduplication strategies with primary keys, window functions, and incremental processing. |
| 6 | Inefficient File Sizes | Writes produce many small files, reducing query performance and increasing costs. | Data ingestion, transformations | Bronze and Silver Layers | Use repartition() and coalesce() to optimize file sizes; compact files regularly. |
| 7 | Missing Data Validation | Invalid or incomplete data is processed without checks, propagating errors downstream. | Ingestion and transformation jobs | Silver and Gold Layers | Implement validation rules and error handling logic in ETL pipelines. |
| 8 | Ineffective Partitioning | Data isn’t partitioned optimally, hurting read performance and increasing scan costs. | Table creation, data writes | Silver and Gold Layers | Partition by relevant columns (e.g., date) and validate partition strategies with query plans. |
| 9 | Lack of Reusable Transformation Logic | ETL pipelines duplicate logic across jobs, increasing maintenance burden and risk of inconsistency. | Pipeline development | Notebooks, Workflows | Create shared libraries or notebooks for reusable transformation functions and validation logic. |
| 10 | No Clear Data Lineage | Teams can’t trace how raw data was transformed into processed tables, limiting transparency. | Processing workflows | Silver and Gold Layers | Use Delta transaction history and Unity Catalog lineage features to track transformations end to end. |

**Quick Reference**

* **Bronze Layer:** Raw ingestion zone storing unprocessed data.
* **Silver Layer:** Cleaned, enriched, and standardized data.
* **Gold Layer:** Aggregated, business-ready datasets.
* **Schema Enforcement:** Strict checks ensuring data adheres to expected structure.
* **Deduplication:** Logic to remove duplicates based on keys or timestamps.

**Example Mitigation Actions and Configurations**

**Ingest Data with Auto Loader and Schema Evolution:**

python

CopyEdit

df = (

spark.readStream

.format("cloudFiles")

.option("cloudFiles.format", "json")

.option("**cloudFiles.schemaEvolutionMode**", "addNewColumns")

.load("s3://raw-data/incoming/")

)

**Enforce Data Types:**

python

CopyEdit

from pyspark.sql.types import StructType, StringType, IntegerType

schema = StructType() \

.add("id", StringType()) \

.add("amount", IntegerType())

df = spark.read.schema(schema).json("s3://raw-data/")

**Deduplicate Records:**

python

CopyEdit

from pyspark.sql.window import Window

import pyspark.sql.functions as F

window\_spec = Window.partitionBy("record\_id").orderBy(F.col("ingest\_timestamp").desc())

df = (

df.withColumn("row\_number", F.row\_number().over(window\_spec))

.filter("row\_number = 1")

.drop("row\_number")

)

**Optimize File Sizes:**

python

CopyEdit

(

df.repartition(10)

.write

.format("delta")

.mode("append")

.save("/mnt/silver/transactions/")

)

**Implement Data Validation:**

python

CopyEdit

if df.filter("amount IS NULL").count() > 0:

raise ValueError("Null amounts detected!")

**Document Lineage:**

* Use Delta log and Unity Catalog:

sql

CopyEdit

DESCRIBE HISTORY delta.`/mnt/silver/transactions/`;

**Partition Data Strategically:**

python

CopyEdit

(

df.write

.format("delta")

.mode("overwrite")

.partitionBy("event\_date")

.save("/mnt/gold/aggregates/")

)