**Detecting Partial Write in Databricks**

**Use Transaction Log (Delta Lake \_delta\_log)**

**Delta Lake maintains an atomic transaction log (\_delta\_log) for every table. You can inspect the log to check if:**

* **A write job was committed (add entries exist),**
* **But expected row counts are inconsistent or missing.**

**DESCRIBE HISTORY delta.`/path/to/delta\_table`**

**ORDER BY version DESC**

**operation → Should be WRITE, MERGE, etc.**

**OperationMetrics.numOutputRows → Zero or unusually low can indicate partial write.**

**2. Write Audit Log (Custom Monitoring Table)**

**Track expected vs. actual written rows using a monitoring table.**

**Example:**

**python**

**CopyEdit**

**expected\_rows = 100000**

**# Write operation**

**df.write.format("delta").mode("append").save("/mnt/delta/my\_table")**

**# Validate actual write**

**actual\_rows = spark.read.format("delta").load("/mnt/delta/my\_table").count()**

**if actual\_rows < expected\_rows:**

**raise Exception("Partial write detected!")**

**Or use a logging table:**

**python**

**CopyEdit**

**log\_df = spark.createDataFrame([{**

**"table\_name": "my\_table",**

**"run\_id": run\_id,**

**"expected\_rows": expected\_rows,**

**"actual\_rows": actual\_rows,**

**"write\_success": actual\_rows == expected\_rows,**

**"detected\_at": datetime.now(timezone.utc).isoformat()**

**}])**

**log\_df.write.mode("append").saveAsTable("monitoring.delta\_write\_audit")**

**3. Streaming with Checkpoints + Watermarking**

**In Structured Streaming, partial writes are less likely due to checkpointing. But if they happen, offsets will not move forward.**

**python**

**CopyEdit**

**df.writeStream**

**.format("delta")**

**.option("checkpointLocation", "/mnt/checkpoints/my\_stream")**

**.start("/mnt/delta/my\_table")**

**If a failure happens mid-microbatch:**

* **The write will not be committed to Delta.**
* **You can detect repeated processing of same offsets (stuck jobs).**

**4. Validation with Hash or Counts**

**Store and validate row-level checksums, record counts, or primary keys.**

**from pyspark.sql.functions import md5, concat\_ws**

**# Add hash column**

**df = df.withColumn("row\_hash", md5(concat\_ws("||", \*df.columns)))**

**# After write, re-read and verify all hashes**

**df\_written = spark.read.format("delta").load("/mnt/delta/my\_table")**

**invalid = df\_written.exceptAll(df)**

**if invalid.count() > 0:**

**print("Partial write or data corruption detected")**

**5. Leverage MERGE with Data Quality Checks**

**When using MERGE INTO, validate row counts:**

**sql**

**CopyEdit**

**MERGE INTO delta\_table AS target**

**USING updates AS source**

**ON target.id = source.id**

**WHEN MATCHED THEN UPDATE SET \***

**WHEN NOT MATCHED THEN INSERT \***

**-- Post check (if using SQL interface)**

**SELECT COUNT(\*) FROM delta\_table WHERE updated\_at = current\_run\_time**

**6. Delta Table Constraints (Not Strict, But Useful)**

**While not directly preventing partial writes, Delta constraints (e.g., NOT NULL) can fail and alert inconsistencies early.**

**ALTER TABLE my\_table ADD CONSTRAINT valid\_id CHECK (id IS NOT NULL);**

**7. Databricks Job and Alerting Integration**

**Combine the above with:**

* **Databricks Jobs API or Workflows**
* **Slack/Email alerts on validation failures**
* **Delta Live Tables (DLT) expectations for automatic error flagging**

**Summary Table**

|  |  |  |
| --- | --- | --- |
| **Method** | **Detects Partial Writes?** | **Use Case** |
| **Delta DESCRIBE HISTORY** | **✅** | **Audit-based detection** |
| **Write Audit Logging** | **✅** | **Custom ETL pipelines** |
| **Streaming Checkpoints** | **✅** | **Structured Streaming** |
| **Row-level Hash Check** | **✅** | **Critical data integrity** |
| **Row Count Validation** | **✅** | **Batch and streaming** |
| **Delta Constraints** | **⚠️ (limited)** | **Data quality enforcement** |
| **DLT Expectations** | **✅** | **Declarative pipeline QA** |

**Sample.csv file**

|  |
| --- |
| **id, name, age**  **1, Alice, 30**  **2, Bob, 25**  **3, Charlie, 35**  **4, David, 40**  **5, Eva, 28** |

**Step 1: Setup – Load the CSV and Write to Delta Table**

**python**

**CopyEdit**

**# Load sample CSV**

**df = spark.read.csv("/path/sample\_data.csv", header=True, inferSchema=True)**

**# Write to Delta table**

**df.write.format("delta").mode("overwrite").save("/mnt/delta/sample\_table")**

**1. Delta DESCRIBE HISTORY (Audit-based detection)**

**sql**

**CopyEdit**

**-- SQL version (in a notebook cell)**

**DESCRIBE HISTORY delta.`/mnt/delta/sample\_table`;**

**Look at:**

* **operation: should be WRITE**
* **operationMetrics.numOutputRows: should be 5**
* **userMetadata: optionally track job\_id**

**If you see fewer rows than expected, that indicates a partial write.**

**2. Write Audit Logging (Custom ETL pipelines)**

**Add audit logs for tracking write activity:**

**python**

**CopyEdit**

**from datetime import datetime**

**import uuid**

**run\_id = str(uuid.uuid4())**

**expected\_rows = 5**

**actual\_rows = spark.read.format("delta").load("/mnt/delta/sample\_table").count()**

**log\_df = spark.createDataFrame([{**

**"run\_id": run\_id,**

**"table\_name": "sample\_table",**

**"expected\_rows": expected\_rows,**

**"actual\_rows": actual\_rows,**

**"write\_success": actual\_rows == expected\_rows,**

**"logged\_at": datetime.now().isoformat()**

**}])**

**log\_df.write.mode("append").saveAsTable("monitoring.write\_audit\_log")**

**3. Streaming with Checkpoints (Structured Streaming)**

**Let’s simulate streaming ingestion using the same file (append mode):**

**python**

**CopyEdit**

**from pyspark.sql.functions import \***

**stream\_df = spark.readStream**

**.schema(df.schema)**

**.option("maxFilesPerTrigger", 1)**

**.csv("/path/stream\_input")**

**stream\_df.writeStream**

**.format("delta")**

**.option("checkpointLocation", "/mnt/checkpoints/sample\_stream") \**

**.outputMode("append") \**

**.start("/mnt/delta/streaming\_table")**

**Partial writes are avoided due to checkpointing, but if the stream reprocesses the same batch (look for repeated offsets), you may have a problem.**

**4. Row-level Hash Check (Critical data integrity)**

**python**

**CopyEdit**

**from pyspark.sql.functions import md5, concat\_ws**

**df = df.withColumn("row\_hash", md5(concat\_ws("||", \*df.columns)))**

**# Write with hash**

**df.write.format("delta").mode("overwrite").save("/mnt/delta/hashed\_table")**

**# Validate written data**

**df\_written = spark.read.format("delta").load("/mnt/delta/hashed\_table")**

**# Detect any row differences**

**discrepancy = df.exceptAll(df\_written)**

**if discrepancy.count() > 0:**

**print("Partial write or corruption detected")**

**5. Row Count Validation (Batch and streaming)**

**python**

**CopyEdit**

**expected\_count = 5**

**actual\_count = spark.read.format("delta").load("/mnt/delta/sample\_table").count()**

**if actual\_count != expected\_count:**

**raise Exception(f"Partial write detected! Expected {expected\_count}, got {actual\_count}")**

**6. Delta Constraints (Data quality enforcement)**

**Delta constraints help indirectly by catching missing/invalid values:**

**sql**

**CopyEdit**

**-- In SQL notebook cell**

**ALTER TABLE delta.`/mnt/delta/sample\_table`**

**ADD CONSTRAINT id\_not\_null CHECK (id IS NOT NULL);**

**Now if you write rows with null IDs, the write will fail and trigger alerts.**

**7. Delta Live Tables (DLT) Expectations (Declarative QA)**

**In a DLT pipeline (Python syntax), use expectations:**

**python**

**CopyEdit**

**@dlt.table**

**@dlt.expect("valid\_id", "id IS NOT NULL")**

**def cleaned\_data():**

**return spark.read.format("csv").option("header", True).load("/path/sample\_data.csv")**

**DLT will automatically log failed rows and job status, preventing bad writes.**