📌 ACID Compliance in Delta Lake  
  
Delta Lake is an open-source storage layer built on top of data lakes (e.g., Azure Data Lake Storage, S3, GCS) that brings ACID transactions, schema enforcement, time travel, and data reliability to big data pipelines.  
  
  
1. What is ACID in Delta Lake?  
  
ACID stands for:  
  
A → Atomicity → A transaction is all or nothing  
  
C → Consistency → Data is always in a valid state  
  
I → Isolation → Concurrent writes don’t conflict  
  
D → Durability → Once committed, data is never lost  
  
Delta Lake achieves ACID compliance using a transaction log called the Delta Log.  
  
📌The Secret Behind ACID: Delta Transaction Log  
  
Whenever you write, update, delete, or merge data into a Delta table, Delta Lake doesn’t overwrite files directly.  
Instead, it:  
  
1. Writes new Parquet files for updated data.  
  
2. Creates a JSON commit file inside the \_delta\_log folder.  
  
3. Uses optimistic concurrency control to handle multiple writers.  
  
4. Atomically switches pointers to the latest snapshot.  
  
📌 Delta Table Structure  
  
/mnt/delta/sales/  
 ├── part-0000.snappy.parquet  
 ├── part-0001.snappy.parquet  
 ├── \_delta\_log/  
 ├── 00000000000000000001.json  
 ├── 00000000000000000002.json  
 ├── 00000000000000000003.json  
  
Each JSON file represents a commit → keeps track of which files were added or removed.  
  
  
3. How Each ACID Property Works in Delta Lake  
  
A. Atomicity  
  
Either the entire operation succeeds or none of it is applied.  
  
Example:  
  
from delta.tables import DeltaTable  
  
df = spark.read.format("csv").load("/mnt/raw/sales.csv")  
  
df.write.format("delta").mode("append").save("/mnt/delta/sales")  
  
If the job fails midway, no partial data is committed.  
  
Commit happens only after successful JSON entry in \_delta\_log.  
  
B. Consistency  
  
Delta validates schema and data integrity before committing.  
  
Schema Enforcement Example:  
  
df.write.format("delta").mode("append").save("/mnt/delta/sales")  
  
If new data has a different schema, Delta throws an error unless you enable schema evolution:  
  
df.write.option("mergeSchema", "true").format("delta").mode("append").save("/mnt/delta/sales")  
  
Ensures consistent schema across all files.  
  
C. Isolation  
  
Delta uses Optimistic Concurrency Control (OCC) to handle concurrent writes.  
  
Multiple writers can read the same snapshot.  
  
When they try to write, Delta checks the latest commit version.  
  
If two transactions conflict, one fails → retry ensures correctness.  
  
Example Scenario:  
  
Job A writes 10K records → Commit 0000000001.json  
  
Job B tries to write at the same time → Delta compares snapshots.  
  
If Job B uses an outdated snapshot, Delta blocks it and asks for a retry.  
  
D. Durability  
  
Once data is committed, it’s never lost.  
  
All commits are stored in the \_delta\_log.  
  
You can time travel back to any historical state.  
  
  
Example:  
  
# Read previous version of table  
df = spark.read.format("delta").option("versionAsOf", 3).load("/mnt/delta/sales")  
[**df.show**](http://df.show/)()  
  
✅ This makes Delta Lake highly reliable for data recovery.

Activate to view larger image,

