**Problem 4 — Incremental Processing (CDC, Upserts, Deduplication)**

**Problem Statement**

You receive daily **incremental files** (CSV/Parquet) with customer transactions.

* The data may contain **duplicates** (same record repeated).
* The same record may arrive **in multiple batches** with different timestamps.
* Files may arrive late (out-of-order updates).
* Requirement:
  1. Deduplicate by **business key** (e.g., customer\_id + txn\_id).
  2. Only keep the **latest record per key** (updated\_at max).
  3. Merge into **Silver Delta table** (idempotent, atomic upsert).

**Step 1: Load Incremental Batch**

from pyspark.sql.functions import row\_number, col

from pyspark.sql.window import Window

incoming\_df = (spark.read.format("parquet")

.load("dbfs:/mnt/raw/incoming/\*.parquet"))

**Step 2: Deduplicate within Batch**

* Remove duplicates by (customer\_id, txn\_id) keeping latest updated\_at.

window\_spec = Window.partitionBy("customer\_id", "txn\_id").orderBy(col("updated\_at").desc())

deduped\_df = (incoming\_df

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

.filter(col("rn") == 1)

.drop("rn"))

**Step 3: Merge into Silver (Idempotent Upsert)**

from delta.tables import DeltaTable

target = DeltaTable.forName(spark, "silver.customer\_txn")

(target.alias("t")

.merge(deduped\_df.alias("s"),

"t.customer\_id = s.customer\_id AND t.txn\_id = s.txn\_id")

.whenMatchedUpdate(condition="s.updated\_at > t.updated\_at",

set={"amount": "s.amount",

"status": "s.status",

"updated\_at": "s.updated\_at",

"\_last\_updated": "current\_timestamp()"})

.whenNotMatchedInsert(values={

"customer\_id": "s.customer\_id",

"txn\_id": "s.txn\_id",

"amount": "s.amount",

"status": "s.status",

"updated\_at": "s.updated\_at",

"\_last\_updated": "current\_timestamp()"})

.execute())

* This ensures:
  + Newer updates overwrite old ones.
  + Older updates (late-arriving) are ignored.
  + Insert-only rows are added.
  + Safe reprocessing (idempotent).

**Step 4: Late Data Handling**

* Current logic already ignores late events (s.updated\_at > t.updated\_at).
* If business requires storing history, use **Type 2 SCD** pattern → insert a new version instead of overwrite.

**Expected Thought Process in Interview**

* Always **deduplicate first** before merge (avoid multiple updates per key in one run).
* Use **Delta Merge** for atomic idempotency.
* Ensure **watermarking/updated\_at logic** to ignore late updates or handle them per business rules.
* Add **audit/logging** to track each batch processed.

**Follow-up Q&A (Interviewer Style)**

**Q1. How do you guarantee idempotency?**  
👉 By using Delta MERGE with conditional updates. Re-running the same file doesn’t create duplicates because business key ensures one version of each row.

**Q2. What if the same record appears across multiple files?**  
👉 Deduplicate using row\_number() or dropDuplicates with (key, updated\_at).

**Q3. How do you handle late data beyond a cutoff window?**  
👉 Use **watermarking** in streaming jobs:

df.withWatermark("updated\_at", "7 days") # only keep 7 days late data

**Q4. What if schema evolves (new columns in file)?**  
👉 Use mergeSchema=true in write, but better is explicit schema validation and controlled evolution process.

**Q5. How do you handle millions of records daily?**  
👉 Partition target table (e.g., by date), compact files (OPTIMIZE), enable **ZORDER** on join keys.

✅ With this, you’ve covered incremental loads, deduplication, idempotency, late data, and schema evolution — exactly what interviewers want to test in **Silver layer processing**.