**🔹 Follow-up Questions & Expected Deep Answers for CDC MERGE**

**Q1. How do you ensure idempotency in your CDC MERGE pipeline?**

**Expected Answer:**

* The MERGE is naturally idempotent: if the same CDC rows arrive again, conditions (updated\_at > \_last\_updated) ensure no duplicate/older updates overwrite data.
* For audit, I generate a **deterministic audit key** (hash of customer\_id|op|updated\_at|source\_file).
* Before inserting into audit table, I check (left anti join) to avoid duplicates.
* This way, the job can be retried safely without corrupting data or duplicating logs.

**Q2. How do you handle late-arriving events (old updated\_at)?**

**Expected Answer:**

* I compare incoming.updated\_at against target.\_last\_updated.
* If incoming is **older**, the row is ignored (MERGE condition prevents update).
* This preserves the latest known state.
* For lineage, I still capture the event in the **audit table** with a flag like is\_applied=false if required for debugging.
* Optionally, business rules can override (e.g., regulatory backdated updates), but by default: *keep newest wins*.

**Q3. Schema evolution — what if the CDC file adds a new column?**

**Expected Answer:**

* By default, I don’t allow silent schema drift. Instead, I:
  1. Validate input schema vs expected schema.
  2. If additive columns are allowed, I use **mergeSchema** on write, or explicitly add null columns to maintain compatibility.
  3. For breaking changes (renames, type changes), I fail early and trigger schema governance workflow.
* Explicit mapping ensures correctness. Auto mergeSchema is convenient but risky for production.

**Q4. Partial failure scenario — MERGE succeeded but audit write failed. How do you fix consistency?**

**Expected Answer:**

* Delta doesn’t support multi-table atomic writes, so I avoid writing audit first.
* Preferred pattern: **Enable Change Data Feed (CDF)** on prod.customers. After MERGE, generate audit from CDF — guaranteed consistent and replayable.
* If CDF not available:
  + I make audit writes idempotent with audit keys.
  + If audit fails, re-run audit step independently (since merge is committed).
* For high criticality: use **transaction markers** or **orchestration retries** to reconcile.

**Q5. What if you have to scale this for millions of CDC records daily (high volume, many small files)?**

**Expected Answer:**

* Use **Auto Loader** (cloudFiles) to ingest and dedupe into a **staging Delta table** first.
* Compact small files before merge (OPTIMIZE, ZORDER, or coalesce at batch level).
* Partition target table (e.g., by country, date) and **merge per partition** to avoid scanning full table.
* Use spark.sql.shuffle.partitions tuning, autoscaling clusters, and limit merges to keys present in batch.
* For downstream consumers, rely on **CDF** instead of replaying big merges.
* If SLA is tight, switch to **Structured Streaming with foreachBatch** for continuous upserts.

**Q6. How do you optimize the MERGE for performance?**

**Expected Answer:**

* Restrict MERGE to **only affected keys/partitions** (using WHERE clause or partition pruning).
* Compact CDC batch before merge to reduce shuffle overhead.
* Periodically run OPTIMIZE on target with **ZORDER(customer\_id)** to cluster data for faster lookups.
* Tune shuffle partitions (spark.sql.shuffle.partitions).
* Consider **salting/join hints** if skew exists on customer\_id.
* Monitor Spark UI for shuffle bottlenecks and long tasks.

**Q7. How would you design the audit table to support compliance queries?**

**Expected Answer:**

* Audit table has fields:  
  customer\_id, op, updated\_at, processed\_at, actor, source\_file, audit\_key, is\_applied.
* **Partition by processed\_date** → makes queries efficient.
* Store audit\_key for idempotency and deduplication.
* Store both event time (updated\_at) and processing time (processed\_at) for lineage.
* Optionally, include cluster/job run ID for traceability.
* Enables compliance to ask: *“Show me all updates for customer 101 between Aug 25 and Sep 1”*.

**Q8. How do you ensure concurrency safety if multiple jobs/processes write to the same table?**

**Expected Answer:**

* Delta uses **optimistic concurrency control** with transaction logs.
* If two jobs try to MERGE simultaneously and conflict, one will fail.
* I serialize heavy merges (e.g., one writer per table) using orchestrator or stream design.
* For streaming, I use a **single writer structured streaming job** with checkpointing for exactly-once guarantees.
* For batch, I rely on Delta’s conflict detection and job retries.