How would you perform a zero-downtime schema migration on RDS ?

STAR

Situation
We had an **e-commerce app running on MySQL RDS**, and business wanted to add a new column order_status to support advanced tracking — things like “order packed”, “shipped”, “out for delivery”.
The problem was, the **orders table had millions of records**, and a simple ALTER TABLE would **lock the table** for several minutes — which meant customers couldn’t place or update orders during that time.
That was **unacceptable during live traffic hours.**

Task
My job was to **add this new column and index** to improve query speed — **without any downtime** or impact on customers.
So, I had to plan a **zero-downtime schema migration**.

Action
Here’s how I approached it step by step:

1. **Plan and test safely first:**
   I took an **RDS snapshot** and restored it in a **staging environment**.
   I ran the schema migration there to measure how long it might take and whether it caused any locks.
2. **Used an online schema migration tool:**
   Instead of a direct ALTER TABLE, I used **gh-ost** — it’s an open-source tool by GitHub that performs schema changes **in the background** without blocking writes.
   It creates a shadow table, copies data slowly, applies live changes, and then swaps tables almost instantly.
3. **Implemented backward compatibility:**
   I ensured that the **new app code** could handle both old and new schemas.
   So, for a while, the app was doing **dual writes** — writing both to old and new columns.
   This meant we could safely test the new setup without breaking anything.

4. **Did a batch backfill:**
   Since the new column was empty for existing rows, I ran a **background job** that filled old records in **small batches** during off-peak hours, while monitoring **CPU, IOPS, and replica lag** via CloudWatch.
5. **Monitored continuously:**
   I kept an eye on metrics like **replication lag** and **query performance**, to ensure there was no load spike or replication issue.
6. **Switched gradually:**
   Once all data was in sync and verified, we flipped the **feature flag** in the app so it started reading from the new column.
   After a week of stability, we safely **dropped the old field**.

Result

✅ The migration was completed with **zero downtime** — customers didn't notice anything.
✅ The new index improved query performance by **~40%**.
✅ Our approach became a **standard playbook** for all future schema changes in the company.

Interview

"**backward/forward compatibility**, test in staging with production-like data, use **online schema tools** and replicas, implement **dual-write & feature flags** for a smooth transition, batch backfills, monitor aggressively, and have a clear rollback (snapshot/promote) plan. For RDS, prefer gh-ost/replicas because of privilege limits and always validate on the specific engine/version."

What does **"zero downtime"** mean during a schema migration?

A. The migration happens instantly
B. Users can continue using the application without noticing any outage
C. The database does not take backups during migration
D. Writes are paused but reads continue

✅ **Correct Answer: B.**

💡 **Explanation: Zero-downtime** migration means the app continues to function normally — no interruptions for end users during schema or data changes.

Which of the following statements is **TRUE** about safe schema changes?

A. Always drop old columns first before adding new ones
B. Avoid making backward-incompatible changes until all app instances are updated
C. Always perform ALTER TABLE directly on production
D. Add a NOT NULL column with default immediately

✅ **Correct Answer: B.**

💡 **Explanation:** Safe schema changes are **backward-compatible first** — add before removing, ensuring both old and new app versions work together.

On MySQL RDS, which tool can perform **online schema migrations** without locking tables?

A. mysqldump
B. gh-ost or pt-online-schema-change
C. AWS DMS
D. CloudFormation
✅ **Correct Answer: B.**
💡 **Explanation: gh-ost** and **pt-online-schema-change** are online migration tools that alter tables without full table locks — ideal for RDS.

What is the **safest first step** before running any schema migration on RDS production?
A. Disable Multi-AZ
B. Take a recent snapshot or backup
C. Drop unused tables to free space
D. Disable automatic backups
✅ **Correct Answer: B.**
💡 **Explanation:** A **snapshot** provides a rollback point if something goes wrong during schema modification.

During a zero-downtime migration, **dual-write logic** means:
A. Writing the same data to two DBs simultaneously for failover
B. Writing updates to both old and new schema columns temporarily
C. Logging DB writes twice for redundancy
D. Using replicas for read/write separation
✅ **Correct Answer: B.**
💡 **Explanation: Dual-write** ensures consistency between old and new schema fields during transition — critical for non-blocking migrations.
