Key Questions to Answer

Q1: Why is broadcast join faster for small tables compared to shuffle join?

Broadcast joins are faster when one of the tables is small because spark copies or broadcasts the small table to all executor nodes so that each task can perform the join locally.This avoids expensive shuffling of large datasets across the network, which happens in shuffle joins.It's especially efficient for star schemas or small lookup tables. So,in short, Broadcast join eliminates network shuffling, reducing time and improving performance.

Q2: How does Adaptive Query Execution (AQE) optimize Spark jobs at runtime?

AQE dynamically adjusts query plans while the job is running, based on actual data statistics. It optimizes spark jobs by changing join strategies dynamically on runtime like broadcast join if table is small, handling skewed joins by splitting large partitions,coalescing shuffle partitions to avoid too many small tasks improving task parallelism.So,AQE adapts the query plan to real runtime conditions, improving performance and resource usage.

Q3: Why do we use partitioning and Z-ORDER in Delta tables?

Partitioning and Z-ORDER help optimize performance and reduce I/O.Partitioning organizes data by specific columns to enable partition pruning so that spark reads only the necessary partitions.Z-ORDER sorts data within files across multiple columns improving data skipping and read efficiency, especially for selective queries.So basically, partitioning reduces what files spark reads and z-ORDER reduces how much of each file it reads.

Q4: What problem does salting solve in Spark joins?

Salting addresses data skew, where certain keys dominate and create uneven data distribution.During a join, skewed keys can cause some tasks to handle huge data, while others do little which results in performance bottlenecks.Salting is technique to add a random "salt" value to keys to spread skewed data across partitions, balancing the workload.It evenly distributes skewed keys to avoid slow, unbalanced join.

Q5: How does GitHub Actions help in enforcing CI/CD for data pipelines?

GitHub Actions helps with CI/CD in data pipelines by automating all the tasks that we'd usually have to do manually like running tests, checking code quality, and deploying jobs. For example, when someone pushes a change to the pipeline code, GitHub Actions can automatically run unit tests to make sure nothing breaks, validate things like schema changes or data quality rules, and even deploy the updated pipeline to your production environment. It just keeps things more consistent and reliable because we're not relying on people to remember every step, it's all built into the workflow. Plus, it catches mistakes early, so we're not finding out about broken pipelines after they've already messed with your data.