■ Spark Interview Playbook (Detailed Version)

Context: Banking & Retail Data Pipelines with DBT & Databricks

■ How Spark Fits into the Banking & Retail Data Pipeline

Data Sources (Core Banking, Retail POS, CRM, Cards) → Bronze (Raw Landing via Ingestion) → Silver (Cleansed & Transformed using Spark + DBT) → Gold (Aggregated & De-identified Reporting) → Business Dashboards & Analytics.

1. Spark Core & Architecture

Q: How does Spark fit into your pipeline?

A: Spark executes the transformations defined in DBT or Databricks notebooks. It processes large volumes of transactional and customer data in a distributed manner, leveraging cluster computing.

Q: How does Spark handle distributed processing?

A: Spark splits data into partitions and processes them in parallel across executors, enabling high-performance distributed computation.

Q: What Spark components do you use?

A: Spark SQL for structured transformations, DataFrame API for aggregations, and Structured Streaming for incremental or event-driven updates.

2. Transformations, Actions & Use Cases

Q: Explain narrow vs. wide transformations.

A: Narrow transformations (e.g., filter, map) operate within partitions, while wide transformations (e.g., groupBy, join) trigger shuffles across nodes.

Q: How do you manage large joins?

A: Used broadcast joins for small lookup tables, repartitioned large tables, and leveraged Delta ZORDER for join optimization.

Q: How do you address data skew?

A: Added salting keys, used skew hints, or repartitioned by balanced keys to handle uneven data distribution.

3. Performance Tuning & Shuffle Optimization

Q: Which configurations do you tune for performance?

A: Adjusted spark.sql.shuffle.partitions, spark.default.parallelism, and enabled autoBroadcastJoinThreshold for optimized shuffles.

Q: How do you handle small file issues?

A: Used coalesce/repartition, Delta OPTIMIZE, and periodic compaction jobs.

Q: How do you cache and persist data?

A: Cached frequently reused intermediate DataFrames; persisted key datasets across multiple DBT model runs.

4. Schema Handling, Incremental Loads & Data Quality

Q: How do you handle schema drift?

A: Enabled mergeSchema for Delta tables and validated schema changes in DBT before production runs.

Q: How do you manage nulls and bad records?

A: Applied na.drop(), na.fill(), and UDF-based cleaning; directed bad records to a quarantine path for review.

Q: How do you perform incremental loads?

A: Implemented watermark-based ingestion, and Delta MERGE INTO to efficiently update only changed data.

5. Spark + DBT + Databricks Integration

Q: How do DBT models interact with Spark?

A: DBT compiles SQL logic that runs on Spark's engine in Databricks. Incremental DBT models are optimized using Spark SQL for large-scale updates.

Q: How do you orchestrate Spark jobs?

A: Used Airflow or Databricks Workflows to trigger Spark notebooks and DBT models with dependency management.

Q: How do you ensure reusability?

A: Parameterized transformations and used modular DBT macros for shared logic across multiple Spark jobs.

6. Monitoring, Debugging & Real-World Scenarios

Q: How do you monitor Spark job performance?

A: Used Databricks Spark UI for stage analysis, task skew, and shuffle metrics; tracked job run durations in Databricks Job metrics.

Q: Example of a performance optimization you implemented?

A: Optimized a large aggregation from 3 hours to 45 minutes by reducing shuffle partitions, broadcasting small tables, and caching intermediate data.

Q: How do you debug failing Spark stages?

A: Checked executor logs in the Spark UI, reviewed lineage in Unity Catalog, and reran partitions with lower concurrency to isolate bad data.

7. Governance, Security & Compliance

Q: How do you handle confidential data?

A: Masked or hashed customer identifiers and ensured data encryption at rest; enforced table-level permissions via Unity Catalog.

Q: How do you ensure lineage and traceability?

A: Delta Lake version history and Unity Catalog lineage tracking provide full visibility of source → transformation → output.

Q: What best practices do you follow for production pipelines?

A: Use job clusters, CI/CD deployments, parameterized notebooks, and automated validation checks in DBT + Spark.

■ Bonus Quick Reference: Spark Commands & Optimization Tips

Topic	Example / Command / Description
Read & Write Delta	spark.read.format('delta').load('/mnt/sales'); df.write.format('delta').save('/mnt/output')
Broadcast Join	df_large.join(broadcast(df_small), 'id')
Shuffle Partitions	spark.conf.set('spark.sql.shuffle.partitions', 400)
Handle Skew	df.repartition('region') or use salting technique
Optimize Table	OPTIMIZE gold.sales ZORDER BY (customer_id)
Cache Data	df.cache(); df.count()
Schema Evolution	df.write.option('mergeSchema', 'true').format('delta').save(path)
Incremental Load	MERGE INTO gold USING updates ON keys WHEN MATCHED THEN UPDATE WHEN NOT

These commands and best practices are ideal for Spark-based transformations within Databricks + DBT pipelines in a secure, governed banking environment.