

■ Databricks Interview Playbook

For Experienced Data Professionals (Banking & Retail Domain)

Comprehensive Q&A; Guide + Quick Reference for Senior-Level Interviews

1. Databricks Core Concepts

Q: What is Databricks and why is it used?

A: Databricks is a unified analytics and AI platform built on Apache Spark that combines data engineering, data science, and machine learning. In my project, we use it as a central data processing layer to transform raw banking transactions and customer data into curated, anonymized datasets for analytics.

Q: How does Databricks differ from traditional Spark?

A: Databricks provides managed clusters, collaborative notebooks, built-in Delta Lake, and optimized I/O performance, improving reliability compared to self-managed Spark.

Q: What is Unity Catalog and its role?

A: Unity Catalog provides centralized governance — access control, data lineage, and audit logging — across workspaces. We used it to manage permissions on de-identified retail data and track lineage across DBT models and reporting schemas.

2. Apache Spark on Databricks

Q: How do you optimize Spark jobs in Databricks?

A: Adjusted shuffle partitions, used broadcast joins for smaller tables, persisted intermediate data, and monitored via Spark UI.

Q: What's data skew and how do you handle it?

A: Occurs when certain keys have disproportionate data. Resolved using salting, repartitioning, or skew join hints.

Q: Difference between narrow and wide transformations?

A: Narrow (map, filter) don't require shuffles; wide (join, groupBy) do. Optimizing these is key for large datasets like transactions.

Q: What are the Catalyst Optimizer and Tungsten?

A: Catalyst performs query optimization (logical and physical plans). Tungsten handles in-memory computation and code generation for performance gains.

3. Delta Lake

Q: What is Delta Lake and why is it important?

A: Delta Lake brings ACID transactions, schema enforcement, and versioning to data lakes — ensuring reliability in banking data.

Q: Explain time travel in Delta Lake.

A: Query older versions of data using timestamps or version numbers — helpful for audit and rollback.

Q: How do you perform incremental loads?

A: Used MERGE INTO with watermark logic on timestamps to process only new or updated records.

Q: What's OPTIMIZE and ZORDER?

A: OPTIMIZE compacts small files; ZORDER collocates related data (e.g., by customer_id or region) for better performance.

Q: How do you handle schema evolution?

A: Enabled automatic schema evolution and used mergeSchema in writes to support changing data structures.

4. Medallion Architecture (Bronze–Silver–Gold)

Q: Explain the Medallion architecture.

A: Bronze: raw ingestion; Silver: cleaned and conformed data (via DBT); Gold: aggregated, anonymized business views.

Q: How do you ensure data quality?

A: DBT tests and Databricks expectations validate data integrity. Alerts trigger on failures.

Q: How do you handle schema drift?

A: Enabled automatic schema evolution in Delta and tracked metadata changes via Unity Catalog.

Q: How do you implement incremental pipelines?

A: Used Delta change data feed (CDF) and MERGE logic to propagate only changed data between layers.

5. Performance & Cost Optimization

Q: How do you optimize Databricks jobs for cost and speed?

A: Used auto-scaling clusters, auto-termination, caching, and efficient partitioning.

Q: How do you handle large joins efficiently?

A: Broadcasted smaller reference data, optimized shuffle partitions, and used ZORDER for co-location.

Q: How do you monitor performance?

A: Used Databricks Ganglia metrics, Spark UI, and job run history to analyze stages, shuffles, and task skew.

6. Integrations & Orchestration

Q: How do you orchestrate data pipelines?

A: DBT handles SQL-based transformations, orchestrated via Airflow with Databricks Jobs.

Q: How do you integrate Databricks with BI tools?

A: Exposed Gold tables as Delta Live Tables or SQL endpoints for Power BI dashboards.

Q: How do you use Git in Databricks?

A: Used Databricks Repos with GitHub for CI/CD and model versioning.

Q: How do you handle dependencies between jobs?

A: Used Databricks Jobs API and Airflow DAGs to maintain execution order and dependencies.

7. Real-World Project Scenarios

Q: Describe an end-to-end pipeline you built.

A: Raw retail transactions ingested to Bronze; DBT transformations create Silver; Gold contains daily spend, KPIs, and de-identified analytics.

Q: How did you manage data privacy?

A: Applied masking and tokenization via DBT macros before Gold; restricted raw access via Unity Catalog.

Q: What's a challenge you solved in Databricks?

A: Solved small-file performance degradation with OPTIMIZE, ZORDER, and better partitioning.

Q: How do you ensure regulatory compliance?

A: Implemented lineage through Unity Catalog and maintained audit-ready Delta version history.

■ Bonus Quick Reference: Databricks Commands & Optimization Tips

1. Common Delta Lake SQL Commands

Command / Topic	Example / Description
Create Delta Table	<code>CREATE TABLE sales_delta USING DELTA AS SELECT * FROM parquet.`/mnt/data/sales/`;</code>
Upsert (Merge)	<code>MERGE INTO target t USING updates u ON t.id = u.id WHEN MATCHED THEN UPDATE SET * WHEN NOT MATCHED THEN INSERT *;</code>
Time Travel Query	<code>SELECT * FROM sales_delta VERSION AS OF 3;</code>
Optimize & ZORDER	<code>OPTIMIZE sales_delta ZORDER BY (customer_id);</code>
Vacuum	<code>VACUUM sales_delta RETAIN 168 HOURS;</code>

2. Useful PySpark Commands

Command / Topic	Example / Description
Read Delta Table	<code>df = spark.read.format('delta').load('/mnt/data/sales_delta')</code>
Write Delta Table	<code>df.write.format('delta').mode('overwrite').save('/mnt/data/sales_delta')</code>
Repartition Data	<code>df = df.repartition(50)</code>
Cache Table	<code>df.cache()</code>
Display Data (Databricks)	<code>display(df)</code>

3. Performance Optimization Tips

Command / Topic	Example / Description
Partition Strategy	Partition large Delta tables by logical keys (e.g., region, month).
Small File Problem	Use OPTIMIZE regularly to compact files.
Broadcast Joins	Broadcast small lookup tables using broadcast(df).
Shuffle Optimization	Set spark.sql.shuffle.partitions based on data volume (e.g., 200–400).
Schema Evolution	Enable mergeSchema option for evolving data sources.

4. Data Governance & Security

Command / Topic	Example / Description
Unity Catalog Permissions	<code>GRANT SELECT ON TABLE gold.transactions TO analyst_role;</code>
Mask Sensitive Columns	Use DBT macros or CASE statements to hash/mask PII.
Audit Trail	Use Delta's version history and Unity Catalog lineage for audits.