Bank of America PySpark and SQL Interview Questions (0-3 years) 25 LPA

PySpark Interview questions

1. What is the Catalyst Optimizer in Spark SQL and how does it improve query performance?
Explanation:
Catalyst is Spark SQL's query optimization engine. It performs:

- Logical plan optimization: Simplifies expressions, constant folding, predicate pushdown.
- Physical plan optimization: Chooses efficient join strategies (broadcast vs shuffle hash join).
- Code generation: Uses whole-stage codegen for faster execution by generating optimized JVM bytecode.
- Result → Queries run faster with fewer shuffles and optimized execution plans.

2. Explain Tungsten Project in Spark and its role in performance tuning.
Explanation:

- Tungsten improves CPU and memory efficiency in Spark.
- Features:
  - Off-heap memory management → avoids GC overhead.
  - Cache-friendly binary row format → reduces serialization/deserialization costs.
  - Whole-stage code generation → generates Java bytecode to avoid virtual function calls.
  - Effect: Up to 10x faster execution compared to RDD-based operations.

3. How do you decide the cluster size for processing X GB of data in PySpark?
Explanation:
Cluster sizing depends on input data size, transformations, shuffle volume, and file format.

- **Rule of thumb:**
  - **Compressed Parquet/ORC → 1 core can process ~2–4 GB.**
  - **Raw JSON/CSV → 1 core can process ~1–2 GB.**
- **Memory requirement:**
  - **Keep at least 2–3x data size in memory to handle shuffles & caching.**
  - **Example: For 1 TB JSON, need ~500 cores (at ~2 GB/core).**
- **Partitioning: Target 128–256 MB per partition for efficient parallelism.**

4. Explain Adaptive Query Execution (AQE) in Spark 3.x.
Explanation:
AQE dynamically optimizes queries at runtime based on actual statistics. Key features:

- **Dynamic partition coalescing (reduce small files).**
- **Skew join optimization (split skewed partitions).**
- **Join strategy switching (switch shuffle join → broadcast join if table is small).**
- **Helps avoid over-provisioning clusters and reduces shuffle cost.**

5. PySpark JSON Handling: Parse nested JSON into a DataFrame.
Explanation:
Often interviewers test your ability to flatten deeply nested JSON.

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, explode

data = [
   ('{"id":1,"user":{"name":"Alice","emails":["a@gmail.com","a@yahoo.com"]}}',),
   ('{"id":2,"user":{"name":"Bob","emails":["b@gmail.com"]}}',)
]
df = spark.createDataFrame(data, ["json_str"])

# Parse JSON
parsed_df = spark.read.json(df.rdd.map(lambda r: r.json_str))

# Flatten
final_df = parsed_df.select(
   "id",
   col("user.name").alias("name"),
   explode("user.emails").alias("email")
)
final_df.show()
```

@mohitmotwani16

6. When would you use a broadcast join vs a shuffle join in PySpark?
Explanation:

- Broadcast join: If one dataset is small (<10MB by default, configurable via spark.sql.autoBroadcastJoinThreshold), Spark sends it to all executors to avoid shuffle.
- Shuffle join: Used when both datasets are large. Expensive due to data movement across nodes.
- Broadcast joins dramatically reduce shuffle overhead for skewed joins.

7. How do you tune shuffle partitions in PySpark and why is it important?
Explanation:

- Default: spark.sql.shuffle.partitions = 200.
- Too few → tasks are heavy, slow.
- Too many → overhead in task scheduling.
- Best practice:
- Set partitions ≈ 2–3x total cores in cluster.
- For large joins/aggregations, tune based on shuffle file size (aim for 128–256 MB per shuffle file).

8. How do you handle skewed data in a large JSON file during joins in PySpark?
Explanation:

- Identify skewed keys (e.g., groupBy("user_id").count().orderBy(desc("count"))).
- Techniques:
  - Salting → Add random prefix to skewed keys, join, then remove prefix.
  - Broadcast small dataset to all executors.
  - Skew join optimization (if using AQE in Spark 3.x).
- Example salting approach:

SQL Interview questions

1. Write a SQL query to find users who placed orders on 3 consecutive days.
Explanation:
Tests problem-solving skills with date sequences. Interviewers want to see if you can use window functions creatively to detect consecutive patterns instead of brute force joins.

\

2. Query Optimization in a Data Lakehouse
Explanation:
Tests if you understand performance tuning in big data SQL (partition pruning, predicate pushdown, ZORDER, bucketing). It separates someone who only writes SQL from someone who designs efficient pipelines.

3. Running Totals Using Window Functions
Explanation:
Checks your ability to use window frames (ROWS BETWEEN ...) for cumulative aggregations. Important in finance (balances, P&L) and clickstream analytics.

4. Join Question with Sample Tables
Two small tables are given:
Table A:

id	name
1	Alice
2	Bob
3	Charlie

Table B:

id	city
2	London
3	New York
4	Paris

Question: What will be the result of applying each of the following joins on A.id = B.id?

- INNER JOIN
- LEFT JOIN
- RIGHT JOIN
- FULL OUTER JOIN

Explanation:
This checks if the candidate can mentally simulate join outputs. Most people confuse which side produces NULLs. It’s an excellent way to test whether the candidate memorized join definitions or actually understands them with real rows.

5. Find employees who earn more than their managers
Explanation:
This question tests self-joins. The table has emp_id, name, salary, and manager_id. You need to join the table with itself (employees e JOIN employees m ON e.manager_id = m.emp_id) and then compare salaries. It checks if you can handle hierarchical data in flat tables.

6. Find the department(s) with the highest average salary
Explanation:
Here you must use GROUP BY to calculate avg salary per department, then compare against the maximum average across all departments. Tests your knowledge of nested aggregation or window functions.