PySpark Interview Questions with Sample Scripts and Explanations

Why does Spark shuffle data, and how can you minimize it?

### Example: `groupByKey` vs `reduceByKey`

```python
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("Shuffle Example").getOrCreate()
from pyspark.sql.functions import col


rdd = spark.sparkContext.parallelize([("a", 1), ("b", 2), ("a", 3), ("b", 4)])


# Inefficient (shuffles all values to a single key)
result1 = rdd.groupByKey().mapValues(list)
print(result1.collect())


# Optimized (performs partial aggregation before shuffle)
result2 = rdd.reduceByKey(lambda x, y: x + y)
print(result2.collect())
```

Explanation: `reduceByKey` avoids shuffling all data by performing local aggregation before shuffle, reducing network overhead.

How does Spark handle skewed data in groupBy operations?

### Example: Salting Technique

```python
from pyspark.sql import functions as F
```

```
data = [("a", 100), ("a", 200), ("a", 300), ("b", 10), ("b", 20)]

df = spark.createDataFrame(data, ["key", "value"])


# Add random salt to skewed keys

df_salted = df.withColumn("salt", (F.rand() * 10).cast("int"))

df_salted = df_salted.withColumn("key_salted", F.concat(col("key"), col("salt")))


# Group by salted key

result = df_salted.groupBy("key_salted").sum("value")

result.show()
```

Explanation: Adding a random salt distributes skewed keys across multiple partitions.

What's the secret behind Catalyst Optimizer in Spark SQL?

### Example: Query Execution Plan

```python

df = spark.createDataFrame([(1, "Alice"), (2, "Bob")], ["id", "name"])

df.select("id").explain(True)

```

Explanation: `explain(True)` shows logical and physical plans optimized by Catalyst.

Repartition() vs Coalesce() - when should you use each?

### Example: `repartition` vs `coalesce`

```python

df = spark.range(0, 1000000)


# Increase partitions (shuffle involved)

df_repartitioned = df.repartition(10)
```

```
print(df_repartitioned.rdd.getNumPartitions())

# Decrease partitions (no shuffle)
df_coalesced = df_repartitioned.coalesce(2)
print(df_coalesced.rdd.getNumPartitions())
```

Explanation: `repartition()` is used for load balancing, while `coalesce()` is optimized for reducing partitions without shuffle.

Why is a broadcast join faster than a shuffle join?

### Example: Using `broadcast()`

```python
from pyspark.sql.functions import broadcast
small_df = spark.createDataFrame([(1, "USA"), (2, "India")], ["id", "country"])
large_df = spark.range(1000000).withColumnRenamed("id", "user_id")

# Broadcast the small DataFrame
result = large_df.join(broadcast(small_df), large_df.user_id == small_df.id, "left")
result.explain()
```

Explanation: Broadcasting avoids shuffling by sending the small dataset to all nodes.