PySpark

🔹 Basic PySpark Questions

Q1. What is PySpark? How is it different from Apache Spark?

PySpark is the Python API for Apache Spark, allowing Python developers to write Spark applications. Apache Spark is the core distributed processing engine, written in Scala, whereas PySpark provides a Pythonic interface to Spark features.

Q2. How do you create a SparkSession? Why is it important?

`SparkSession` is the entry point for working with DataFrames and SQL in PySpark. It manages Spark configurations and provides APIs.

```python
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("myApp").getOrCreate()
```

Q3. What are the different types of RDD transformations and actions?

- **Transformations** (lazy):
  - `map()`
  - `filter()`
  - `flatMap()`
  - `groupByKey()`
- **Actions** (trigger execution):
  - `collect()`
  - `count()`
  - `reduce()`
  - `first()`

Q4. Difference between RDD, DataFrame, and Dataset?

- **RDD**: Low-level, unstructured data; more control, less optimization
- **DataFrame**: Structured with schema; optimized via Catalyst
- **Dataset**: Combines RDD and DataFrame benefits; not available in PySpark (Scala/Java only)

Q5. How do you read a CSV/JSON/Parquet file in PySpark?

```python
spark.read.csv("file.csv", header=True, inferSchema=True)
spark.read.json("file.json")
spark.read.parquet("file.parquet")
```

Q6. What is lazy evaluation in Spark?

- Spark delays execution of transformations until an action is called.
- This allows it to optimize the entire data flow.

Q7. How do you show the schema of a DataFrame?

```python
df.printSchema()
```

🔹 DataFrame Operations

Q8. How do you filter rows in a DataFrame?

```python
df.filter(df.age > 30).show()
df.where(df.age < 50).show()
```

Q9. How do you select and rename columns in PySpark?

```python
df.select("name", "age")
df.withColumnRenamed("old_name", "new_name")
```

Q10. What is the difference between `select()` and `selectExpr()`?

- `select()` uses column objects
- `selectExpr()` allows SQL expressions as strings

```python
df.selectExpr("age + 10 as age_plus_10")
```

Q11. How do you perform joins in PySpark?

```python
df1.join(df2, df1.id == df2.id, "inner")
df1.join(df2, "id", "left")
df1.join(df2, "id", "right")
df1.join(df2, "id", "outer")
```

Q12. How do you group data and perform aggregations?

```python
df.groupBy("dept").agg({"salary": "avg", "age": "max"})
```

Q13. How do you handle null values in PySpark?

```python
df.dropna()
df.fillna(0)
df.na.replace("old", "new")
```

Q14. Explain the use of `withColumn()` and `withColumnRenamed()`

- `withColumn()` creates or updates a column
- `withColumnRenamed()` changes column name

```python
df.withColumn("new_col", df.age + 10)
df.withColumnRenamed("old", "new")
```

Q15. How do you apply a UDF (User Defined Function) in PySpark?

```python
from pyspark.sql.functions import udf
from pyspark.sql.types import StringType

def to_upper(s): return s.upper()
upper_udf = udf(to_upper, StringType())
df.withColumn("upper_name", upper_udf(df.name))
```

🔹 Performance Optimization

Q16. What are broadcast joins and when would you use them?

- Broadcast joins replicate the smaller table to all workers, avoiding shuffle.
- Use when one table is small enough to fit in memory.

```python
from pyspark.sql.functions import broadcast
df_large.join(broadcast(df_small), "id")
```

Q17. What is caching and persistence in Spark?

Used to store intermediate results in memory/disk to avoid recomputation.

```python
df.cache()        # memory only
df.persist()      # memory and/or disk based on storage level
```

Q18. How do you check the execution plan of a DataFrame?

```python
df.explain()
```

Q19. Difference between narrow and wide transformations?

- **Narrow**: Data resides in same partition (e.g., `map`, `filter`)
- **Wide**: Data moves between partitions (e.g., `groupBy`, `join`)

Q20. How to optimize shuffle operations in Spark?

- Use broadcast joins
- Reduce data skew
- Repartition smartly
- Avoid unnecessary wide transformations

🔹 PySpark SQL and Window Functions

Q21. How do you use SQL queries in PySpark using `spark.sql()`?

```python
df.createOrReplaceTempView("people")
spark.sql("SELECT * FROM people WHERE age > 25")
```

Q22. What are window functions? Give an example.

Window functions operate on a partition of rows without collapsing them.

Example:

```python
from pyspark.sql.window import Window
from pyspark.sql.functions import row_number

windowSpec = Window.partitionBy("dept").orderBy("salary")
df.withColumn("row_num", row_number().over(windowSpec))
```

Q23. How do you rank or row-number records partitioned by a column?

Use window functions like `row_number()`, `rank()`, `dense_rank()` with a partition spec.

Q24. Difference between `rank()`, `dense_rank()`, and `row_number()`?

- `rank()`: Gaps in ranking for ties
- `dense_rank()`: No gaps
- `row_number()`: Unique sequence per row

🔹 Advanced Topics

Q25. How do you handle skewed data in Spark joins?

- Use salting technique
- Broadcast smaller table
- Repartition on join key
- Avoid skewed keys in grouping

Q26. What is the role of Catalyst Optimizer in Spark SQL?

Catalyst Optimizer analyzes and rewrites queries using optimization rules, leading to better performance.

Q27. What is Tungsten in Spark?

Tungsten is Spark's execution engine that improves memory and CPU efficiency through binary processing and code generation.

Q28. How do you read/write to Hive tables using PySpark?

```python
spark.sql("SELECT * FROM my_hive_table")
df.write.mode("overwrite").saveAsTable("my_hive_table")
```

Q29. How do you manage schema evolution in Spark?

Use schema merging in Parquet and manage `StructType` schemas with nullable fields.

```python
spark.read.option("mergeSchema", "true").parquet("path")
```

Q30. Explain checkpointing in Spark. When should you use it?

- Checkpointing saves RDD/DataFrame to reliable storage to break long lineage and avoid recomputation.
- Used in long DAGs or streaming jobs.

```python
sc.setCheckpointDir("hdfs://path")
df.checkpoint()
```

🔹 Scenario-Based Questions

Q31. How would you handle a dataset with 1 billion rows in PySpark efficiently?

- Use partitioned Parquet
- Repartition smartly
- Avoid wide transformations
- Use caching and optimized joins

Q32. If a PySpark job is failing due to memory error, what steps will you take to debug it?

- Tune executor memory
- Reduce data skew
- Use checkpoint or persist to disk
- Avoid caching large datasets

Q33. You have to join a small lookup table with a large fact table — how will you optimize it?

Broadcast the small lookup table to avoid shuffle:

```python
df_large.join(broadcast(df_small), "key")
```

Q34. How would you read nested JSON and flatten it in PySpark?

```python
df = spark.read.json("nested.json")
df.select("id", "address.city", "address.zip")
```

Q35. How do you write unit tests for PySpark code?

Use Python `unittest` or `pytest` with a local `SparkSession`.

```python
import unittest
from pyspark.sql import SparkSession

class MyTest(unittest.TestCase):
    def setUp(self):
        self.spark =
SparkSession.builder.master("local").appName("Test").getOrCreate(
)

    def test_count(self):
        df = self.spark.createDataFrame([(1, "A"), (2, "B")],
["id", "val"])
        self.assertEqual(df.count(), 2)
```
