General Questions

1. What is PySpark, and how does it function?
2. Can you explain the architecture of PySpark?
3. What are the key differences between RDD, DataFrame, and Dataset in PySpark?
4. How does Spark ensure fault tolerance?
5. What types of transformations are available in PySpark?
6. What is the difference between narrow and wide transformations in PySpark?
7. What is the role of the Directed Acyclic Graph (DAG) in Spark?
8. What are the functions of the driver and executor in PySpark?
9. How does PySpark handle schema inference?
10. What types of joins are supported in PySpark?

Performance Optimization Questions

11. How do `cache()` and `persist()` differ in PySpark?
12. What is the significance of broadcasting in Spark?
13. How can you optimize Spark jobs for better performance?
14. What are the advantages of partitioning in PySpark?
15. What is the difference between `repartition()` and `coalesce()` in PySpark?
16. How do you handle skewed data in PySpark?
17. How does Spark execute operations lazily?
18. What are the different storage levels in Spark?
19. How can you optimize join operations in PySpark?
20. What is Adaptive Query Execution (AQE) in Spark?

PySpark Coding Questions

21. Write a PySpark program to find the top 3 most frequent words in a given dataset.
22. Given a DataFrame, write a PySpark code to remove duplicate records.
23. How do you perform word count using PySpark?
24. Write a PySpark job to group data by a column and calculate the average value.
25. How do you handle missing or null values in a PySpark DataFrame?
26. Write a PySpark program to count distinct values in a column.
27. Given a DataFrame, write a PySpark code to filter records where the salary is greater than 50,000.

28. Write a PySpark job to read a JSON file and convert it into a DataFrame.
29. Write a PySpark query to find the second-highest salary from an employee table.
30. Write a PySpark job to join two DataFrames and select specific columns.

PySpark SQL Questions

31. How do you create a temporary view in PySpark?
32. What is the difference between `df.select()` and `df.withColumn()`?
33. How do you execute SQL queries in PySpark?
34. What is the difference between `explode()` and `posexplode()`?
35. How do you convert a DataFrame into a SQL table?
36. How do you use window functions in PySpark?
37. How can you rank rows in PySpark using `RANK()` and `DENSE_RANK()`?
38. Write a PySpark SQL query to calculate the cumulative sum of a column.
39. What is the difference between `groupBy()` and `rollup()`?
40. How do you perform pivot operations in PySpark?

Streaming and Real-time Processing

41. What is Spark Streaming, and how does it function?
42. What is the difference between structured and unstructured streaming?
43. How do you manage late-arriving data in Spark Streaming?
44. What are watermarks in Spark Streaming?

Answers :-

Here are the answers to all the questions:

Here are the answers to all the questions:

General Questions

1. **What is PySpark, and how does it function?**
   PySpark is the Python API for Apache Spark, which is an open-source, distributed computing system. PySpark allows Python developers to interact with Spark and take advantage of its distributed computing capabilities. It functions by processing data in parallel across a cluster of machines, utilizing concepts like RDDs (Resilient Distributed Datasets) and

DataFrames.

2. **Can you explain the architecture of PySpark?**
   The architecture of PySpark consists of several key components:

   - **Driver**: The main process that controls the execution of the Spark job. It coordinates the tasks and schedules them.
   - **Cluster Manager**: This is responsible for allocating resources across the cluster.
   - **Executors**: These are worker nodes responsible for running individual tasks and storing data.
   - **Tasks**: The smallest unit of work in Spark.
   - **RDD**: The basic data structure representing a distributed collection of objects.

3. **What are the key differences between RDD, DataFrame, and Dataset in PySpark?**

   - **RDD**: Represents a distributed collection of objects in memory. It provides low-level operations but lacks optimizations like catalyst query optimization.
   - **DataFrame**: A higher-level abstraction over RDDs, similar to a table in a relational database. It supports SQL operations and benefits from Spark's optimizations.
   - **Dataset**: Introduced in Spark 1.6, it combines the best of both RDDs and DataFrames, providing the benefits of both object-oriented programming (strong typing) and optimizations (like in DataFrames).

4. **How does Spark ensure fault tolerance?**
   Spark ensures fault tolerance using **RDD lineage**. Each RDD keeps track of the operations that created it, so in case of failure, Spark can recompute the lost data from its original source, ensuring the integrity of the dataset.

5. **What types of transformations are available in PySpark?**
   There are two main types of transformations:

   - **Narrow Transformations**: These transformations involve one-to-one mapping (e.g., `map()`, `filter()`).
   - **Wide Transformations**: These require data from multiple partitions (e.g., `groupBy()`, `join()`).

6. **What is the difference between narrow and wide transformations in PySpark?**

    - **Narrow Transformations**: Operate on a single partition and do not require data from other partitions (e.g., `map()`, `filter()`).

    - **Wide Transformations**: Require data from multiple partitions and involve shuffling (e.g., `groupBy()`, `reduceByKey()`).

7. **What is the role of the Directed Acyclic Graph (DAG) in Spark?**
   The DAG represents the sequence of operations Spark performs on the RDDs. It allows Spark to optimize execution by minimizing the number of stages and narrowing down transformations that can be performed together.

8. **What are the functions of the driver and executor in PySpark?**

    - **Driver**: Coordinates the execution of the job and distributes tasks to the executors.

    - **Executor**: Runs tasks assigned by the driver and stores data for the job.

9. **How does PySpark handle schema inference?**
   PySpark automatically infers the schema when loading data from sources like JSON, CSV, or Parquet by analyzing the first few rows of the dataset and deriving column names and data types.

10. **What types of joins are supported in PySpark?**
    PySpark supports the following types of joins:

    - **Inner Join**

    - **Outer Join** (Full, Left, Right)

    - **Cross Join**

    - **Semi Join**

    - **Anti Join**

Performance Optimization Questions

11. **How do `cache()` and `persist()` differ in PySpark?**

- **cache()**: Stores the DataFrame or RDD in memory (default storage level: MEMORY_AND_DISK).

- **persist()**: Allows you to specify a custom storage level, which can be memory, disk, or both, providing more flexibility.

12. **What is the significance of broadcasting in Spark?**
    Broadcasting is used to send a copy of a large variable (such as a lookup table) to all worker nodes, so they don't have to access the data multiple times. This can significantly reduce data shuffling in join operations.

13. **How can you optimize Spark jobs for better performance?**
    Some common optimization strategies include:

    - Use **cache()** or **persist()** for frequently used data.

    - Optimize **join operations** by using broadcast joins.

    - Repartition data when necessary to improve parallelism.

    - Tune the number of **executors** and **partitions** based on the workload.

    - Enable **adaptive query execution (AQE)** for better physical plan optimization.

14. **What are the advantages of partitioning in PySpark?**
    Partitioning splits the data into smaller chunks that can be processed in parallel, improving the performance of distributed operations. Proper partitioning reduces the amount of data shuffled across the network.

15. **What is the difference between `repartition()` and `coalesce()` in PySpark?**

    - **repartition()**: Shuffles the data across partitions, and is typically used when you want to increase the number of partitions.

    - **coalesce()**: Reduces the number of partitions without moving much data (more efficient when reducing partitions).

16. **How do you handle skewed data in PySpark?**

    - Use **salting** (adding a random key to partition the data) to distribute data more evenly.

    - Apply **broadcast joins** when one of the datasets is small.

17. **How does Spark execute operations lazily?**
    Spark builds a DAG of operations and only triggers execution when an action (such as `collect()` or `save()`) is called. This allows Spark to optimize the execution plan before actually performing the computation.

18. **What are the different storage levels in Spark?**
    Storage levels include:
    - **MEMORY_ONLY**
    - **MEMORY_AND_DISK**
    - **DISK_ONLY**
    - **MEMORY_ONLY_SER**
    - **MEMORY_AND_DISK_SER**
    - **OFF_HEAP**

19. **How can you optimize join operations in PySpark?**
    - Use **broadcast joins** when one of the datasets is small.
    - Repartition the data to optimize partitioning for join operations.
    - Avoid **shuffling** by using appropriate partitioning strategies.

20. **What is Adaptive Query Execution (AQE) in Spark?**
    AQE dynamically optimizes the physical query plan during runtime by adjusting the plan based on the data being processed, improving performance.

PySpark Coding Questions

Write a PySpark program to find the top 3 most frequent words in a given dataset.

```
from pyspark.sql import SparkSession

from pyspark.sql.functions import explode, split
```

```
spark = SparkSession.builder.appName("WordCount").getOrCreate()
data = spark.read.text("data.txt")
words = data.select(explode(split(data.value, " ")).alias("word"))
word_counts = words.groupBy("word").count().orderBy("count", ascending=False)
word_counts.show(3)
```

21.

Given a DataFrame, write a PySpark code to remove duplicate records.

```
df = df.dropDuplicates()
```

22.

How do you perform word count using PySpark?

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

spark = SparkSession.builder.appName("WordCount").getOrCreate()
text_file = spark.read.text("data.txt")
words = text_file.select(explode(split(text_file.value, " ")).alias("word"))
word_count = words.groupBy("word").count()
word_count.show()
```

23.

Write a PySpark job to group by a column and calculate the average value.

```
df.groupBy("column_name").avg("value_column").show()
```

24.

How do you handle missing/null values in a PySpark DataFrame?

```
df.fillna({"column_name": "default_value"})
```

25.

Write a PySpark program to count distinct values in a column.

```
df.select("column_name").distinct().count()
```

26.

Given a DataFrame, write a PySpark code to filter records where salary > 50,000.

```
df.filter(df.salary > 50000).show()
```

27.

Write a PySpark job to read a JSON file and convert it into a DataFrame.

```
df = spark.read.json("data.json")
```

28.

Write a PySpark query to find the second highest salary from an employee table.

```
from pyspark.sql import functions as F
df.select("salary").distinct().orderBy(F.col("salary").desc()).limit(2).tail(1)
```

29.

Write a PySpark job to join two DataFrames and select specific columns.

```
df1.join(df2, df1.id == df2.id).select(df1.name, df2.salary).show()
```

30.

PySpark SQL Questions

How do you create a temporary view in PySpark?

```
df.createOrReplaceTempView("temp_view_name")
```

31.
32. **What is the difference between `df.select()` and `df.withColumn()`?**

- **select()**: Used to select columns and transform data.
- **withColumn()**: Used to create a new column or modify an existing one.

How do you execute SQL queries in PySpark?

```
spark.sql("SELECT * FROM temp_view_name")
```

33.
34. **What is the difference between `explode()` and `posexplode()`?**

- **explode()**: Flattens an array into multiple rows.
- **posexplode()**: Flattens an array and also includes the position of each element.

How do you convert a DataFrame into a SQL table?

```
df.createOrReplaceTempView("sql_table_name")
```

35.

How do you use window functions in PySpark?

```
from pyspark.sql.window import Window
from pyspark.sql import functions as F
windowSpec = Window.partitionBy("column_name").orderBy("column_name")
df.withColumn("rank", F.rank().over(windowSpec)).show()
```

36.

How can you rank rows in PySpark using `RANK()` and `DENSE_RANK()`?

```
from pyspark.sql import functions as F
from pyspark.sql.window import Window

windowSpec = Window.orderBy(F.desc("column_name"))
df.withColumn("rank", F.rank().over(windowSpec))
```

df.withColumn("dense_rank", F.dense_rank().over(windowSpec))

37.

Write a PySpark SQL query to calculate the cumulative sum of a column.

from pyspark.sql import functions as F

from pyspark.sql.window import Window

windowSpec = Window.orderBy("id").rowsBetween(Window.unboundedPreceding, Window.currentRow)

df.withColumn("cumulative_sum", F.sum("column_name").over(windowSpec))

38.

39. **What is the difference between `groupBy()` and `rollup()`?**

    - **groupBy()**: Aggregates data based on specified columns.
    - **rollup()**: Performs a multi-level aggregation across multiple columns.

How do you perform pivot operations in PySpark?

df.groupBy("column1").pivot("column2").agg(F.sum("column3"))

40.

Streaming and Real-time Processing

41. **What is Spark Streaming, and how does it function?**
    Spark Streaming is an extension of Spark that enables processing of real-time data streams. It divides the stream into small batches (micro-batches), which are then processed using Spark's RDDs.

42. **What is the difference between structured and unstructured streaming?**

    - **Structured Streaming**: Processes data as a continuous table (i.e., DataFrames).

- **Unstructured Streaming**: Processes raw data without the abstraction of DataFrames or SQL.

43. **How do you handle late-arriving data in Spark Streaming?**
    You can use **watermarks** to specify how late data can be before it is dropped, and you can also use windowing to aggregate over a time period.

44. **What are watermarks in Spark Streaming?**
    Watermarks allow Spark to handle late-arriving data by defining a threshold to define how far back in time the data can arrive before it is considered too late to process.