PySpark :-

1️⃣ RDD (Resilient Distributed Dataset):-

- RDD PySpark का **low-level distributed collection** है।
- Immutable होता है (create होने के बाद modify नहीं होता)।
- Parallel processing के लिए optimized।
- 2 तरीके से बनाया जा सकता है: **parallelize()** और **textFile()**।

📌 Syntax:-

```
# From Collection
rdd = spark.sparkContext.parallelize([1, 2, 3])

# From File
rdd = spark.sparkContext.textFile("data.txt")
```

📌 Example:-

```
# parallelize example
data = [1, 2, 3, 4, 5]
rdd = spark.sparkContext.parallelize(data)
print(rdd.collect())

# textFile example
rdd2 = spark.sparkContext.textFile("/path/to/file.txt")
print(rdd2.first())
```

2️⃣ Actions in RDD:-

📌 **Theory**

- Actions वह operations हैं जो **result return करते हैं** या **data write करते हैं**।
- यह transformations को execute करते हैं।

📌 Syntax:-

```
rdd.collect()

rdd.count()

rdd.take(5)

rdd.first()
```

📌 Example:-

```
rdd = spark.sparkContext.parallelize([10, 20, 30, 40])

print(rdd.collect())  # [10, 20, 30, 40]

print(rdd.count())    # 4

print(rdd.take(2))    # [10, 20]
```

3️⃣ **Transformations in RDD**

📌 **Theory**

- Transformations RDD पर operations apply करते हैं और **new RDD return करते हैं**।
- Lazy evaluation होती है (जब तक action call नहीं करते execute नहीं होता)।

📌 Syntax:-

```
rdd.map(lambda x: x*2)

rdd.filter(lambda x: x > 10)

rdd.flatMap(lambda x: x.split())
```

📌 Example:-

```
data = ["hello world", "spark rdd"]
rdd = spark.sparkContext.parallelize(data)
words = rdd.flatMap(lambda line: line.split())
print(words.collect())  # ['hello', 'world', 'spark', 'rdd']
```

4️⃣ DataFrames

📌 Theory

- DataFrame SQL table जैसा structured data format है।
- इसमें named columns होते हैं और यह optimized API provide करता है।

📌 Syntax:-

```
data = [("Alice", 25), ("Bob", 30)]
df = spark.createDataFrame(data, ["Name", "Age"])
df.show()
```

📌 Example:-

```
data = [("Rahul", 5000), ("Amit", 6000)]
df = spark.createDataFrame(data, ["Name", "Salary"])
df.show()
```

5️⃣ Joins in DataFrames

📌 Theory

- Joins दो DataFrames को common column (key) पर combine करते हैं।
- Types: Inner, Left, Right, Full, Cross।

📌 Syntax:-

```
df1.join(df2, "id", "inner")
```

📌 Example:-

```
df1 = spark.createDataFrame([(1, "A"), (2, "B")], ["id", "val1"])
df2 = spark.createDataFrame([(1, "X"), (3, "Y")], ["id", "val2"])
df1.join(df2, "id", "inner").show()
```

6️⃣ Aggregations in DataFrames

📌 Theory

- Aggregations data को summarize करने के लिए use होती हैं (जैसे SUM, COUNT, AVG)।
- groupBy() के साथ aggregate functions apply होते हैं।

📌 Syntax:-

```
from pyspark.sql.functions import sum, avg, count
df.groupBy("column").agg(sum("col"), avg("col"))
```

📌 Example:-

```
data = [("HR", 5000), ("IT", 6000), ("HR", 5500)]
df = spark.createDataFrame(data, ["Dept", "Salary"])
df.groupBy("Dept").agg(sum("Salary"), avg("Salary")).show()
```

7️⃣ Window Functions

📌 Theory

- Window functions data को partition करके operate करती हैं।
- Row ranking, running total, lag-lead जैसी calculations possible हैं।

📌 Syntax:-

```
from pyspark.sql.window import Window
from pyspark.sql.functions import row_number
windowSpec = Window.partitionBy("Dept").orderBy("Salary")
df.withColumn("Rank", row_number().over(windowSpec))
```

📌 Example:-

```
data = [("HR", "A", 5000), ("HR", "B", 6000), ("IT", "C", 5500)]
df = spark.createDataFrame(data, ["Dept", "Emp", "Salary"])
windowSpec = Window.partitionBy("Dept").orderBy("Salary")
df.withColumn("Rank", row_number().over(windowSpec)).show()
```

8️⃣ File Formats

📌 Theory

- PySpark multiple file formats read/write कर सकता है: CSV, JSON, Parquet, ORC।

📌 Syntax:-

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

📌 Example:-

```
df_csv = spark.read.csv("data.csv", header=True)
df_json = spark.read.json("data.json")
df_parquet = spark.read.parquet("data.parquet")
```

9️⃣ Partitioning

📌 Theory

- Partitioning data को छोटे chunks में divide करता है ताकि parallel processing fast हो।

📌 Syntax:-

```
df.repartition(4)    # Increase partitions
df.coalesce(2)       # Reduce partitions
```

📌 Example:-

```
df = spark.range(0, 20)
df2 = df.repartition(4)
```

```
print(df2.rdd.getNumPartitions())
```

🔟 Bucketing

📌 Theory

- Bucketing data को fixed number of buckets में divide करता है hash function के base पर।

📌 Syntax:-

```
df.write.bucketBy(4, "name").sortBy("age").saveAsTable("bucketed_table")
```

📌 Example:-

```
df = spark.createDataFrame([("A", 25), ("B", 30)], ["name", "age"])
df.write.bucketBy(2, "name").saveAsTable("bucketed_data")
```

1️⃣1️⃣ UDF (User Defined Function)

📌 Theory

- UDF का use तब होता है जब PySpark के built-in functions आपकी requirement को cover न करें।
- UDF Python function को SQL function की तरह use करने देता है।

📌 Syntax:-

```
from pyspark.sql.functions import udf
from pyspark.sql.types import StringType
def my_upper(name):
    return name.upper()

upper_udf = udf(my_upper, StringType())
df.withColumn("UpperName", upper_udf(df["name"]))
```

📌 Example:-

```
data = [("alice",), ("bob",)]
df = spark.createDataFrame(data, ["name"])
```

```
def my_upper(name):
    return name.upper()
upper_udf = udf(my_upper, StringType())
df.withColumn("UpperName", upper_udf(df["name"])).show()
```

1️⃣2️⃣ Performance Optimization

📌 Theory

- Performance improve करने के लिए cache, broadcast, repartition जैसे techniques use करते हैं।

📌 Syntax:-

```
df.cache()
from pyspark.sql.functions import broadcast
df1.join(broadcast(df2), "id")
```

📌 Example:-

```
df.cache()
df.show()
from pyspark.sql.functions import broadcast
df1.join(broadcast(df2), "id").show()
```

1️⃣3️⃣ Cache & Persist

📌 Theory

- Cache memory में data store करता है।
- Persist memory + disk दोनों में data store कर सकता है।

📌 Syntax:-

```
df.cache()
from pyspark import StorageLevel
df.persist(StorageLevel.MEMORY_AND_DISK)
```

📌 Example:-

df.cache()

df.show()

df.persist(StorageLevel.MEMORY_AND_DISK)

df.show()

1️⃣4️⃣ Repartition & Coalesce

📌 **Theory**

- repartition() partitions increase/decrease करता है (shuffle करता है)।
- coalesce() partitions decrease करता है (shuffle नहीं करता)।

📌 Syntax:-

df.repartition(6)

df.coalesce(2)

📌 Example:-

df = spark.range(0, 20)

df_repart = df.repartition(6)

df_coal = df.coalesce(2)

1️⃣5️⃣ Broadcast Join

📌 **Theory**

- Broadcast join small dataset को सभी nodes में copy करता है जिससे large dataset के साथ join fast होता है।

📌 Syntax:-

from pyspark.sql.functions import broadcast

df1.join(broadcast(df2), "id")

📌 Example:-

small_df = spark.createDataFrame([(1, "A")], ["id", "val"])

large_df = spark.range(0, 1000).withColumnRenamed("id", "id")

```
large_df.join(broadcast(small_df), "id").show()
```

1️⃣6️⃣ Error Handling in PySpark

📌 Theory

- PySpark jobs में runtime errors (schema mismatch, file not found, data type errors) आ सकते हैं।
- Error handling से program crash होने से बचता है और controlled messages दिखाए जा सकते हैं।

📌 Syntax:-

```
try:
    df = spark.read.csv("data.csv", header=True)
    df.show()
except Exception as e:
    print("Error occurred:", e)
```

📌 Example:-

```
try:
    df = spark.read.csv("/invalid/path/file.csv", header=True)
    df.show()
except Exception as e:
    print("File read error:", e)
```

1️⃣7️⃣ Schema Evolution

📌 Theory

- Schema evolution से files में नए columns add/remove होने पर भी read possible है।
- Parquet और ORC formats में use होता है।

📌 Syntax:-

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

📌 Example:-

```
df = spark.read.option("mergeSchema", "true").parquet("/data/year=2024")
df.show()
```

1️⃣8️⃣ Deployment & Job Scheduling

📌 Theory

- Deployment का मतलब PySpark job को production environment में चलाना।
- Scheduling का मतलब job को regular interval (daily, weekly) में run करना।

📌 Syntax:-

```
# Local / cluster submit
spark-submit --master yarn job.py
# Airflow DAG (schedule job)
```

📌 Example:-

```
# Airflow example (pseudo)
from airflow import DAG
from airflow.operators.bash import BashOperator

dag = DAG('pyspark_job', schedule_interval='@daily')

task = BashOperator(
    task_id='run_spark',
    bash_command='spark-submit --master yarn /path/job.py',
    dag=dag
)
```

Question & Answer

1. CSV file read करके RDD बनाना।
2. Server log files process करना।
3. IoT sensor data parallel load करना।
4. Large dataset filtering RDD level पर करना।
5. Text analytics (word count) करना।

---

6. • Sales data count निकालना।
7. • IoT device readings का total records निकालना।
8. • Top 5 highest sales records निकालना।
9. • Customer feedback dataset का first record देखना।
10. • Final output को collect करके Python processing में use करना।

---

11. • Sales price को discount के साथ map करना।
12. • Negative readings को filter करना।
13. • Text files से words निकालना।
14. • IoT sensor readings को transformations से clean करना।
15. • Customer feedback से unwanted characters remove करना।

---

16. CSV sales data को DataFrame में load करना।
17. HR employee records analyze करना।
18. IoT readings का structured report बनाना।
19. E-commerce orders data query करना।
20. SQL जैसी aggregations perform करना।

---

21. • Customer table और Orders table join करना।
22. • Product table और Sales table join करना।
23. • IoT device info और readings join करना।
24. • HR employee और department data join करना।
25. • Campaign और lead data merge करना।

---

26. Department-wise employee salaries का sum।
27. Product category-wise total sales।
28. Region-wise average revenue।

29. Month-wise customer count।

30. IoT device readings का daily average।

---

31. • Department-wise top 3 salaries।

32. • Product category-wise best-selling products।

33. • Daily running total of sales।

34. • IoT readings का moving average।

35. • Customer transaction ranking।

---

36. • CSV में sales data load करना।

37. • JSON logs parse करना।

38. • Parquet format में big data store करना।

39. • ORC format में financial data save करना।

40. • Multiple formats combine करके reporting करना।

---

41. • Large dataset को 10 partitions में split करना।

42. • Output को month-wise partition करना।

43. • Data को region-wise partition करना।

44. • IoT readings को device_id पर partition करना।

45. • Cloud storage पर optimized writes।

---

46. Customer data को id-based buckets में divide करना।

47. Product catalog को category hash buckets में रखना।

48. Transaction history को fast join के लिए bucket करना।

49. IoT readings को sensor_id के buckets में रखना।

50. Analytics queries speed up करना।

---

51. • Customer names को uppercase करना।

52. • Product descriptions को clean करना।

53. • IoT readings को custom transformation apply करना।

54. • Mobile numbers को format करना।

55. • Text processing (sentiment, keyword extraction)।

---

56. • बड़े datasets को cache करके multiple queries run करना।

57. • Small lookup tables को broadcast join करना।

58. • Skewed data को repartition करना।

59. • Query optimization for dashboard refresh।

60. • Batch jobs speed improve करना।

---

61. • Machine learning pipeline में intermediate results cache करना।
62. • Repeated transformations cache करना।
63. • Heavy aggregations के बाद result persist करना।
64. • Streaming queries के लिए data cache करना।
65. • Large ETL jobs optimize करना।

---

66. • Output को evenly distribute करना।
67. • Writes को optimized size में करना।
68. • Skewed data को balance करना।
69. • Multiple partitions को reduce करके job speed बढ़ाना।
70. • Aggregation के बाद shuffle minimize करना।

---

71. • Product master table को sales table से join करना।
72. • Currency exchange rates join करना।
73. • Region lookup table join करना।
74. • Small config tables join करना।
75. • ML model parameters join करना।

---

76. Missing file path पर error handle करना।
77. Schema mismatch handle करना।
78. Null values processing के दौरान error catch करना।
79. Transformation failures log करना।
80. Large job failure पर retry logic लगाना।

---

81. • Year-wise parquet files merge करना।
82. • Monthly sales files में नए columns add करना।
83. • IoT sensor readings schema change handle करना।
84. • Slowly changing dimensions read करना।
85. • Cloud data lake schema updates handle करना।

---

86. Daily sales ETL pipeline run करना।
87. IoT readings batch processing schedule करना।
88. Weekly reporting jobs deploy करना।
89. Data cleaning jobs automatic run करना।
90. ML model retraining schedule करना।