**Spark questions:**

1.What is narrow and wide transformation?

2.How to calculate stages in spark? or scenario-based stage calculation (how many stages in wordcount problem).

3.What are the optimizations you have done with spark application or techniques you know?

4.Difference repartition and coalesce?

5.How to decide num of executor and memory, core per executor?

6.What is rdd lineage?

7.What is difference between reduceBy and groupBy?

8.What is difference between sortBy and orderBy?

9.What is data skewness?

10.Arichtecture of spark?

11.What is LEFT SEMI J and LEFT ANTI join?

12.What are the join strategies?

13.Read Json/CSV/parquet and convert in to DF and perform some operation.

14.Analytics function in spark.

1.**explode**

df = sc.parallelize([('a1',1),('b1',2)]).toDF(["cl","value"])

df = spark.createDataFrame([(1, "A", [1,2,3]), (2, "B", [3,5])],["col1", "col2", "col3"]

df2 = df.withColumn('col3',explode('col3'))

df2.show()

+----+----+----+

|col1|col2|col3|

+----+----+----+

| 1| A| 1|

| 1| A| 2|

| 1| A| 3|

| 2| B| 3|

| 2| B| 5|

**2. StructType**

from pyspark.sql.types import StructType,StructField, StringType, IntegerType

schema = StructType([ \

StructField("firstname",StringType(),True), \

StructField("middlename",StringType(),True), \

StructField("lastname",StringType(),True), \

StructField("id", StringType(), True), \

StructField("gender", StringType(), True), \

StructField("salary", IntegerType(), True) \

])

df = spark.createDataFrame(data=data,schema=schema)

3.**skewness**

<https://www.clairvoyant.ai/blog/optimizing-the-skew-in-spark#:~:text=What%20is%20skewed%20Data%3F,is%20not%20properly%2Fevenly%20distributed>.

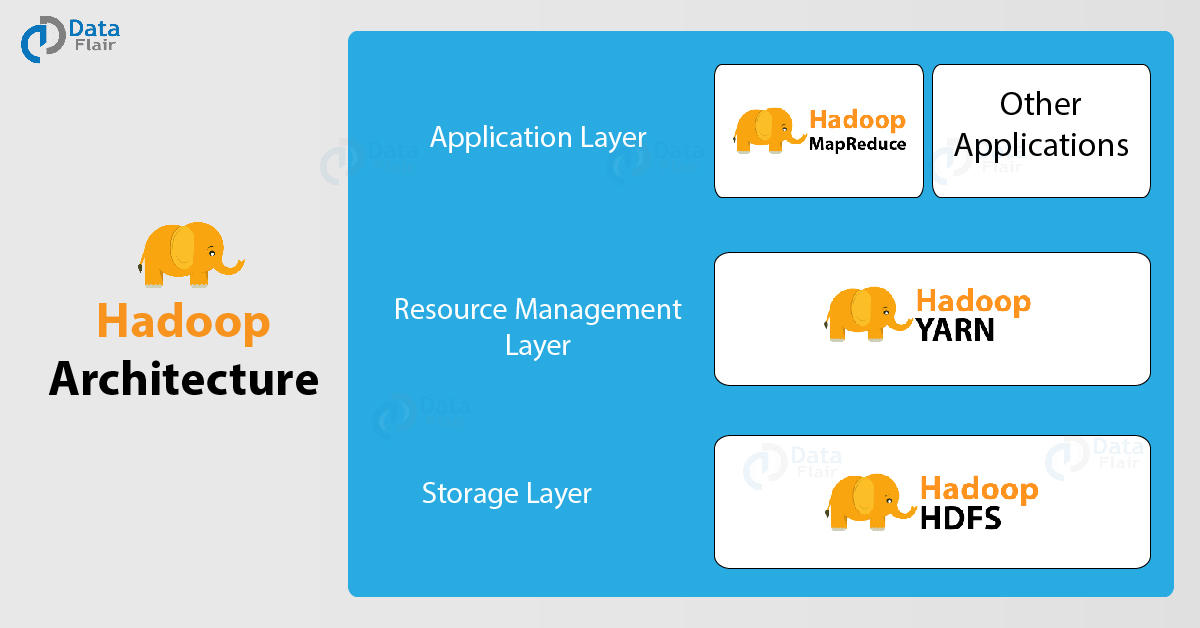
<https://itnext.io/handling-data-skew-in-apache-spark-9f56343e58e8>

4.**joins**

<https://blog.clairvoyantsoft.com/apache-spark-join-strategies-e4ebc7624b06>

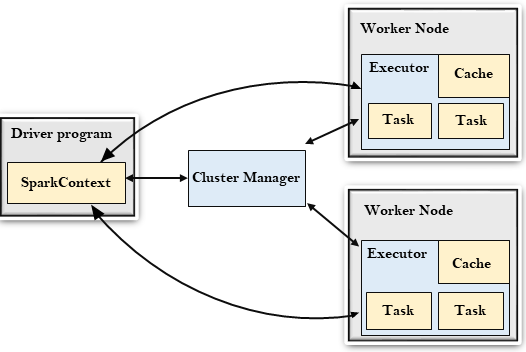
<https://www.educba.com/join-in-spark-sql/>

**5.Hadoop architecture**



**6.Spark Architecture**

[**https://www.javatpoint.com/apache-spark-architecture**](https://www.javatpoint.com/apache-spark-architecture)



**7.import re**

<https://www.guru99.com/python-regular-expressions-complete-tutorial.html#:~:text=match()%20function%20of%20re,it%20returns%20the%20match%20object>.

>>> c\_string = "mongodb://tracking-etl-user:cv45678@localhost:27017/mojo?authSource=admin"

>>> m = re.match("mongodb://(.\*?):(.\*?)@(.\*?):(.\*?)/(.\*)",c\_string)

>>> m[2]

'cv45678'

>>>

**8.sql**

**select concat(name,'(',substring(OCCUPATION,1,1),')') from OCCUPATIONS order by name;**

**select concat('There are a total of ',count(occupation),' ',lower(occupation),'s.') from occupations group by occupation order by count(occupation) ;**

import pyspark

from pyspark.sql import SparkSession

spark = SparkSession.builder \

.master('local[1]') \

.appName('SparkByExamples.com') \

.getOrCreate()

AVRO vs. PARQUET

1. AVRO is a row-based storage format, whereas PARQUET is a columnar-based storage format.
2. PARQUET is much better for analytical querying, i.e., reads and querying are much more efficient than writing.
3. Writing operations in AVRO are better than in PARQUET.
4. AVRO is much matured than PARQUET when it comes to schema evolution. PARQUET only supports schema append, whereas AVRO supports a much-featured schema evolution, i.e., adding or modifying columns.
5. PARQUET is ideal for querying a subset of columns in a multi-column table. AVRO is ideal in the case of ETL operations, where we need to query all the columns.

ORC vs. PARQUET

1. PARQUET is more capable of storing nested data.
2. ORC is more capable of Predicate Pushdown.
3. ORC supports ACID properties.
4. ORC is more compression efficient.