1. Why Spark processing is faster than MapReduce jobs?
2. Spark vs Mapreduce
3. Why Spark was Developed ?
4. What is spark ?
5. What is PySpark?
6. What are the characteristics of PySpark?
7. Feature of Spark and Advantages & Disadvantages of pyspark ?
8. What is Spark Driver ?
9. PySpark Architecture ?
10. PySpark Modules & Packages:
11. Spark Components?
12. What is SparkContext?
13. What is SparkSession Explained ?
14. SparkContext vs SparkSession
15. Repartition() vs Coalesce() ?
16. Difference between Cache and Persist ?
17. What is Unpersist ?
18. What is diffrance between brodcast variable and Accumulator variable ?
19. What is shuffling in spark ?
20. difrance between Groupbykey() vs reduceByKey() vs aggregateByKey() vs sortBy() vs sortByKey()
21. What is RDD ?
22. How to Create RDD ?
23. Types of RDD ?
24. When to use RDDs?
25. What is RDD Operations Transformations and RDD Actions ?
26. map vs flatmap vs filter ?
27. collect vs collectlAslist vs select()
28. Why DF is faster than RDD?
29. RDDs vs. Dataframes vs. Datasets ?
30. Pivot and Unpivot a Spark Data Frame ?
31. What is Spark Schema ?
32. Groupby clause ?
33. What is Spark SQL DataFrame ?
34. Why DataFrame?
35. Is PySpark faster than pandas?
36. What is DAG and lineage graph, RDD lineage?
37. What is paired RDD?
38. What is skewness?
39. How to mitigate skewed data?
40. Optimization Techniques in spark :
41. How to read CSV File Using delimiter ‘,’?
42. What is Star Schema & snowflake Schema Differentiate Star & Snowflake?
43. What is Data Skewness?
44. What is catalyst optimizer?
45. Explain Serialization and Deserialization?
46. What are PySpark serializers?
47. Salting Techniques?
48. Explain MapPartition in Spark?
49. How to track failed Jobs in Spark?

50. What is Broadcast Join?
51. deployment Mode ? Cluster mode and Client Mode
52. Spark Submit Command?
53. Orc vs Parquet vs Csv vs Json
54. Deal with bad data :
55. Why out of memory issue occure ?
56. How to Remove Duplicate Rows ?
57.Create SparkContext & sparkSession
58.How to Create RDD :
59.Create (Read) Spark DataFrame from CSV ,Txt ,JSON,XML

----

1. Why Spark processing is faster than MapReduce jobs?

- **Spark processes data in-memory (RAM) computation .**
- **Hadoop MapReduce has to persist data back to the disk after every Map or Reduce action.**

2. Spark vs Mapreduce

1. **Spark** : is an open-source distributed system for handling Big Data workloads.
- It improves query processing performance on varying data sizes by using efficient query execution and in-memory caching.

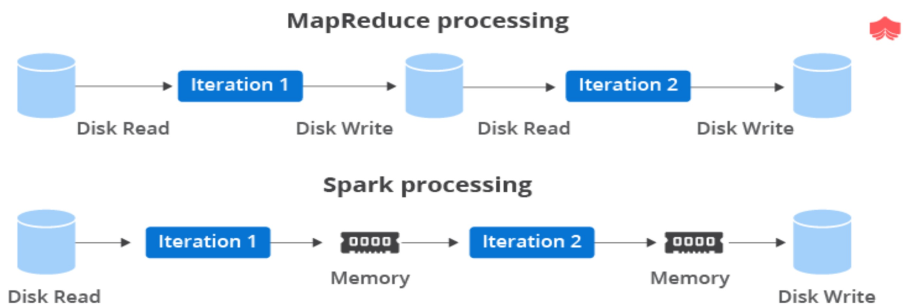
2. **Hadoop MapReduce** : MapReduce is a Java-based distributed computing programming model within the Hadoop framework. **[HDFS : hadoop file system : Mapper Reducer]**
- MapReduce can also be used for dealing with large data sets that don't fit in memory.

Mapper is responsible for sorting all the available data, while **Reducer** is in charge of aggregating it and turning it into smaller chunks

Spark	**Hadoop MapReduce**
In - memory computation . 100x time in memory and 10 times faster on disk than MapRduce. store data in memeory. Spark takes a lot of RAM to operate effectively.	MapReduce reads and writes from disk . store data in disk. MapReduce does not provide data caching
It can process real-time data,batch data.	It can process only batch data.
Apache Spark can schedule all its tasks by itself,	Hadoop MapReduce requires external schedulers like Oozie.
It is data analytics engine.	It is basic data processing engine.
Apache Spark – Scala, Java, Python, R, SQL.	Primarily Java, other languages like C, C++, etc.

3. Why Spark was Developed ?

- The biggest problem in any big data project is to achieve the "Scale".
- The RDBMS databases like Oracle, sql server etc are the oldest approaches of storing and processing the data. But as data grows, they are unable to scale accordingly.
- Stateless machine – read and write to disk before or after each map and reduce stages. This repeated performance of disk I/O took its toll: large MR jobs could run for hours, or even days.
- Only support batch processing – Not good for streaming, Machine Learning or interactive sql like queries.



4. What is spark ?
Spark is a **unified analytics engine mainly designed for large-scale distributed data processing**

5. What is PySpark?
- PySpark is a **Spark library written in Python to run Python applications using Apache Spark** capabilities, using PySpark we can **run applications parallelly on the distributed cluster** (multiple nodes).
- **python + spark = pyspark**
- Spark basically written in Scala.
- later on due to its industry adaptation it's API PySpark released for Python using Py4J.
- **Py4J is a Java library.**
- that is integrated within PySpark and allows python to dynamically interface with JVM objects, hence to run PySpark you also need Java to be installed along with Python, and Apache Spark.

6. What are the characteristics of PySpark?
There are 4 characteristics of PySpark:
1. **Abstracted Nodes:** This means that the individual worker nodes can not be addressed.
2. **Spark API:** PySpark provides APIs for utilizing Spark features.**[high-level APIs in Java, Scala, Python and R]**
3. **Map-Reduce Model:** PySpark is based on Hadoop's Map-Reduce model this means that the programmer provides the map and the reduce functions.
4. **Abstracted Network:** which means that only possible communication is implicit communication.

7. Feature of Spark and Advantages & Disadvantages of pyspark ?
- **Speed:** Spark helps to run an application in Hadoop cluster, up to 100 times faster in memory, and 10 times faster when running on disk.
- **Supports multiple languages:** Spark provides built-in APIs in Java, Scala, or Python.
- **Real-time stream processing:** Spark is designed to handle real-time data streaming.
- **It is flexible:** Spark can run independently in cluster mode, and it can also run on Hadoop YARN, Apache Mesos, Kubernetes, and even in the cloud.
- **Fault Tolerance :** It can recover the failure itself, Self-recovery property in RDD.
- **Lazy Evaluation:** Spark does not evaluate any transformation immediately.execution will not start until an action is triggered.
- **In Memory Computing:** It provides faster execution for iterative jobs. Keeping the data in-memory improves the performance by an order of magnitudes.
- **Cost efficient:** Apache Spark is an open source software, it comes inbuilt for stream processing**.**

Advantages pyspark	Disadvantages pyspark
In-memory, distributed processing engine.	**Slow-** Python is slow as compared to Scala when it comes to performance.

100x faster than traditional systems.	**Hard to express-** PySpark is generally considered hard.
It used to process real-time data using Streaming and Kafka.	Spark takes a lot of RAM to operate effectively.
Open-source community,Supports multiple languages.	it need mid and high level hardware.

8. What is Spark Driver ?
def :

- pyspark works using the master(i.s driver) – slave (i.e. worker) architecture in which the spark system uses a group of machines **known as a cluster**. [**Hadoop YARN,**Standalone, Apache Mesos,Kubernetes **]**
- These machines coordinate with each other across their network to get the work done.
- we need a solo machine that controlled these clusters such a machine is a **Spark driver**.

DRIVER

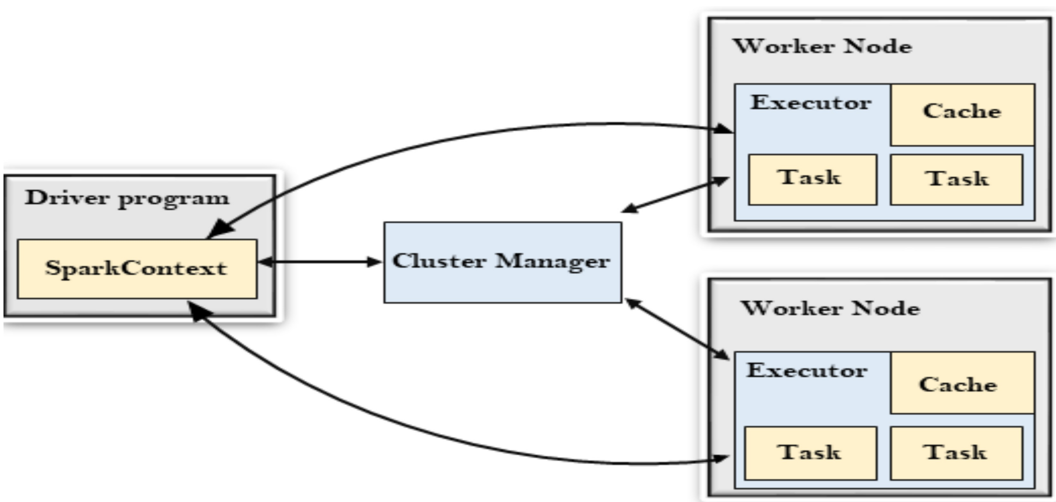
- **Driver is a Java process. This is the process where the main() method are runs Python program. It executes the user code and creates a SparkSession or SparkContext and the SparkSession is responsible to create DataFrame, DataSet, RDD, execute SQL, perform Transformation & Action, etc.**

9. PySpark Architecture ?

- **Apache Spark is a unified analytics engine for large-scale data processing."**
- **It is an in-memory computation processing engine and is processed in parallel.**

- **Apache Spark works in a master-slave architecture where the master is called "Driver" and slaves are called "Workers".**
- **The Spark architecture depends upon two abstractions:**

1. **RDD :** group of data items that can be stored in-memory on worker nodes.
   [RDD : Resilient: Restore the data on failure. **Distributed:** Data is distributed among different nodes. **Dataset:** Group of data **]**
2. **Directed Acyclic Graph (DAG):**It is a finite direct graph that performs a sequence of computations on data.



When you run a Spark application,
Spark Driver creates a context that is an entry point to your application, and all operations (transformations and actions) are executed on worker nodes, and the resources are managed by Cluster Manager.

1. Driver Program :

- The Driver Program is a process that runs the main() function of the application .
- creates the SparkContext or spark session .
- The purpose of SparkContext is to coordinate the spark applications, running as independent sets of processes on a cluster.
- To run on a cluster,the SparkContext connects to a different type of cluster managers.
- **then perform the following tasks: -**

a. It acquires executors on nodes in the cluster.
b. Then, it sends your application code to the executors.
c. application code can be defined by JAR or Python files passed to SparkContext.
d. At last, the SparkContext sends tasks to the executors to run.

2. Cluster Manager:

- The role of the cluster manager is to allocate resources across applications.
- The Spark is capable enough of running on a large number of clusters.
- It consists of various types of cluster managers, such as Hadoop YARN, Apache Mesos and Standalone Scheduler.
- Here, the Standalone Scheduler is a standalone spark cluster manager that facilitates to install Spark on an empty set of machines.

3. Worker Node:

- Worker Node is a slave node,Its role is to run the application code in the cluster.

4. Executor:

- An executor is a process launched for an application on a worker node.
- It runs tasks and keeps data in memory or disk storage across them.
- It read and write data to the external sources.
- Every application contains its executor.

5. Task

- A unit of work that will be sent to one executor.

6. Cluster Manager Types ?

1.Standalone – a simple cluster manager included with Spark that makes it easy to set up a cluster.
2.Apache Mesos – Mesons is a Cluster manager that can also run Hadoop MapReduce and PySpark applications.
3.Hadoop YARN – the resource manager in Hadoop 2. This is mostly used, cluster manager.
4.Kubernetes – an open-source system for automating deployment, scaling, and management of containerized applications.
5.local – which is not really a cluster manager, "local" for master() in order to run Spark on your laptop/computer.

10. PySpark Modules & Packages:

- **PySpark SQL** is the module in Spark. that manages the **structured data** and it **natively supports Python programming language**.
- PySpark provides APIs that support heterogeneous data sources to read the data for processing with

Spark Framework.

- **Packages**: It is simple directory having collection of modules.

1. PySpark RDD (pyspark.RDD)
2. PySpark DataFrame and SQL (pyspark.sql)
3. PySpark Streaming (pyspark.streaming)
4. PySpark MLib (pyspark.ml, pyspark.mllib)
5. PySpark GraphFrames (GraphFrames)
6. PySpark Resource (pyspark.resource) It's new in PySpark 3.0.

11. Spark Components?

1.Spark Core :

- **The Spark Core is the heart of Spark and performs the core functionality.**
- It holds the components for task scheduling, fault recovery and memory management.

2.Spark SQL :

- **It provides support for structured data.**
- It also supports various sources of data like Hive tables, Parquet, and JSON.

3.Spark Streaming:

- it supports scalable and fault-tolerant processing of streaming data.
- It accepts data in mini-batches and performs RDD transformations on that data.

4.MLlib:

- it is a Machine Learning library that contains various machine learning algorithms.
- It is nine times faster than the disk-based implementation used by Apache Mahout.

5.GraphX:

- The GraphX is a library that is used to manipulate graphs and perform graph-parallel computations. It facilitates to create a directed graph.

12. What is SparkContext?

- SparkContext is available since Spark 1.x , it is entry point to PySpark functionality.
- it is used to communicate with the cluster and to create an RDD,accumulator,broadcast variables.

Note: that you can create only one SparkContext per JVM,in order to create another first you need to stop the existing one using stop() method.

- ***Create SparkContext :***

from pyspark import SparkContext sc = SparkContext("local", "Spark") print(sc.appName) Stop sparkcontext: sc.stop()	from pyspark import SparkConf, SparkContext conf = SparkConf( ) conf.setMaster("local").setAppName("Spark") sc = SparkContext.getOrCreate(conf) print(sc.appName)

13.What is SparkSession Explained ?

- SparkSession was introduced in version 2.0.
- It is an entry point to underlying PySpark functionality in order to programmatically create PySpark RDD, DataFrame.
- ***Create SparkSession from builder :***

```
import pyspark
from pyspark.sql import SparkSession
spark = SparkSession.builder.master( "local[1]" ) \
               .appName( ' Spark ' ) \
               .getOrCreate()
```

SparkSession.builder () :

- Return SparkSession.Builder class.This is a builder for SparkSession.
- master(), appName() and getOrCreate() are methods of SparkSession.Builder.

master() – If you are running it on the cluster you need to use your master name as an argument to master().usually,it would be either yarn or mesos depends on your cluster setup.

Use local[x] - when running in Standalone mode. x should be an integer value and should be greater than 0; this represents how many partitions it should create when using RDD, DataFrame, and Dataset.Ideally, x value should be the number of CPU cores you have.

For standalone use - spark://master:7077

appName() – Sets a name to the Spark application that shows in the Spark web UI. If no application name is set, it sets a random name.

getOrCreate() – This returns a SparkSession object if already exists.Creates a new one if not exist.

 # check UI : http://localhost:4040/jobs/

14. SparkContext vs SparkSession

SparkContext	**SparkSession**
SparkContext is available since Spark 1.x	SparkSession was introduced in version 2.0.
SparkContext is an entry point to Spark programming with RDD and to connect to Spark Cluster.	SparkSession has been introduced and became an entry point to start programming with DataFrame and Dataset.

15. Repartition() vs Coalesce() ?

- Spark repartition() and coalesce() are very expensive operations .
- that cause shuffle the data across many partitions hence try to minimize repartition as much as possible.

Repartition()	**Coalesce()**
repartition() is used to increase or decrease the partitions. it suport RDD,DF,Dataset i.e wide transformation.	coalesce() is used to only decrease the number of partitions in RDD. narrow transformation.
Repartition can be used : _when you want your output partitions to be of equally distributed chunks. _perform a shuffle of data and create partitions	No shufling and negligible narrow transformation. This is optimized or improved version of repartition()
rdd1 = rdd.repartition(6) print(rdd1.getNumPartitions())	rdd2 = rdd.coalesce(4) print(rdd2.getNumPartitions())

repartition: df1 = df.repartition(4).withColumn("partition_id",spark_partition_id()) df.rdd.getNumPartitions() -- ---- **repartation column basis :** df1 = df.repartition(4,"DEPARTMENT_ID").withColumn("partition_id",spark_partition_id()) df.rdd.getNumPartitions()
coalesce : df2 = df1.rdd.coalesce(4,**True**).toDF().withColumn("partition_id",spark_partition_id()) df2.rdd.getNumPartitions()

16. Difference between Cache and Persist ?

Spark Cache and persist are optimization techniques.

Both caching and persisting are used to save the Spark RDD, Dataframe, and Dataset's.

Cache()	**Persist()**
RDD cache() method default saves it to memory.	RDD persist() method is used to store it to the user-defined storage level.
Cache() technique we can save intermediate results in memory only. (MEMORY_ONLY).	Persist() technique we can save the intermediate results in 5 storage. (Memory_only, Memory_and_disk, Disk_only , memory_only_ser, Memory_and_disk_ser).
Syntax : cache() : Dataset.this.type	Syntax : 1) persist() : Dataset.this.type 2) persist(newLevel : org.apache.spark.storage.StorageLevel) : Dataset.this.type

Advantages Caching and Persistence :

Cost efficient – Spark computations are very expensive hence reusing the computations are used to save cost.

Time efficient – Reusing the repeated computations saves lots of time.

17. What is Unpersist ?

- **Unpersist :** We can also unpersist the persistence DataFrame or Dataset to remove from the memory or storage.
- Syntax:
- **unpersist() : Dataset.this.type**
- **unpersist(blocking : scala.Boolean) : Dataset.this.type**

18.What is diffrance between brodcast variable and Accumulator variable ?

Brodcast variable	**Accumulator variable**
Broadcast variables are read-only shared variables	Accumulators are write-only shared variables
which can be used to cache a value in memory on all nodes.	add() function is used to add/update a value in accumulator

Broadcast variables used to give every node a copy of a large input dataset in an efficient manner.	accumulator variable is used by multiple workers and returns an accumulated value.
broadcastVar = spark.sc.broadcast([0, 1, 2, 3]) broadcastVar.value o/p : [0, 1, 2, 3]	accum = spark.sparkContext.accumulator(0) rdd = spark.sparkContext.parallelize([1,2,3,4,5]) rdd.foreach(lambda x:accum.add(x)) print(accum.value) o/p : 15

19. What is shuffling in spark ?

- Reallocation of data between multiple Spark stages.
- Shuffling is a mechanism Spark uses to redistribute the data across different executors and even across machines.
- Spark shuffling triggers for **transformation operations like gropByKey(), reducebyKey(), join(), groupBy(), repartition() , cogroup() e.t.c**
- Spark shuffle is a very expensive operation as it moves the data between executors or even between worker nodes in a cluster so try to avoid it when possible.
- **When you have a performance issue on Spark jobs, you should look at the Spark transformations that involve shuffling.**

20. difrance between Groupbykey() vs reduceByKey() vs aggregateByKey() vs sortBy() vs sortByKey()

gropByKey()	**reduceByKey()**	**aggregateByKey()**
both work with PairRDD (key,value) and both returns GroupedData object. both used group the datasets based on the key to perform an aggregation.		
groupByKey() is just to group your dataset based on a key. It will result in data shuffling when RDD is not already partitioned. It is expesive operation.here more shufling so out of memory issues. It is not used combiner . In which data shuffle first then merged So the performance of groupByKey is not efficient.	reduceByKey() is something like aggregation (within the partition) and then grouping which will result in shuffling but very less comapred to groupByKey() It gives better performance when compared to groupByKey. because reduceByKey uses combiner. So before shuffling the data first merged then shuffle.	aggregateByKey() is logically same as reduceByKey() but it lets you return result in different type. In another words, it lets you have a input as type x and aggregate result as type y. For example (1,2),(1,4) as input and (1,"six") as output.

sortBy() :
is used to sort the data by value efficiently in pyspark. It is a method available in rdd. It uses a lambda expression to sort the data based on columns**.**

sortByKey() : is used to sort the values of the key by ascending or descending order.
sortByKey() function operates on pair RDD (key/value pair)

21. What is RDD ?

Resilient Distributed Dataset [RDD : Resilient: Restore the data on failure. **Distributed:** Data is distributed among different nodes. **Dataset:** Group of data **]**

- RDDs is a fault-tolerant collection of elements that can be operated on in parallel.

- RDDS is immutable , fault-tolerant distributed collection of dataset.
- It is work on low level APIs.
- **Each record in RDD is divided into logical partitions, which can be computed on different nodes of the cluster.**

Benefits :

1. **In-Memory Processing :**loads data from disk and process in memory and keeps the data in memory.
2. **Immutability :**RDDs are created you cannot modify.When apply transformations on RDD, PySpark creates a new RDD.
3. **Fault Tolerance :** any RDD operation fails, it automatically reloads the data from other partitions.
4. **Lazy Evolution :**Spark will not start the execution of the process until an ACTION.

Limitations :

1. No Input Optimization Engine
2. RDDs is that the execution process does not start instantly.
3. RDD lacks enough storage memory.
4. The run-time type safety is absent in RDDs.

22.How to Create RDD ?

1. parallelizing an existing collection .
2. referencing a dataset in an external storage system(HDFS,S3 etc)

parallelizing ()	**rdd = sc.parallelize([1,2,3,4,5,6,7,8,9,10]) print(rdd.collect())**
external storage system	rdd2 = spark.sparkContext.textFile("/path/NameFile.txt")

- **Create empty RDD : rdd = spark.sparkContext.emptyRDD**
- **Creating empty RDD with partition : rdd2 = spark.sparkContext.parallelize([ ],10)**
- **Using wholeTextFiles() : rdd3 =spark.sparkContext.wholeTextFiles("/path/textFile.txt")**

23. Types of RDD ?

PairRDDFunctions or PairRDD – Pair RDD is a key-value pair ,ShuffledRDD,DoubleRDD – SequenceFileRDD ,HadoopRDD ,ParallelCollectionRDD

24.When to use RDDs?

- When we want to do low-level transformations on the dataset.
- It does not automatically infer the schema of the ingested data, we need to specify the schema of each and every dataset when we create an RDD.

25.What is RDD Operations Transformations and RDD Actions ?

1. **Transformations :** Transformations are lazy operations.

[It is a function that produces new RDD from the existingSpark will not start the execution of the process until an ACTION]
such as flatMap(), map(), reduceByKey(), filter()

2. **Actions –** operations that trigger computation and return RDD values.

such as collect() , count() , first() , max() etc.

- **Their are two type : Narrow and Wider Transformation**

1.Narrow Transformation :is compute data that live on a single partition.
like map() , flatMap() ,filter()

2.Wider Transformation : is compute data that live on many partitions.
like groupByKey() ,aggregateByKey() ,repartition() etc.

Narrow transformations:	**Wide transformations:**
compute data that live on a single partition.[like map() and filter()]	compute data that live on many partitions.
Narrow transformations transform data without any shuffle involved.	it involve a shuffle of the data between the partitions.
Narrow transformations convert each input partition to only one output partition.	Wide transformations convert input partition to only many output partition.
Functions such as map(), mapPartition(), flatMap(), filter(), union()	groupByKey(), aggregateByKey(), aggregate(), join(), repartition().

26. map vs flatmap vs filter ?

map()	**flatMap()**	**filter()**
Spark map function expresses a one-to-one transformation	Spark flatMap function expresses a one-to-many transformation.	transformation is used to filter the records in an RDD.
used to apply lambda function on every element of RDD/DataFrame. it returns a new RDD .	it is similar to map() it returns a new RDD.	

27.collect vs collectlAslist vs select()

collect()	collectAsList()	select ()
collect() is an action that returns the entire data set in an Array to the driver.	collectAsList() action function is similar to collect() but it returns Java util list.	select() is a transformation that returns a new DataFrame and holds the columns that are selected.
it is used to retrieve the action output when you have very small result set and calling .	it is used to retrieve all the elements of the RDD/DataFrame/Dataset (from all nodes) to the driver node	It used to select the single, multiple, column by the index, all columns from the list and also the nested columns from the DataFrame.

28.Why DF is faster than RDD?

RDD – RDD API is slower to perform simple grouping and aggregation operations**.**

DataFrame – DataFrame API is very easy to use. It is faster for exploratory analysis, creating aggregated statistics on large data sets.

DataSet – In Dataset it is faster to perform aggregation operation on plenty of data sets.

29. RDDs vs. Dataframes vs. Datasets ?

RDD	**DataFrame**	**DataSet**

RDD is a distributed collection of data elements without any schema.	It is also the distributed collection organized into the named columns	It is an extension of Dataframes with more features like type-safety and object-oriented interface.
No in-built optimization engine for RDDs.	It uses a catalyst optimizer for optimization.	It also uses a catalyst optimizer for optimization purposes.
Here, we need to define the schema manually.	It will automatically find out the schema of the dataset.	It will also automatically find out the schema of the dataset by using the SQL Engine.
RDD is slower than both Dataframes and Datasets to perform simple operations like grouping the data.	It provides an easy API to perform aggregation operations. It performs aggregation faster than both RDDs and Datasets	Dataset is faster than RDDs but a bit slower than Dataframes.

30. Pivot and Unpivot a Spark Data Frame ?

- Spark pivot() function is used to pivot/rotate the data from one DataFrame/Dataset column into multiple columns (transform row to column)
- Unpivot is used to transform it back (transform columns to rows).
- Pivot() is an aggregation where one of the grouping columns values transposed into individual columns with distinct data.

31. What is Spark Schema ?

- Spark schema is the structure of the DataFrame or Dataset, we can define it using StructType class which is a collection of StructField that define the column name(String), column type (DataType), nullable column (Boolean) and metadata (MetaData)
- Create Schema using StructType & StructField.
- Create Nested struct Schema
- df.printSchema()

32. Groupby clause ?

groupBy() : it is used to collect group of data. in DF, RDD etc. and perform aggregate functions on the grouped data.
like count(), mean(),max(),min(),sum(),avg(),agg()

33. What is Spark SQL DataFrame ?

- DataFrames are datasets,which is ideally organized into named columns.
- Dataframe is used, for processing of a large amount of structured data.
- contains rows with a schema,It is more powerful than RDD
- There are several features those are common to RDD distributed computing capability, immutability, in-memory, resilient.
- It's API is available on several platforms, such as Scala, Java, Python, and R as well.

34.Why DataFrame?

- dataframe is one step ahead of RDD.
- There is no built-in optimization engine in RDD.RDD cannot handle structured data.
- DF provides memory management and optimized execution plan.
- Dataframes are able to process the data in different sizes, like the size of kilobytes to petabytes on a single node cluster to large cluster.

- **PySpark DataFrame from data sources like TXT, CSV, JSON, ORV, Avro, Parquet, XML formats by reading from HDFS, S3, DBFS, Azure Blob file systems e.t.c.**

35. Is PySpark faster than pandas?

- **yes pandas run operations on a single node whereas PySpark runs on multiple machines.**
- **PySpark processes operations many times faster than pandas**
- PySpark supports parallel execution of statements in a distributed environment.
- i.e on different cores and different machines which are not present in Pandas.

36. What is DAG and lineage graph, RDD lineage?
DAG :

- **Directed Acyclic Graph which has no perticular direction.**
- **It is collection of task with directional dependancy.**

Directed – Means which is directly connected from one node to another. This creates a sequence
Acyclic – Defines that there is no cycle or loop available.
Graph – it is a combination of vertices and edges.

Lineage graph :all the dependencies between the RDDs will be logged in a graph, rather than the actual data. This graph is called the lineage graph.

RDD lineage : RDD are immutable in nature, transformations always create new RDD without updating an existing one hence, this creates an RDD lineage.

37.What is paired RDD?
Pair RDD stores the elements/values in the form of key-value pairs. It will store the key-value pair in the format (key,value).
Distributed key-value pairs are represented as Pair RDDs in Spark.

38.What is skewness?
The state of partitions where data is unevenly distributed.
This is common problem with big data after shuffling.
Key distribution is not uniform (highly skewed), causing some partitions to be very large and not allowing spark to process data in parallel

39.How to mitigate skewed data?
SALTING is a technique that adds random values to the join keys, then spark can partition data evenly.

40.Optimization Techniques in spark :
1.Serialization:

- Pysaprk By default use bytecode serializer
- It can use Kyro serializer as well. It offers 10x faster speed than bytecode serializer
- We need to set serializer properties

2.API Selection:
What dataset is there and what we are using as on the basis of this we need to take care of selection.

i. For RDD:For low level operation,Less optimization technique is available in it

ii. For Dataframe: It is catalyst optimizer ,Low garbage collection (GC) overhead
iii. Dataset:Highly type safe,User tungsten for serializer
iv. Type Safety: It means compiler will validate types while compiling and throw an error if you try to assign a wrong type of variable

3.Advanced Variables (Broadcast variable and accumulator):

- When one dataset is small and the other is very large then we can use broadcast join.
- It broadcasted dataset is available for each partition in the spark so operation will get quickly executed.
- The PySpark Accumulator is a shared variable that is used with RDD and DataFrame to perform sum and counter operations similar to Map-reduce counters.

4.Cache and Persist:

cache() is available always in memory
persist() can be available in memory and disk

5.By Key operations:

- I mean to say I will try to avoid the groupByKey() so that data will not get shuffled too much so that operation cost will be saved.

i. Sometimes out of memory error can occur by such type of operations
ii. I will also try to use mapper to avoid the direct shuffling when there is a necessity to use the groupByKey() operation.

- I will try to use reduceByKey() which aggregates the data and then performs the transformation.

6.File format selection:

- Parquet will provide compression.
- Parquet is having metadata along with actual data.

When you create parquet file you will see metadata file on the same directory along with file.

41. How to read CSV File Using delimiter ‘,’?
df = spark.read.options (delimiter=',') .csv("C:/path/filename.csv")

42. What is Star Schema & snowflake Schema Differentiate Star & Snowflake?

Snowflake schema:
Snowflake Schema is an extension of a Star Schema, and it adds additional dimensions. The dimension tables are normalized which splits data into additional tables.
Star Schema:
In data warehouse, in which the centre of the star can have one fact table and a number of associated dimension tables

Star Schema	**Snowflake Schema:**
In star schema, The fact tables and the dimension tables are contained.	While in snowflake schema, The fact tables, dimension tables as well as sub dimension tables are contained.
Star schema is a top-down model.	While it is a bottom-up model.
It has high data redundancy	While it has low data redundancy.

43. What is Data Skewness?

Skewness is the statistical term, which refers to the value distribution in a given dataset. When we say that there is highly skewed data, it means that some column values have more rows and some column very few i.e the data is not properly/evenly distributed. Data Skewness affects the performance and parallelism in any distributed system.
Solution: we need to divide table into two parts. The first part will contain all the rows that don’t

have a null key and the second part will contain all the data with no null values

44. What is catalyst optimizer?

Spark SQL is one of the newest and most technically involved components of Spark. It powers both SQL queries and the new Data Frame API. At the core of Spark SQL is the catalyst optimizer.
We use Catalyst's general tree transformation framework in four phases:
(1) analyzing a logical plan to resolve references,
(2) logical plan optimization,
(3) physical planning,
(4) code generation to compile parts of the query to Java bytecode.
In the physical planning phase, Catalyst may generate multiple plans and compare them based on cost. All other phases are purely rule-based. Each phase uses different types of tree nodes; Catalyst includes libraries of nodes for expressions, data types, and logical and physical operators. We now describe each of these phases.

45. Explain Serialization and Deserialization?

Serialization refers to converting objects into a stream of bytes and vice-versa (de-serialization) in an optimal way to transfer it over nodes of network or store it in a file/memory buffer.
In distributed systems, data transfer over the network is the most common task. If this is not handled efficiently, you may end up facing numerous problems, like high memory usage, network bottlenecks, and performance issues.
There are 2 types of serialization supported in spark:
1.Java serialization: default serialization, easy but slow.
2.Kryo Serialization - 10 times faster than default serialization.

46. What are PySpark serializers?

- The serialization process is used to conduct performance tuning on Spark.
- The data sent or received over the network to the disk or memory should be persisted.
- PySpark supports serializers for this purpose.
- It supports two types of serializers, they are:

PickleSerializer: (by default)
- This serializes objects using Python's PickleSerializer (class
- pyspark.PickleSerializer).
- This supports almost every Python object.

MarshalSerializer:
- This performs serialization of objects.
- We can use it by using **class pyspark.MarshalSerializer** .
- This serializer is faster than the PickleSerializer but it supports only limited types.

47.Salting Techniques?

The idea here is to divide larger partitions into smaller ones using "salt" (our own created newcolumn) but it also comes with side effect of getting smaller partitions divided into even more smaller ones.
This strategy is more of guaranting execution of all tasks (avoiding OOM's errors) and NOT a uniform duration of each task.

48. Explain MapPartition in Spark?

This is exactly the same as map ();the difference being, Spark mapPartitions() provides a facility to do heavy initializations (for example Database connection) once for each partition instead of doing it on every DataFrame row. This helps the performance of the job when you dealing with heavy-weighted initialization on larger datasets.

49. How to track failed Jobs in Spark?

When a Spark job or application fails, you can use the Spark logs to analyse the failures. The application UI provides links to the logs in the Application UI and Spark Application UI. If you are running the Spark job or application from the Analyse page, you can access the logs via the Application UI and Spark Application UI.

50. What is Broadcast Join?

It is a join operation of a large data frame with a smaller data frame in PySpark Join model.
It reduces the data shuffling by broadcasting the smaller data frame in the nodes of PySpark cluster.
The data is sent and broadcasted to all nodes in the cluster.
This is an optimal and cost-efficient join model that can be used in the PySpark application.

51. deployment Mode ? Cluster mode and Client Mode

Cluster	Client
In Cluster Mode, the Driver & Executor both run inside the Cluster. This is the approach used in Production use cases.	In Client Mode ,Driver is outside of the Cluster. The Executors will be running inside the Cluster.
In cluster mode, the Spark driver runs inside an application master process which is managed by YARN on the cluster. and the client can go away after initiating the application.	client mode only the driver runs locally and all tasks run on cluster worker nodes client mode is majorly used for interactive and debugging purposes.
spark-submit --deploy-mode cluster --driver-memory xxxx	**spark-submit --deploy-mode client --driver-memory xxxx ......**

52. Spark Submit Command?

**./bin/spark-submit **
**--master <master-url> **
**--deploy-mode <deploy-mode> **
python_file_code.py

master: The master URL for the cluster (e.g. spark://23.195.26.187:7077)

53. Orc vs Parquet vs Csv vs Json

CSV	JSON	Parquet	Avro
comma-separated values	key-value pairs, in a partially structured format.	It is a columnar format Only the required columns will be retrieved/read,	It is a row-based format that has a high degree of splitting.

CSV is a row-column based. CSV provides a simple scheme;	JSON consumes more memory due to repeatable column names;	data are stored in columns, they can be highly compressed.	data is stored in binary format, which minimizes file size and maximizes efficiency.
properties of CSV files is that they are only splittable when it is a raw, uncompressed file or when splittable compression format is used such as bzip2 or lzo	JSON supports lists of objects, JSON is not very splittable;	parquet files are binary files that contain metadata about their contents.	support for schema evolution by managing added, missing, and changed fields.
CSV allows you to work with flat data.	Poor support for special characters; JSON is not very splittable;	good choice for heavy workloads	handling large amounts of records as it is easy to add new rows.

Properties	CSV	JSON	Parque
Columnar	✗	✗	✓
Compressable	✓	✓	✓
Splittable	✓*	✓*	✓
Readable	✓	✓	✗
Complex data structure	✗	✓	✓
Schema evolution	✗	✗	✓

@lu

Spark Format Showdown		File Format		
		CSV	JSON	Parquet
Attribute	Columnar	No	No	Yes
	Compressable	Yes	Yes	Yes
	Splittable	Yes*	Yes**	Yes
	Human Readable	Yes	Yes	No
	Nestable	No	Yes	Yes
	Complex Data Structures	No	Yes	Yes
	Default Schema: Named columns	Manual	Automatic (full read)	Automatic (instant
	Default Schema: Data Types	Manual (full read)	Automatic (full read)	Automatic (instant

ORC vs Parquet

Comparison Chart

ORC	Parquet
ORC was inspired from the row columnar format which was developed by Facebook.	Parquet was inspired from the nested data storage format outlined in the Google Dremel paper.
ORC is developed by Hortonworks in collaboration with Facebook.	Parquet is developed and backed by Cloudera, in collaboration with Twitter.
ORC only supports Hive and Pig.	Parquet has a broader range of support for the majority of the projects in the Hadoop ecosystem.
ORC is better optimized for Hive.	Parquet works really well with Apache Spark.
ORC files are organized into stripes of data, which are the basic building blocks for data.	It stores data in pages and each page contains header information, information about definition levels and repetition levels, and the actual data.

DB Difference Between.net

54. Deal with bad data : when df reading or creating with specified schema, if posible that data in file does not match with schema.

1.PREMISSIVE : it allow bad records. and store in specific column .
2.DROPMALFORMD : it dont allow bad records
3.FAILFAST : stop when it meet corrupt records

```
df = spark.read .schema(df_schema).option(columnNameOfCorruptRecord","_corrupt_record") \
      .option("mode","PERMISSIVE") \
      .csv("path\\filename.csv",header = True,inferSchema = True)

df.show()    >>>> show whole data bada + good record in add new column ( u need to add new column with help struct methode)
```

```
df.filter("_corrupt_record" is null).show()      >>> show valid data
df.filter("_corrupt_record is not null ")        >>> get only bad data
df.drop("_corrupt_record  ")    >>> delete column
```

55. Why out of memory issue occure ?

 # OutOfMemory at the Executor Level
formula :
Total executor memory = total RAM per instance / number of executors per instance

 # reasons : out of memory issue occure due to incorrect usage of Spark Inefficient queries ,High concurrency Incorrect configuration.

solution :out of memory issue occure

1. it can occur across cluster driver and excuter memoery

handling tech. driver memory
--driver-memory <XX>G
#or
--conf spark.driver.memory = <XX>G

handling tech. executer memory
--executer -memory <XX>G
#or
--conf spark.executer.memory = <XX>G

#or
spark.driver.maxResultSize

2. coalesce() and repartation
3. brodcast variable ()
4. serilization

56. How to Remove Duplicate Rows ?

Duplicate rows could be remove or drop from Spark SQL DataFrame using distinct() and dropDuplicates() functions,

distinct() can be used to remove rows that have the same values on all columns

dropDuplicates() can be used to remove rows that have the same values on multiple selected columns.

Use distinct() – Remove Duplicate Rows on :

```
# Distinct all columns + count deleted record

DF = dataframe.distinct()
print("Distinct count: ",DF.count())
DF.show()
```

Use dropDuplicate() – Remove Duplicate Rows on DataFrame

```
DF2 = dataframe.dropDuplicates()
print("Distinct count: ",DF2.count())
DF2.show()
```

Create SparkContext : **Create SparkSession from builder :**

from pyspark import SparkContext sc = SparkContext("local", "Spark") print(sc.appName) Stop sparkcontext: sc.stop() from pyspark import SparkConf, SparkContext conf = SparkConf( ) conf.setMaster("local").setAppName("Spark") sc = SparkContext.getOrCreate(conf) print(sc.appName)	from pyspark import SparkContext,SparkConf from pyspark.sql import SparkSession from pyspark.sql.functions import * from pyspark.sql import Window from pyspark.sql.types import * import pyspark from pyspark.sql import SparkSession spark = SparkSession.builder.master("local[1]") \\ .appName(' Spark ') \\ .getOrCreate()

# find partitions :	print("Number of partition : ", rdd.getNumPartitions())

- **How to Create RDD :**

parallelizing ()	rdd = sc.parallelize([1,2,3,4,5,6,7,8,9,10]) print(rdd.collect())
external storage system	rdd2 = spark.sparkContext.textFile("//path//NameFile.txt")
Create empty RDD :with partition :	rdd = spark.sparkContext.emptyRDD rdd2 = spark.sparkContext.parallelize([ ],10)
Using wholeTextFiles() :	rdd3 =spark.sparkContext.wholeTextFiles("/path/textFile.txt")

- **To convert DataSet or DataFrame to RDD just use rdd() method**

```
rdd=df.rdd
print(rdd.collect())
```

- **Convert RDD back to DataFrame by using toDF()**

```
df2 = rdd2.toDF(["name","bonus"])
df2.show()
```

- **How to Create DF :**

Spark Create DataFrame from RDD : using toDF()

```
df = rdd.toDF()
```

Using Spark createDataFrame() : from SparkSession

```
df= spark.createDataFrame(rdd).toDF(columns:_*)
```

Create Spark DataFrame from List and Seq

```
import spark.implicits.
df = data.toDF()

#From Data (USING createDataFrame)
df = spark.createDataFrame(data).toDF(columns:_*)
```

Create (Read) Spark DataFrame from CSV ,Txt ,JSON,XML

```
CSVdf2 = spark.read.csv(r"/path/filename.csv", header = True,inferSchema = True).show()

Txtdf3 = spark.read.text(r"/path/filename.txt").show()

JSONdf4 = spark.read.json(r"/path/filename.json").show()

XMLdf5 = spark.read .format("com.databricks.spark.xml")
                    .option("rowTag", "person")
                     .xml(r "/path/filename . xml")

parqDF = spark.read.parquet(r"/path/filename . parquet").show()
```

withColumn() : it is a transformation function of DataFrame which is used to change the value, convert the datatype of an existing column, create a new column, and many more.

1.Change DataType using PySpark withColumn() : need to use cast() function along with withColumn().

```
df.withColumn("salary",col("salary").cast("Integer")).show()
```

2. Update The Value of an Existing Column

```
df2=df.withColumn("salary", df.salary*3)
df2.show()
```

3. Update Column Based on Condition : **updates gender column with value Male for M, Female for F**

```
from pyspark.sql.functions import when
df3 = df.withColumn("gender", when(df.gender == "M","Male") \
      .when(df.gender == "F","Female") \
      .otherwise(df.gender))
df3.show()
```

4.Update DataFrame Column Data Type : updates salary column to String type.

```
df4=df.withColumn("salary",df.salary.cast("String"))
df4.printSchema()
root
 |-- firstname: string (nullable = true)
 |-- lastname: string (nullable = true)
 |-- gender: string (nullable = true)
 |-- salary: string (nullable = true)
```

5. **withColumnRenamed to Rename Column on DataFrame**

1. withColumnRenamed() to rename a DataFrame column, we often need to rename one column or multiple (or all) columns on PySpark DataFrame

- **PySpark withColumnRenamed() Syntax:**

```
withColumnRenamed(existingName, newNam)
```

- existingName – The existing column name you want to change
- newName – New name of the column
- Returns a new DataFrame with a column renamed.

ex :

```
df.withColumnRenamed("dob","DateOfBirth").printSchema()
```

2. PySpark withColumnRenamed – To rename multiple columns

ex:

```
df2 = df.withColumnRenamed("dob","DateOfBirth") \
   .withColumnRenamed("salary","salary_amount")
df2.printSchema()
```

3.**Using PySpark DataFrame withColumn – To rename nested columns**

```
from pyspark.sql.functions import *
df4 = df.withColumn("fname",col("name.firstname")) \
         .withColumn("mname",col("name.middlename")) \
         .withColumn("lname",col("name.lastname")) \
         .drop("name")
df4.printSchema()
```

4.How to Filter Rows with NULL Values ?

- filter rows with NULL/None values on columns we can checking IS NULL or IS NOT NULL conditions.
- using filter() or where() functions of DataFrame we can filter rows with NULL values by checking isNULL()

1. Filter Rows with NULL Values in DataFrame :[state is column name]

```
df.filter("state is NULL").show()

df.filter("state is NULL").show()

df.filter(col("state").isNull()).show()
```

2. Filter Rows with NULL on Multiple Columns

```
df.filter("state IS NULL AND gender IS NULL").show()

df.filter(df.state.isNull() & df.gender.isNull()).show()
```

3. Filter Rows with IS NOT NULL or isNotNull

isNotNull() is used to filter rows that are NOT NULL in DataFrame columns.

```
from pyspark.sql.functions import col

df.filter("state IS NOT NULL").show()

df.filter("NOT state IS NULL").show()

df.filter(df.state.isNotNull()).show()

df.filter(col("state").isNotNull()).show()
```

4. PySpark SQL Filter Rows with NULL Values :

```
df.createOrReplaceTempView("DATA")

spark.sql("SELECT * FROM DATA where STATE IS NULL").show()

spark.sql("SELECT * FROM DATA where STATE IS NULL AND GENDER IS NULL").show()

spark.sql("SELECT * FROM DATA where STATE IS NOT NULL").show()
```

5.Count of Not null in DataFrame single column

```
from pyspark.sql.functions import col

print(df.filter(col("FIRST_NAME").isNotNull()).count())
```

6. Count of Not null in DataFrame muiltiple columns

```
from pyspark.sql.functions import col, when, count

df.select([count(when(col(c).isNotNull() , c)).alias(c) for c in df.columns]).show()
```

4.Filter startsWith(), endsWith() :

- startsWith() – Returns Boolean value true when DataFrame column value starts with a string specified as an argument to this method, when not match returns false.
- endsWith() – Returns Boolean True when DataFrame column value ends with a string specified as an argument to this method, when not match returns false.

```
import org.apache.spark.sql.functions.col
df.filter(col("name").startsWith("James")).show()
+---+-----------+
| id|       name|
+---+-----------+
|  1|James Smith|
+---+-----------+


df.filter(! col("name").startsWith("James")).show()
df.filter( col("name").startsWith("James") === false).show()
+---+---------------+
| id|           name|
+---+---------------+
|  2|   Michael Rose|
|  3|Robert Williams|
|  4|     Rames Rose|
|  5|     Rames rose|
+---+---------------+
```

Spark Filter endsWith() :

```
df.filter(col("name").endsWith("Rose")).show()
+---+------------+
| id|        name|
+---+------------+
|  2|Michael Rose|
|  4|  Rames Rose|
+---+------------+

//NOT ends with a string :

df.filter(! col("name").endsWith("Rose")).show()
df.filter(col("name").endsWith("Rose")==false).show()
+---+---------------+
| id|           name|
+---+---------------+
|  1|    James Smith|
|  3|Robert Williams|
|  5|     Rames rose|
+---+---------------+
```

Using Spark SQL Expression :

```
df.createOrReplaceTempView("DATA")

//Starts with a String
spark.sql("select * from DATA where name like 'James%'").show()

//NOT starts with a String
spark.sql("select * from DATA where name not like 'James%'").show()
```

Drop column one or multiple : using drop()

```
# df2 = df.drop("FIRST_NAME")
# df2.printSchema()

df.drop("FIRST_NAME","LAST_NAME","EMAIL").printSchema()
```

- **transform() Function : Available since Spark 3.0**

Syntax
DataFrame.transform(func: Callable[[…], DataFrame], *args: Any, **kwargs: Any) → pyspark.sql.dataframe.DataFrame

- func – Custom function to call.
- *args – Arguments to pass to func.
- *kwargs – Keyword arguments to pass to func.

Create Custom Functions :

```
# Custom transformation 1 :
from pyspark.sql.functions import upper

def to_upper_str_columns(DF):
   return DF.withColumn("name",upper(DF.name))

ex: df2 = empDF.transform(to_upper_str_columns).show()

# Custom transformation 2 :
def reduce_price(df,reduceBy):
   return df.withColumn("newsalary",df.salary - reduceBy)
ex: empDF. .transform(reduce_price,(1000)).show()

# PySpark transform() Usage
df2 = empDF.transform(to_upper_str_columns) \
      .transform(reduce_price,(1000)).show()

# Custom transformation 3 :
def apply_discount(df):
   return df.withColumn("discounted_fee",  \
          df.new_fee - (df.new_fee * df.discount) / 100)

ex : df .transform(apply_discount).show()
```

1. Get number of rows and columns of PySpark dataframe

1. df.count(): This function is used to extract number of rows from the Dataframe.
2. df.distinct().count(): This functions is used to extract distinct number rows which are not duplicate/repeating in the Dataframe.
3. len(df.columns): This function is used to count number of items present in the list.

DataFrame : repartition vs coalesce :

find data frame partation : (128 mb data by default) : df.rdd.getNumPartitions()

from pyspark.sql.function
import spark_partition_id

repartition:
df1 = df.repartition(4).withColumn("partition_id",spark_partition_id())
df.rdd.getNumPartitions()

repartation column basis :
df1 = df.repartition(4,"DEPARTMENT_ID").withColumn("partition_id",spark_partition_id())
df.rdd.getNumPartitions()

coalesce :
df2 = df1.rdd.coalesce(4,**True**).toDF().withColumn("partition_id",spark_partition_id())
df2.rdd.getNumPartitions()

df1.rdd.glom().collect()