Live Interview Questions:

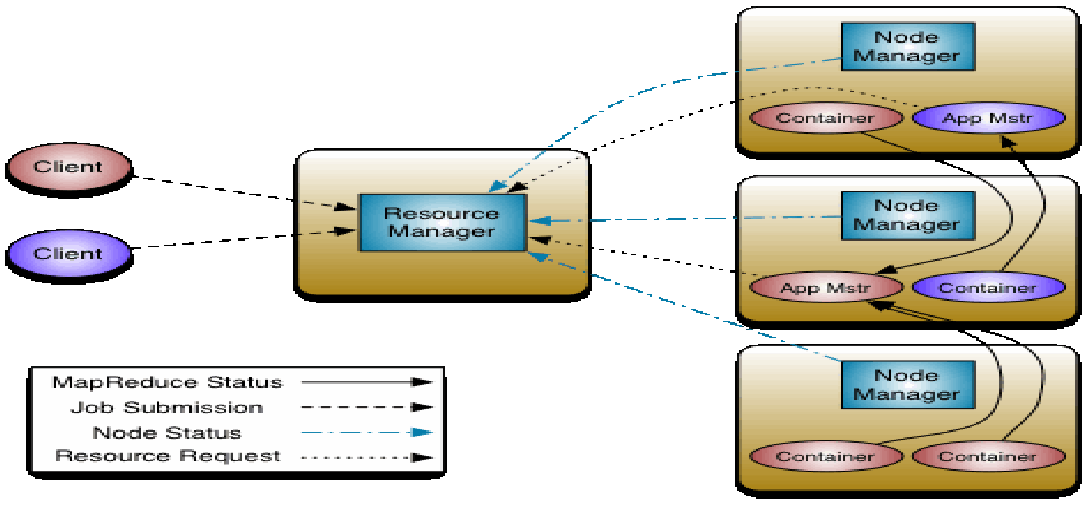
1. ***What is MapReduce?***

MapReduce is a programming framework that allows us to process a huge amount of data in a parallel distributed system. It works on Key, Value pairs (k1, v1). MapReduce consists of two distinct tasks — Map and Reduce. The reducer phase takes place after the mapper phase has been completed.
There are two main components of MapReduce

- **JobTracker**- It is the master that creates and runs the job in MapReduce. It runs on the name node and allocates the job to TaskTrackers. The JobTracker first receives the request from the client and talks to the name node to determine the location of the data. Then, it finds the best TaskTracker node to execute the tasks based on the data locality. JobTracker also monitors the individual TaskTrackers and submits the overall status of the job to the client.
- **TaskTracker** - It is the slave that runs on the data node. TaskTracker runs the job sent by the JobTracker and reports the status of the task back to it. The TaskTracker will be assigned with the Mapper and Reducer tasks to execute the job.

2. ***What is YARN?***

YARN (Yet Another Resource Negotiator) is a resource management and job scheduling technology in the Hadoop distributed processing framework. It allows the data stored in HDFS to be processed and run by various data processing engines such as batch processing, stream processing, interactive processing, graph processing.



3. ***Why are industries moving to scala rather than python or java?***

Answer: Scala is a functional language as well as it is object oriented programming language. Another reason is mostly companies are using spark because of it's faster computation. Spark is written in Scala and that's why it is more convenient to use scala rather than other languages.

4. ***What is spark and why spark?***

Spark is a parallel processing framework for running large-scale data analytics applications across clustered computers. It can handle both batch and real-time analytics and data processing workloads..
Why Spark: We are using spark because:

a. It is in-memory computation
b. Fast processing
c. Real-time stream processing

d. Lazy evaluation
e. Support multiple languages like java, scala, python…
f. Fault tolerance
g. Cost efficient

5. ***What is the optimization technique you used in spark?***
   - Data Serialization. Here, an in-memory object is converted into another format that can be stored in a file or sent over a network. ...
   - Cache and persist
   - Data Structure Tuning. ...
   - Garbage collection optimization.
   - Memory Management
   - Broadcast join/ filtering

6. ***What is Broadcast Join in spark?***

Broadcast join is an important part of Spark SQL's execution engine. It is very efficient for joins between a large dataset with a small dataset. It can avoid sending all data of the large table over the network. To use this feature we can use broadcast function or broadcast hint to mark a dataset to broadcast when used in a join query.
Example: Let’s think we have 2 dataframes named people and cities .Let's pretend that people dataframe is bigger and cities dataframe is smaller.

```
val peopleDF = Seq(
  ("andrea", "medellin"),
  ("rodolfo", "medellin"),
  ("abdul", "bangalore")
).toDF("first_name", "city")
peopleDF.show()

val citiesDF = Seq(
  ("medellin", "colombia", 2.5),
  ("bangalore", "india", 12.3)
).toDF("city", "country", "population")
citiesDF.show()

Let’s broadcast the citiesDF and join it with the peopleDF.
peopleDF.join(
  broadcast(citiesDF),
  peopleDF("city") <=> citiesDF("city")
).show()
```

Note: DataFrames up to 2-8GB can be broadcasted

7. **What is Partition in Spark?**

Partition basically is a logical chunk of a large distributed data set. It provides the possibility to distribute the work across the cluster, divide the task into smaller parts, and reduce memory requirements for each node. Partition is the main unit of parallelism in Apache Spark.

How do I choose a partition in spark?
The best way to decide on the number of partitions in an RDD is to make the number of partitions equal to the number of cores in the cluster so that all the partitions will process in parallel and the resources will be utilized in an optimal way.
**** We should have 2-3 times more partitions than the total number of cores.***

8. ***What is lazy evaluation and why spark is lazy?***

Mainly two things happen in spark that is Transformation and Action. Lazy evaluation means the execution will not start until an action is triggered. Transformations are lazy in nature means when we call some operation on RDD, it does not execute immediately. Spark adds them to a DAG of computation and only when the driver requests some data, this DAG actually gets executed.
Advantages of lazy evaluation.
1) It is an optimization technique that means it provides optimization by reducing the number of queries.
2) It saves the round trips between driver and cluster, thus speeds up the process.

9. ***What is the difference between Beeline and Hive CLI?***

The main difference between Beeline and Hive CLI is how the clients connect to Hive. The Hive CLI connects directly to the Hive Driver and requires that Hive be installed on the same machine as the client. On the other hand Beeline connects to HiveServer2 and does not require the installation of Hive libraries on the same machine as the client. Beeline is a thin client that also uses the Hive JDBC driver but it executes queries through HiveServer2, which allows multiple concurrent client connections and supports authentication.

10. ***What is the optimization technique you used in hive?***
    - Execution Engine.
    - Usage of suitable file format.
    - By partitioning.
    - Use of bucketing.
    - Use of vectorization.
    - Cost based optimization.
    - Use of indexing (to improve the speed of query lookup on certain columns of a table)

11. ***Can you give me an example where we can use the partitioning and where we can use bucketing?***

Partitioning means the way of dividing a table into related parts based on the values of partitioned columns such as date, city, and department. Using partition, it is easy to query a portion of the data. In the Hive Partition, each partition will be created as a directory. Let's think of US census data, we can partition it by states like NY.
When we subdivided the partition into more manageable parts, this is called Bucket. In Hive Buckets, each bucket will be created as a file. In NY state we can bucket people's name columns into the number of equal buckets.

```
CREATE TABLE mytable (
name string, city string, employee_id int )
PARTITIONED BY (year STRING, month STRING, day STRING)
CLUSTERED BY (employee_id) INTO 256 BUCKETS
```

12. ***What is Cache and Persist? When you use cache and when you use persistence?***

Using cache() and persist() methods, Spark provides us an optimization mechanism to store the intermediate result of an RDD so they can be reused in subsequent actions. When we persist or cache an RDD, each worker node stores it's partitioned data in memory or disk and reuses them in other actions on that RDD. And Spark's persisted data on nodes are fault-tolerant meaning if any partition is lost, it will automatically be recomputed using the original transformations that created it. It reduces the computation overhead.
`The difference between cache() and persist() is that using cache() the default storage level is MEMORY_ONLY while using persist() we can use various storage levels.` Using Cache technique we can save intermediate results in memory only when needed but in Persist() we can save the intermediate results in 5 storage levels(MEMORY_ONLY, MEMORY_AND_DISK, MEMORY_ONLY_SER, MEMORY_AND_DISK_SER, DISK_ONLY).

13. How much data do you process everyday?
It depends on each project (40 - 60 GB of data everyday).

14. Which distribution system do you use in your project? What is the size of the cluster?
We use the cloudera distribution system in our project. Cluster size depends on each project.
In Dev:
Cluster size : 25 slaves and 3 masters
Hardware Configuration:
Each slave : 16 GB RAM , 2 TB HDD, quad processor
Each master : 24 GB RAM , 2 TB HDD, dual quad processor

In Prod:
cluster size = 151 slaves + 4 masters = 155 nodes
Hardware configuration:
Each slave : 24 GB RAM , 2 TB HDD , quad processor
Each master : 48 GB RAM , 2 TB HDD , dual quad processor

15. Spark parameter configuration
****Cluster Config:****
10 Nodes
16 cores per Node
64GB RAM per Node
Normally We assign 5 core per executors
--executor-cores = 5 (for good HDFS throughput)

- We will leave 1 core per node for Hadoop/Yarn daemons = Num cores available per node = 16-1 = 15
- So, Total available of cores in cluster = 15 x 10 = 150
- Number of available executors = (total cores / num-cores-per-executor) = 150/5 = 30
- We have to leave 1 executor for ApplicationManager = 30 - 1 = 29

--num-executors = 29

- Number of executors per node = Number of available executors/number of nodes=30/10 =3
- Memory per executor = memory per node / num of executors per node =64GB/3 = 21GB
- Counting off heap overhead = 7% of 21GB = 2GB. So, actual

--executor-memory = 21 - 2 = 19GB
So, recommended config is: 29 executors, 19GB memory each and 5 cores each!!

16. What is the RDD and why are they immutable?

RDD is an immutable, fault tolerant distributed collection of objects, which is partitioned across nodes in a cluster that can be operated in parallel with a low-level API.

Spark RDD is an immutable collection of objects for the following reasons:

- Immutable data can be shared safely across various processes and threads
- It allows us to easily recreate the RDD
- We can enhance the computation process by caching RDD

17. Can you describe two transformations which create a new stage?

Group by key and reduce by key

Difference between Reducebykey and Groupbykey:

Both keys are wide transformations. When we use ReduceByKey on a RDD the data which is going to shuffle from one executor to another executor, data is combined first then sent to the next executor. If we are using GroupByKey the data is not combined and sent as is from one executor to another executor.

ReduceByKey: Data is combined so that at each partition there should be at least one value for each key. And then shuffle happens and it is sent over the network to some particular executor for some action such as reduce.

Use of `reduceByKey`:

- `reduceByKey` can be used when we run on a large data set.
- `reduceByKey` when the input and output value types are of same type over `aggregateByKey`
- ReduceBykey takes 1 parameter only which is a function for merging.

GroupByKey: It doesn't merge the values for the key but directly the shuffle process happens and here a lot of data gets sent to each partition, almost the same as the initial data. And the merging of values for each key is done after the shuffle. Here a lot of data is stored on the final worker node so resulting in out of memory issues. GroupByKey takes no parameter and groups everything. Also, it is an overhead for data transfer across partitions.

18. What transformation process do you use in spark?

Spark Transformation is a function that produces new RDD from the existing RDDs

There are two types of transformations:

a. Narrow Transformation

Narrow transformation – In Narrow transformation, all the elements/data required to compute for the single partition of new RDD, and are available in the single partition of parent RDD. (i.e. map, flatmap, union, mappartion, filter, sample)

b. Wide Transformation

Wide transformation – In wide transformation, all the elements/data required to compute for a single partition of a new RDD may live in many partitions of parent RDD. (i.e. Intersection, Distinct, GroupBykey, ReduceBykey, join, Cartesian, Repartition, Coalesce)

19. What is the difference between Reduce and ReduceByKey?

Basically, Reduce is an action which Aggregates the elements of the dataset using a function func (which takes two arguments and returns one), also we can use reduce for single RDDs. On the other hand ReduceByKey is a transformation and it is one value for **each** key. When called on a dataset of (K, V) pairs, returns a dataset of (K, V) pairs where the values for each key are aggregated using the given reduce function func, which must be of type (V,V) => V.

20. There are two types of file format, row base and column base, when to use those?

Avro is the row base file format and Parquet is the column base file format.
Avro is used for principally two things

- Data serialization
- RPC (Remote procedure call) protocol.

When we should use avro:

- When we need to store a large set of data on disk and to conserve space.
- We would be able to read the data from disk with applications written in other languages besides java or the JVM.
- We would be able to transfer data across a remote system and we don't want the overhead of java serialization.
- Because avro produces a smaller binary output compared to java serialization, we also get a better remote data transfer throughput using avro for RPC.
- We want our data to be splittable on read using hadoop InputFormat classes.
- We still get the splitability while rightfully using data compression.
- We want a data container technology that will not require us to keep the data schema separately from the data itself.

When we should use Parquet:
Parquet is a columnar format supported by many data processing systems. The benefits of having a columnar storage are –

- Columnar storage limits IO(input/output) operations.
- Columnar storage can fetch specific columns that you need to access.
- Columnar storage consumes less space.
- Columnar storage gives better-summarized data and follows type-specific encoding.

21. What is the difference between an ORC and Parquet file?

ORC files are made of stripes of data where each stripe contains index, row data, and footer (where key statistics such as count, max, min, and sum of each column are conveniently cached). ... Parquet files consist of row groups, header, and footer, and in each row group data in the same columns are stored together

22. Which file format is better orc or parquet?

ORC indexes are used only for the selection of stripes and row groups and not for answering queries. AVRO is a row-based storage format whereas PARQUET is a columnar based storage format. PARQUET is much better for analytical querying i.e. reads and querying are much more efficient than writing.

23. What is the difference between Avro and Parquet file?

The biggest difference between ORC, Avro, and Parquet is how they store the data. Parquet and ORC both store data in columns, while Avro stores data in a row-based format. ... While column-oriented stores like Parquet and ORC excel in some cases, in others a row-based storage mechanism like Avro might be the better choice.
Avro

- Widely used as a serialization platform
- Row-based, offers a compact and fast binary format
- Schema is encoded on the file so the data can be untagged
- Files support block compression and are splittable
- Supports schema evolution

Parquet
- Column-oriented binary file format
- Uses the record shredding and assembly algorithm described in the Dremel paper
- Each data file contains the values for a set of rows
- Efficient in terms of disk I/O when specific columns need to be queried

24. Which file format works best with spark and why?

Spark does not have any default file format but Parquet file format is the best suitable in spark.
WHY: Parquet is a columnar file format that provides optimizations to speed up queries and is a far more efficient file format than CSV or JSON, supported by many data processing systems. Spark SQL provides support for both reading and writing parquet files that automatically capture the schema of the original data

25. How to Optimize or improve Sqoop import?
- split-by and boundary-query
- direct
- Num-mapper
- fetch-size

26. For sqoop export you transfer some partial data and then stop, in that case what do you do?

In this case we can use Incremental append mode.

Use Sqoop incremental append mode to import only new records
- Based on value of last record in specified column

```
sqoop import --table (table_name) \
--connect jdbc:mysql://localhost:/database_name \
--username (root) \
--password (cloudera) \
--incremental append \
--check-column column_name \
--last-value 1000
```

Sqoop incremental lastmodified mode also imports new and modified records
- Based on a timestamp in a specified column
- We must ensure timestamps are updated when records are added or changed in the database

```
sqoop import --table (table_name) \
--connect jdbc:mysql://localhost:/database_name \
--username (root) \
--password (cloudera) \
--incremental lastmodified \
--check-column column_name \
--last-value '2018-12-25 06: 00: 00'
```

27. What is the difference between --split-by and --boundary-query in SQOOP?

--split-by:

I would say --split-by is mostly being used when we have a table that hasn't got a primary key, sqoop will normally spit out an error message if the table has no primary key. so….--split-by is being used to determine another column to be used to compute the min() & max in the absence of a primary key. Some requirements are:

- Table should have numeric values
- Table should not contain null

--boundary-query:

The boundary query is used for splitting the value according to id_no of the database table.

- To boundary query, we can take a minimum value and maximum value to split the value.
- To make a split using boundary queries, we need to know all the values in the table.
- To import data from the database to HDFS using boundary queries.

Example--boundary-query:
"SELECT min(id_value), max(id_value) from table_name"

28. What is the difference between Sqoop and Spark SQL?

Sqoop: uses map reduces to ingest data, that is it performs all the processing on the disk of the nodes available in the cluster, this is time consuming and would also cause performance issues and down stream might always not get the latest and greatest data.

Spark: SparkSql can be used which is far more efficient in terms of performance, processing time as all the computations are done in the memory which is 10x faster then computing on disk.

29. What is the difference between Join and Union?

Difference between JOIN and UNION in SQL :

JOIN	UNION
JOIN combines data from many tables based on a matched condition between them.	SQL combines the result-set of two or more SELECT statements.
It combines data into new columns.	It combines data into new rows
Number of columns selected from each table may not be the same.	Number of columns selected from each table should be the same.
Datatypes of corresponding columns selected from each table can be different.	Datatypes of corresponding columns selected from each table should be the same.
It may not return distinct columns.	It returns distinct rows.

30. What is the difference between flume and sqoop?

Sqoop:

- Sqoop can perform import/export from RDBMS to HDFS/HIVE/HBASE
- sqoop only import/export structured data not unstructured or semi structured.

Flume:

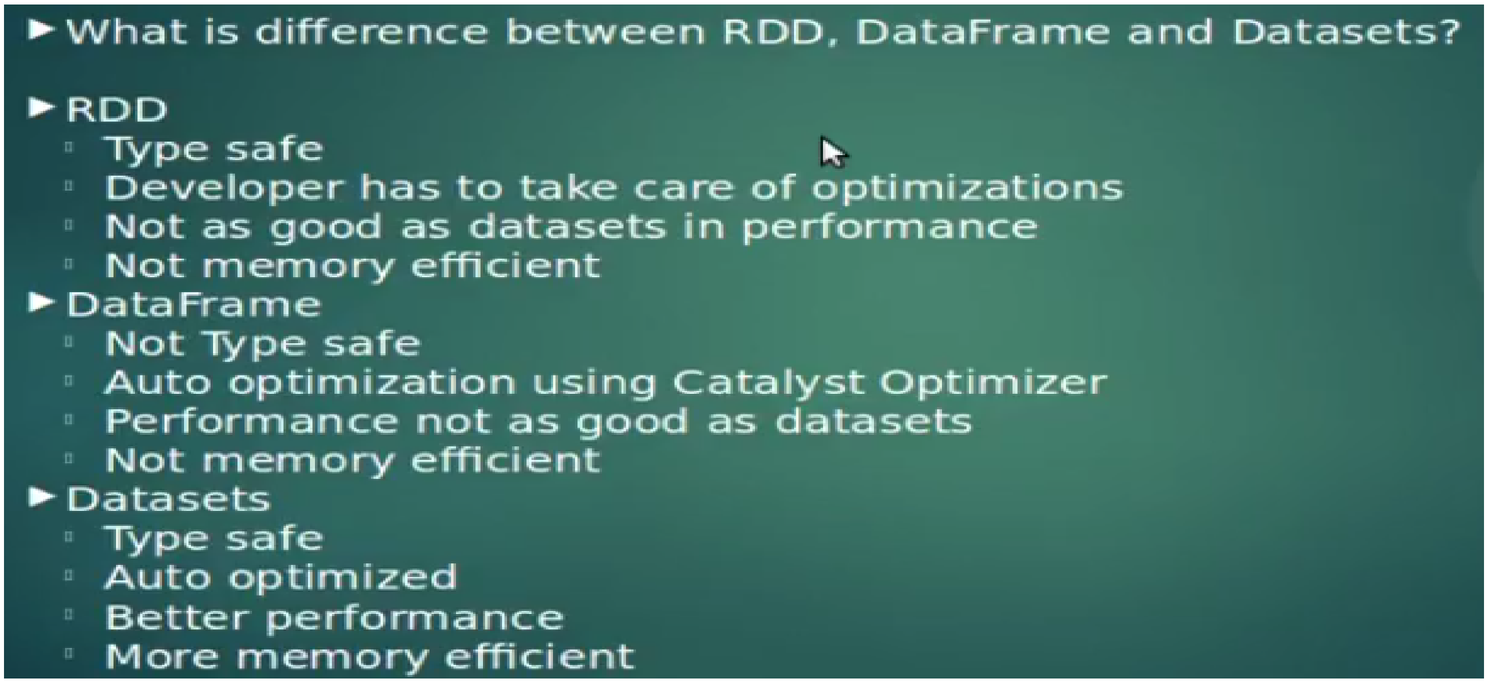
- import stream data from multiple sources mostly semi-structured and unstructured in nature. Now Kafka is a better alternative for flume.

31. What is the difference between Var and Val in scala?
val defines a constant, a fixed value which cannot be modified once declared and assigned while **var** defines a variable, which can be modified or reassigned. So val is immutable and var (variable) is mutable.

32. What is the difference between RDD and DataFrame?

DataFrame is a distributed collection of data organized into named columns.Basically Data Frame is a table, or a two-dimensional array-like structure, in which each column contains measurements on one variable, and each row contains one case. It is a high level API.	**Resilient Distributed Dataset**(RDD) is an immutable, fault tolerant distributed collection of objects, which is partitioned across the nodes in a cluster that can be operated in parallel with a low-level API
DataFrame has additional metadata due to its tabular format, which allows Spark to run certain optimizations on the finalized query.	Each dataset in RDD is divided into logical partitions, which may be computed on different nodes of the cluster.

- My opinion is to use a DataFrame where possible due to the built-in query optimization.


What is difference between RDD, DataFrame and Datasets?

- RDD
  - Type safe
  - Developer has to take care of optimizations
  - Not as good as datasets in performance
  - Not memory efficient
- DataFrame
  - Not Type safe
  - Auto optimization using Catalyst Optimizer
  - Performance not as good as datasets
  - Not memory efficient
- Datasets
  - Type safe
  - Auto optimized
  - Better performance
  - More memory efficient

33. What is SparkSQL? What is the difference between DataFrame and SparkSQL?
Spark SQL is a Spark module for structured data processing. It provides a programming abstraction called DataFrames and can also act as a distributed SQL query engine. It enables unmodified Hadoop Hive queries to run up to 100x faster on existing deployments and data.

The main difference between DataFrame and Spark SQL is:

In our Spark SQL string queries, we won't know a syntax error until runtime (which could be costly), on the other hand, DataFrames syntax errors can be caught at compile time.

34. How To Read CSV File Using PySpark?

```
from pyspark.sql import SparkSession
spark = SparkSession \       ## (first initialize our spark session)##
    .builder \
    .appName("how to read csv file") \
    .getOrCreate()
df = spark.read.csv('file path', header=True) ## read CSV file ##
df.show()
df = df.withColumnRenamed('A1','B1') ## if we want to change the column name)
```

Load CSV file:

```
df=spark.read.format("path").option("header","true").load(path)
df = spark.read.option("header","true").option("inferSchema","true").csv("hdfs://localhost
:9000/airports.csv")
df.show()
```

35. What is AWS GLUE, S3, Redshift and Lambda?

GLUE: AWS Glue is a fully managed extract, transform, and load (ETL) service that we can use to catalog our data, clean it, enrich it, and move it reliably between data stores. With AWS Glue, we can significantly reduce the cost, complexity, and time spent creating ETL jobs. AWS Glue is serverless, so there is no infrastructure to set up or manage.

S3: Amazon S3 (Simple Storage Service) is a scalable, high-speed, low-cost web-based cloud storage service designed for online backup and archiving of data and application programs. It allows users to upload, store, and download any type of files up to 5 TB in size.

RedShift: Amazon Redshift is a fast, simple, cost-effective data warehousing service. It offers Significant Query Speed Upgrades and gives us Robust Security Tools.

Exceptionally fast. Redshift is very fast when it comes to loading data and querying it for analytical and reporting purposes. ...
High Performance. ...
Horizontally Scalable. ...
Massive Storage capacity. ...
Attractive and transparent pricing. ...
SQL interface. ...
AWS ecosystem. ...
Security.

AWS Lambda: AWS Lambda is a compute service that lets us run code without provisioning or managing servers. AWS Lambda executes your code only when needed and scales automatically, from a few requests per day to thousands per second.

Using AWS glue I was doing 2 things:

- Move data from MySQL and Oracle to AWS Redshift.

JDBC connection:

```
jdbc:protocol://host:port/db_name
jdbc:mysql://xxx-cluster.cluster-xxx.us-east-1.rds.amazonaws.com:3306/employee
jdbc:oracle:thin://@xxx-cluster.cluster-xxx.us-east-
1.rds.amazonaws.com:1521/employee
```

- Convert JSON files to AVRO, CSV and PARQUET using spark, then move the files from S3 bucket to AWS Redshift.

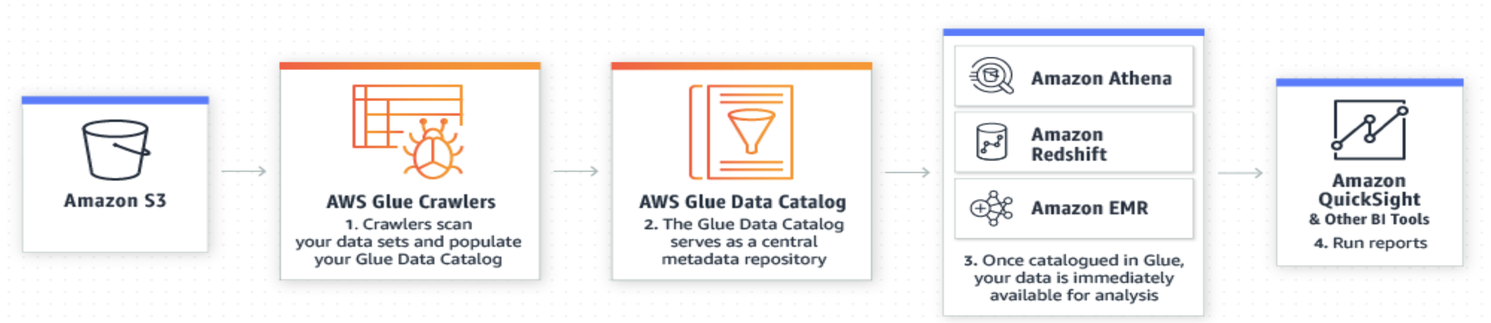
```
//read json file into dataframe
  val df = spark.read.json("file path.json")## in Python:df = spark.read.json(path)
  df.printSchema()                          ##          df.printSchema()
  df.show(false)                            ##          df.show

  //convert to avro
  df.write.format("avro").save("/tmp/avro/zipcodes.avro")

  //convert to parquet
  df.write.parquet("/tmp/parquet/zipcodes.parquet")

  //convert to csv
  df.write.option("header","true").csv("/tmp/csv/zipcodes.csv")
```

overwrite – mode is used to overwrite the existing file, alternatively, you can use SaveMode.Overwrite.
append – To add the data to the existing file, alternatively, you can use SaveMode.Append.
ignore – Ignores write operation when the file already exists, alternatively you can use SaveMode.Ignore.
errorifexists or error – This is a default option when the file already exists, it returns an error, alternatively, you can use SaveMode.ErrorIfExists.


Amazon S3
AWS Glue Crawlers
1. Crawlers scan your data sets and populate your Glue Data Catalog
AWS Glue Data Catalog
2. The Glue Data Catalog serves as a central metadata repository
Amazon Athena
Amazon Redshift
Amazon EMR
3. Once catalogued in Glue, your data is immediately available for analysis
Amazon QuickSight & Other BI Tools
4. Run reports

36. How to read data from AWS S3 using spark.

In the Spark sparkContext.textFile() and sparkContext.wholeTextFiles()methods to use to read test file from Amazon AWS S3 into RDD and **spark.read.text()** and **spark.read.textFile()** methods to read from Amazon AWS S3 into DataFrame.

37. What is windowing (window function) in Hive?

Windowing allows features to create a window on the set of data in order to operate aggregation like COUNT, AVG, MIN, MAX and other analytical functions such as LEAD, LAG, FIRST_VALUE, and LAST_VALUE.
The syntax of the query with windows:

SELECT <columns_name>, <aggregate>(column_name) OVER (<windowing specification>) FROM <table_name>;
where,
column_name – column name of the table
Aggregate – Any aggregate function(s) like COUNT, AVG, MIN, MAX
Windowing specification – It includes following:

- PARTITION BY – Takes a column(s) of the table as a reference.
- ORDER BY – Specified the Order of column(s) either Ascending or Descending.
- Frame – Specified the boundary of the frame by stat and end value. The boundary either be a type of RANGE or ROW followed by PRECEDING, FOLLOWING and any value.

These three (PARTITION, ORDER BY, and Window frame) are either alone or together.

38. Difference between Accumulator and Broadcast variable is spark?

Broadcast variables are read-only variables that are distributed across worker nodes in-memory instead of shipping a copy of data with tasks. Broadcast variables are mostly used when the tasks across multiple stages require the same data or when caching the data in the deserialized form is required.

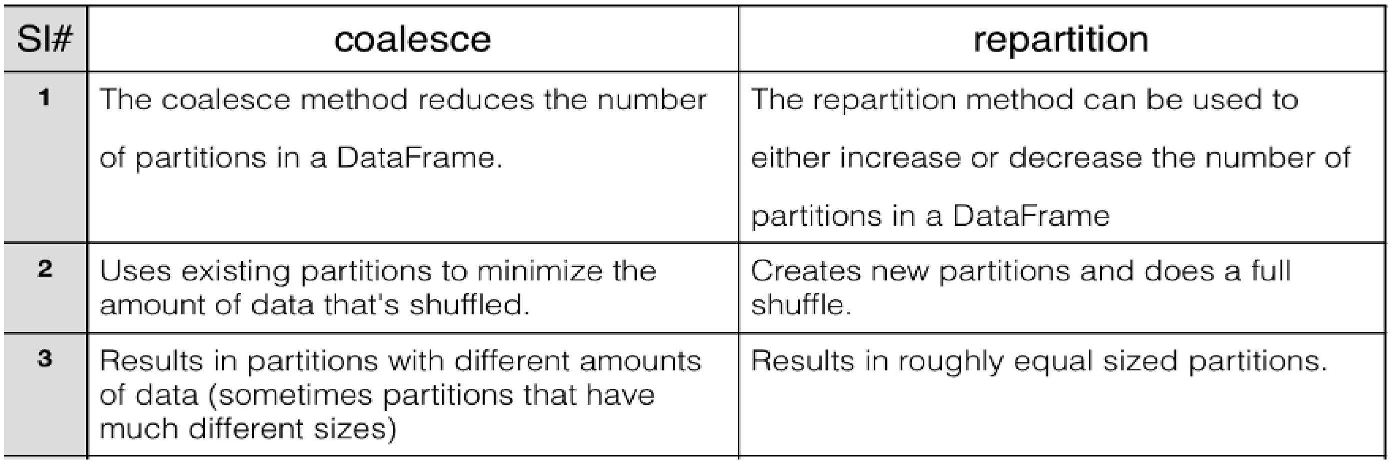
```
scala> val broadcastVar = sc.broadcast(Array(0, 1, 2, 3))
broadcastVar: org.apache.spark.broadcast.Broadcast[Array[Int]] = Broadcast(0)
```

Accumulators are a special kind of variable that we basically use to update some data points across executors. They can be used to implement counters (as in MapReduce) or sums.

Accumulators:	***Broadcast Variable:***
● Accumulators are shared variables ● They are used to aggregate values from worker nodes back to the driver program ● Only driver program can read an accumulator's value, not the tasks ● Accumulators are variables that can only be "added" to through an associative operation used to implement counters and sums efficiently in parallel ● One of the most common uses of accumulator is to count events that occur during job execution for debugging purposes	● Read only variables ● Immutable ● Fits in memory ● Distributed efficiently to the cluster ● Do not modify after shipped ● Preferred for machine learning and Lookup tables ● We cannot have dataframes in cache

39. Difference between coalesce and repartition

In Spark or PySpark repartition() method is used to increase or decrease the RDD, DataFrame, Dataset partitions whereas the Spark coalesce() method is used to only decrease the number of partitions in an efficient way.



Sl#	coalesce	repartition
1	The coalesce method reduces the number of partitions in a DataFrame.	The repartition method can be used to either increase or decrease the number of partitions in a DataFrame
2	Uses existing partitions to minimize the amount of data that's shuffled.	Creates new partitions and does a full shuffle.
3	Results in partitions with different amounts of data (sometimes partitions that have much different sizes)	Results in roughly equal sized partitions.

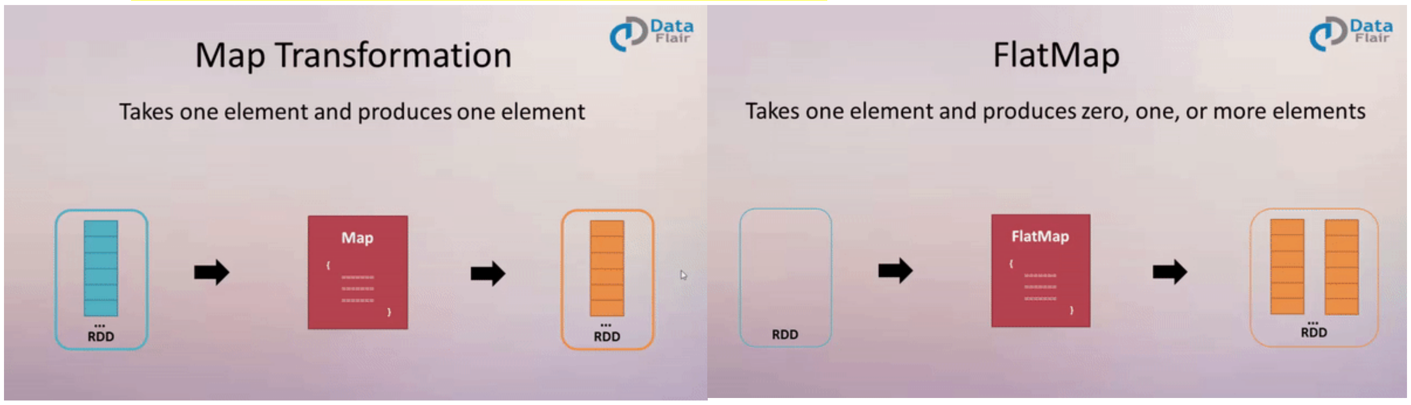
Coalesce avoids full shuffle, instead of creating new partitions, it shuffles the data using Hash Partitioner (Default), and adjusts into existing partitions, this means it can only decrease the number of partitions. **Repartition** shuffles all data over the network

40. What is RDD Lineage?

RDD Lineage (aka RDD operator graph or RDD dependency graph) is a graph of all the parent RDDs of an RDD. It is built as a result of applying transformations to the RDD and creates a logical execution plan.

41. What is the Difference between Map and FlatMap?

Spark map function expresses a one-to-one transformation. It transforms each element of a collection into one element of the resulting collection. `val rdd3:RDD[(String,Int)]= rdd2.map(m=>(m,1))`
While **Spark flatMap function** expresses a one-to-many transformation. It transforms each element to 0 or more elements. `val rdd2 = rdd.flatMap(f=>f.split(" "))`


Map Transformation
Takes one element and produces one element
RDD
Map
RDD
FlatMap
Takes one element and produces zero, one, or more elements
RDD
FlatMap
RDD
Data Flair

42. What do we do when we see Out of Memory issues when we submit a spark job?

- Have more executors (more than 2, defined by total-executor-cores in spark-submit and spark.executor.core in SparkSession)
- Have less cores per executor (3-5). (spark.executor.core)
- Add memory to executors (spark.executor.memory)
- Add memory to driver (driver-memory in spark-submit script)
- Make more partitions (make partitions smaller in size) (.config("spark.sql.shuffle.partitions", numPartitionsShuffle) in SparkSession)
- Look at PeakExecutionMemory of a Tasks in Stages (one of the additional metrics to turn on) tab to see if it is not to big

43. What kind of challenge do we face when we submit a spark job?

- Out of Memory Exceptions
  - Driver Memory Exceptions
    - Exception due to Spark driver running out of memory
  - Executor Memory Exceptions
    - Exception because executor runs out of memory
    - FetchFailedException due to executor running out of memory

Solution: Set a higher value for the driver memory,

`--conf spark.driver.memory= <XX>g`

- Executor container killed by YARN for exceeding memory limits - Set a higher value for spark.yarn.executor.memoryOverhead based on the requirements of the job. As a best practice, modify the executor memory value accordingly. --conf spark.yarn.executor.memoryOverhead=XXXX

- **FileAlreadyExistsException in Spark jobs** - *we see the FileAlreadyExistsException error when the executor runs out of memory, the individual tasks of that executor are scheduled on another executor. Also, when any Spark executor fails, Spark retries to start the task, which might result into FileAlreadyExistsException error after the maximum number of retries.*

Solution:

1. *Identify the original executor failure reason that causes the FileAlreadyExistsException error.*
2. *Verify size of the nodes in the clusters.*
3. *Upgrade them to the next tier to increase the Spark executor's memory overhead.*

36. What is Action and Transformation in Spark?

Action: An action is one of the ways of sending data from the Executor to the driver. Actions are Spark RDD operations that give non-RDD values. The values of action are stored to drivers or to the external storage system. It brings the laziness of RDD into motion. RDD actions are operations that return the raw values, In other words we can say, any RDD function that returns other than RDD[T] is considered as an action in spark programming.
Some of the actions of Spark are: count, collect, take, show, top, reduce etc.

Transformation: Spark Transformation is a function that produces new RDD from the existing RDDs. It takes RDD as input and produces one or more RDD as output. Each time it creates a new RDD when we apply any transformation. Thus, the input RDDs cannot be changed since RDD are immutable in nature. Applying transformation built an RDD lineage, with the entire parent RDDs of the final RDD(s). RDD lineage, also known as RDD operator graph or RDD dependency graph. It is a logical execution plan i.e., it is Directed Acyclic Graph (DAG) of the entire parent RDDs of RDD. (map, flatmap, map partition, filter, union, distinct, join, group by key, reduce by key,

37. What is Teradata and how to import data from teradata to Hive/Hadoop?

Teradata is a Terabyte scale Relational Database Management System (RDBMS) catering to data warehousing needs and offers high speed, high efficiency, and multi-user access by utilizing the concept of 'Parallelism'. Teradata is based on Massively Parallel Processing (MPP) Architecture.
I use Sqoop and the Teradata JDBC driver to import data from teradata to hive/HDFS. First install the TeraData JDBC driver and then follow the commands:

```
sqoop import \
-libjars ${LIB_JARS},${TDCHJARS} \
--driver com.teradata.jdbc.TeraDriver \
--connect $JDBCURL \
--table $TDTABLE \
--hive-table ${HIVEDB}.${HIVETABLE} \
--where "${TDWHERE}" \
--username $TDUSER \
--password $TDPASS \
--map-column-java EFF_TSP=String \
--num-mappers 1 \
--map-column-hive EFF_TSP=STRING \
--hive-import
```

38. What is the purpose of split by column in sqoop

The command --split-by is used to specify the column of the table used to generate splits for imports. This means that it specifies which column will be used to create the split while importing the data into the cluster. Basically it is used to improve the import performance to achieve faster parallelism.

39. How do you improve the performance of sqoop job?

a. Increase the mappers (by default 4)
b. Increase fetch size (we can set 1,000-10,000 or more) by default 1000.
c. Increase heap size (**Heap space** in Java is used for dynamic memory allocation for Java objects and JRE classes at the runtime.
d. Increase memory (`-Dmapreduce.map.memory.mb=8192 -Dmapreduce.map.java.opts=-Xmx7200m)`
e. split-by and boundary-query

40. What is DAG?

A DAG is a directed acyclic graph. They are commonly used in computing systems for task execution. DAG is like a flow chart that tells the system which tasks to execute and in what order. Directed Acyclic Graph) DAG in Spark is a set of Vertices and Edges, where vertices represent the RDDs and the edges represent the Operation to be applied on RDD.

Load data from DB2/DashDB into Apache Spark using DataFrames API

```
import com.ibm.spark.ibmdataserver.Constants

import org.apache.spark.sql.SQLContext
import org.apache.spark.{SparkContext, SparkConf}

val sparkContext = new SparkContext(conf)
val sqlContext = new SQLContext(sparkContext)

val df = spark.read
    .format("JDBC")
    .option(Constants.JDBCURL, DB2_CONNECTION_URL:user, password) //Specify the JDBC
connection URL
    .option(Constants.TABLE,tableName) //Specify the table from which to read
    .load()
df.show()
```

Persist data from Apache Spark DataFrames into DB2/DashDB

```
import com.ibm.spark.ibmdataserver.Constants
import org.apache.spark.sql.SQLContext
import org.apache.spark.{SparkContext, SparkConf}

val sparkContext = new SparkContext(conf)
val sqlContext = new SQLContext(sparkContext)

val df = sqlContext.read.json("/Users/sparkuser/data.json")
df.write
    .format("com.ibm.spark.ibmdataserver")
    .option(Constants.JDBCURL, DB2_CONNECTION_URL) //Specify the JDBC connection URL
    .option(Constants.TABLE,tableName) //Specify the table (will be created if not present)
to which data is to be written
    .option(Constants.TMPPATH, tmpPath) //Temporary path to be used for generating
intermediate files during processing [System tmp path will be used by default]
    .mode("Append")
    .save()
```

41. Beeline VS Hive CLI

The Hive CLI, which connects directly to HDFS and the Hive Metastore, and can be used only on a host with access to those services.

Beeline, which connects to HiveServer2 and requires access to only one .jar file: hive-jdbc-<version>-standalone.jar.

42. What is the difference between RANK and DENSE_RANK?

Both are window functions. RANK gives us the ranking within our ordered partition. Ties are assigned the same rank, with the next ranking(s) skipped. It doesn't give us consecutive integer numbers. On the other hand DENSE_RANK gives us the ranking within our ordered partition, but the ranks are consecutive in it. Also, no ranks are skipped if there are ranks with multiple items.

For example, consider the set {25, 25, 50, 75, 75, 100}. For such a set, RANK() will return {1, 1, 3, 4, 4, 6} (note that the values 2 and 5 are skipped), whereas DENSE_RANK() will return {1,1,2,3,3,4}.

Example: `SELECT empno, deptno, sal, RANK() OVER (PARTITION BY deptno ORDER BY sal) AS rank FROM emp;`

`SELECT empno, deptno, sal, DENSE_RANK() OVER (PARTITION BY deptno ORDER BY sal)AS dense_rank" FROM emp;`

43. How will you get top 3 salaries?

We can use the window function here.

`select dept_name, emp_name, salary from ( select d.name as dept_name, e.name as emp_name, e.salary, rank() over(partition by d.name order by e.salary desc) as rank_count from employee e, department d where d.id = e.dept_id) a where a.rank_count <4`

```
select * from(
select ename, sal, dense_rank()
over(order by sal desc)r from Employee)
where r=&n;
```

44. How to find Nth highest salary from a table?

```
select * from(
select ename, sal, dense_rank()
over(order by sal desc)r from Employee)
where r=&n;

To find to the 2nd highest sal set n = 2
To find 3rd highest sal set n = 3 and so on.
```

38. What is the case class in scala and what are the benefits?
Case classes are almost the same as regular classes with a few key differences. A case class requires the keywords `case class`, an identifier, and a parameter list (which may be empty):
Benefits of case class:

- Case class constructor parameters are public val fields by default, so accessor methods are generated for each parameter.
- An apply method is created in the companion object of the class, so you don’t need to use the new keyword to create a new instance of the class.
- An unapply method is generated, which lets you use case classes in more ways in match expressions.
- A copy method is generated in the class. You may not use this feature in Scala/OOP code, but it’s used all the time in Scala/FP.
- equals and hashCode methods are generated, which let you compare objects and easily use them as keys in maps.
- A default toString method is generated, which is helpful for debugging.

Difference between case class and regular class in Scala:
1.Case Class doesn't need explicit new, while class need to be called with new
```
val classInst = new MyClass(...)  // For classes
val classInst = MyClass(..)       // For case class
```

2.By Default constructors parameters are private in class , while its public in case class
```
// For class
class MyClass(x:Int) { }
val classInst = new MyClass(10)
classInst.x   // FAILURE : can't access
// For caseClass
case class MyClass(x:Int) { }
val classInst = MyClass(10)
classInst.x   // SUCCESS
```

3.case class compare themselves by value
```
// case Class
class MyClass(x:Int) { }
val classInst = new MyClass(10)
val classInst2 = new MyClass(10)
classInst == classInst2 // FALSE
// For Case Class
case class MyClass(x:Int) { }
```

```
val classInst = MyClass(10)
val classInst2 = MyClass(10)
classInst == classInst2 // TRUE
```

39. What is a factory type design pattern or singleton design pattern in scala?
The factory type design pattern is used to offer a single interface to instantiate one of the multiple classes. It is used when we have a super class with multiple sub-classes and based on input, we need to return one of the sub-class.
Benefits of factory design pattern:

a. Loose Coupling between Object Creation logic and Client.
b. Clear separation of Responsibilities.
c. Easy to change object creation logic without affecting Client program

40. What is Map Side join in Hive?
Map side join is a process where joins between two tables are performed in the Map phase without the involvement of Reduce phase. Map-side Joins allows a table to get loaded into memory ensuring a very fast join operation, performed entirely within a mapper and that too without having to use both map and reduce phases.
40. What is the dining problem in scala?
41. What is the eld key in scala?

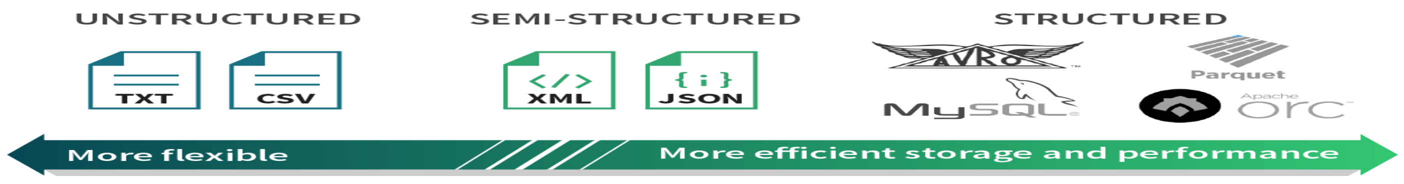
42. What is Traits in scala?
A trait is like an interface with a partial implementation. In scala, trait is a collection of abstract and non-abstract methods. We can create traits that can have all abstract methods or some abstract and some non-abstract methods. **Traits** are used to share interfaces and fields between classes.

What is Tableau:
Tableau is a data visualization and data analytics tool that aims to help people see and understand data. On the other hand I can say it simply converts raw data into a very easily understandable format.
Hive - Connecting to Tableau - Hands-on

- Open tableau.
- Go to > To a Server > Hortonworks Hadoop Hive.
- Enter c.cloudxlab.com as Server and port as 10000.
- Select HiveServer2 as type.
- Select Username and Password as Authentication.
- Enter your lab username and password.
- Click on Sign in and wait for the connection to establish.
- Type in your database name in schema and press enter. Now select your database.
- Click on search under the table to list all the tables in your database.
- Double click on nyse table.
- Now click on Go to Worksheet.
- Drag and drop symbol1 to columns and Price High to rows.
- Click on the Show Me button on the top right corner and select the recommended chart by tableau.
- Now drag Ymd to filters and select 2009-12-31 then click on OK.
- Now sort the data points to see the Top 10 values.

1. **What is map side join in hive?**

Map side join is a process where joins between two tables are performed in the Map phase without the involvement of Reduce phase.
Map-side Joins allows a table to get loaded into memory ensuring a very fast join operation, performed entirely within a mapper and that too without having to use both map and reduce phases.



2. **How to load a streaming Dataset from Kafka using Spark SQL ?**

```
val df = spark.readStream
        .format("kafka")
        .option("kafka.bootstrap.servers", "192.168.1.100:9092")
        .option("subscribe", "json_topic")
        .option("startingOffsets", "earliest") // From starting
        .load()
df.printSchema()
```

3. **Difference between Union and UnionAll in Spark**

Union() method of the DataFrame is used to combine two DataFrame's of the same structure/schema. If schemas are not the same it returns an error.
UnionAll() is deprecated since Spark "2.0.0" version and replaced with union().
Note: In other SQL's, Union eliminates the duplicates but UnionAll combines two datasets including duplicate records. But, in spark both behave the same and useDataFrame duplicate function to remove duplicate rows.

What is Kafka?
Kafka is a highly scalable, fault-tolerant distributed publish-subscribe enterprise messaging system.

So, we can say Kafka is a distributed streaming platform.
- It can be used as an enterprise messaging system.
- It can be used for stream processing.
- It also provides connectors to import and export bulk data from databases and other systems.

We have few things in Kafka, such as:
1. **Producer, Consumer, Broker, Cluster, Topics, Partitions,Offset, Consumer groups, Leader, Follower**

Producer: Producer is an application that sends messages to Kafka Broker.

Consumer: Consumer is an application that reads data from Kafka Broker.

Broker: Broker is a Kafka server. It is a meaningful name just given to Kafka server. The producer and consumer don't interact directly, they use Kafka server as an agent or a broker to exchange messages.

Cluster: Cluster is a group of computers sharing workload for a common purpose.

Kafka's having more than one broker is called Kafka Cluster.

Topic: A Topic is a unique name for Kafka stream. Data is stored in Topics.

Partition: Kafka brokers store the data in a Topics, but the data can be huge like larger than the storage capacity of a single machine and brokers have a challenge to store the data. In this case brokers break it into two or more parts and distribute it to multiple computers and store the data.
This process we called partition. Every partition sits on a single machine and we can not break it again.

Partition offset: Each partitioned message has a unique sequence id called as offset.
Consumer group: It is a group of consumers acting as a single logical unit.

Leader: Leader is the node responsible for all reads and writes for the given partition. Every partition has one server acting as a leader.
Follower: Node which follows leader instructions are called as follower. If the leader fails, one of the follower will automatically become the new leader. A follower acts as normal consumer, pulls messages and up-dates its own data store.

ZooKeeper: ZooKeeper is used for managing and coordinating Kafka brokers. ZooKeeper service is mainly used to notify producers and consumers about the presence of any new broker in the Kafka system or failure of the broker in the Kafka system. As per the notification received by the Zookeeper regarding presence or failure of the broker then producer and consumer takes decision and starts coordinating their task with some other broker.

Kafka Workflow:

- First starts Zookeeper: bin/zookeeper-server-start.sh config/zookeeper.properties
- Start Kafka broker: bin/kafka-server-start.sh config/server.properties
- Create Kafka Topic: bin/kafka-topics.sh --create --zookeeper localhost:2181 \ --replication-factor 1 \ --partitions 1 \ --topic text_topic
- List all Topics: bin/kafka-topics.sh --zookeeper localhost:2181 –list
- Describe Topic: bin/kafka-topics.sh --zookeeper localhost:2181 –describe
- Run Kafka Producer: bin/kafka-console-producer.sh \ --broker-list localhost:9092 --topic text_topic
- Run Kafka Consumer: bin/kafka-console-consumer.sh \ --bootstrap-server localhost:9092 --topic text_topic --from-beginning

Spark SQL Batch Processing – Producing Messages to Kafka Topic:

```
package com.sparkbyexamples.spark.streaming.batch
import org.apache.spark.sql.SparkSession
object WriteDataFrameToKafka {
def main(args: Array[String]): Unit = {
val spark: SparkSession = SparkSession.builder()
.master("local[1]")
.appName("SparkByExamples.com")
.getOrCreate()

val data = Seq (("iphone", "2007"),("iphone 3G","2008"),
("iphone 3GS","2009"),
("iphone 4","2010"),
("iphone 4S","2011"),
("iphone 5","2012"),
("iphone 8","2014"),
("iphone 10","2017"))

val df = spark.createDataFrame(data).toDF("key","value")
/* since we are using dataframe which is already in text,        selectExpr is
optional. If the bytes of the Kafka records represent UTF8 strings, we can simply
use a cast to convert the binary data       into the correct type.
     df.selectExpr("CAST(key AS STRING)", "CAST(value AS STRING)")     */
df.write
.format("kafka")       .option("kafka.bootstrap.servers","192.168.1.100:9092")
.option("topic","text_topic")
.save()
}
}
```

Spark SQL Batch Processing – Consuming Messages from Kafka Topic:

```
package com.sparkbyexamples.spark.streaming.batch
import org.apache.spark.sql.SparkSession
object ReadDataFromKafka {

def main(args: Array[String]): Unit = {
val spark: SparkSession = SparkSession.builder()
.master("local[1]")
.appName("https://SparkByExamples.com")
.getOrCreate()
val df = spark.read.format("kafka").option("kafka.bootstrap.servers",
"192.168.1.100:9092").option("subscribe", "text_topic")
.load()

df.printSchema()
val df2 = df.selectExpr("CAST(key AS STRING)",
          "CAST(value AS STRING)","topic")
df2.show(false)
}
}
```

Reading Data from Kafka

- **Creating a Kafka Source for Streaming Queries**

```
// Subscribe to a pattern
val df = spark
  .readStream
  .format("kafka")
  .option("kafka.bootstrap.servers", "host1:port1,host2:port2")
  .option("subscribePattern", "topic.*")
  .load()
df.selectExpr("CAST(key AS STRING)", "CAST(value AS STRING)")
```

```
  .as[(String, String)]
```

Creating a Kafka Source for Batch Queries

```
// Subscribe to a pattern, at the earliest and latest offsets
val df = spark
  .read
  .format("kafka")
  .option("kafka.bootstrap.servers", "host1:port1,host2:port2")
  .option("subscribePattern", "topic.*")
  .option("startingOffsets", "earliest")
  .option("endingOffsets", "latest")
  .load()
df.selectExpr("CAST(key AS STRING)", "CAST(value AS STRING)")
  .as[(String, String)]
```

How to Drop an external hive table with data?

First we can alter the table on all tables and change the external table to an internal table then drop the table.

```
hive> ALTER TABLE <table-name> SET TBLPROPERTIES('EXTERNAL'='False'); //changing the tbl properties to to make the table as internal
hive> drop table <table-name>; //now the table is internal if you drop the table data will be dropped automatically.
```

How to Delete and Replace columns in the Hive table?

Let's suppose we have a table name emp and we have 3 columns named e_id int, e_name string, e_dept_id int. Nowe we want to delete the column **e_dept_id**. We can use:

```
ALTER table emp replace columns (e_id  int, e_name  string);
```

It will delete the column **e_dept_id.**

If we want to replace some columns, we can use:

```
ALTER table emp replace columns (id  int, name  string);
```

It will replace the new column name.

How to UPDATE data in the Hive table?

Let's suppose we have tables named employee1 with 3 columns and 3 rows (id, name and salary) and employee2. with 3 columns and 2 rows. We can update employee1 table by:

INSERT INTO TABLE employee1 SELECT * FROM employee2;

Output will be 5 rows.

How to insert rows:

Let's suppose we want to insert 2 more rows in the table employee1.

INSERT INTO TABLE employee1 VALUES(new_id, new_name, new_salary);

What is Internal table and External table:

Internal or manage table: When we create a table in hive, it by default manages the data, meaning that hive moves the data into it's warehouse directory.

By default tables created in hive are managed.

Data Load: When we load data into managed table, then Hive moves data into Hive warehouse directory which is **`hdfs://user/hive/warehouse`**

Drop Table: The metadata information along with the table data is deleted from the Hive warehouse if we drop the manage table.

External Table: External tables are not created by default, we have to use External Keyword during table creation along with the data location.

```
CREATE EXTERNAL TABLE <table_name>(dummy STRING)
LOCATION '/USER/<directory_name>/<table_name>';
LOAD DATA INPATH 'user/<directory_name>/data.txt INTO TABLE <table_name>';
```

External tables refer to the data that is defined location outside the warehouse directory. The location of the external data is specified at the table creation time.
Drop Table: Hive just deletes the metadata information regarding the table, but it leaves the table data present in HDFS untouched.

Security:

- Managed tables - hive solely controls the managed table security. Within hive, security needs to be managed; probably at the schema level (depends on organization policies).
- External Tables: these tables' files are accessible to anyone who has access to HDFS file structure. So, it needs to manage security at the HDFS file/folder lever.

When to use Managed table and when to use External table:
We use managed table when-

- We want hive to completely manage the lifecycle of the data and table
- Data is temporary

We use External table when-

- Data is used outside of Hive. For example, the data files are read and processed by an existing program that does not lock the files.
- We are creating a table based on the existing table
- We need data to remain in the underlying location even after a DROP TABLE
- The hive doesn't own data and control settings, directories …. We may have another program or process that will do these things.

How to create a PARTITION TABLE IN HIVE?

```
CREATE TABLE Customer_transactions (
Customer_id VARCHAR(40),
txn_amout DECIMAL(38, 2),
txn_type  VARCHAR(100))
PARTITIONED BY (txn_date STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '|' LINES TERMINATED BY '\n'
STORED AS TEXTFILE;
```

Note: For partition we have to set this property:

```
set hive.exec.dynamic.partition.mode=nonstrict
```

Loading data into partition table:

```
INSERT OVERWRITE TABLE <table_name> PARTITION(state<partition_column>)
SELECT district,enrolments,state from  allstates;
```

Alter Partitions

We can alter/change partitions (add/change/drop) with the help of below commands.

Adding Partitions:

We can add partitions to an existing table with ADD PARTITION clause as shown below.

```
ALTER TABLE <table_name> ADD IF NOT EXISTS
PARTITION (country = 'US', state = 'XY') LOCATION '/hdfs/external/file/path1'
PARTITION (country = 'CA', state = 'YZ') LOCATION '/hdfs/external/file/path2'
PARTITION (country = 'UK', state = 'ZX') LOCATION '/hdfs/external/file/path2'
```

Changing Partitions:

We can change a partition location with commands like below. This command does not move the data from the old location and does not delete the old data but the reference to old data file will be lost.

```
ALTER TABLE <table_name> PARTITION (country='US', state='CA')
SET LOCATION '/hdfs/partition/newpath';
```

Drop Partitions:

We can drop partitions of a table with DROP IF EXISTS PARTITION clause as shown below.

```
ALTER TABLE partitioned_user DROP IF EXISTS PARTITION(country='US', state='CA');
```

Use of HiveQL WHERE, ORDER BY, GROUP BY AND JOINS

```
SELECT * FROM employee WHERE salary>30000;
SELECT Id, Name, Dept FROM employee ORDER BY DEPT;
SELECT Dept,count(*) FROM employee GROUP BY DEPT;
```

CUSTOMERS TABLE:

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
| 1  | Ramesh   | 32  | Ahmedabad | 2000.00  |
| 2  | Khilan   | 25  | Delhi     | 1500.00  |
| 3  | kaushik  | 23  | Kota      | 2000.00  |
| 4  | Chaitali | 25  | Mumbai    | 6500.00  |
| 5  | Hardik   | 27  | Bhopal    | 8500.00  |
| 6  | Komal    | 22  | MP        | 4500.00  |
| 7  | Muffy    | 24  | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

ORDERS TABLE:

```
+-----+---------------------+-------------+--------+
|OID  | DATE                | CUSTOMER_ID | AMOUNT |
+-----+---------------------+-------------+--------+
| 102 | 2009-10-08 00:00:00 |           3 | 3000   |
| 100 | 2009-10-08 00:00:00 |           3 | 1500   |
| 101 | 2009-11-20 00:00:00 |           2 | 1560   |
| 103 | 2008-05-20 00:00:00 |           4 | 2060   |
+-----+---------------------+-------------+--------+
```

```
SELECT c.ID, c.NAME, c.AGE, o.AMOUNT
FROM CUSTOMERS c JOIN ORDERS o
ON (c.ID = o.CUSTOMER_ID);
```

OUTPUT:

```
+----+----------+-----+--------+
| ID | NAME     | AGE | AMOUNT |
+----+----------+-----+--------+
| 3  | kaushik  | 23  | 3000   |
| 3  | kaushik  | 23  | 1500   |
| 2  | Khilan   | 25  | 1560   |
| 4  | Chaitali | 25  | 2060   |
+----+----------+-----+--------+
```

LEFT OUTER JOIN:

```
SELECT c.ID, c.NAME, o.AMOUNT, o.DATE
FROM CUSTOMERS c
LEFT OUTER JOIN ORDERS o
ON (c.ID = o.CUSTOMER_ID);
```

OUTPUT:

```
+----+----------+--------+---------------------+
| ID | NAME     | AMOUNT | DATE                |
+----+----------+--------+---------------------+
| 1  | Ramesh   | NULL   | NULL                |
| 2  | Khilan   | 1560   | 2009-11-20 00:00:00 |
| 3  | kaushik  | 3000   | 2009-10-08 00:00:00 |
| 3  | kaushik  | 1500   | 2009-10-08 00:00:00 |
| 4  | Chaitali | 2060   | 2008-05-20 00:00:00 |
| 5  | Hardik   | NULL   | NULL                |
| 6  | Komal    | NULL   | NULL                |
| 7  | Muffy    | NULL   | NULL                |
+----+----------+--------+---------------------+
```

RIGHT OUTER JOIN:

```
SELECT c.ID, c.NAME, o.AMOUNT, o.DATE FROM CUSTOMERS c RIGHT OUTER JOIN ORDERS o ON
(c.ID = o.CUSTOMER_ID);
OUTPUT:
```

```
+------+----------+--------+---------------------+
| ID   | NAME     | AMOUNT | DATE                |
+------+----------+--------+---------------------+
| 3    | kaushik  | 3000   | 2009-10-08 00:00:00 |
| 3    | kaushik  | 1500   | 2009-10-08 00:00:00 |
| 2    | Khilan   | 1560   | 2009-11-20 00:00:00 |
| 4    | Chaitali | 2060   | 2008-05-20 00:00:00 |
+------+----------+--------+---------------------+
```

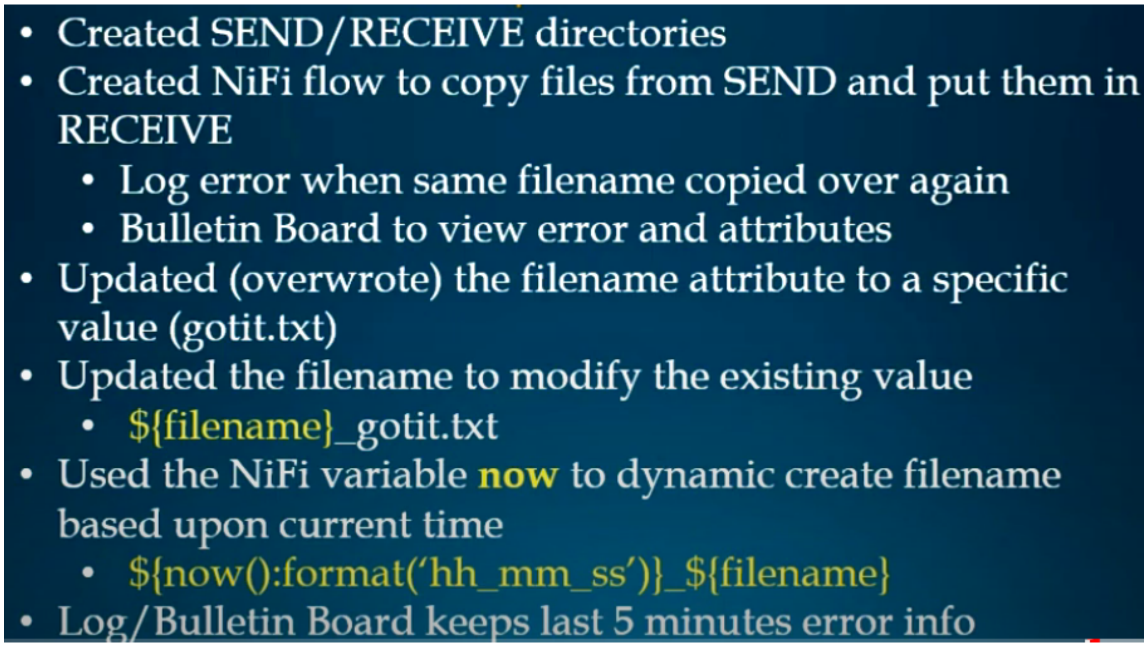
FULL OUTER JOIN:

```
SELECT c.ID, c.NAME, o.AMOUNT, o.DATE
FROM CUSTOMERS c
FULL OUTER JOIN ORDERS o
ON (c.ID = o.CUSTOMER_ID);
```

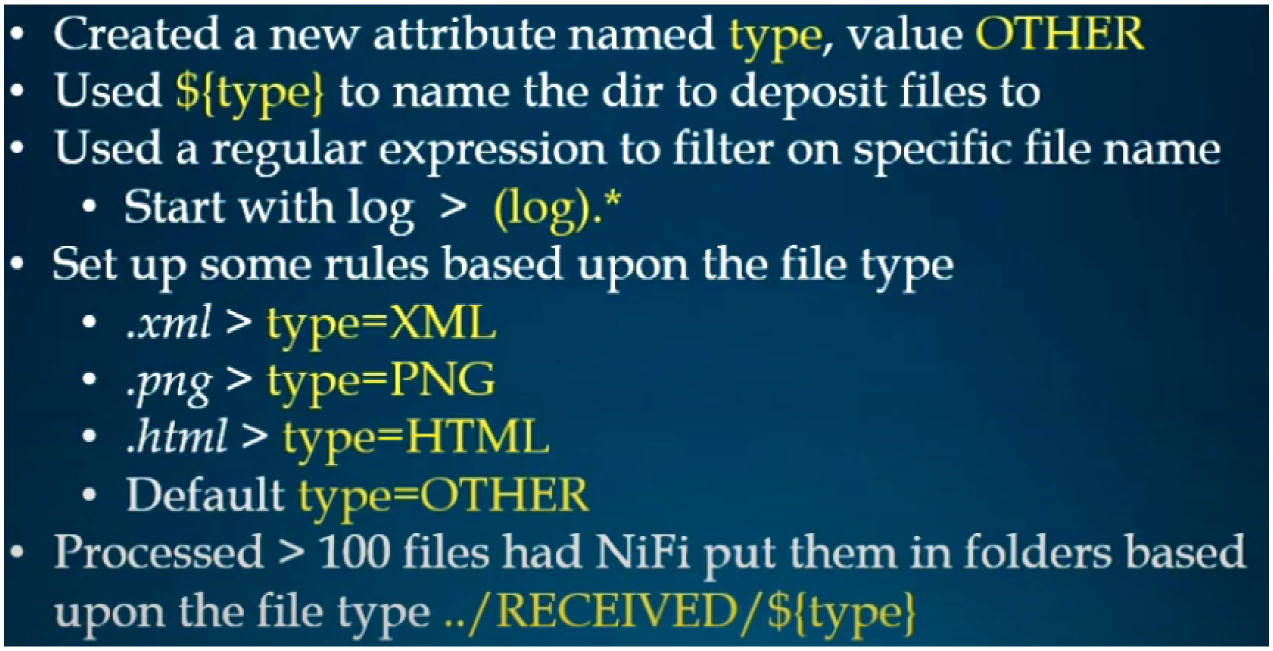
OUTPUT:

```
+------+----------+--------+---------------------+
| ID   | NAME     | AMOUNT | DATE                |
+------+----------+--------+---------------------+
| 1    | Ramesh   | NULL   | NULL                |
| 2    | Khilan   | 1560   | 2009-11-20 00:00:00 |
| 3    | kaushik  | 3000   | 2009-10-08 00:00:00 |
| 3    | kaushik  | 1500   | 2009-10-08 00:00:00 |
| 4    | Chaitali | 2060   | 2008-05-20 00:00:00 |
| 5    | Hardik   | NULL   | NULL                |
| 6    | Komal    | NULL   | NULL                |
| 7    | Muffy    | NULL   | NULL                |
| 3    | kaushik  | 3000   | 2009-10-08 00:00:00 |
| 3    | kaushik  | 1500   | 2009-10-08 00:00:00 |
| 2    | Khilan   | 1560   | 2009-11-20 00:00:00 |
| 4    | Chaitali | 2060   | 2008-05-20 00:00:00 |
+------+----------+--------+---------------------+
```

NiFi Workflow:



- Created SEND/RECEIVE directories
- Created NiFi flow to copy files from SEND and put them in RECEIVE
  - Log error when same filename copied over again
  - Bulletin Board to view error and attributes
- Updated (overwrote) the filename attribute to a specific value (gotit.txt)
- Updated the filename to modify the existing value
  - ${filename}_gotit.txt
- Used the NiFi variable **now** to dynamic create filename based upon current time
  - ${now():format('hh_mm_ss')}_${filename}
- Log/Bulletin Board keeps last 5 minutes error info



- Created a new attribute named type, value OTHER
- Used ${type} to name the dir to deposit files to
- Used a regular expression to filter on specific file name
  - Start with log > (log).*
- Set up some rules based upon the file type
  - *.xml* > type=XML
  - *.png* > type=PNG
  - *.html* > type=HTML
  - Default type=OTHER
- Processed > 100 files had NiFi put them in folders based upon the file type ../RECEIVED/${type}