Big Data Notes

Big Data

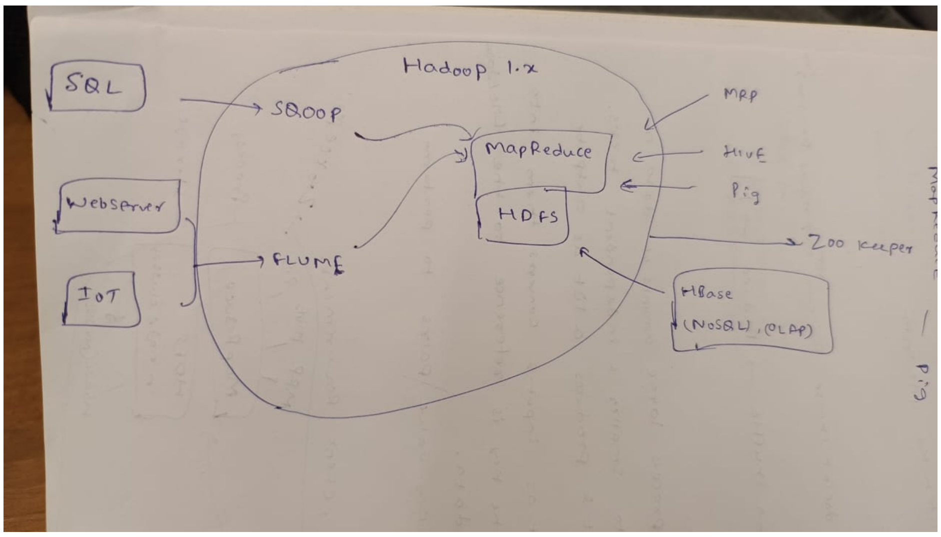
- RDMS is only for limited volume of data.
- Big data is a concept (not a software or hardware). To apply this there are two frameworks are there they are i) Hadoop and ii) Spark.

Big data means:

1. Variety: Structured, semi-structured and unstructured
2. Volume: The size and amounts of data
3. Velocity: The speed at companies receive data
4. Veracity: Trust worthiness or accuracy of data
5. Value: The benefits that the big data can provide

- Hadoop is used to analyse in Disk (Hard disk). Spark is used to analyse in Memory(RAM).
- Spark is much faster than Hadoop. But both produce the same result.
- Spark is very expensive. Hadoop and Spark both are owned by Apache. They are open source.
- MySQL, Oracle, etc., have storage in themselves, they store the data inside them only. Hadoops and Spark work as engine with the data. The data itself is stored in DFS (Distributed file system).
- In DFS the data is divided into smaller chunks and stored it in multiple systems.
- DFS is not a database or a data warehouse, because it does not have any engines, APIs connected to it. it is just an offline flat file system.
- In Hadoop, we don't use a database at all. We use DFS, Data warehouse (modern), Data lake, Delta lake, Data lake house.
- The output is also stored in the same DFS; we don't need other place.

Hadoop



It is a software.

The engine in Hadoop is MapReduce, it is the first distributed programming language.

Hadoop = HDFS + MapReduce. MapReduce is used to process and HDFS is used for storage.

In Hadoop cluster, HDFS knows only MapReduce language. Even if you write code in SQL it is converted into MapReduce.

SQOOP is used to bring data from traditional data sources like MySQL into DFS. This tool is available in Hadoop itself. This process is called ETL. SQOOP is also used to export the data.

SQOOP can be used only for only static sources.

To get data from dynamic sources we need to use **FLUME**. We cannot do exporting using FLUME.

The output should always be in structured format only; the input format doesn't matter.

Importing data is called Data ingestion.

Pipeline: Connection from source to target to get the data.

SQOOP and FLUME are not used for any kind of analysis. It is just used for data ingestion.

We can do processing and analysis in Hadoop itself. In order to analyze we need to use MapReduce language. MapReduce code is very big so, **HIVE** is created. The hive code is then converted into MapReduce. Hive code is similar to SQL. Hive is a Data Warehouse; it uses HDFS as storage. It is dependent on HDFS, so it is not independent.

We can also apply transformations using Hadoop, but only some are possible.

Pig is also used for analysis language. But it is not currently used.

All these languages (MRP, Hive, Pig) are used for analysis called Batch Processing (offline processing) and HBase (NoSQL, online processing, OLAP). HBase is not converted into MapReduce, it is directly connected to HDFS with the help of Zookeeper.

If any request is going through MapReduce it is offline processing. HBase is on top of HDFS (It doesn't need MapReduce).

Online processing: If you want to filter records based on a column, offline processing reads only that column and not the entire file or table.

Offline processing: It scans the entire file (like SQL).

Zoo keeper manages all these components (SQOOP, FLUME, MapReduce, HDFS, HBase) and cluster management (like name node, etc.,).

Oozie is the component of Hadoop eco system that is used to schedule the jobs.

Drawbacks of Hadoop 1 cluster

1. No resource management (It follows Queue structure, one process after other, no parallel working)
2. No High availability (only 1 master machine)

To overcome these problems, Hadoop 2 has been introduced. In this version, YARN (yet another resource negotiator) has been introduced as a replacement of MapReduce. There are two master machines (high configuration system) which always run in parallel. YARN performs all the jobs parallelly.

Note: We cannot modify the data when the data in HDFS because it is a file system and not a database.

HDFS

Cluster -> Data Centers -> Racks -> Nodes

Name node is the master node and it is also called meta node.

Within 3 seconds if the slave doesn't respond then the slave is treated as dead node.

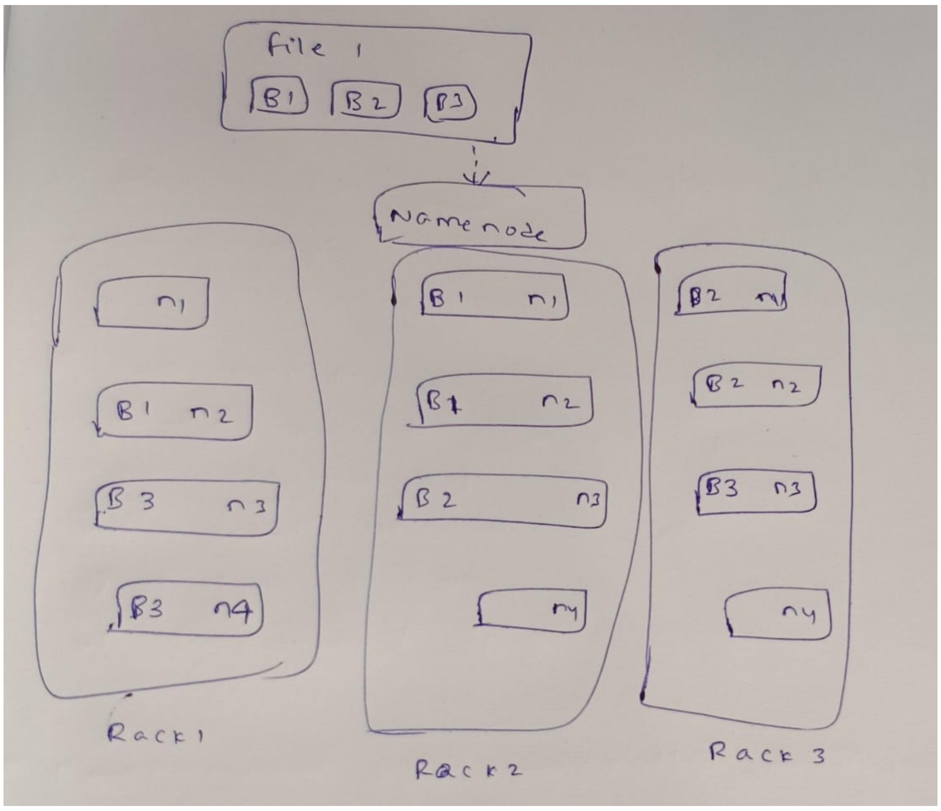
Name node is the master node. Data nodes are slave nodes.

Min 50GB storage for each data node.

HDFS default replication size is 3

DFS is also called as Data lake.

Default block size is 128MB recommended



We can check the HDFS files in link: http://localhost:50070

We can check the currently running tasks in MapReduce only. HDFS does not store the details of it Link for MapReduce: http://localhost:50030

DFS in cloud:

1. In AWS: S3
2. Azure: ADLS (Azure data lake Service)
3. Google cloud storage

The Name nodes has two files associated with it, they are FSImage and EditLog. The FS Image has the information about complete state of the file system since the start of the Hadoop. The edit log contains all the recent modifications made to the file system.

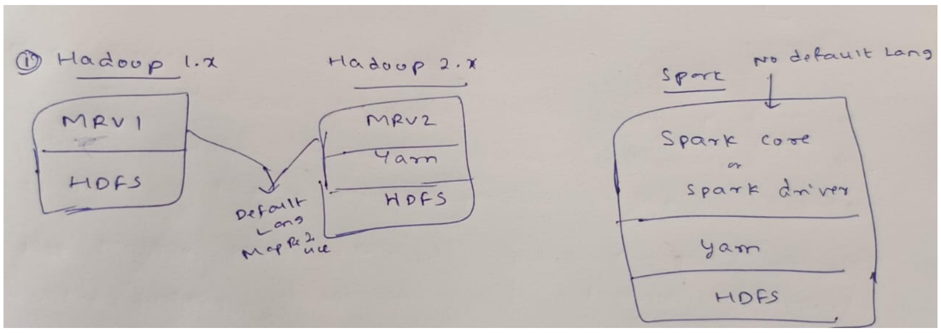
The data node is a block server that stores and maintains the data blocks.

Secondary name node is the component that constantly reads the meta data. It takes snapshots of data of active name node and store the data in FS Image. It is like a helper node.

Zookeeper: Is responsible for the maintaining of the health status of the architecture as well. Zookeeper server is a collection of all server nodes, it allows distributed processing to coordinate with each other through a shared hierarchical namespace. It also follows master slave architecture. There will ne one leader server and other slave servers.

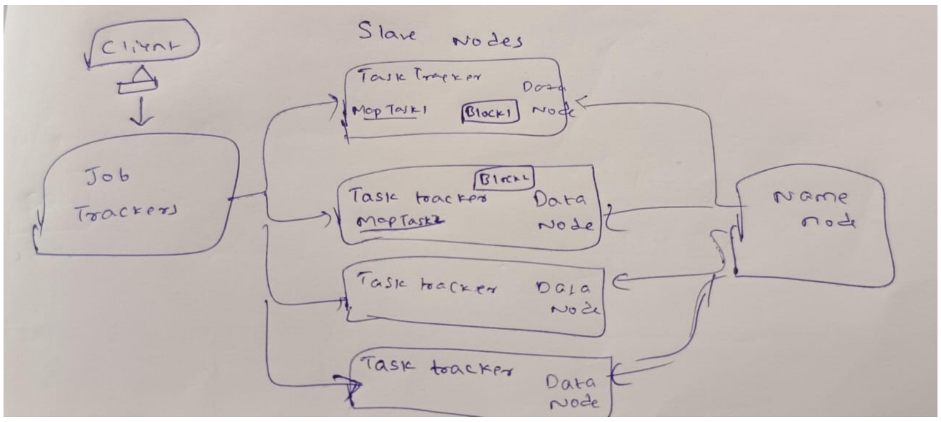
Erasure coding: Erasure coding (EC) is one of the methods of data protection through which the data is broken into sectors. Then they are expanded and encoded with redundant data pieces and stored across different storage media. Erasure coding adds the redundancy to the system that tolerates failures.

MapReduce Engine



It has Job tracker(s) as master to control the running of the Job. There are Task trackers which work as the slave nodes for the Job Tracker which control the running of the task.

The job trackers take the job (operation or work), then it goes to the name node and asks for the details of files needed. Then the task is divided and given to task trackers. These smaller tasks are called map tasks; each map task is executed in each block. Therefore, the number of map tasks is equal to number of blocks.



Client -> job tracker -> Name node -> task tracker -> data node -> task tracker -> job tracker-> client

Speculative execution of task:

Max 4 duplicate tasks (Total 5 tasks) will be run in each node then the task fails. If the duplicate tasks complete successfully in time, then the actual task is killed.

Number of blocks = number of map tasks

Analysis using Hadoop 2 or YARN cluster:

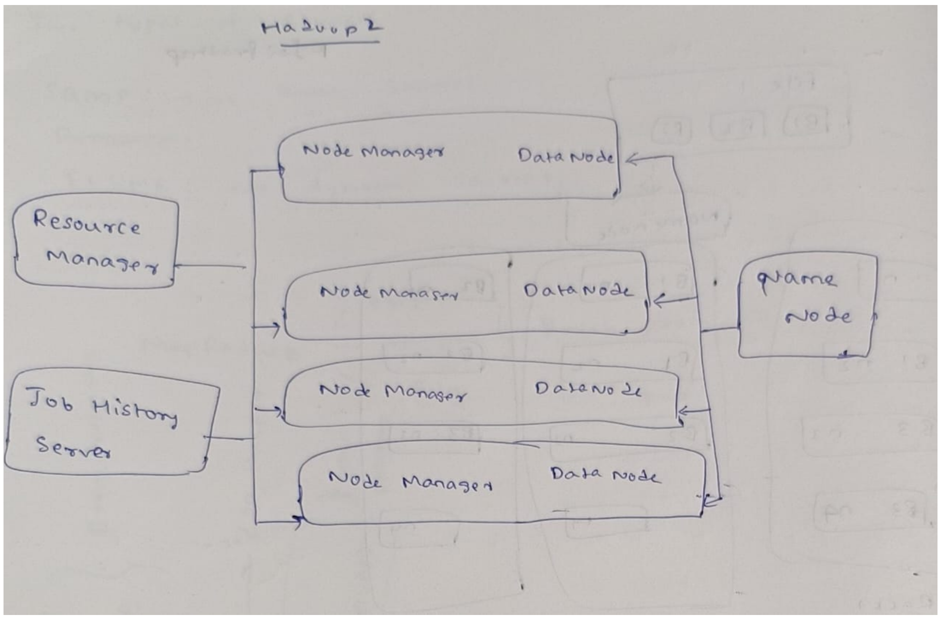
We use hive, pig, MRP

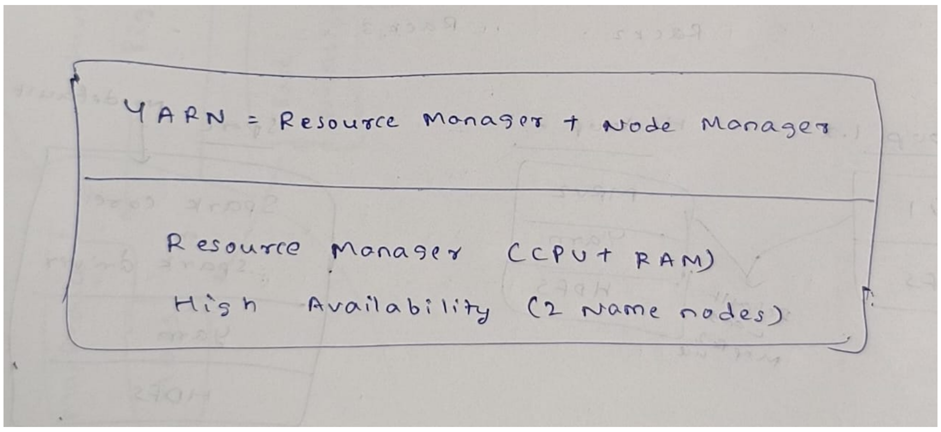
Each job has one application master (AM), if AM fails the job gets failed.

Parallelly these jobs get executed.

The number of map reducers = number of tables. For every table one map reducer is allocated.

Resource manager is the master machine in YARN.





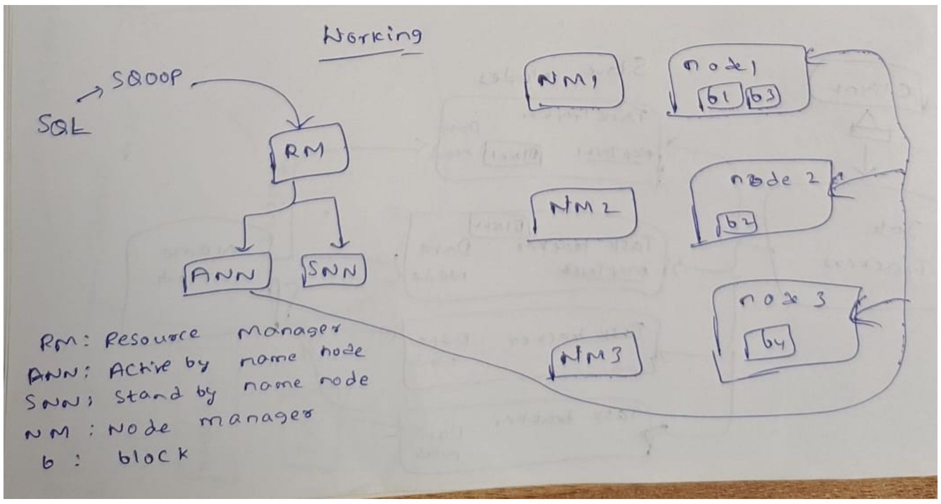
YARN = Resource Manager + Node Manager

Resource Manager (CPU + RAM)

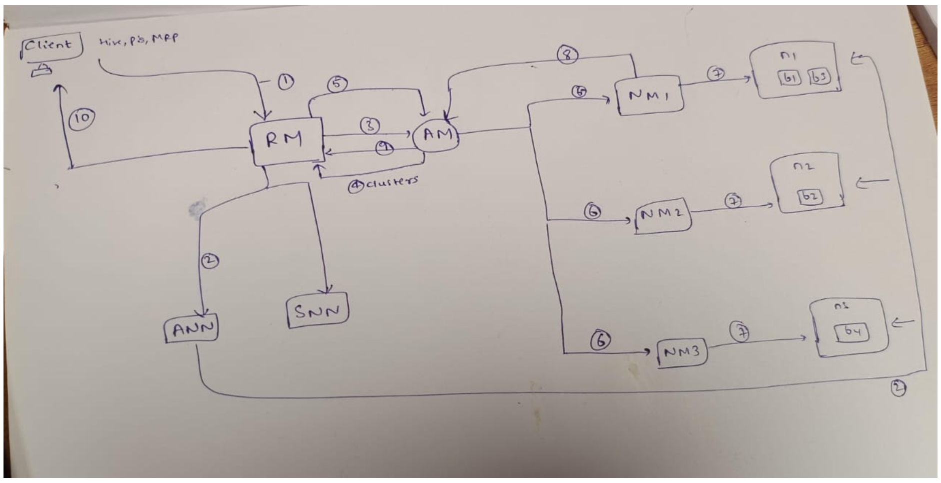
High Availability (2 Name nodes)

Note: -

1. When we run the ETL pipeline in Hadoop cluster(Sqoop) there is a 4 default Map task (m 4 command option) will be executing and zero reducers will be executing in backend.
2. When doing analysis in hadoop cluster number of blocks = number of map tasks executing and 1 reducer by default.



RM : Resource Manager
ANN : Active by name node
SNN ; Stand by name node
NM : Node manager
b : block



Process

1. Client (DE) request is sent to Resource manager (RM)
2. The RM will give the task to Active by name node(ANN), ANN will return the meta data
3. RM will create Application Master(AM) and assign this job to it
4. The AM will request the resources to RM,
5. RM gives the containers (CPU + Memory) to AM
6. AM will divide the task into smaller parts and gives the Map Tasks to the node managers
7. The node managers(NM) will now give the task to nodes
8. After completing the execution, NM will return the task to AM
9. AM will now give the collected tasks to RM
10. RM will give the result to client.

Containers = Resources = Memory + Processor. These containers will be allocated by resource manager depending on the size of the file.

Task = 1 block execution, small part of the entire job.s

After the completion of the task the resources are given back to Resource manager by the node managers.

The application master is present in any one of the node managers until the completion of the job.

To schedule the jobs (ETL pipelines and analysis), we use Air Flow.

Hadoop Commands

Hadoop commands:

1. To open Hadoop:
   a. Hadoop fs
2. List:
   a. Hadoop fs - ls
3. Remove directory:
   a. hadoop fs -rmr <diraname>
4. Move LFS (Local file system) to HDFS:
   a. hadoop fs -put /home/training/rama.txt / user/training/moresalesdata
   b. hadoop fs -copyFromLoacal
5. Read data in HFS:
   a. hadoop fs -cat moresalesdata/rama.txt
6. Get the data into LFS from HFS:

   a. hadoop fs -get moresalesdata/titanic_data.csv /home/training/miless.csv
   b. hadoop fs -copyToLocal

      it copies the data of titanic_data.csv in hadoop into local(LFS) miless.csv file

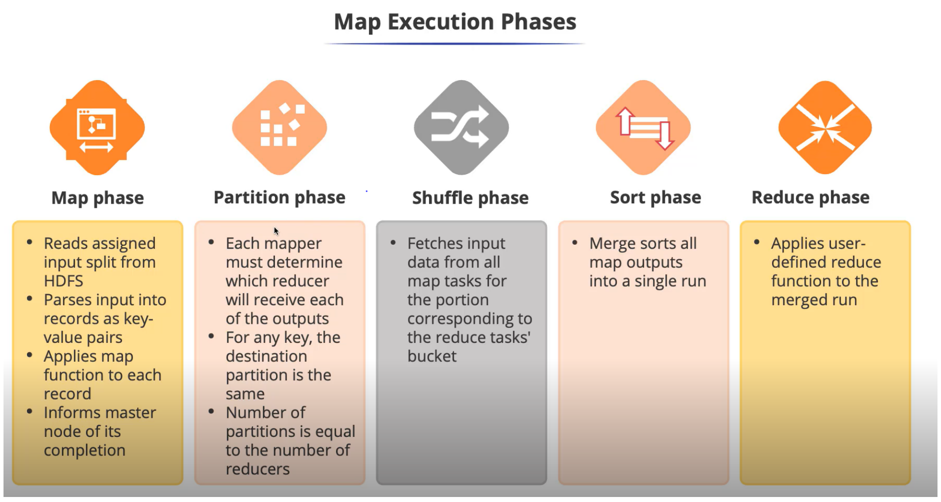
7. To see the node data in CLI:
   a. hadoop dfsadmin –report
8. To see the health status of blocks:
   a. hadoop fsck / -blocks
9. To check whether safe mode is on or off:
   a. hadoop dfsadmin -safemode get
   b. enter to enter into safe mode
   c. leave to leave from safemode
10. To see the statistics of the files:
    a. hadoop fs -stat %r moresalesdata ---> It shows the number of replications made in this
    b. hdfs fsck file_name -files -locations –blocks we can see the number of blocks created, replications, locations of this file.

Linux Commands:

1. Cat: >, <
2. Vi
3. Gedit
4. Mkdir
5. Move all the similarly named files: mv filename* /dir

Map Reduce

1. MapReduce is a programming model that simultaneously processes and analyses huge datasets logcally into separate clusters.
2. If you want to analyze the data in distributed file system. We use these two functions Map and Reduce.
3. Map function splits the data by using the delimiter. The input and output of the map function is both key value pairs.
4. Output of the Map function is the input of the reduce function. The reducer aggregates the data and returns. Reducer output is the final output and it is stored in HDFS.
5. Process:
   a. Blocks -> Map -> shuffle -> Sort -> Reduce
6. MapReduce Data type: long writable, int writable
7. printing in MapReduce: output.Collect ()
8. Default number of reducers = 1

9. 

Map Execution Phases

Map phase
- Reads assigned input split from HDFS
- Parses input into records as key-value pairs
- Applies map function to each record
- Informs master node of its completion

Partition phase
- Each mapper must determine which reducer will receive each of the outputs
- For any key, the destination partition is the same
- Number of partitions is equal to the number of reducers

Shuffle phase
- Fetches input data from all map tasks for the portion corresponding to the reduce tasks' bucket

Sort phase
- Merge sorts all map outputs into a single run

Reduce phase
- Applies user-defined reduce function to the merged run

10.

Running the map reduce program

Step 1: Set the Build Path of hadoop to Java

Step 2: Run the configuration file

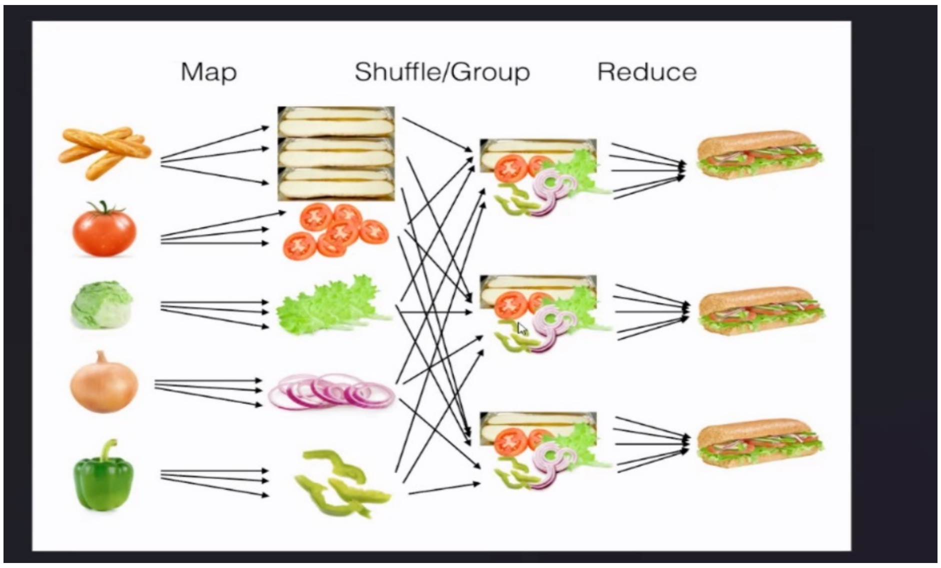
Step 3: Create the jar and export the jar into local

Step 4: Using the program by passing an input file:

hadoop jar jarfilename.jar dirname/inputfilename output filename

hadoop fs –ls outputfilename

hadoop fs –cat outputfilename/part-00000



Advantages of MapReduce

1. Data is processed in parallel
2. Data Locality (We process the data where it is)

MapReduce Programs

3 types of modes:

1. Stand alone: works in local machine, no hdfs, no configurations, no master slave architecture, just used for debugging
2. Pseudo-distributed mode: One machine will work as master and slave. It is known as single node cluster. Both name node and data node are present in the same machine. Mainly used for testing. All the daemons (Name node, data node,

secondary name node, resource manager, node manager) will be running in the machine.

3. Fully distributed mode: This is used in production purpose. Here the work is fully distributed across all the machines. It fully supports the master slave architecture. Here, each daemon is present in each individual machine.

Hadoop configuration files

1. hadoop-env.sh: Environment variables that are used in Hadoop scripts
2. core-site.xml: Configuration settings like I/O that are common to HDFS and MapReduce
3. hdfs-site.xml: Configuration for HDFS daemons the name node and secondary name node
4. mapred-site.xml: Configuration files for MapReduce daemons, Job tracker and Task tracker
5. master: Contains list of machines that run name node and secondary name node
6. slave: List of machines that run a data node and a task tracker

hadoop jar /home/cloudera/Documents/hadoop-streaming-2.7.3.jar -input /user/saiprathap/word_count_py/word_count.txt -output word_count_py/output.txt -mapper /home/cloudera/Documents/mapper.py -reducer /home/cloudera/Documents/reducer.py

Combiners: Mini reducers, help optimize the data transfer between the Mapper and Reducer phases by performing a preliminary reduction of data with the same key on the Mapper nodes.

Partitioners: determine which Reducer will receive the data for each key.

Counters: It is a function used to gather statistics about the MapReduce Job.

Structured Data analysis using MapReduce programs

Using Java:

1. Open eclipse and write code for the map and reduce programs
2. Run the configuration file as java application
3. Export the jar file and save it to a directory
4. use this jar file and write the hadoop jar command like this:
    a. hadoop jar jarfilename.jar [local file] dirname/inputfilename [hdfs file] output filename [hdfs file]

Using Python:

1. Write programs for map and reduce.
2. Check the programs using the pipe in Linux terminal
3. put the jar file (download from geeks for geeks) and put it into the local file system
4. Now run the hadoop jar command like this:
    a. hadoop hadoop-streaming-2.7.3.jar[LFS] -input word_count.txt [HDFS] -output output.txt [HDFS] -mapper mapper.py [LFS] -reducer reducer.py [LFS]

Sqoop

(SQ)L + Had(oop)

It is a tool used to transfer bulk data between HDFS & relational database servers

It works on top of YARN, so it provides parallelism.

Features of Sqoop:

1. Full Load: Loads the entire data from the database in one command
2. Incremental Data: Loading Data from the database as it gets updated.
3. It provides parallelism, it works on top of YARN
4. It provides compression by using gzip algorithm
5. Kerberos Security authentication
6. Load data directly into Apache hive for analysis

IMPORT

1. MySQL –u root
2. Sqoop import –connect jdbc:mysql://localhost/employees –username root –table employees
3. Go to localhost:50070(default) and check the folders

To import data into a specific directory

1. Sqoop import –connect jdbc: mysql://localhost/employees –username root –table employees –m 1 – target-dir / employee10
   a. sqoop import --connect jdbc: mysql://10.10.205.153:3307/hr?characterEncoding=latin1 --username sai --password password --table employees --target-dir mores

To import data by filtering it on the way

1. Sqoop import –connect jdbc:mysql://localhost/employees –username root –table employees –m 3 –where “emp_no > 49000” – target-dir / employee10

To import all the tables:

1. Sqoop import-all-tables - -connect jdbc:mysql://localhost/employees - -username root
   a. Worked: sqoop import-all-tables --connect jdbc:mysql://10.10.205.153:3307/hr?characterEncoding=latin1 --username sai --password password --warehouse-dir mores

Incremental import:

1. sqoop import --connect jdbc:mysql://10.10.205.153:3307/rand?characterEncoding=latin1 --username sai --password password --table trips -m 1 --target-dir trips --incremental append --check-column id --last-value 5

Filter and import results:

1. sqoop import --connect jdbc: mysql://10.10.205.153:3307/rand?characterEncoding=latin1 --username sai --password password --table trips -m 1 --target-dir trips --where "id<5"

Importing specific columns:

1. sqoop import --connect jdbc:mysql://localhost:3306/training --table vandana --columns "id, name" --target-dir vandana

Querying:

1. sqoop eval --connect jdbc:mysql://locahost:3306/training --query "select * from increment";

Updating:

1. sqoop eval --connect jdbc:mysql://localhost:3306/training --query "update abc set name = 'Sai Prathap' where id = 123";

EXPORT

It is mandatory that the table structure exists in SQL (target)

To see the command used to create the table use: show create table table_name;

Command to export

1. Sqoop export - - connect jdbc:mysql://localhost//employees - - username root - - table employees
   a. sqoop export --connect jdbc:mysql://10.10.205.153:3307/rand?characterEncoding=latin1 --username sai --password password --table users --export-dir more
   b. sqoop export --connect jdbc:mysql://10.10.205.153:3307/rand?characterEncoding=latin1 --username sai --password password --table users --export-dir more - -input-fields-terminated-by ‘,’

List Databases

1. Sqoop list-databases –connect jdbc:mysql://localhost/employees –username root
   a. sqoop list-databases --connect jdbc:mysql://10.10.205.153:3307?characterEncoding=latin1 --username sai --password password

List tables

1. sqoop list-tables --connect jdbc:mysql://10.10.205.153:3307/rand?characterEncoding=latin1 --username sai --password password

Sqoop commands:

1. Import
    a. Source: MySQL >>>>>>>>>>>>>>>>>>>>>>>>>>>> HDFS
        i. username: root
        ii. Password:
        iii. MySQL server: IP address
        iv. Driver: JDBC
        v. Port: 3306
        vi. Database name: training
        vii. Table: countries
    b. To import all tables, use warehouse-dir instead of target-dir
    c. If there is no primary key and we want to split the data into some (by default 4) parts in the HDFS system, we can use --split-by
        i. sqoop import --connect jdbc:mysql://localhost:3306/retail_db --username root --password cloudera --table order_items --split-by order_item_product_id --target-dir /user/hdfs/order_items
    d. There are no reducers will be executing while doing an import or export. Reducers only work during analysis only.
2. Export
3. Incremental (append & lastmodified)
4. eval

Creating Jobs

Incremental Append:

1. Creating Job
    a. sqoop job --create incrAppend -- import --connect jdbc: mysql://localhost:3306/training --username root --table increment --target-dir increment --incremental append --check-column id -m 1
    b. If you don’t want to create job then, you need to mention –last-value option
2. List jobs:
    a. sqoop job - - list
3. See the details of the job:
    a. sqoop job --show incrAppend
4. Execute the job:
    a. sqoop job --exec incrAppend

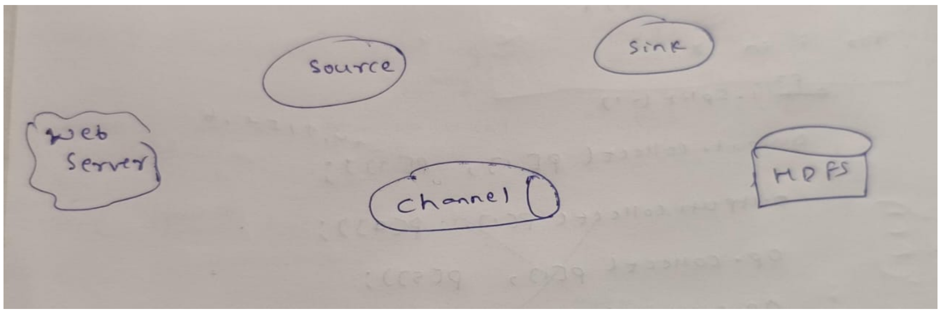
Incremental Last Modified:

1. Creating job:
    a. sqoop job --create incrementLM -- import --connect jdbc:mysql://localhost:3306/training --username root --table incrementLM --target-dir incrementLM --incremental lastmodified --check-column time -m 1 –append
    b. If you don’t want to create job then, you need to mention –last-value option
    c. When you create a job for incremental, the last value is stored.
2. Generating Codegen:
    a. sqoop codegen --connect jdbc: mysql://localhost:3306/training --username root --table incrementLM --outdir /home/training/last_modified
3. Merging the files to get update:
    a. sqoop merge --new-data /user/training/incrementLM/part-m-00001 --onto /user/training/incrementLM/part-m-00000 --target-dir incrementLMmerged --jar-file /tmp/sqoop-training/compile/4b5fec17d7ff321b3ee7afd3f94c1fbc/incrementLM.jar --class-name incrementLM --merge-key id

b. You can see this jar file location while compilation of the sqoop codegen

c. –new-data, --onto, --jar-file, --class-name, --merge-key

FLUME



It is used to bring live data from web servers. It brings data in unstructured data. As it is loading data from a dynamic source, flume will not stop automatically like sqoop.

The channel is buffer, it is RAM. The data is stored temporarily before going to HDFS.

Agent file is a configuration file, it has: source + channel + Sink

We can run the agent file using the command FLUME –ng

Flume puts the data in HDFS but Kafka puts data into NoSQL.

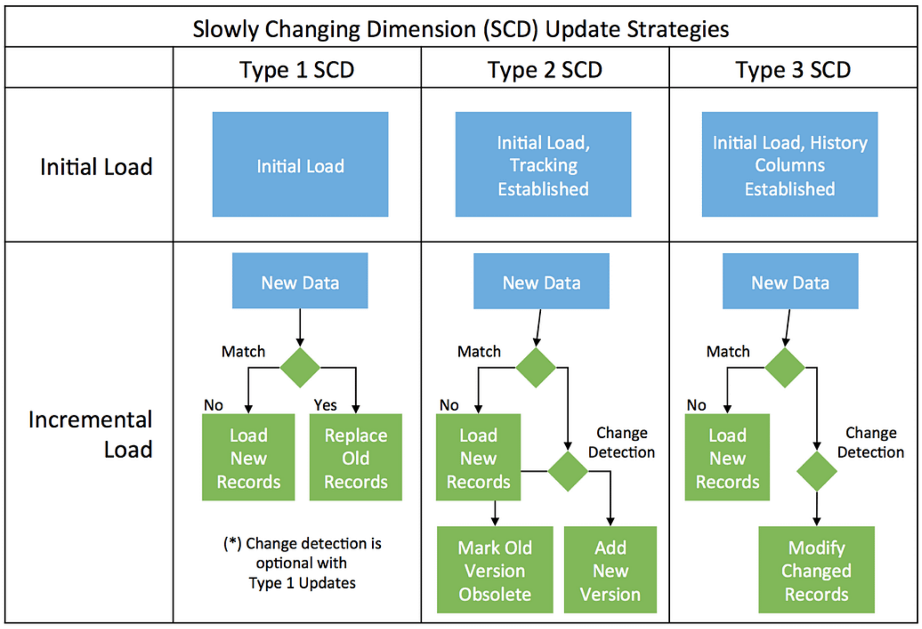
Slowly changing dimensions:

SCD1: Historical data is not maintained

SCD2: Historical data is maintained by storing new(updated) record.

SCD3: Loading the data into as new record by changing some columns and keeping a flag.

These strategies will not work for ETL pipeline while bringing the live data. Once data is stored in Big data database we can apply these strategies.



HIVE

Documentation: https://hive.apache.org/

Course: https://www.simplilearn.com/learn-hadoop-spark-basics-skillup?utm_campaign=Hadoop-rr17cbPGWGA&utm_medium=Description&utm_source=youtube

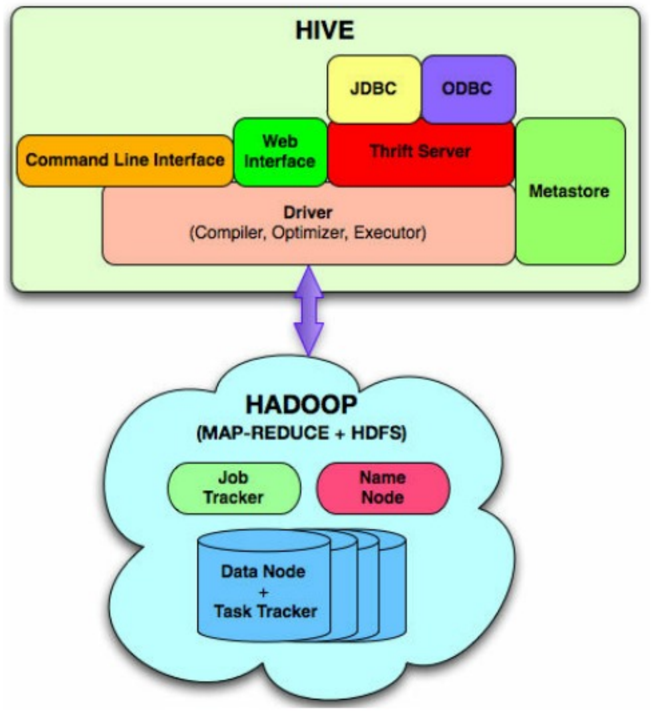
Hive is a data warehouse on Hadoop. It is the first Big data warehouse. It is open source.

Database and Data Warehouse:

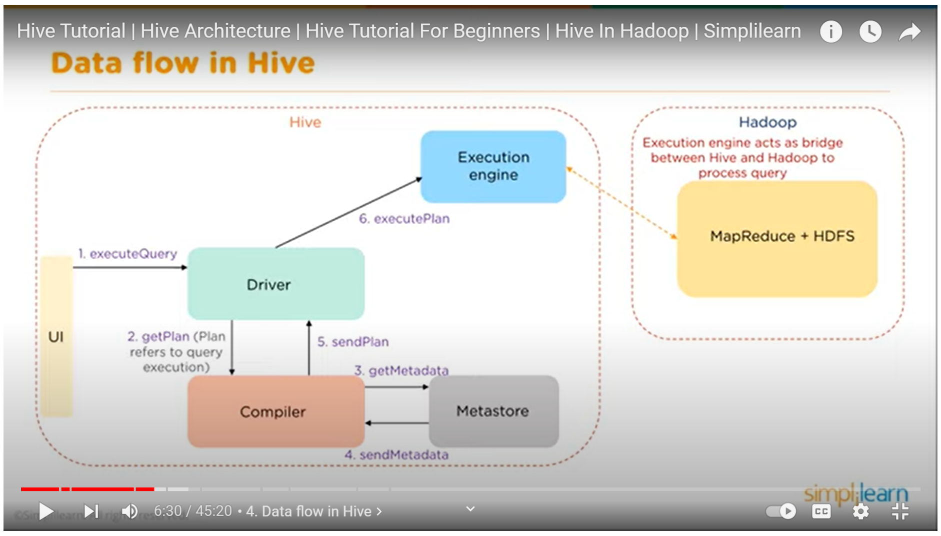
1. A data warehouse is used for decision making purpose. It stores historical data.
2. Traditional data warehouses are not built on top of DFS. It can handle only TBs of data. They are based on single server.
3. Data warehouse is OLAP used for analysis purpose. Databases are OLTP, they are used for application purpose.
4. Data warehouse in azure synaps, in aws red shift.

Hive is a dependent API, it depends on MapReduce programs and HDFS to store data.

Mainly HIVE is used to process structured data and sometimes semi-structured can also be processed.



1. **Thrift Server:** The Hive Thrift Server (HiveServer2) is a service that exposes Hive functionality through Thrift, a framework for building cross-language services. It allows clients to connect remotely and submit queries to Hive, making it possible to interact with Hive from various programming languages.
2. **Metastore:** The Hive Metastore is a centralized repository that stores metadata about Hive tables, partitions, and schemas. It contains information about table structures, column data types, storage locations, and more. The Metastore is essential for query optimization and schema management. It stores the schema of the tables stored in the HDFS. If you make any changes in the HIVE table, the changes also made effective in Meta store.


Hive Tutorial | Hive Architecture | Hive Tutorial For Beginners | Hive In Hadoop | Simplilearn
Data flow in Hive
Hive
Execution engine
6. executePlan
1. executeQuery
Driver
UI
2. getPlan (Plan refers to query execution)
5. sendPlan
3. getMetadata
Compiler
Metastore
4. sendMetadata
Hadoop
Execution engine acts as bridge between Hive and Hadoop to process query
MapReduce + HDFS
simplilearn
6:30 / 45:20 • 4. Data flow in Hive ›

When you write a HIVE query, the backend is converted into MapReduce in Hadoop cluster.

Hive Data types:

There are two types of data types: 1) Primitive 2) Complex datatypes

1. Primitive: int, float, double, long, string
2. Complex: Array, List, Structs

All sql commands work in hive except: insert, update, delete as per hive 2.2 because, it works on top of HDFS, so there is no modification of the data can be done on the data.

Hive 2.3 onwards it supports partial ACID properties.

We can create a table in hive but cannot insert data into it. We can just load the data into the table from HDFS. When you load the data into the warehouse, the data in the HDFS is deleted. If you want to see the data use hive only (Hive folder). When you load the data from LFS the data is not deleted.

So, if I want to get the data from any relational database, we can directly import the data into HIVE instead of HDFS but, in the backend the data is stored in the HDFS only.

Note: There are two joins map side join and reduce side join for internal optimization implemented by MapReduce engine. When one table is small and the other table is large, then the map side join gets executed. When both the tables are big then reducer side join is executed.

Working with HIVE from HDFS:

1. Put the data into hive
2. open hive console and create table using a command like this:
   a. create table house_survey (id int, city string, country string) row format delimited fields terminated by ',';
3. Load data into the table by using this command
   a. load data inpath '/user/saiprathap/House_Survey.csv' into table house_survey;
4. select * from house_survey;

Import data from MySQL:

1. Single table import
   a. sqoop import --connect jdbc: mysql://localhost:3306/retail_db --username root --password cloudera --table customers --hive-import -m 1
2. Import all tables
   a. Firstly, create a database in hive and use - -hive-database option
   b. sqoop import-all-tables --connect jdbc: mysql://localhost:3306/retail_db --username root --password cloudera --hive-import --hive-database retail_db -m 1

Storing the output of the queries into a table:

1. Create an empty table
2. Run this command:
    a. insert overwrite table op1 select name, sum (bill1 + bill2 + bill3 + bill4) from host group by name;
    b. when you use a cte:
        i. with cte as (select name, sum(bill3) as b from host group by name) insert overwrite table op3 select name, b from cte order by b desc limit 1;

Export data to MySQL:

1. create table in MySQL
2. Run this command
    a. sqoop export --connect jdbc: mysql://localhost:3306/hive --username root --password cloudera --table op1 --export-dir /user/hive/warehouse/op1/000000_0 --input-fields-terminated-by '\0001'

There are two types of tables in hive warehouse

1. Internal tables (Managed tables)
2. External tables (Non managed tables)

Internal tables:

1. They are also known as managed tables, because we have total control of the table.
2. The tables which are stored in warehouse are called as internal tables (Normal tables). We can manage everything like inserting, etc.
3. If you drop an internal table then file, table and metadata will be deleted.
4. It is recommended to use internal tables for analysis.

External table:

1. The table which are stored externally. This table can be created using external keyword.
2. When you drop a table, only the metadata will be deleted not the actual table.
3. The external tables are used to store the output of analysis done in Hive.
4. Command to create external table:
    a. create external table external_table (id int, city string) row format delimited fields terminated by ' ' location '/user/hive/warehouse/external_table/sample.txt';
    b. Now drop the table and check for the tables. The table will not be present but the file of the data will be present in the HDFS file system. So, use external tables only to store the output don't use it for input of analysis. Because, we cannot use it as a table, operations like incremental import etc. cannot be done on the external table.
5. Command to create external table in the location where the data is present.
    a. By using this type of creation, we need not load the data manually in the table.
    b. To create external table and put the data automatically we need to first put the data into HDFS (non hive folder) and create a table with specifying the directory of the file, not up to filename. Please note that, there should be only one file inside that folder specifying one table. If there are multiple files, then we cannot insert data automatically. We will get an error saying that, not a directory.
    c. hadoop fs -put sample.txt /user/cloudera/salesdata
    d. create external table myextb (id int, city string) row format delimited fields terminated by ' ' location '/user/cloudera/salesdata';
    e. Give the location up to directory level only not up to the file level.
    f. select * from myextb;
6. To create external table and store the output into it, the process is same as the insert overwrite of a normal table:
    a. Firstly, create an external table:
    b. create external table result1(pname string, totalbill int) row format delimited fields terminated by ' ' location '/user/hive/warehouse/result1';

   c. Then overwrite the result into that table:
      i. Insert overwrite table result1 select name, sum (bill1 + bill2 + bill3 + bill4) from host group by name;

Hive modes: Local mode (single node) and MapReduce mode (more than one node)

Hive Optimization Techniques

1. Hive partitions
2. Bucketing
3. File formats
4. Indexes

Hive Partitions:

1. It is one of the optimization techniques in hive to improve the performance of queries when the table is large and when you have to write multiple queries on that table.
2. We create partition table for the table that is already present in the hive. We can create partition table on both internal and external table but it is best to create partition on internal table.
3. There are two types of partitions dynamic and static partitions.
4. You need to decide which column has the high important to create a partition on. You can create partition based on more than one column. You should use the column which is used to partition in where, group by, etc. clauses in every query.
5. The partitioned table is stored as a directory in HDFS. Number of directories will be dependent on the number of distinct values of the partitioned column.
6. Note: Partitioning or any optimization technique is not allowed by default in hive, we need to activate it by using set commands.
7. In non-partitioned tables, hive would have to read all the files in a table's data directory and then apply filters. This is more time taking and expensive.
8. When a partitioned table is queried then only the relevant directories are read. If you filter data based on the column that is not used to partition, then it will be more expensive than a non-partition query. So, don't over partition the data, it will cause slow performance and it will be more burden for the name node.
9. Partitioning multiple columns create hierarchical directories.
10. Setting the partition
    a. SET hive.exec.dynamic.partition = true;
    b. SET hive.exec.dynamic.partition.mode = nonstrict;
    c. SET hive.exec.max.dynamic.partitions = 5000;
    d. SET hive.exec.max.dynamic.partitions.pernode=5000;

Types of partitions:

1. Static partitions:
   a. In static partitions we need to insert the data manually for each partitions.
   b. Say you are partitioning with city column; the you need to do this for each city (partition)
   c. insert into table table_name partition (city = 'Chennai') select name, age from original_table where city = 'Chennai';
2. Dynamic partitioning:
   a. It is automatic, we don't need to insert the data manually into each partition.
   b. All we do in this notes are dynamic partition only.

Partitioning process (Dynamic):

1. Creating table:
   a. create table part (doj string, room int, bill1 int, bill2 int, bill3 int, bill4 int) partitioned by (name string) row format delimited fields terminated by ' ';
2. creating partitions:

    a. insert overwrite table part partition(name) select doj, room, bill1, bill2, bill3, bill4, name from host;

3. It is good to maintain the last column as the partition column to identify the column.
4. To see the partitions on a table, the use: show partitions table_name;
5. Now open the file system and see the number of directories and files created.

Hive Bucketing:

1. In this we can mention the number of parts of the data we want to make.
2. The data is stored in file formats, not in the directories format.
3. The data is divided into each bucket based on the hashing algorithm, the result will be going to a particular bucket based on the hash value of the value of the column.
4. Bucketing is also not enabled by default, we need to run this command
    a. set hive.enforce.bucketing = true
5. Creating hive buckets
    a. create table y (id int, name string, country string) clustered by(country) into 9 buckets row format delimited fields terminated by ‘,’;
    b. insert overwrite table y select id, name, country from x;
6. Worked
    a. create table bucketBbsalesdataRegion (Region string, Country string, Item_Type string, Sales_Channel string, Order_Priority string, Order_Date string, Order_ID string, Ship_Date string, Units_Sold int, Unit_Price float, Unit_Cost float, Total_Revenue float, Total_Cost float, Total_Profit float) clustered by(Region) into 3 buckets row format delimited fields terminated by ' ';
    b. insert overwrite table bucketBbsalesdataRegion select region, country, Item_Type, Sales_Channel, Order_Priority, Order_Date, Order_ID, Ship_Date, Units_Sold, Unit_Price, Unit_Cost, Total_Revenue, Total_Cost, Total_Profit from bbsalesdata;
7. Sampling queries:
    a. When you want to query the results from only specific buckets then we use sampling queries.
    b. But the catch is you need to know which bucket you want to choose.
    c. select region, id from table_name TABLESAMPLE (bucket 3 out of 5 on region)
8. Decide the number of buckets = CEIL (log2(table_size/128))
    a. table_size is in MB

Partition and Bucketing at one go:

create table table_name (id int, amount int, country string) partitioned by (name string) clustered by(country) into 3 buckets row format delimited fields terminated by ','

Hive Indexing

The goal of hive indexing is to improve the speed of query lookup on certain columns of a table. Indexes are pointers on a particular column

Two types of indexing in HIVE:

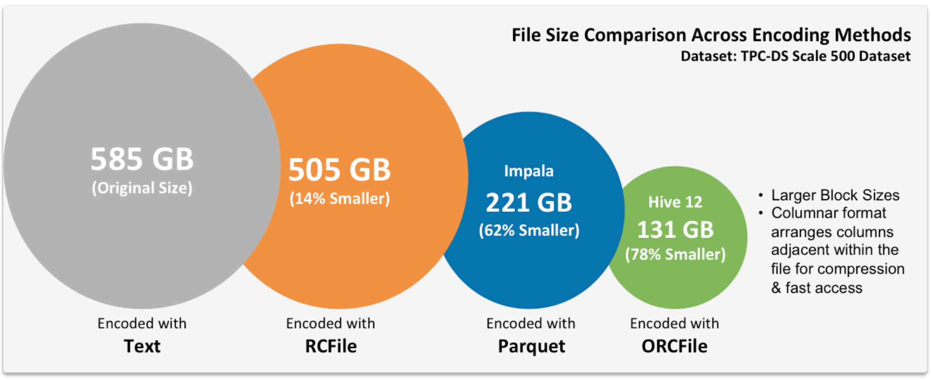
1. Compact index (most commonly used): Very less buffer ram in hive warehouse no matter the size of the data
2. Bit map index: Takes very high buffer ram.

It is best practice to delete the index after analysis, because it takes the buffer memory. We cannot do indexing in partitioned and bucketed tables

Worked:

1. create index index_origin on table airlines(origin) as 'org.apache.hadoop.hive.ql.index.compact.CompactIndexHandler' with deferred rebuild;
2. show formatted index on airlines;
3. drop index index_origin on airlines;

File Formats



1. Parquet is compressed with Snappy, ORC file is compressed with Gzip. Before RC file Avro file was there, it produces only 8% compression.
2. Compressing:
   a. Text:
      i. beeline
      ii. ! connect jdbc:hive2://localhost:10000
      iii. create external table xyz (id int, name string, amount int) row format delimited fields terminated by ',' stored as textfile location '/user/saiprathap/cat' tblproperties ("skip.header.line.count"="1");
      iv. You can also create an internal table; it is not mandatory that the table is external. Stored as text file is not mandatory to put, y default it is stores in text file
      v. show tblproperties xyz;
   b. Avro File:
      i. beeline
      ii. create table my_avro (id int, name string, age int) stored as avro;
      iii. insert overwrite table my_avro select * from xyz;
      iv. select * from my_avro;
   c. Parquet file:
      i. beeline
      ii. create table my_parquet (id int, name string, amount int) stored as parquet;
   d. ORC file:
      i. beeline
      ii. create table my_orc (id int, name string, amount int) stored as orc;
   e. Now check the storage of these tables, parquet takes very less storage than others.
   f. Do the above process for a large table?

Semi Structured data analysis:

1. We use serde jar to use semi structured data analysis with hive. SERDE: serialization and De serialization, it is used to convert semi structured to structured data (in hive, not in spark)
2. Download the jar file (it is there in Big data folder) into the LFS
3. now add the jar file to hive using the command
   a. ADD JAR json-serde-1.3.7.jar;
4. create table json_nested (country string, languages array<string>, religions map<string, array<int>>) row format SERDE 'org.openx.data.jsonserde.JsonSerDe' stored as textfile;

5. load data inpath '/user/saiprathap/nestedtext.txt' into table json_nested;

Hive Joins

Two types of joins for join queries they are: i) map side join ii) reduce side join

When one table is small and the other is large then internally map reduce engine executes map side join to join these tables. When the two tables are large then the reduce side join is used internally.

In map side join the small table is taken in the cache of the memory (distributed cache)

Worked:

Two data sets used: NYSE_daily.txt, NYSE_dividends.txt

1. create table NYSEdailyhead (stexchange String, stock_symbol String, stock_date String, stock_price_open double, stock_price_high double, stock_price_low double, stock_price_close double, stock_volume int, stock_price_adj_close double) row format delimited fields terminated by "\t" lines terminated by "\n" tblproperties("skip.header.line.count"="0");
2. create table nyse_dividends (divstock_exchange STRING, divstock_symbol STRING, divstock_date STRING, dividends FLOAT) row format delimited fields terminated by '\t';
3. Load data into the tables
4. Map side join:
    a. select /*+MAPJOIN(nyse_dividends) */ * from nysedailyheads a join nyse_dividends b on a. stock_symbol = b. divstock_symbol and a. stock_date = b. divstock_date where a. adj_close <= 20;
    b. The above command has +, so the comment is excluded

Hive commands

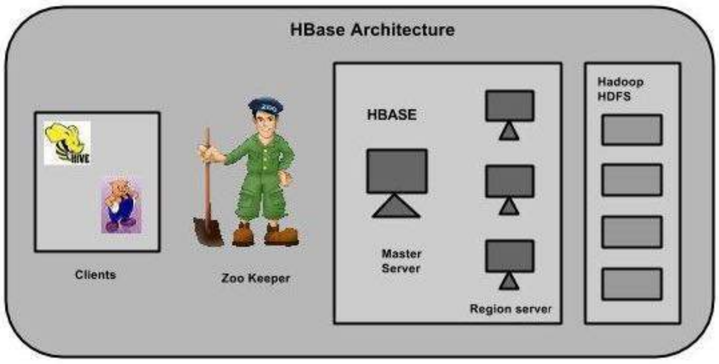
1. create table table_name (col datatype, …) row format delimited fields terminated by ‘,’ tblproperties (“skip.header.line.count” = ”1”);
2. show tables
3. describe table_name
4. load data inpath '/user/saiprathap/House_Survey.csv' into table house_survey;
5. insert overwrite table op1 select * from house_survey;
6. External table:
    a. create external table external_table (id int, city string) row format delimited fields terminated by ' ' location '/user/hive/warehouse/external_table/sample.txt';
7. Partition:
    a. create table part (doj string, room int, bill1 int, bill2 int, bill3 int, bill4 int) partitioned by (name string) row format delimited fields terminated by ' ';
    b. set hive.exec.dynamic.partition=true;
    c. set hive.exec.dynamic.partition.mode=nonstrict;
    d. insert overwrite table part partition(name) select doj, room, bill1, bill2, bill3, bill4, name from host;
8. Bucketing:
    a. create table y (id int, name string, country string) clustered by(country) into 9 buckets row format delimited fields terminated by ‘,’;
9. File formats:
    a. create table xyz (id int, ...) stored as parquet/ avro/ textfile

HBASE

1. Database: OLTP, Transactional, SQL
2. Data warehouse: OLAP, Historical, Hive
3. Big Data Database: OLAP,Transactional and analytical, NoSQL
4. Big data warehouse: Data lake

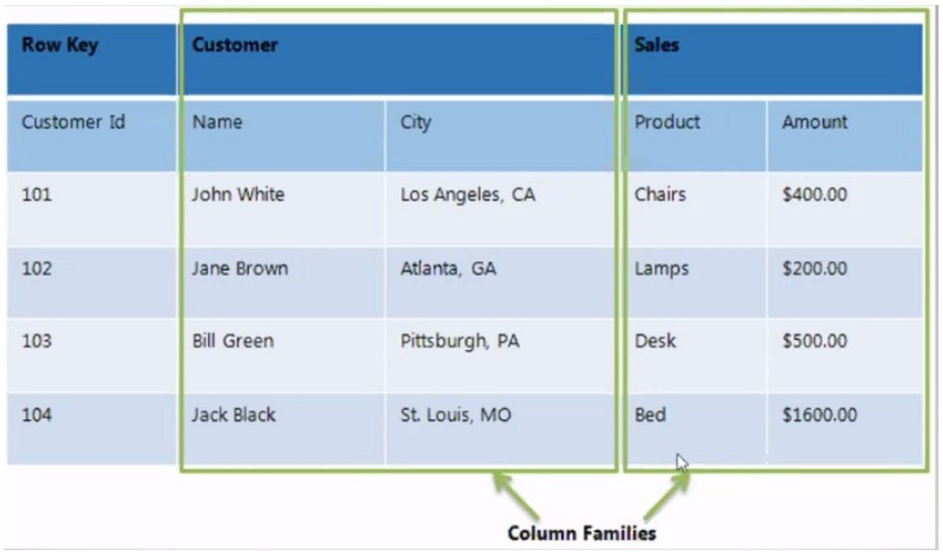
5. Data Lake: Data is a centralized repository that allows to store vast amounts of structured and unstructured data at any scale.
6. Delta Lake: Data lake + ACID properties
7. Delta Lakehouse: Data lake + data warehouse
8. Direct lake = Delta lake + BI tools
9. NoSQL Technologies: HBase, MongoDB, Cassandra, Couch DB, Neo4J. In AWS cloud: Dynamo DB, Azure: Cosmos DB.
10. HBase is the first NoSQL database in this IT world.
11. A data warehouse cannot be used as a backend for application. We can use this NOSQL as a backend for an application and analysis as well.
12. HBase is OLAP, it is column oriented DB. It can store any large volume of structured and semi structured data.
13. NoSQL is schema less. It stores the data as key value storage.
14. HBase is an independent API, it does not depend on MapReduce.
15. HBase = HDFS + HBase API
16. HBase API architecture is master-slave.
17. It has HBase query language.
18. It has support to connect with BI tools.

HBase Architecture



HBase master server: It is like name node. It stores the metadata of every table.

The master server sends the request to the region server and the region server executes the task.



Row Key	Customer		Sales	
Customer Id	Name	City	Product	Amount
101	John White	Los Angeles, CA	Chairs	$400.00
102	Jane Brown	Atlanta, GA	Lamps	$200.00
103	Bill Green	Pittsburgh, PA	Desk	$500.00
104	Jack Black	St. Louis, MO	Bed	$1600.00

Column Families

Region servers maintain column families as regions. There can be any number of column families.

In HBase, tables are split into regions and are served by the region servers. Regions are vertically divided by column families into “Stores”. Stores are saved as files in HDFS.

Tables, Rows, Column families, columns, Cells: value, versions, timestamp

HBase commands: Create (create table with column families), Delete (delete specific rows), list, put (insert update), get (select), scan (retrieve complete table data), enable, disable, describe, drop.

The region server manages all the column families created. When you create a table the table by default it is enabled, if you want to delete the table then you need to first disable it.

We cannot have random read and write in HDFS, in HBase we can have random read and write.

Importing from MySQL

1. For Creating table with one column family
    a. sqoop import --connect jdbc: mysql://localhost:3306/retail_db --username root --password cloudera --table categories --hbase-table 'categories' --column-family category_details --hbase-create-table --columns category_id,category_department_id,category_name --hbase-row-key category_id -m 1
2. For creating multiple column families
    a. sqoop import \
    b. -connect jdbc:mysql://localhost:3306/retail_db \
    c. -username root \
    d. -password cloudera \
    e. -table categories \
    f. -hbase-table 'categories' \
    g. -hbase-create-table \
    h. -columns category_id,category_department_id,category_name \
    i. -hbase-row-key category_id \
    j. -hbase-column-mapping category_id:category_details,category_department_id:additional_details \
    k. m 1

Importing data from HDFS flat file:

1. Create the table in HBase and mention the required column families
    a. create ‘sample’ ‘personal’
2. Now write this command in normal terminal not in HBase
    a. hbase org.apache.hadoop.hbase.mapreduce.ImportTsv
    b. Dimporttsv.separator=',' -Dimporttsv.columns="HBASE_ROW_KEY,personal:lname,personal:age"
    c. sample
    d. '/user/saiprathap/sample.csv'
3. Check the data in HBase. The first column in the file is automatically taken as the row key.
4. If you want to change the row key column, then place the HBASE_ROW_KEY in the required position of the column. (If you want to metion the 2nd column as row key then first mention the first column and then mention HBASE_ROW_KEY in second place).
    a. Dimporttsv.columns= "personal:fname, HBASE_ROW_KEY, personal:age"

Hive table to HBase

1. HBase is not suitable to perform complex queries, so we integrate HBase with Hive
2. Hive HBase integration with HBase storage handler with serde properties
3. Worked:
    a. Create an HBase recognized table in Hive with the column families mentioned in the HBase table

        i. create table hbase_table(key string, city string, cost int) row format delimited fields terminated by ',' stored by 'org.apache.hadoop.hive.hbase.HBaseStorageHandler' with serdeproperties ("hbase.columns.mapping"=":key,location:city, price:cost") tblproperties("hbase.table.name"="hbase_demo");

    b. Now load the data into new hive table using a normal table, it will be automatically reflected inside the HBase table
        i. insert into hbase_table select * from hbase_demo;
    c. Check the data in HBase, a new table is created in HBase, the data will be there automatically in the HBase
    d. Don't mention the key column in mapping, the left out column is automatically taken as row key.
        i. create table hbase_table(key string, city string, cost int) row format delimited fields terminated by ',' stored by 'org.apache.hadoop.hive.hbase.HBaseStorageHandler' with serdeproperties ("hbase.columns.mapping"="location:city, price:cost") tblproperties("hbase.table.name"="hbase_demo");

4. The Hive table and HBase table are both synchronized i.e., what ever the changes made in the hive are reflected in HBase table. If you drop table in Hive then the table in HBase is also deleted.

HBase table to Hive

1. Create an external table in Hive
    a. create external table hbase_cars(key int, name string, price int, rating int, year int, color string, brand string) stored by 'org.apache.hadoop.hive.hbase.HBaseStorageHandler' with serdeproperties("hbase.columns.mapping"="details:name, details:price, details:rating, details:year, details:color, details:brand") tblproperties("hbase.table.name" = "cars");
2. This automatically brings the data into hive table from HBase

HBase commands

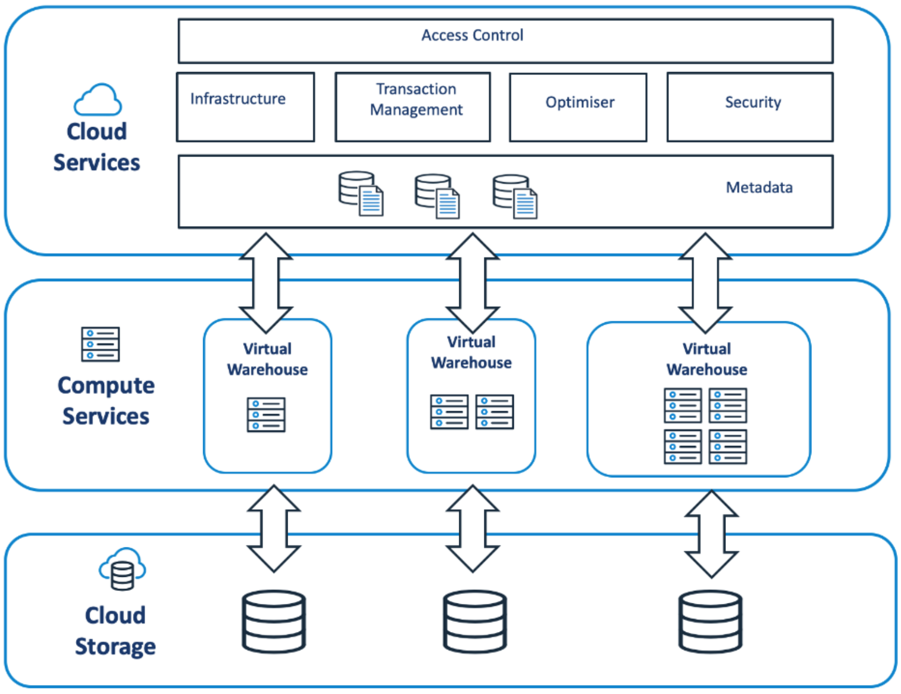
1. list: to list out tables.
2. scan table_name: to print the entire table.
3. create 'table_name', 'column_family_name': To create a table with Column families, you can mention any number of column families separated with comma.
4. describe 'table_name': to get the details of the table.
5. disable 'table_name'
6. drop 'table_name'
7. Inserting:
    a. put 'cars', '123', 'details:name', 'Brezza'
8. Updating:
    a. put 'cars', '123', 'details:name', 'Swift'
9. Delete the data from the table:
    a. delete 'cars', '123', 'details: price'
10. To get the data from a specific row and a specific column
    a. get 'host', '3', {COLUMN=> 'patient_details: name'}
11. Getting some columns:
    a. get 'host', '1', {COLUMN => ['bill_details: rBill', 'bill_details: mBill']}
12. Getting some columns within a time
    a. range get 'host', '1', {COLUMN => ['bill_details: rBill', 'bill_details: mBill'], TIMERANGE => [1694430107619,1694430156228]}
13. Limit:
    a. scan 'categories', {'LIMIT' => 5}
14. Setting the number of versions to be maintained:
    a. alter 'cars', {NAME => 'details', VERSIONS => 100}
15. Getting the number of versions:

    a. scan 'cars', {VERSIONS => 3}

16. Getting the versions in get:
    a. get 'cars', '123', {COLUMN => 'details:price', VERSIONS => 3}
17. Selecting some columns and time range:
    a. scan 'cars', {COLUMN=>['details:name','details:color'] ,TIMERANGE => [1694428574271, 1694428666332]}
18. Adding a column family:
    a. First disable the table
    b. alter 'cars', {NAME => 'xy'}

Snowflake

1. There are two data models are there, star schema and snowflake schema. Star schema allows limited amount of normalization. But snowflake allows high normalization.
2. Snowflake is a data warehouse. It is also a data model (Snowfalke schema).
3. RDMS like MySQL use star schema, but snowflake uses snowflake schema.
4. Snowflake is good in performance, when having multiple table and large volume of data.
5. Snowflake uses very less storage.
6. Every table by default stores as columnar storage. By default data is compressesd into some format like ORC.
7. You can use snowflake to analyse structured, semi-structured and unstructured data.
8. It is completely a cloud based data warehouse.



Snowflake Architecture

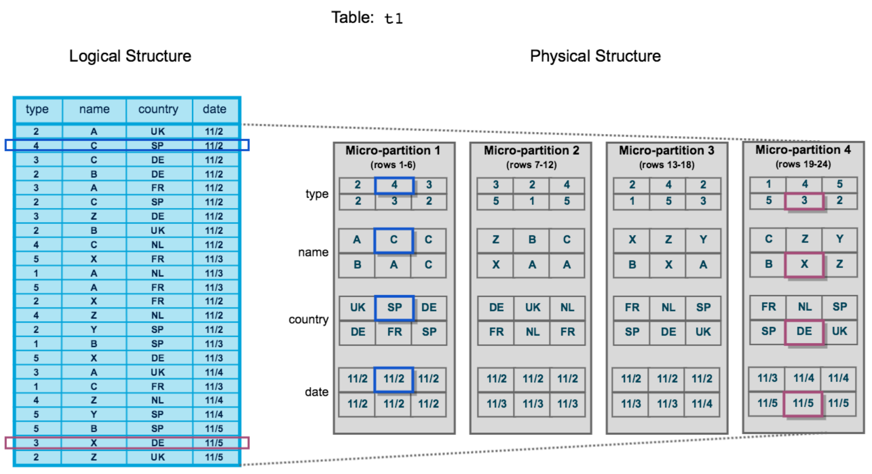
1. Fact Table: Most of the times fact table is used to store numerical data.
2. Dimension table: Every dimension table is related to fact table.
3. We can do analysis in snowflake using both SQL and Python.
4. *One cluster can have only one warehouse.
5. It processes the data in disk, if you want to use memory processing we can use snowflake connector and connect to spark to perform processing in memory.

6. There are 3 layers in the snowflake:
    a. Storage Layer (AWS – S3, Azure – Datalake, GCP – GCS)
    b. Virtual warehouse: Compressed column
    c. Access control layer
7. Snowflake architecutre is also called as multi cluster shared disk architecture.

Differences between Star schema and Snowflake schema

Star Schema	**Snowflake schema**
Partially normalization design, because only one dimension schema.	Highly normalized schema, because multiple sub dimension schema.
It requires more storage	It uses less storage space, because of default compression
Scalability is limited	High scalability
It is always good for simle queries	It is good for complex queries
Limited volume of data	High volume of data

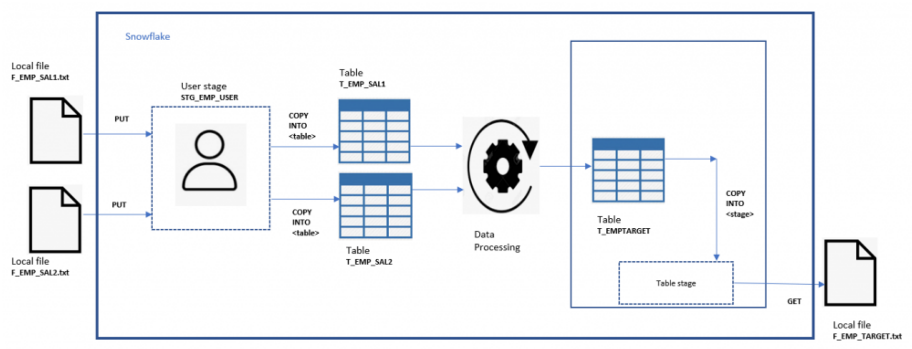
Micro Partitions:

1. When you upload a 500MB data into snowflake it takes 50MB. Even this 50MB data is taken into micro partitions.
2. In micro partitions, the data will be divided into blocks (column-wise) in physical structure of the snowflake data warehouse.
3. If a table has 24 rows and 4 columns, then it is can be divided into 4 micro partitoins. Each micro partition can contain 4 rows, each column will be divided into separate parts and placed in different places.
4. 
Table: t1
Logical Structure
Physical Structure
type
name
country
date
Micro-partition 1 (rows 1-6)
Micro-partition 2 (rows 7-12)
Micro-partition 3 (rows 13-18)
Micro-partition 4 (rows 19-24)

Bringing data into Snowflake:

Two types of stages for data transfer:

1. Internal stage: Loading the local file to snowflake
    a. Internal table staging working



   b. Here the stages act like extraction
   c. Use *put* command to put the table into the staging table.
   d. Then we load the data into snowfalke warehoues using the *COPY INTO* injestion service
   e. Steps to create stage and load data
      i. create stage:
         1. *create stage product_stage;*
      ii. put file in stage:
         1. *put file://C:\Users\Futurense\product_data.csv @product_stage;*
      iii. create table in snowflake database
      iv. copy data into table:
         1. *copy into products from @product_stage/product_data.csv file_format=(type='csv',skip_header=1);*
      v. After the completion of the loading we need to delete the data otherwise we will be charged money for the staging data also
         1. *drop stage product_stage*;
      vi. Copying data from snowflake to local
         1. create stage op_stage;
         2. copy into @op_stage/op1 from (select * from products where product_id = 1);
         3. list op_stage;
         4. get @op_stage/op1_0_0_0.csv.gz file://D:;

2. External stage
   a. We create external stages to load data from cloud.
   b. for AWS open the pdf: snowflake external stage.pdf
   c. For Azure: C:\Users\Futurense\Futurense Training\BigData\ Azure BLOB Storage to snowflake.word
      i. External stage
      ii. Snowpipe (Ingestion service)
      iii. Unloading
      iv. Data Sharing
      v. Snowflake integration with PowerBI
      vi. Time travel

Azure storage(Blob storage) -> Integration -> External Stage -> Copy Into -> snowflake warehouse -> client db -> BI -> report.

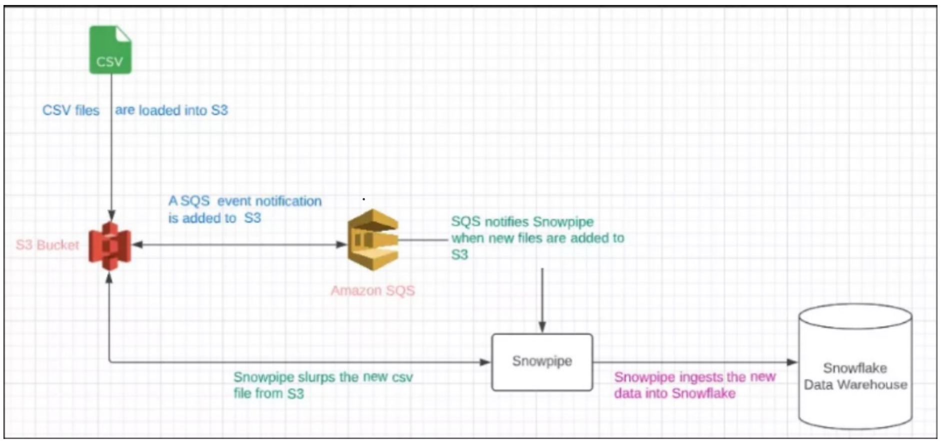
Azure blob storage and AWS S3 are not distributed file systems. They are object storages. This object storage is not suitable for analysis purpose. This object storage is suitable to only store the data. When you want to do analysis, bring the data into ADLS(Azure data lake storage) or AWS data lake.

Snowpipe:

For importing data.

Pdf: Snowpipe.pdf

Copy Into	Snowpipe
1. No automation.	1. Automation.
1. Need to run pipeline again if new data comes.	1. New data is brought automatically.


CSV
CSV files
are loaded into S3
A SQS event notification is added to S3
S3 Bucket
SQS notifies Snowpipe when new files are added to S3
Amazon SQS
Snowpipe
Snowflake Data Warehouse
Snowpipe slurps the new csv file from S3
Snowpipe ingests the new data into Snowflake

Snowpipe Diagram

Snowflake with PowerBI

step 1: open PowerBi

step2: select get

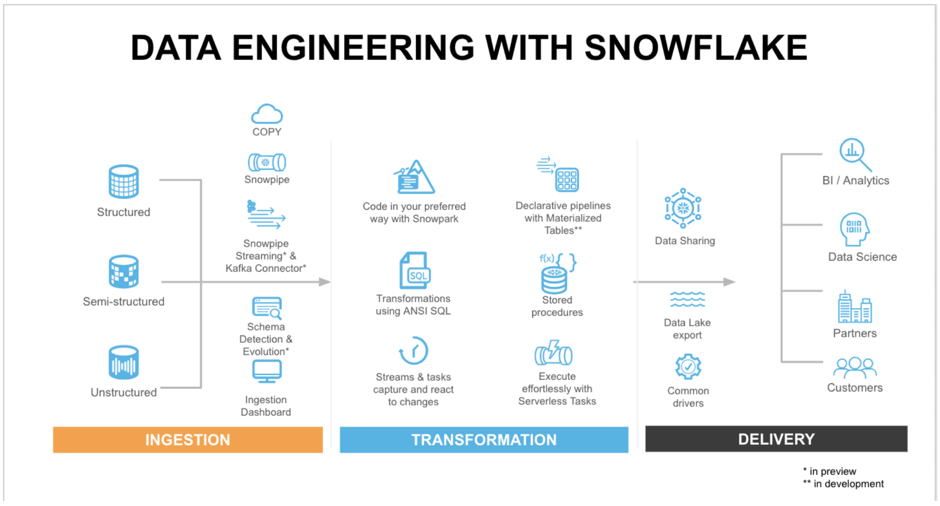
step 3: search for snowflake

step 4: enter credentials

step 5: select the table

Snowflake time travel:

1. it used to access the historical data at any point within a defined period. Time travel is used to analyze the data meaning that backing up data from keypoints in the past.
2. Using time travel, you can query the data in the past (updated data or deleted data). There are two functions used for the time travel data analysis. 1. at 2. before
3. Example Time travel query using to select the historical data from the tables as of the date and time represented by timestamp. Using time travel you can retrieve the historical data from the table as of 30 minutes ago
4. *select * from details at(offset=> -10*60);*
5. *select * from details at(timestamp => 'Fri, 15 Sep 2023 16:20:00 +0530'::timestamp_tz);*


DATA ENGINEERING WITH SNOWFLAKE
Structured
Semi-structured
Unstructured
COPY
Snowpipe
Snowpipe Streaming* & Kafka Connector*
Schema Detection & Evolution*
Ingestion Dashboard
Code in your preferred way with Snowpark
Declarative pipelines with Materialized Tables**
Transformations using ANSI SQL
Stored procedures
Streams & tasks capture and react to changes
Execute effortlessly with Serverless Tasks
Data Sharing
Data Lake export
Common drivers
BI / Analytics
Data Science
Partners
Customers
INGESTION
TRANSFORMATION
DELIVERY
* in preview
** in development

4 Tier Architecture Of Snowflake Data Engineering

Snowflake Commands

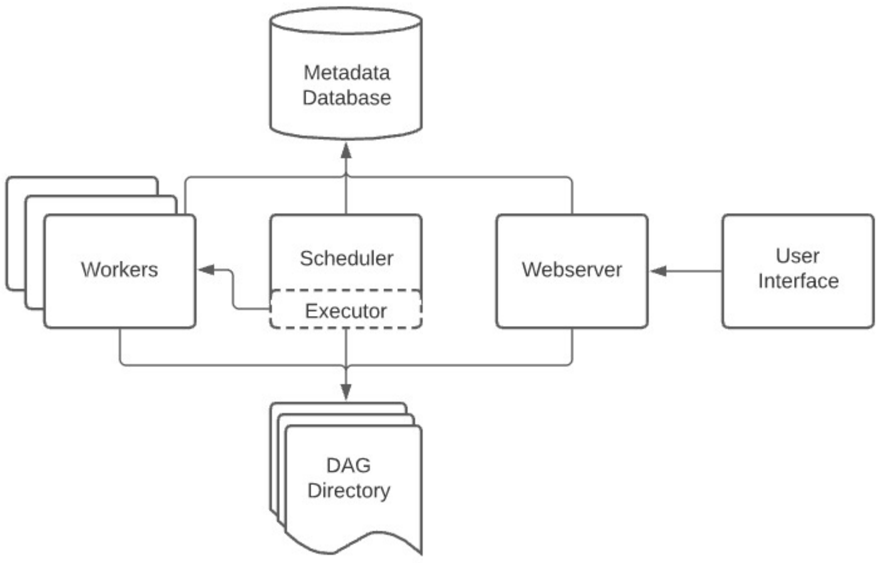
1. create warehouse identifier('"training"') comment = '' warehouse_size = 'x-small' auto_resume = true auto_suspend = 300 enable_query_acceleration = false warehouse_type = 'standard' min_cluster_count = 1 max_cluster_count = 1 scaling_policy = 'standard'

2. drop warehouse training;
3. create stage stage_name;
4. drop stage stage_name;
5. Removes the files inside the stage but not the stage: rm @stage_name;
6. Overwrite the already existing file in stage: copy into @op_stage/op1 from (select * from products where product_id = 1) overwrite = true;

Airflow

Oozie It is a part of Hadoop eco sysyem, shell scripting, scheduling, not suitable for multinode cluster.

Apache NIFi Scheduling with all components, graph based, not suitable for multinode cluster.



1. Airflow allows you to scheudle, monitor data pipelines and workflows (tasks or jobs) using web interface.
2. It is open souce
3. It can be used with any services like DBs,cloud etc
4. User friendly
5. It will not create ETL pipelines, it will just schedule the pipelines to get executed.
6. Everything is drag and drop in airflow, sometimes you write code in python.
7. Executor is JVM and executes the actual task, it is assigned some processor cores and RAM.
8. DAG directory stores all the details of airflow jobs. DAG means directed acyclic graphs, it shows where the job is started and where it is ended graphically. We can see the timings of different phases of execution of a job in DAG directory.
9. One Airflow job has one DAG. The job is known also known as one workflow.
10. We should create workflow using python scripts, we can write the script to execute the job immediately or schedule the job on basis of time or we can trigger the workflow based events.
11. We can write the script to retry the execution of the job incase of failures.

DAG

1. DAG (Directed acyclic graph): It is a collection of tasks and describes how to run a workflow written in Python.
2. Pipelines are designed as a DAG by dividing the pipeline into tasks that can be executed independently.
3. Then these tasks are combined logially as a graph.

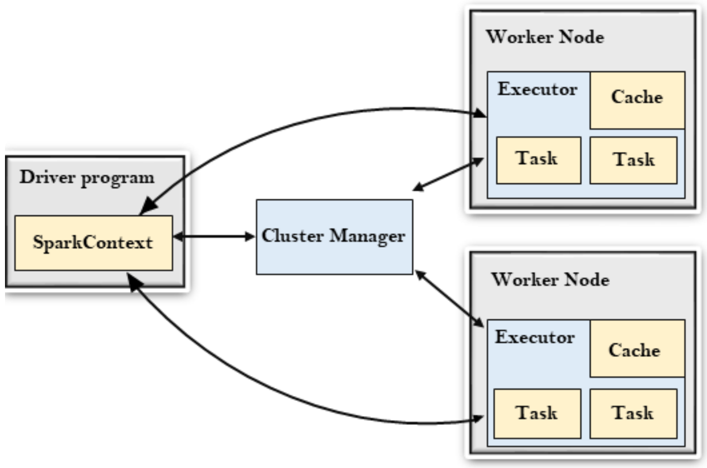
Task

1. A unit of work within a DAG

2. Hers tasks will be imlementing using operators.
3. Example of operators: Python operator, Bash operator (running bash commands), Postgre operators, S3 operator, sensor operators. These operators are using in a DAG to describe the task in pipeline.
4. A standard template of the DAG to create workflow or a job.
    a. import modules and packages
    b. define default arguments
    c. Initiate the DAG (Dag object)
    d. Define the tasks
    e. Define the dependency

Spark

Koushik Notes

1. Spark is an opensource cluster computing framework.
2. It is for processing large scale data (not for data storage)
3. Spark can read data from a cluster(Like HDFS)
4. It is built using scala (scala is built using Java)
5. It runs the processing in memory only. Not in disk.
6. Spark is 100 times faster in memory and 10 times faster in disk as compared to MapReduce.
7. Spark reads data from everywhere like HDFS, Mesos, Standalone cluster, data sources (AWS S3, azure blob)
8. Spark is a unified platform because it provides data frames, sql, complex programming, ML, streaming (for real time data processing).
9. Master slave architecture
10. Spark use scala, scala uses lazy evaluation. (not do the work until it is required).
11. 

    1. Architecture in terms of Execution process.
    2. In JVM there are Executors. Within that executors we have threads available.

RDD:

Read only.

RDD resilient distributed datasets. It is the main abstraction of spark.

Distributed collection of elements which are read only(immutable), partitioned (parallelism), runs in parallel, fault tolerant, lazy evaluation.

A "distributed collection of data elements" refers to a dataset that is spread across multiple machines or nodes in a distributed computing environment. In this context, "collection" simply means a group or set of data elements, such as numbers, text, records, or any other data type.

Characteristics:

1. Resilient (Fault tolerant)
2. Distributed
3. Immutable
4. Lazy Evaluation
5. Parallel Processing
6. In-Memory Storage

Transformations The functions applied are called transformations. The work is not done yet.

DAG Upon applying a transformation spark only creates a logical plan of execution (estimating computing required).

Action Upon calling an action, the actual execution will happen using the DAG reference.

Parallelization:

CPU

- Core
- Thread
- VCPU (Core in cloud)
- SLOT (Threads that are available to perform a task)

When you use hyper v you can use 2 threads in a single core.

With HyperV: 1 socket = 4 cores = 8 threads (8 VCPU in cloud)

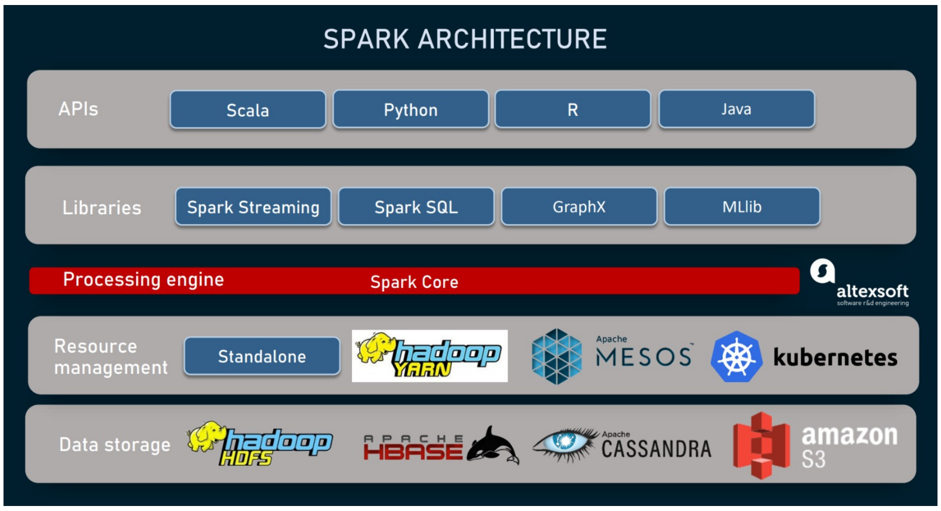
Without HyperV: 1 socket = 4 cores = 4 threads (4VCPU in cloud)

Vertical scaling: Increasing the resources in the computer.

Horizantal Scaling: Increasing the number of computers.

One Partition will have one task. By default the number of partitions of an RDD = number of cores.

Architecture:



The amount of data that fits into the ram it processes the data in memory, after that it will go to the other parts of the data. It is called spill.

Spark needs a resouce manager it could be YARN, Mesos or stand alone.

The core API of spark is RDD.

Spark Session:

You need to register in order to use.

There are other sessions like spark context in the earlier versions. They are: SQL context, Hive context, Streaming Context. But now there is no need to use all these contexts seperately. We can directly use the Spark Session to have all these contexts.

Transformations:

1. Narrow transformation (does on one data value)
   a. They are all put in one stage
   b. Eg: map, filter, mapPartitions
2. Wide transformation(does on multiple data values like group by)
   a. They involve in multiple stages
   b. Eg: reduceByKey, groupByKey, joins, repartitions

Joins

1. safe join
   a. sort merge, uses shuffling, use when both the files are large
   b. broadcast join: copy the smaller file in every node as read only and then join it with the original tables.

Shared variables

1. Accumulators
   a. Implements counter
2. Broadcast variables
   a. Implement read only data from the smaller file to the larger file

DataFrames

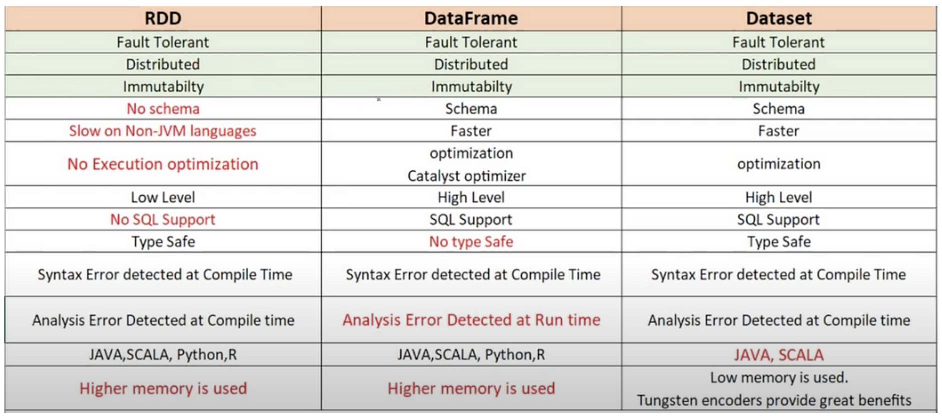
1. Internally dataframe is stored as row objects.

RDD vs Data Frame vs Data sets:

DataFrame: distributed collection of row objects

RDD vs DataFrame vs DataSet

RDD	**DataFrame**	**DataSets**
Distributed Collection	Distributed Collection of Row Objects.	
Immutable	Infer schema, it gives structure of data	
Fault Tolerance	Hive,SQL Compatibility	
Lazy evaluations	Lazy evaluation	Lazy evaluation
Unstructured Data Doesn’t infer schema, optimization of code is manual	optimization → Catalyst Optimiser (Tungsten)	
Scala, Java, R, Python	Available in Java, Scala, Python, R	Datasets are available only for Java and scala
It is compile safe. i.e., if error is present then it tells in the compile time only.	It throws error only when you run not when you compile.	
Advantages: OOPs programming, type safety	Relational format, optimization	Both the advantages of RDD and DF.
Suitable for smaller datasets as it stores data in onHeap.	Suitable for large dataset, it uses offHeap.	



RDD	DataFrame	Dataset
Fault Tolerant	Fault Tolerant	Fault Tolerant
Distributed	Distributed	Distributed
Immutabilty	Immutabilty	Immutabilty
No schema	Schema	Schema
Slow on Non-JVM languages	Faster	Faster
No Execution optimization	optimization Catalyst optimizer	optimization
Low Level	High Level	High Level
No SQL Support	SQL Support	SQL Support
Type Safe	No type Safe	Type Safe
Syntax Error detected at Compile Time	Syntax Error detected at Compile Time	Syntax Error detected at Compile Time
Analysis Error Detected at Compile time	Analysis Error Detected at Run time	Analysis Error Detected at Compile time
JAVA,SCALA, Python,R	JAVA,SCALA, Python,R	JAVA, SCALA
Higher memory is used	Higher memory is used	Low memory is used. Tungsten encoders provide great benefits

On heap memory:

1. Serialization and de serialization is needed.
2. It is present within the executors.
3. It is controlled by JVM
4. Garbage collection happens

Off heap memory:

1. Serde is not needed.
2. It is controlled by OS.
3. spark.memory.offHeap.size
4. No garbage collection

On Heap	Off heap
1. Better perfrormance, bcoz object allocation and deallocation is automatic	1. Slower than onHeap, but better than disc.
1. Manged and controlled by garbage collector.	1. Managed by OS no need of GC
1. Data stored in form of java bytes(serialized)	1. In form of array. No overhead of ser and de ser.
1. Suitable for smaller sets of data	1. for bigger data

Unified memory: It has executor memory and storage memory.

Data Validation modes in DataBricks DF:

There are 3 types of Data validation rules they are

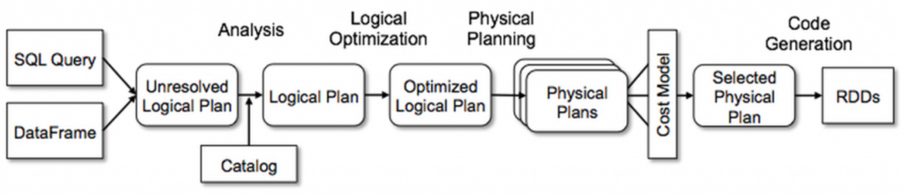
df=spark.read.format("csv").option('sep',",").option('header',False).option('mode','permissive').schema(df_schema).load("dbfs:/Fil

1. Permissive: Allow all the records in the new column.
2. DropMalforced: Drop the corrupted records.
3. FailFast: Fail the importing process if there are corrupt records.

Catalyst optmizer

Cost based optimizer: Based on the statistics cost is estimated for diffrenet approaches of the executions and the best is used.

Rule based optimizer: Set of predefined rules to design the execution plan.



Dynamic coalescing shuffle partitions.(Avoids the difficulty in selecting the correct number of partitions)

Dynamic switching of join strategies.

Dynamically optmizing the skew joins.

Adaptive Query Optmization

In Spark 3.0, Adaptive Query Execution (AQE) was introduced. One difference between AQE and Catalyst Optimizer is that AQE modifies the Physical Plan based on Runtime Statistics, so AQE can tune your queries further on the flight.

Spark Streaming

Spark has a library called Spark Streaming. It enables scalable, high-throughput, fault-tolerant stream processing of live streaming data.

Two categories of streaming: Basic and advanced.

Basic sources: File systems and socket connections

Advanced sources: Kafka, Flume, Twitter etc

There will be a thread for handling inputting of data, and then we should have another thread to process the data.

Dstreams: It reperesents a stream of data that comes continually.

KAFKA:

Functions:

1. Create context:
    a. sc.getOrCreate()
2. Create RDD:
    a. sc.parallelize([1,2,3,4])
3. Read text file as RDD:
    a. sc.textFile('file_path', num_of_partitions)
4. See the number of partitions made:
    a. file. getNumPartitions()
5. See the entire data:
    a. file .collect()
6. Count the number of lines:
    a. file.count()
7. Read the data partition wise:
    a. file.glom().collect()
8. Read specific number of lines:
    a. file.take(num_of_lines)
9. Apply a function to a RDD having single partition:
    a. file.map(functoin)
10. Apply a function on key value pair sequences(the function will be applied to values only)
    a. file.mapValues(function)
11. Apply a function to a RDD partition seperately:
    a. file.mapPartitions(function)
12. Filter the data in RDD:
    a. file. filter(function)
    b. Note that the function should return boolean
13. Get the top elements in an RDD:
    a. x_part.top(n)
14. Get the max and min elements

   a. x.max()
   b. x.min()
15. Find the stats like count, mean, std, max, min
   a. x.stats()
16. Get the data from an RDD as a dictionary
   a. rdd.collectAsMap()
17. Reduce the number of partitions (Not the actual rdd is partitioned but it returns the reduced one)
   a. coalsesce
      i. x_part.coalesce(2).glom().collect()
   b. repartition
      i. x_part.repartition(2).glom().collect()
18. Get the default parallelism set
   a. sc.defaultParallelism
19. Get the default min partitions alowed
   a. sc.defaultMinPartitions
20. To save the result of a transformation in user dfined storage level
   a. x_part.persist()
   b. The allowed memory locations are:
      i. MEMORY_ONLY
      ii. MEMORY_AND_DISK
      iii. MEMORY_ONLY_SER
      iv. MEMORY_AND_DISK_SER
      v. DISK_ONLY
   c. Once a result is stored in a location, it cannot be changed again. You can change the upcoming locations of the results but not the one which is already placed in a location.
21. to save the result in memory
   a. x.cache()
22. To remove the saved data from the memory
   a. x.unpersist()
   b. x.unpersist(blocking = True)
23. get all the persisted resulsts:
   a. spark.sparkContext._jsc.getPersistentRDDs().items()
24. To unpersist all the items:
   a. use a loop and then unpersist the items
   b. for (id,rdd) in spark.sparkContext._jsc.getPersistentRDDs().items():

> rdd.unpersist()

DataFrames

1. Set the user defined column data types in a dataframe:
   a. Two things invovled StructField and StructType
   b. StructField is to define the data type and name of the column, StructType is the list of StructFields
   c. StructField(col_name, data_type, True)

2. Change the column names of the dataframe
    a. df.withColumnRenamed(‘old_col_name’, ‘new_col_name’)
3. Create a dataframe from an RDD:
    a. rdd.createDataFrame(StructType(fields = [StructField(…),…]))
4. Infer Schema from the file automatically while importing from a csv
    a. spark.read.csv(‘file’, inferSchema = True)
5. Filter DataFrame records
    a. df.filter(df[‘age’] == 18).show()
6. dropna
    a. df.dropna(thresh = 2)
    b. using thresh we can mention the number of allowed null values in a row
7. modify, create, perform calculation on a column
    a. df.withColumn(‘new_col’, col)
    b. Here the col is an expression or function that defines the new column or updated column.
8. Create an SQL table
    a. df.createOrReplaceTempView (‘SQL_table_name’)
9. Query an SQL table from pyspark
    a. spark.sql(“select * from sql_table”)
    b. The above is a transformation, not an action.
10. Read a CSV file into pyspark
    a. spark.read.csv(‘file’, inferSchema= True, header = True)
11. Group by
    a. df.groupby(‘city’).sum(‘sales’).show()
12. Create a dataframe from an RDD
    a. spark.createDataFrame()
13. convert a string to datetime
    a. df.to_datetime()

Broadcasting

1. Broadcast a file:
    a. sc.broadcast(var)

DataBricks fs and dbutils

1. %fs ls
2. dbutils.fs.ls('/FileStore/tables')
3. dbutils.fs.head('/FileStore/tables/sales_info.csv')
4. db dbutils.fs.mkdirs("NewDir")utils.fs.help()
5. dbutils.fs.cp("old_path", "new_path")
6. dbutils.fs.rm("/FileStore/tables/mydata/b1.csv")
7. dbutils.notebook.help()
8. dbutils.notebook.run(notebook, timeout, args)
9. dbutils.notebook.exit()
10. dbutils.widgets.text("file_name", "", "")

11. dbutils.widgets.removeAll()
12. folder_path = dbutils.widgets.get("file_name")

DataBricks sql

1. %sql select * from sales_csv limit 5;

DataBricks DataFrame

1. df4.union(df4).show()
2. df4.unionAll(df4).show()
3. df4_union.drop_duplicates().show()
4. df=spark.read.format("csv").option('sep',",").option('header',False).option('mode','permissive').schema(df_schema).option(“columnNameOfCorruptRecord”, “corrupt_record”).load("dbfs:/FileStore/tables/mydata/currupt_data.csv")
5. df.rdd.getNumPartitions()
6.

Modules

1. from pyspark import SparkContext
2. from pyspark.sql import SparkSession
    a. SparkSession.builder.appname(‘name’).getOrCreate()
3. from pyspark.sql.types import StructType, StructField
4. import pyspark.sql.functions as f