* Hive

- It is an Open source data warehouse system built on top of hadoop for querying & analyzing large dataset stored in Hadoop files.
- Processes structured & semi-structured data in hadoop
- Uses Hive QL language similar to SQL.
- Runs on our workstation & converts SQL query into series of jobs for execution on a hadoop cluster.
- Organizes data into tables.

Why?

- Traditional DBs were not able to handle large datasets.
- Used Map Reduce but it was difficult to program & needed SQL knowledge.
- Hive overcame the challenges they were facing

With hive, they performed

- Schema flexibility & evolution.
- Tables can be partitioned & bucketed.
- Tables of hive are directly defined in HDFS
- JDBC/ODBC drivers are available.

- Fast & Scalable, provides summarization, analysis & query of data.

- Hive shell (command line interface for hive)
- There are two modes that we can run the hive shell.

a. Non-Interactive mode - can be run using -f option we can specify location of file which contains HQL queries
eg. hive -f my-script.q

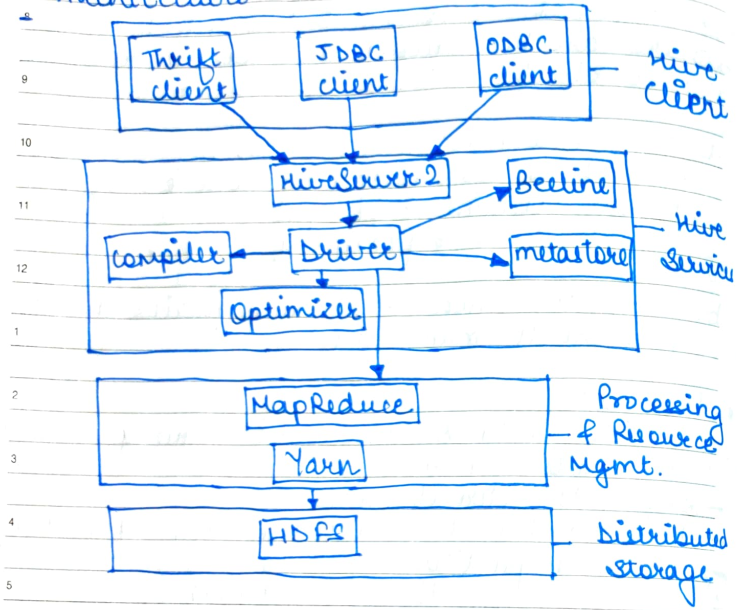
b. Interactive mode - We run queries on the hive shell manually.

- Features
- Provides data summarization, query, & analysis in easier manner.
- Supports external tables.
- fits the low-level interface req. of hadoop.
- supports partitioning & bucketing.
- Scalable, familiar, & extensible.

- Limitations
- Does not offer real time queries & row level updates.
- ~~Provides~~ acse Latency for hive queries is generally high.
- Not good for online transaction processing.

# Data warehouse here refers to inspecting, cleaning, transforming & modeling data with goal of discovering useful information.

- Architecture



1. Hive Client

→ Hive provides diff. for communication with a diff. type of applications.

→ These drivers & clients in turn again communicate with Hive server in hive services.

2. Hive Services

→ Client interaction with hive can be performed through Hive Services.

a. Beeline

→ It is a JDBC client providing command shell where user can submit its queries.

b. Hive Server 2.

→ It enables clients to execute queries in Hive

→ Allows multiple clients to submit requests to hive & retrieve the final results.

c. Driver

→ receives HQL ~~qeer~~ statement from user through command shell.

→ Creates session handles for query & sends query to compiler

d. Compiler

→ It parses the query. i.e ~~Is performs~~ (generating a syntactic structure of query)

→ It performs semantic analysis (understanding of query) & type checking on diff. query blocks & query expressions by using the metadata stored in metastore & generates an execution plan (i.e DAG (Directed Acyclic Graph), where each stage is map/reduce job.). (creating a structure of query)

e. Optimizer

→ It performs operations on execution plan & splits task to improve efficiency & scalability.

f. Execution Engine

→ It executes the execution plan created by the compiler in order of their dependencies using hadoop (YARN).

g. Metastore

→ It is a central repository that stores the metadata information about the structure of tables & partitions, including column & column type information.

→ Also stores information of serializer & deserializer, required for read/write operation & info. of HDFS files where data is stored.

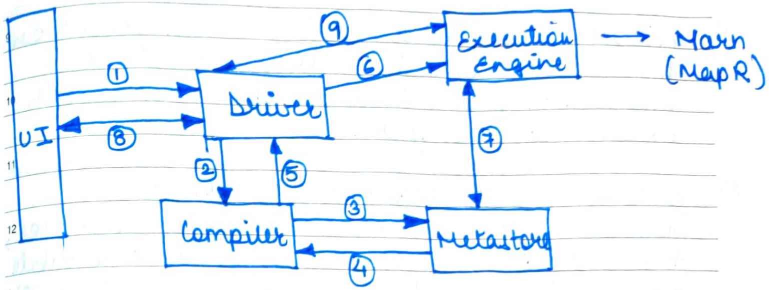
→ It is generally a relational database.

→ We can configure metastore in two modes.

1. Remote : Metastore runs in ~~own~~ seperate JVMs, not in Hive service
   - If someone wants to connect they can connect through Thrift Network APIs.

2. Embedded : Metastore runs on same JVM as the Hive service.
   - It uses derby database stored on local file system.
   - But only one Hive session could be open at a time.

• Job Execution Flow



① Execute Query : Query from user Interface is sent to Driver. ~~[illegible]~~

② getPlan : The driver accepts the query, creates a session handle for the query, & passes the query to the compiler for generating execution plan.

③ getMetaData : The compiler sends the metadata request to the metastore.

④ sendMetaData : The metastore sends the metadata to the compiler.

⑤ - Compiler uses this metadata for performing type-checking & semantic analysis (understanding the query) on the expressions.
- Then it generates the execution plan (DAG)
- For M/R jobs, the plan contains map operator trees (operator trees which are executed on mapper) & reduce operator tree (operator trees which are executed on reducer).

(5) sendPlan : The compiler then sends the generated execution plan to driver.

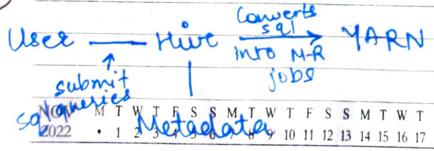
(6) executePlan : After receiving the execution plan from compiler, driver sends the execution plan to execution engine for executing the plan.

(7) Submit job to MapReduce | Yarn :

- For each task, either mapper or reducer, the deserializer associated with the table are used to read the rows from HDFS files, and passed to associated tree.
- Once o/p is generated, it is stored temporarily in HDFS thr. serialization.
- These serialized temporary files are then used to provide data to map | reduce stages of plan.
- FOR DML operation, the final temporary file is then moved to table's location.

(8) & (9) : Now the execution engine reads the contents of temporary files directly from HDFS as part of fetch call from Driver.

- The driver sends the results to Hive Interface



- DDL Commands

Cmds	Use with
- CREATE	Database, Table
- SHOW	DB, Table, ~~Table~~ Partitions, Table prop.
- DESCRIBE	Database, Table, view function Index
- USE	Database
- DROP	Database, Table
- ALTER	Database, Table
- TRUNCATE	Table.

- DML Commands

LOAD - Move data files into locations corresponding to hive tables.

SELECT

INSERT

DELETE

UPDATE

EXPORT - Exports table or partition data along with metadata to specified o/p location in HDFS.

IMPORT - Imports data from specified loc. to a new table or existing table.

- Views

→ Saving any result set data as views.

→ We can use all DML operations

Index

→ It is a pointer on a particular column of a table.

→ Creating Index means creating pointer

- Types of Tables

a. Managed Table

→ When we load data into a managed table, hive moves data into hive warehouse directory.

→ If we drop the table, this would delete the table structure as well as data.

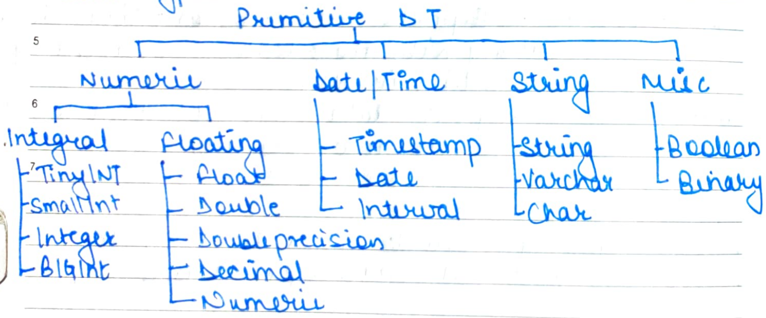
b. External Table

→ At the time of creation, the location of data is specified.

→ It does not even check whether loc. exists as the time it is defined

→ If we drop the table, only the structure is deleted, not the data.

- Data Types.



Complex DT

Array Map Struct Union

Partitions

→ Partitioning is a way of dividing a Table into related parts based on the values of particular columns, like date, city, dept etc.

→ Each table can have one or more partition key to identify a particular partition.

→ Partitions are kept as a sub-record inside the table's record present in HDFS.

Types

1. Static Partioning

→ In this, it is required to pass the values of partitioned columns manually while loading the data into table

eg. when loading data

load data local inpath ' ' into table < > partition (col = 'value').

2. Dynamic Partitioning (takes more time).

→ The values of partitioned columns exist within the table, so, not req. to pass values manually

→ Cannot perform alter on dynamic partition.

For static

set.hive. mapred.mode = strict is set.

For dynamic

set. ~~hive~~exec. dynamic. partition. mode = Non strict

exe true;

- Buckets

→ It is an organizing technique.

→ It divides large datasets into more manageable parts known as buckets.

→ Segregates hive tables data into multiple files or directories.

→ Used for efficient querying.

→ Divisions is performed based on hashing algorithm.

Properties

set. hive. enforce. bucketing = true:

Calculation

column value % n_buckets = hash value

→ With help of [clustered by clause] & optional [sorted By] clause in [create table] statement we can create bucketed tables.

→ Partitioned data can also be bucketed.

- Columnar Data Storage

→ If your data access patterns mostly involves selecting few columns to perform aggregation then use columnar storage.

→ Saves disk space.

→ reduce I/O when fetching data.

→ Improves query execution time.

→ Efficient for accessing, retrieving

1. ORC (Optimized row columnar)

→ Highly optimized for reading, writing, and processing.

→ Has best compression rate because of strips

Advantages

→ Single file as O/P of each task, reduces NN's load,

→ Ability to split files without scanning.

→ ORC stores collection of rows in one file & within the collection, the row data is stored in columnar format.

2. Parquet

→ stores data in nested form.