



Apache Hive

# HIVE

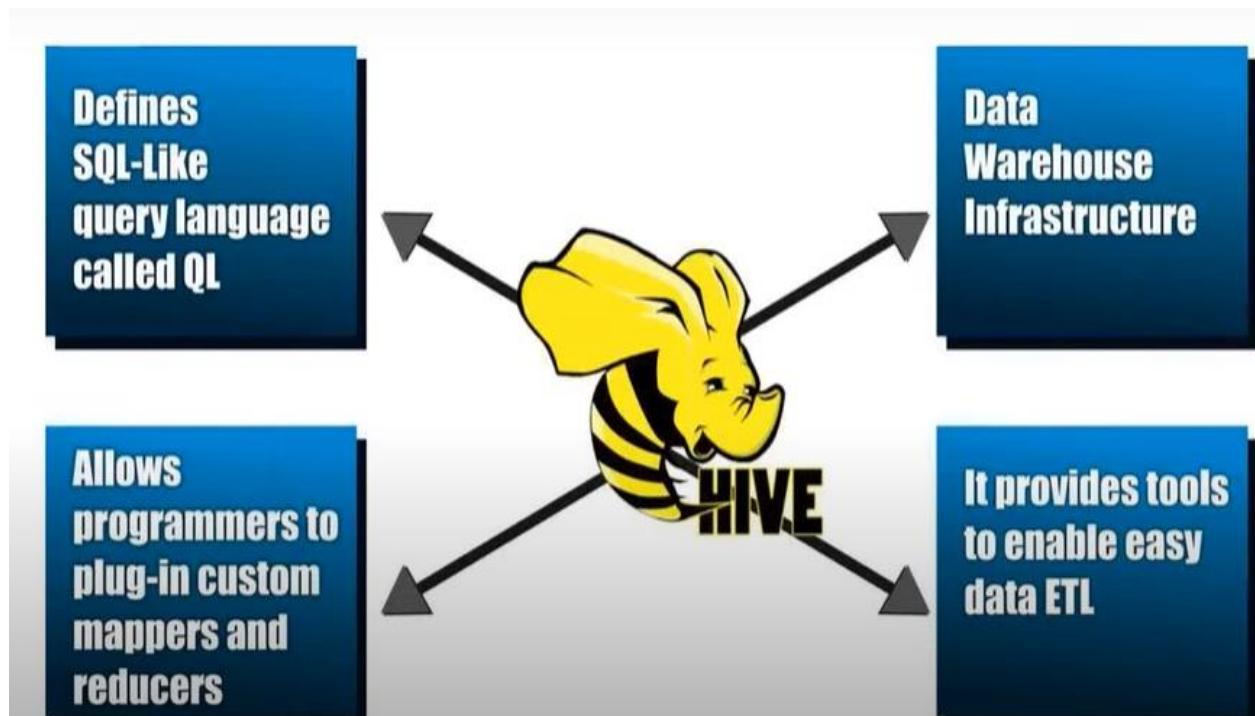
- Apache Hive is an open-source software utility **built on top of Hadoop** to provide easy **data storage and analysis in style of SQL**.
- Hive - Basically, a tool which we call a **data warehousing tool** is Hive. However, Hive gives **SQL queries** to perform an analysis and also an abstraction. Although, Hive it is not a database it gives you **logical abstraction over the databases and the tables**
- **It is a distributed, fault-tolerant data warehouse system** that enables analytics at a massive scale and facilitates **reading, writing, and managing** petabytes of data residing in distributed storage using SQL.
- Hive is built on top of Apache Hadoop and supports storage on S3, adls, gs etc though HDFS.
- It is not built for Online Transactional Processing (OLTP) workloads. **Limited concurrency**
- It is designed to **enhance scalability, extensibility, performance, fault-tolerance** and loose-coupling with its input formats.
- Hive uses a language called **HiveQL**, which is similar to SQL, to allow users to express data queries, transformations, and analyses in a familiar syntax.
- HiveQL statements are compiled into MapReduce jobs, which are then executed on the Hadoop cluster to process the data.



# What is HIVE?

- Data Warehousing package built on top of Hadoop
- Used for data analysis
- Targeted towards users comfortable with SQL
- It is similar to SQL and called HiveQL
- For managing and querying structured data
- Abstracts complexity of Hadoop
- No need learn java and Hadoop APIs
- Developed by Facebook and contributed to community
- Facebook analyzed several Terabytes of data everyday using Hive

# Features of HIVE

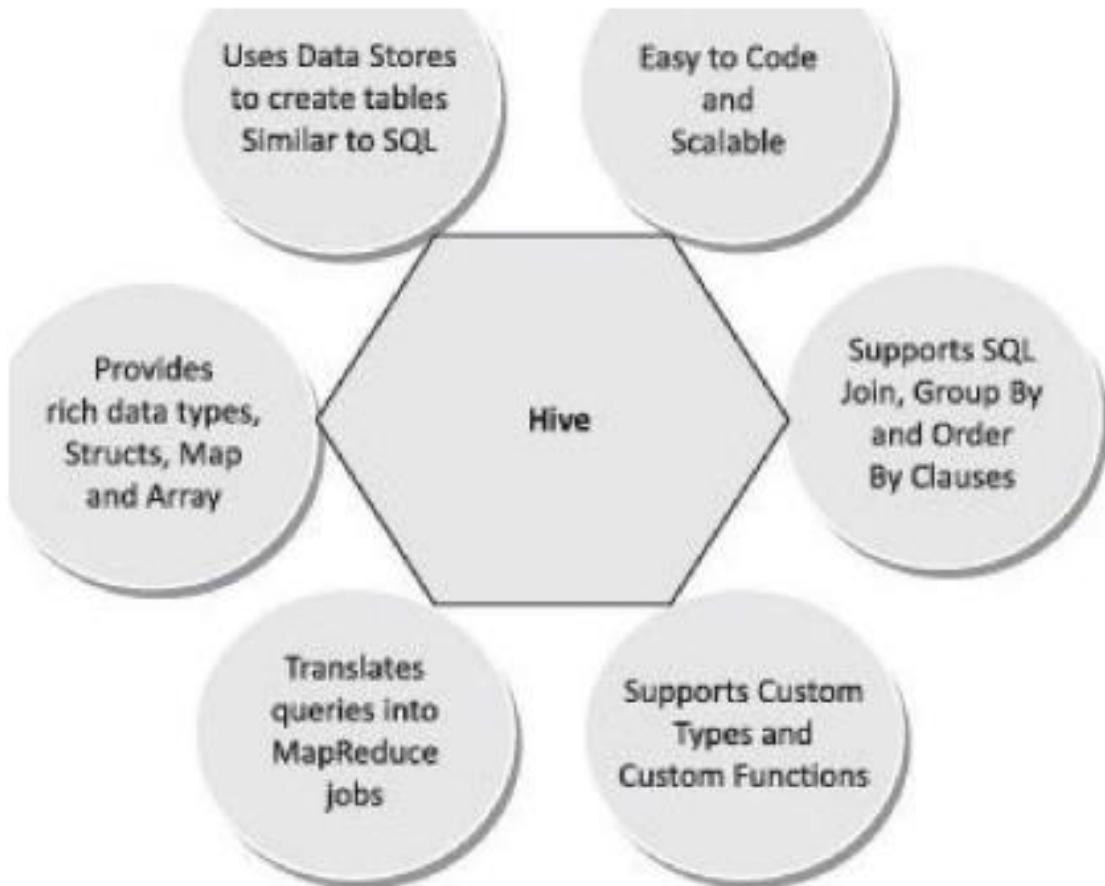


- Hive supports any client application written in Java, PHP, Python, C++ or Ruby by exposing its **Thrift server**. (You can use these client - side languages embedded with SQL for accessing a database such as DB2, etc.).
- As the **metadata information of Hive is stored in an RDBMS**, it significantly **reduces the time** to perform **semantic checks** during query execution.
- Useful for people who aren't from a programming background as it **eliminates the need to write complex MapReduce program**.
- **Extensible** and **scalable** to cope up with the growing volume and variety of data, without affecting performance of the system.
- It is as an **efficient ETL** (Extract, Transform, Load) tool.

## Features of HIVE

- It provides indexes, including bitmap indexes to accelerate the queries. Index type containing compaction and bitmap index as of 0.10.
- **Metadata storage in a RDBMS, reduces the time to function semantic checks** during query execution.
- **Built in user-defined functions (UDFs) to manipulation of strings, dates, and other data-mining tools.** Hive is reinforced to extend the UDF set to deal with the use-cases not reinforced by predefined functions.
- **DEFLATE, BWT, snappy**, etc are the **algorithms to operation on compressed data** which is stored in Hadoop Ecosystem.
- It is built for Online Analytical Processing (OLAP).
- It delivers various types of querying language which are frequently known as Hive Query Language (HQL or HiveQL).

# Hive Features...



- **Traditional relational databases** are designed for interactive queries on small to medium datasets and **do not process huge datasets** well.
- Hive instead **uses batch processing** so that it works quickly across a very large distributed database.
- It **queries data stored** in a distributed storage solution, like the **Hadoop Distributed File System (HDFS) or Amazon S3**
- Hive stores its database and table metadata in a **metastore**
- Hive includes **HCatalog**, which is a **table and storage management layer** that reads data from the Hive metastore to facilitate seamless integration between Hive, Apache Pig, and MapReduce.
- **By using the metastore, HCatalog allows Pig and MapReduce to use the same data structures as Hive**, so that the metadata doesn't have to be redefined for each engine.

# Hive Characteristics:

- ❖ Capability to “Translate Queries in to MapReduce Jobs”
- ❖ This makes Hive “scalable”, able to handle Datawarehouse applications and therefore suitable for the analysis of static data of an extremely large size, where the “fast – response time” is not a criterion.
- ❖ Supports web interfaces , Application API as well as web-browser clients, can access the Hive DB Server.
- ❖ Provides an SQL “Dialect” (Hive Query Language, abbreviated HIVEQL or HQL).
- ❖ Results of HiveQL query and the data load in the tables which store at Hadoop cluster at HDFS.
- ❖ Hive as data warehouse is built to manage and query only structured data which is residing under tables.
- ❖ **Tables and databases get** created first; then data gets loaded into the proper tables
- ❖ Hive supports four file formats: **ORC, SEQUENCEFILE, RCFILE** (Record Columnar File), and **TEXTFILE**
- ❖ difference between the Hive Query Language (HQL) and SQL is that **Hive executes queries on Hadoop's infrastructure instead of on a traditional database.**
- ❖ Hive supports Data Definition Language (DDL), Data Manipulation Language (DML), and User Defined Functions (UDF).

## Limitations of Hive

- Hive supports Online Analytical Processing (OLAP), but not Online Transaction Processing (OLTP).
- It doesn't support subqueries.
- It has a high latency.
- Hive tables don't support delete or update operations.
- Hive is not capable of handling real-time data.

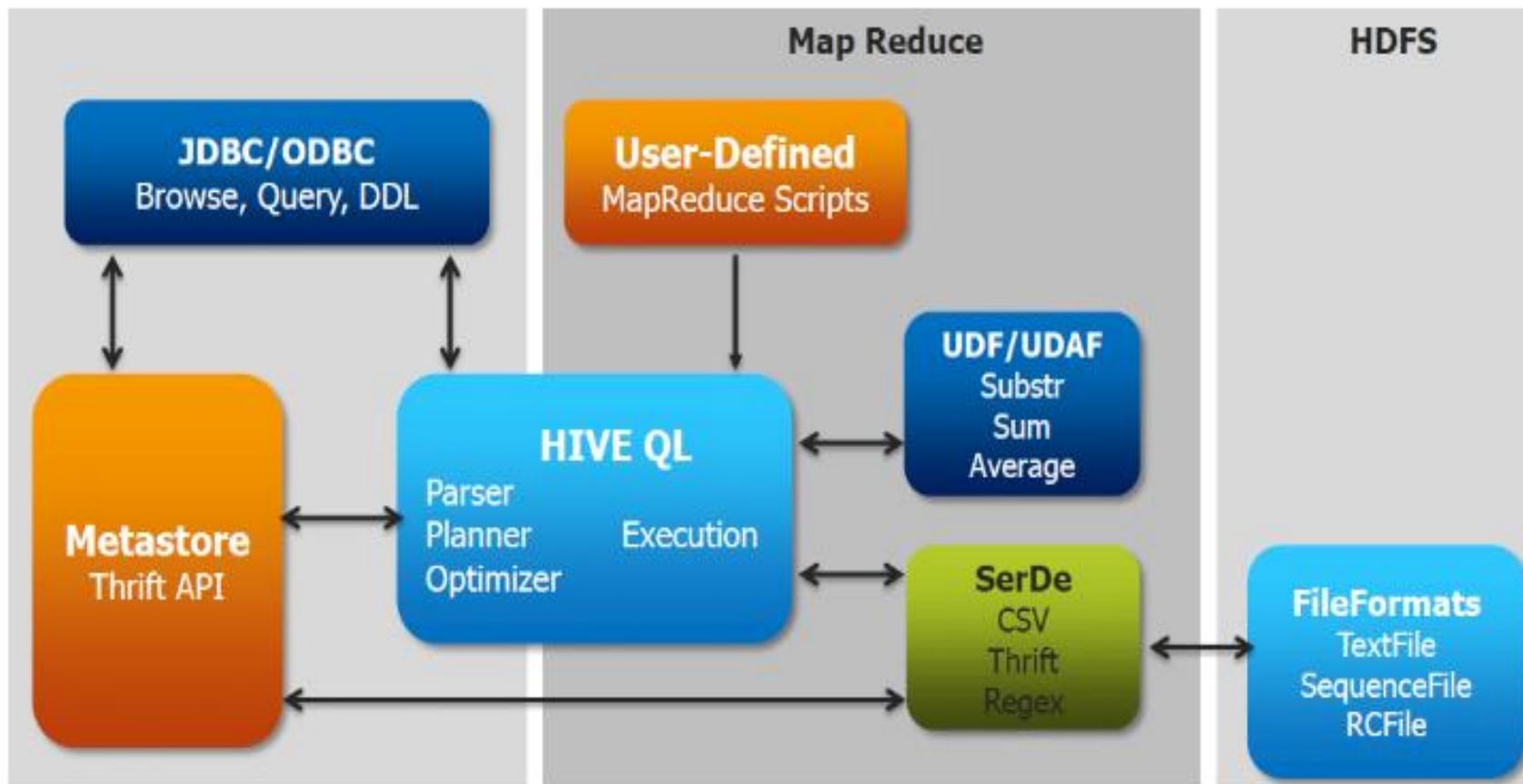
## Differences between Hive and Pig

| Hive                                     | Pig                                      |
|--|--|
| Hive is commonly used by Data Analysts.  | Pig is commonly used by programmers.     |
| It follows SQL-like queries.             | It follows the data-flow language.       |
| It can handle structured data.           | It can handle semi-structured data.      |
| It works on server-side of HDFS cluster. | It works on client-side of HDFS cluster. |
| Hive is slower than Pig.                 | Pig is comparatively faster than Hive.   |

## Hive Vs. SQL

| Feature            | Apache Hive                            | SQL                                   |
|--------------------|--|---------------------------------------|
| Purpose            | Batch and Interactive Query Processing | Relational Database Management System |
| Data Analysis      | Complex Data Processing                | Detailed Data Querying                |
| Architecture       | Data Warehousing Project               | RDBMS-Based Programming Language      |
| Data Types         | 9 Types Supported                      | 5 Types Supported                     |
| Multitable Inserts | Supported                              | Not Supported                         |
| MapReduce          | Supports MapReduce                     | No MapReduce Concept                  |
| OLTP               | Not Supported                          | Supports OLTP                         |
| Schema Support     | Supported                              | Used for Data Storage                 |
| Views              | Read-Only Format                       | Updateable Views                      |
| Data Size          | Can handle petabytes of data           | Can only handle terabytes of data     |

# Apache Hive Components:



Hive consists of the following major components:

**Metastore** – To store the metadata.

**JDBC/ODBC** – Query Compiler and Execution Engine to convert SQL queries to a sequence of MapReduce.

**SerDe and ObjectInspectors** – For data formats and types.

**UDF/UDAF** – For User Defined Functions.

**Clients** – Similar to MySQL command line and a web UI.

## Major Components

| Unit Name                | Operation  |
|--------------------------|--|
| User Interface           | Hive is a data warehouse infrastructure software that can create interaction between user and HDFS. The user interfaces that Hive supports are Hive Web UI, Hive command line, and Hive HD Insight (In Windows server).                                  |
| Meta Store               | Hive chooses respective database servers to store the schema or Metadata of tables, databases, columns in a table, their data types, and HDFS mapping.   |
| HiveQL<br>Process Engine | HiveQL is similar to SQL for querying on schema info on the Metastore. It is one of the replacements of traditional approach for MapReduce program. Instead of writing MapReduce program in Java, we can write a query for MapReduce job and process it. |
| Execution Engine         | The conjunction part of HiveQL process Engine and MapReduce is Hive Execution Engine. Execution engine processes the query and generates results as same as MapReduce results. It uses the flavor of MapReduce.  |
| HDFS or HBASE            | Hadoop distributed file system or HBASE are the data storage techniques to store data into file system.  |

# Hive Architecture:

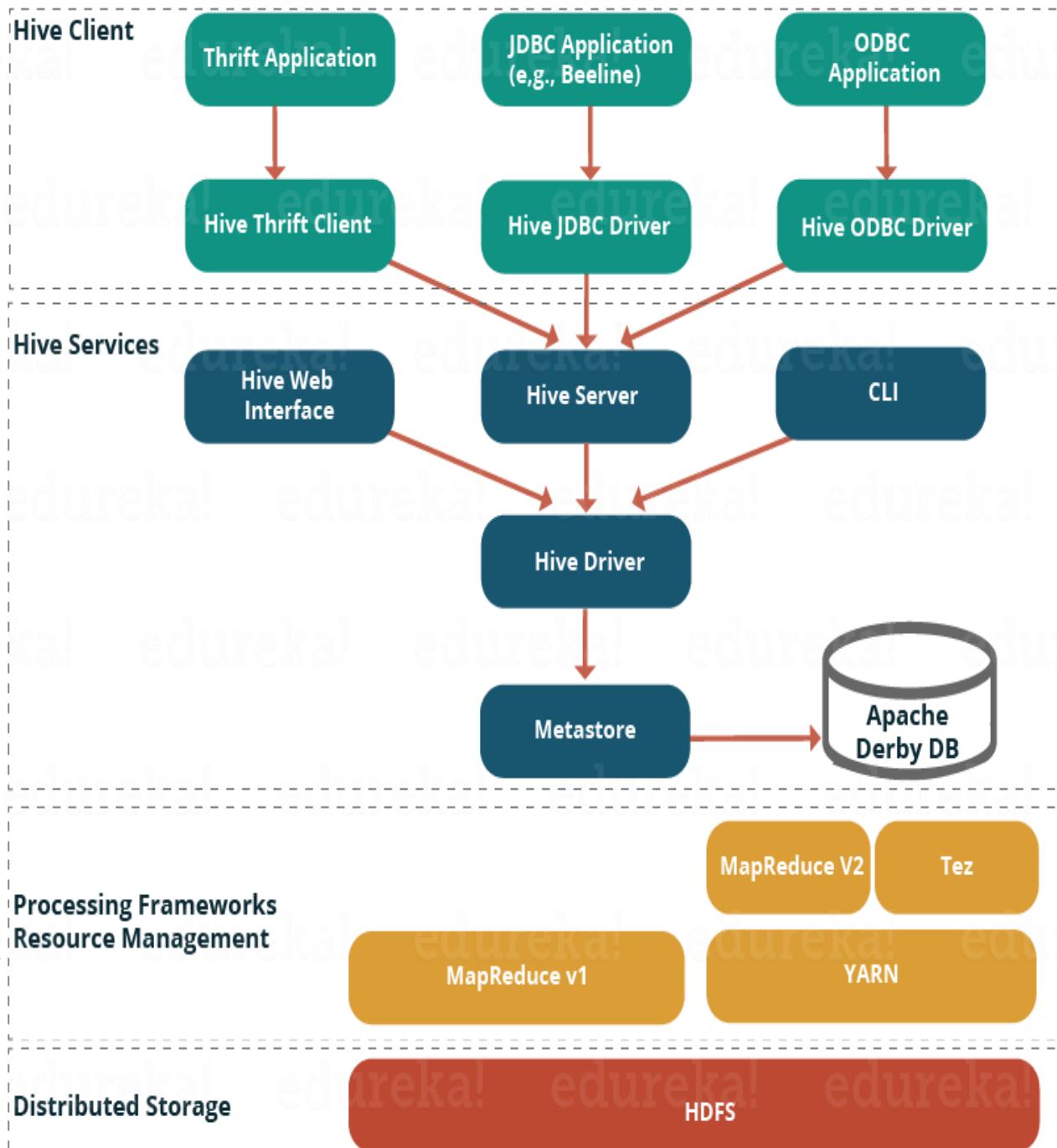
Hive Architecture can be categorized into the following components:

- **Hive Clients:** Hive supports application written in many languages like Java, C++, Python etc. using JDBC, Thrift and ODBC drivers. Hence one can always write hive client application written in a language of their choice.

- **Hive Services:** Apache Hive provides various services like CLI, Web Interface etc. to perform queries.

- **Processing framework and Resource Management:** Internally, Hive uses Hadoop MapReduce framework as execution engine to execute the queries.

- **Distributed Storage:** As Hive is installed on top of Hadoop, it uses the underlying HDFS for the distributed storage.



## Hive Client

- **Thrift Server** - It is a cross-language service provider platform that serves the request from all those programming languages that supports Thrift.
- **JDBC Driver** - It is used to establish a connection between hive and Java applications. The JDBC Driver is present in the class org.apache.hadoop.hive.jdbc.HiveDriver.
- **ODBC Driver** - It allows the applications that support the ODBC protocol to connect to Hive.

## Hive Services

- **Hive CLI** - The Hive CLI (Command Line Interface) is a shell where we can execute Hive queries and commands.
- **Hive Web User Interface** - The Hive Web UI is just an alternative of Hive CLI. It provides a web-based GUI for executing Hive queries and commands.
- **Hive MetaStore** - It is a central repository that stores all the structure information of various tables and partitions in the warehouse. It also includes metadata of column and its type information, the serializers and deserializers which is used to read and write data and the corresponding HDFS files where the data is stored.
- **Hive Server** - It is referred to as Apache Thrift Server. It **accepts the request from different clients** and provides it to **Hive Driver**.
- **Hive Driver** - It receives queries from different sources like **web UI, CLI, Thrift, and JDBC/ODBC driver**. It transfers the queries to the **compiler**.

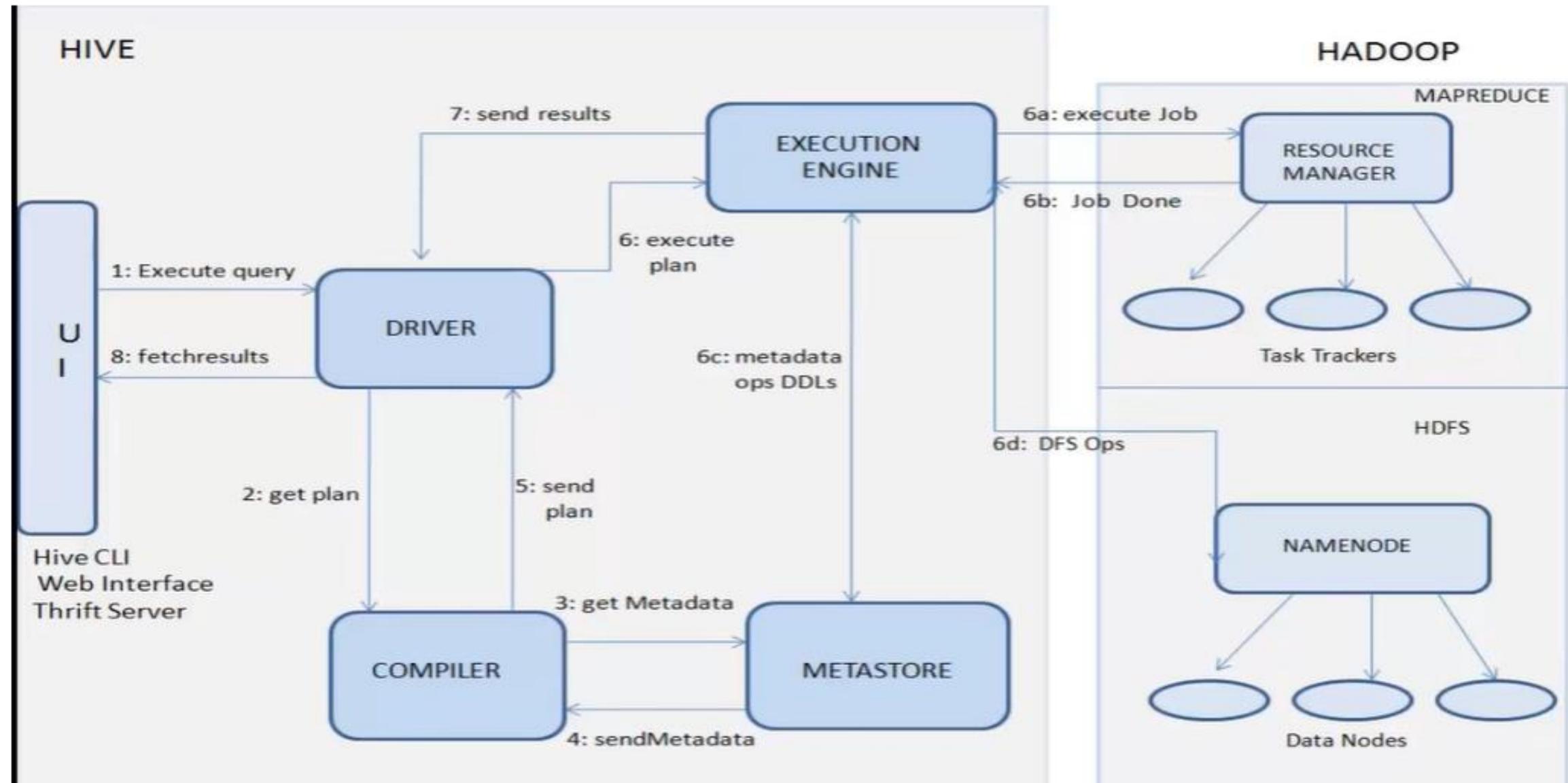
## Hive Services

- **Hive Compiler** - The purpose of the compiler is **to parse the query and perform semantic analysis** on the different query blocks and expressions. It **converts HiveQL statements into MapReduce jobs**.
- **Hive Execution Engine** - Optimizer generates the **logical plan in the form of DAG** of map-reduce tasks and HDFS tasks. In the end, the execution engine executes the incoming tasks in the order of their dependencies.

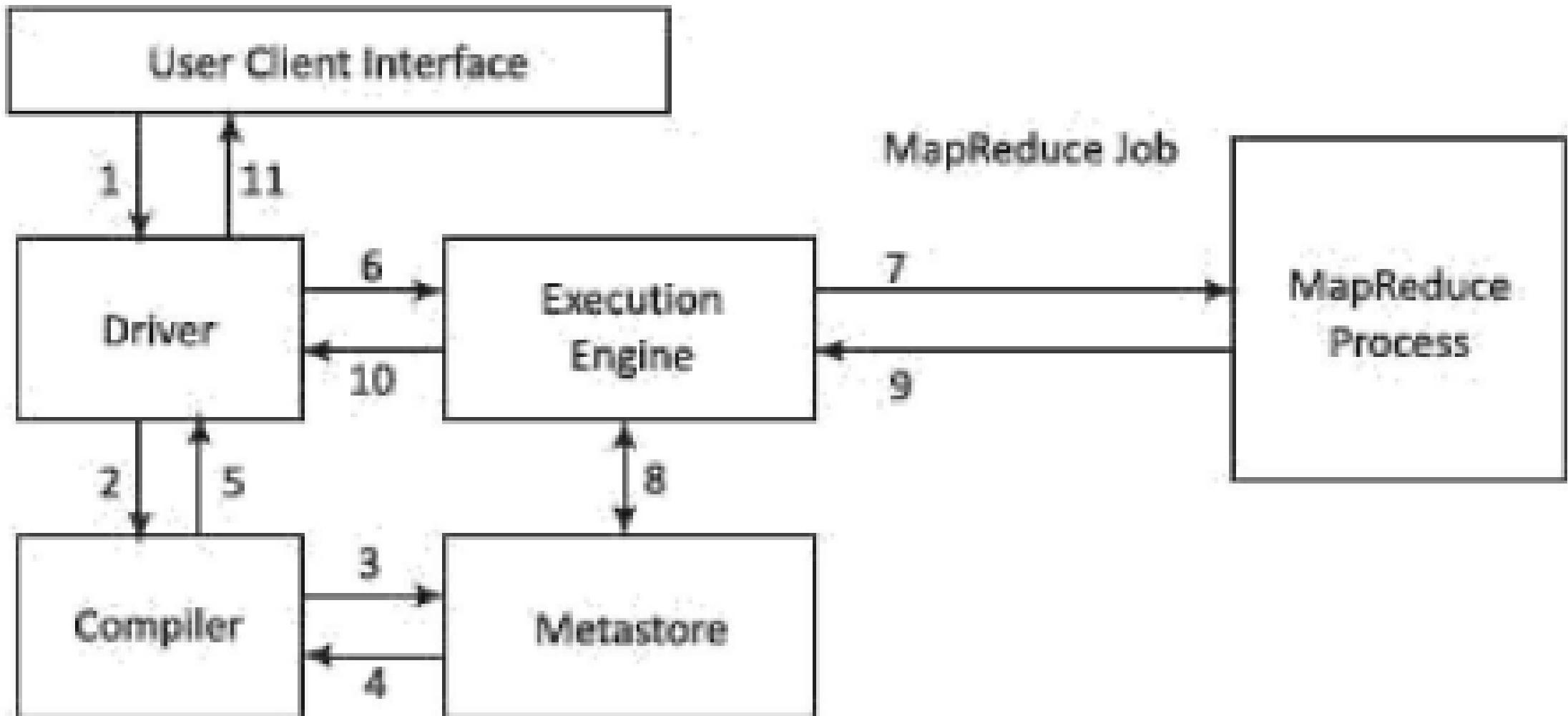
### Hive makes use of the following:

- HDFS for storage
- MapReduce for execution
- Stores metadata in RDBMS

# Program Execution



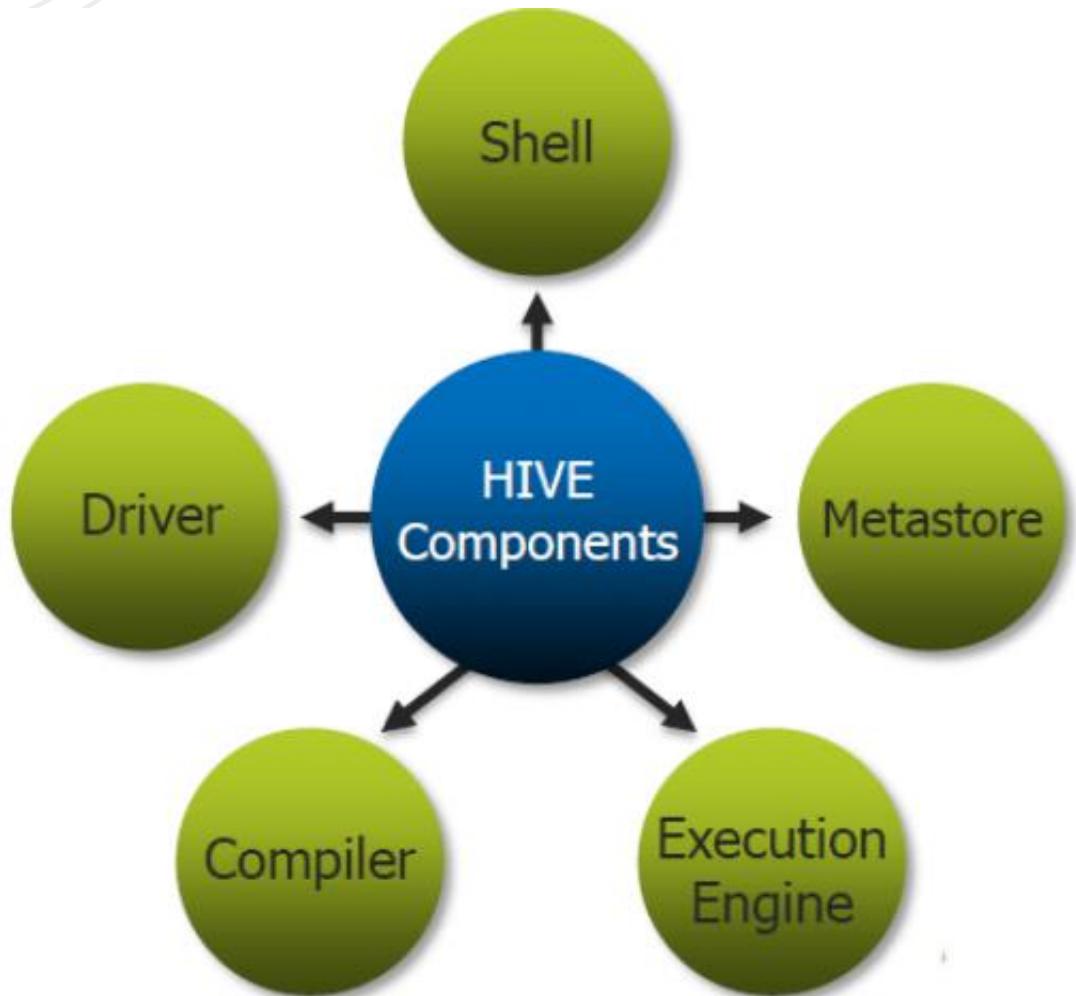
# Dataflow Sequence & Workflow Steps:



| No. | OPERATION   |
|-----|---|
| 1   | <b>Execute Query:</b> Hive interface (CLI or Web Interface) sends a query to Database Driver to execute the query.  |
| 2   | <b>Get Plan:</b> Driver sends the query to query compiler that parses the query to check the syntax and query plan or the requirement of the query.   |
| 3   | <b>Get Metadata:</b> Compiler sends metadata request to Metastore (of any database, such as MySQL).   |
| 4   | <b>Send Metadata:</b> Metastore sends metadata as a response to compiler.   |
| 5   | <b>Send Plan:</b> Compiler checks the requirement and resends the plan to driver. The parsing and compiling of the query is complete at this place.   |
| 6   | <b>Execute Plan:</b> Driver sends the execute plan to execution engine.   |
| 7   | <b>Execute Job:</b> Internally, the process of execution job is a MapReduce job. The execution engine sends the job to JobTracker, which is in Name node and it assigns this job to TaskTracker, which is in Data node. Then, the query executes the job. |
| 8   | <b>Metadata Operations:</b> Meanwhile the execution engine can execute the metadata operations with Metastore.  |
| 9   | <b>Fetch Result:</b> Execution engine receives the results from Data nodes.   |
| 10  | <b>Send Results:</b> Execution engine sends the result to Driver.   |
| 11  | <b>Send Results:</b> Driver sends the results to Hive Interfaces.   |

# Dataflow Sequence & Workflow Steps:

# Hive Components:



## Shell:-

Provides User interface provide an interface between user and hive. It enables user to submit queries and other operations to the system.

## Metastore –

All the structured data or information of the different tables and partition in the warehouse containing attributes and attributes level information are stored in the metastore.

## Execution Engine –

Execution of the execution plan made by the compiler is performed in the execution engine.

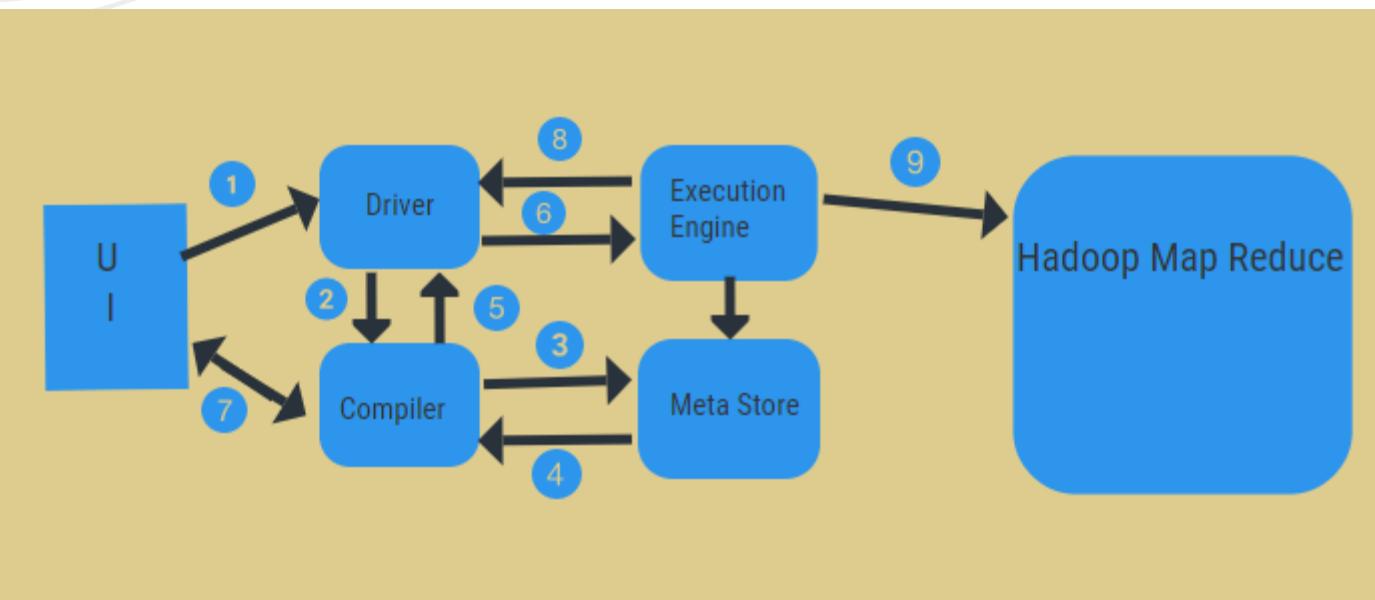
## Compiler –

Queries are parses, semantic analysis on the different query blocks and query expression is done by the compiler.

## Drivers:-

Queries of the user after the interface are received by the driver within the Hive. Concept of session handles is implemented by driver

# Hive Components:



## The steps include:

- 1.execute the Query from UI
- 2.get a plan from the driver tasks DAG stages
- 3.get metadata request from the meta store
- 4.send metadata from the compiler
- 5.sending the plan back to the driver
- 6.Execute plan in the execution engine
- 7.fetching results for the appropriate user query
- 8.sending results bi-directionally
- 9.execution engine processing in HDFS with the map-reduce and fetch results from the data nodes created by the job tracker. it acts as a connector between Hive and Hadoop.

# Hive Metastore:

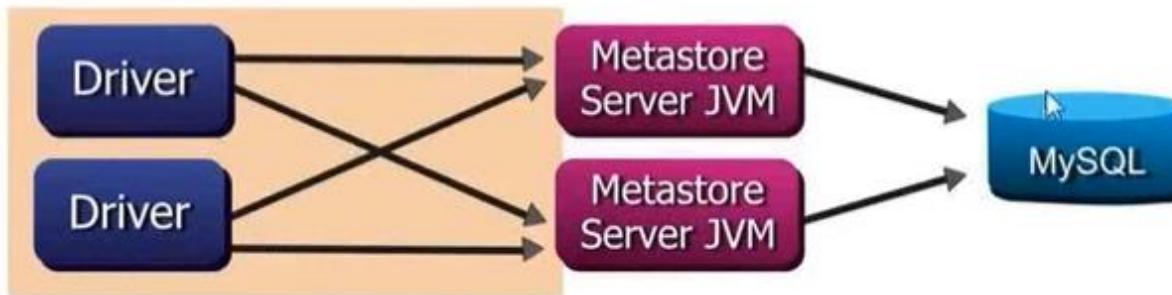
**Embedded Metastore**



**Local Metastore**



**Remote Metastore**



## Embedded Metastore:

It offers an embedded Derby database instance backed by the local disk for the Metastore, by **default**. It is what we call embedded Metastore configuration.

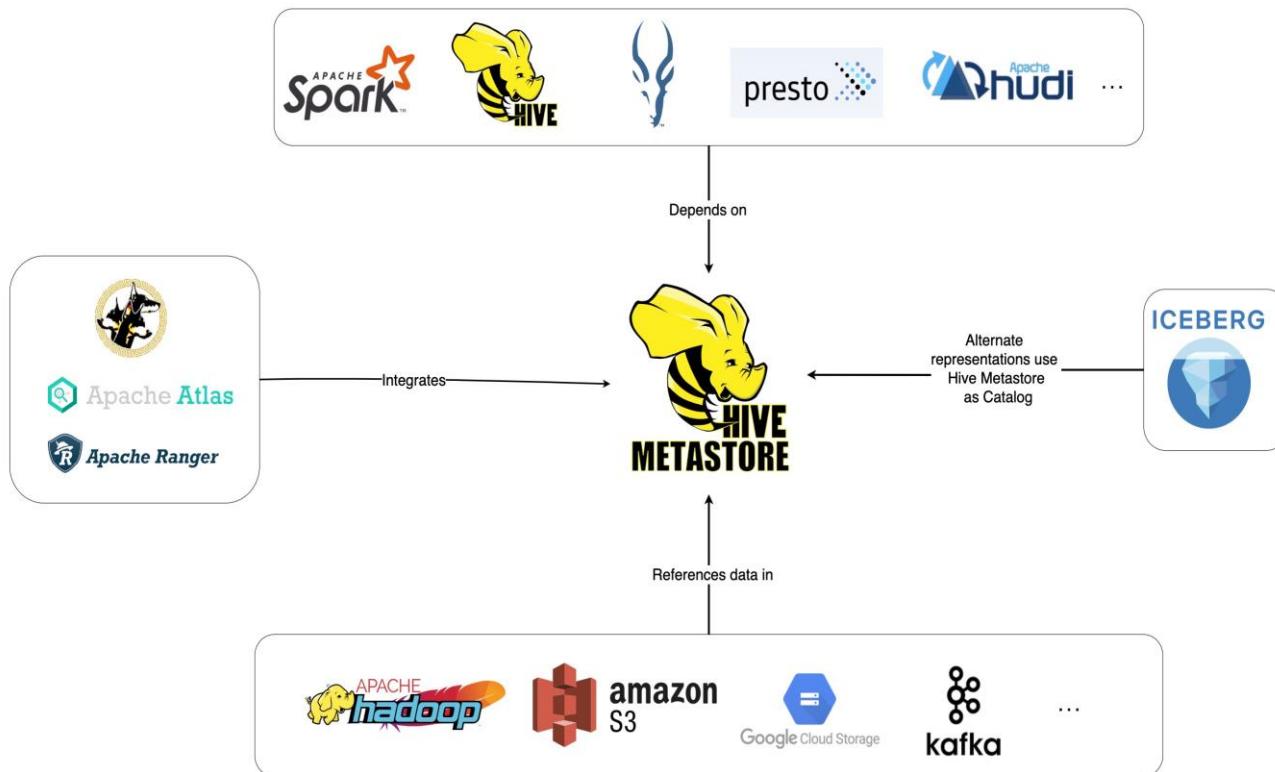
## Local Metastore:

**It is the Metastore service runs in the same JVM** in which the Hive service is running and connects to a database running in a separate JVM. Either on the same machine or on a remote machine.

## Remote Metastore:

In this configuration, **the Metastore service runs on its own separate JVM** and not in the Hive service JVM.

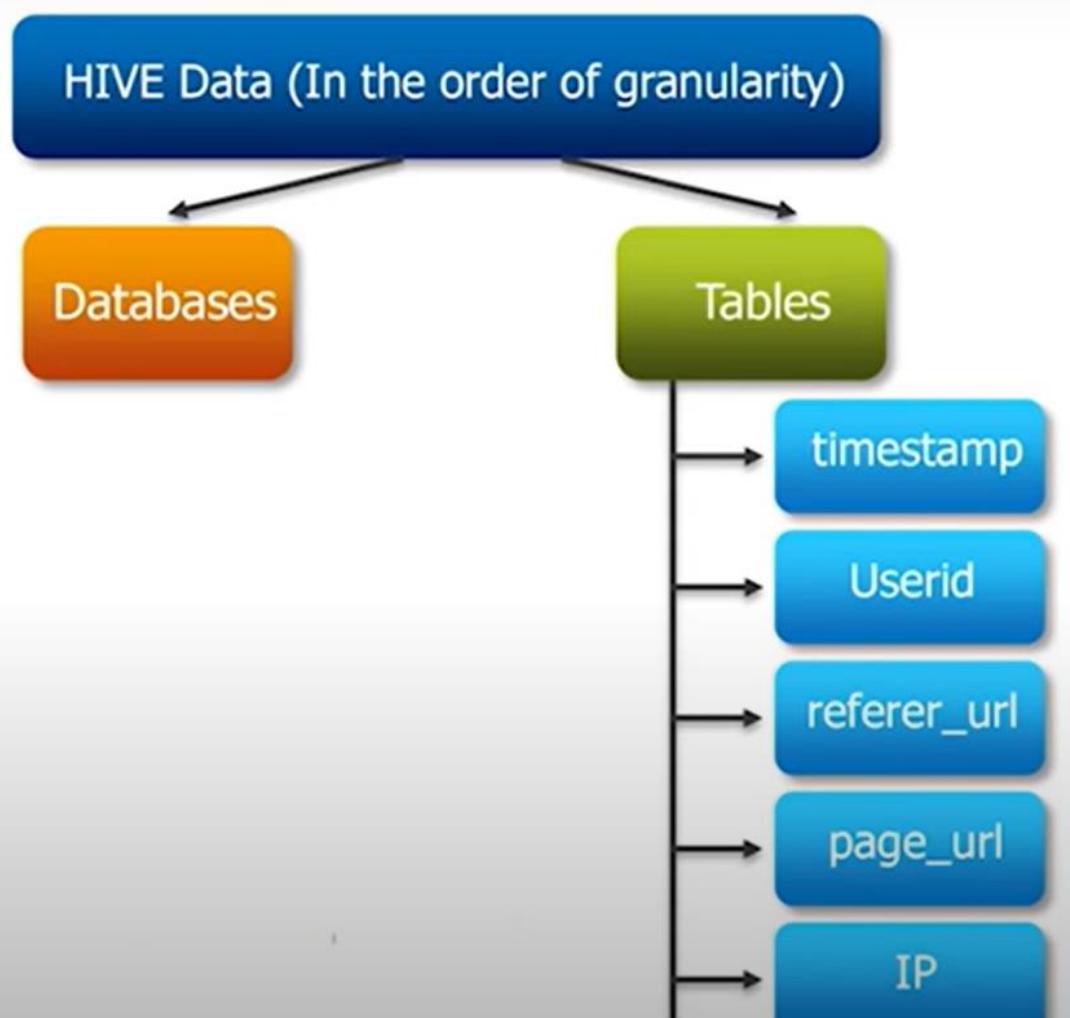
# Hive Metastore(HMS)



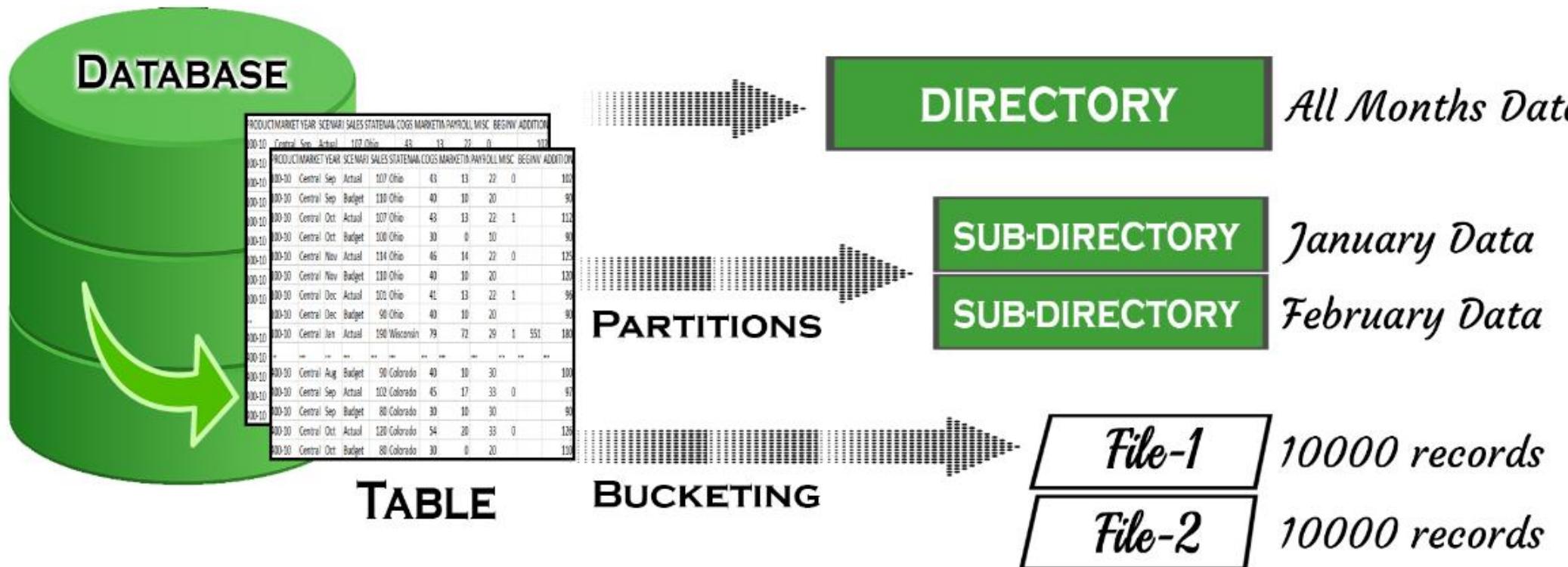
- The Hive Metastore (HMS) is a **central repository** of metadata for Hive tables and partitions in a relational database, and provides clients (including Hive, Impala and Spark) access to this information using the metastore service API.
- It has become a **building block for data lakes** that utilize the diverse world of open-source software, such as Apache Spark and Presto. In fact, a whole ecosystem of tools, open-source and otherwise, are built around the Hive Metastore.

# Hive Data Models

- ✓ Databases
  - ✓ Namespaces
- ✓ Tables
  - ✓ Schemas in namespaces
- ✓ Partitions
  - ✓ How data is stored in HDFS
  - ✓ Grouping data bases on some column
  - ✓ Can have one or more columns
- ✓ Buckets or Clusters
  - ✓ Partitions divided further into buckets bases on some other column
  - ✓ Use for data sampling



# Hive Data Models



# Create Database :

- ✓ **Create Database**
  - ✓ `Create database retail;`
- ✓ **Use Database**
  - ✓ `Use retail;`
- ✓ **Create table for storing transactional records**
  - ✓ `Create table txnrecords(txnno INT, txndate STRING, custno INT, amount DOUBLE, category STRING, product STRING, city STRING, state String, Spendby String )`
  - ✓ `Row format delimited`
  - ✓ `Fields terminated by ',' stored as textfile`

`CREATE DATABASE |SCHEMA [IF NOT EXISTS] <database name>`

`hive> CREATE DATABASE [IF NOT EXISTS] userdb;`

`hive> CREATE SCHEMA userdb;`

## Tables:

- Apache **Hive tables** are the same as the **tables** present in a **Relational Database**.
- We can perform **filter, project, join and union operations** on tables.
- Hive stores the **metadata** in a **relational database** and not in HDFS.
- Hive has **two types of tables** which are as follows:

### 1. Managed Table:

*Command:*

```
CREATE TABLE <table_name> (column1 data_type, column2 data_type);  
LOAD DATA INPATH <HDFS_file_location> INTO table managed_table;
```

### 2. External Table:

*Command:*

```
CREATE EXTERNAL TABLE <table_name> (column1 data_type, column2 data_type) LOCATION '<table_hive_location>';  
LOAD DATA INPATH '<HDFS_file_location>' INTO TABLE <table_name>;
```

For *external table*, Hive is not responsible for managing the data.

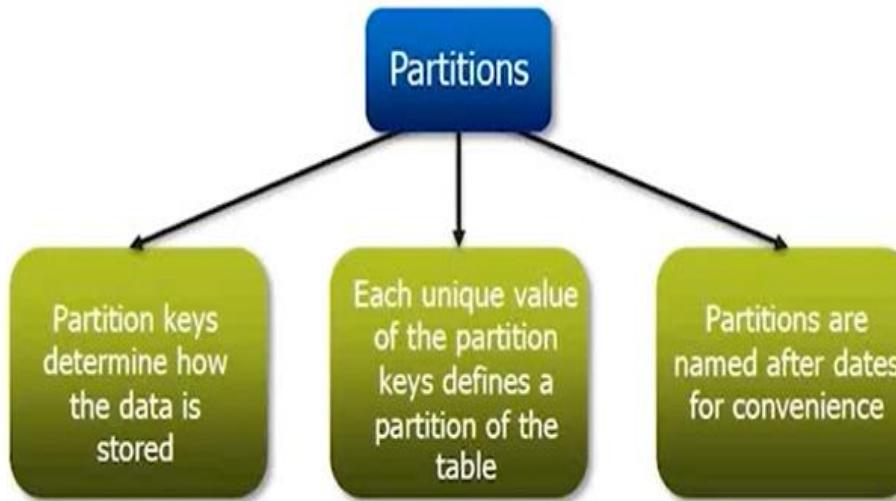
# External Table

- ✓ **Create the table in another hdfs location** and not in warehouse directory
- ✓ **Not managed by hive**
  - ✓ `CREATE EXTERNAL TABLE external_Table (dummy STRING)`
  - ✓ `LOCATION '/user/notroot/external_table';`
- ✓ **Hive does not delete the table (or hdfs files) even when the tables are dropped.**  
It leaves the table untouched and only metadata about the tables are deleted.

Need to specify the hdfs location

# Partition:

**Partition** means dividing a table into coarse grained parts based on the value of a partition column such as a date. This make it faster to do queries on slices of the data.



- student\_details containing the student information of students like student\_id, name, department, year, etc. Now, if I perform partitioning based on department column, the information of all the students belonging to a particular department will be stored together in that very partition.

- Partition: is used to group similar data types **together based on column or partition key**.
- Hive organizes tables into partitions.
- In other words, we can say that **partition is used to create a sub-directory in the table directory**.
- Each table can **have one or more partition keys** to identify a particular partition.
- Partitioning is used in **Hive to reduce the query latency**. Instead of scanning the entire tables, it scans only the relevant partitions and corresponding datasets.

# Dynamic Partitioning (Columns - values known – Execution Time)

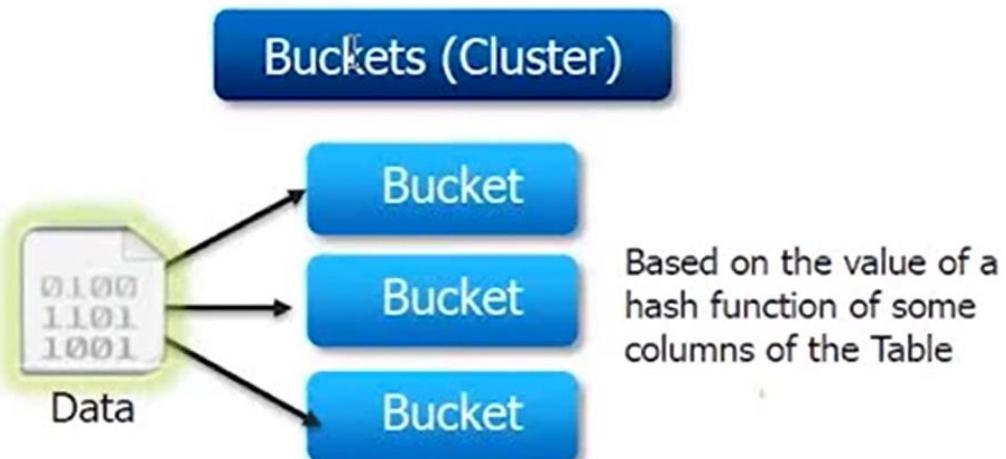
- A partitioning is called dynamic partitioning while loading the data into the Hive table. In other words, we can say that dynamic partitioning values for partition columns in the runtime.
- **Dynamic partitioning is used in the following cases:**
- While we Load data from an existing non-partitioned table, it is used to improve the sampling. Thus it decreases the query latency.
- While we do not know all the values of the partitions beforehand, so, finding these partition values manually from a huge dataset is a tedious task.

# Hive Data Model :

| Name       | Description   |
|------------|---|
| Database   | Namespace for tables  |
| Tables     | Similar to tables in RDBMS<br>Support filter, projection, join and union operations<br>The table data stores in a directory in HDFS         |
| Partitions | Table can have one or more partition keys that tell how the data stores   |
| Buckets    | Data in each partition further divides into buckets based on hash of a column in the table.<br>Stored as a file in the partition directory. |

# Buckets:

- ✓ Buckets give extra structure to the data that may be used for more efficient queries.
- ✓ A join of two tables that are bucketed on the same columns – including the join column can be implemented as a Map Side Join.
- ✓ Bucketing by user ID means we can quickly evaluate a user based query by running it on a randomized sample of the total set of users.



# Bucketing & Reasoning:

- Bucketing is the concept of **breaking data down into ranges**, which are known as buckets.
- Bucketing is mainly a **data organizing technique**.
- It is similar to partitioning in Hive with an **added functionality** that it divides large datasets into more **manageable parts known as buckets**.
- The partitioning into buckets can give **extra structure to the data** to use for more **efficient queries**.
- The range for a bucket is determined by **the hash value of one or more columns** in the dataset.
- **Reasons:**

**There are two main reasons for performing bucketing to a partition:**

- We perform bucketing to a partition because a **map side join** requires the data belonging to a unique join key to be **present in the same partition**.
- Bucketing facilitates us to decrease the **query time**, and it also makes the **sampling process more efficient**.

# Optimization: Indexing

- Indexing in Hive is a Hive query optimization technique.
- It is mainly used to speed up the access of a column or set of columns in a Hive database.
- With the use of the index, the Hive database system does not need to read all rows in the table, especially that one has selected.

# Hive Data Model:

## Partitions:

*Command:*

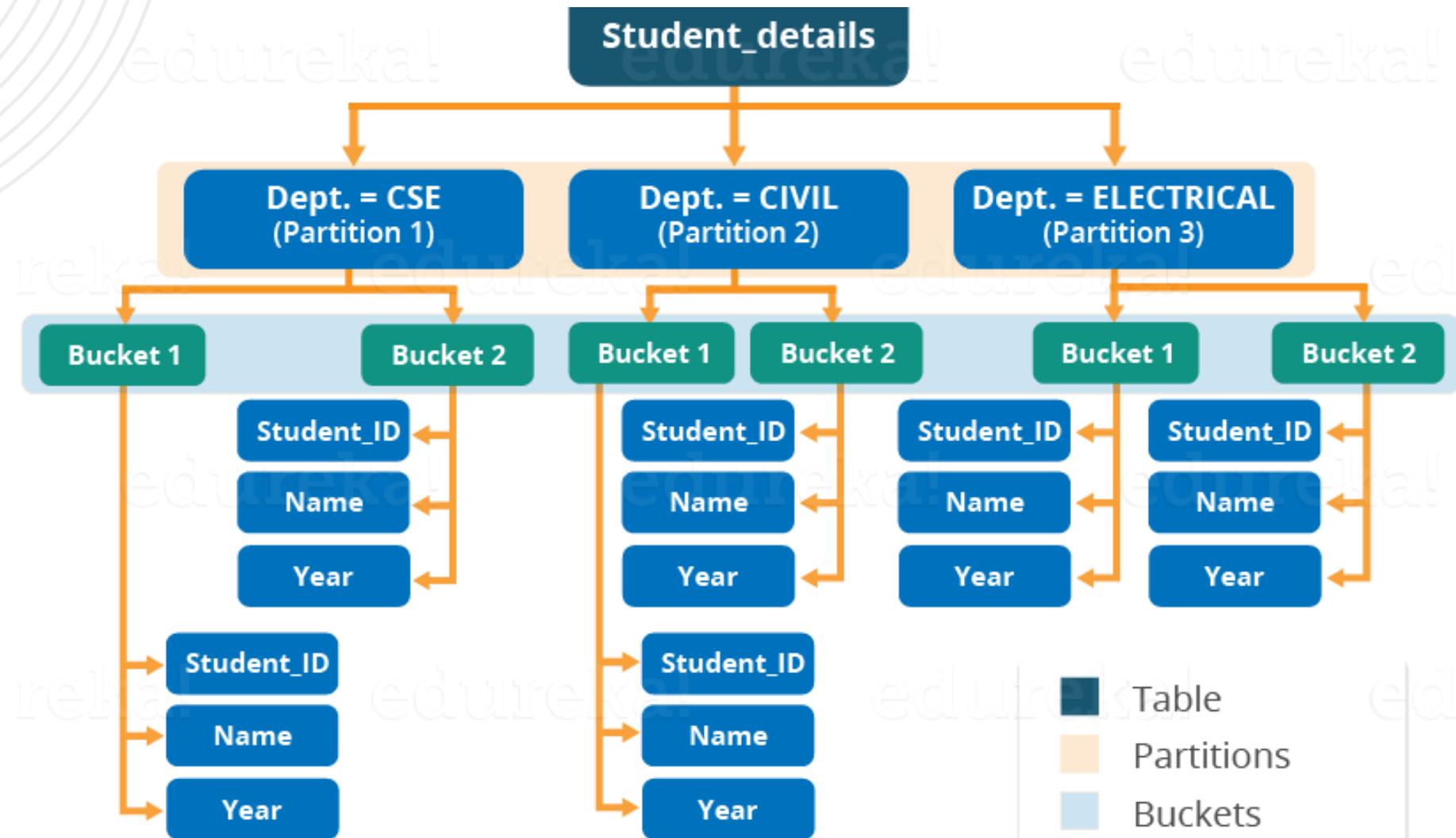
```
CREATE TABLE table_name (column1 data_type, column2 data_type) PARTITIONED BY (partition1 data_type, partition2 data_type,...);
```

Hive organizes tables into partitions for grouping similar type of data together based on a column or partition key. Each Table can have one or more partition keys to identify a particular partition. This allows us to have a faster query on slices of the data.

## Buckets:

*Commands:*

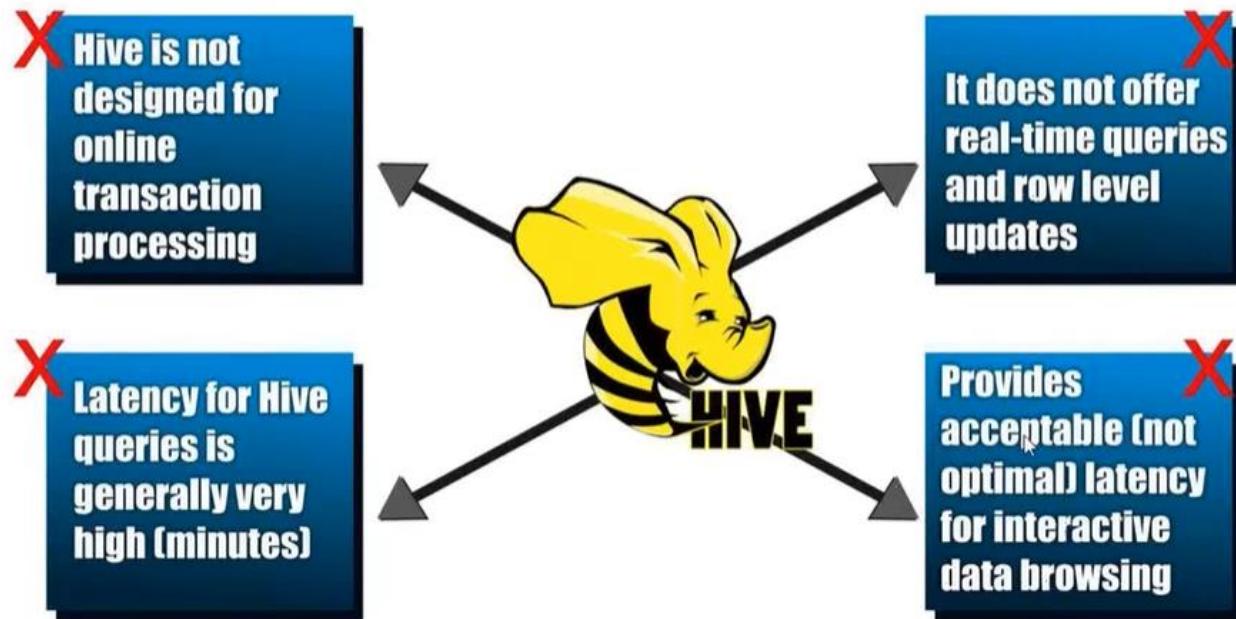
```
CREATE TABLE table_name PARTITIONED BY (partition1 data_type, partition2 data_type,...) CLUSTERED BY (column_name1, column_name2, ...) SORTED BY (column_name [ASC|DESC], ...)] INTO num_buckets BUCKETS;
```



# MODES OF HIVE

- **Local Mode**
- **MapReduce Mode**
  - In Hive, the Map reduce mode is used in the following conditions:
  - To perform on a large amount of data sets and query going to execute in a parallel way .
  - When Hadoop has multiple data nodes and is distributed across different nodes, we should use this mode.
  - To achieve better performance.

# Limitation of Hive:



- Hive Does not provide **update, alter and deletion of records** in the database.
- Not developed for "**Unstructured Data**"
- Not designed for "**Real Time Queries**".
- Performs the **partition** always from the **last column**.
- OLTP:** No. Because Hive does not provide **insert and update at the row level**, it is not suitable for the OLTP system.

Table 1: Primitive, String, Date/time datatypes

# Hive Data Types :

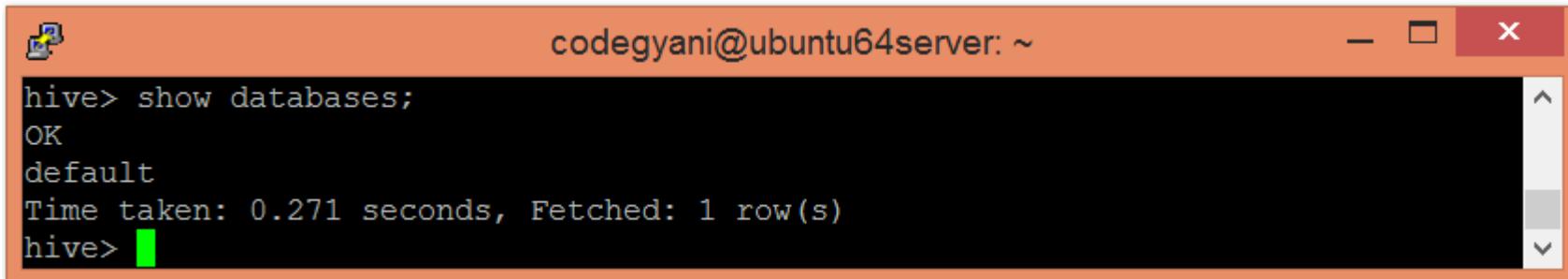
- Hive Defines various primitive, complex, string, date/time, collection data types and file formats.
- For handling & Storing different data formats.
- Table 1: Gives primitive, string , date/time and complex Hive data types & its description.

| Data Type Name | Description  |
|----------------|--|
| TINYINT        | 1 byte signed integer. Postfix letter is Y.  |
| SMALLINT       | 2 byte signed integer. Postfix letter is S.  |
| INT            | 4 byte signed integer  |
| BIGINT         | 8 byte signed integer. Postfix letter is L.  |
| FLOAT          | 4 byte single-precision floating-point number  |
| DOUBLE         | 8 byte double-precision floating-point number  |
| BOOLEAN        | True or False  |
| TIMESTAMP      | UNIX timestamp with optional nanosecond precision. It supports java.sql.Timestamp format “YYYY-MM-DD HH:MM:SS.fffffffff” |
| DATE           | YYYY-MM-DD format  |
| VARCHAR        | 1 to 65355 bytes. Use single quotes (‘ ’) or double quotes (“ ”)   |
| CHAR           | 255 bytes  |
| DECIMAL        | Used for representing immutable arbitrary precision. DECIMAL (precision, scale) format                                   |

## Hive - Create Database

- In Hive, the database is considered as a **catalog or namespace of tables**. So, we can maintain **multiple tables within a database** where a **unique name is assigned to each table**. Hive also provides a **default database** with a name **default**.
- check the list of existing databases, follow the below command: -

```
hive> show databases;
```

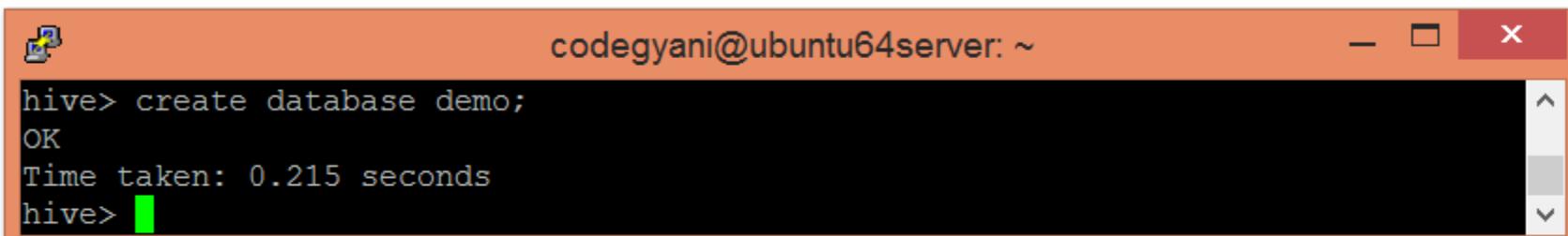


A screenshot of a terminal window titled "codegyani@ubuntu64server: ~". The window contains the following text:

```
hive> show databases;
OK
default
Time taken: 0.271 seconds, Fetched: 1 row(s)
hive> █
```

- Let's create a new database by using the following command: -

```
hive> create database demo;
```



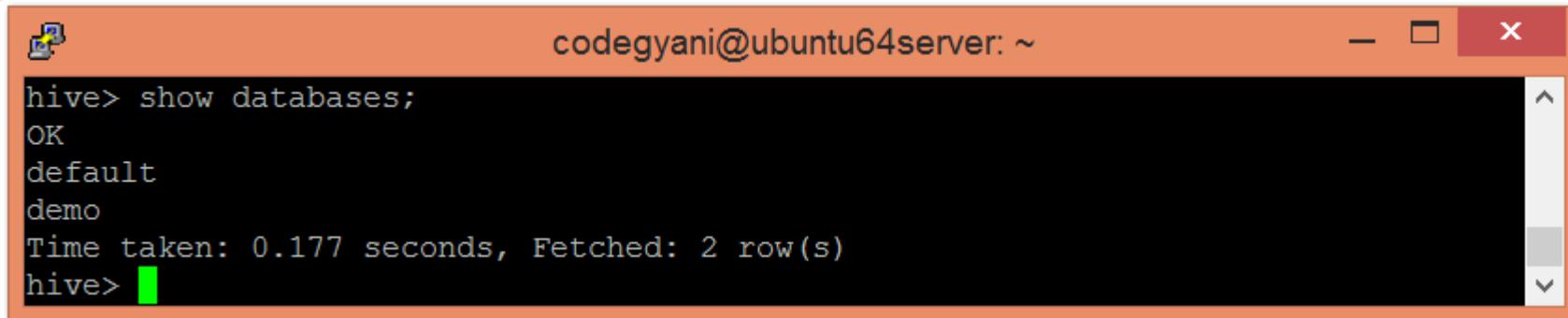
A screenshot of a terminal window titled "codegyani@ubuntu64server: ~". The window contains the following text:

```
hive> create database demo;
OK
Time taken: 0.215 seconds
hive> █
```

## Hive - Create Database

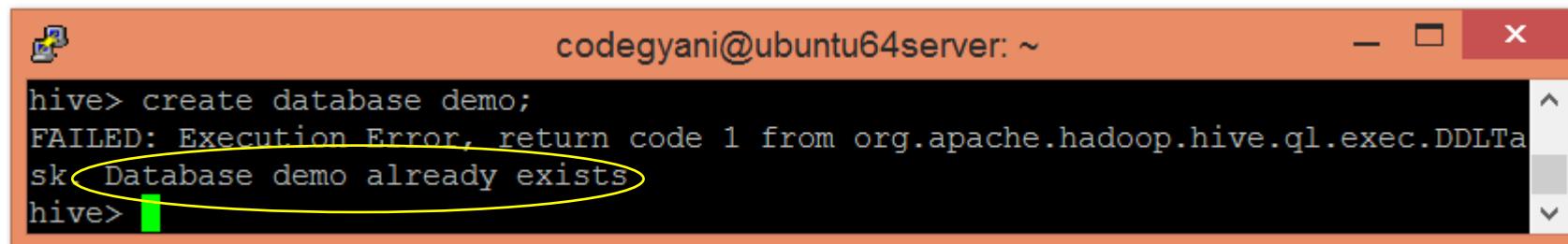
- Let's check the existence of a newly created database.

```
hive> show databases;
```



```
hive> show databases;
OK
default
demo
Time taken: 0.177 seconds, Fetched: 2 row(s)
hive>
```

- Each database must contain a **unique name**. If we create **two databases with the same name**, the following **error** generates: -



```
hive> create database demo;
FAILED: Execution Error, return code 1 from org.apache.hadoop.hive.ql.exec.DDLTask
sk Database demo already exists
hive>
```

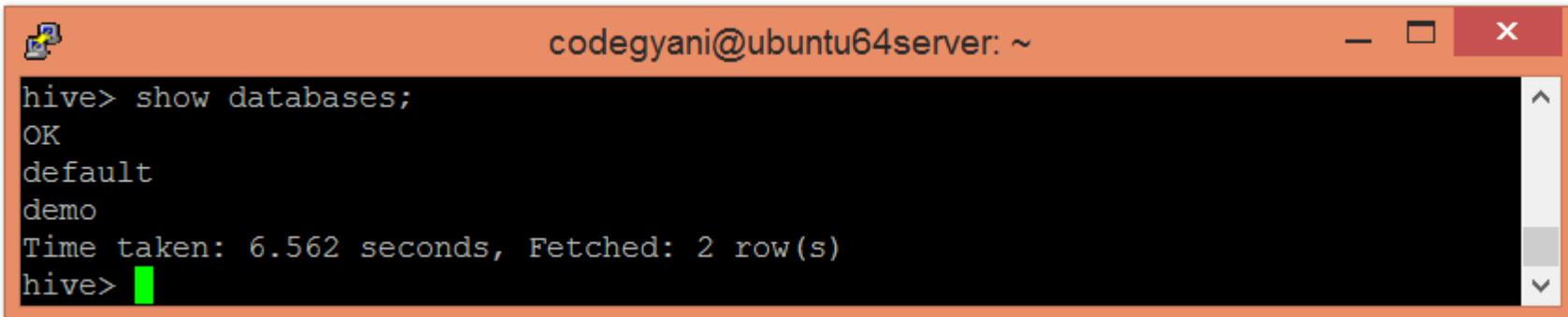
- If we want to **suppress the warning** generated by Hive on creating the database with the same name, follow the below command: -

```
hive> create database if not exists demo;
```

## Hive - Drop Database

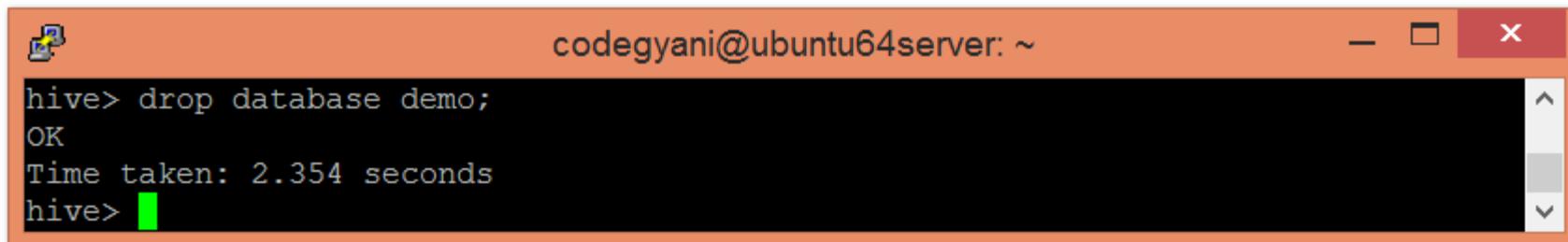
- check the **list of existing databases** by using the following command:-

**hive> show databases;**



```
hive> show databases;
OK
default
demo
Time taken: 6.562 seconds, Fetched: 2 row(s)
hive> █
```

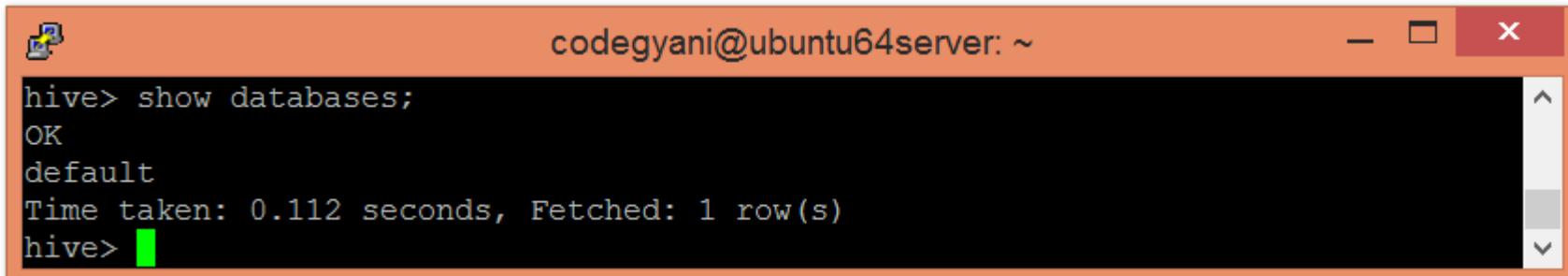
**hive> drop database demo;**



```
hive> drop database demo;
OK
Time taken: 2.354 seconds
hive> █
```

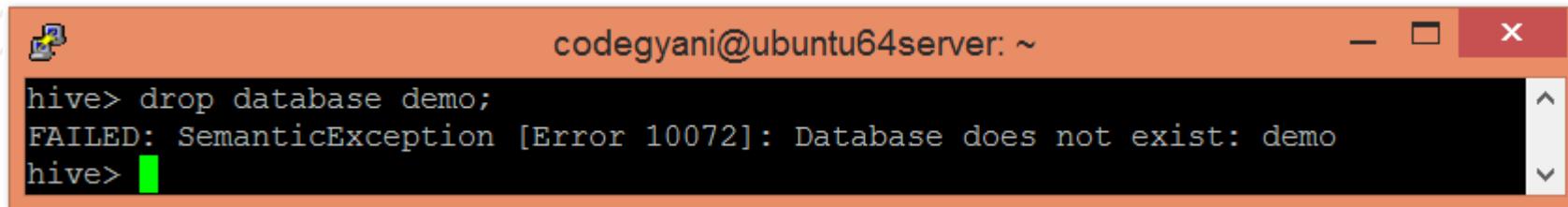
Let's check whether the database is dropped or not.

**hive> show databases;**



```
hive> show databases;
OK
default
Time taken: 0.112 seconds, Fetched: 1 row(s)
hive> █
```

- If we try to drop the database that doesn't exist, the following error generates:



A screenshot of a terminal window titled "codegyani@ubuntu64server: ~". The window contains the following text:  
hive> drop database demo;  
FAILED: SemanticException [Error 10072]: Database does not exist: demo  
hive>

- if we want to suppress the warning generated by Hive on creating the database with the same name, follow the below command:-

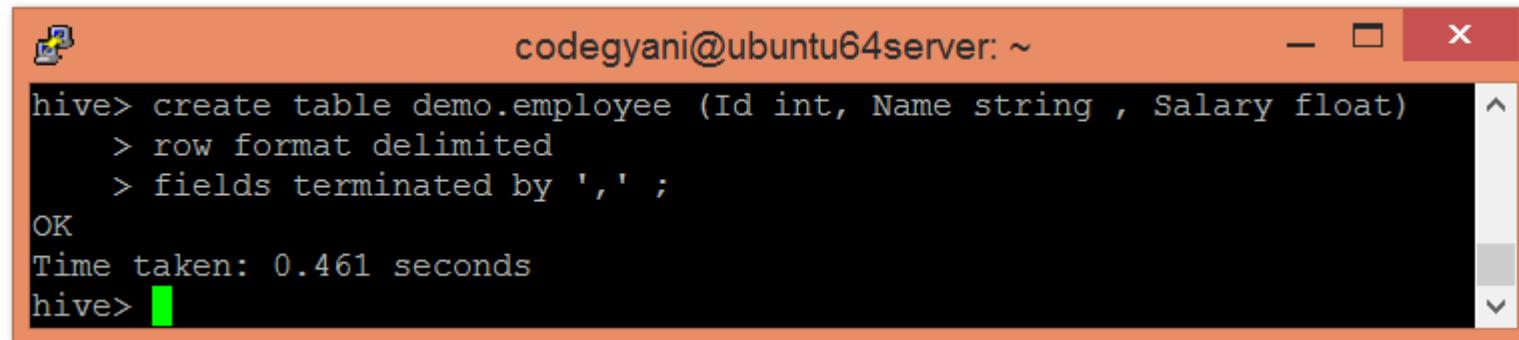
**hive> drop database if exists demo;**

- In Hive, it is not allowed to drop the database that contains the tables directly. In such a case, we can drop the database either by dropping tables first or use **Cascade keyword** with the command.

**hive> drop database if exists demo cascade;**

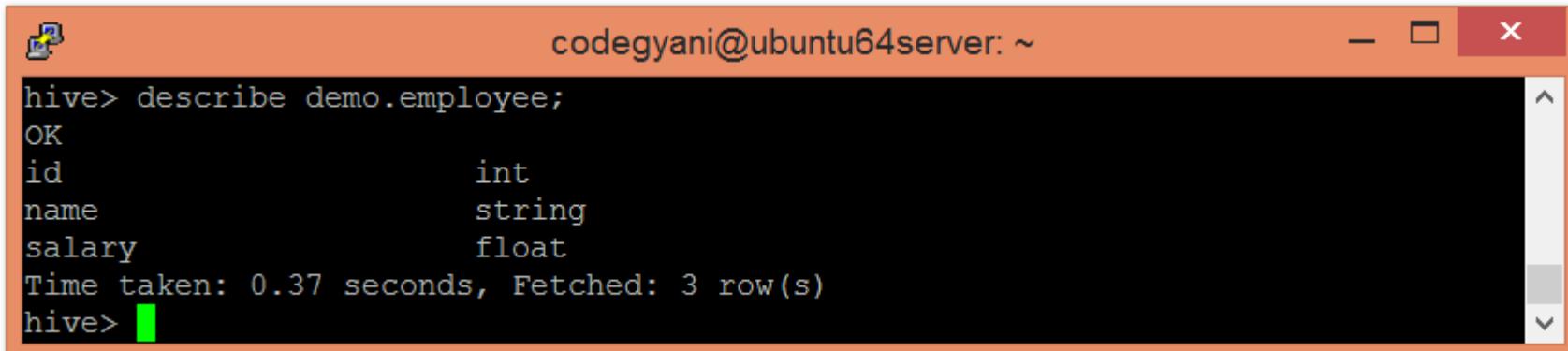
## Hive - Create Table

- In Hive, we can create a table by using the **conventions similar to the SQL**. It supports a wide range of flexibility where the data files for tables are stored. It provides two types of table:-
  - create an **internal/managed table** by using the following command:-  
**hive> create table demo.employee (Id int, Name string , Salary float)  
row format delimited  
fields terminated by ',';**



```
codegyani@ubuntu64server: ~
hive> create table demo.employee (Id int, Name string , Salary float)
      > row format delimited
      > fields terminated by ',' ;
OK
Time taken: 0.461 seconds
hive>
```

- The **metadata** of the created table by using the following command:-



```
codegyani@ubuntu64server: ~
hive> describe demo.employee;
OK
id                  int
name                string
salary              float
Time taken: 0.37 seconds, Fetched: 3 row(s)
hive>
```

## Hive - Create Table

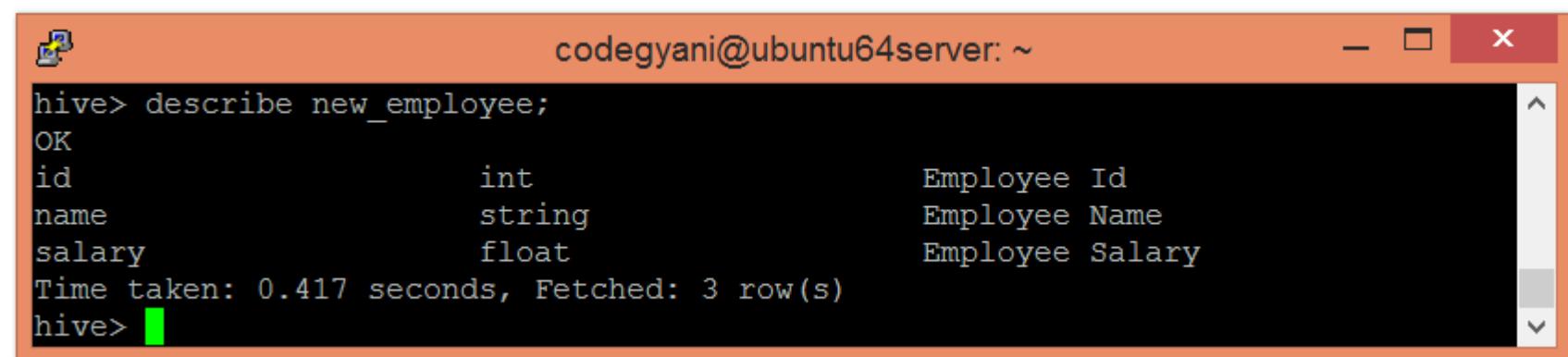
- Let's see the result when we try to create the existing table again.
- In such a case, the exception occurs. If we want to ignore this type of exception, we can use **if not exists** command while creating the table.
- While creating a table, we can add the comments to the columns and can also define the table properties.

```
codegyani@ubuntu64server: ~
hive> create table demo.employee (Id int, Name string , Salary float)
      > row format delimited
      > fields terminated by ',' ;
FAILED: Execution Error, return code 1 from org.apache.hadoop.hive.ql.exec.DDLTask. AlreadyExistsException(message:Table employee already exists)
hive>
```

```
codegyani@ubuntu64server: ~
hive> create table if not exists demo.employee (Id int, Name string , Salary float)
      > row format delimited
      > fields terminated by ',' ;
OK
Time taken: 0.166 seconds
hive>
```

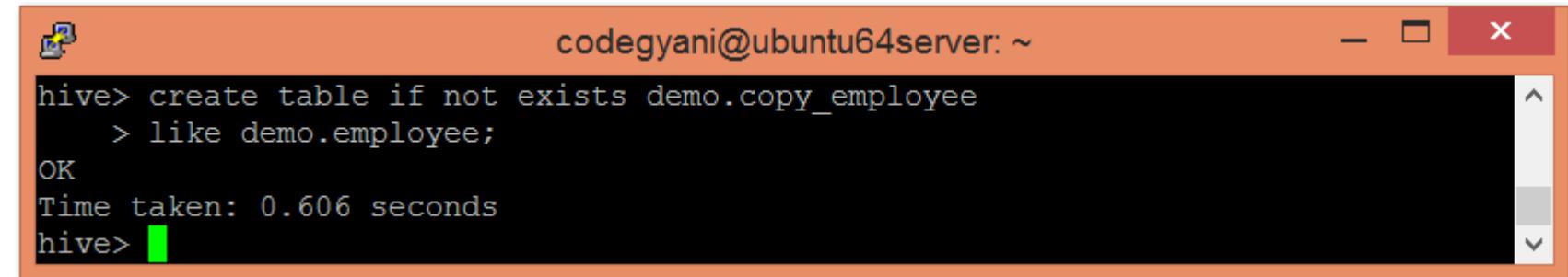
```
codegyani@ubuntu64server: ~
hive> create table demo.new_employee (Id int comment 'Employee Id', Name string
      comment 'Employee Name', Salary float comment 'Employee Salary')
      > comment 'Table Description'
      > TBLProperties ('creator'='Gaurav Chawla', 'created_at' = '2019-06-06 11:00')
      > ;
OK
Time taken: 3.236 seconds
```

```
hive> describe new_employee;
```



```
hive> describe new_employee;
OK
id          int          Employee Id
name        string        Employee Name
salary      float         Employee Salary
Time taken: 0.417 seconds, Fetched: 3 row(s)
hive>
```

- Hive allows creating a **new table by using the schema of an existing table.**



```
hive> create table if not exists demo.copy_employee
      > like demo.employee;
OK
Time taken: 0.606 seconds
hive>
```

## Hive - Load Data

Once the **internal table has been created**, the next step is to **load the data into it**. So, in Hive, we can easily load data from **any file to the database**.

- Let's load the data of the file into the database by using the following command:-
- Here, **emp\_details** is the file name that contains the data.
- If we want to **add more data** into the current database, execute the same query again by **just updating the new file name**.

```
load data local inpath  
'/home/codegyani/hive/emp  
_details1' into table  
demo.employee;
```

```
hive> load data local inpath '/home/codegyani/hive/emp_details' into table demo.  
employee;  
Loading data to table demo.employee  
Table demo.employee stats: [numFiles=1, numRows=0, totalSize=50, rawDataSize=0]  
OK  
Time taken: 7.505 seconds  
hive>
```

```
hive> select * from demo.employee;  
OK  
1      "Gaurav"      30000.0  
2      "Aryan"       20000.0  
3      "Vishal"       40000.0  
Time taken: 3.32 seconds, Fetched: 3 row(s)  
hive>
```

Hadoop Overview Datanodes Snapshot Startup Progress Utilities ▾

## Browse Directory

/user/hive/warehouse/demo.db/employee

Go!

| Permission | Owner     | Group      | Size | Last Modified         | Replication | Block Size | Name        |
|------------|-----------|------------|------|-----------------------|-------------|------------|-------------|
| -rwxr-xr-x | codegyani | supergroup | 50 B | 4/15/2019, 7:41:53 PM | 1           | 128 MB     | emp_details |

# External Table

To create an external table, follow the below steps: -

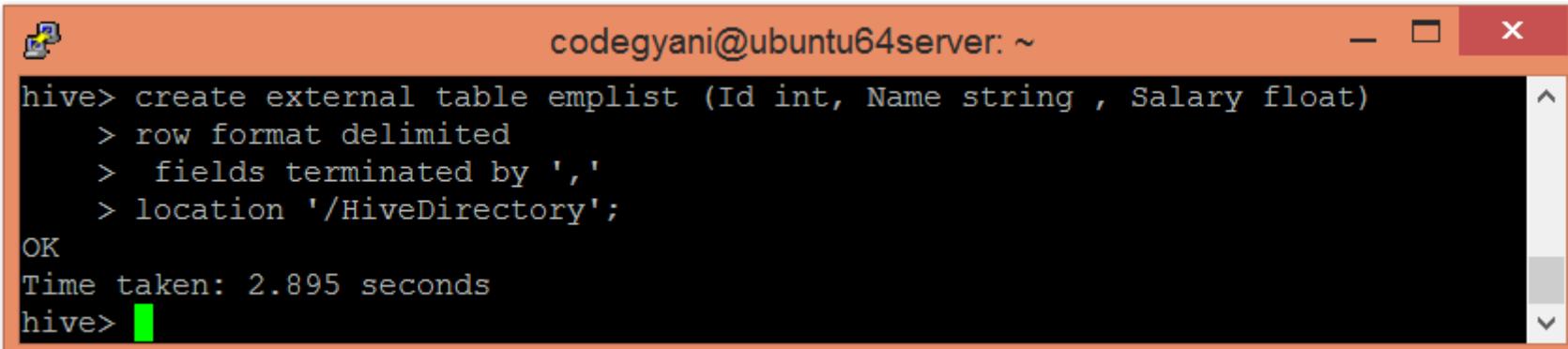
- Let's create a directory on HDFS by using the following command: -

```
hdfs dfs -mkdir /HiveDirectory
```

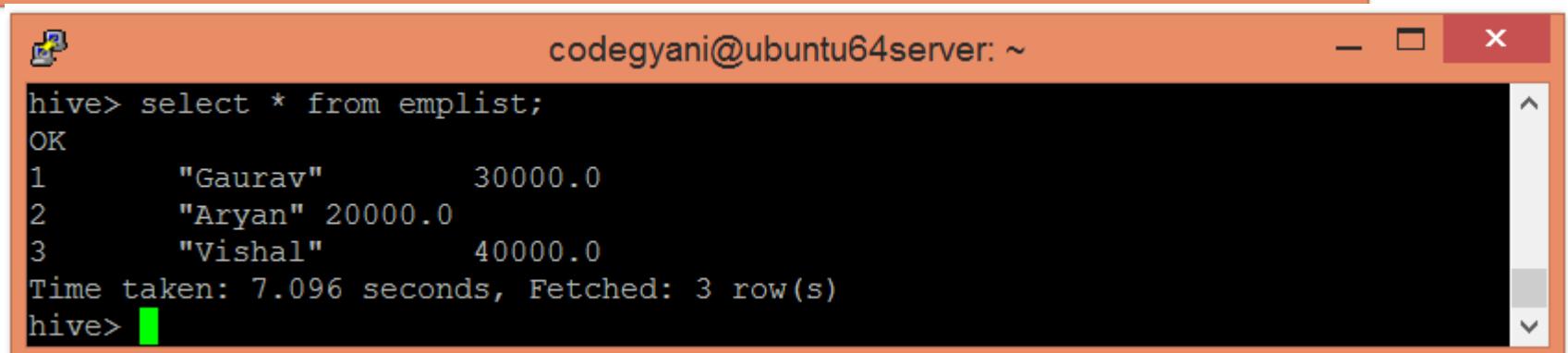
- Now, store the file on the created directory.

```
hdfs dfs -put hive/emp_details /HiveDirectory
```

- Let's create an external table using the following command: -



```
hive> create external table emplist (Id int, Name string , Salary float)
      > row format delimited
      > fields terminated by ','
      > location '/HiveDirectory';
OK
Time taken: 2.895 seconds
hive>
```



```
hive> select * from emplist;
OK
1      "Gaurav"      30000.0
2      "Aryan"       20000.0
3      "Vishal"       40000.0
Time taken: 7.096 seconds, Fetched: 3 row(s)
hive>
```

- Now, we can use the following command to retrieve the data: -

- In Hive, if we try to **load unmatched data** (i.e., one or more column data doesn't match the data type of specified table columns), it will **not throw any exception**. However, **it stores the Null value at the position of unmatched tuple**.

- Let's add one more file to the current table. This file contains the unmatched data.

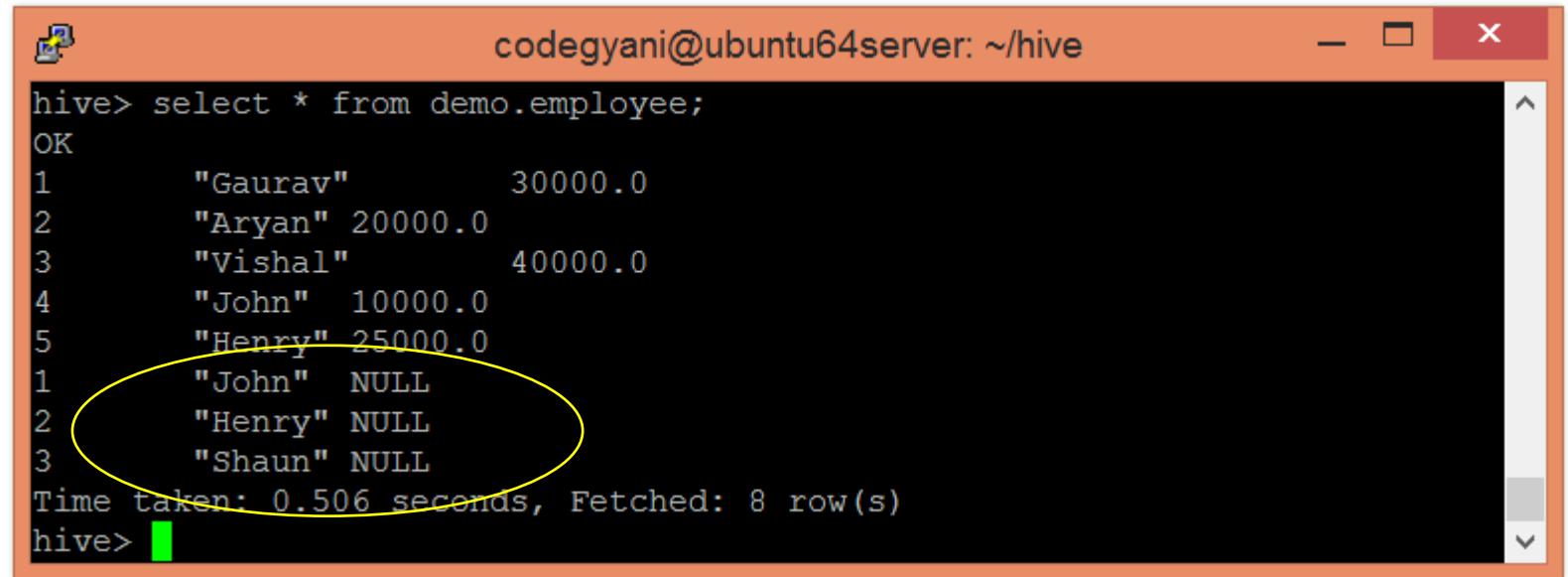
```
GNU nano 2.2.6          File: emp_details2
1, "John", "Woakes"
2, "Henry", "William"
3, "Shaun", "Morris"

[ Read 3 lines ]
^G Get Help ^O WriteOut ^R Read File^Y Prev Page^K Cut Text ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is ^V Next Page^U Uncut Tex^T To Spell
```

- load the data into the table.

```
codegyani@ubuntu64server: ~/hive
hive> load data local inpath '/home/codegyani/hive/emp_details2' into table demo.employee;
Loading data to table demo.employee
Table demo.employee stats: [numFiles=3, numRows=0, totalSize=138, rawDataSize=0]
OK
Time taken: 15.812 seconds
hive>
```

- Null values at the position of unmatched data.

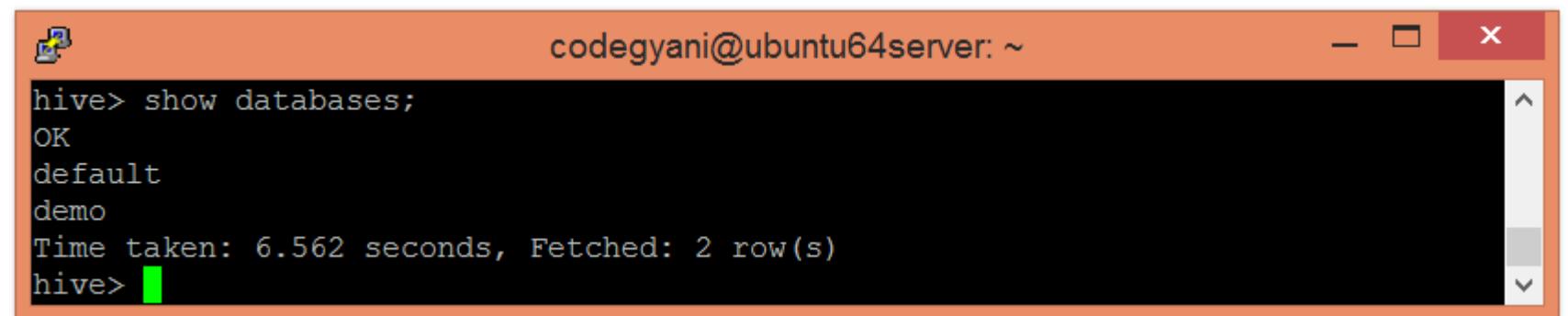


```
codegyani@ubuntu64server: ~/hive
hive> select * from demo.employee;
OK
1 "Gaurav" 30000.0
2 "Aryan" 20000.0
3 "Vishal" 40000.0
4 "John" 10000.0
5 "Henry" 25000.0
1 "John" NULL
2 "Henry" NULL
3 "Shaun" NULL
Time taken: 0.506 seconds, Fetched: 8 row(s)
hive>
```

## Hive - Drop Table

Hive facilitates us to drop a table by using the **SQL drop table command**. Let's follow the below steps to drop the table from the database.

- check the list of existing databases by using the following command: -
- select the **database from which we want to delete the table**

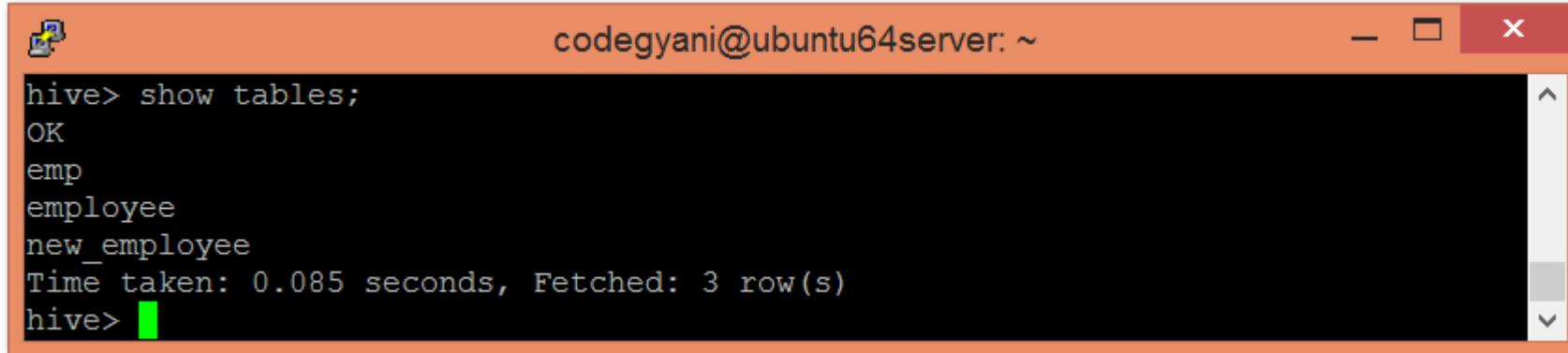


```
codegyani@ubuntu64server: ~
hive> show databases;
OK
default
demo
Time taken: 6.562 seconds, Fetched: 2 row(s)
hive>
```



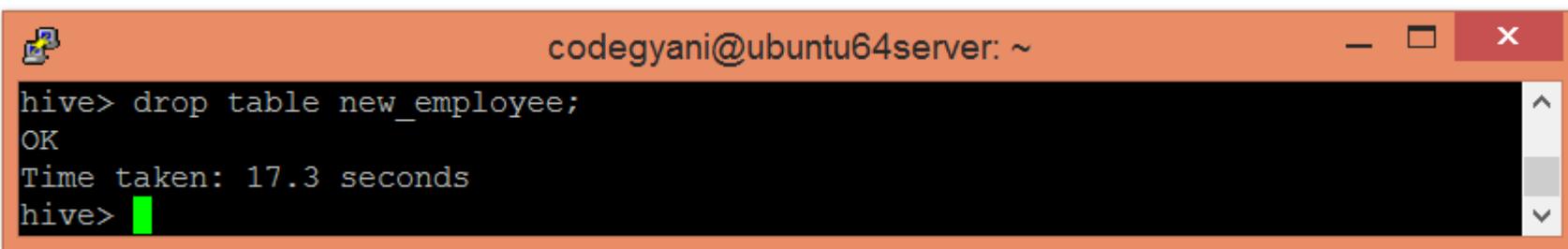
```
codegyani@ubuntu64server: ~
hive> use demo;
OK
Time taken: 0.148 seconds
hive>
```

- check the list of **existing tables** in the corresponding database.



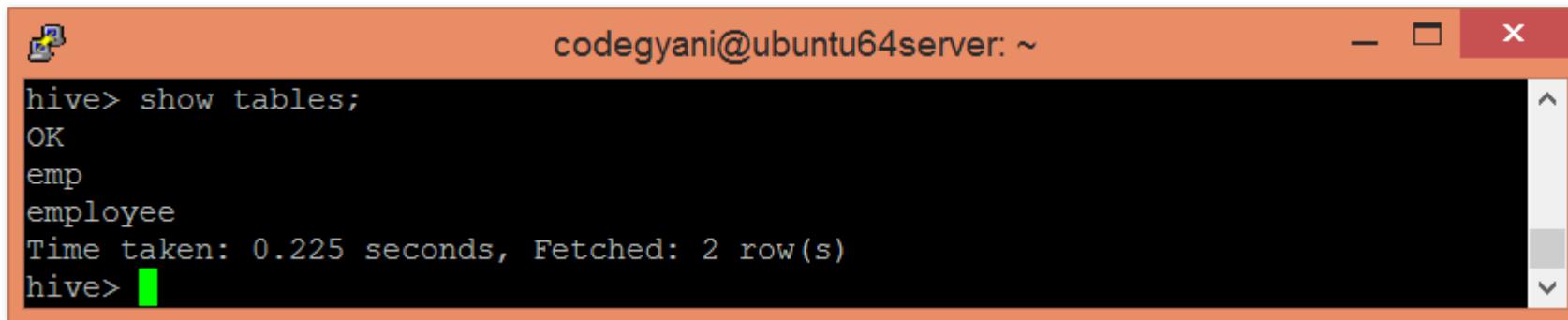
```
hive> show tables;
OK
emp
employee
new_employee
Time taken: 0.085 seconds, Fetched: 3 row(s)
hive>
```

- drop the table by using the following command:



```
hive> drop table new_employee;
OK
Time taken: 17.3 seconds
hive>
```

- check whether the table is dropped or not.
- As we can see, the table **new\_employee** is not present in the list. Hence, the table is dropped successfully.

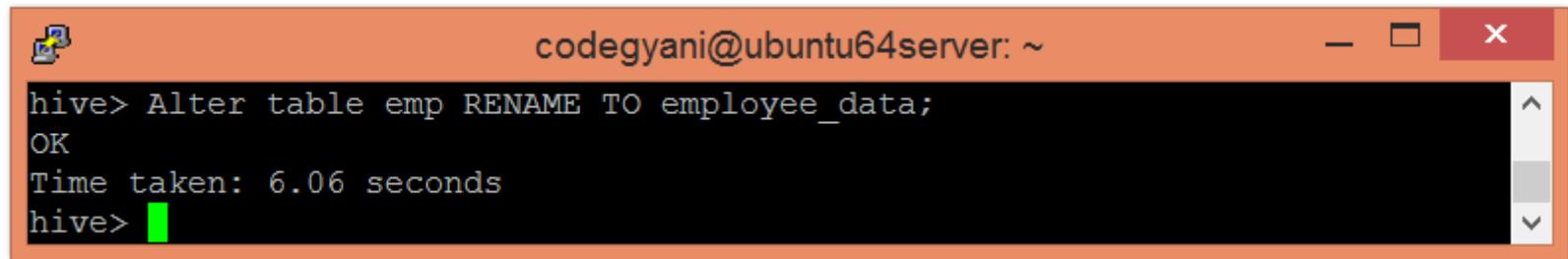


```
hive> show tables;
OK
emp
employee
Time taken: 0.225 seconds, Fetched: 2 row(s)
hive>
```

# Hive - Alter Table

In Hive, we can perform modifications in the existing table like changing the table name, column name, comments, and table properties. It provides SQL like commands to alter the table.

- change the name of the table by using the following command



A screenshot of a terminal window titled 'codegyani@ubuntu64server: ~'. The window contains the following text:  
hive> Alter table emp RENAME TO employee\_data;  
OK  
Time taken: 6.06 seconds  
hive>

- In Hive, we can add one or more columns in an existing table by using the following signature:

**Alter table employee\_data add columns (age int);**

- change the name of the column by using the following command: -

**Alter table employee\_data change name first\_name string;**

- alter table employee\_data replace columns( id string, first\_name string, age int);**

# Partitioning in Hive

The partitioning in Hive can be executed **in two ways** -

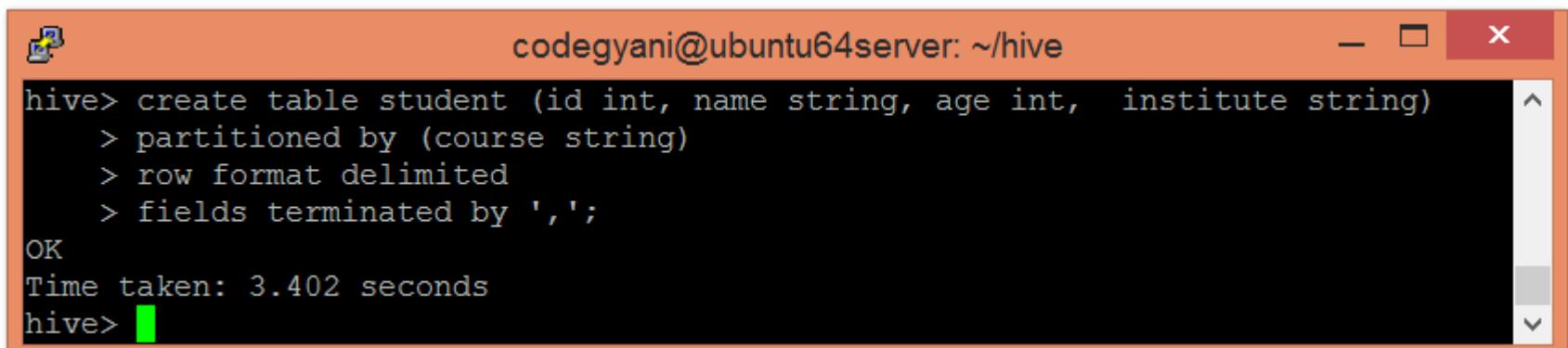
- Static partitioning
  - Dynamic partitioning
- 
- In static or manual partitioning, it is required to **pass the values of partitioned columns manually while loading the data into the table**. Hence, the data file doesn't contain the partitioned columns.

## Example of Static Partitioning

First, select the database in which we want to create a table.

```
hive> use test;
```

- Create the table and provide the partitioned columns by using the following command



A screenshot of a terminal window titled "codegyani@ubuntu64server: ~/hive". The window shows the following command being run:

```
hive> create table student (id int, name string, age int, institute string)
      > partitioned by (course string)
      > row format delimited
      > fields terminated by ',';
```

The command is completed successfully with the message "OK" and a time taken of "3.402 seconds".

- Let's retrieve the information associated with the table.

```
hive> describe student;
OK
id          int
name        string
age         int
institute   string
course      string

# Partition Information
# col_name          data_type          comment
course      string

Time taken: 1.833 seconds, Fetched: 10 row(s)
hive>
```

- Load the data into the table and pass the values(key) of partition columns with it by using the following command: -

- Here, we are **partitioning** the students of an institute based on courses.

```
hive> load data local inpath '/home/codegyani/hive/student_details1' into table student
      > partition(course= "java");
Loading data to table test.student partition (course=jav
Partition test.student{course=jav} stats: [numFiles=1, numRows=0, totalSize=122
, rawDataSize=0]
OK
Time taken: 8.057 seconds
hive>
```

- Load the data of another file into the same table and pass the values of partition columns with it by using the following command: -

```
hive> load data local inpath '/home/codegyani/hive/student_details2' into table student
      > partition(course= "hadoop");
Loading data to table test.student partition (course=hadoop)
Partition test.student{course=hadoop} stats: [numFiles=1, numRows=0, totalSize=7
5, rawDataSize=0]
OK
Time taken: 2.402 seconds
hive>
```

- In the following screenshot, we can see that the **table student** is divided into two categories.

Hadoop Overview Datanodes Snapshot Startup Progress Utilities ▾

## Browse Directory

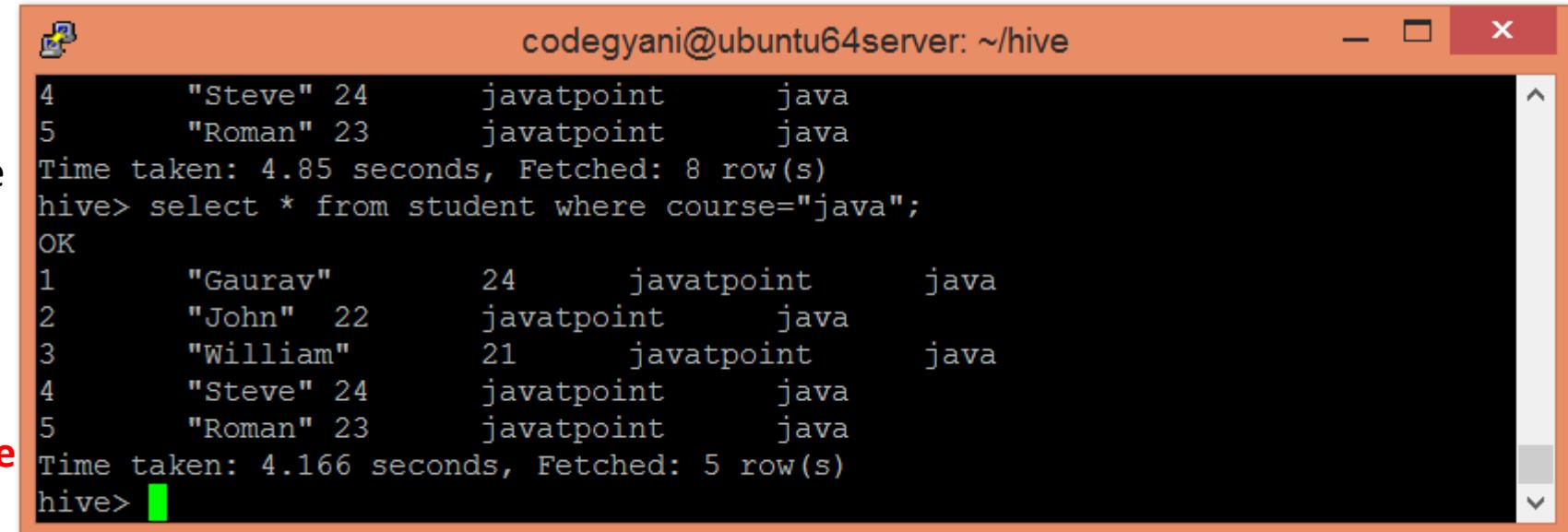
/user/hive/warehouse/test.db/student

| Permission | Owner    | Group      | Size | Last Modified        | Replication | Block Size | Name          |
|------------|----------|------------|------|----------------------|-------------|------------|---------------|
| drwxr-xr-x | codegyni | supergroup | 0 B  | 8/1/2019, 5:39:27 PM | 0           | 0 B        | course=hadoop |
| drwxr-xr-x | codegyni | supergroup | 0 B  | 8/1/2019, 5:37:13 PM | 0           | 0 B        | course=java   |

- Let's retrieve the entire data of the table by using the following command

```
codegyni@ubuntu64server: ~/hive
hive> select * from student;
OK
6      "Chris" 22      javatpoint      hadoop
7      "Hariss"   21      javatpoint      hadoop
8      "Angelina"24    NULL      NULL      hadoop
1      "Gaurav"   24      javatpoint      java
2      "John"     22      javatpoint      java
3      "William"  21      javatpoint      java
4      "Steve"    24      javatpoint      java
5      "Roman"   23      javatpoint      java
Time taken: 4.85 seconds, Fetched: 8 row(s)
hive>
```

- try to retrieve the data based on **partitioned columns** by using the following command: -
- In this case, we are **not examining the entire data**. Hence, this approach **improves query response time**.



```

codegyani@ubuntu64server: ~/hive
4      "Steve" 24      javatpoint      java
5      "Roman" 23      javatpoint      java
Time taken: 4.85 seconds, Fetched: 8 row(s)
hive> select * from student where course="java";
OK
1      "Gaurav"     24      javatpoint      java
2      "John"       22      javatpoint      java
3      "William"    21      javatpoint      java
4      "Steve"      24      javatpoint      java
5      "Roman"      23      javatpoint      java
Time taken: 4.166 seconds, Fetched: 5 row(s)
hive>

```

## Dynamic Partitioning

In dynamic partitioning, the **values of partitioned columns exist within the table**. So, it is not required to pass the values of partitioned columns manually.

- First, select the **database** in which we **want to create a table**.
- Enable the dynamic partition** by using the following →  
`hive> set hive.exec.dynamic.partition=true;`  
`hive> set hive.exec.dynamic.partition.mode=nonstrict;`

- Create a **dummy table** to store the data.

```
codegyani@ubuntu64server: ~/hive
hive> create table stud_demo(id int, name string, age int, institute string, cou
rse string)
      > row format delimited
      > fields terminated by ',';
OK
```

- Now, load the data into **dummy table**.

```
codegyani@ubuntu64server: ~/hive
hive> load data local inpath '/home/codegyani/hive/student_details' into table s
tud_demo;
Loading data to table show.stud_demo
Table show.stud_demo stats: [numFiles=1, totalsize=152]
OK
```

- Create a **partition table** by using the following command

```
codegyani@ubuntu64server: ~/hive
hive> create table student_part (id int, name string, age int, institute string)
      > partitioned by (course string)
      > row format delimited
      > fields terminated by ',';
OK
```

- Now, **insert the data of dummy table into the partition table.**

→ `hive> insert into student_part  
partition(course)  
select id, name, age, institute, course  
from stud_demo;`

- In the following screenshot, we can see that the table **student\_part** is divided into two categories.

The screenshot shows the Hadoop Web UI's 'Browse Directory' interface. The URL in the address bar is /user/hive/warehouse/show.db/student\_part. The table has columns: Permission, Owner, Group, Size, Last Modified, Replication, Block Size, and Name. There are three rows corresponding to the partitions:

| Permission | Owner     | Group      | Size | Last Modified        | Replication | Block Size | Name                              |
|------------|-----------|------------|------|----------------------|-------------|------------|-----------------------------------|
| drwxr-xr-x | codegyani | supergroup | 0 B  | 8/1/2019, 3:53:17 PM | 0           | 0 B        | course=__HIVE_DEFAULT_PARTITION__ |
| drwxr-xr-x | codegyani | supergroup | 0 B  | 8/1/2019, 3:53:15 PM | 0           | 0 B        | course=hadoop                     |
| drwxr-xr-x | codegyani | supergroup | 0 B  | 8/1/2019, 3:53:16    | 0           | 0 B        | course=java                       |

- Let's retrieve the entire data of the table by using the following command

```
hive> select * from student_part;
OK
NULL      NULL      NULL      NULL      __HIVE_DEFAULT_PARTITION__
2          "John"    22        javatpoint      hadoop
3          "William" 21        javatpoint      hadoop
1          "Gaurav"   24        javatpoint      java
4          "Steve"    24        javatpoint      java
5          "Roman"   23        javatpoint      java
Time taken: 1.231 seconds, Fetched: 6 row(s)
hive>
```

- Now, try to retrieve the data based on partitioned columns by using the following command:

```
codegyani@ubuntu64server: ~/hive
hive> select * from student_part where course= "java";
OK
1      "Gaurav"        24    javatpoint      java
4      "Steve"  24      javatpoint      java
5      "Roman"  23      javatpoint      java
Time taken: 0.569 seconds, Fetched: 3 row(s)
hive> 
```

- Let's also retrieve the data of another partitioned dataset by using the following command

```
codegyani@ubuntu64server: ~/hive
hive> select * from student_part where course= "hadoop";
OK
2      "John"   22    javatpoint      hadoop
3      "William" 21    javatpoint      hadoop
Time taken: 0.914 seconds, Fetched: 2 row(s)
hive> 
```

## Bucketing in Hive

- The bucketing in Hive is a data organizing technique. It is similar to partitioning in Hive with an added functionality that it **divides large datasets into more manageable parts** known as buckets.

- First, select the **database** in which we want to create a table.

```
hive> use showbucket;
OK
Time taken: 0.111 seconds
hive>
```

- Create a dummy table to store the data.

```
hive> create table emp_demo (Id int, Name string , Salary float)
      > row format delimited
      > fields terminated by ',' ;
OK
Time taken: 0.373 seconds
hive>
```

- Now, load the data into the table.

```
hive> load data local inpath '/home/codegyani/hive/emp_details' into table emp_demo;
Loading data to table showbucket.emp_demo
Table showbucket.emp_demo stats: [numFiles=1, totalSize=131]
OK
Time taken: 1.333 seconds
hive>
```

# Bucketing in Hive

- Enable the bucketing by using the following command

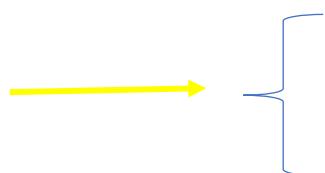
```
hive> set hive.enforce.bucketing = true;
codegyani@ubuntu64server: ~
hive> create table emp_bucket(Id int, Name string , Salary float)
      > clustered by (Id) into 3 buckets
      > row format delimited
      > fields terminated by ',' ;
OK
Time taken: 0.308 seconds
hive>
```

- Create a **bucketing table** by using the following command:

- Now, **insert the data of dummy table** into the bucketed table.

↓  
**hive> insert overwrite table emp\_bucket select \* from emp\_demo;**

- Here, we can see that the data is divided into three buckets.



Hadoop Overview Datanodes Snapshot Startup Progress Utilities ▾

## Browse Directory

/user/hive/warehouse/showbucket.db/emp\_bucket Go!

| Permission | Owner     | Group      | Size | Last Modified        | Replication | Block Size | Name     |
|------------|-----------|------------|------|----------------------|-------------|------------|----------|
| -rwxr-xr-x | codegyani | supergroup | 56 B | 8/2/2019, 4:11:20 AM | 1           | 128 MB     | 000000_0 |
| -rwxr-xr-x | codegyani | supergroup | 53 B | 8/2/2019, 4:11:20 AM | 1           | 128 MB     | 000001_0 |
| -rwxr-xr-x | codegyani | supergroup | 54 B | 8/2/2019, 4:11:20 AM | 1           | 128 MB     | 000002_0 |

- Let's retrieve the data of bucket 0.

According to hash function :

$$6 \% 3 = 0$$

$$3 \% 3 = 0$$

So, these columns stored in bucket 0.

```
codegyani@ubuntu64server: ~
codegyani@ubuntu64server:~$ hdfs dfs -cat /user/hive/warehouse/showbucket.db/emp
    _bucket/000000_0;
\N,\N,\N
\N,\N,\N
6,"William",9000.0
3,"Vishal",40000.0
```

- Let's retrieve the data of bucket 1

According to hash function :

$$7 \% 3 = 1$$

$$4 \% 3 = 1$$

$$1 \% 3 = 1$$

So, these columns stored in bucket 1.

```
codegyani@ubuntu64server: ~
codegyani@ubuntu64server:~$ hdfs dfs -cat /user/hive/warehouse/showbucket.db/emp
    _bucket/000001_0;
7,"Lisa",25000.0
4,"John",10000.0
1,"Gaurav",30000.0
```

- Let's retrieve the data of bucket 2.

According to hash function :

$$8 \% 3 = 2$$

$$5 \% 3 = 2$$

$$2 \% 3 = 2$$

So, these columns stored in bucket 2.

```
codegyani@ubuntu64server: ~
codegyani@ubuntu64server:~$ hdfs dfs -cat /user/hive/warehouse/showbucket.db/emp
    _bucket/000002_0;
8,"Ronit",20000.0
5,"Henry",25000.0
2,"Aryan",20000.0
```

# File Formats & Descriptions:

| File Format     | Description  |
|-----------------|--|
| Text file       | The default file format, and a line represents a record. The delimiting characters separate the lines. Text file examples are CSV, TSV, JSON and XML |
| Sequential file | Flat file which stores binary key-value pairs, and supports compression.   |
| RCFile          | Record Columnar file   |
| ORCFILE         | ORC stands for Optimized Row Columnar which means it can store data in an optimized way than in the other file formats                               |

## Hive Text File Format

- Hive **Text file format is a default storage format.**
- You can use the text format to **interchange** the data with **other client application.**
- Create a **TEXT** file by add **storage option** as '**STORED AS TEXTFILE**' at the end of a Hive **CREATE TABLE** command.

### Hive Text File Format Examples

Below is the Hive **CREATE TABLE** command with **storage format** specification:

```
Create table textfile_table (column_specs) stored as textfile;
```

## Hive Sequence File Format

- Sequence files are Hadoop flat files which stores values in **binary key-value pairs**.
- The sequence files are **in binary format** and these files are **able to split**.
- The main **advantages** of using sequence file is to **merge two or more files** into one file.
- Create a sequence file by add storage option as '**STORED AS SEQUENCEFILE**' at the end of a Hive CREATE TABLE command.

### Hive Sequence File Format Example

Below is the Hive **CREATE TABLE** command **with storage format** specification:

**Create table sequencefile\_table (column\_specs) stored as sequencefile;**

## Hive RC File Format

- RCFfile is **row columnar file format**.
- This is another form of Hive file format which **offers high row level compression rates**.
- If you have requirement to **perform multiple rows at a time** then you can use RCFfile format.
- The RCFfile are very much similar to the sequence file format. This file format also stores the **data as key-value pairs**.
- Create RCFfile by specifying '**STORED AS RCFFILE**' option at the end of a CREATE TABLE Command:

### Hive RC File Format Example

Below is the Hive CREATE TABLE command with storage format specification:

**Create table RCFfile\_table (column\_specs) stored as rcfle;**

## Hive RC File Format

RC files partitions the table first horizontally, and then vertically to serialize the data

**Table 9.1** A table with four columns

| C1 | C2 | C3 | C4 |
|----|----|----|----|
| 11 | 12 | 13 | 14 |
| 21 | 22 | 23 | 24 |
| 31 | 32 | 33 | 34 |
| 41 | 42 | 43 | 44 |
| 51 | 52 | 53 | 54 |

**Table 9.2** Table with two row groups

| Row Group 1 |    |    |    | Row Group 2 |    |    |    |
|-------------|----|----|----|-------------|----|----|----|
| C1          | C2 | C3 | C4 | C1          | C2 | C3 | C4 |
| 11          | 12 | 13 | 14 | 41          | 42 | 43 | 44 |
| 21          | 22 | 23 | 24 | 51          | 52 | 53 | 54 |
| 31          | 32 | 33 | 34 |             |    |    |    |

**Table 9.3** Table in RCFile Format

| Row Group 1 | Row Group 2 |
|-------------|-------------|
| 11, 21, 31; | 41, 51;     |
| 12, 22, 32; | 42, 52;     |
| 13, 23, 33; | 43, 53;     |
| 14, 24, 34; | 44, 54;     |

## Hive ORC File Format

- The ORC file stands for **Optimized Row Columnar** file format.
- The ORC file format provides a **highly efficient way to store data in Hive table**.
- This file system was actually designed to **overcome limitations of the other Hive file formats**.
- The Use of **ORC files improves performance** when **Hive is reading, writing, and processing data from large tables**.
- Due to their columnar storage format, they can **efficiently skip over irrelevant columns** when executing queries, leading to faster query performance.

Create ORC file by specifying '**STORED AS ORC**' option at the end of a CREATE TABLE Command.

### Hive ORC File Format Examples

Below is the Hive CREATE TABLE command with storage format specification:

**Create table orc\_table (column\_specs) stored as orc;**

## Hive Parquet File Format

- Parquet is a **column-oriented binary file format**.
- The parquet is **highly efficient for the types of large-scale queries**.
- Parquet is especially good for **queries scanning particular columns** within a particular table.
- The Parquet table uses compression **Snappy, gzip**; currently **Snappy by default**.

Create Parquet file by specifying ‘STORED AS PARQUET’ option at the end of a CREATE TABLE Command.

### Hive Parquet File Format Example

Below is the Hive CREATE TABLE command with storage format specification:

**Create table parquet\_table (column\_specs) stored as parquet;**

## Hive Parquet File Sample

File: C:\Users\ \Downloads\yellow\_tripdata\_2022-01.parquet

File Edit Tools Help

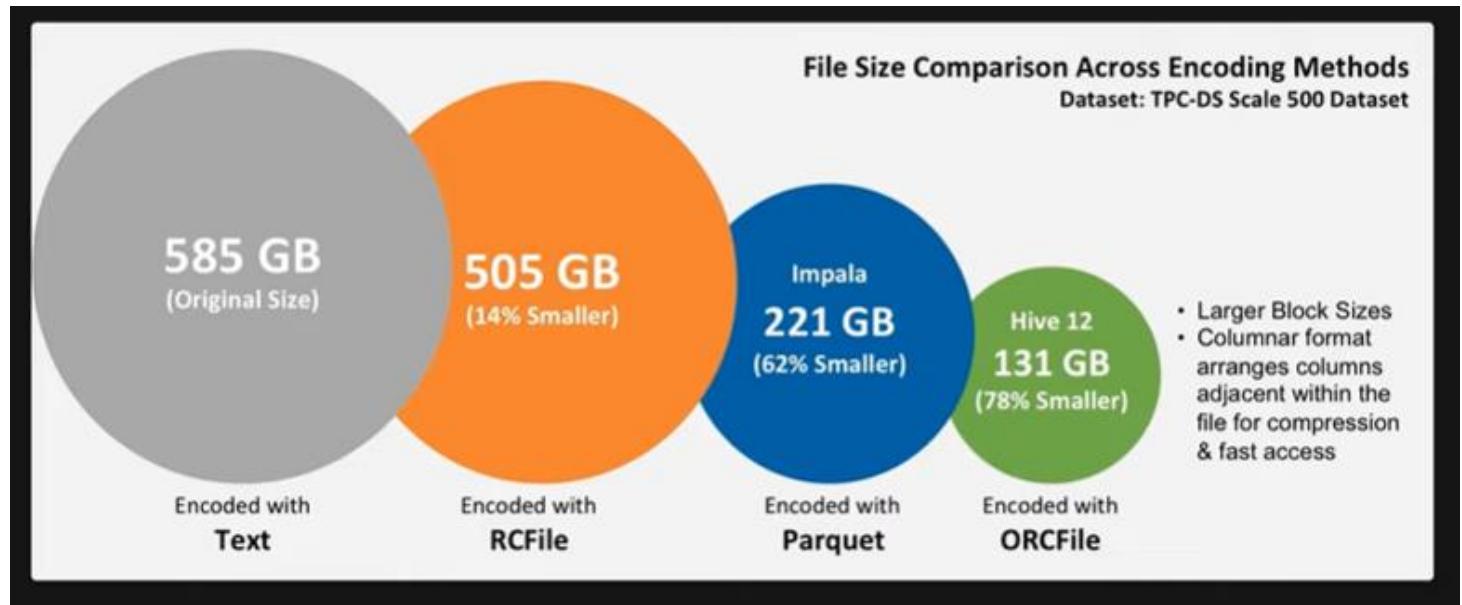
Filter Query (?): WHERE (tip\_amount \* 100) / fare\_amount > 60

Execute Clear Record Offset: 0 Record Count: 1000 ▶

|   | VendorID | fare_amount | tip_amount | tpep_pickup_datetime    | tpep_dropoff_datetime   | passenger_count | trip_distance | RatecodeID | store_and_f |
|---|----------|-------------|------------|-------------------------|-------------------------|-----------------|---------------|------------|-------------|
| ▶ | 2        | 21          | 15         | 2022-01-01 00:55:48.000 | 2022-01-01 01:14:24.000 | 1               | 6.67          | 1          | N           |
|   | 1        | 7.5         | 20         | 2022-01-01 00:37:13.000 | 2022-01-01 00:44:46.000 | 1               | 1.9           | 1          | N           |
|   | 1        | 5.5         | 4          | 2022-01-01 00:57:44.000 | 2022-01-01 01:02:34.000 | 0               | 0.9           | 1          | N           |
|   | 2        | 4           | 10         | 2022-01-01 00:15:46.000 | 2022-01-01 00:17:41.000 | 2               | 0.6           | 1          | N           |
|   | 2        | 5.5         | 15         | 2022-01-01 00:23:02.000 | 2022-01-01 00:28:20.000 | 1               | 0.75          | 1          | N           |
|   | 1        | 3.5         | 3          | 2022-01-01 00:48:11.000 | 2022-01-01 00:50:08.000 | 1               | 0.4           | 1          | N           |
|   | 2        | 5.5         | 5          | 2022-01-01 00:28:57.000 | 2022-01-01 00:34:16.000 | 1               | 0.67          | 1          | N           |
|   | 2        | 6           | 4          | 2022-01-01 00:15:47.000 | 2022-01-01 00:21:18.000 | 1               | 1.27          | 1          | N           |
|   | 1        | 6           | 20         | 2022-01-01 00:10:22.000 | 2022-01-01 00:15:53.000 | 2               | 1.2           | 1          | N           |
|   | 2        | 3           | 2.04       | 2022-01-01 00:25:52.000 | 2022-01-01 00:27:12.000 | 1               | 0.26          | 1          | N           |

# Memory usage of different file types

- ORC using “Columnar Format of Storage”.
- Columnar oriented storage is always **faster** than row-oriented storage.



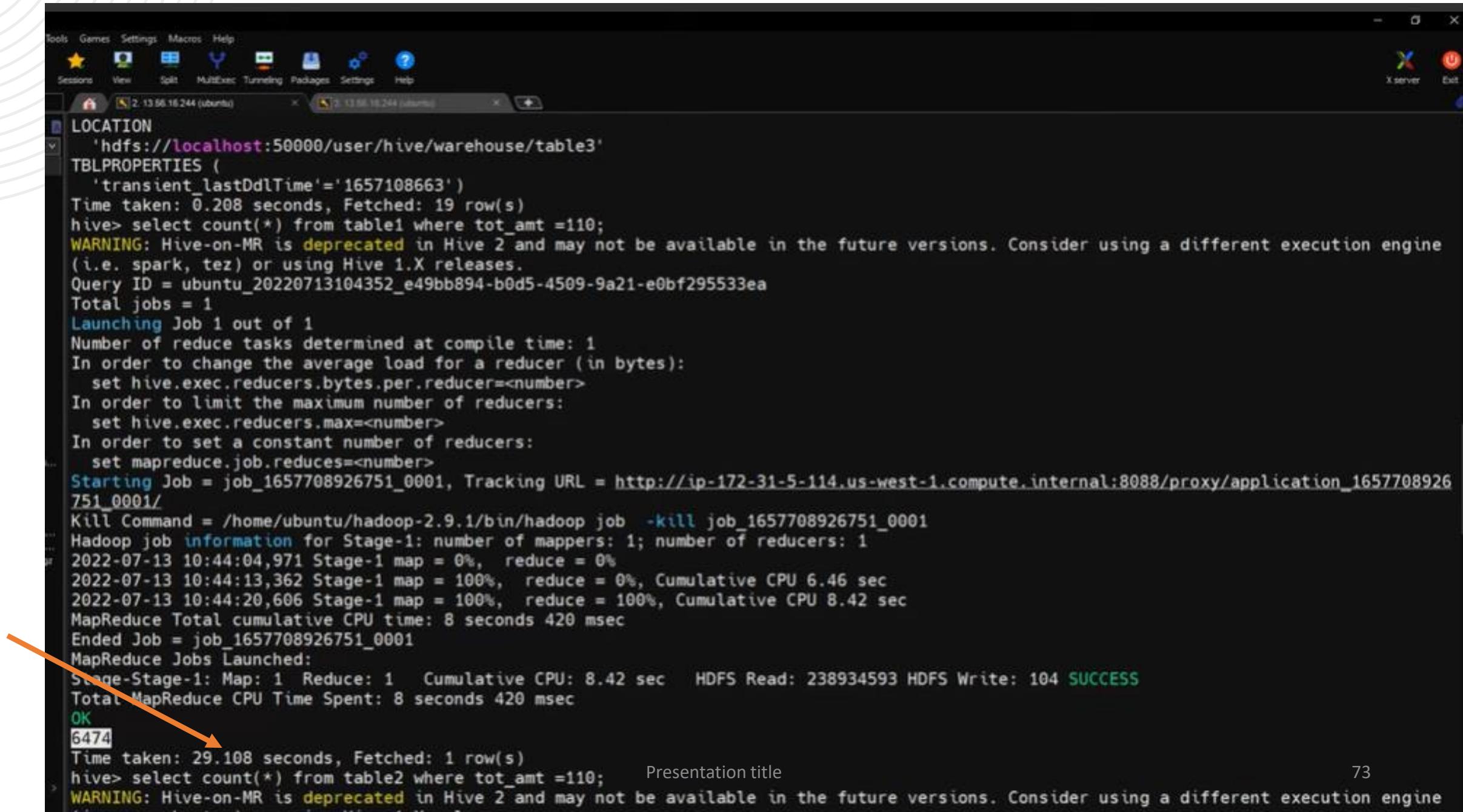
## Example (File Format): Time Difference & volume of Data

```
w20 - Notepad++  
File Search View Encoding Language Settings Tools Macro Run Plugins Window ?  
New 20  
1 create table table1(pid INT, pname STRING, drug STRING, gender STRING, tot_amt INT) row format delimited fields  
terminated by ',' stored as textfile;  
2  
3 load data local inpath '/home/ubuntu/data_301.txt' into table table1;  
4 load data local inpath '/home/ubuntu/data_301.txt' into table table1;  
5  
6  
7 show create table table1;  
8  
9  
=====  
0  
1  
2 create table table2(pid INT, pname STRING, drug STRING, gender STRING, tot_amt INT) row format delimited fields  
terminated by ',' STORED AS orc;  
3  
4 insert overwrite table table2 select * from table1;  
5  
6 show create table table2;  
7  
8 select count(*) from table1 where tot_amt =110;  
9 select count(*) from table2 where tot_amt =110;  
0
```

# Table 1: Text Format (60lakh count )

```
essions View Split MultiExec Tuning Packages Settings Help
 2: 13.66.16.244 (ubuntu) 3: 13.66.16.244 (ubuntu)
'hdfs://localhost:50000/user/hive/warehouse/table2'
TBLPROPERTIES (
  'transient_lastDdlTime'='1657108076')
Time taken: 0.049 seconds, Fetched: 19 row(s)
hive> select count(*) from table2;
OK
6000000
Time taken: 0.4 seconds, Fetched: 1 row(s)
hive> select count(*) from table1;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine
(i.e. spark, tez) or using Hive 1.X releases.
Query ID = ubuntu_20220713105649_992f7438-37f2-47cf-87d0-6803e29effd
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1657708926751_0003, Tracking URL = http://ip-172-31-5-114.us-west-1.compute.internal:8088/proxy/application_1657708926
751_0003/
Kill Command = /home/ubuntu/hadoop-2.9.1/bin/hadoop job -kill job_1657708926751_0003
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2022-07-13 10:56:56,977 Stage-1 map = 0%, reduce = 0%
2022-07-13 10:57:04,183 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 4.85 sec
2022-07-13 10:57:10,322 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 6.18 sec
MapReduce Total cumulative CPU time: 6 seconds 180 msec
Ended Job = job_1657708926751_0003
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 6.18 sec HDFS Read: 238933789 HDFS Write: 107 SUCCESS
Total MapReduce CPU Time Spent: 6 seconds 180 msec
OK
6000000
Time taken: 22.29 seconds, Fetched: 1 row(s)
```

# Text File Format:



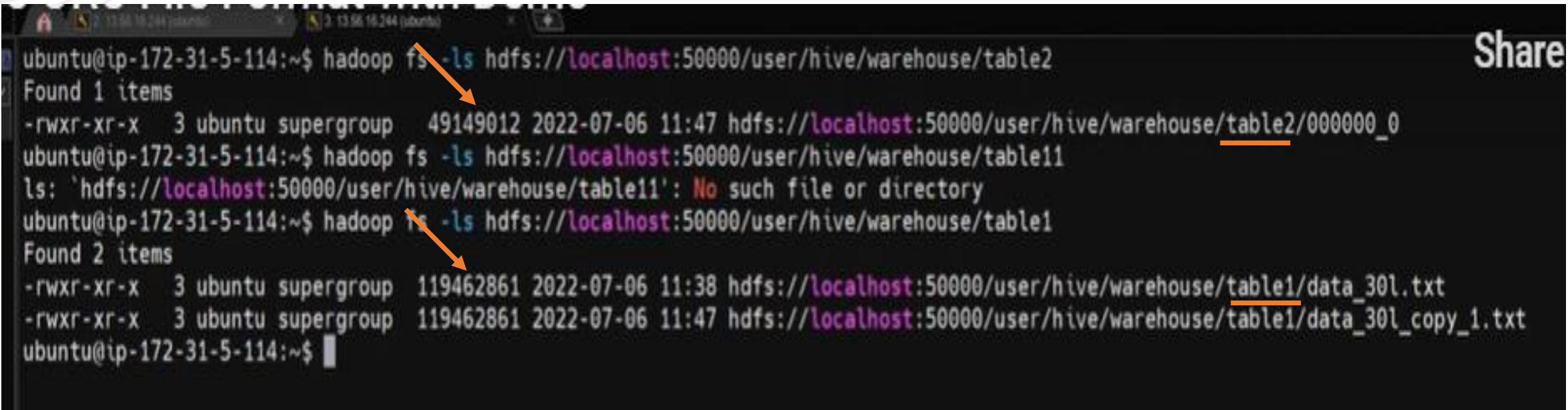
```
Tools Games Settings Macros Help
Sessions View Split MultiExec Tunneling Packages Settings Help
X server Exit

LOCATION
'hdfs://localhost:50000/user/hive/warehouse/table3'
TBLPROPERTIES (
  'transient_lastDdlTime'='1657108663')
Time taken: 0.208 seconds, Fetched: 19 row(s)
hive> select count(*) from table1 where tot_amt =110;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine (i.e. spark, tez) or using Hive 1.X releases.
Query ID = ubuntu_20220713104352_e49bb894-b0d5-4509-9a21-e0bf295533ea
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1657708926751_0001, Tracking URL = http://ip-172-31-5-114.us-west-1.compute.internal:8088/proxy/application_1657708926751_0001/
Kill Command = /home/ubuntu/hadoop-2.9.1/bin/hadoop job -kill job_1657708926751_0001
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2022-07-13 10:44:04,971 Stage-1 map = 0%, reduce = 0%
2022-07-13 10:44:13,362 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 6.46 sec
2022-07-13 10:44:20,606 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 8.42 sec
MapReduce Total cumulative CPU time: 8 seconds 420 msec
Ended Job = job_1657708926751_0001
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 8.42 sec HDFS Read: 238934593 HDFS Write: 104 SUCCESS
Total MapReduce CPU Time Spent: 8 seconds 420 msec
OK
6474
Time taken: 29.108 seconds, Fetched: 1 row(s)
hive> select count(*) from table2 where tot_amt =110;      Presentation title
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine
```

# ORC : Time Difference

```
OK
6474
Time taken: 29.108 seconds, Fetched: 1 row(s)
hive> select count(*) from table2 where tot_amt =110;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine
(i.e. spark, tez) or using Hive 1.X releases.
Query ID = ubuntu_20220713104557_950d5971-092e-4e66-b4c3-59e5ceb973dc
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1657708926751_0002, Tracking URL = http://ip-172-31-5-114.us-west-1.compute.internal:8088/proxy/application_1657708926
751_0002/
Kill Command = /home/ubuntu/hadoop-2.9.1/bin/hadoop job -kill job_1657708926751_0002
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2022-07-13 10:46:03,565 Stage-1 map = 0%, reduce = 0%
2022-07-13 10:46:11,832 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 3.69 sec
2022-07-13 10:46:18,041 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 5.38 sec
MapReduce Total cumulative CPU time: 5 seconds 380 msec
Ended Job = job_1657708926751_0002
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 5.38 sec HDFS Read: 9114117 HDFS Write: 104 SUCCESS
Total MapReduce CPU Time Spent: 5 seconds 380 msec
OK
6474
Time taken: 21.902 seconds, Fetched: 1 row(s)
hive>
hive> show create table table1;
OK
CREATE TABLE `table1`(
  `pid` int,
```

# Volume Compression of Text & ORC File Formats:



The screenshot shows a terminal session on an Ubuntu system (ip-172-31-5-114) using the hadoop fs command to list files in HDFS. The session is split into three parts by command-line separators:

- Table 2:** The user runs `hadoop fs -ls hdfs://localhost:50000/user/hive/warehouse/table2`. The output shows one item: a file named `000000_0` with a size of 49149012 bytes.
- Table 11:** The user runs `hadoop fs -ls hdfs://localhost:50000/user/hive/warehouse/table11`. The output shows an error message: "ls: 'hdfs://localhost:50000/user/hive/warehouse/table11': No such file or directory".
- Table 1:** The user runs `hadoop fs -ls hdfs://localhost:50000/user/hive/warehouse/table1`. The output shows two items: two files named `data_301.txt` and `data_301_copy_1.txt`, both with a size of 119462861 bytes.

Two orange arrows point from the text "Table 2" and "Table 1" in the question below to the file sizes in the terminal output. The word "Share" is visible in the top right corner of the terminal window.

- $119462861 * 2 = 238,925,722$  (Text format) [Table 1]
- 49149012 (ORC Format) [Table 2]

# Apache Hive Command Line Options

| Hive Command Line Option | Description  |
|--------------------------|--|
| -d,-define <key=value>   | Variable substitution to apply to Hive commands.<br><br>e.g. -d A=B or –define A=B |
| -e <quoted-query-string> | Execute SQL from command line.<br><br>e.g. -e 'select * from table1'               |
| -f <filename>            | Execute SQL from file. e.g. -f '/home/sql_file.sql'                                |
| -H,—help                 | Print Hive help information. Usually, print all command line options.              |

- SQL queries written in a file should be saved with **.hql extension**
- **Silent Mode:** without showing any progress or output information apart from the result set itself.

## Hive Command Line Options Usage Examples

### Execute query using hive command line options

```
$ hive -e 'select * from test';
```

### Execute query using hive command line options in silent mode

```
$ hive -S -e 'select * from test'
```

### Dump data to the file in silent mode

```
$hive -S -e 'select col from tab1' > a.txt
```

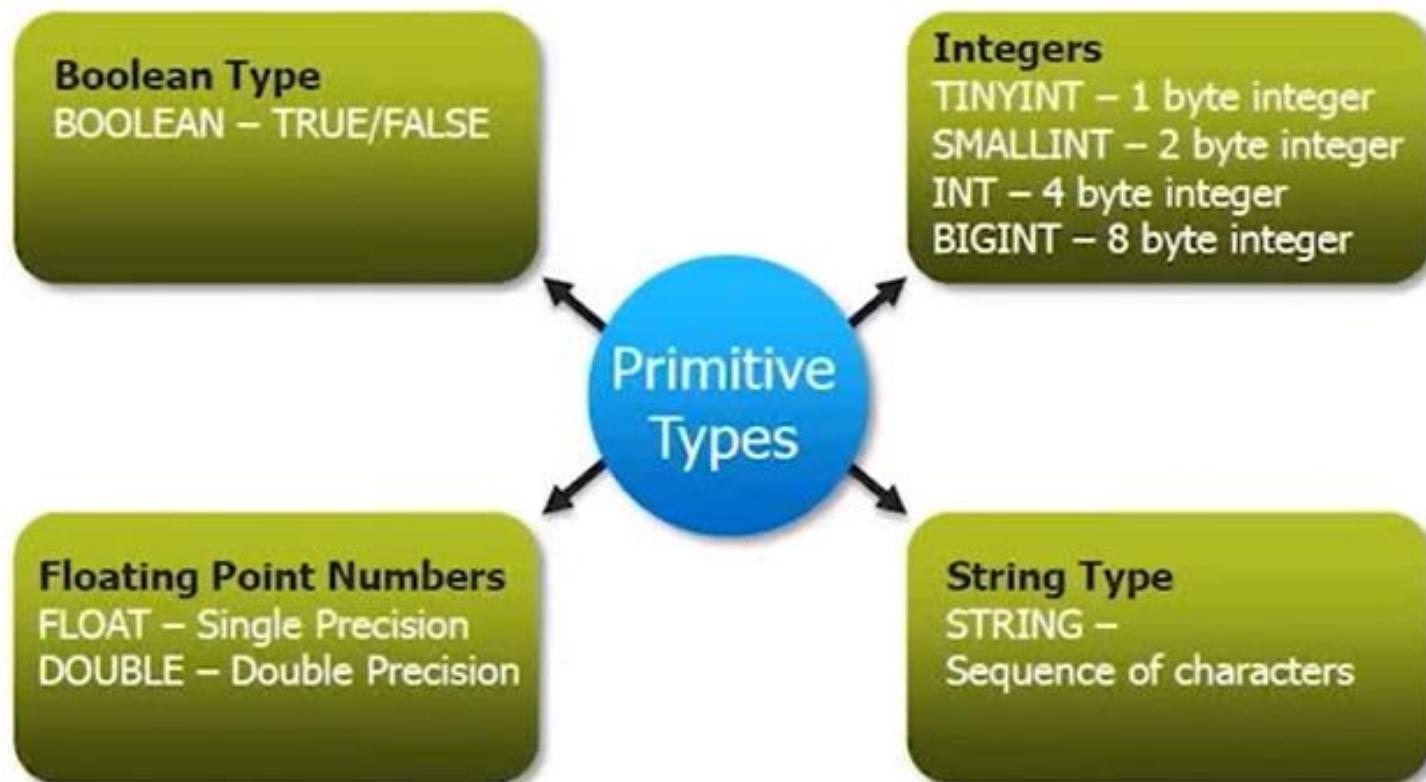
# HIVE V/S RDBMS

| SL.NO | Characteristics      | HIVE               | RDBMS                   |
|-------|----------------------|--------------------|-------------------------|
| 1     | Record Level Queries | No Update & Delete | Insert, Update & Delete |
| 2     | Transaction Support  | No                 | Yes                     |
| 3     | Latency              | Minutes or more    | In fraction of seconds  |
| 4     | Data Size            | Petabytes          | Terabytes               |
| 5     | Data Per Query       | Petabytes          | Gigabytes               |
| 6     | Query Language       | HiveQL             | SQL                     |
| 7     | Support JDBC/ODBC    | Limited            | Full                    |

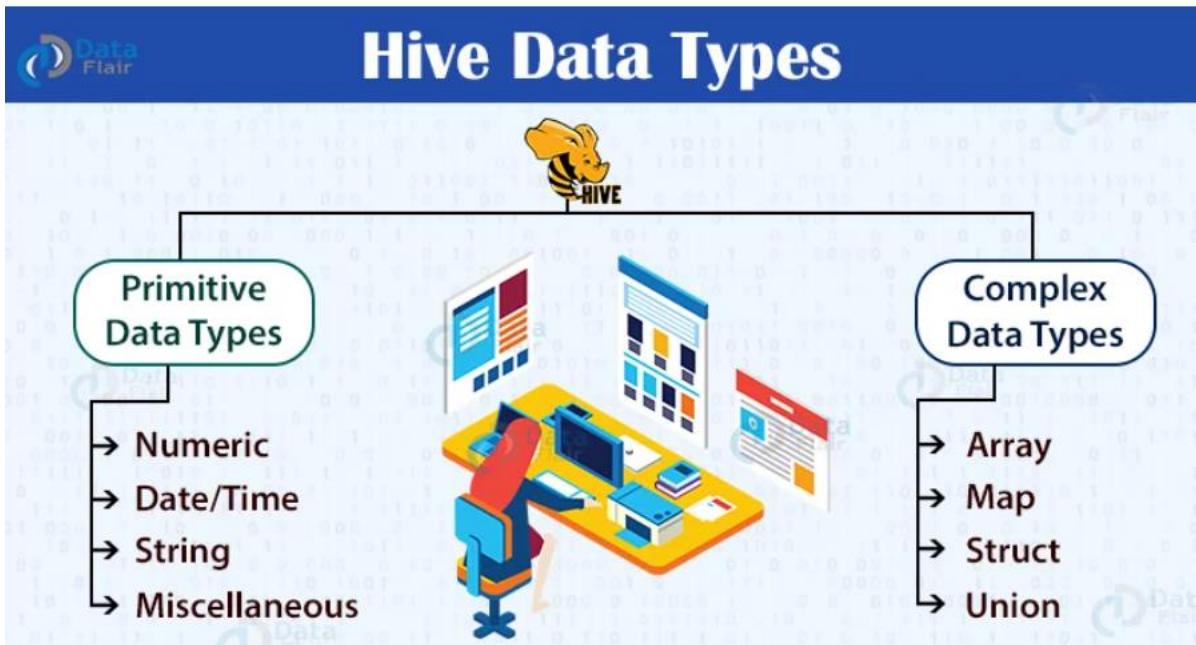
# RDBMS V/S HIVE

- ✓ **Schema on Read vs Schema on Write**
  - ✓ **Hive does not verifies the data when it is loaded**, but rather when a query is issued.
  - ✓ Schema on read makes for a **very fast initial load**, since the data does not have to be read, parsed and serialized to disk in the database's internal format. The load operation is just a file copy or move.
- ✓ **No Updates, Transactions and Indexes.**

# Data Types - Hive



# Collection Data Types:



| Name   | Description   |
|--------|---|
| STRUCT | Similar to 'C' struc, a collection of fields of different data types. An access to field uses dot notation.<br>For example, struct ('a', 'b') |
| MAP    | A collection of key-value pairs. Fields access using [] notation.<br>For example, map ('key1', 'a', 'key2', 'b')                              |
| ARRAY  | Ordered sequence of same types. Accesses to fields using array index.<br>For example, array ('a', 'b')  |

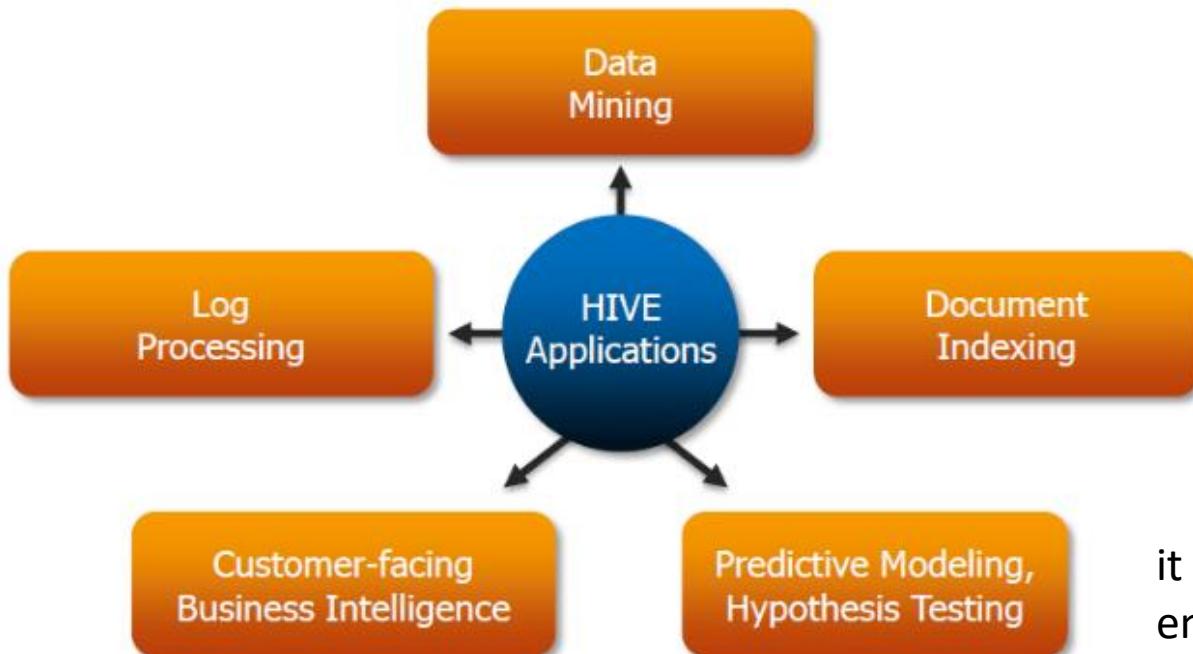
- **Map in Hive** is a collection of key-value pairs, where the fields are accessed using array notations of keys (e.g., ['key']).
- **map<primitive\_type, data\_type>**
- **Example:** ‘first’ -> ‘John’, ‘last’ -> ‘Deo’, represented as map(‘first’, ‘John’, ‘last’, ‘Deo’). Now ‘John’ can be accessed with map[‘first’].
- **STRUCT in Hive** is similar to the STRUCT in C language. It is a record type that encapsulates a set of named fields, which can be any primitive data type.
- We can access the elements in STRUCT type using DOT (.) notation.
- **STRUCT <col\_name : data\_type [ COMMENT col\_comment], ...>**
- Example: For a column c3 of type STRUCT {c1 INTEGER; c2 INTEGER}, the c1 field is accessed by the expression c3.c1.

|        |   |   |
|--------|---|---|
| BIGINT | round(double a)                           | Returns the rounded BIGINT (8 Byte integer) value of the 8 Byte double-precision floating point number a  |
| BIGINT | floor(double a)                           | Returns the maximum BIGINT value that is equal to or less than the double.  |
| BIGINT | ceil(double a)                            | Returns the minimum BIGINT value that is equal to or greater than the double.   |
| double | rand(), rand(int seed)                    | Returns a random number (double) that distributes uniformly from 0 to 1 and that changes in each row. Integer seed ensured that random number sequence is deterministic.                                      |
| string | concat(string str1, string str2, ...)     | Returns the string resulting from concatenating str1 with str2, ....  |
| string | substr(string str, int start)             | Returns the substring of str starting from a start position till the end of string str.   |
| string | substr(string str, int start, int length) | Returns the substring of str starting from the start position with the given length.  |
| string | upper(string str), ucase (string str)     | Returns the string resulting from converting all characters of str to upper case.   |
| string | lower(string str), lcase(string str)      | Returns the string resulting from converting all characters of str to lower case.   |
| string | trim(string str)                          | Returns the string resulting from trimming spaces from both ends. trim ('12A34 56') returns '12A3456'   |
| string | ltrim(string str);<br>rtrim(string str)   | Returns the string resulting from trimming spaces (only one end, left or right hand side or right-handside spaces trimmed).<br>ltrim('12A34 56') returns '12A3456' and rtrim(' 12A34 56 ') returns '12A3456'. |

# Built In Functions:

## Return Type, Syntax & Description

# Apache Hive – Applications



- Data Mining (If u want to identify Buying behaviour)
- Predictive Modeling (To enhance performance)
- Document Indexing
- Custom Facing UI

it can play a crucial role in the data preparation and feature engineering stages of the predictive modeling pipeline

# Apache Hive Use Case – Facebook:



# Facebook – Use Case

- Objective – To store & process large amount of data.
  - In 2007 , FB used “Hadoop” – To store and process the generated data. ETL (Via Python)
  - But In Hadoop – Programming became difficult, as Hadoop default based is map reduce (java based), to execute small query larger codes need to be written (Structured Data type).
  - So Hive was developed by FB.
- 
- FB Stats: RDBMS was not suitable
  - Hadoop usage Started
  - 500 B/Day
  - 70k Queries /Day
  - 300 Million Photos /Day

# Hive QL:

- HiveQL: **Querying the large dataset** which reside in the **HDFS** environment.
- HiveQL script commands enable **data definition, data manipulation and query processing**.
- Supports a large base of SQL users who are acquainted with SQL to “**Extract information from data warehouse**”

# HiveQL : Data Definition Language

HiveQL database commands for data definition for DBs and Tables are CREATE DATABASE, SHOW DATABASE (list of all DBs), CREATE SCHEMA, CREATE TABLE. Following are HiveQL commands which create a table:

```
CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS] [<database name>.]  
<table name>  
[(<column name> <data type> [COMMENT <column comment>], ...)]  
[COMMENT <table comment>]  
[ROW FORMAT <row format>]  
[STORED AS <file format>]
```

|           |   |
|-----------|---|
| DELIMITED | Specifies a delimiter at the table level for structured fields. This is default.<br>Syntax: FIELDS TERMINATED BY, LINES TERMINATED BY |
| SERDE     | Stands for Serializer/Deserializer. SYNTAX: SERDE 'serde.class.name'  |

# Commands: DDL

- Commands : CREATE DATABASE, SHOW DATABASE, CREATE SCHEMA, CREATE TABLE

How do you create a database named toys\_companyDB and table named toys\_tbl?

## SOLUTION

```
$HIVE_HOME/binhive - service cli  
hive>set hive.cli.print.current.db=true;  
hive> CREATE DATABASE toys_companyDB  
hive>USE toys_companyDB  
hive (toys_companyDB)> CREATE TABLE toys_tbl (  
>puzzle_code STRING,  
>pieces SMALLINT  
>cost FLOAT);  
hive (toys_company)> quit;  
&ls/home/binadmin/Hive/warehouse/toys_companyDB.db
```

# Partition: Reducing Query Processing Time

Assume that following file contains toys\_tbl.

```
/table/toy_tbl/file1  
Category, id, name, price  
Toy_Airplane, 10725, Lost Temple, 1.25  
Toy_Airplane, 31047, Propeller Plane, 2.10  
Toy_Airplane, 31049, Twin Spin Helicopter, 3.45
```

```
Toy_Train, 31054, Blue Express, 4.25  
Toy_Train, 10254, Winter Holiday Toy_Train, 2.75
```

A table *toy\_tbl* contains many values for categories of toys. Query is required to identify all *toy\_airplane* fields. Give reasons why partitioning reduces query processing time.

## SOLUTION

Here, a table named *toy\_tbl* contains several toy details (category, id, name and price). Suppose it is required to identify all the airplanes. A query searches the whole table for the required information. However, if a partition is created on the *toy\_tbl*, based on category and stores it in a separate file, then it will reduce the query processing time.

Let the data partitions into two files, file 2 and file 3, using category.

```
/table/toys/toy_airplane/file2  
toy_airplane, 10725, Lost Temple, TP, 1.25  
toy_airplane, 31047, Propeller Plane, 2.10  
toy_airplane, 31049, Lost Temple, 3.45  
/table/toys/toy_train/file3  
Toy_Train, 31054, Blue Express, 4.25  
Toy_Train, 10254, Winter Holiday Toy_Train, 2.75
```

# Partition : Advantages & Limitations

## Advantages of Partition

1. Distributes execution load horizontally.
2. Query response time becomes faster when processing a small part of the data instead of searching the entire dataset.

## Limitations of Partition

1. Creating a large number of partitions in a table leads to a large number of files and directories in HDFS, which is an overhead to NameNode, since it must keep all metadata for the file system in memory only.
2. Partitions may optimize some queries based on Where clauses, but they may be less responsive for other important queries on grouping clauses.
3. A large number of partitions will lead to a large number of tasks (which will run in separate JVM) in each MapReduce job, thus creating a lot of overhead in maintaining JVM start up and tear down (A separate task will be used for each file). The overhead of JVM start up and tear down can exceed the actual processing time in the worst case.

# Bucketing:

A table `toy_tbl` contains many values for categories of toys. Assume the number of buckets to be created = 5. Assume a table for `Toy_Airplane` of product code 10725.

1. How will the bucketing enforce?
2. How will the bucketed table partition `toy_airplane_10725` create five buckets?
3. How will the bucket column load into `toy_tbl`?
4. How will the bucket data display?

## SOLUTION

#Enforce bucketing

```
set hive.enforce.bucketing=true;  
#Create bucketed Table for toy_airplane of product code 10725 and create cluster of 5 buckets
```

```
CREATE TABLE IF NOT EXISTS  
toy_airplane_10725(ProductCategory STRING,  
ProductId INT, ProductName STRING, PrdocutMfgDate  
YYYY-MM-DD, ProductPrice_US$ FLOAT) CLUSTERED BY  
(Price) into 5 buckets;
```

# Load data to bucketed table.

```
FROM toy_airplane_10725 INSERT OVERWRITE TABLE  
toy_tbl SELECT ProductCategory, ProductId,  
ProductName, PrdocutMfgDate, ProductPrice;
```

- To display the contents for Price\_US\$ selected for the ProductId from the second bucket.

Pres      SELECT DISTINCT ProductId FROM toy\_tbl\_buckets  
TABLE FOR 10725(BUCKET 2 OUT OF 5 ON Price\_US\$);

## Arithmetic Operators in Hive

| Operators | Description  |
|-----------|--|
| A + B     | This is used to add A and B.   |
| A - B     | This is used to subtract B from A.                                       |
| A * B     | This is used to multiply A and B.  |
| A / B     | This is used to divide A and B and returns the quotient of the operands. |
| A % B     | This returns the remainder of A / B.                                     |
| A   B     | This is used to determine the bitwise OR of A and B.                     |
| A & B     | This is used to determine the bitwise AND of A and B.                    |
| A ^ B     | This is used to determine the bitwise XOR of A and B.                    |
| ~A        | This is used to determine the bitwise NOT of A.                          |

```
hive> select id, name, salary + 50 from employee;
```

```
hive> select id, name, salary - 50 from employee;
```

```
hive> select id, name, (salary * 10) /100 from employee;
```

## Relational Operators in Hive

| Operator      | Description  |
|---------------|--|
| A=B           | It returns true if A equals B, otherwise false.  |
| A <> B, A !=B | It returns null if A or B is null; true if A is not equal to B, otherwise false.             |
| A<B           | It returns null if A or B is null; true if A is less than B, otherwise false.                |
| A>B           | It returns null if A or B is null; true if A is greater than B, otherwise false.             |
| A<=B          | It returns null if A or B is null; true if A is less than or equal to B, otherwise false.    |
| A>=B          | It returns null if A or B is null; true if A is greater than or equal to B, otherwise false. |
| A IS NULL     | It returns true if A evaluates to null, otherwise false.                                     |
| A IS NOT NULL | It returns false if A evaluates to null, otherwise true.                                     |

```
hive> select * from employee where salary >= 25000;
```

```
hive> select * from employee where salary < 25000;
```

# Aggregate Functions in Hive

In Hive, the aggregate function returns a single value resulting from computation over many rows. Let's see some commonly used aggregate functions: -

| Return Type | Operator          | Description   |
|-------------|-------------------|---|
| BIGINT      | count(*)          | It returns the count of the number of rows present in the file. |
| DOUBLE      | sum(col)          | It returns the sum of values.                                   |
| DOUBLE      | sum(DISTINCT col) | It returns the sum of distinct values.                          |
| DOUBLE      | avg(col)          | It returns the average of values.                               |
| DOUBLE      | avg(DISTINCT col) | It returns the average of distinct values.                      |
| DOUBLE      | min(col)          | It compares the values and returns the minimum one form it.     |
| DOUBLE      | max(col)          | It compares the values and returns the maximum one form it.     |

# HIVE V/S HBASE

| Hive  | HBase   |
|---|---|
| Hive is a query engine.                                       | Hbase is data storage mainly for unstructured data.             |
| Hive allows most of the SQL queries.                          | HBase does not allow SQL queries.                               |
| Hive is mainly used for batch processing.                     | Hbase is mainly used for transactional processing.              |
| Hive is not real-time processing.                             | HBase is real-time processing.                                  |
| Hive is only used for analytical queries.                     | HBase is used for real-time querying.                           |
| Hive runs on the top of MapReduce.                            | HBase runs on the top of HDFS (Hadoop distributed file system). |
| Hive is not a full database. It is a data warehouse framework | HBase supports the NoSQL database.                              |
| Hive provides SQL features to Spark/Hadoop data.              | HBase is used to store and process Hadoop data in real-time.    |
| Hive has a schema model.                                      | HBase is free from the schema model.                            |
| Hive is made for high latency operations.                     | HBase is made for low-level latency operations.                 |
| Hive is not suited for real-time querying.                    | HBase is used for real-time querying of Big Data.               |