Introduction to Apache Hive

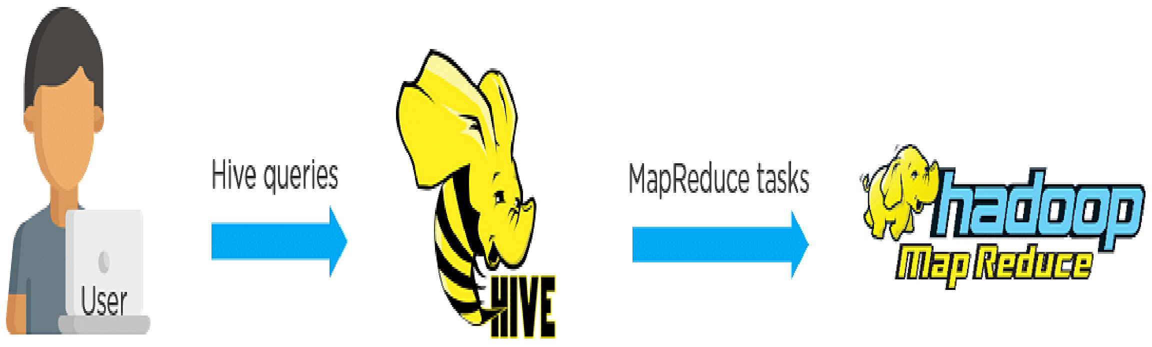
History of Hive

- Hive has a fascinating history related to the world's largest social networking site: Facebook. Facebook adopted the Hadoop framework to manage their big data.
- Big data is nothing but massive amounts of data that cannot be stored, processed, and analyzed by traditional systems.
- As we know, Hadoop uses MapReduce to process data. With MapReduce, users were required to write long and extensive Java code. Not all users were well-versed with Java and other coding languages. Users were comfortable with writing queries in SQL (Structured Query Language), and they wanted a language similar to SQL. Enter the HiveQL language. The idea was to incorporate the concepts of tables and columns, just like SQL.

com/akp03112000

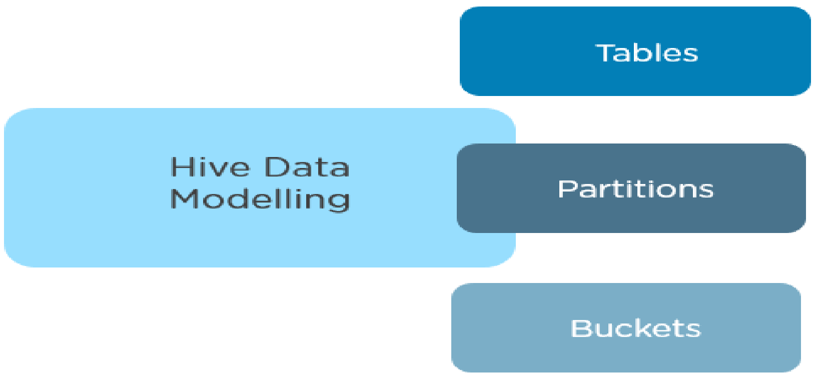
What is Hive ?

- Hive is an data warehouse infrastructure that is built on the top of Hadoop.
- Hive is a data warehouse system that is used to query and analyze large datasets stored in the HDFS. Hive uses a query language called HiveQL, which is similar to SQL.



Hive Data Modeling

- That was how data flows in the Hive. Let's now take a look at Hive data modeling, which consists of tables, partitions, and buckets:
  - Tables - Tables in Hive are created the same way it is done in RDBMS
  - Partitions - Here, tables are organized into partitions for grouping similar types of data based on the partition key
  - Buckets - Data present in partitions can be further divided into buckets for efficient querying



Data Types in Hive

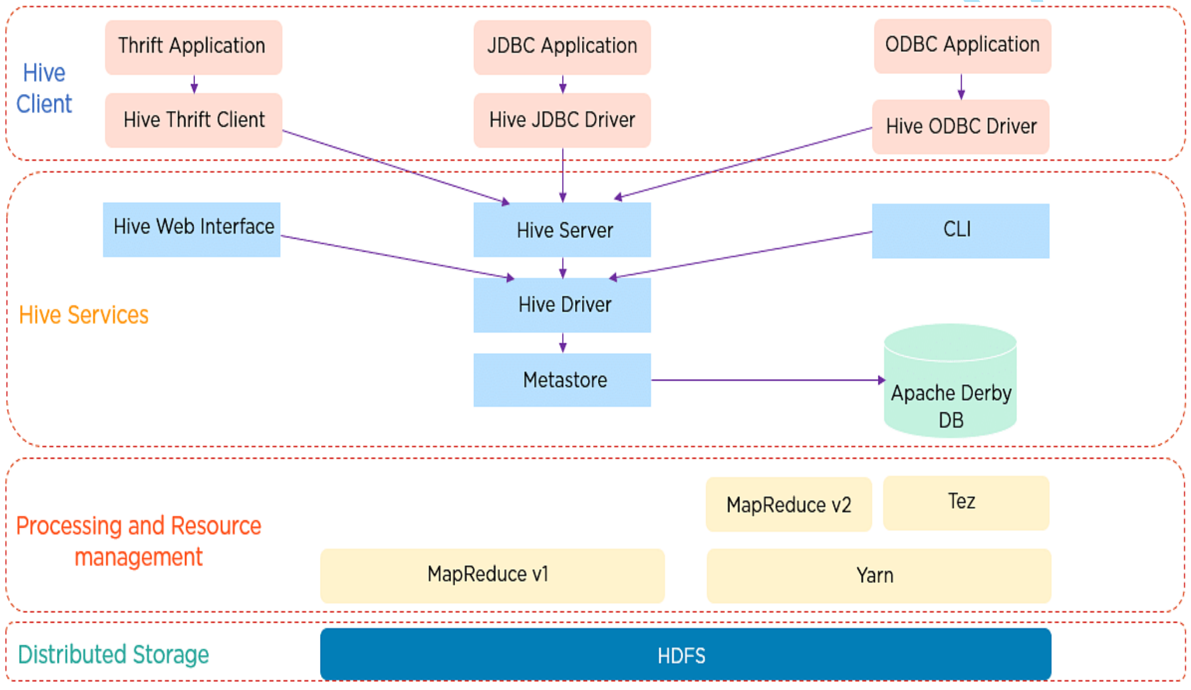
Primitive Data Types	Complex Data Types
• Numeric Data types - integral, float, decimal • String Data type - char, string • Date/ Time Data type - timestamp, date, interval • Miscellaneous Data type - Boolean and binary	• Arrays - A collection of the same entities. The syntax is: array<data_type> • Maps - A collection of key-value pairs and the syntax is map<primitive_type, data_type> • Structs - A collection of complex data with comments. Syntax: struct<col_name : data_type [COMMENT col_comment],…..>

github.com

Where Not to Use Hive ?

- If data doesn’t cross GB's.
- If we need response in seconds and w latency application.
- If RDBMS can solve don’t invest time in Hive.

Hive Architecture



github.

- 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 - MetaStore is a repository for Hive metadata. It stores metadata for Hive tables, and we can think of this as our schema. This is located on the Apache Derby DB.
  - Hive Driver - It receives queries from different sources like web UI, CLI, Thrift, and JDBC/ODBC driver. It transfers the queries to the compiler.

github.com/akp0311200

Hive Commands

Create database	create database demo; or create a database if not exists demo;
Show database	show databases;
Describe database	describe database demo;
Drop database	drop database demo; drop database if exists demo;
Use Database	use demo;

Create table	create table employee (Id int, Name string , Salary float) fields terminated by ',' ; or create table if not exists employee (Id int, Name string , Salary float) row format delimited fields terminated by ',' ;

Describe table	describe employee;
Load data	load data local inpath '/home/users/hive/emp_details' into table employee; select * from employee; Note: 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.
Drop table	drop table employee;

Alter Table	
Renaming a table	Alter table emp RENAME to employee_data;
Adding column	Alter table employee_data ADD columns (age int);

Change column	Alter table table_name change old_column_name new_column_name datatype;
	
	Alter table employee_data CHANGE name first_name string;

Delete column	alter table employee_data replace columns(id string, first_name string, age int);

github.com/akp0311200

Partitioning in Hive

The partitioning in Hive means dividing the table into some parts based on the values of a particular column like date, course, city or country. The advantage of partitioning is that since the data is stored in slices, the query response time becomes faster.

As we know that Hadoop is used to handle the huge amount of data, it is always required to use the best approach to deal with it. The partitioning in Hive is the best example of it.

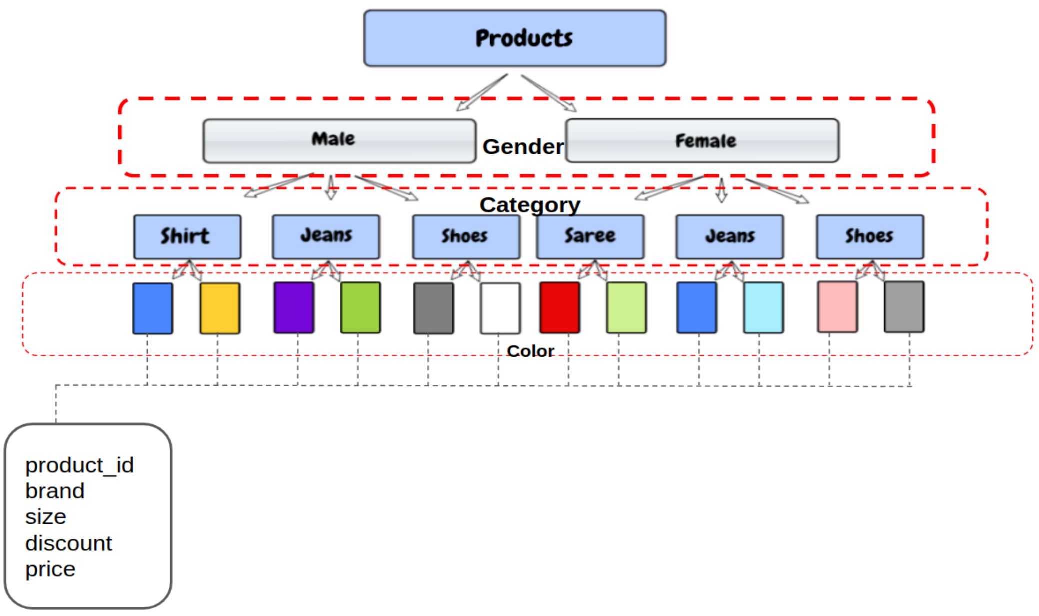
Let's assume we have a data of 10 million students studying in an institute. Now, we have to fetch the students of a particular course. If we use a traditional approach, we have to go through the entire data. This leads to performance degradation. In such a case, we can adopt the better approach i.e., partitioning in Hive and divide the data among the different datasets based on particular columns.

Apache Hive allows us to organize the table into multiple partitions where we can group the same kind of data together. It is used for distributing the load horizontally. Let's understand it with an example:

- Suppose we have to create a table in the hive which contains the product details for a fashion e-commerce company. It has the following columns:

Column	Description
Product ID	Unique ID string for each of the product
Gender	Whether the product is for male/female/other
Category	Category of the product - Shirt, Jeans, Saree, Shoes, etc
Brand	Brand of the product
Color	Color of the product - Yellow, Green, etc
Size	Size of the product - S, M, L, XL, XXL etc
Discount	Discount price if available
Price	Final Sale price of the product

- Now, the first filter that most of the customer uses is Gender then they select categories like Shirt, its size, and color. Let's see how to create the partitions for this example.



- On the other hand, do not create partitions on the columns with very high cardinality. For example- product IDs, timestamp, and price because will create millions of directories which will be impossible for the hive to manage.

  - Step-1 : Create an external table to load the data:

    ```
    CREATE TABLE your_table ( State STRING, City STRING, Pop INT )
    ROW FORMAT DELIMITED
    FIELDS TERMINATED BY ','
    STORED AS TEXTFILE;
    ```

  - Step-2 : Load the data into table :

    ```
    LOAD DATA LOCAL INPATH 'F:\Hadoop\Hive\partitionDataNew.txt' INTO TABLE
    your_table;
    ```

  - Step-3 : Create a partitioned table:

    ```
    CREATE TABLE partitioned_table ( City STRING, Pop INT )
    PARTITIONED BY (State STRING);
    ```

  - Step-4 : Load data from the external table into the partitioned table with dynamic partitioning :

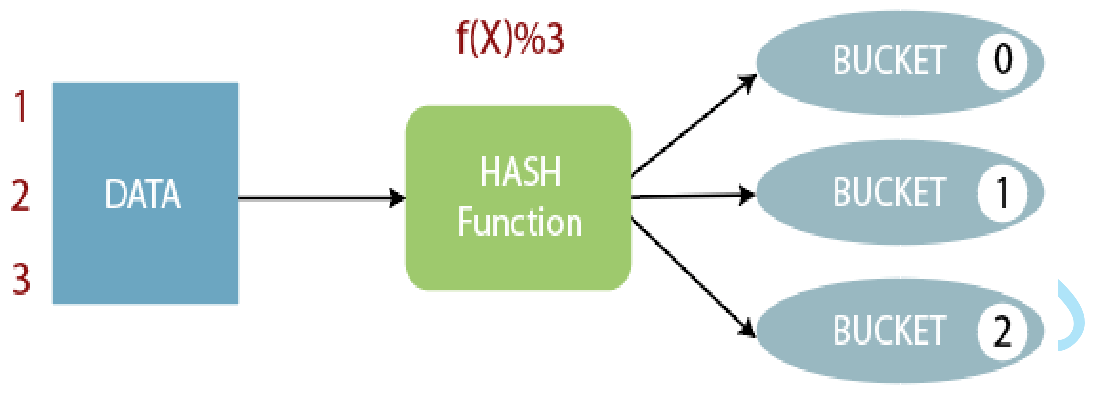
    ```
    FROM your_table
    INSERT OVERWRITE TABLE partitioned_table
    PARTITION (State)
    SELECT City, Pop, State;
    ```

Note: Enabling Dynamic Partitioning in hive

```
SET hive.exec.dynamic.partition=true;
SET hive.exec.dynamic.partition.mode=nonstrict;
```

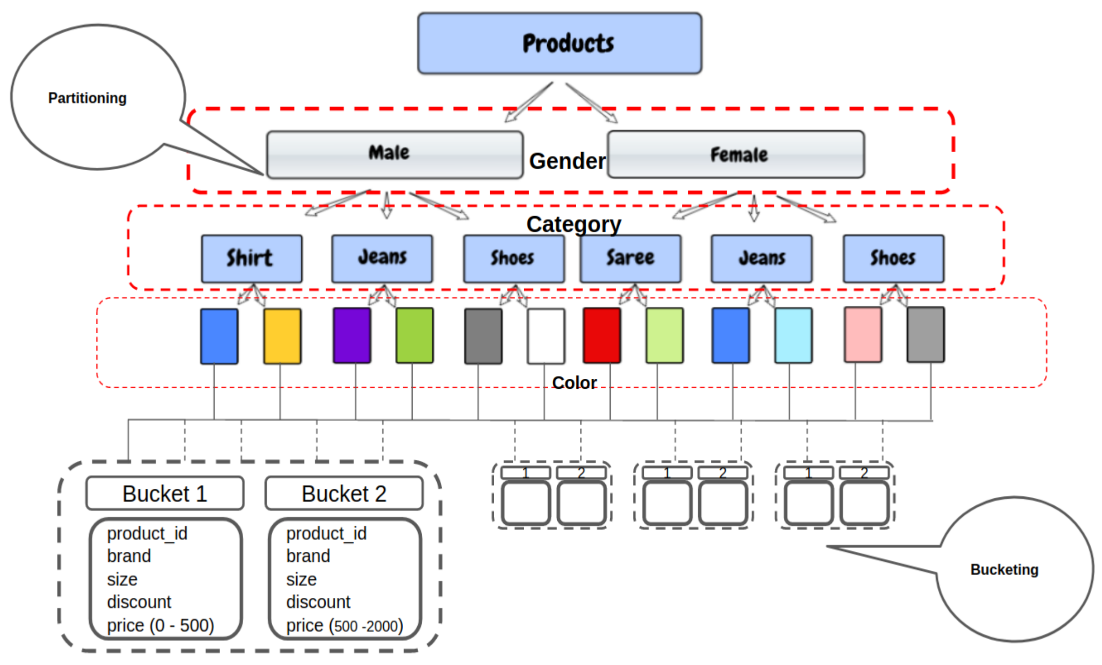
github.com/akp0312000

Bucketing in Hive



- In the above example, we know that we cannot create a partition over the column price because its data type is float and there is an infinite number of unique prices are possible.
- Hive will have to generate a separate directory for each of the unique prices and it would be very difficult for the hive to manage these. Instead of this, we can manually define the number of buckets we want for such columns.
- The concept of bucketing is based on the hashing technique.
- Here, modules of current column value and the number of required buckets is calculated (let say, F(x) % 3).
- Now, based on the resulted value, the data is stored into the corresponding bucket.

.com/akp0312



- Step-01: Create a table to store data :

```
CREATE TABLE tab12( Id INT, Name  STRING, Sal FLOAT)
FORMAT DELIMITED
FIELDS TERMINATED BY ',';
```

- Step-02: Load data into table :

```
LOAD DATA LOCAL INPATH "F:\Hadoop\bucket.txt" INTO TABLE tab12;
```

- Step-03: Enable the bucketing :

```
SET hive.enforce.bucketing = true;
```

- Step-04: Creating bucketing table :

```
CREATE TABLE tab12_bucket(Id INT, Name  STRING, Sal FLOAT)
CLUSTERED BY (Id) INTO 3 BUCKETS
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',';
```

- Step-05: Insert the data of original table (i.e., tab12) into bucketed table (i.e., tab12_bucket) :

```
INSERT OVERWRITE TABLE tab12_bucket
SELECT * FROM tab12;
```

github.com/akp0311200