Hive Partitioning vs Bucketing:

Both Partitioning and Bucketing in Hive are used to improve performance by eliminating table scans when dealing with a large set of data on a Hadoop file system (HDFS). The major difference between Partitioning vs Bucketing lives in the way how they split the data.

Hive Partition is a way to organize large tables into smaller logical tables based on values of columns; one logical table (partition) for each distinct value. In Hive, tables are created as a directory on HDFS. A table can have one or more partitions that correspond to a sub-directory for each partition inside a table directory. let’s assume you have a US census table which contains zip code, city, state and other columns.

Creating a partition on state splits the table into around 50 partitions, when searching for a zipcode with in a state (state=’CA’ and zipCode =’92704′) results in faster as it need to scan only in a state=CA partition directory.

When creating partitions you have to be very cautious with the number of partitions it creates,as having too many partitions creates too many sub-directories in a table directory which bring unnecessarily and overhead to NameNode since it must keep all metadata for the file system in memory.

Hive Bucketing is a technique to split the data into more manageable files,

(By specifying the number of buckets to create). The value of the bucketing column will be hashed by a user-defined number into buckets.

Bucketing can be created on just one column, you can also create bucketing on a partitioned table to further split the data which further improves the query performance of the partitioned table.

Each bucket is stored as a file within the table’s directory or the partitions directories. Note that partition creates a directory and you can have a partition on one or more columns; these are some of the differences between Hive partition and bucket.

Some Points to Remember:

Creating a partition on zipcode of US population is not a good practice as it creates nearly 42,000 directories on HDFS (US has nearly 42,000 zip codes).

Since Bucketing works on hashing, if the data is not equally distributed between hashes, it results in in-equal files and may get into performance issues.

Hive Partitioning Example

CREATE TABLE zipcodes(

RecordNumber int,

Country string,

City string,

Zipcode int)

PARTITIONED BY(state string)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ',';

Hive Bucketing Example:

In the below example, we are creating a bucketing on zipcode column on top of partitioned by state.

CREATE TABLE zipcodes(

RecordNumber int,

Country string,

City string,

Zipcode int)

PARTITIONED BY(state string)

CLUSTERED BY Zipcode INTO 10 BUCKETS

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ',';

Differences Between Hive Partitioning vs Bucketing

| PARTITIONING | BUCKETING |
| --- | --- |
| Directory is created on HDFS for each partition. | File is created on HDFS for each bucket. |
| You can have one or more Partition columns | You can have only one Bucketing column |
| You can’t manage the number of partitions to create | You can manage the number of buckets to create by specifying the count |
| NA | Bucketing can be created on a partitioned table |
| Uses PARTITIONED BY | Uses CLUSTERED BY |

Conclusion:

These two approaches split the table into defined partitions and/or buckets,

which distributes the data into smaller and more manageable parts.

This eliminates table scans when you performing queries on partition and bucket columns.

Creating NON-ORC Table:

First let us Non-ORC table as STUDENT, It is easy that we no need to specify that this table is ORC, by default all the tables that we create are non-orc tables.

CREATE TABLE STUDENT

(

STD\_ID INT,

STD\_NAME STRING,

STD\_GRADE STRING

)

PARTITIONED BY (COUNTRY STRING)

CLUSTERED BY (STD\_GRADE) INTO 3 BUCKETS

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

STORED AS TEXTFILE;

LOAD DATA from local file system to Non-orc Table

Using LOAD DATA INPATH hiveql, we can load the data into non orc table.

LOAD DATA LOCAL INPATH '/home/cloudera/Desktop/student.txt' OVERWRITE INTO TABLE STUDENT

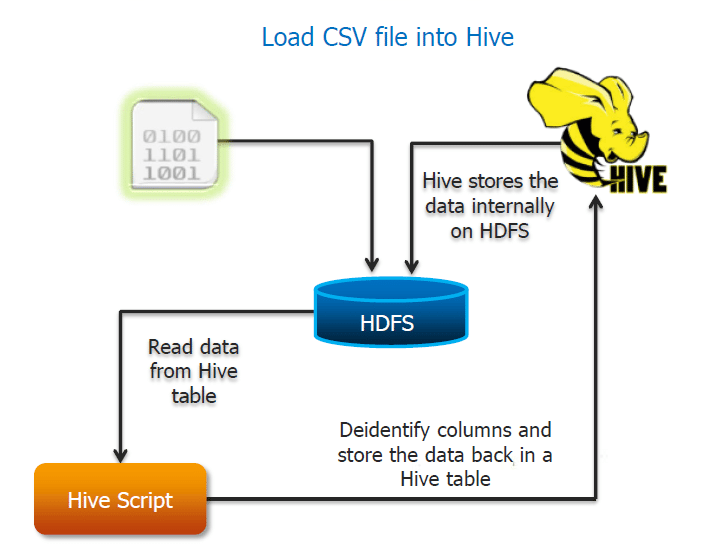
PARTITION(COUNTRY='USA');

CREATING ORC TABLE using non-ORC table

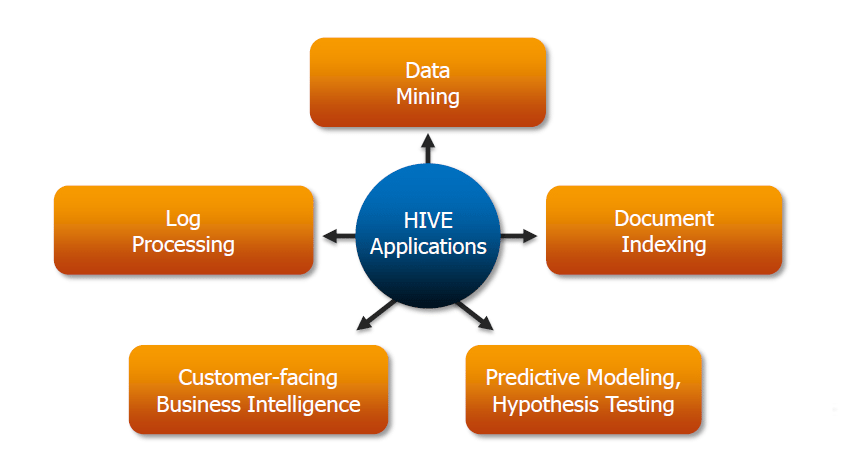
create table studentORC

stored as orc

as select \* from student;



Where to use Hive



Hive consists of the following major components:

Metastore – To store the metadata.

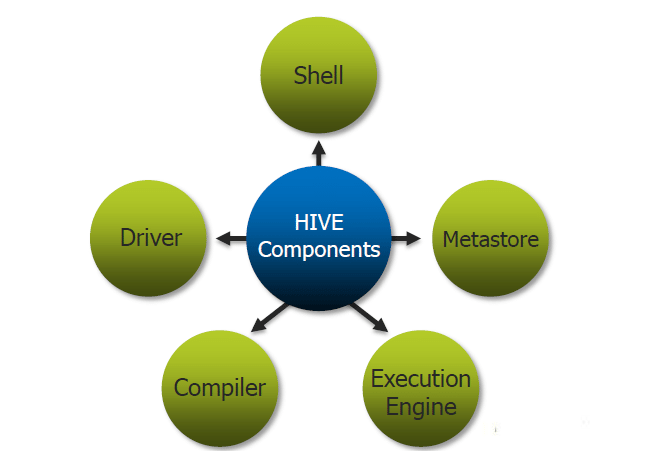
JDBC/ODBC – Query Compiler and Execution Engine to convert SQL queries to a sequence of MapReduce.

SerDe and ObjectInspectors – For data formats and types.

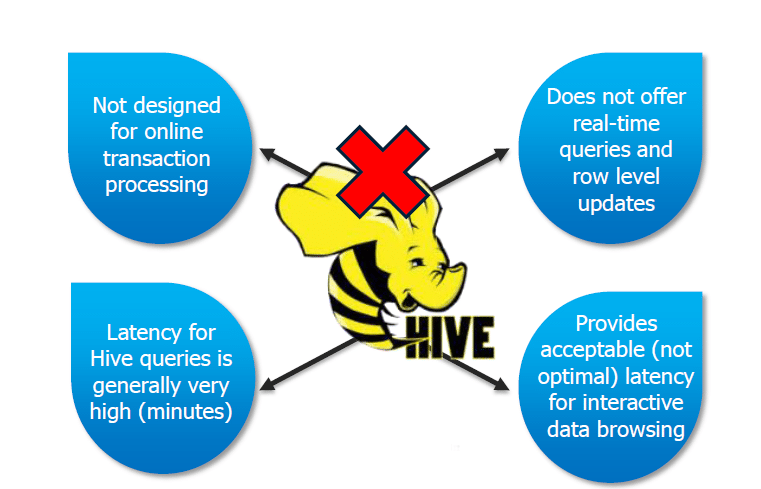
UDF/UDAF – For User Defined Functions.

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

## Components of Hive:



## Limitations of Hive:



Hive has the following limitations and cannot be used under such circumstances:

Not designed for online transaction processing.

Provides acceptable latency for interactive data browsing.

Does not offer real-time queries and row level updates.

Latency for Hive queries is generally very high.