**Bucketing in Hive**

Usually Partitioning in Hive offers a way of segregating hive table data into multiple files/directories. But partitioning gives effective results when, There are limited number of partitions Comparatively equal sized partitions

But this may not possible in all scenarios, like when are partitioning our tables based geographic locations like country, some bigger countries will have large partitions (ex: 4­5 countries itself contributing 70­80% of total data) where as small countries data will create small partitions (remaining all countries in the world may contribute to just 20­30 % of total data). So, In these cases Partitioning will not be ideal. To overcome the problem of over partitioning, Hive provides Bucketing concept, another technique for decomposing table data sets into more manageable parts.

**Features**

1. Bucketing concept is based on (hashing function on the bucketed column) mod (by total number of buckets). The hash\_function depends on the type of the bucketing column.
2. Records with the same bucketed column will always be stored in the same bucket.
3. We use CLUSTERED BY clause to divide the table into buckets.
4. Physically, each bucket is just a file in the table directory, and Bucket numbering is 1­ based.
5. Bucketing can be done along with Partitioning on Hive tables and even without partitioning.
6. Bucketed tables will create almost equally distributed data file parts.

**Advantages**

1. Bucketed tables offer efficient sampling than by non­bucketed tables. With sampling, we can try out queries on a fraction of data for testing and debugging purpose when the original data sets are very huge.
2. As the data files are equal sized parts, map­side joins will be faster on bucketed tables than non­bucketed tables. In Map­side join, a mapper processing a bucket of the left table knows that the matching rows in the right table will be in its corresponding bucket, so it only retrieves that bucket (which is a small fraction of all the data stored in the right table).
3. Similar to partitioning, bucketed tables provide faster query responses than nonbucketed tables
4. Bucketing concept also provides the flexibility to keep the records in each bucket to be sorted by one or more columns. This makes map­side joins even more efficient, since the join of each bucket becomes an efficient merge sort.

**Limitations**

Specifying bucketing doesn’t ensure that the table is properly populated. Data Loading into buckets needs to be handled by our self.

**Example Use Case**

Lets explore the remaining features of Bucketing in Hive with an example Use case, by creating buckets for sample user records provided in the previous post on partitioning –> UserRecords

Let us create the table partitioned by country and bucketed by state and sorted in ascending order of cities.

**Creation of Bucketed Tables**

We can create bucketed tables with the help of CLUSTERED BY clause and optional SORTED BY clause in CREATE TABLE statement. With the help of the below HiveQL we can create bucketed\_user table with above given requirement.

CREATE TABLE bucketed\_user(

firstname VARCHAR(64),

lastname VARCHAR(64),

address STRING,

city VARCHAR(64),

state VARCHAR(64),

post STRING,

phone1 VARCHAR(64),

phone2 STRING,

email STRING,

web STRING

)

COMMENT 'A bucketed sorted user table'

PARTITIONED BY (country VARCHAR(64))

CLUSTERED BY (state) SORTED BY (city) INTO 32 BUCKETS

STORED AS SEQUENCEFILE;

Note:

Unlike partitioned columns (which are not included in table columns definition) , Bucketed columns are included in table definition as shown in above code for state and city columns.

**Inserting data Into Bucketed Tables:**

Similar to partitioned tables, we can not directly load bucketed tables with LOAD DATA (LOCAL) INPATH command, rather we need to use INSERT OVERWRITE TABLE … SELECT …FROM clause from another table to populate the bucketed tables. For this, we will create one temporary table in hive with all the columns in input file from that table we will copy into our target bucketed table.

Lets assume we have created temp\_user temporary table, and below is the HiveQL for populating bucketed table with temp\_user table.

To populate the bucketed table, we need to set the property hive.enforce.bucketing = true, so that Hive knows to create the number of buckets declared in the table definition.

set hive.enforce.bucketing = true;

INSERT OVERWRITE TABLE bucketed\_user PARTITION (country) SELECT firstname , lastname , address , city , state , post , phone1 , phone2 , email , web , country FROM temp\_user;

**Note:**

1. The property hive.enforce.bucketing = true similar to hive.exec.dynamic.partition=true property in partitioning. By Setting this property we will enable dynamic bucketing while loading data into hive table.
2. It will automatically sets the number of reduce tasks to be equal to the number of buckets mentioned in the table definition (for example 32 in our case) and automatically selects the clustered by column from table definition
3. If we do not set this property in Hive Session, we have to manually convey same information to Hive that, number of reduce tasks to be run (for example in our case, by using set mapred.reduce.tasks=32) and CLUSTER BY (state) and SORT BY (city) clause in the above INSERT …SELECT statement at the end.

Lets see the execution of these scripts and their output in the next page.

**Solution For Example Use Case**

Below is the combined HiveQL along with script required for temporary hive table creation. Lets save this HiveQL into **bucketed\_user\_creation.hql** and save the input file provided in example use case section into **user\_table.txt** file in home directory.

set hive.exec.dynamic.partition=true;

set hive.exec.dynamic.partition.mode=nonstrict;

set hive.exec.max.dynamic.partitions.pernode=1000;

set hive.enforce.bucketing = true;

DROP TABLE IF EXISTS bucketed\_user;

CREATE TEMPORARY TABLE temp\_user(

firstname VARCHAR(64),

lastname VARCHAR(64),

address STRING,

country VARCHAR(64),

city VARCHAR(64),

state VARCHAR(64),

post STRING,

phone1 VARCHAR(64),

phone2 STRING,

email STRING,

web STRING

)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

LINES TERMINATED BY '\n'

STORED AS TEXTFILE;

LOAD DATA LOCAL INPATH '/home/user/user\_table.txt' INTO TABLE temp\_user;

CREATE TABLE bucketed\_user

(

firstname VARCHAR(64),

lastname VARCHAR(64),

address STRING,

city VARCHAR(64),

state VARCHAR(64),

post STRING,

phone1 VARCHAR(64),

phone2 STRING,

email STRING,

web STRING

)

COMMENT 'A bucketed sorted user table'

PARTITIONED BY (country VARCHAR(64))

**Note:**

From the above box, we can see that mapreduce job initiated 32 reduce tasks for 32 buckets and four partitions are created by country.

**Table Sampling in Hive**

Table Sampling in hive is nothing but extraction small fraction of data from the original large data sets. It is similar to LIMIT operator in Hive.

But below are the difference between LIMIT and TABLESAMPLE in Hive.

1. In many cases a LIMIT clause executes the entire query, and then only returns a limited results
2. But Sampling will only select a portion of data to perform query.

Now we will do sampling on these bucketed tables to see the performance difference between bucketed and non­bucketed tables. Lets pull the records present in the last bucket of bucketed\_user will return either all the 129 records from first country CA or last country US but we can’t evenly distributed sample records from all countries and states. This can be seen in the below screen. table created above. In the above sampling we can see the sample records from various countries and covering many states and cities, But if we use LIMIT operator on non­bucketed tables it

hive> SELECT firstname, country, state, city FROM temp\_user LIMIT 129 ;

OK

first\_name country state city

Rebbecca AU TA Leith

Stevie AU QL Proston

Mariko AU WA Hamel

Gerardo AU NS Talmalmo

Mayra AU NS Lane Cove

Idella AU WA Cartmeticup

Sherill AU WA Nyamup

Ena AU NS Bendick Murrell

Vince AU QL Purrawunda

Theron AU SA Blanchetown

Amira AU QL Rockside

...............

Louann AU QL Wyandra

William AU QL Goondi Hill

Time taken: 0.232 seconds, Fetched: 129 row(s)

user@tri03ws-386:~$

hive> SELECT firstname, country, state, city FROM bucketed\_user TABLESAMPLE(1 PERCENT);