● Bucketing

Hive partition divides table into number of partitions and these partitions can be further subdivided into more manageable parts known as Buckets or Clusters. The Bucketing concept is based on Hash function, which depends on the type of the bucketing column. Records which are bucketed by the same column will always be saved in the same bucket.

In [Hive Partition,](http://www.hadooptpoint.com/introduction-hive-partition-big-data/)each partition will be created as directory. But in Hive Buckets, each bucket will be created as file. Bucketing can also be done even without partitioning on Hive tables.

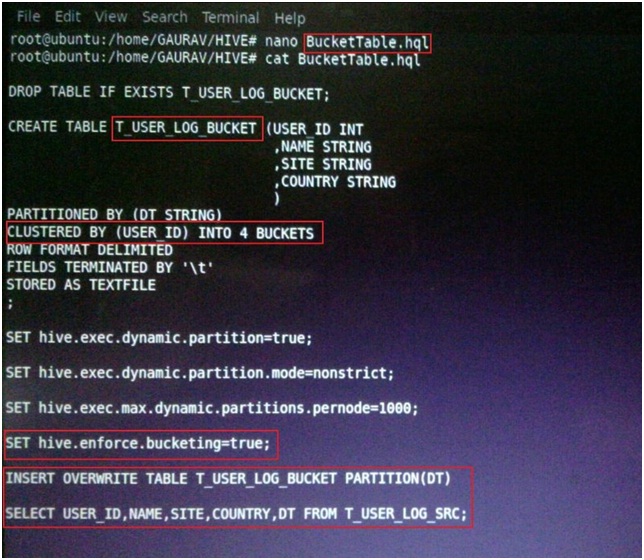
Advantages of Bucketing:

Bucketed tables allows much more efficient [sampling](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Sampling) than the non-bucketed tables. With sampling, we can try out queries on a section of data for testing and debugging purpose when the original data sets are very huge. Here, the user can fix the size of buckets according to the need.

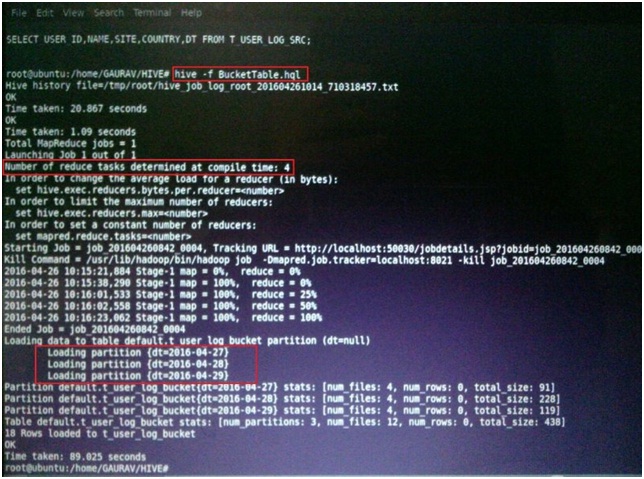
Bucketing concept also provides the flexibility to keep the records in each bucket to be sorted by one or more columns. Since the data files are equal sized parts, map-side joins will be faster on the bucketed tables.

When we write data in bucketed table in hive, it places the data in distinct buckets as files. Hive uses some hashing algorithm to generate a number in range of 1 to N buckets [as mentioned in DDL] and based on the result of hashing, data is placed in a particular buckets as a file. Let's create a hive bucketed table T\_USER\_LOG\_BUCKET with a partition column as DT and having 4 buckets. We specify bucketing column in CLUSTERED BY (column\_name) clause in hive table DDL as shown below in hive script files:

LOAD SCRIPT:

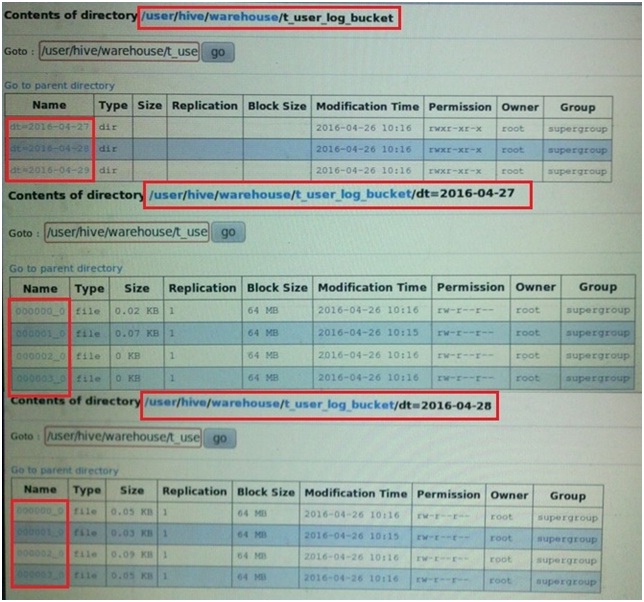


Below screenshots shows the hive script execution to create and load Bucketed table:



As we know that the number of reduce task determine the number of output file so here we have defined table as four buckets so the number of reduce taks is 4 as highlighted in above screenshot.

OUTPUT: Here the output is partitioned on DT column and each partition will contain the four buckets as files. So all three partitions will have four files each as shown in below screenshots



● Bucketing V/S Partitionin

Partitioning data is often used for distributing load horizontally, this has performance benefit, and helps in organizing data in a logical fashion. Example: if we are dealing with a large employee table and often run queries with WHERE clauses that restrict the results to a particular country or department . For a faster query response Hive table can be PARTITIONED BY (country STRING, DEPT STRING). Partitioning tables changes how Hive structures the data storage and Hive will now create subdirectories reflecting the partitioning structure like

.../employees/country=ABC/DEPT=XYZ.

If query limits for employee from country=ABC, it will only scan the contents of one directory country=ABC. This can dramatically improve query performance, but only if the partitioning scheme reflects common filtering. Partitioning feature is very useful in Hive, however, a design that creates too many partitions may optimize some queries, but be detrimental for other important queries. Other drawback is having too many partitions is the large number of Hadoop files and directories that are created unnecessarily and overhead to NameNode since it must keep all metadata for the file system in memory.

Bucketing is another technique for decomposing data sets into more manageable parts. For example, suppose a table using date as the top-level partition and employee\_id as the second-level partition leads to too many small partitions. Instead, if we bucket the employee table and use employee\_id as the bucketing column, the value of this column will be hashed by a user-defined number into buckets. Records with the same employee\_id will always be stored in the same bucket. Assuming the number of employee\_id is much greater than the number of buckets, each bucket will have many employee\_id. While creating table you can specify like CLUSTERED BY (employee\_id) INTO XX BUCKETS; where XX is the number of buckets . Bucketing has several advantages. The number of buckets is fixed so it does not fluctuate with data. If two tables are bucketed by employee\_id, Hive can create a logically correct sampling. Bucketing also aids in doing efficient map-side joins etc.

● Sampling

Sampling is concerned with the selection of a subset of data from a large dataset to run queries and verify results. The dataset may be too large to run queries on the whole data. Therefore in development and testing phases it is a good idea to run queries on a sample of dataset.

TABLESAMPLE Clause

We can run Hive queries on a sample of data using the TABLESAMPLE clause. Any column can be used for sampling the data. We need to provide the required sample size in the queries.

Sampling by Bucketing

We can use TABLESAMPLE clause to bucket the table on the given column and get data from only some of the buckets.

TABLESAMPLE (BUCKET x OUT OF y [ON colname])

colname indicates the column to be used to bucket the data into y buckets[1-y]. All the rows which are in the bucket x are returned.

If the table is not bucketed on the column(s) used in sampling, TABLESAMPLE will scan the entire table and fetch the sample.

If the hive table is bucketed on some column(s), then we can directly use that column(s) to get a sample. In this case Hive need not read all the data to generate sample as the data is already organized into different buckets using the column(s) used in the sampling query. Hive will read data only from some buckets as per the size specified in the sampling query.

Block Sampling

Block sampling is available starting with Hive 0.8.

block\_sample: TABLESAMPLE (n PERCENT)

This will allow Hive to pick up at least n% data size (notice it doesn't necessarily mean number of rows) as inputs. Only CombineHiveInputFormat is supported and some special compression formats are not handled. If we fail to sample it, the input of MapReduce job will be the whole table/partition. We do it in HDFS block level so that the sampling granularity is block size. For example, if block size is 256MB, even if n% of input size is only 100MB, you get 256MB of data.

In the following example the input size 0.1% or more will be used for the query.

SELECT \*

FROM source TABLESAMPLE(0.1 PERCENT) s;

Sometimes you want to sample the same data with different blocks, you can change this seed number:

set hive.sample.seednumber=<INTEGER>;