**Assignment 26.2**

**Problem Statement:**

Explain in brief with an example

● Bucketing

● Bucketing V/S Partitioning

● Sampling

**Answer:**

* **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.

Here, CLUSTERED BY clause is used to divide the table into buckets.

In Hive Partition, 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 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.

***For example,***

CREATE TABLE mytable (

name string,

city string,

employee\_id int )

PARTITIONED BY (year STRING, month STRING, day STRING)

CLUSTERED BY (employee\_id) INTO 256 BUCKETS

* **Bucketing V/S Partitioning:**

Partitioning data is often used for distributing load horizontally, this has performance benefit, and helps in organizing data in a logical fashion**.**

**For 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.

***For example,***

table\_sample: TABLESAMPLE (BUCKET x OUT OF y [ON colname])

The TABLESAMPLE clause allows the users to write queries for samples of the data instead of the whole table. The TABLESAMPLE clause can be added to any table in the FROM clause. The buckets are numbered starting from 1.

colname indicates the column on which to sample each row in the table. colname can be one of the non-partition columns in the table or rand() indicating sampling on the entire row instead of an individual column.

The rows of the table are 'bucketed' on the colname randomly into y buckets numbered 1 through y. Rows which belong to bucket x are returned.

In the following example the 3rd bucket out of the 32 buckets of the table source. 's' is the table alias.

SELECT \* FROM source TABLESAMPLE(BUCKET 3 OUT OF 32 ON rand()) s;

Input pruning: Typically, TABLESAMPLE will scan the entire table and fetch the sample. But, that is not very efficient. Instead, the table can be created with a CLUSTERED BY clause which indicates the set of columns on which the table is hash-partitioned/clustered on.

If the columns specified in the TABLESAMPLE clause match the columns in the CLUSTERED BY clause, TABLESAMPLE scans only the required hash-partitions of the table.

So in the above example, if table 'source' was created with 'CLUSTERED BY id INTO 32 BUCKETS'

TABLESAMPLE(BUCKET 3 OUT OF 16 ON id)

would pick out the 3rd and 19th clusters as each bucket would be composed of (32/16)=2 clusters.

On the other hand the tablesample clause

TABLESAMPLE(BUCKET 3 OUT OF 64 ON id)

would pick out half of the 3rd cluster as each bucket would be composed of (32/64)=1/2 of a cluster.

***Block Sampling:***

Block sampling allows Hive to select at least n% data from the whole dataset.

Sampling granularity is at the HDFS block size level. If HDFS block size is 64MB and n% of input size is only 10MB, then 64MB of data is fetched.

***For example,***

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>;

Or user can specify total length to be read, but it has same limitation with PERCENT sampling. (As of Hive 0.10.0 - https://issues.apache.org/jira/browse/HIVE-3401)

block\_sample: TABLESAMPLE (ByteLengthLiteral)

ByteLengthLiteral : (Digit)+ ('b' | 'B' | 'k' | 'K' | 'm' | 'M' | 'g' | 'G')

In the following example the input size 100M or more will be used for the query.

SELECT \* FROM source TABLESAMPLE(100M) s;

Hive also supports limiting input by row count basis, but it acts differently with above two.

First, it does not need CombineHiveInputFormat which means this can be used with non-native tables.

Second, the row count given by user is applied to each split. So total row count can be vary by number of input splits.

block\_sample: TABLESAMPLE (n ROWS)

For example, the following query will take the first 10 rows from each input split.

SELECT \* FROM source TABLESAMPLE(10 ROWS);