1.Bucketing:

As we all know, Partition helps in increasing the efficiency when performing a query on a table. Now, let’s assume a condition that there is a huge [dataset](https://acadgild.com/big-data/big-data-development-training-certification). At times, even after partitioning on a particular field or fields, the partitioned file size doesn’t match with the actual expectation and remains huge and we want to manage the partition results into different parts. To overcome this problem of partitioning, Hive provides Bucketing concept, which allows user to divide table data sets into more manageable parts.

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.

2.Partitioning vs Bucketing

**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.

3.Sampling.

**Sampling bucketized table:**

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.

**Block sampling:**

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.