

|  |
| --- |
| * **Bucketing** |
|  |
|  |

|  |
| --- |
| 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. |
|  |

|  |
| --- |
| Records with the same bucketed column will always be stored in the same bucket. |
|  |

|  |
| --- |
| We use CLUSTERED BY clause to divide the table into buckets. |
|  |

|  |
| --- |
| Physically, each bucket is just a file in the table directory, and Bucket numbering is 1-based. |
|  |

|  |
| --- |
| Bucketing can be done along with Partitioning on Hive tables and even without partitioning. |
|  |

|  |
| --- |
| Bucketed tables will create almost equally distributed data file parts, unless there is skew in data. |
|  |

|  |
| --- |
| Bucketing is enabled by setting hive.enforce.bucketing = true; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| **Advantages** |
|  |

|  |
| --- |
| Bucketed tables offer efficient sampling than by non-bucketed 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. |
|  |

|  |
| --- |
| As the data files are equal sized parts, map-side joins will be faster on bucketed tables than non-bucketed tables. |
|  |

|  |
| --- |
| Bucketing concept also provides the flexibility to keep the records in each bucket to be sorted by one or more columns. This makes map-side joins even more efficient, since the join of each bucket becomes an efficient merge-sort. |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| * **Bucketing V/S Partitioning** |
|  |
|  |

|  |
| --- |
| Partitioning helps in elimination of data, if used in WHERE clause, where as bucketing helps in organizing data in each partition into multiple files, so that the same set of data is always written in same bucket. |
|  |

|  |
| --- |
| Bucketing helps a lot in joining of columns. |
|  |

|  |
| --- |
| Hive Bucket is nothing but another technique of decomposing data or decreasing the data into more manageable parts or equal parts. |
|  |

|  |
| --- |
| For example we have table with columns like date, employee\_name, employee\_id, salary, leaves etc . |
|  |

|  |
| --- |
| In this table just use date column as the top-level partition and the employee\_id as the second-level partition leads to too many small partitions. |
|  |

|  |
| --- |
| We can use HASH value for bucketing or a range to bucket the data. |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| A Hive table partitioned on year and bucketed into N buckets |
|  |
|  |

|  |
| --- |
|  |

|  |
| --- |
| * **Sampling** |
|  |
|  |

|  |
| --- |
| TABLESAMPLE() gives more disordered and random records from a table as compared to LIMIT. |
|  |

|  |
| --- |
| We can sample using the rand() function, which returns a random number. |
|  |

|  |
| --- |
| SELECT \* from users TABLESAMPLE(BUCKET 3 OUT OF 10 ON rand()) s; |
|  |

|  |
| --- |
|  |

|  |
| --- |
| Here rand() refers to any random column. |
|  |

|  |
| --- |
| The denominator in the bucket clause represents the number of buckets into which data will be hashed. |
|  |

|  |
| --- |
| The numerator is the bucket number selected. SELECT \* from users TABLESAMPLE(BUCKET 2 OUT OF 4 ON name) s; |
|  |

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

SELECT \* FROM buck\_users TABLESAMPLE(BUCKET 1 OUT OF 2 ON id) s LIMIT 1;