Explain in brief with an example

● Bucketing

● Bucketing V/S Partitioning

● Sampling

**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,](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.

**Example:**Let us see how to create and populate bucketed table in the following example, where we try to subdivide a partition table into multiple manageable parts based on a field by using *bucketing* technique.

***Input Dataset to Perform Bucketing Operation:***

To perform the bucketing operation on a dataset, we require an input dataset.

***Dataset Description:***

Column 1 : Street

Column 2 : City

Column 3 : Zip

Column 4 : State

Column 5 : Beds

Column 6 : Baths

Column 7 : Sq\_feet

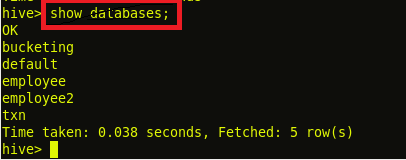
Column 8 : flat\_type

Column 9 : Price

***Selecting the Database:***

Let us begin with the coding part by selecting a database and performing queries on the bucketed table.

Here, we have used a database named ***‘****bucketing****’***which already existsin the Hive metastore database and we will create new tables in the ‘*bucketing*’ database, to perform Bucketing operation on the newly created tables.

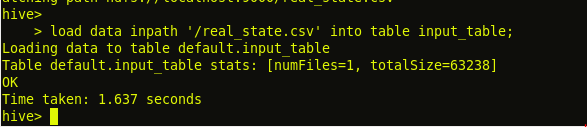


***Creating a New Input Table:***

Next, we have created a new input table with name ***‘input\_table’,***where we will save the contents of the input dataset.

***Load the Input Dataset:***

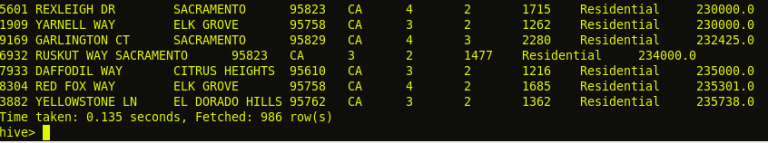
Now, in the below image, we can see that we have loaded the contents of ***real\_state*** into input***\_table*** by using the load command.



***Display the Contents of Table input\_table to Ensure Whether the Input File has been Loaded Successfully or Not:***

By using ‘*select’* statement, we can check the contents of the ***input\_table,***whether we have successfully loaded the contents or not.

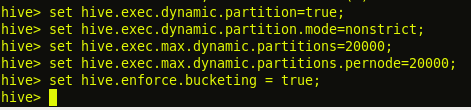
https://i2.wp.com/s3.amazonaws.com/acadgildsite/wordpress_images/bigdatadeveloper/bucketing+in+hive/1.4.1+select+all+from+input_table_1.PNG?resize=357%2C26&ssl=1



We can see from the above image that we have successfully loaded the contents of the input dataset ***real\_state***into ***input\_table***.

***Set the Below Properties in Hive Command Line Before Proceeding Further for Bucketing Scripts:***

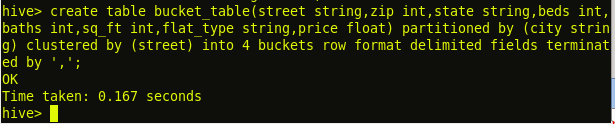
To populate the bucketed table, we have to set **hive.enforce.bucketing** property to ‘true’, so that the Hive knows to create the number of buckets declared in the table definition.



The property **hive.enforce.bucketing = true**is similar to **hive.exec.dynamic.partition = true**, in Hive partitioning. By setting this property, we will enable dynamic bucketing while loading data into the Hive table.

The above **hive.enforce.bucketing = true** property sets the number of reduce tasks to be equal to the number of buckets mentioned in the table definition (Which is ‘4’ in our case) and automatically selects the clustered by column from table definition.

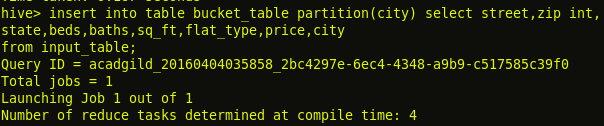
***Creating Bucket Table:***

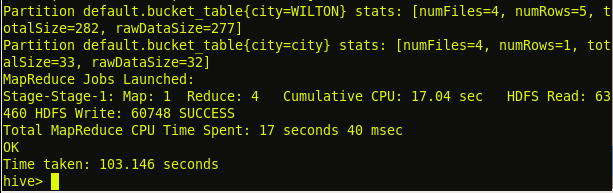


From the above image, we can see that we have created a new bucket table with name ***‘bucket\_table’****,*which is partitioned by ***‘city’***and clustered by ***‘street’*** field with the bucket size of ***‘4’***.

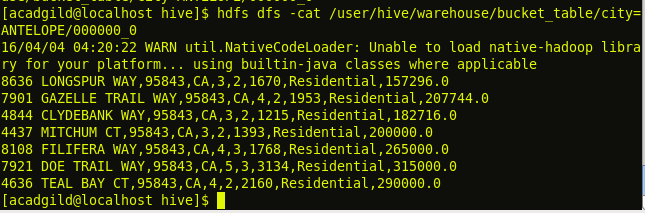
Here, we have decomposed Hive Buckets into ***‘4’***parts.

***Query to Retrieve Data from Bucketed Table:***





**Output:**



Bucketing V/s Partitioning:

Partitioning:

* Hive Partitioning dividing the large amount of data into number pieces of folders based on table columns value.
* Hive Partition is often used for distributing load horizontally, this has performance benefit, and helps in organizing data in a logical fashion.

If you want to use Partition in hive then you should use **PARTITIONED BY** (COL1,COL2…etc) command while hive table creation.

* We can perform partition on any number of columns in a table by using hive partition concept.
* We can perform Hive Partitioning concept on [Hive Tables](http://www.hadooptpoint.com/hive-create-table-examples/) like Managed tables or External tables
* Partitioning is works better when the cardinality of the partitioning field is not too high .

Supposes if we perform partition on Date column then new partition directories created for every date this very burden to name node metadata.

* Partitioning works best when the cardinality of the partitioning field is not too high.

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:

Hive bucketing is responsible for dividing the data into number of equal parts

* If you want to use bucketing in hive then you should use **CLUSTERED BY** (Col) command while creating a table in Hive
* We can perform Hive bucketing concept on Hive Managed tables or External tables
* We can perform Hive bucketing optimization only on one column only not more than one.
* The value of this column will be hashed by a user-defined number into buckets.
* bucketing works well when the field has high cardinality and data is evenly distributed among buckets

If you want to perform queries on Date or Timestamp or other columns which are having high records fields at that time Hive bucketing concept is perfectible.

* We can assign number of number buckets while creating the table.
* Bucketing also very useful in doing efficient map-side joins etc.

**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:**

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

Syntax:

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;