[**https://www.youtube.com/watch?v=wSF5q6nQoqw**](https://www.youtube.com/watch?v=wSF5q6nQoqw)

[**https://acadgild.com/blog/bucketing-in-hive/**](https://acadgild.com/blog/bucketing-in-hive/)

**Limitations of Partitioning:**

1. Partitions may leads to so many files which is overhead to Name node. Because too many partitions leads to too much info needs to be stored on metastore.
2. It may happen that data may not be distributed properly across reducers.

**Advantages of Bucketing:**

1. Joining is very easier in case of bucketing.

Consider a table – employee with – emp\_id, name, location.

And the bucketing is based on emp\_id. (Internally this segregation of data into buckets is based on the hashcode of the emp\_id).

There is one more table – Assets with – emp\_id, assests as columns.

And consider Assets table is also bucketed based on emp\_id.

Now it is very easier to join both the tables since the data is properly segregated into buckets.

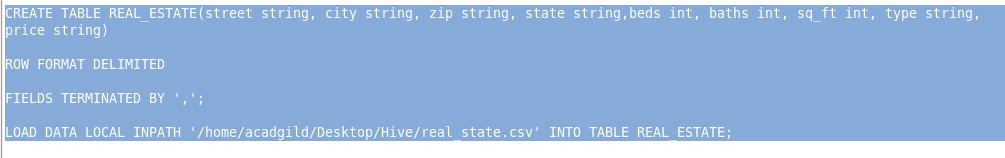
**No of buckets = No of reducers.**

1. Table sampling:

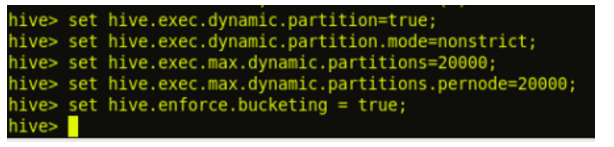
Suppose for any testing purpose I don’t want to test on complete data. Instead I want to test on sample data.

Itinerary ID is unsuitable for partitioning as we learned but it is used frequently for join operations. We can optimize joins by bucketing ‘similar’ IDs so Hive can minimise the processing steps, and reduce the data needed to parse and compare for join operations. Itinerary IDs, of course, have no real similarity and we only need to achieve that the same itinerary IDs from two tables end up in the same processing bucket. A simple trick to do this is to hash the data and store it by hash results, which is what bucketing does.

1. First create a normal table and load the real estate data.

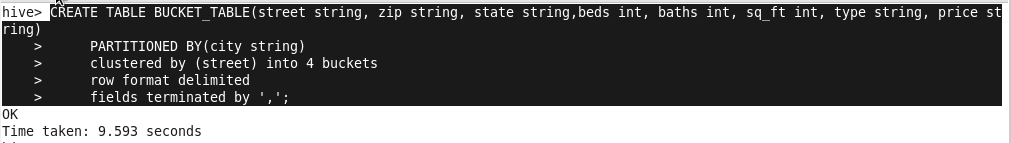


**Before bucketing we need to set some important properties:**



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.

1. Create a bucketed table on street, partitioning with city



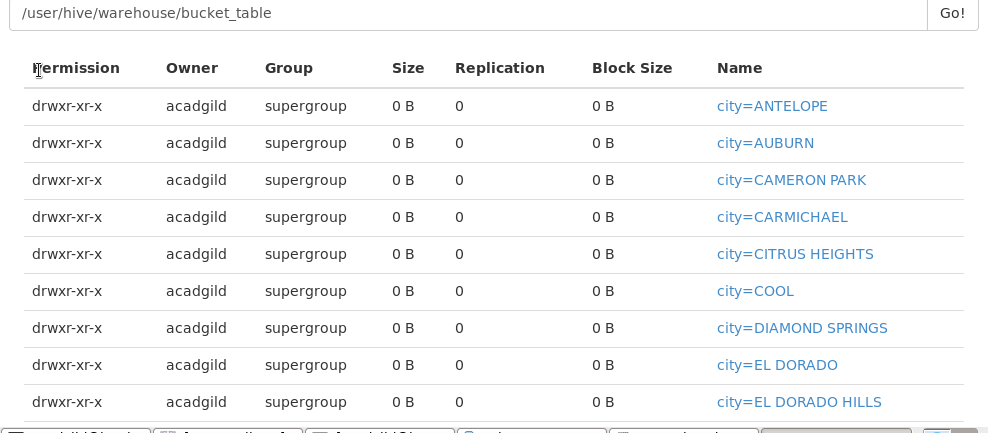
Here, we have decomposed Hive Buckets into ***‘4’***parts.

1. Inserting into bucketed table from sample table.



We can observe from the above image that we are selecting the columns from input\_table and inserting it into our bucketed table ‘bucket\_table’, which is partitioned by city.

Input Data is partitioned based on city



**Sampling**

<https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Sampling>

**JOINS AND TYPES**

[**https://acadgild.com/blog/map-side-joins-in-hive/**](https://acadgild.com/blog/map-side-joins-in-hive/)

edureka: <https://www.edureka.co/blog/map-side-join-vs-join/>

Reduce side join:

If both the datasets are very large then this takes place.

Map side join:

Map side join is a process where joins between two tables are performed in the Map phase without the involvement of Reduce phase

Map-side Joins allows a table to get loaded into memory ensuring a very fast join operation, performed entirely within a mapper and that too without having to use both map and reduce phases.

If one of the data set is relatively small then the smaller one is loaded into memory.