**Explain the following in brief with an example.**

**● 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.

In case your queries frequently run with small table joins , you might see a very substantial decrease in the time taken to compute the queries after usage of map-side joins.

let us create two tables to store these two datasets.

CREATE TABLE IF NOT EXISTS dataset1 ( eid int, first\_name String, last\_name String, email String, gender String, ip\_address String) row format delimited fields terminated BY ',' tblproperties("skip.header.line.count"="1");

CREATE TABLE IF NOT EXISTS dataset2 ( eid int, first\_name String, last\_name String) row format delimited fields terminated BY ',' tblproperties("skip.header.line.count"="1");

 load the data into two tables.

Mapside join can be performed by:

SELECT /\*+ MAPJOIN(dataset2) \*/ dataset1.first\_name, dataset1.eid,dataset2.eid FROM dataset1 JOIN dataset2 ON dataset1.first\_name = dataset2.first\_name;

As it is a Map-side join, the number of reducers will be set to 0 automatically.

**● Reduce side Join**

As the name suggests, in the reduce side join, the reducer is responsible for performing the join operation. It is comparatively simple and easier to implement than the map side join as the sorting and shuffling phase sends the values having identical keys to the same reducer and therefore, by default, the data is organized for us.

The reduce side join is a process where the join operation is performed in the reducer phase. Basically, the reduce side join takes place in the following manner:

* Mapper reads the input data which are to be combined based on common column or join key.
* The mapper processes the input and adds a tag to the input to distinguish the input belonging from different sources or data sets or databases.
* The mapper outputs the intermediate key-value pair where the key is nothing but the join key.
* After the sorting and shuffling phase, a key and the list of values is generated for the reducer.
* Now, the reducer joins the values present in the list with the key to give the final aggregated output.

**ADVANTAGES:**

* It is very easy to implement as we are taking advantage of the inbuilt sorting and shuffling algorithm in the MapReduce framework which combines values of the same key and send it to the same reducer.
* In the reduce side join, input does not require to follow any strict format and therefore, join operation can be performed on unstructured data as well.

**● Bucket Map Join**

The constraint for performing Bucket-Map join is:

If tables being joined are bucketed on the join columns, and the number of buckets in one table is a multiple of the number of buckets in the other table, the buckets can be joined with each other.

To perform bucketing, we need to have bucketed tables.

CREATE TABLE IF NOT EXISTS dataset1\_bucketed ( eid int,first\_name String, last\_name String, email String, gender String, ip\_address String) clustered by(first\_name) into 4 buckets row format delimited fields terminated BY ',';

CREATE TABLE IF NOT EXISTS dataset2\_bucketed (eid int,first\_name String, last\_name String) clustered by(first\_name) into 8 buckets row format delimited fields terminated BY ',' ;

insert the data into the dataset1\_bucketed table.

insert into dataset1\_bucketed select \* from dataset1;

insert the data into dataset2\_bucketed table.

insert into dataset2\_bucketed select \* from dataset2;

Here, for the first table we have created 4 buckets and for the second table we have created 8 buckets on the same column. Now, we can perform Bucket-map join on these two tables.

For performing Bucket-Map join, we need to set this property in the Hive shell.

set hive.optimize.bucketmapjoin = true

SELECT /\*+ MAPJOIN(dataset2\_bucketed) \*/ dataset1\_bucketed.first\_name,dataset1\_bucketed.eid, dataset2\_bucketed.eid FROM dataset1\_bucketed JOIN dataset2\_bucketed ON dataset1\_bucketed.first\_name = dataset2\_bucketed.first\_name ;

**● SMBM Join**

If the tables being joined are sorted and bucketized on the join columns and have the same number of buckets, a sort-merge join can be performed. The corresponding buckets are joined with each other at the mapper.

Here we have 4 buckets for dataset1 and 8 buckets for dataset2.

For performing the SMB-Map join, we need to set the following properties:

Set hive.input.format=org.apache.hadoop.hive.ql.io.BucketizedHiveInputFormat;

set hive.optimize.bucketmapjoin = true;

set hive.optimize.bucketmapjoin.sortedmerge = true;

To perform this join, we need to have the data in the bucketed tables sorted by the join column.

insert overwrite table dataset1\_bucketed select \* from dataset1 sort by first\_name;

The above command will overwrite the data in the old table and insert the data as per the query.

So now the data in the dataset1\_bucketed table is sorted by first\_name.

We will now overwrite the data into the dataset2\_bucketed table, using the following command:

insert overwrite table dataset2\_bucketed select \* from dataset2 sort by first\_name;

let us perform the join between tables having 4 buckets and 8 buckets.

SELECT /\*+ MAPJOIN(dataset2\_bucketed) \*/ dataset1\_bucketed.first\_name,dataset1\_bucketed.eid, dataset2\_bucketed.eid FROM dataset1\_bucketed JOIN dataset2\_bucketed ON dataset1\_bucketed.first\_name = dataset2\_bucketed.first\_name ;