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

**● Map side Join**

**● Reduce side Join**

**● Bucket Map Join**

**● SMBM Join**

**SOLUTION:**

**MAP SIDE JOIN:**

1. Map side join allows a table to be loaded into memory so that a (very fast) join could be performed entirely within a mapper without having to use a Map/Reduce step.

2. If your queries frequently rely on small table joins (e.g. cities or countries, etc.) you might see a very substantial speed-up from using mapjoins.

3.To enable map side joins, set hive.auto.convert.join to true in your config, and Hive will automatically use mapjoins for any tables smaller than hive.mapjoin.smalltable.filesize (default is 25MB).

4. Assume that we have two tables of which one of them is a small table. When we submit a map reduce task, a Map Reduce local task will be created before the original join Map Reduce task which will read data of the small table from HDFS and store it into an in-memory hash table.

5. After reading, it serializes the in-memory hash table into a hash table file.

6. In the next stage, when the original join Map Reduce task is running, it moves the data in the hash table file to the Hadoop distributed cache, which populates these files to each mapper’s local disk. So all the mappers can load this persistent hash table file back into the memory and do the join work as before.

Example:

1. By specifying the keyword, /\*+ MAPJOIN(b) \*/ in the join statement.

2. By setting the following property to true.

hive.auto.convert.join=true

3. For performing Map-side joins, there should be two files, one is of larger size and the other is of smaller size. You can set the small file size by using the following property:

hive.mapjoin.smalltable.filesize=(default it will be 25MB)

4. Now, let us perform Map-side joins and join the two datasets based on their IDs.

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

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

**REDUCE SIDE JOIN:**

1. As the name suggests, in the reduce side join, the reducer is responsible for performing the join operation.

2. 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.

3. Mapper reads the input data which are to be combined based on common column or join key.

4. 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.

5. The mapper outputs the intermediate key-value pair where the key is nothing but the join key.

6. After the sorting and shuffling phase, a key and the list of values is generated for the reducer.

7. Now, the reducer joins the values present in the list with the key to give the final aggregated output.

8. By default, the joins in hive are reduce side joins unless the following property is change:

hive.auto.convert.join=true

**Bucket Map Join:**

1. 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.

2. 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 ',' ;

3. Now we will insert the data into the dataset1\_bucketed table.

insert into dataset1\_bucketed select \* from dataset1;

4. We will now insert the data into dataset2\_bucketed table.

insert into dataset2\_bucketed select \* from dataset2;

5. Here, we have two tables that are bucketed. We can now perform Bucket-map join between these two datasets.

6. 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.

7. 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:**

1. 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.

2. Here we have 4 buckets for dataset1 and 8 buckets for dataset2. Now, we will create another table with 4 buckets for dataset2.

3. 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;

4. To perform this join, we need to have the data in the bucketed tables sorted by the join column. Now, we will re-insert the data into the bucketed tables by using sorting the records.

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

5. 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.

6. Now, let us perform the SMB-Map join on the two tables with 4 buckets in one table and 8 buckets in one table.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

7. Now, 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 ;

8. To perform SMB-Map join, we need to have the same number of buckets in both the tables with the bucketed column sorted.

9. Now, we will create another table for dataset2 having 4 buckets and will insert the data that is sorted by first\_name.

CREATE TABLE IF NOT EXISTS dataset2\_bucketed1 (eid int,first\_name String, last\_name String) clustered by(first\_name) into 4 buckets row format delimited fields terminated BY ',' ;

insert overwrite table dataset2\_bucketed1 select \* from dataset2 sort by first\_name;

10. Now, we have two tables with 4 buckets and the joined column sorted. Let us perform the join query again.

SELECT /\*+ MAPJOIN(dataset2\_sbucketed1) \*/dataset1\_bucketed.first\_name, dataset1\_bucketed.eid, dataset2\_bucketed1.eid FROM dataset1\_bucketed JOIN dataset2\_bucketed1 ON dataset1\_bucketed.first\_name = dataset2\_bucketed1.first\_name ;