1)**Tez Execution Engine** – Hive Optimization Techniques, to increase the Hive performance of our hive query by using our execution engine as Tez.

**set hive.execution.engine=tez;**

2)**Partioning**

3)**Bucketing**

Bucketing improves the join performance if the bucket key and join keys are common. Bucketing in Hive distributes the data in different buckets based on the hash results on the bucket key. It also reduces the I/O scans during the join process if the process is happening on the same keys (columns).

Additionally it’s important to ensure the bucketing flag is set (**SET hive.enforce.bucketing=true;**) every time before writing data to the bucketed table. To leverage the bucketing in the join operation we should **SET hive.optimize.bucketmapjoin=true**. This setting hints to Hive to do bucket level join during the map stage join. It also reduces the scan cycles to find a particular key because bucketing ensures that the key is present in a certain bucket.

4) **ORC format**

Create Table Employee\_ORC (EmployeeID int, EmployeeName varchar(100),Age int)

STORED AS ORC tblproperties("compress.mode"="SNAPPY");

Select \* from Employee Insert into Employee\_ORC;

Create Table Employee\_Details\_ORC (EmployeeID int, Address varchar(100)

,Designation Varchar(100),Salary int)

STORED AS ORC tblproperties("compress.mode"="SNAPPY");

Select \* from Employee\_Details Insert into Employee\_Details\_ORC;

Select a.EmployeeID, a.EmployeeName, b.Address,b.Designation from Employee\_ORC a

Join Employee\_Details\_ORC b

On a.EmployeeID=b.EmployeeID;

**5) Vectorization in hive**

hive> set hive.vectorized.execution.enabled = true;

hive> set hive.vectorized.execution.reduce.enabled = true;

hive> set hive.vectorized.execution.reduce.groupby.enabled = true;

**6) Cost Bases optimization in HIve**

To use CBO, set the following parameters at the beginning of your query:

set hive.cbo.enable=true;

set hive.compute.query.using.stats=true;

set hive.stats.fetch.column.stats=true;

set hive.stats.fetch.partition.stats=true;

Then, prepare the data for CBO by running Hive’s “analyze” command to collect various statistics on the tables for which we want to use CBO.

**7) Hive Index** – Hive Optimization Techniques, one of the best ways is Indexing. To increase your query performance indexing will definitely help.

Basically, for the original table use of indexing will create a separate called index table which acts as a reference.

**8) Enable compression in HIve** –

**a) Enable compression on intermediate Table**

hive> set hive.exec.compress.intermediate=true;

hive> set hive.intermediate.compression.codec=org.apache.hadoop.io.compress.SnappyCodec;

hive> set hive.intermediate.compression.type=BLOCK;

**b) Enable compression on finalOutput table**

hive> set hive.exec.compress.output=true;

hive> set mapreduce.output.fileoutputformat.compress=true;

hive> set mapreduce.output.fileoutputformat.compress.codec=org.apache.hadoop.io.compress.GzipCodec;

hive> set mapreduce.output.fileoutputformat.compress.type=BLOCK;

**9) Avoid Global Sorting in HIVE**

Global Sorting in Hive can be achieved in Hive with **ORDER BY** clause but this comes with a drawback. ORDER BY produces a result by setting the number of reducers to one, making it very inefficient for large datasets.

When a globally sorted result is not required, then we can use **SORT BY** clause. SORT BY produces a sorted file per reducer.

If we need to control which reducer a particular row goes to, we can use **DISTRIBUTE BY** clause, for example,

SELECT id, name, salary, dept FROM employee

DISTRIBUTE BY dept

SORT BY id ASC, name DESC;

**10) Map JOIN**

Map joins are really efficient if a table on the other side of a join is small enough to fit in the memory . Hive supports a parameter, **hive.auto.convert.join**, which when it’s set to “true” suggests that Hive try to map join automatically. When using this parameter, be sure the auto convert is enabled in the Hive environment.