**Hive optimization**

Hive compression codes

Predicate push down

Vectorization

Cost based optimization

**Join ==**Map,skew,bucket

Parallel execution

**Map join:**

**🡺Many time we face a situation that we have very small tables in hive but when we query these tables then it takes long time.**

**🡺Join is a clause that combines the records of two tables (or Data-Sets).**

**🡺There are two ways to enable it.**

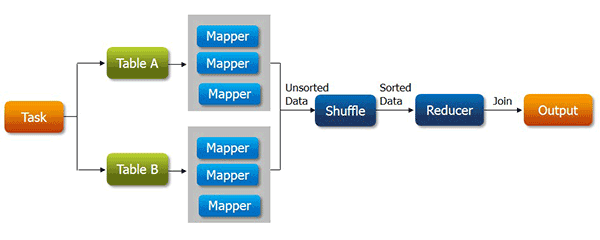
**First is by using a hint, which looks like /\*+ MAPJOIN(aliasname), MAPJOIN(anothertable) \*/. This C-style comment should be placed immediately following the SELECT. It directs Hive to load aliasname (which is a table or alias of the query) into memory.**

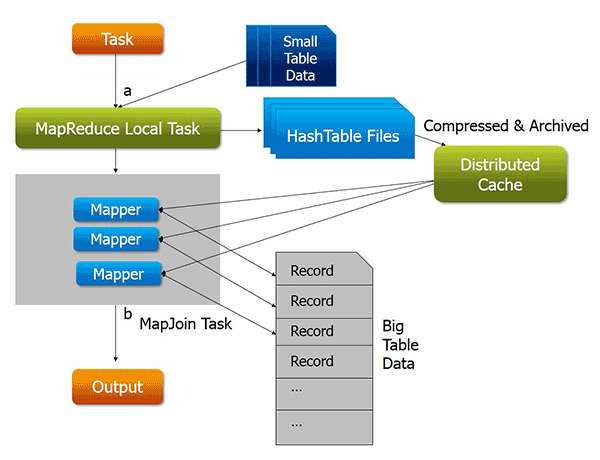
**Eg:-SELECT /\*+ MAPJOIN(c) \*/ \* FROM orders o JOIN cities c ON (o.city\_id = c.id);**

**🡺Another (better, in my opinion) way to turn on mapjoins is to let Hive do it automatically.**

**Simply set hive.auto.convert.join=true in your config, and Hive will automatically use mapjoins for any tables smaller than hive.mapjoin.smalltable.filesize (default is 2,50,00000 i.e; 25MB (1mb=10,00000)**

**With Mapreduce task:**



**With MapJoin:**

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

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.

**Advantages of using map side join:**

• 🡺Map-side join helps in minimizing the cost that is incurred for sorting and merging in the shuffle and reduce stages.

•🡺Map-side join also helps in improving the performance of the task by decreasing the time to finish the task.

**Disadvantages of Map-side join:**

🡺Map side join is adequate only when one of the tables on which you perform map-side join operation is small enough to fit into the memory. Hence it is not suitable to perform map-side join on the tables which are huge data in both of them.

**one of the tables is small enough to fit in memory to complete the job in a short span of time.**

## using a bucket map join

A bucket map join is used when the tables are large and all the tables used in the join are bucketed on the join columns. In this type of join, one table should have buckets in multiples of the number of buckets in another table. For example, if one table has 2 buckets then the other table must have either 2 buckets or a multiple of 2 buckets (2, 4, 6, and so on). If the preceding condition is satisfied then the joining can be done at the mapper side only, otherwise a normal inner join is performed. This means that only the required buckets are fetched on the mapper side and not the complete table. That is, only the matching buckets of all small tables are replicated onto each mapper. Doing this, the efficiency of the query is improved drastically. In a bucket map join, data is not sorted.

**Set hive.optimize.bucketmapjoin=true;**

## Using a skew join

## A skew join is used when there is a table with skew data in the joining column. A skew table is a table that is having values that are present in large numbers in the table compared to other data. Skew data is stored in a separate file while the rest of the data is stored in a separate file.If there is a need to perform a join on a column of a table that is appearing quite often in the table,

## the data for that particular column will go to a single reducer, which will become a bottleneck while performing the join. To reduce this, a skew join is used.

**set hive.optimize.skewjoin=true;**

**set hive.skewjoin.key=100000;**

**create table emp(....,...,..,..)SKEWED BY (col\_name, col\_name, ...) ON ((col\_value, col\_value, ...), (col\_value, col\_value, ...), ...)[STORED AS DIRECTORIES];**

**Parallel execution:**

Hadoop can execute MapReduce jobs in parallel, and several queries executed on Hive automatically use this parallelism. However, single, complex Hive queries commonly are translated to a number of MapReduce jobs that are executed by default sequencing. Often though, some of a query’s MapReduce stages are not interdependent and could be executed in parallel.

**SET hive.exec.parallel=true**.

**Vectorization:**

Vectorized query execution is a Hive feature that greatly **reduces the CPU usage** for **typical query operations like scans, filters, aggregates, and joins**

Vectorization allows Hive to process a batch of rows together instead of processing one row at a time. Each batch is usually an array of primitive types. Operations are performed on the entire column vector, which improves the instruction pipelines and cache usage.

To use vectorized query execution, you must store your data in[**ORC**](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+ORC)format

**SET hive.vectorized.execution.enabled=true ;**

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

**Supported Functionality**

**->The current implementation supports only single table read-only queries. DDL queries or DML queries are not supported.**

-> The supported operators are **selection**, **filter** and **group by**.

**-> Partitioned** tables are supported.

Seeing whether vectorization is used for a query:

>create table vectorizedtable(state string,id int) stored as orc tblproperties (“orc.compress" = “SNAPPY”);;

>insert into vectorizedtable values('haryana',1);

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

>explain select count(\*) from vectorizedtable;

.