Hive Advanced

BAS Academy

Agenda

- Dynamic Partitions
- Hive Buckets
- Hive Skew
- Multi-Table/File Insert
- Views
- Index
- Hive Analytics Functions
- Hive Cost Based Optimization
- Vectorization
- Hive on Tez
- Smart Queries
- Optimized Joins
- Hive Optimization Tips
- Hands On

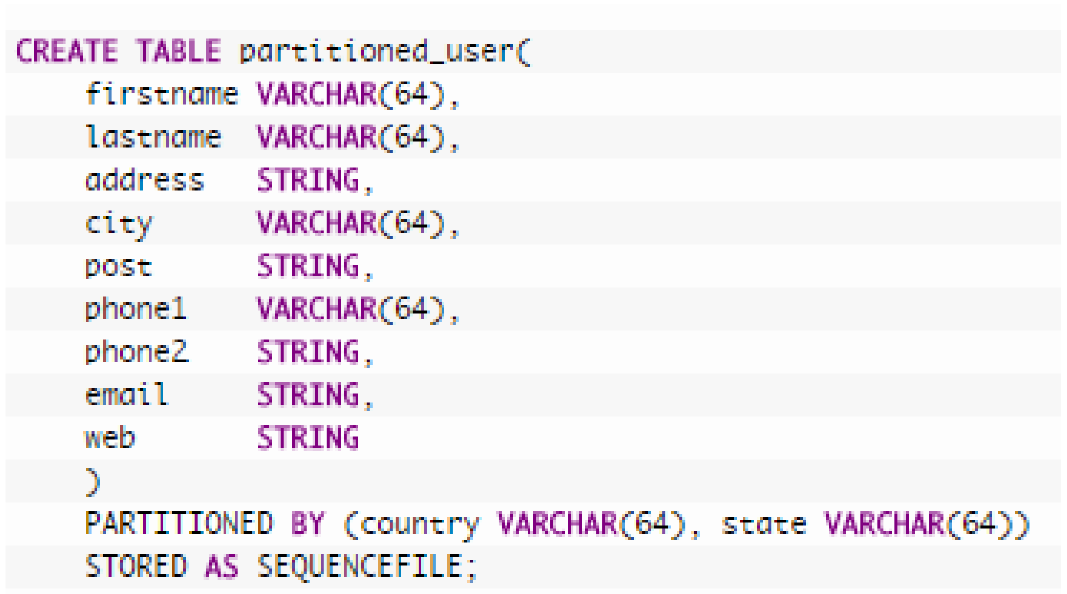
Dynamic Partitions

- Used when the values for partition columns are known only during loading of the data into a Hive table
- Hive automatically takes care of updating the Hive metastore when using dynamic partitions.
- Table creation semantics are the same for both static and dynamic partitioning.
- In order to detect the values for partition columns automatically, partition columns must be specified at the end of the “SELECT” clause in the same order as they are specified in the “PARTITIONED BY” clause while creating the table.

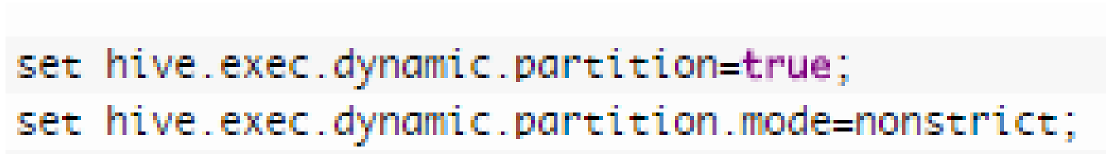
Do the following settings. Dynamic partitioning is disabled by default

- SET hive.exec.dynamic.partition.mode=nonstrict;
- SET hive.exec.dynamic.partition=true;

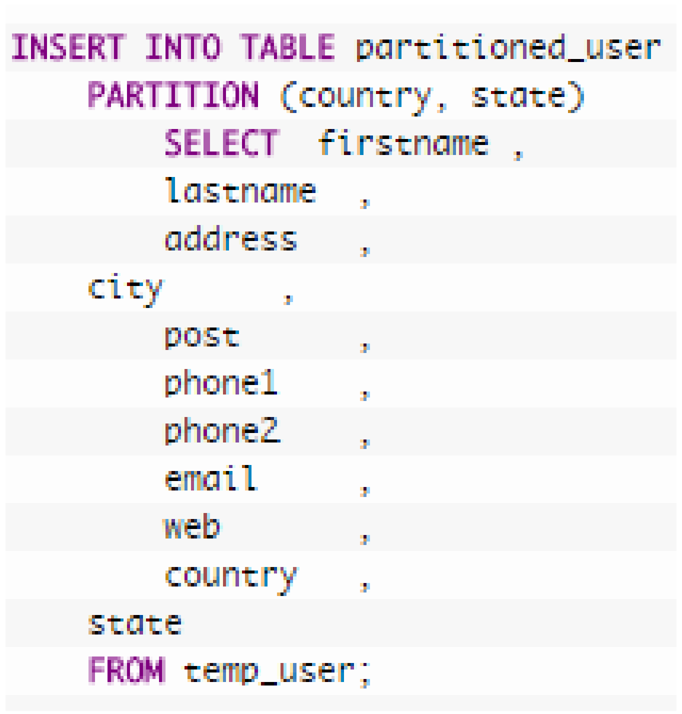
Example - Dynamic Partitions



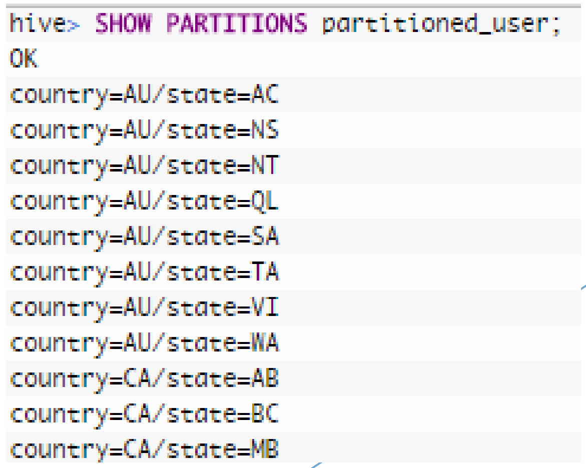
```
CREATE TABLE partitioned_user(
    firstname VARCHAR(64),
    lastname  VARCHAR(64),
    address   STRING,
    city      VARCHAR(64),
    post      STRING,
    phone1    VARCHAR(64),
    phone2    STRING,
    email     STRING,
    web       STRING
    )
    PARTITIONED BY (country VARCHAR(64), state VARCHAR(64))
    STORED AS SEQUENCEFILE;
```



```
set hive.exec.dynamic.partition=true;
set hive.exec.dynamic.partition.mode=nonstrict;
```



```
INSERT INTO TABLE partitioned_user
    PARTITION (country, state)
        SELECT  firstname ,
        lastname      ,
        address       ,
    city          ,
        post          ,
        phone1        ,
        phone2        ,
        email         ,
        web           ,
        country       ,
    state
    FROM temp_user;
```



```
hive> SHOW PARTITIONS partitioned_user;
OK
country=AU/state=AC
country=AU/state=NS
country=AU/state=NT
country=AU/state=QL
country=AU/state=SA
country=AU/state=TA
country=AU/state=VI
country=AU/state=WA
country=CA/state=AB
country=CA/state=BC
country=CA/state=MB
```

Hive Buckets

- Bucketing decomposes data into more manageable or equal parts
- Buckets are created using the clustered by clause.
- It can be done with or without partitioning
- Each bucket is a file in the table directory
- Hive distribute the rows across the buckets by using the function

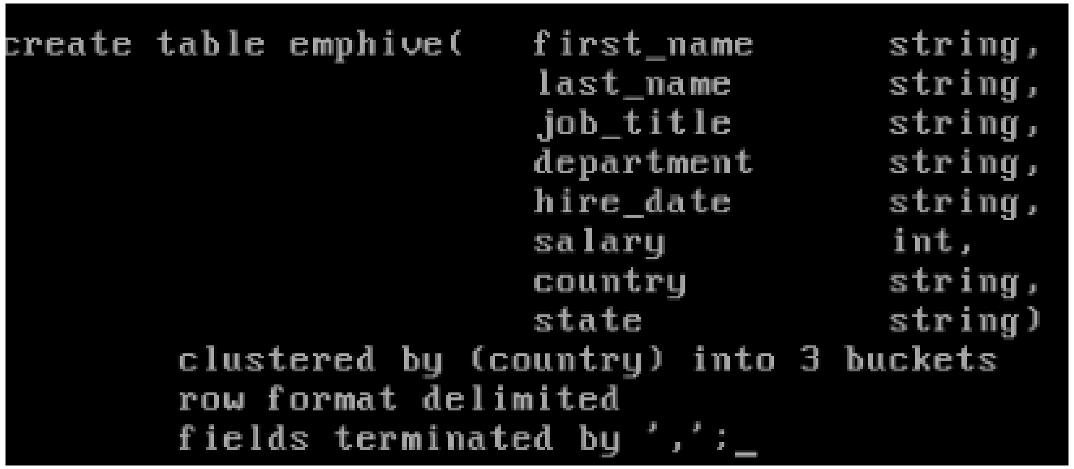
hash_function(bucketing_column) mod num_buckets

- More efficient queries - Especially when performing joins on the same bucketed columns
- More efficient sampling - Because the data is already split up into smaller pieces

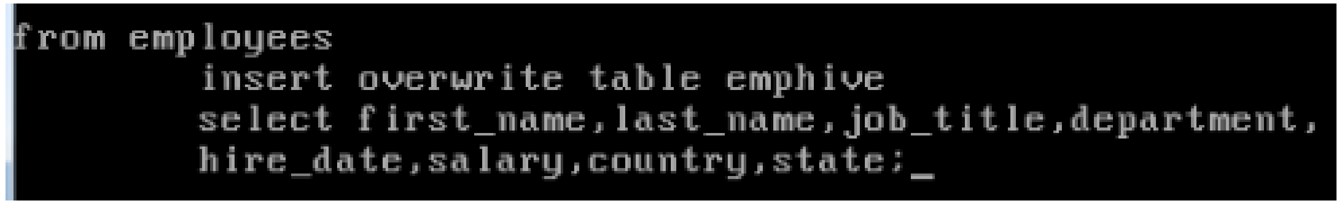
Property:

- set hive.enforce.bucketing = true

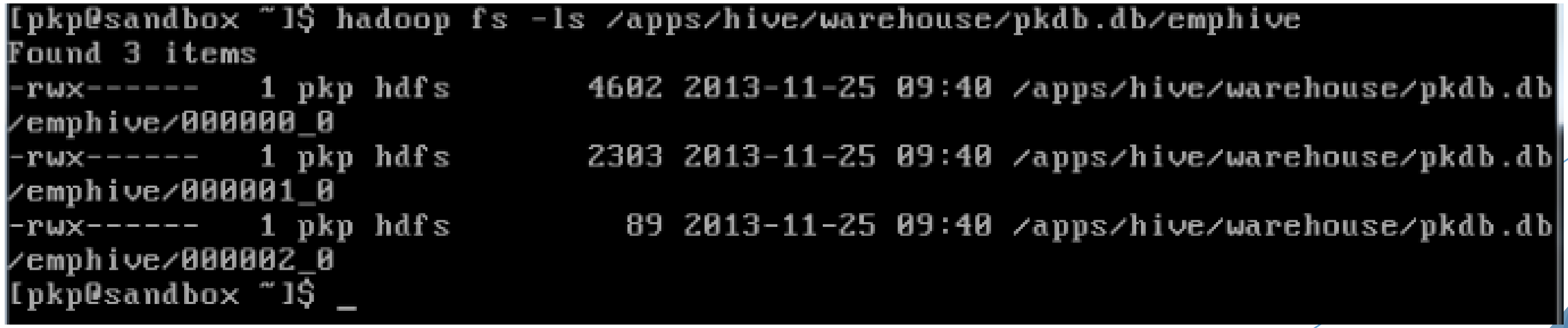
Example - Hive Buckets



```
create table emphive(   first_name      string,
                        last_name       string,
                        job_title       string,
                        department      string,
                        hire_date       string,
                        salary          int,
                        country         string,
                        state           string)
        clustered by (country) into 3 buckets
        row format delimited
        fields terminated by ',';_
```



```
from employees
        insert overwrite table emphive
        select first_name,last_name,job_title,department,
        hire_date,salary,country,state;_
```



```
[pkp@sandbox ~]$ hadoop fs -ls /apps/hive/warehouse/pkdb.db/emphive
Found 3 items
-rwx------   1 pkp hdfs       4602 2013-11-25 09:40 /apps/hive/warehouse/pkdb.db
/emphive/000000_0
-rwx------   1 pkp hdfs       2303 2013-11-25 09:40 /apps/hive/warehouse/pkdb.db
/emphive/000001_0
-rwx------   1 pkp hdfs         89 2013-11-25 09:40 /apps/hive/warehouse/pkdb.db
/emphive/000002_0
[pkp@sandbox ~]$ _
```

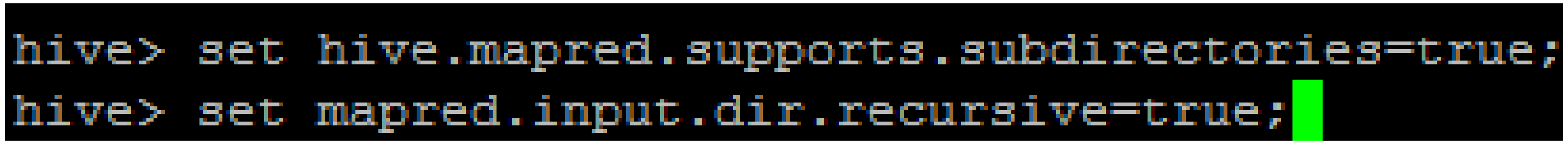
Hive Skew

- A skewed table is a special type of table where the values that appear very often (heavy skew) are split out into separate files and rest of the values go to some other file.
- By specifying the skewed values Hive will split those out into separate files automatically
- During queries it can skip (or include) whole files thereby enhancing the performance.

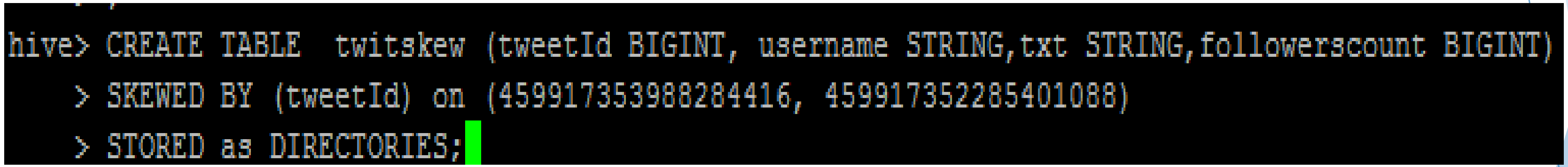
Properties:

- set hive.mapred.supports.subdirectories=true;
- set mapred.input.dir.recursive=true;

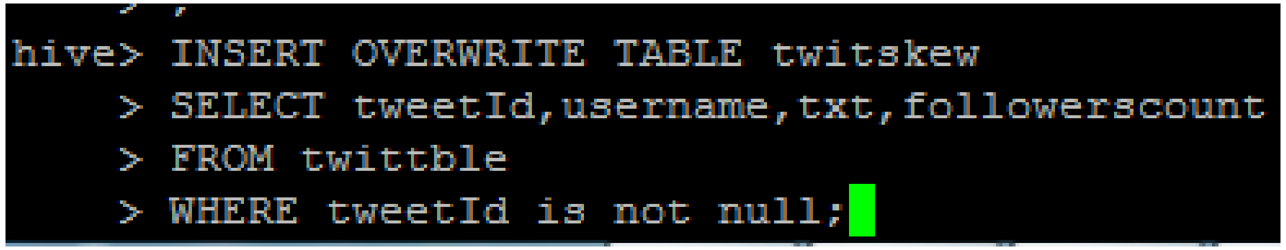
Example – Hive Skew



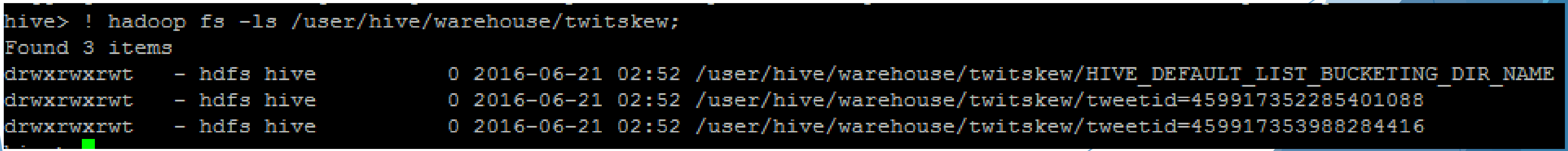
```
hive> set hive.mapred.supports.subdirectories=true;
hive> set mapred.input.dir.recursive=true;
```



```
hive> CREATE TABLE  twitskew (tweetId BIGINT, username STRING,txt STRING,followerscount BIGINT)
    > SKEWED BY (tweetId) on (459917353988284416, 459917352285401088)
    > STORED as DIRECTORIES;
```



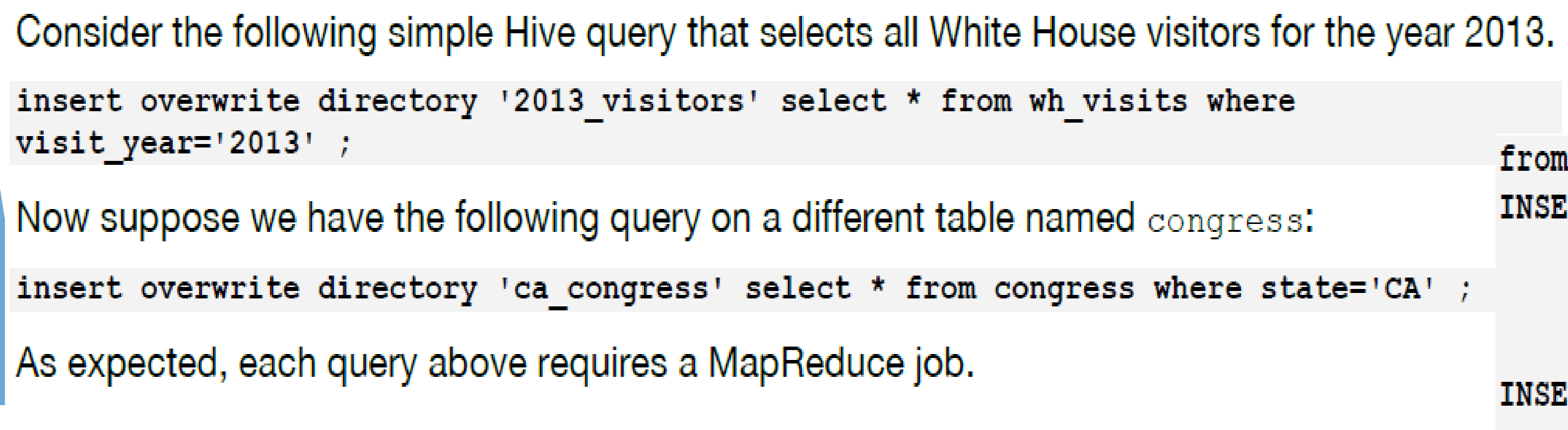
```
hive> INSERT OVERWRITE TABLE twitskew
    > SELECT tweetId,username,txt,followerscount
    > FROM twittble
    > WHERE tweetId is not null;
```



```
hive> ! hadoop fs -ls /user/hive/warehouse/twitskew;
Found 3 items
drwxrwxrwt   - hdfs hive          0 2016-06-21 02:52 /user/hive/warehouse/twitskew/HIVE_DEFAULT_LIST_BUCKETING_DIR_NAME
drwxrwxrwt   - hdfs hive          0 2016-06-21 02:52 /user/hive/warehouse/twitskew/tweetid=459917352285401088
drwxrwxrwt   - hdfs hive          0 2016-06-21 02:52 /user/hive/warehouse/twitskew/tweetid=459917353988284416
```

Multi-Table/File Insert

- Run multiple queries within a single MapReduce job.
- You can gain a lot of performance by running two tasks at the same time instead of running two separate MapReduce jobs.



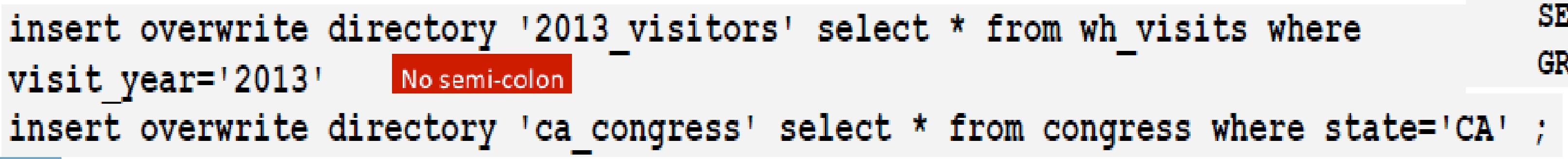
Consider the following simple Hive query that selects all White House visitors for the year 2013.

```
insert overwrite directory '2013_visitors' select * from wh_visits where
visit_year='2013' ;
```

Now suppose we have the following query on a different table named `congress`:

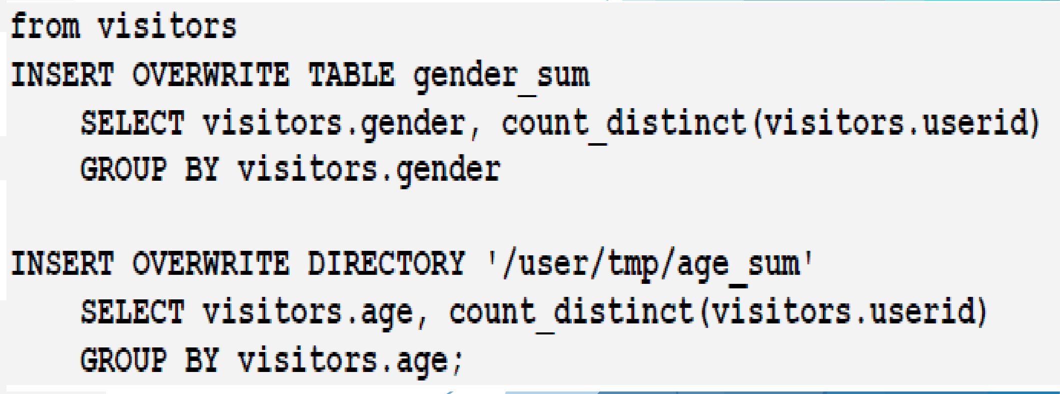
```
insert overwrite directory 'ca_congress' select * from congress where state='CA' ;
```

As expected, each query above requires a MapReduce job.



```
insert overwrite directory '2013_visitors' select * from wh_visits where
visit_year='2013'
insert overwrite directory 'ca_congress' select * from congress where state='CA' ;
```

No semi-colon



```
from visitors
INSERT OVERWRITE TABLE gender_sum
    SELECT visitors.gender, count_distinct(visitors.userid)
    GROUP BY visitors.gender

INSERT OVERWRITE DIRECTORY '/user/tmp/age_sum'
    SELECT visitors.age, count_distinct(visitors.userid)
    GROUP BY visitors.age;
```

Views

- A view in Hive is defined by a SELECT statement and allows you to treat the result of the query like a table.
- Define a view to reduce the complexity of a query.
- Define a view that contains only the columns and rows that the user needs

```
CREATE VIEW 2010_visitors AS
    SELECT fname, lname, time_of_arrival, info_comment
    FROM wh_visits
    WHERE
       cast(substring(time_of_arrival,6,4) AS int) >= 2010
    AND
       cast(substring(time_of_arrival,6,4) AS int) < 2011;
```

```
hive> describe 2010_visitors;
OK
fname                      string                        None
lname                      string                        None
time_of_arrival            string                        None
info_comment               string                        None
```

```
hive> show tables;
OK
2010_visitors
wh_visits
```

```
DROP VIEW 2010_visitors;
```

Index

- An Index acts as a reference to the records.
- It improves the performance of hive query
- Instead of searching all the records, we can refer to the index to search for a particular record
- Hive doesn't provide automatic index maintenance, so you need to rebuild the index if you overwrite or append data to the table
- Maintaining an index requires extra disk space and building an index has a processing cost.

```
CREATE INDEX zip_index
ON TABLE customer(zip)
AS 'org.apache.hadoop.hive.ql.index.compact.CompactIndexHandler'
WITH DEFERRED REBUILD;
```

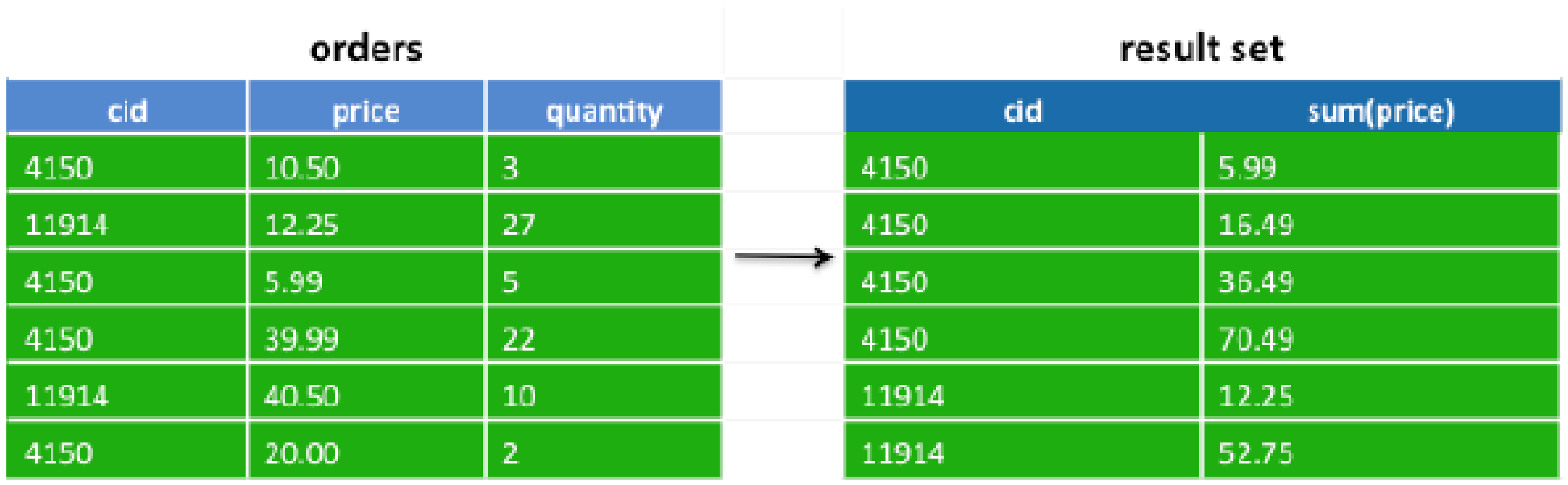
- WITH DEFERRED REBUILD portion of the command prevents the index from immediately being built. To build the index you can issue the following command:

```
ALTER INDEX zip_index
ON customer
REBUILD;
```

- In addition to the CompactIndexHandler, BitMap indexes are supported as of version 0.8.
- A BitMap index is ideal for columns with a limited number of distinct values.

Windows

- The OVER clause allows you to define a window over which to perform a specific calculation.



orders

cid	price	quantity
4150	10.50	3
11914	12.25	27
4150	5.99	5
4150	39.99	22
11914	40.50	10
4150	20.00	2

result set

cid	sum(price)
4150	5.99
4150	16.49
4150	36.49
4150	70.49
11914	12.25
11914	52.75

CID	Price (after order by)	Sum(Price)
4150	5.99	5.99
4150	10.5	16.49
4150	20	36.49
4150	39.99	70.49
11914	12.25	12.25
11914	40.5	52.75

```
SELECT cid, sum(price) OVER (PARTITION BY cid ORDER BY price
   ROWS BETWEEN 2 PRECEDING AND CURRENT ROW) FROM orders;
```

Hive Analytics Functions

RANK	Returns the rank of each row within the partition of a result set Ties are assigned the same rank, with the next ranking(s) skipped.
DENSE_RANK	Returns the rank of rows within the partition of a result set without any gaps in the ranking No ranks are skipped if there are ranks with multiple items.
PERCENT_RANK	Calculates the relative rank of a row within a group of rows
ROW_NUMBER	Returns the sequential number of a row within a partition of a result set
NTILE	Distributes the rows in an ordered partition into a specified number of groups. For each row, NTILE returns the number of the group to which the row belongs

Hive ORC Files

- The Optimized Row Columnar (ORC) file format provides a highly efficient way to store Hive data.
- Using ORC files improves performance when Hive is reading, writing, and processing data.
- Breaks file into set of rows called as stripes
- Default stripe size is 256 MB

Footer:

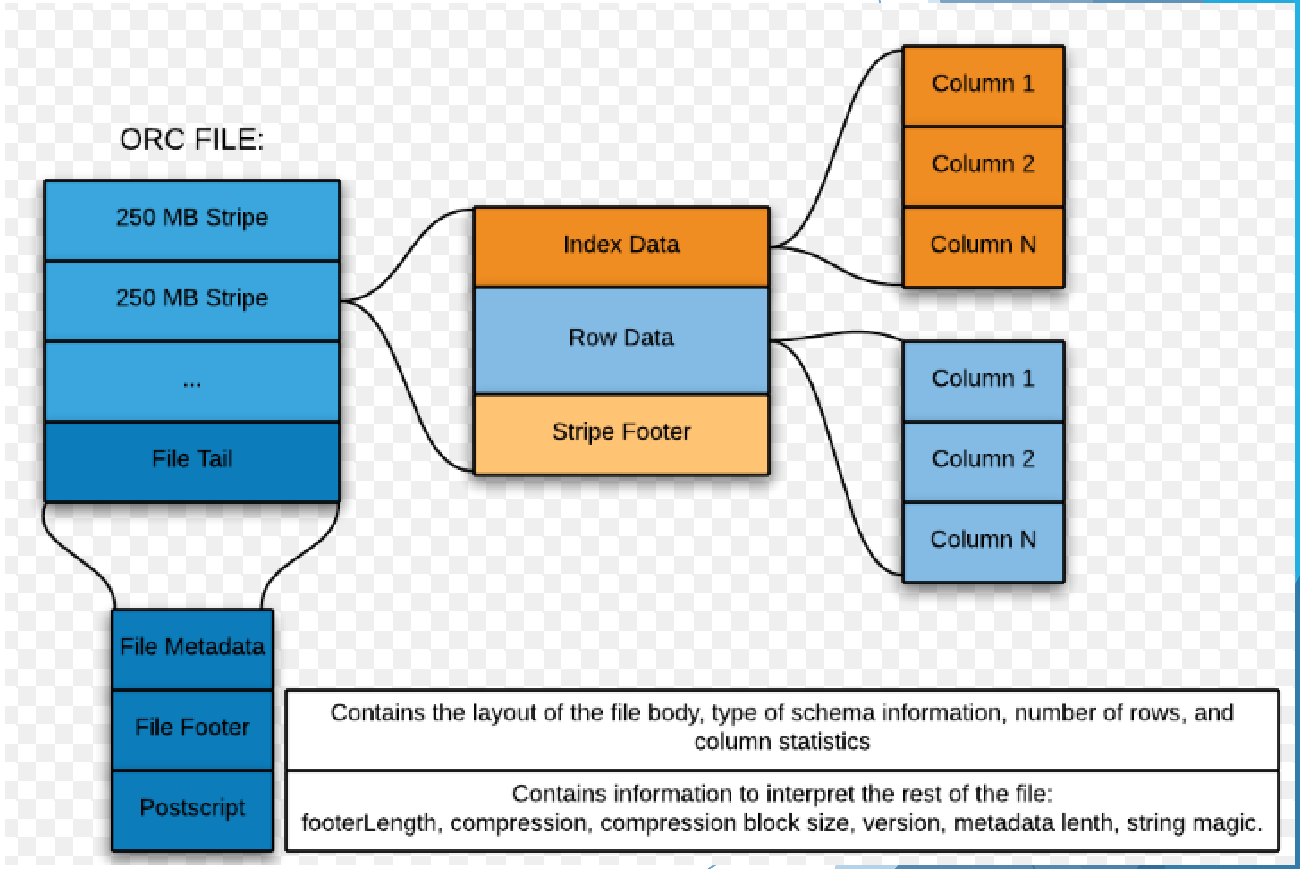
- Contains list of stripe locations
- Count, min, max and sum of each column

PostScript:

- Contains compression parameters

Stripe:

- It contains data, index and footer


ORC FILE:
250 MB Stripe
250 MB Stripe
...
File Tail
Index Data
Row Data
Stripe Footer
Column 1
Column 2
Column N
Column 1
Column 2
Column N
File Metadata
File Footer
Postscript
Contains the layout of the file body, type of schema information, number of rows, and column statistics
Contains information to interpret the rest of the file: footerLength, compression, compression block size, version, metadata lenth, string magic.

Hive Cost Based Optimization

- Hive optimizes each query’s logical and physical execution plan before submitting for final execution.
- CBO performs further optimizations based on query cost, resulting in potentially different decisions: how to order joins, which type of join to perform, degree of parallelism and others.
- To use cost-based optimization (also known as 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.
- Hive can store table and partition statistics in its metastore
- For example, in a table tweets we want to collect statistics about the table and about 2 columns: “sender” and “topic”:

```
analyze table tweets compute statistics;

analyze table tweets compute statistics for columns sender, topic;
```

- That’s it. Now executing a query using this table should result in a different execution plan that is faster because of the cost calculation and different execution plan created by Hive.

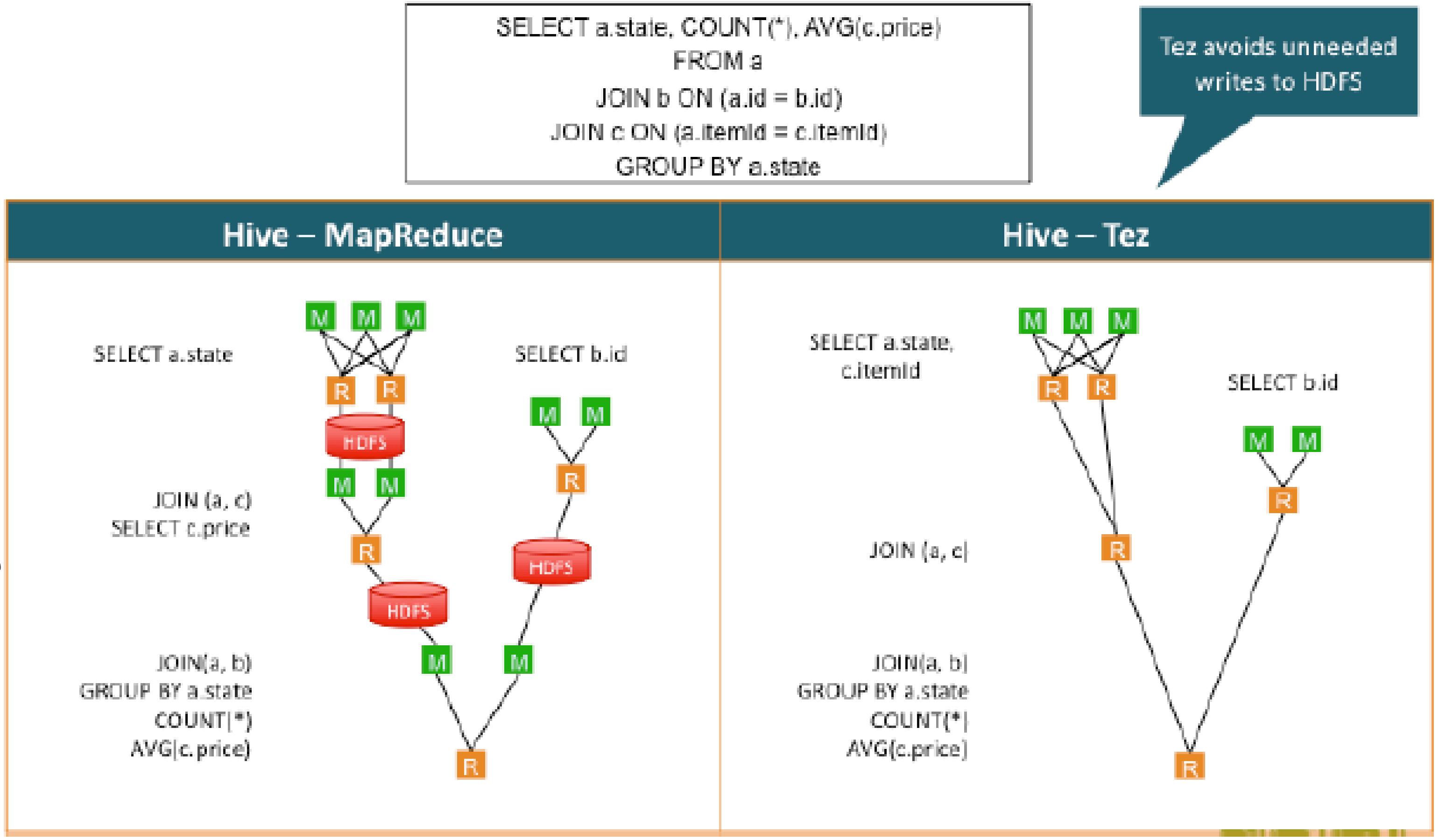
Vectorization

- Vectorized query execution improves performance of operations like scans, aggregations, filters and joins, by performing them in batches of 1024 rows at once instead of single row each time.
- Vectorization is a joint effort between Hortonworks and Microsoft.

- To take advantage of vectorization, your table needs to be in the ORC format and you need to enable vectorization with the following property:
- **set hive.vectorized.execution.enabled=true**
- **set hive.vectorized.execution.reduce.enabled = true**
- When vectorization is enabled, Hive examines the query and the data to determine whether vectorization can be supported.
- If it cannot be supported, Hive will execute the query with vectorization turned off.

Hive on Tez

- Tez is a A Framework for YARN-based, Data Processing Applications In Hadoop
- Hive query without Tez can consist of multiple MapReduce jobs.
- Tez performs a Hive query in a single job, avoiding the intermediate writes to disk that were a result of the multiple MapReduce jobs.
- To use Tez for a Hive query, you need to define the following property in your Hive script or in hive-site.xml:
- **set hive.execution.engine=tez;**
- With the above setting, every HIVE query you execute will take advantage of Tez.
- Note that this property is set to mr by default



Distribute By

- All rows with the same distribute by columns will go to the same reducer
- Distribute by is typically used in conjunction with an insert statement

```
salaries'
F              66     41000.0        95103
M              40     76000.0        95102
F              58     95000.0        95103
F              68     60000.0        95105
M              85     14000.0        95102
...
```

- The two records with age = 66 are sent to the same reducer, but they are not adjacent.
- You can use sort by to make the distributed by column appear together

`distribute by` on the `age` column:

```
set mapreduce.job.reduces=2;
insert overwrite table mytable
   select gender, age, salary from salaries
   distribute by age;
```

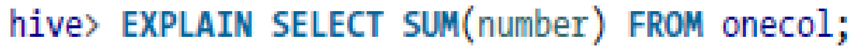
```
M,66,84000.0
F,58,95000.0
M,40,76000.0
F,66,41000.0
```

```
insert overwrite table mytable
   select gender, age, salary from salaries
   distribute by age
   sort by age;
```

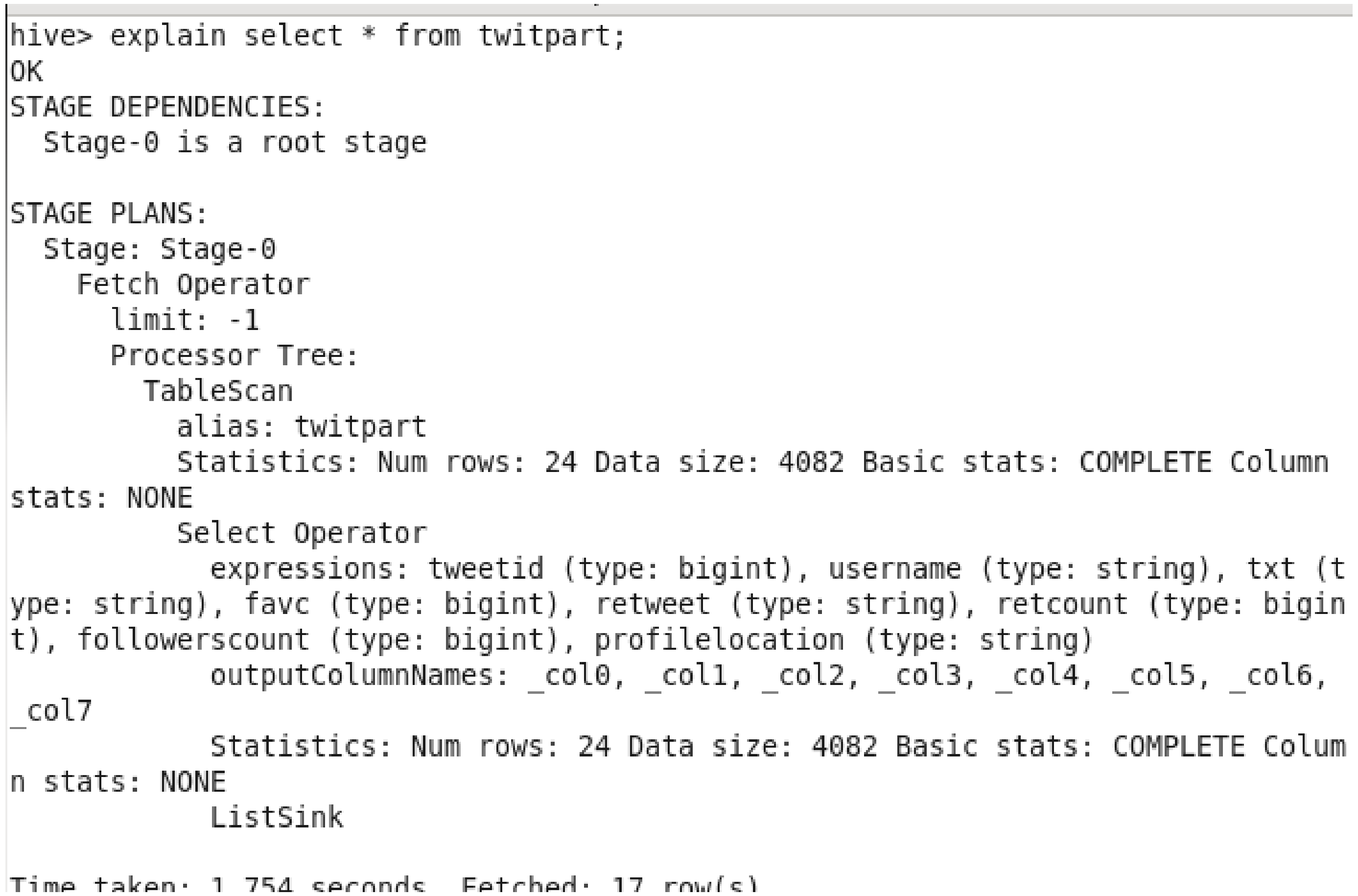
```
F,58,95000.0
M,66,84000.0
F,66,41000.0
M,68,15000.0
F,68,60000.0
M,72,83000.0
```

Explain

- EXPLAIN feature is used to learn how Hive translates queries into MapReduce jobs



```
hive> EXPLAIN SELECT SUM(number) FROM onecol;
```



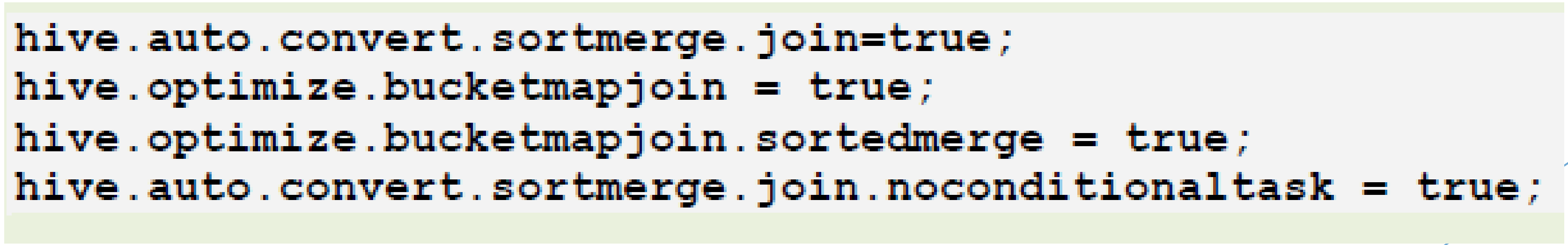
```
hive> explain select * from twitpart;
OK
STAGE DEPENDENCIES:
  Stage-0 is a root stage

STAGE PLANS:
  Stage: Stage-0
    Fetch Operator
      limit: -1
      Processor Tree:
        TableScan
          alias: twitpart
          Statistics: Num rows: 24 Data size: 4082 Basic stats: COMPLETE Column
stats: NONE
          Select Operator
            expressions: tweetid (type: bigint), username (type: string), txt (t
ype: string), favc (type: bigint), retweet (type: string), retcount (type: bigin
t), followerscount (type: bigint), profilelocation (type: string)
            outputColumnNames: _col0, _col1, _col2, _col3, _col4, _col5, _col6,
_col7
            Statistics: Num rows: 24 Data size: 4082 Basic stats: COMPLETE Colum
n stats: NONE
            ListSink

Time taken: 1.754 seconds, Fetched: 17 row(s)
```

Optimized Joins

- A shuffle join is the default join technique for Hive
- This is the most expensive join from a network utilization standpoint as all records from both sides of the join need to be processed by a mapper and then shuffled and sorted, even the records that are not a part of the result set.
- Map (Broadcast) Joins can be used to join a big table with a small table.
- Small table is stored in cache and is distributed (broadcast) to each mapper and then joined with the big table in the Map phase
- Loading a small table into cache will save read time on each data node
- **set hive.auto.convert.join = true**
- Then Hive will automatically use mapjoins for any tables smaller than hive.mapjoin.smalltable.filesize (default is 25MB)



```
hive.auto.convert.sortmerge.join=true;
hive.optimize.bucketmapjoin = true;
hive.optimize.bucketmapjoin.sortedmerge = true;
hive.auto.convert.sortmerge.join.noconditionaltask = true;
```

Smart Queries

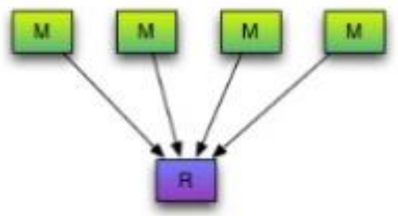
Avoid "SELECT count(DISTINCT field) FROM tbl"

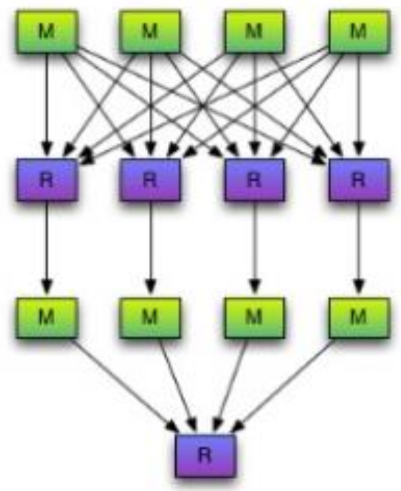
- This query does the count on the map side.
- Each mapper emits one value, the count
- Then all values have to be aggregated to produce the total count, and that is the job of one single reducer
- Rewrite the query as

SELECT count(1)

FROM (SELECT DISTINCT field FROM tbl) t

- This query has two stages and multiple reducers





Considering the Cardinality within GROUP BY

- There's a probability where GROUP BY becomes a little bit faster, by carefully ordering a list of fields within GROUP BY in an order of high cardinality.
- good: SELECT GROUP BY uid, gender
- bad: SELECT GROUP BY gender, uid

Hive Optimization Tips

1. Use ORC File format
2. Optimize Joins
   - Enable auto convert Map Joins and enable optimization of skew joins
   - Imbalance Join is called as skew joins. You can optimize it by setting

     set hive.optimize.skewjoin = true (in hive_site.xml)
3. Use map (broadcast) joins whenever possible
4. Avoid global sorting in Hive when possible
   - Order by produces result by setting no of reducers to 1 , making it inefficient for large data sets.
   - Use Sort by or distribute by
5. Enable Tez execution Engine
   - Performance can be improved 100% to 300% by running on Tez execution engine

Hive Optimization Tips…

6. Optimize Limit operator

- By default limit executes entire query and returns limited results
- set hive.limit.optimize.enable = true
- The below properties now control limit operation
- hive.limit.row.max.size (How much size we need to guarantee each row to have at least)
- hive.limit.optimize.limit.file (Maximum number of files we can sample)
- hive.limit.optimize.fetch.max (Maximum number of rows allowed for a smaller subset of data)

7. Enable Parallel Execution

- Independent tasks in a query can run in parallel.
- set hive.exec.parallel=true;

8. Enable MapReduce Strict Mode

- In strict mode, some risky queries are not allowed to run.
- Ex: Cartesian product, No partition being picked up for a query, Comparing bigints and strings, Comparing bigints and doubles, Orderby without limit

Hive Optimization Tips…

9. Enable Vectorization
10. Enable Cost Based Optimization
11. Enable Compression in Hive
12. Write Good Sql (Use Hive Analytical Functions When Needed)
13. Use Partitions
    - Be careful of Over Partitioning – avoid fragmenting the data too much.
14. Use Bucketing
    - Sort and Bucket on common join keys
15. Use Skewing

    Divide data amongst different files that can be pruned out by using partitions, buckets, and skews
16. Control the number of parallel reduce tasks to a fixed value
    - set mapred.reduce.tasks = Number

Hands On

Thank You

Keerthiga Barathan