NOTES ON APACHE HIVE OPTIMIZATION TECHNIQUES PART 02

linkedin.com/vaishnavi-muralidhar

QUERY LEVEL OPTIMIZATION:

Joins Optimization:

1. **Reduce no. of join operations**
2. **Use partitioning and bucketing to reduce no. of joins** (we can get value to be joined fastly if table is partitioned/bucketed)
3. **Covert joins to map side joins whenever possible**

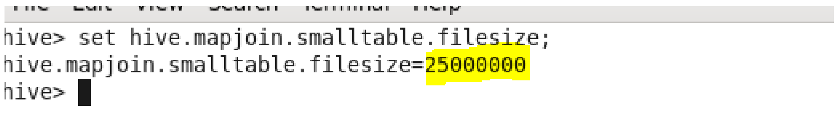
-->We will talk about "joins" as it required more processing and complex to perform due to which it needs to be optimized
-->No. of joins = No. of map-reduce jobs
-->So we have to minimize no. of joins which in turn reduce MR jobs which makes our query to run faster
-->Bucketing and Partitioning on joined column makes the join operation faster as we can look out in particular partitions & buckets for the joined column rather than scanning the entire dataset.

-->As we know mapper takes input in the form of (key, value)
Output from mapper:
key = join column
Value = all remaining columns
table1 - (id column_list1, id2 column_list2, id3 column_list3)
Mapper - (id3,column_list3)

-->Reducer combines all the columns having the same key
table1 - (id, column_list1)
table2- (prod_id, column_list2)
Reducer (id, column_list1, column_list2)

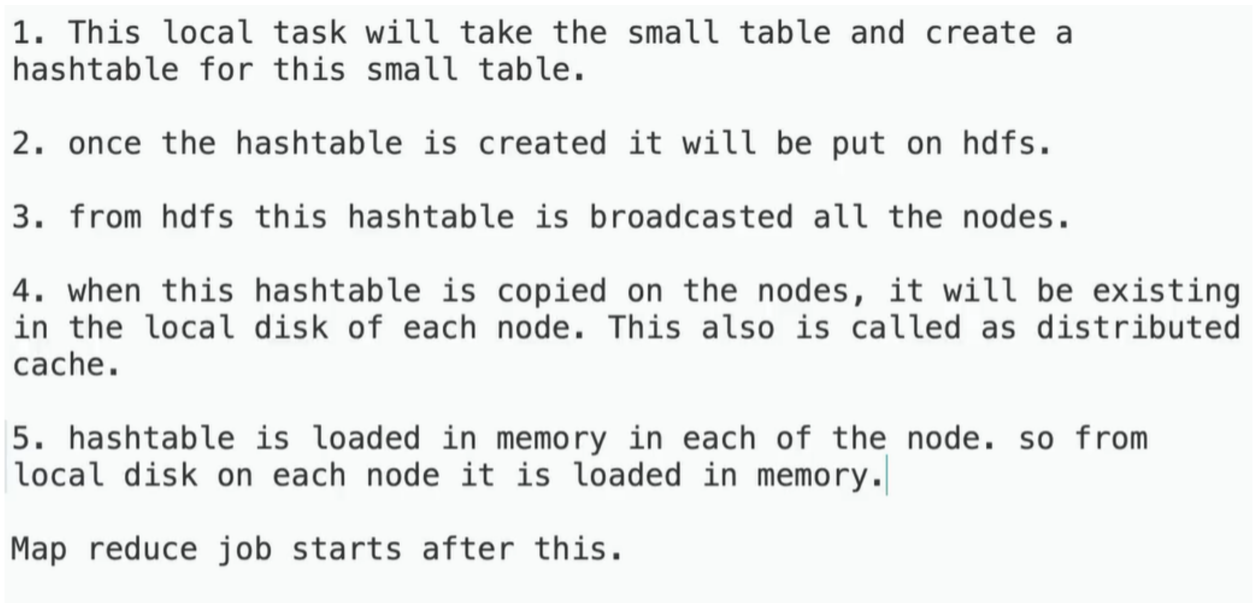
-->Reducers have more work to do as they have to aggregate huge no. of records (based on table size) and give the result set.
-->Due to this we have introduced a concept of MAP SIDE JOINS where mappers do all the work and give the output rather than sending it to reducers whenever it is feasible.

when we can consider a table to be small?
-->If the table size is <=25 mb then it is considered to be small

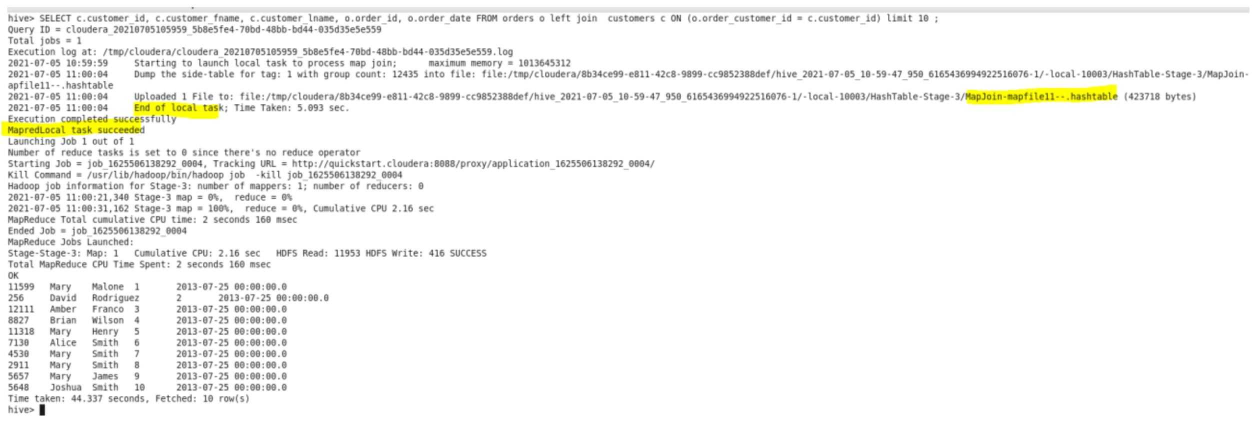


```
hive> set hive.mapjoin.smalltable.filesize;
hive.mapjoin.smalltable.filesize=25000000
hive> 
```

Map Side Join:
-->To join tables and give results by making use of only mappers and not involving reducers.
-->Before the map-reduce invokes, below local task happens in the background.
-->As we have to load the entire table/hash table into memory it needs to be small only then it can work well



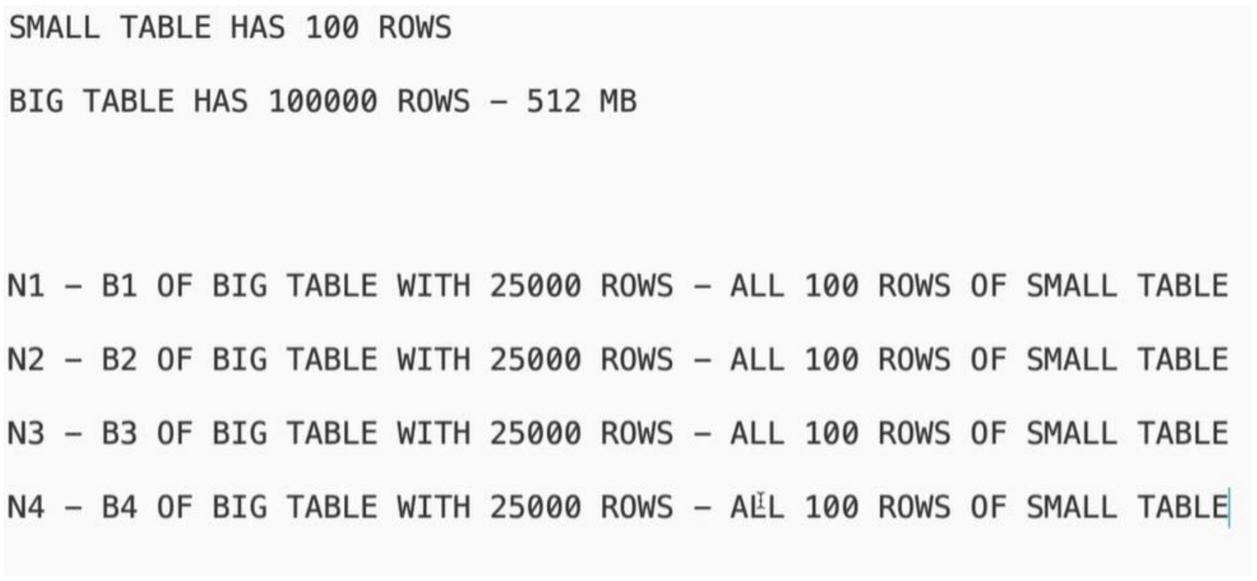
```
1. This local task will take the small table and create a
hashtable for this small table.

2. once the hashtable is created it will be put on hdfs.

3. from hdfs this hashtable is broadcasted all the nodes.

4. when this hashtable is copied on the nodes, it will be existing
in the local disk of each node. This also is called as distributed
cache.

5. hashtable is loaded in memory in each of the node. so from
local disk on each node it is loaded in memory.

Map reduce job starts after this.
```



```
hive> SELECT c.customer_id, c.customer_fname, c.customer_lname, o.order_id, o.order_date FROM orders o left join  customers c ON (o.order_customer_id = c.customer_id) limit 10 ;
Query ID = cloudera_20210705105959_5b8e5fe4-70bd-48bb-bd44-035d35e5e559
Total jobs = 1
Execution log at: /tmp/cloudera/cloudera_20210705105959_5b8e5fe4-70bd-48bb-bd44-035d35e5e559.log
2021-07-05 10:59:59     Starting to launch local task to process map join;      maximum memory = 1013645312
2021-07-05 11:00:04     Dump the side-table for tag: 1 with group count: 12435 into file: file:/tmp/cloudera/8b34ce99-e811-42c8-9899-cc9852388def/hive_2021-07-05_10-59-47_950_6165436994922516076-1/-local-10003/HashTable-Stage-3/MapJoin-
apfile11--.hashtable
2021-07-05 11:00:04     Uploaded 1 File to: file:/tmp/cloudera/8b34ce99-e811-42c8-9899-cc9852388def/hive_2021-07-05_10-59-47_950_6165436994922516076-1/-local-10003/HashTable-Stage-3/MapJoin-mapfile11--.hashtable (423718 bytes)
2021-07-05 11:00:04     End of local task; Time Taken: 5.093 sec.
Execution completed successfully
MapredLocal task succeeded
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_1625506138292_0004, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1625506138292_0004/
Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1625506138292_0004
Hadoop job information for Stage-3: number of mappers: 1; number of reducers: 0
2021-07-05 11:00:21,340 Stage-3 map = 0%,  reduce = 0%
2021-07-05 11:00:31,162 Stage-3 map = 100%,  reduce = 0%, Cumulative CPU 2.16 sec
MapReduce Total cumulative CPU time: 2 seconds 160 msec
Ended Job = job_1625506138292_0004
MapReduce Jobs Launched:
Stage-Stage-3: Map: 1   Cumulative CPU: 2.16 sec   HDFS Read: 11953 HDFS Write: 416 SUCCESS
Total MapReduce CPU Time Spent: 2 seconds 160 msec
OK
11599   Mary    Malone  1       2013-07-25 00:00:00.0
256     David   Rodriguez       2       2013-07-25 00:00:00.0
12111   Amber   Franco  3       2013-07-25 00:00:00.0
8827    Brian   Wilson  4       2013-07-25 00:00:00.0
11318   Mary    Henry   5       2013-07-25 00:00:00.0
7130    Alice   Smith   6       2013-07-25 00:00:00.0
4530    Mary    Smith   7       2013-07-25 00:00:00.0
2911    Mary    Smith   8       2013-07-25 00:00:00.0
5657    Mary    James   9       2013-07-25 00:00:00.0
5648    Joshua  Smith   10      2013-07-25 00:00:00.0
Time taken: 44.337 seconds, Fetched: 10 row(s)
hive> 
```

Conditions to use map-side joins:

4. we need to have only 1 BIG table and all other tables should be small.



```
SMALL TABLE HAS 100 ROWS

BIG TABLE HAS 100000 ROWS - 512 MB



N1 - B1 OF BIG TABLE WITH 25000 ROWS - ALL 100 ROWS OF SMALL TABLE

N2 - B2 OF BIG TABLE WITH 25000 ROWS - ALL 100 ROWS OF SMALL TABLE

N3 - B3 OF BIG TABLE WITH 25000 ROWS - ALL 100 ROWS OF SMALL TABLE

N4 - B4 OF BIG TABLE WITH 25000 ROWS - ALL 100 ROWS OF SMALL TABLE
```

TYPES OF JOIN:

5. Inner Join : To output the matching records from both tables
6. Left Outer Join: Matching records + all records from left table with null values on right
7. Right Outer Join: Matching records + all records from right table with null values on left
8. Full Outer Join: Left Outer Join + Right Outer Join

-->when we perform INNER JOIN keeping the small table to be on the Left that fits into memory then we can consider **Inner Join as Map side join** as we get the results directly from the mapper and no need to involve reducer.

-->Left Outer Join: If smaller table is on left, we get matching records but we can't display all records on left as they might haven't matched on 1 partition of node but they can be matched on the other node. so we are not sure whether all our small table records are present/not present based on a single partition. **Left join can't be MAP SIDE JOIN when left table is small**

-->Right Outer Join: If big table is on right, we get all matching records and all right table records with null values on left. Because every partition has the same left table data so we know if matching records are present or not in that partition. **Right Join can be MAP SIDE JOIN when right table is big**

-->Full Outer Join: Left outer join (NO) + Right outer join (YES) ==> NO so **full outer can't be MAP side join**

Steps to follow to run a map-side join:

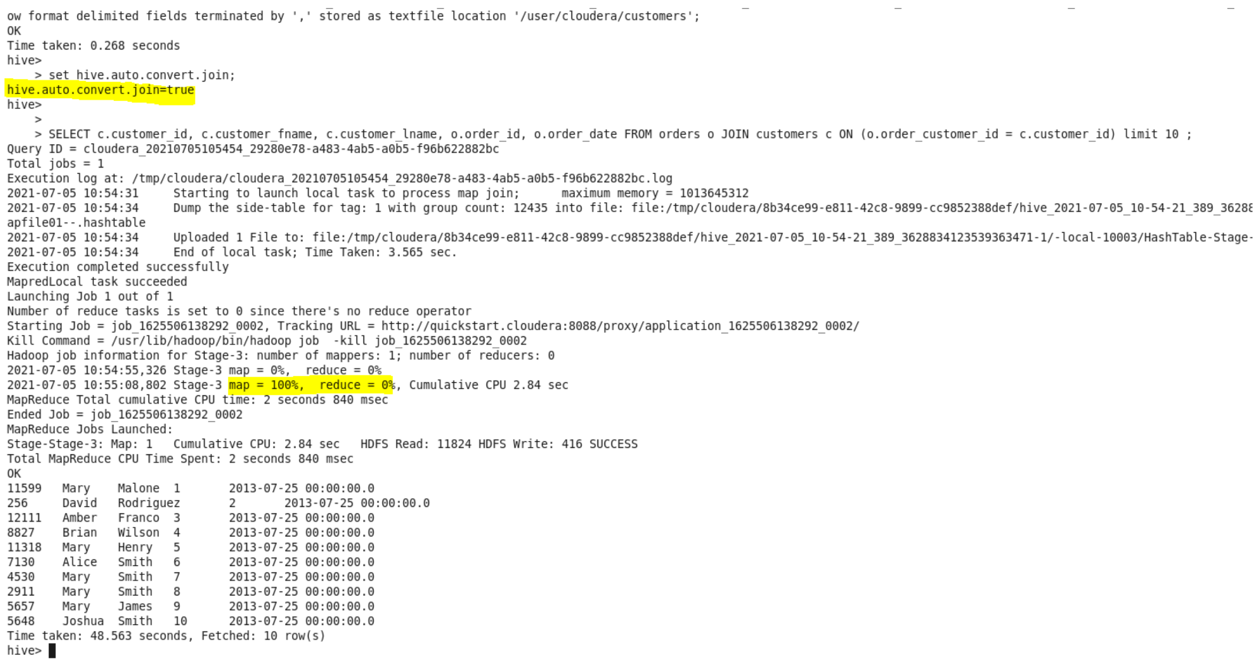
9. Using sqoop import, import tables to hdfs
10. create hive external tables and load data into it

CREATE **external** TABLE orders(order_id int,order_date string,order_customer_id int,order_status string) row format delimited fields terminated by ',' stored as textfile **location '/user/cloudera/orders';**

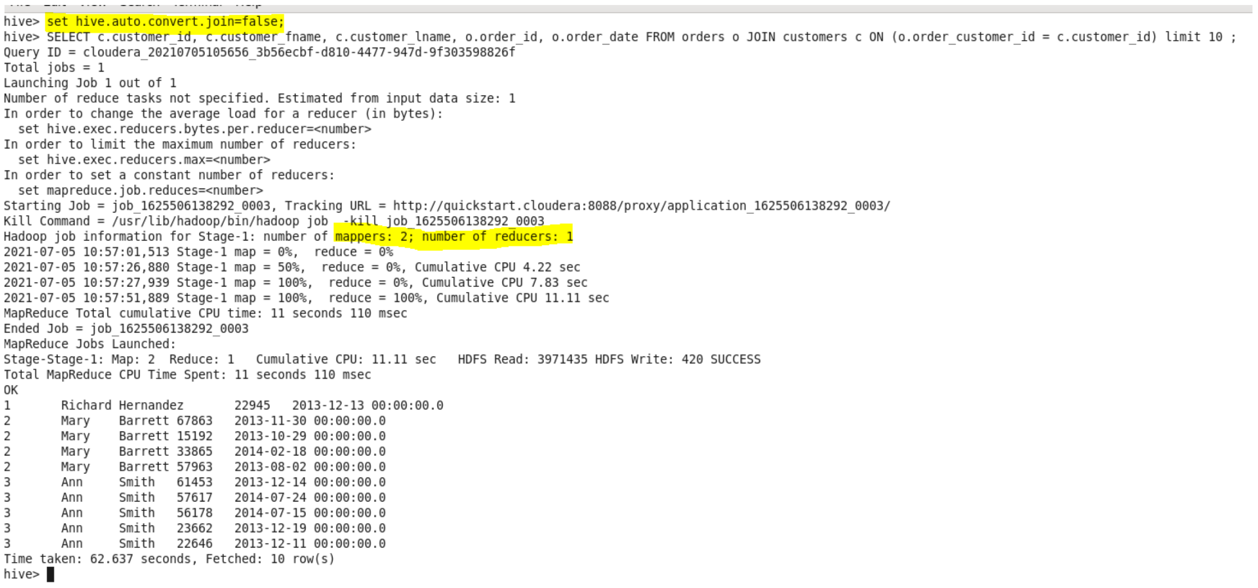
11. Perform a join operation (inner join) and observe it has invoke map side join without using reducers

12. Perform Left/ full outer join to see it has invoked reducers as well (small table on left)

-->The property 'set hive.auto.convert.join' by default is true tries to perform map-side join if the conditions were met. Purposely we set this property to "false" and observed that it has invoked reducer as well

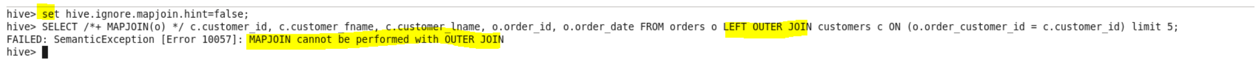


```
ow format delimited fields terminated by ',' stored as textfile location '/user/cloudera/customers';
OK
Time taken: 0.268 seconds
hive>
    > set hive.auto.convert.join;
hive.auto.convert.join=true
hive>
    >
    > SELECT c.customer_id, c.customer_fname, c.customer_lname, o.order_id, o.order_date FROM orders o JOIN customers c ON (o.order_customer_id = c.customer_id) limit 10 ;
Query ID = cloudera_20210705105454_29280e78-a483-4ab5-a0b5-f96b622882bc
Total jobs = 1
Execution log at: /tmp/cloudera/cloudera_20210705105454_29280e78-a483-4ab5-a0b5-f96b622882bc.log
2021-07-05 10:54:31     Starting to launch local task to process map join;      maximum memory = 1013645312
2021-07-05 10:54:34     Dump the side-table for tag: 1 with group count: 12435 into file: file:/tmp/cloudera/8b34ce99-e811-42c8-9899-cc9852388def/hive_2021-07-05_10-54-21_389_36288
apfile01--.hashtable
2021-07-05 10:54:34     Uploaded 1 File to: file:/tmp/cloudera/8b34ce99-e811-42c8-9899-cc9852388def/hive_2021-07-05_10-54-21_389_3628834123539363471-1/-local-10003/HashTable-Stage-
2021-07-05 10:54:34     End of local task; Time Taken: 3.565 sec.
Execution completed successfully
MapredLocal task succeeded
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_1625506138292_0002, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1625506138292_0002/
Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1625506138292_0002
Hadoop job information for Stage-3: number of mappers: 1; number of reducers: 0
2021-07-05 10:54:55,326 Stage-3 map = 0%,  reduce = 0%
2021-07-05 10:55:08,802 Stage-3 map = 100%,  reduce = 0%, Cumulative CPU 2.84 sec
MapReduce Total cumulative CPU time: 2 seconds 840 msec
Ended Job = job_1625506138292_0002
MapReduce Jobs Launched:
Stage-Stage-3: Map: 1   Cumulative CPU: 2.84 sec   HDFS Read: 11824 HDFS Write: 416 SUCCESS
Total MapReduce CPU Time Spent: 2 seconds 840 msec
OK
11599   Mary    Malone  1       2013-07-25 00:00:00.0
256     David   Rodriguez       2       2013-07-25 00:00:00.0
12111   Amber   Franco  3       2013-07-25 00:00:00.0
8827    Brian   Wilson  4       2013-07-25 00:00:00.0
11318   Mary    Henry   5       2013-07-25 00:00:00.0
7130    Alice   Smith   6       2013-07-25 00:00:00.0
4530    Mary    Smith   7       2013-07-25 00:00:00.0
2911    Mary    Smith   8       2013-07-25 00:00:00.0
5657    Mary    James   9       2013-07-25 00:00:00.0
5648    Joshua  Smith   10      2013-07-25 00:00:00.0
Time taken: 48.563 seconds, Fetched: 10 row(s)
hive>
```



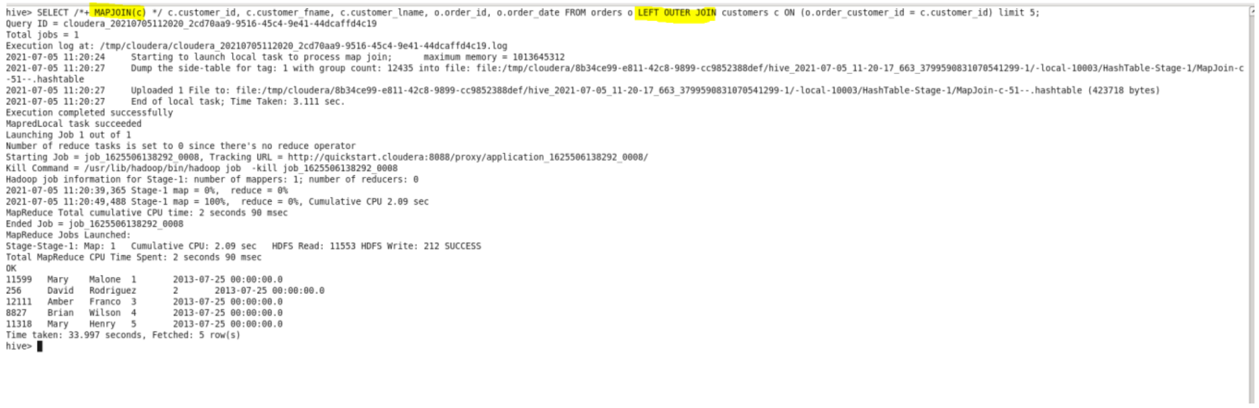
```
hive> set hive.auto.convert.join=false;
hive> SELECT c.customer_id, c.customer_fname, c.customer_lname, o.order_id, o.order_date FROM orders o JOIN customers c ON (o.order_customer_id = c.customer_id) limit 10 ;
Query ID = cloudera_20210705105656_3b56ecbf-d810-4477-947d-9f303598826f
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1625506138292_0003, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1625506138292_0003/
Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1625506138292_0003
Hadoop job information for Stage-1: number of mappers: 2; number of reducers: 1
2021-07-05 10:57:01,513 Stage-1 map = 0%,  reduce = 0%
2021-07-05 10:57:26,880 Stage-1 map = 50%,  reduce = 0%, Cumulative CPU 4.22 sec
2021-07-05 10:57:27,939 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 7.83 sec
2021-07-05 10:57:51,889 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 11.11 sec
MapReduce Total cumulative CPU time: 11 seconds 110 msec
Ended Job = job_1625506138292_0003
MapReduce Jobs Launched:
Stage-Stage-1: Map: 2  Reduce: 1   Cumulative CPU: 11.11 sec   HDFS Read: 3971435 HDFS Write: 420 SUCCESS
Total MapReduce CPU Time Spent: 11 seconds 110 msec
OK
1       Richard Hernandez       22945   2013-12-13 00:00:00.0
2       Mary    Barrett 67863   2013-11-30 00:00:00.0
2       Mary    Barrett 15192   2013-10-29 00:00:00.0
2       Mary    Barrett 33865   2014-02-18 00:00:00.0
2       Mary    Barrett 57963   2013-08-02 00:00:00.0
3       Ann     Smith   61453   2013-12-14 00:00:00.0
3       Ann     Smith   57617   2014-07-24 00:00:00.0
3       Ann     Smith   56178   2014-07-15 00:00:00.0
3       Ann     Smith   23662   2013-12-19 00:00:00.0
3       Ann     Smith   22646   2013-12-11 00:00:00.0
Time taken: 62.637 seconds, Fetched: 10 row(s)
hive>
```

-->The property "set hive.ignore.mapjoin.hint=false" is set to 'false' which means it can't ignore hints. we hinted the join to be "map-side join" **having a small table on the left** and to perform left join. we tried to perform left join, as seen above we can't perform map side join through LEFT join and it has resulted in error.



```
hive> set hive.ignore.mapjoin.hint=false;
hive> SELECT /*+ MAPJOIN(o) */ c.customer_id, c.customer_fname, c.customer_lname, o.order_id, o.order_date FROM orders o LEFT OUTER JOIN customers c ON (o.order_customer_id = c.customer_id) limit 5;
FAILED: SemanticException [Error 10057]: MAPJOIN cannot be performed with OUTER JOIN
hive> 
```

-->Left join when large table is on left. It works now. Refer above for explanation. The hint /*MAPJOIN(c) suggests customers table("c" ie large table) is on left



```
hive> SELECT /*+ MAPJOIN(c) */ c.customer_id, c.customer_fname, c.customer_lname, o.order_id, o.order_date FROM orders o LEFT OUTER JOIN customers c ON (o.order_customer_id = c.customer_id) limit 5;
Query ID = cloudera_20210705112020_2cd70aa9-9516-45c4-9e41-44dcaffd4c19
Total jobs = 1
Execution log at: /tmp/cloudera/cloudera_20210705112020_2cd70aa9-9516-45c4-9e41-44dcaffd4c19.log
2021-07-05 11:20:24     Starting to launch local task to process map join;      maximum memory = 1013645312
2021-07-05 11:20:27     Dump the side-table for tag: 1 with group count: 12435 into file: file:/tmp/cloudera/8b34ce99-e811-42c8-9899-cc9852388def/hive_2021-07-05_11-20-17_663_3799590831070541299-1/-local-10003/HashTable-Stage-1/MapJoin-c
-51--.hashtable
2021-07-05 11:20:27     Uploaded 1 File to: file:/tmp/cloudera/8b34ce99-e811-42c8-9899-cc9852388def/hive_2021-07-05_11-20-17_663_3799590831070541299-1/-local-10003/HashTable-Stage-1/MapJoin-c-51--.hashtable (423718 bytes)
2021-07-05 11:20:27     End of local task; Time Taken: 3.111 sec.
Execution completed successfully
MapredLocal task succeeded
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_1625506138292_0008, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1625506138292_0008/
Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1625506138292_0008
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 0
2021-07-05 11:20:39,365 Stage-1 map = 0%,  reduce = 0%
2021-07-05 11:20:49,488 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.09 sec
MapReduce Total cumulative CPU time: 2 seconds 90 msec
Ended Job = job_1625506138292_0008
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1   Cumulative CPU: 2.09 sec   HDFS Read: 11553 HDFS Write: 212 SUCCESS
Total MapReduce CPU Time Spent: 2 seconds 90 msec
OK
11599   Mary    Malone  1       2013-07-25 00:00:00.0
256     David   Rodriguez       2       2013-07-25 00:00:00.0
12111   Amber   Franco  3       2013-07-25 00:00:00.0
8827    Brian   Wilson  4       2013-07-25 00:00:00.0
11318   Mary    Henry   5       2013-07-25 00:00:00.0
Time taken: 33.997 seconds, Fetched: 5 row(s)
hive> 
```

Bucket Map Join & Sort Merge Bucket Join:

Bucket Map Join:
-->We can used this join when we have **2 big tables and all others are small tables**.

Conditions to use bucket map join:

13. Both the tables join columns should be bucketed

14. Buckets on one table should be integral multiple of buckets on the other

Table 1 -->500 values Table2--> 500,(2*500),(3*500) etc
ie X ==>(n)X

-->here we have divided id/4 (table1) and id/8 (table2), remainders were given below
-->As we can see **if we have id=4 we have look for it in bucket0 (table1) and bucket4 (table2)**

TABLE 01:
0 - 4,8,12,16,20
1 – 1,5,12,13,17
2 – 2,6,10,14
3 - 3,7,11,9,19

TABLE 02:
0– 8,16,24,32
1- 1, 9, 17, 25
2-2,10,14,18
3-3, 11, 19,27

-->if buckets are not integral multiples, then we can't predict specifically in which buckets we have to check for
-->id=3 bucket1(table1) and bucket0 (table2)

-->The main difference between map-side and bucket map join is that once the hash tables are loaded into all the nodes, **only 1 bucket at a time is loaded into memory as we are dealing with 2 big tables.**

Steps to follow to perform bucket map join:

15. Bucket the tables on join column
16. set the bucketing property in hive to true (set hive.enforce.bucketing=true, set hive.optimize.bucketmapjoin=true)

Sort Merge Bucket Join:

Conditions to use it:

1. Both the tables join columns should be bucketed
2. Buckets on one table should be exactly same as buckets on the other table
3. Both the tables should be sorted on bucket column

-->Here 1-to-1 mapping of buckets happen as both tables have same no. of buckets and they are even sorted.
-->mapper 1 works on bucket 0 of table1 and bucket0 of table2

If we want to join 2 tables and both of them were small?
-->We can then use traditional databases like mysql, oracle to do the task
-->As we are dealing with large tables where data is huge and distributed among various systems we go with "hive"
-->This is one of the main purpose of hive , to query on huge amount of data

To view what kind of join operation is performed in hive?

EXPLAIN EXTENDED SELECT c.customer_id, c.customer_fname, c.customer_lname, o.order_id, o.order_date FROM customers_bucketed c JOIN orders_bucketed o ON (c.customer_id = o.order_customer_id) limit 10;

EXPLAIN EXTENDED (QUERY)

Simple queries to be written:(WINDOW FUNCTIONS)

-->We make use of "WINDOW functions" to write simpler queries.
-->Window functions are used to analyze the data mainly
-->They are used to reduce the load on intermediate tables to store temporary data

https://www.youtube.com/watch?v=TzsrO4zTQj8
Window functions operate on a set of rows and return a single value for each row from the underlying query.
The term **window** describes the set of rows on which the **function** operates

OVER CLAUSE:
-->OVER clause is used in partitioning or ordering of rows before the window function is applied onto it. It defines the Window or user defined rows onto which window function are applied and computes the value for each row in the window.

Different categories of window functions:

17. Aggregate functions: SUM, AVERAGE,COUNT,MIN,MAX
18. Range functions: RANK, ROW_NUMBER
19. Analytical functions: LAG, LEAD,FIRST_VALUE

ORDER BY: Defines the logical ordering of rows (asc or desc)
PARTITION BY: Used to partition the data rows based on condition. Window function is applied to each partition separately

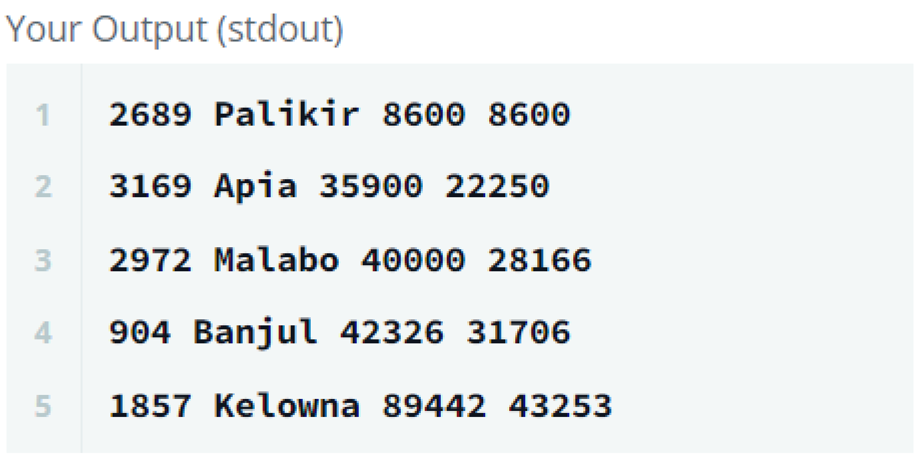
RANGE: Further limits the rows in the partition by specifying start and end points

By default ,RANGE BETWEEN UNBOUNDED PRECEDING to CURRENT ROW

Practiced in : https://www.hackerrank.com/challenges/average-population/problem

Case 1: To compute average population

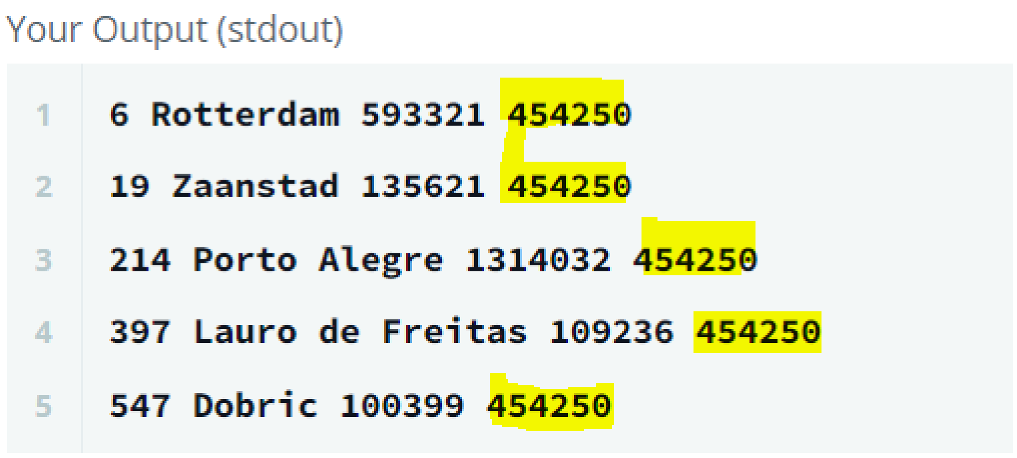
select top 5 id,name,population,
avg(population) over (order by population)//no range is mentioned
from city


Your Output (stdout)

```
2689 Palikir 8600 8600
3169 Apia 35900 22250
2972 Malabo 40000 28166
904 Banjul 42326 31706
1857 Kelowna 89442 43253
```

-->As range is not explicitly mentioned it takes from starting row to current row (observe last column)
Ie at each row average is computed from starting to current row

Case 2: Average population using RANGE


Your Output (stdout)

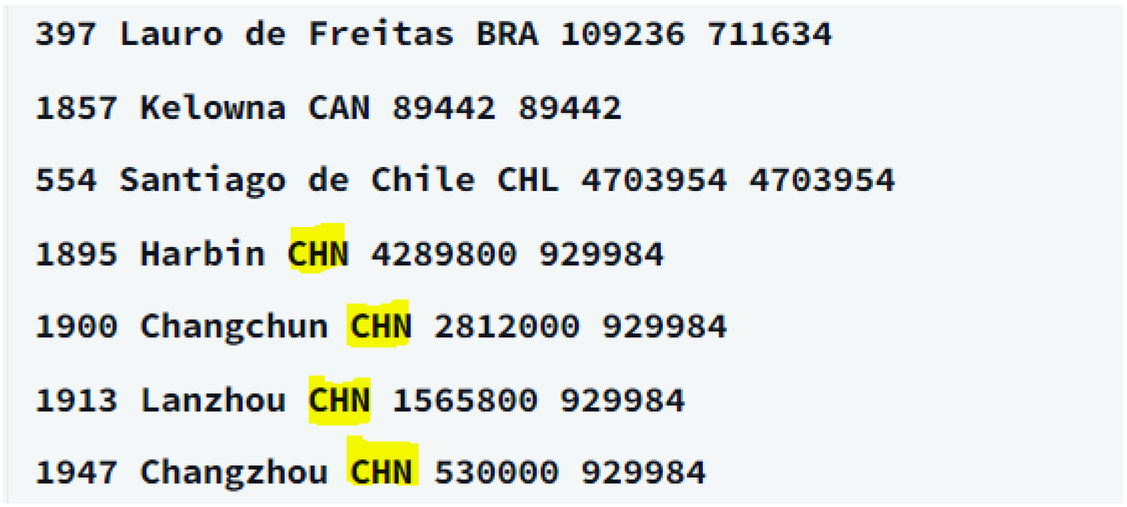
```
6 Rotterdam 593321 454250
19 Zaanstad 135621 454250
214 Porto Alegre 1314032 454250
397 Lauro de Freitas 109236 454250
547 Dobric 100399 454250
```

-->here we have mentioned the range and each row has average computed from starting to ending row

select top 5 id,name,population,
avg(population) over (order by population **range between unbounded preceding and unbounded following**)
from city

Case 3: Partition based on country code
-->Here the data is divided into partitions based on country code onto which average function is applied
Each partition has one average value computed

select top 50 id,name,countrycode,population,
avg(population) over (**partition by countrycode**)
from city
**Note: If ordered by is not mentioned then range is taken between 'unbounded preceding' and 'unbounded following'



```
397 Lauro de Freitas BRA 109236 711634

1857 Kelowna CAN 89442 89442

554 Santiago de Chile CHL 4703954 4703954

1895 Harbin CHN 4289800 929984

1900 Changchun CHN 2812000 929984

1913 Lanzhou CHN 1565800 929984

1947 Changzhou CHN 530000 929984
```

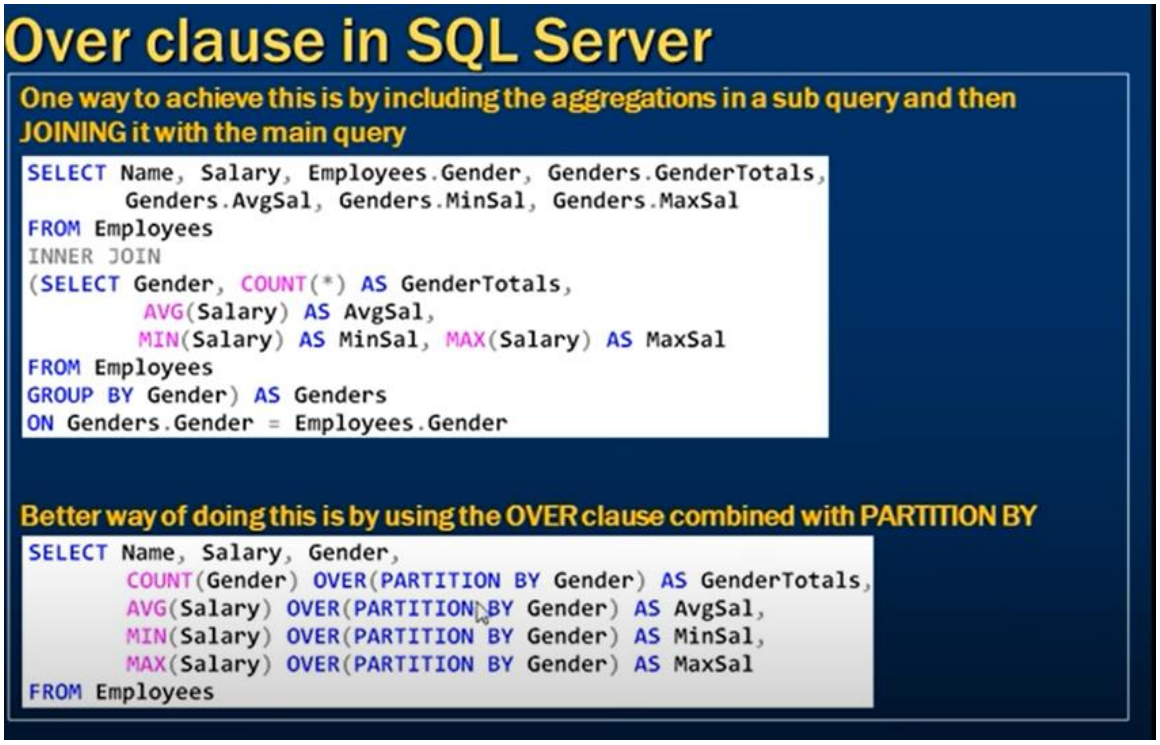
Case 4: when average is computed for a row preceding and following
-->here for each row average computed is (row before+ current row+ row after)

select top 50 id,name,countrycode,population,
avg(population) over (order by population **range between 1 preceding and 1 following**)
from city
Other ranges: 3 preceding and current row
--> 1 preceding and unbounded following

Hacker rank doesn't support this operation so couldn't get result set

OVER CLAUSE PRACTICAL EXAMPLE:
https://www.youtube.com/watch?v=KwEjkpFltjc

To combine the aggregate values (sum, count, average) with non aggregate values(employee name, employee id) we can use OVER Clause


Over clause in SQL Server

One way to achieve this is by including the aggregations in a sub query and then JOINING it with the main query

```
SELECT Name, Salary, Employees.Gender, Genders.GenderTotals,
       Genders.AvgSal, Genders.MinSal, Genders.MaxSal
FROM Employees
INNER JOIN
(SELECT Gender, COUNT(*) AS GenderTotals,
        AVG(Salary) AS AvgSal,
        MIN(Salary) AS MinSal, MAX(Salary) AS MaxSal
FROM Employees
GROUP BY Gender) AS Genders
ON Genders.Gender = Employees.Gender
```

Better way of doing this is by using the OVER clause combined with PARTITION BY

```
SELECT Name, Salary, Gender,
       COUNT(Gender) OVER(PARTITION BY Gender) AS GenderTotals,
       AVG(Salary) OVER(PARTITION BY Gender) AS AvgSal,
       MIN(Salary) OVER(PARTITION BY Gender) AS MinSal,
       MAX(Salary) OVER(PARTITION BY Gender) AS MaxSal
FROM Employees
```

How to find duplicate data?
ROW NUMBER FUNCTION:
https://www.youtube.com/watch?v=cvrwOoGwgz8

-->It is used to return the row number of the records we have in the table
-->It needs to be combined with ORDER BY CLAUSE
-->when PARTITION BY clause is used, row number changes to 1 with each new partition

select top 50 id,name,countrycode,population,
row_number() over (partition by countrycode order by countrycode)
from city

Use case: It is used to remove the duplicates based on row_number

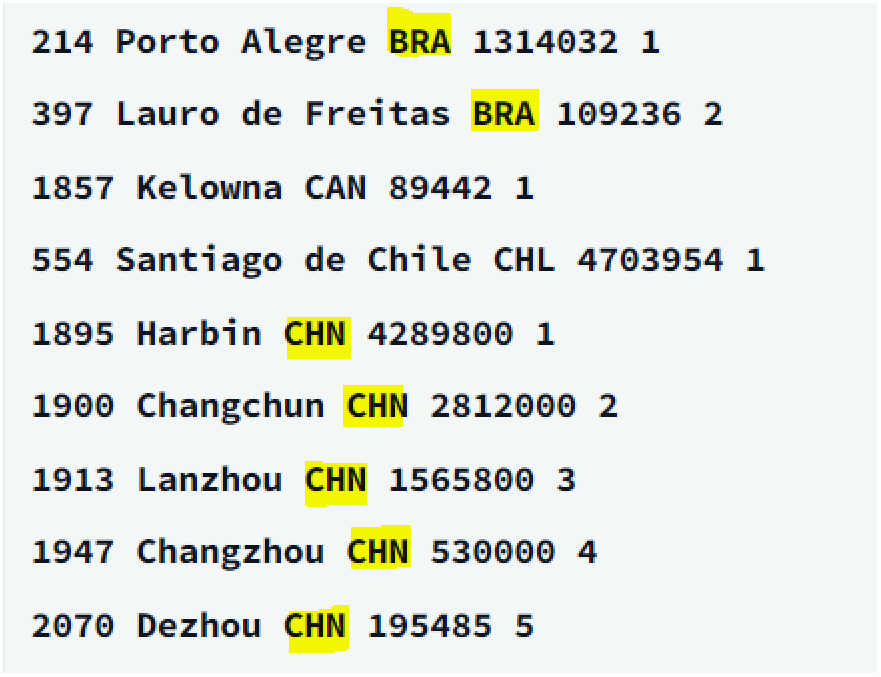
with cityRow as (this is an example of CTE)
(

```
select top 50 id,name,countrycode,population,
row_number() over (partition by countrycode order by countrycode ) as
rowNumber
from city
   )
   delete from cityRow where rowNumber > 1
```

CTE (common table expression)

-->It is a temporary named result set which is used to store the query result set temporarily

-->so all the rows whose count is greater than 1 is considered as duplicate for country code and is deleted



```
214 Porto Alegre BRA 1314032 1

397 Lauro de Freitas BRA 109236 2

1857 Kelowna CAN 89442 1

554 Santiago de Chile CHL 4703954 1

1895 Harbin CHN 4289800 1

1900 Changchun CHN 2812000 2

1913 Lanzhou CHN 1565800 3

1947 Changzhou CHN 530000 4

2070 Dezhou CHN 195485 5
```

RANKING FUNCTIONS:
How to rank the data?

RANK() AND DENSE_RANK() FUNCTIONS

-->It is used to rank the data records based on the condition given.

-->order by clause is required

Rank():
-->it skips the rankings if there are records with same rank

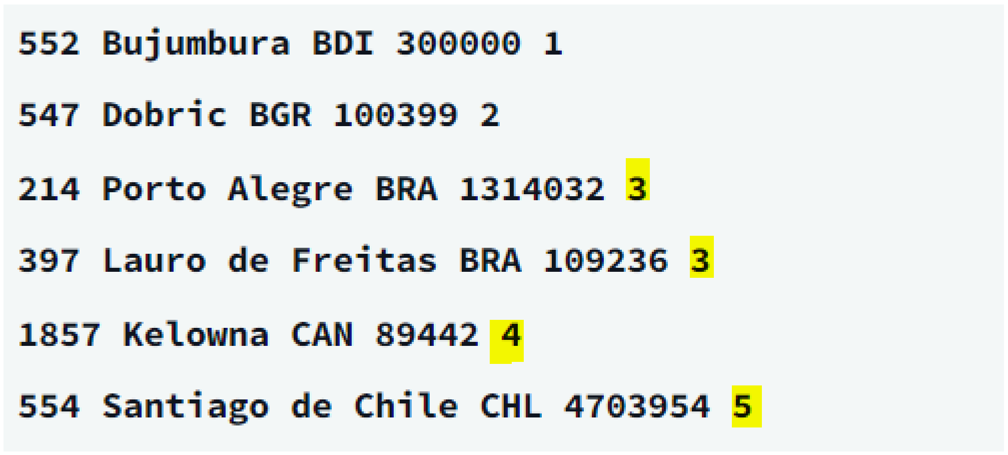
select top 50 id,name,countrycode,population,
rank() over (order by countrycode) as rowNumber
from city

Rank 4 is skipped because we have 2 records for 3

```
552 Bujumbura BDI 300000 1

547 Dobric BGR 100399 2

214 Porto Alegre BRA 1314032 3

397 Lauro de Freitas BRA 109236 3

1857 Kelowna CAN 89442 5
```

Dense_rank():
-->It doesn't skip any rank

select top 50 id,name,countrycode,population,
dense_rank() over (order by countrycode) as rowNumber
from city



```
552 Bujumbura BDI 300000 1

547 Dobric BGR 100399 2

214 Porto Alegre BRA 1314032 3

397 Lauro de Freitas BRA 109236 3

1857 Kelowna CAN 89442 4

554 Santiago de Chile CHL 4703954 5
```

SORTING:

what are the ways in which data can be sorted?

-->There are mainly 3 ways of sorting the data in hive namely:

20. Order By:

-->It provides the sorted list on the entire table data. By default in descending order.

-->It makes use of only one reducer so the entire sorting done by it.

-->**It is not advisable to order data on big tables as entire load falls on 1 reducer**

-->Ordering of null values is NULL when sorted in ascending order

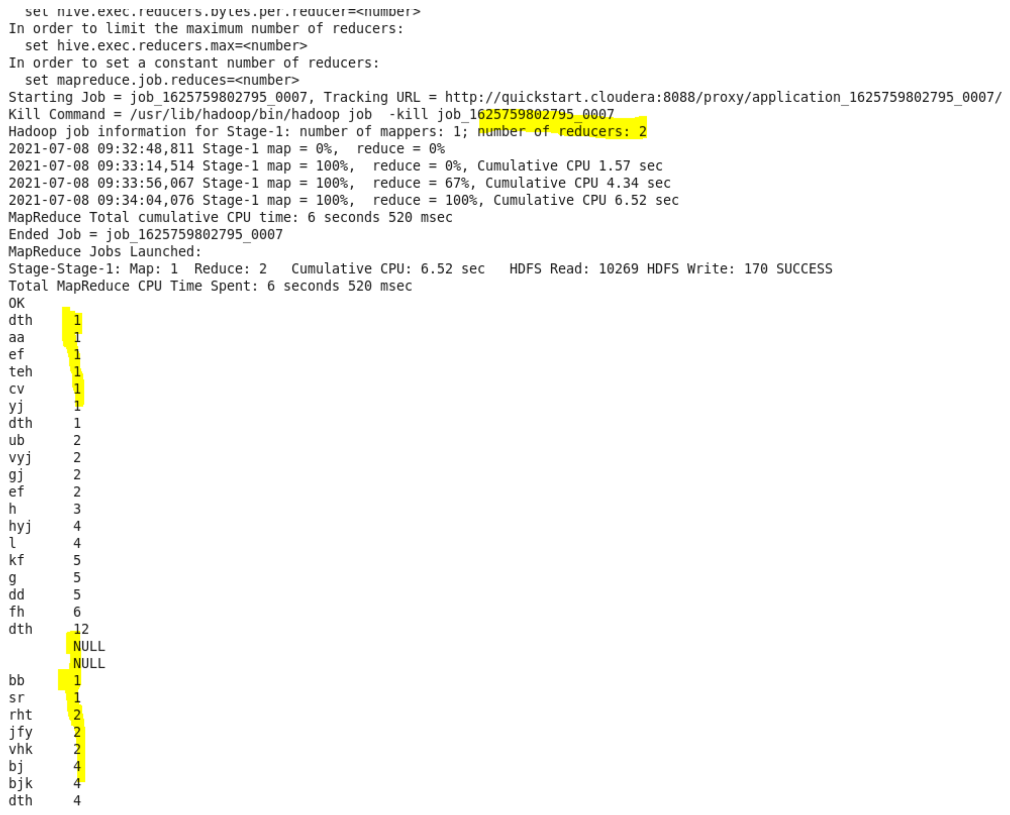
21. Sort By:

-->It uses multiple reducers(more than one)

-->No. of reducers= No. of sorted lists (ie if we have 3 reducers then we get 3 sorted lists on the same data)

-->By default 1 reducer can work on ~250mb ie if our table is of size 400mb then only 2 reducers will be invoked

reducer=2 , so 2 sets of data in sorted o



```
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1625759802795_0007, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1625759802795_0007/
Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1625759802795_0007
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 2
2021-07-08 09:32:48,811 Stage-1 map = 0%,  reduce = 0%
2021-07-08 09:33:14,514 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.57 sec
2021-07-08 09:33:56,067 Stage-1 map = 100%,  reduce = 67%, Cumulative CPU 4.34 sec
2021-07-08 09:34:04,076 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 6.52 sec
MapReduce Total cumulative CPU time: 6 seconds 520 msec
Ended Job = job_1625759802795_0007
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 2   Cumulative CPU: 6.52 sec   HDFS Read: 10269 HDFS Write: 170 SUCCESS
Total MapReduce CPU Time Spent: 6 seconds 520 msec
OK
dth     1
aa      1
ef      1
teh     1
cv      1
yj      1
dth     1
ub      2
vyj     2
gj      2
ef      2
h       3
hyj     4
l       4
kf      5
g       5
dd      5
fh      6
dth     12
        NULL
        NULL
bb      1
sr      1
rht     2
jfy     2
vhk     2
bj      4
bjk     4
dth     4
```

22. Distribute By:

-->It uses multiple reducers to distribute the data

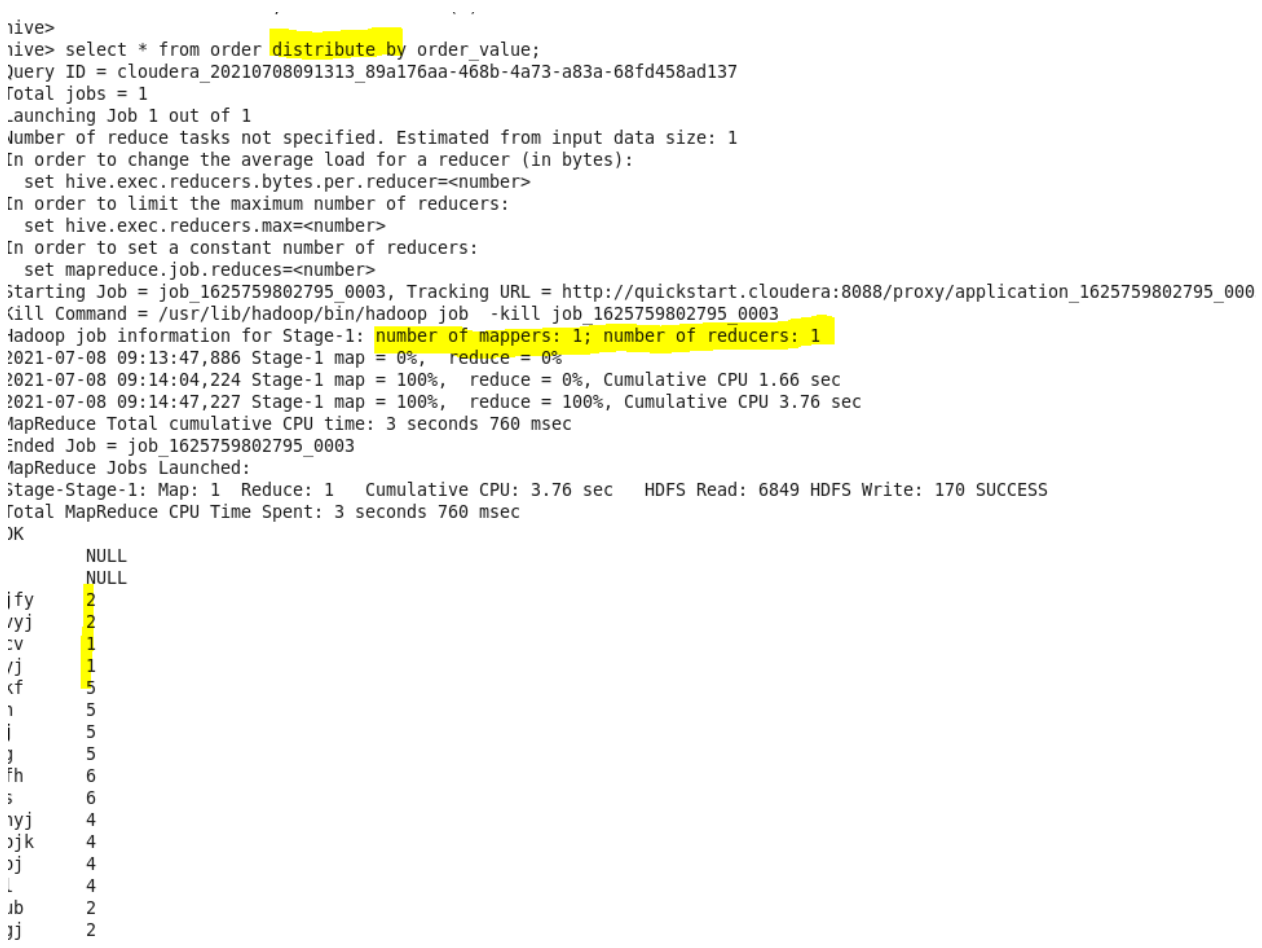
-->It makes sure there is no overlapping of data (ie same data is not distributed to to other reducers)

-->If we have value of 10 in reducer1 it will not be present in any other reducers.

-->It uses hash function to distribute data values among reducers

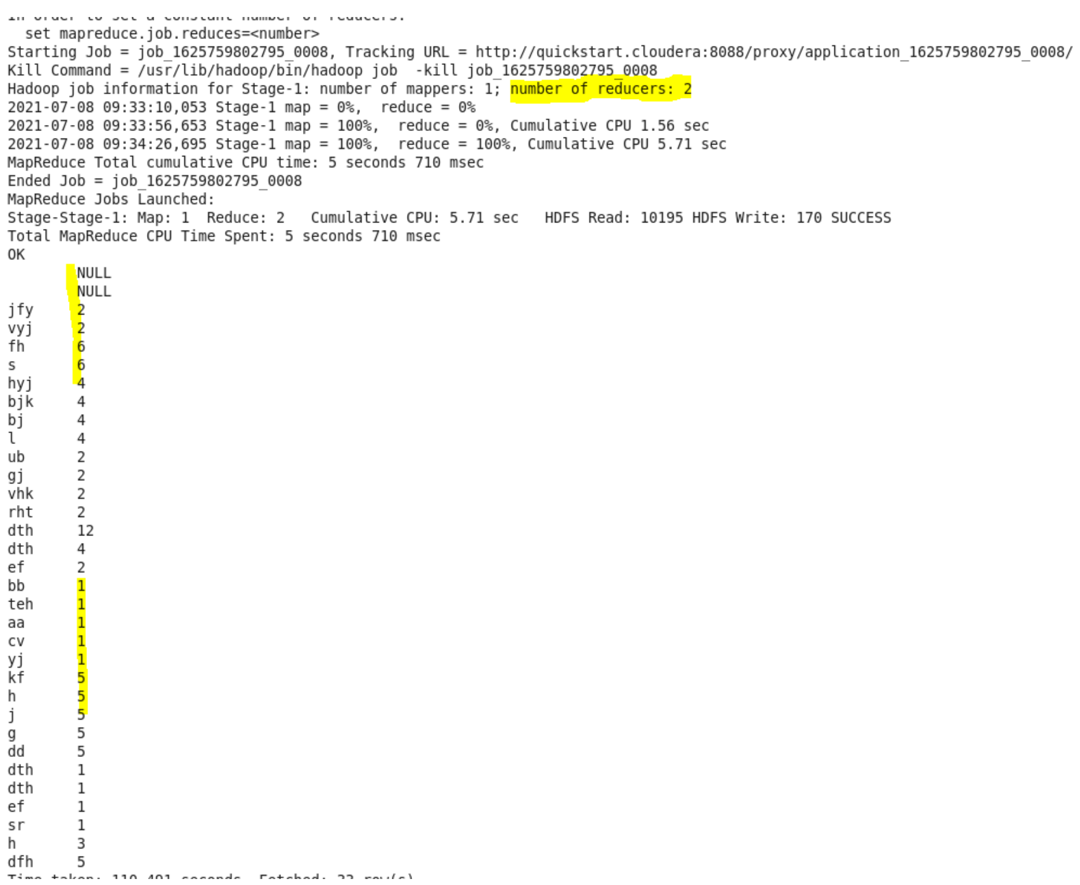
-->**It doesn't sort the data it just distributes the data**

Reducer=1 so data is distributed randomly



```
ive>
ive> select * from order distribute by order_value;
uery ID = cloudera_20210708091313_89a176aa-468b-4a73-a83a-68fd458ad137
otal jobs = 1
aunching Job 1 out of 1
umber of reduce tasks not specified. Estimated from input data size: 1
n order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
n order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
n order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
tarting Job = job_1625759802795_0003, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1625759802795_000
ill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1625759802795_0003
adoop job information for Stage-1: number of mappers: 1; number of reducers: 1
021-07-08 09:13:47,886 Stage-1 map = 0%,  reduce = 0%
021-07-08 09:14:04,224 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.66 sec
021-07-08 09:14:47,227 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 3.76 sec
apReduce Total cumulative CPU time: 3 seconds 760 msec
nded Job = job_1625759802795_0003
apReduce Jobs Launched:
tage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 3.76 sec   HDFS Read: 6849 HDFS Write: 170 SUCCESS
otal MapReduce CPU Time Spent: 3 seconds 760 msec
K
        NULL
        NULL
jfy     2
yj      2
v       1
j       1
f       5
        5
        5
        5
h       6
        6
yj      4
jk      4
j       4
        4
b       2
j       2
```

reducer =2 , so 2 reducers distributed data based on 2 hash functions into 2 parts

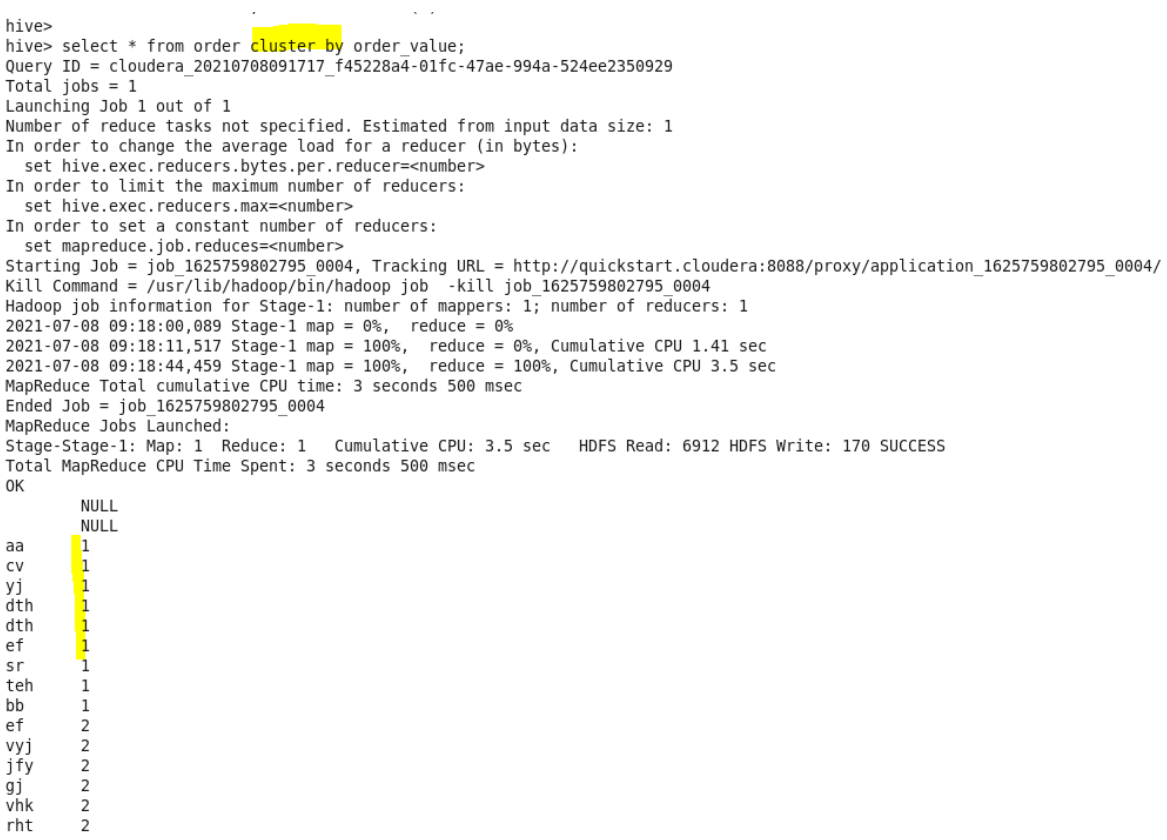


```
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1625759802795_0008, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1625759802795_0008/
Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1625759802795_0008
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 2
2021-07-08 09:33:10,053 Stage-1 map = 0%,  reduce = 0%
2021-07-08 09:33:56,653 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.56 sec
2021-07-08 09:34:26,695 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 5.71 sec
MapReduce Total cumulative CPU time: 5 seconds 710 msec
Ended Job = job_1625759802795_0008
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 2   Cumulative CPU: 5.71 sec   HDFS Read: 10195 HDFS Write: 170 SUCCESS
Total MapReduce CPU Time Spent: 5 seconds 710 msec
OK
        NULL
        NULL
jfy     2
vyj     2
fh      6
s       6
hyj     4
bjk     4
bj      4
l       4
ub      2
gj      2
vhk     2
rht     2
dth     12
dth     4
ef      2
bb      1
teh     1
aa      1
cv      1
yj      1
kf      5
h       5
j       5
g       5
dd      5
dth     1
dth     1
ef      1
sr      1
h       3
dfh     5
Time taken: 110.401 seconds, Fetched: 33 row(s)
```

23.   Cluster By:

-->It is the shortcut for (distribute by + sort by)

-->ie it distributes the data and then sorts it out


```
hive>
hive> select * from order cluster by order_value;
Query ID = cloudera_20210708091717_f45228a4-01fc-47ae-994a-524ee2350929
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1625759802795_0004, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1625759802795_0004/
Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1625759802795_0004
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2021-07-08 09:18:00,089 Stage-1 map = 0%,  reduce = 0%
2021-07-08 09:18:11,517 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.41 sec
2021-07-08 09:18:44,459 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 3.5 sec
MapReduce Total cumulative CPU time: 3 seconds 500 msec
Ended Job = job_1625759802795_0004
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 3.5 sec   HDFS Read: 6912 HDFS Write: 170 SUCCESS
Total MapReduce CPU Time Spent: 3 seconds 500 msec
OK
        NULL
        NULL
aa      1
cv      1
yj      1
dth     1
dth     1
ef      1
sr      1
teh     1
bb      1
ef      2
vyj     2
jfy     2
gj      2
vhk     2
rht     2
```

**

-->By default reducer=1

set mapred.reduce.tasks ~ -1

set mapred.reduce.tasks=3 //we can change no. of reducers

-->For sort by, distribute by, cluster by no. of reducers get auto increased if data > 25mb

set hive.exec.reducers.bytes.per.reducer ~ 25mb

Else only 1 reducer works on them by default

ANALYTICAL FUNCTIONS:

-->They are used to perform the analytical operations on the data

Lead() Function:

-->It is used to access subsequent(after) row data along with current row data

Lead(column_name, offset_value, default_value)

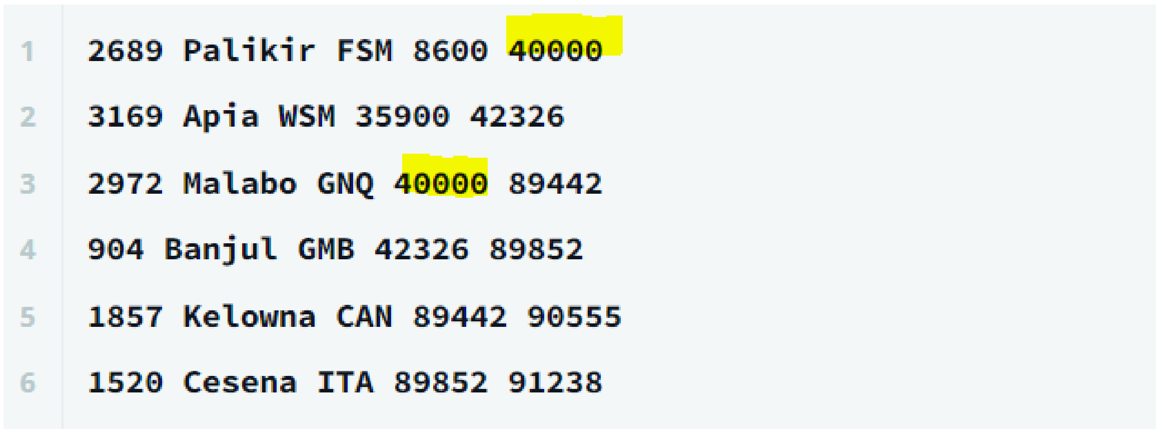
Column_name: which column u want to lead

Offset value: how many rows u want to lead

Default value: if there are no rows to lead give the default value

select top 50 id,name,countrycode,population,
lead(population, 2, -1) over (order by population) as rowNumber
from city

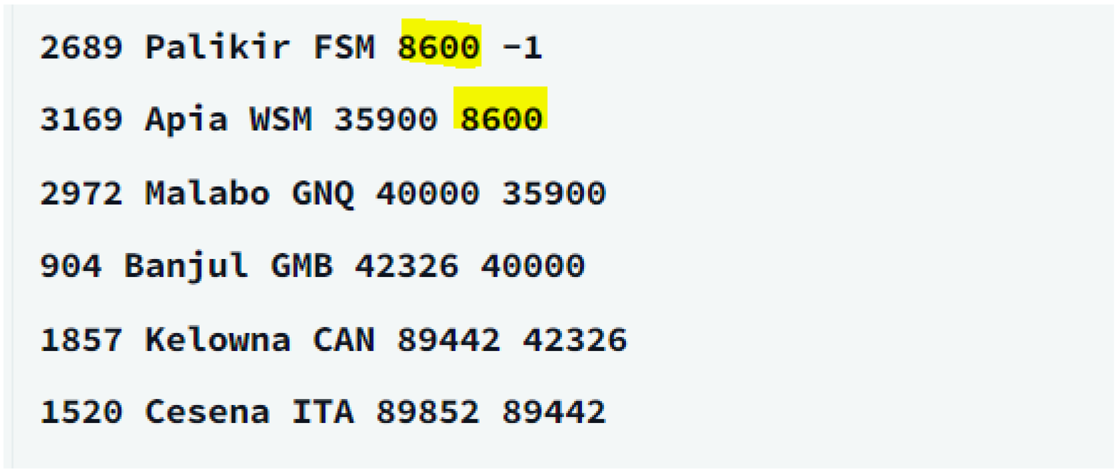
-->As we can see we are leading by two rows



```
2689 Palikir FSM 8600 40000
3169 Apia WSM 35900 42326
2972 Malabo GNQ 40000 89442
904 Banjul GMB 42326 89852
1857 Kelowna CAN 89442 90555
1520 Cesena ITA 89852 91238
```

Lag() Function:
-->It is used to access before row data with current row data

select top 50 id,name,countrycode,population,
lag(population, 1, -1) over (order by population) as rowNumber
from city
-->As the first row don't has any row before it is taken as -1



```
2689 Palikir FSM 8600 -1

3169 Apia WSM 35900 8600

2972 Malabo GNQ 40000 35900

904 Banjul GMB 42326 40000

1857 Kelowna CAN 89442 42326

1520 Cesena ITA 89852 89442
```

Note**

24. will data get transferred from hdfs to hive? Then why map-reduce is invoked?

-->At few instances we can load data from hdfs to hive, data is transferred physically. Hive is meant to work on top of map-reduce to process the data as it was designed that way. When there is a necessity for processing then map reduce is invoked.

25. why for "select" statements MR is not invoked where as for "where" conditions it is invoked?

-->For "select" statements we require a simple copy and paste which isn't based on any logical condition due to which MR is not invoked.

26. 'Load' statement also had similar function which just copies the data from a file path and puts it into another location. No logical condition is invoked which caused MR job to be invoked.