**BIG DATA Bootcamp | Day 6 | Hive Advanced – Hands on**

**Partitions**

**1., 2.**

CREATE TABLE bootcamp.t\_wiki\_partitioned (article STRING, views INT, rank INT)

PARTITIONED BY(year INT, month INT)

STORED AS TEXTFILE

**3.**

INSERT OVERWRITE TABLE bootcamp.t\_wiki\_partitioned PARTITION(year, month)

SELECT t.article, t.views, t.rank, t.year, t.month

FROM bootcamp.t\_wiki t;

**4.**

I chose dynamic mode, because:

* Data are imported from already existing table, no external file;
* Data are structured in a simple and clear way by year and date and would not create too small file structure, if created dynamically.

**5.**

SELECT max(views) FROM t\_wiki WHERE month=’2’

SELECT max(views) FROM t\_wiki\_partitioned WHERE month=’2’

**6.**

lineitem\_avro: 33803 | 1m 41s

lineitem\_ avro\_partitioned: 33803 | 1m 41s

There are no differences in time it took to do QUERY. But theoretically the partitioned file structure should be quicker, because table is already partitioned by month and QUERY doesn’ t have to filter all the table to map only month with a value of 2, but it can use already the partition which is sorted by month with a value of 2. Practically it may have not worked because HDFS currently uses only one node, but the difference would show for many nodes across the cluster.

If the EXPLAIN statement is used to check the differences in map reduce processes, then it is visable that the partitioned table QUERY had to scan much less ‘rows’ (981) with corresponding ‘data size’ compared to non-partitioned table (2949).

EXPLAIN SELECT max(views) FROM t\_wiki WHERE month=’2’ :

Statistics: **Num rows: 2949 Data size: 337179** Basic stats: COMPLETE Column stats: NONE

EXPLAIN SELECT max(views) FROM t\_wiki\_partitioned WHERE month=’2’ :

Statistics: **Num rows: 981 Data size: 25369** Basic stats: COMPLETE Column stats: NONE

**File Storage, inernal vs external tables**

**1.**

To see the file structure I first had to find out where the files ares stored by using these QUERIES and then located the files via HUE interface:

* DESCRIBE FORMATTED t\_wiki;
* DESCRIBE FORMATTED e\_wiki;
* DESCRIBE FORMATTED t\_wiki\_partitioned;

***t\_wiki*** and ***t\_wiki\_partitioned*** directories are saved in a default hive DB storage *warehouse* under the current DB folder ‘bootcamp’ as separte folders for each table. This is because these both are ‘managed tables’ that stores data internally.

***t\_wiki*** has a file created as a default inside the table folder, but ***t\_wiki\_partitioned*** has subfolders as partitions – first subfolder ‘year=2018’ from the first added partition – ‘year’, and 7 subfolders into ‘year’ as each months – from the second added partition - ‘month’.

***e\_wiki,*** on the other hand, is not stored as a file or folder, because It is an external table and is only stored as metadata in HIVE to external files in HDFS system as expected.

**2.**

**Creating Table**

CREATE EXTERNAL TABLE bootcamp.e\_wiki\_partitioned (article STRING, views INT, rank INT)

PARTITIONED BY(year INT, month INT)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

STORED AS TEXTFILE

**Adding Data**

INSERT OVERWRITE TABLE bootcamp.e\_wiki\_partitioned PARTITION(year, month)

SELECT t.article, t.views, t.rank, t.year, t.month

FROM bootcamp.t\_wiki t;

**3.**

**Deleting 1 partition:**

hdfs dfs -rm -R /user/hive/warehouse/bootcamp.db/t\_wiki\_partitioned/year=2018/month=4

hdfs dfs -rm -R /user/hive/warehouse/bootcamp.db/e\_wiki\_partitioned/year=2018/month=4

YES, query for both tables is possible, but the deleted data doesn’t show up.

MSCK REPAIR TABLE doesn’t make any difference. The data are not restored.

When internal table files are deleted, they are deleted from the system. As external table was based on internal table files, it was also not possible to repair.

**UDF**

**4.,5.**

hdfs dfs -put /home/cloudera/Downloads/Jar/\*.jar /libraries/timestamp/

**6.**

CREATE TEMPORARY FUNCTION time\_func AS 'hive.udf.time.StringToEpochMs' USING JAR 'hdfs:///libraries/timestamp/hive-udf-time-0.0.1.jar'

**7., 8.**

I created .csv with timestamps.

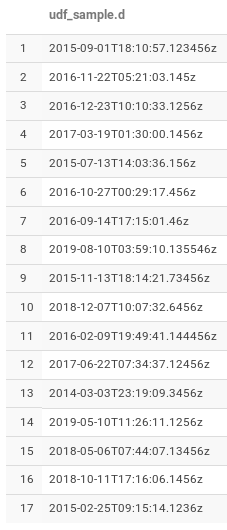
CREATE TABLE udf\_sample (d STRING);

LOAD DATA INPATH '/user/cloudera/times.csv' OVERWRITE INTO TABLE udf\_sample;

**9.**

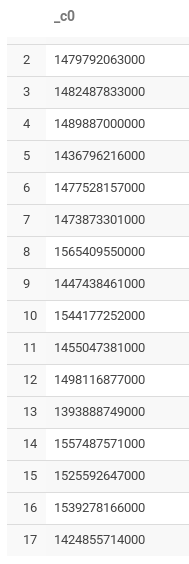
**Original timestamp:**

SELECT \* FROM udf\_sample;



**Transformed timestamp:**

SELECT time\_func(d) FROM udf\_sample;



**10.**

**To check if the data is correct, I can calculate the sum of the seconds in both versions.**

SELECT round(sum(translate(substring(d, 18), "z", ""))), round(sum(time\_func(d)/1000%60))

FROM udf\_sample;



**11.**

SELECT cast(to\_utc\_timestamp(time\_func(d), 'PST') as string) FROM udf\_sample;