**Partition Task**

1. CREATE TABLE t\_wiki\_partitioned

(articles string, views int, rank int)

PARTITIONED BY (year int, month int)

ROW FORMAT DELIMITED

STORED AS ORC;

1. **PARTITIONED BY (year int, month int)**
2. INSERT INTO TABLE t\_wiki\_partitioned

PARTITION (year, month)

SELECT article, views, rank, year, month

FROM t\_wiki;

1. Dynamic partitioning was used, because it is easier and source table contained columns, that the destination table should be partitioned by.
2. The result for **SELECT max(views) FROM t\_wiki WHERE month="2" was 1 minute 29 seconds, for SELECT max(views) FROM t\_wiki\_partitioned WHERE month="2" was 1 minute 26 seconds.**
3. Query containing partitioned table was running faster, because in this case Hive didn’t have to look for all records in the table, but for ones only in partition “month”.

**File Storage, Internal vs External Tables Task**

1. **Describe extended <table\_name>**

**Table\_name = e\_wiki:** hdfs://quickstart.cloudera:8020/user/cloudera/wiki,

**Table\_name = t\_wiki:** hdfs://quickstart.cloudera:8020/user/hive/warehouse/bootcamp.db/t\_wiki

**Table\_name = t\_wiki\_partitioned:** hdfs://quickstart.cloudera:8020/user/hive/warehouse/bootcamp.db/t\_wiki\_partitioned

1. CREATE EXTERNAL TABLE IF NOT EXISTS e\_wiki\_partitioned

(articles\_e string, views\_e int, rank\_e int)

PARTITIONED BY (year\_e int, month\_e int)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ',';

insert into e\_wiki\_partitioned

partition (year\_e, month\_e)

select articles, views, rank, year, month

from t\_wiki\_partitioned;

1. hdfs dfs -rm -r /user/hive/warehouse/bootcamp.db/t\_wiki\_partitioned/year=2018/month=1

hdfs dfs -rm -r /user/hive/warehouse/bootcamp.db/e\_wiki\_partitioned/year\_e=2018/month\_e=1

Yes, it is possible to query data from both tables, but the list of results doesn’t contain values under first partition.

Query list of unique return keys: **select year\_e, month\_e from e\_wiki\_partitioned**

**group by year\_e, month\_e**

**having month\_e>1;**

**UDF Task**

**Put jar on HDFS**: hdfs dfs -put \*.jar /libraries/timestamp/

**Add jar to Hive:** add jar hdfs:///user/cloudera/libraries/timestamp/ hive-example-udf\_2.9.2-0.0.1.jar

add jar hdfs:///user/cloudera/libraries/timestamp/hive-udf-time-0.0.1.jar

**Register temporary function:** CREATE TEMPORARY FUNCTION func\_1 as 'hive.udf.time.StringToEpochMs' using jar 'hdfs:///user/cloudera/libraries/timestamp/hive-udf-time-0.0.1.jar'

**Create table UDF Sample:**

INSERT OVERWRITE TABLE udf\_sample

SELECT

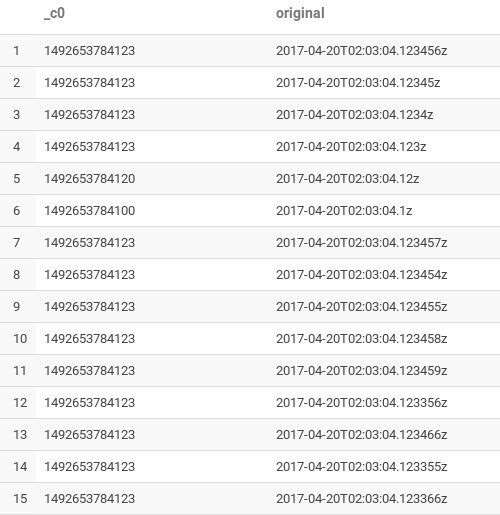
explode(array("2017-04-20T02:03:04.123456z", "2017-04-20T02:03:04.12345z", "2017-04-20T02:03:04.1234z", "2017-04-20T02:03:04.123z",

"2017-04-20T02:03:04.12z", "2017-04-20T02:03:04.1z", "2017-04-20T02:03:04.123457z", "2017-04-20T02:03:04.123454z",

"2017-04-20T02:03:04.123455z", "2017-04-20T02:03:04.123458z", "2017-04-20T02:03:04.123459z", "2017-04-20T02:03:04.123356z",

"2017-04-20T02:03:04.123466z", "2017-04-20T02:03:04.123355z", "2017-04-20T02:03:04.123366z", "2017-04-20T02:03:04.111z")) as d;

**Demonstration of temporary function in action: (screenshot on the next page)**



**Test:**

