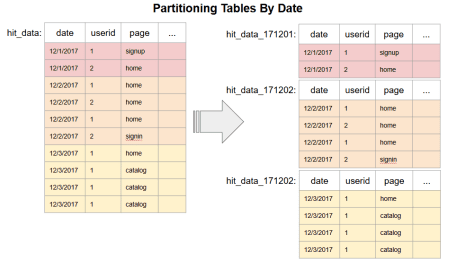
**Introduction**

Companies using Google BigQuery for production analytics often run into the following problem: the company has a large user hit table that spans many years. Since queries are billed based on the fields accessed, and not on the date-ranges queried, queries on the table are billed for all available days and are increasingly wasteful.



A solution is to partition the table by date, so that users can query a particular range of dates; saving costs and decreasing query duration. Partitioning an un-partitioned table can be expensive if done the brute-force way. This article explores one cost-effective partitioning method, and uses the [condusco](https://cran.r-project.org/package=condusco) R Package to automate the query generation and partitioning steps.

**Migrating non-partitioned tables to partitioned tables in Google BigQuery**

Let’s implement the accepted solution on StackOverflow for [migrating from non-partitioned to partitioned tables in Google BigQuery](https://stackoverflow.com/questions/38993877/migrating-from-non-partitioned-to-partitioned-tables).

The brute-force way to partition a non-partitioned table is to repeatedly query the table for anything matching a particular day and then save that data to a new table with the date suffix, ie. \_20171201.

The problem is the cost for this method is the cost of querying the full table’s worth of data, multiplied by the number of days it needs to be partitioned into. For a 10 Terabyte table spanning three years, one SELECT \* might cost $50 (BigQuery charges $5 per TB accessed). Hence, splitting the table into three years of daily partitions will cost $50\*365\*3 = **$54,750**!

The more cost-effective [solution](https://stackoverflow.com/questions/38993877/migrating-from-non-partitioned-to-partitioned-tables) described on StackOverflow is to ARRAY\_AGG the entire table into one record for each day. This requires one query over the table’s data to ARRAY\_AGG each day you are interested in, and then multiple UNNEST queries using a single query on a single column.

This solution queries the full table’s worth of data twice, instead of the number of days. That’s a cost of $100, saving **$54,650**.

Here is an implementation of the solution using [condusco](https://github.com/ras44/condusco) to automate both the query generation and the partitioning:

library(whisker)

library(bigrquery)

library(condusco)

# Set GBQ project

project <- ''

# Configuration

config <- data.frame(

dataset\_id = '',

table\_prefix = 'tmp\_test\_partition'

)

# Set the following options for GBQ authentication on a cloud instance

options("httr\_oauth\_cache" = "~/.httr-oauth")

options(httr\_oob\_default=TRUE)

# Run the below query to authenticate and write credentials to .httr-oauth file

query\_exec("SELECT 'foo' as bar",project=project);

# The pipeline that creates the pivot table

migrating\_to\_partitioned\_step\_001\_create\_pivot <- function(params){

destination\_table <- "{{{dataset\_id}}}.{{{table\_prefix}}}\_partitions"

query <- "

SELECT

{{#date\_list}}

ARRAY\_CONCAT\_AGG(CASE WHEN d = 'day{{{yyyymmdd}}}' THEN r END) AS day\_{{{yyyymmdd}}},

{{/date\_list}}

line

FROM (

SELECT d, r, ROW\_NUMBER() OVER(PARTITION BY d) AS line

FROM (

SELECT

stn, CONCAT('day', year, mo, da) AS d, ARRAY\_AGG(t) AS r

FROM `bigquery-public-data.noaa\_gsod.gsod2017` AS t

GROUP BY stn, d

)

)

GROUP BY line

"

query\_exec(whisker.render(query,params),

project=project,

destination\_table=whisker.render(destination\_table, params),

write\_disposition='WRITE\_TRUNCATE',

use\_legacy\_sql = FALSE

);

}

# Run the pipeline that creates the pivot table

# Create a JSON string in the invocation query that looks like [{"yyyymmdd":"20171206"},{"yyyymmdd":"20171205"},...]

invocation\_query <- "

SELECT

'{{{dataset\_id}}}' as dataset\_id,

'{{{table\_prefix}}}' as table\_prefix,

CONCAT(

'[',

STRING\_AGG(

CONCAT('{\"yyyymmdd\":\"',FORMAT\_DATE('%Y%m%d',partition\_date),'\"}')

),

']'

) as date\_list

FROM (

SELECT

DATE\_ADD(DATE(CURRENT\_DATETIME()), INTERVAL -n DAY) as partition\_date

FROM (

SELECT [1,2,3] as n

),

UNNEST(n) AS n

)

"

run\_pipeline\_gbq(

migrating\_to\_partitioned\_step\_001\_create\_pivot,

whisker.render(invocation\_query,config),

project,

use\_legacy\_sql = FALSE

)

# The pipeline that creates the individual partitions

migrating\_to\_partitioned\_step\_002\_unnest <- function(params){

destination\_table <- "{{{dataset\_id}}}.{{{table\_prefix}}}\_{{{day\_partition\_date}}}"

query <- "

SELECT r.\*

FROM {{{dataset\_id}}}.{{{table\_prefix}}}\_partitions, UNNEST({{{day\_partition\_date}}}) as r

"

query\_exec(whisker.render(query,params),

project=project,

destination\_table=whisker.render(destination\_table, params),

write\_disposition='WRITE\_TRUNCATE',

use\_legacy\_sql = FALSE

);

}

invocation\_query <- "

SELECT

'{{{dataset\_id}}}' as dataset\_id,

'{{{table\_prefix}}}' as table\_prefix,

CONCAT('day\_',FORMAT\_DATE('%Y%m%d',partition\_date)) as day\_partition\_date

FROM (

SELECT

DATE\_ADD(DATE(CURRENT\_DATETIME()), INTERVAL -n DAY) as partition\_date

FROM (

SELECT [1,2,3] as n

),

UNNEST(n) AS n

)

"

run\_pipeline\_gbq(

migrating\_to\_partitioned\_step\_002\_unnest,

whisker.render(invocation\_query,config),

project,

use\_legacy\_sql = FALSE

)