For most scripts you will need to do three things

1. Change the bq load statement to use your flat file instead of loading an Avro file
2. Change the “new” table create sql to fix data formats and data types to match the existing stage table, and implement de-duplicate logic in case the script loads more than batch at a time
3. Call the new archive\_bucket\_files function to move the flat file(s) to archive

The FLOAT exception

* There are some stage tables with FLOAT values - these should have been cleaned up, but were missed
* We are going to fix them as part of this process
* If you look in the util directory, you will see the fix\_pre\_aw\_transaction\_header.bq file, which I used to convert the FLOAT column to a NUMERIC data type
* If you find any float columns in the stage table, please write a fix script the same way I did, save to the util directory and run from there

Deduplicate logic

* To find out which column(s) you should use for the partition by statement (see below), go to the script/val directory
* Open the –validation.bq file for the bq script you’re working on and find the unique key check validation statement
* Please check the validation script – do not assume you know they stage table key
* If key doesn’t seem right to you, talk to Bhaskar or me
* Here is an example from ann\_ann\_aw\_transactions\_header-validation.bq (I highlighted the partition key)

sql='

select count(\*) cnt

from

(

select

h.if\_entry\_no

,count(\*) cnt

from edl\_stage.pre\_aw\_transaction\_header h

group by 1

having count(\*) >1

)

'

run\_validation "$sql" "Unique Key Check"

ann\_aw\_transactions\_header example

* Code changes are highlighted
* Note, removed code is not shown (--use\_avro\_logical\_types=true for example)
* My commentary is in blue
* Please follow this example with your scripts, but if your assigned script doesn’t jive with any part of this example, speak up

#!/usr/bin/bash

. ${HOME}/lib/set\_env.sh

. ${HOME}/lib/common.sh

# Always change from avro load to CSV load

# All of the flags are needed

# You don't need to separate the command on multiple lines

# - I did it here so I could comment on each line

bq load --replace=true --source\_format=CSV --field\_delimiter="|" \

--schema=${schema\_path}/edl\_landing/ann\_aw\_transactions\_header.json \ #no hard coding - use schema\_path env var

edl\_landing.ann\_ann\_aw\_transactions\_header \

"gs://${default\_bucket}/pre/aw/ANN\_AW\_TRANSACTIONS\_HEADER\_\*.dat" #use default\_bucket var; don't forget the '\*';use double quotes ""

rc\_check $? "Load edl\_landing"

# NOTE: test the load before testing the rest of the script.

# You can do that by entering an 'exit 0' after the above rc\_check line

# Before loading stage compare columns from landing to stage to determine which format/data type transforms are required

# If the script has stage load section, you must modify it as well

# bq flags and parameters remain untouched – the sql changes

# note functions for converting date and time, timezone spec "America/New\_York" is very important

# The data types of the selected columns must match the schema of the stage table

bq query --max\_rows 1 --allow\_large\_results --destination\_table edl\_stage.ann\_ann\_aw\_transactions\_header\_new --use\_legacy\_sql=false <<!

SELECT if\_entry\_no

,interface\_control\_flag

,record\_type

,store\_no

,register\_no

,parse\_date("%m/%d/%Y",transaction\_date) as transaction\_date

,parse\_timestamp("%m/%d/%Y %H:%M:%S",entry\_date\_time,"America/New\_York") as entry\_date\_time

,transaction\_series

,transaction\_no

,cashier\_no

,transaction\_category

-- Noticed during testing that tendor\_total was a different format for staged data than what comes from the file

-- In this case dividing by 100 fixes the difference

-- Remember before loading stage, you will need to compare every column from the landed data to the staged version

,tender\_total/100 as tender\_total

,transaction\_void\_flag

,exception\_flag

,deposit\_declaration\_flag

,closeout\_flag

,media\_count\_flag

,tax\_override\_flag

,pos\_tax\_jurisdiction

,employee\_no

,transaction\_remark

,updated\_by\_user\_name

,company\_no

,till\_no

,batch\_id

-- The wild card above allows us to land multiple batches in one run, which provides operational simplicity when dealing with production issues

-- But here is where you have to pay the price - there could be multiple versions of the same record in the landing table

-- The analytic function below groups by the primary key, and selects the max batch\_id

-- Logically, we are grabbing the latest version of each record.

-- If the new table statement in your assigned bq script already has a similar analytic function, you might have to take a different approach

from (select c.\*

,max(c.batch\_id) over (partition by c.if\_entry\_no) as max\_batch\_id

from edl\_landing.ann\_ann\_aw\_transactions\_header c

) curr

where curr.max\_batch\_id = curr.batch\_id

!

rc\_check $? "Load incremental data from edl\_landing into temp table"

# From this point down you should not change any of the logic

bq query --max\_rows 1 --allow\_large\_results --append\_table --destination\_table edl\_stage.ann\_ann\_aw\_transactions\_header\_new --use\_legacy\_sql=false <<!

select c.\*

from edl\_stage.pre\_aw\_transaction\_header c

left join edl\_landing.ann\_ann\_aw\_transactions\_header w

on w.if\_entry\_no = c.if\_entry\_no

where w.if\_entry\_no is null

!

rc\_check $? "append legacy records into the temp table"

bq cp --force edl\_stage.pre\_aw\_transaction\_header edl\_archive.pre\_aw\_transaction\_header

rc\_check $? "archive copy"

bq cp --force edl\_stage.ann\_ann\_aw\_transactions\_header\_new edl\_stage.pre\_aw\_transaction\_header

rc\_check $? "replace the temp table as the stage table"

bq rm --force edl\_stage.ann\_ann\_aw\_transactions\_header\_new

rc\_check $? "drop the temp table"

# This function is only stubbed in – it runs but currently does not archive your files

archive\_bucket\_files "gs://${default\_bucket}/pre/aw/ANN\_AW\_TRANSACTIONS\_HEADER\_\*.dat"