RAMP MLS Playbook

Created by Devesh Ameta, last modified by Chandan Bhandari on Apr 26, 2023

Introduction:

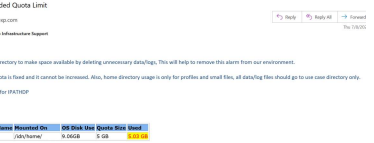
The purpose of RAMP MLS Playbook is to guide and help SAS users to migrate to other Applications i.e. to Pyspark, Python and Hive. The Table of Contents includes the prerequisites required to access certain data used to be available in RAMP, how to import the data in different applications, the platform limitations that users can face, how to make the connections with the different applications and most importantly, how to convert key SAS functionalities into other applications.

1.) Limited CPU Utilization in python

2.) User Home Directory Space Limit in Cornerstone Gold

Cornerstone gold environment has limited space of 5GB in user home directory (/ldn/home/user_ADS_Id). If home directory quota limit exceeds, users will get an email like below:

Volume usage exceeded Quota Limit



BigDataAlerts@aexp.com
EIS DS Hadoop Infrastructure Support

Reply | Reply All | Forward
Thu 7/8/2021

Hi

Kindly clean up the home directory to make space available by deleting unnecessary data/logs. This will help to remove this alarm from our environment.

**Note: Home directory quota is fixed and it cannot be increased. Also, home directory usage is only for profiles and small files, all data/log files should go to use case directory only.

For any queries open ticket for IPATHDP

Volume Details:

Cluster Name	Volume Name	Mounted On	OS Disk Use	Quota Size	Used
gold1cluster		/ldn/home/	16.06GB	5 GB	5.02 GB

If you need more space, you can use your 'use case' path to import the data. Use Case Path Example: **"/axp/gms/user_yarn_queue/dev/User_ads_id"**.

In above example, "user_yarn_queue" is the use case name. The commands to find user specific use case name is documented in this MLS playbook Table of Contents header <#6> and go to <#4>. All users need is to replace the use case name in above example and keep the rest of the path same.

3.) How to download YB data into gold

Just like in RAMP SAS, users can still use ybunload function to download YB data in gold cluster. Using this ybunload function users can download the whole table from YB into Gold user use case. Below are the instructions:

1.) Open putty with the host name "goldgms-edge.gso.aexp.com" and Port "22".

2.) Login with your ads ID and password.

3.) Run the below ybunload - examples to download data from (YB) to gold cluster , Unloads the results of a query to a file csv format.

a.) To download the data from mira yb to gold cluster, use below command:

ybunload –host lgpyb03a.gso.aexp.com –p 5432 –d ybprd002 –format csv –username <**username**> -W -t mira.glbl_mer_dim -o /axp/gms/**your_own_yarn_queue**/dev/**your_ads_id**/unload –prefix glbl_mer_dim –truncate-existing

b.) To download the data from CSIA (YB) to gold cluster, use below command:

/axp/utilities/ybtools/bin/ybunload -h lgpYB10a.gso.aexp.com -d cstonedb3 –username <**username**> -W –format csv -o /axp/gms/**your_yarn_queue**/dev/**your_ads_id**/unload –truncate-existing –select "select product_cd,cust_xref_id from apac_monthly_hist limit 10"

c.) Below query unloads a table from tmp schema:

/axp/utilities/ybtools/bin/ybunload -h lgpYB10a.gso.aexp.com -d cstonedb3 -t tmp.cbr_oscp_all_email –username <**username**> -W -o /axp/gms/**your_yarn_queue**/dev/**your_ads_id**/unload –truncate-existing

Note: keep everything else the same in above example except below:

username = Your ads ID in <username>.

Your_own_yarn_queue: Populate your specific use case name.

Your_ads_id = Keep your ads Id here.

4.) Establish Python or Pyspark Connection using MLS (Access data from Cornerstone)

1.) Browse "https://mlstudio-gold.aexp.com/".

2.) select kernel as Python3.

3.) Click on Create and open notebook.

4.) Check your yarn queue by running the below commands in MLS or in putty respectively and change the same in below python/pyspark connection:

Step1: Run below command to check what CS UseCase(s) you have access:

!mapred queue -showacls | grep SUBMIT **/*this command to use in MLS*/**

mapred queue -showacls | grep SUBMIT **/*just FYI on a separate note: this command to use in putty to check your Use Case(s)*/**

Step2: To run Python or Pyspark query, you need to first use below code snippet to establish connection with CS by using your own Use Case.

/****Important note:**

Copy & paste below code block directly into the notebook. Make sure each and every single command line start from the very beginning of the cell as shown in below screenshot.

```
import os
import sys
import glob
os.environ["SPARK_HOME"]='/opt/mapr/spark/spark'
os.environ['PYSPARK_PYTHON']='/opt/python/python37/bin/python'
spark_python = os.path.join(os.environ.get('SPARK_HOME', None), 'python')
py4j = glob.glob(os.path.join(spark_python, 'lib', 'py4j-*.zip'))[0]
sys.path[0:0]=[spark_python, py4j]
os.environ['PYTHONPATH']=py4j
from pyspark.sql import SparkSession
sqlContext = SparkSession \
    .builder \
    .appName("Python Spark SQL Basic Example") \
    .config("spark.master", "yarn") \
    .config("spark.submit.deployMode", "client") \
    .config("spark.yarn.queue", "admin") \   <-------------- use your yarn queue here i.e. use case name to make the connection
    .config("spark.speculation", "true") \
    .config("spark.sql.parquet.binaryAsString", "true") \
    .config("spark.jars", "/axp/platform/cloak/lib/cloak-spark.jar,/axp/platform/mlplat/app/spy/hive-contrib.jar") \
    .config("spark.driver.extraJavaOptions", "-XX:MaxDirectMemorySize=512m -XX:+UseConcMarkSweepGC -XX:+CMSParallelRemarkEnabled -XX:+UseCMSInitiatingOccupancyOnly -XX:CMSInitiatingOccupancyFraction=30 -XX:+ScavengeBeforeFullGC -XX:+CMSScavengeBeforeRemark") \
    .config("spark.executor.extraJavaOptions", "-XX:MaxDirectMemorySize=512m -XX:+UseConcMarkSweepGC -XX:+CMSParallelRemarkEnabled -XX:+UseCMSInitiatingOccupancyOnly -XX:CMSInitiatingOccupancyFraction=30 -XX:+ScavengeBeforeFullGC -XX:+CMSScavengeBeforeRemark") \
    .config("spark.yarn.dist.files", "/opt/mapr/hive/hive/conf/hive-site.xml") \
    .config("spark.sql.catalogImplementation", "hive") \
    .config("spark.yarn.archive","/axp/platform/cloak/lib/spark-jars.zip") \
    .config("spark.yarn.jars","/axp/platform/cloak/lib/cloak-spark.jar") \
    .config("spark.sql.parquet.writeLegacyFormat","true") \
    .config("spark.sql.execution.arrow.maxRecordsPerBatch","20000") \
    .config("spark.sql.execution.arrow.enabled","true") \
    .enableHiveSupport() \
    .getOrCreate()
#assert isinstance(sqlContext.sparkContext, object)
sc = sqlContext.sparkContext

print(sc)
print(sqlContext)
```

Note:

Users can add or change below config parameters in the above connection code as per their requirements to improve the performance of the whole code. In the below config parameters we are using 4g as just an example for driver and executor memory. Users can change executor and driver memory and instances of executor as per their need.

```
.config("spark.executor.memory", "4g") \
.config("spark.executor.instances","10") \
.config("spark.driver.memory", "4g") \
```

5.) How to Create a Database?

In Order to save your data in your own database, you need to create database first like below by giving your own path in python/pyspark:

```
sqlContext.sql("""CREATE DATABASE IF NOT EXISTS joins LOCATION 'maprfs:/axp/gms/your_yarn_queue/dev/yourads/mlstudio1' """)   <-------------- Use your own yarn queue (i.e. Use Case Name) and ads Id to create database at your own use case folder.
```

Below code is to create a Database in Hive:

```
CREATE DATABASE IF NOT EXISTS joins LOCATION 'maprfs:/axp/gms/your_yarn_queue/dev/yourads/mlstudio1;  <-------------- Use your own yarn queue (i.e. Use Case Name) and ads Id to create database at your own use case folder.
```

How to use your own Hive database before running any code:

```
Use joins;
```

6.) Cornerstone Package

```
from mlfs_cs3 import Cs3

obj = Cs3()

d = {
"qry": '/idn/home/your_ads_id/file.txt',
"of": '/idn/home/Your_ads_Id/cs3_out',
"mdk_log_level": "ERROR"
}

out = obj.extract(**d)
```

Users don't have to run Pyspark connection code to make the connection with the cornerstone if they use above cornerstone package.

Save your Sql query in file.txt and then run the above code in MLS.

After the above code ran successfully. Check your output file at the given 'of' path.

7.) Establish Hive connection in Putty (Access data from Cornerstone)

1.) Open putty with the host name "gcldgms-edge.gso.aexp.com" and Port "22".

2.) Login with your ads ID and password.

3.) Copy "**/axp/platform/cloak/bin/cloak-hive**" in putty by right click and press enter.

In Order to run Hive queries, we need to make database first and use it to run queries.

```
hive> CREATE DATABASE IF NOT EXISTS Joins LOCATION '/idn/home/your_ads_id/hive';

hive > Use joins;
```

8.) Establish Python or Pyspark Connection in Putty (Acess data from Cornerstone)

1.) Open putty with the host name "gcldgms-edge.gso.aexp.com" and Port "22".

2.) Login with your ads ID and password,

3.) Type Python3 in Putty,and press enter.

4.) Check your yarn queue by running the below command in putty and change the same in below python connection code:

Step1: Run below command to check what CS UseCase(s) you have access:

mapred queue -showacls | grep SUBMIT

Step2: To run Python query, you need to first use below code snippet to establish connection with CS by using your own Use Case.

```
import os
import sys
import glob
os.environ['SPARK_HOME']='/opt/mapr/spark/spark'
os.environ['PYSPARK_PYTHON']='/opt/python/python37/bin/python'
spark_python = os.path.join(os.environ.get('SPARK_HOME', None), 'python')
py4j = glob.glob(os.path.join(spark_python, 'lib', 'py4j-*.zip'))[0]
sys.path[:0]=[spark_python, py4j]
os.environ['PYTHONPATH']='py4j'
from pyspark.sql import SparkSession
sqlContext = SparkSession \
    .builder \
    .appName("Python Spark SQL Basic Example") \
    .config("spark.master", "yarn") \
    .config("spark.submit.deployMode", "client") \
    .config("spark.yarn.queue", "admin") \   <-------------- use your yarn queue here i.e. use case name to make the connection
    .config("spark.speculation", "true") \
    .config("spark.sql.parquet.binaryAsString", "true") \
    .config("spark.jars", "/axp/platform/cloak/lib/cloak-spark.jar,/axp/platform/mlplat/app/spy/hive-contrib.jar") \
    .config("spark.driver.extraJavaOptions", "-XX:MaxDirectMemorySize=512m -XX:+UseConcMarkSweepGC -XX:+CMSParallelRemarkEnabled -XX:+UseCMSInitiatingOccupancyOnly -XX:CMSInitiatingOccupancyFraction=30 -XX:+ScavengeBeforeFullGC -XX:+CMSScavengeBeforeRemark") \
    .config("spark.executor.extraJavaOptions", "-XX:MaxDirectMemorySize=512m -XX:+UseConcMarkSweepGC -XX:+CMSParallelRemarkEnabled -XX:+UseCMSInitiatingOccupancyOnly -XX:CMSInitiatingOccupancyFraction=30 -XX:+ScavengeBeforeFullGC -XX:+CMSScavengeBeforeRemark") \
    .config("spark.yarn.dist.files", "/opt/mapr/hive/hive/conf/hive-site.xml") \
    .config("spark.sql.catalogImplementation", "hive") \
    .config("spark.yarn.archive","/axp/platform/cloak/lib/spark-jars.zip") \
    .config("spark.yarn.jars","/axp/platform/cloak/lib/cloak-spark.jar") \
    .config("spark.sql.parquet.writeLegacyFormat","true") \
    .config("spark.sql.execution.arrow.maxRecordsPerBatch","20000") \
    .config("spark.sql.execution.arrow.enabled","true") \
    .enableHiveSupport() \
    .getOrCreate()
#assert isinstance(sqlContext.sparkContext, object)
sc = sqlContext.sparkContext

print(sc)
print(sqlContext)
```

Note:

Users can add or change below config parameters in the above connection code as per their requirements to improve the performance of the whole code. In the below config parameters we are using 4g as just an example for driver and executor memory. Users can change executor and driver memory and instances of executor as per their need.

```
.config("spark.executor.memory", "4g") \
.config("spark.executor.instances","10") \
.config("spark.driver.memory", "4g") \
```

```
sqlContext.sql("""CREATE DATABASE IF NOT EXISTS joins LOCATION 'maprfs:/axp/gms/your_yarn_queue/dev/yourads/mlstudio1' """)   <-------------- Use your own yarn queue (i.e. Use Case Name) and ads id to create database at your own use case folder.
```

› 9.) Execution time of Code (Time function)

If we want to capture the execution time in SAS, MLS or Python.

Use the below code snippets at the start and end of the code to know the execution time.

Time Function:

	Time() in SAS	Time() in MLS	Time() in Python
At the start of the code:	%put NOTE: Starting Execution at %sysfunc(date(),date9.) %sysfunc(time(),time8.) ;	Use %%time at the start of the code.	import time start = time.time()
At the end of the code:	%put NOTE: Ending Execution at %sysfunc(date(),date9.) %sysfunc(time(),time8.) ;	NA	end = time.time() print(end - start)

› 10.) How to import the data ?

How users can import the data is covered below at **"Converting Key SAS Functions into Pyspark, Hive and Python"**

› 11.) EXAMPLE! A User SAS Code End to End Conversion to Pyspark

RAMP User SAS Code End To End Conversion into Pyspark

› 12.) How to move CSV files among RAMP, CS Gold, YB and Your Local D Drive" ?

› 13.) FAQs

Q1: Can users access MIRA YB data in MLS or in Putty ? (MIRA libraries: GMWDM, GMWDMBE, GMWDMGMP, GMWDMOPS, GMWDMPDS, GMWDMUDT, GMWDMPIII, GMWDW, GMWDWBE,GMWDWGMP,GMWDWPDS)?

Ans: NO, Right now users can access above MIRA data only via RAMP. We are working with MIRA and other teams to bring these data into CS. RAMP users will be promptly informed whenever we have updates in the progress. Stay tuned!

Q2. Can users do the joins between Yellowbrick and Cornerstone?

Ans. NO. RAMP team is working on this to get a solution. Will update once the solution is available.

Q3. Can users extract Yellowbrick data in hive?

Ans. NO. RAMP team is working on this to get a solution. Will update once the solution is available.

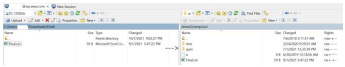
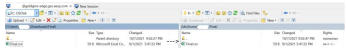
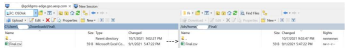
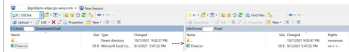
Q4. Can users extract MCRMDM/MCRMRPI data in MLS or in Putty?

Ans. NO, The data in these two libraries reside in MCRM database in YB. Right now users can only access these data via RAMP. However, we are working with MIRA team to access the options to either direct access these data in YB via DBEAVER, and/or bring these data into CS. RAMP users will be promptly informed whenever we have updates in the progress. Stay tuned!

Converting Key SAS Functions into Pyspark, Hive and Python

Note:

PySpark is a Spark library written in Python to run Python application using Apache Spark capabilities. Using PySpark users can run applications parallelly on the distributed cluster (multiple nodes). In other words, PySpark is a Python API for Apache Spark. Apache Spark is an analytical processing engine for large scale powerful distributed data processing. Users can use PySpark to process large data manipulation where Python may have trouble to process.

	SAS	Pyspark	Hive	Python
Query type	SAS Query	Pyspark query	Hive Query	Python Query
Import Data	Users can bring files into SAS servers in below ways: **1.) Via SAS EG :** User can import the file from File option-->Import File, give your required inputs and file will be automatically imported, **2.) Via WinSCP :** **Step1:** Open Winscp **Step2:** Drag your file from local to SAS server like below snapshot:  **Step3:** Click On File--> Import data--> Servers--> go to the path under the files from your server and select your file. **3.) Via Code:** Use the below code snippet to import the file: PROC IMPORT DATAFILE="/idn/home/your_ads/ Final/Final.csv" out=Tracking_file dbms=csv replace; run; **real time 0,02 seconds** **cpu time 0,01 seconds**	**Step1:** Open winscp. **Step2:** Drag your file from local to gold server like below snapshot:  **Step3:** Give your gold server path in the below code snippet to import the file. df = sqlContext.read.csv("/idn/home/your_ads/Final/Final.csv", header=True, inferSchema=True) df.show() **CPU times:user 11,2 ms,** **sys: 5,01 ms,** **total: 16,2 ms** **Wall time: 39 s**	**Step1:** Open winscp. **Step2:** Drag your file from local to gold server like below snapshot:  **Step3:** Create the external table like below code with all the columns and datatypes which you need. hive> CREATE EXTERNAL TABLE IF NOT EXISTS joins.test_table (SE_NO string,FI_CREAT_DT date,STRTUP_PACK_CD int,ACCT_SIGN_CD int,CRNT_SALE_RGN_CD int,CRNT_SALE_DIST_CD string,CRNT_SALE_TERR_CD string,PREV_SALE_TERR_CD int,SE_SRVC_ACS_CD int) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE TBLPROPERTIES("skip.header.line.count"="1"); **Step4**: Give your gold server path of the file in the below code snippet to import the data into the table. hive> LOAD DATA LOCAL INPATH '/idn/home/your_ads_id/Final/Final.csv' INTO TABLE joins.test_table; **Time taken: 13,8 seconds**	**Step1:** Open winscp. **Step2:** Drag your file from local to gold server like below snapshot:  **Step3:** Give your gold server path in the below code snippet to import the file. import pandas as pd df = pd.read_csv("/idn/home/your_ads/Final/Final.csv") df.head() **0,0043 seconds**
Left Join with IN Operator	Proc sql; Create Table Trans_InOp as Select b.subm_se10, b.se_subm_ctry_cd, b.se_subm_srce_sys_id, b.se_trans_am, b.se_trans_dt,b.se_trans_yr_mo from RAMPCS.GMS_MERCHANT_CHAR a left join RAMPCS.gms_transaction b on a.se10=b.subm_se10 where b.se_subm_ctry_cd = '036' and b.se_subm_srce_sys_id ne 'F21' and b.se_trans_yr_mo IN ('202103','202102','202101','202012','202011','202010','202009','202008','202007','202006','202005','202004'); quit; **real time 13:14,63** **cpu time 1:56,85**	sqlContext.sql(""" Create Table joins.Trans_InOp as Select b.subm_se10, b.se_subm_ctry_cd, b.se_subm_srce_sys_id, b.se_trans_am, b.se_trans_dt,b.se_trans_yr_mo from cstonedb3.GMS_MERCHANT_CHAR a left join cstonedb3.gms_transaction b on a.se10=b.subm_se10 where b.se_subm_ctry_cd = '036' and b.se_subm_srce_sys_id != 'F21' and b.se_trans_yr_mo IN ('202103','202102','202101','202012','202011','202010','202009','202008','202007','202006','202005','202004') """) **CPU times: user 112 ms,** **sys: 60,4 ms,** **total: 172 ms** **Wall time: 9min 10s**	Create Table Joins.Trans_InOp as Select b.subm_se10, b.se_subm_ctry_cd, b.se_subm_srce_sys_id, b.se_trans_am, b.se_trans_dt,b.se_trans_yr_mo from cstonedb3.GMS_MERCHANT_CHAR a left join cstonedb3.gms_transaction b on a.se10=b.subm_se10 where b.se_subm_ctry_cd = '036' and b.se_subm_srce_sys_id != 'F21' and b.se_trans_yr_mo IN ('202103','202102','202101','202012','202011','202010','202009','202008','202007','202006','202005','202004'); **Time Taken: 538,145 seconds**	sqlContext.sql(""" Create Table joins.Trans_InOp as Select b.subm_se10, b.se_subm_ctry_cd, b.se_subm_srce_sys_id, b.se_trans_am, b.se_trans_dt,b.se_trans_yr_mo from cstonedb3.GMS_MERCHANT_CHAR a left join cstonedb3.gms_transaction b on a.se10=b.subm_se10 where b.se_subm_ctry_cd = '036' and b.se_subm_srce_sys_id != 'F21' and b.se_trans_yr_mo IN ('202103','202102','202101','202012','202011','202010','202009','202008','202007','202006','202005','202004') """) **454,32 Seconds**
Left Join	Proc SQL; Create Table Trans_B2B_202102 as Select b.subm_se10, b.se_subm_ctry_cd, b.se_subm_srce_sys_id, b.se_trans_am, b.se_trans_dt,b.se_trans_yr_mo from RAMPCS.GMS_MERCHANT_CHAR a left join rampcs.gms_transaction b on a.se10=b.subm_se10 where b.se_subm_ctry_cd = '036' and b.se_subm_srce_sys_id ne 'F21'; and b.se_trans_yr_mo='202102'; quit; **real time 2:00,69** **cpu time 9,88 seconds**	sqlContext.sql(""" Create Table joins.Trans_B2B_202102 as Select b.subm_se10, b.se_subm_ctry_cd, b.se_subm_srce_sys_id, b.se_trans_am, b.se_trans_dt,b.se_trans_yr_mo from cstonedb3.GMS_MERCHANT_CHAR a left join cstonedb3.gms_transaction b on a.se10=b.subm_se10 where b.se_subm_ctry_cd = '036' and b.se_subm_srce_sys_id != 'F21' and b.se_trans_yr_mo='202102' """) **CPU times: user 30,3 ms,** **sys: 17 ms,** **total: 47,4 ms** **Wall time: 2min 10s**	Create Table joins.Trans_B2B_202102 as Select b.subm_se10, b.se_subm_ctry_cd, b.se_subm_srce_sys_id, b.se_trans_am, b.se_trans_dt,b.se_trans_yr_mo from cstonedb3.GMS_MERCHANT_CHAR a left join cstonedb3.gms_transaction b on a.se10=b.subm_se10 where b.se_subm_ctry_cd = '036' and b.se_subm_srce_sys_id != 'F21' and b.se_trans_yr_mo='202102'; **152,889 seconds**	sqlContext.sql(""" Create Table joins.Trans_B2B_202102 as Select b.subm_se10, b.se_subm_ctry_cd, b.se_subm_srce_sys_id, b.se_trans_am, b.se_trans_dt,b.se_trans_yr_mo from cstonedb3.GMS_MERCHANT_CHAR a left join cstonedb3.gms_transaction b on a.se10=b.subm_se10 where b.se_subm_ctry_cd = '036' and b.se_subm_srce_sys_id != 'F21' and b.se_trans_yr_mo='202102' """) **168,59 seconds**
Left Join	Proc SQL; Create Table Trans_B2B_202101 as Select b.subm_se10, b.se_subm_ctry_cd, b.se_subm_srce_sys_id, b.se_trans_am, b.se_trans_dt,b.se_trans_yr_mo from RAMPCS.GMS_MERCHANT_CHAR a left join rampcs.gms_transaction b on a.se10=b.subm_se10 where b.se_subm_ctry_cd = '036' and b.se_subm_srce_sys_id ne 'F21'; and b.se_trans_yr_mo='202101'; quit; **real time 2:01,08** **cpu time 9,86 seconds**	sqlContext.sql(""" Create Table joins.Trans_B2B_202101 as Select b.subm_se10, b.se_subm_ctry_cd, b.se_subm_srce_sys_id, b.se_trans_am, b.se_trans_dt,b.se_trans_yr_mo from cstonedb3.GMS_MERCHANT_CHAR a left join cstonedb3.gms_transaction b on a.se10=b.subm_se10 where b.se_subm_ctry_cd = '036' and b.se_subm_srce_sys_id != 'F21' and b.se_trans_yr_mo='202101' """) **CPU times: user 22,7 ms,** **sys: 7,54 ms,** **total: 30,2 ms** **Wall time: 1min 32s**	Create Table joins.Trans_B2B_202101 as Select b.subm_se10, b.se_subm_ctry_cd, b.se_subm_srce_sys_id, b.se_trans_am, b.se_trans_dt,b.se_trans_yr_mo from cstonedb3.GMS_MERCHANT_CHAR a left join cstonedb3.gms_transaction b on a.se10=b.subm_se10 where b.se_subm_ctry_cd = '036' and b.se_subm_srce_sys_id != 'F21' and b.se_trans_yr_mo='202101'; **122,827 seconds**	sqlContext.sql(""" Create Table joins.Trans_B2B_202101 as Select b.subm_se10, b.se_subm_ctry_cd, b.se_subm_srce_sys_id, b.se_trans_am, b.se_trans_dt,b.se_trans_yr_mo from cstonedb3.GMS_MERCHANT_CHAR a left join cstonedb3.gms_transaction b on a.se10=b.subm_se10 where b.se_subm_ctry_cd = '036' and b.se_subm_srce_sys_id != 'F21' and b.se_trans_yr_mo='202101' """) **122,87 seconds**
Union All	Proc SQL; Create Table Trans_B2B_R12_TMP as Select * from Trans_B2B_202102 union all Select * from Trans_B2B_202101 ; quit; **real time 21,66 seconds** **cpu time 20,47 seconds**	sqlContext.sql(""" Create Table joins.Trans_B2B_R12_TMP as Select * from joins.Trans_B2B_202102 union all Select * from joins.Trans_B2B_202101 """) **CPU times: user 46,1 ms,** **sys: 21,2 ms,** **total: 69,3 ms** **Wall time: 1min 28s**	Create Table joins.Trans_B2B_R12_TMP as Select * from joins.Trans_B2B_202102 union all Select * from joins.Trans_B2B_202101; **169,47 seconds**	sqlContext.sql(""" Create Table joins.Trans_B2B_R12_TMP as Select * from joins.Trans_B2B_202102 union all Select * from joins.Trans_B2B_202101 """) **69,18 seconds**
Extract a data from a table	Proc SQL; Create Table Trans_B2B_R12 as Select *, se_trans_am as SUBM_AM_DCML from Trans_B2B_R12_TMP quit;	sqlContext.sql(""" Create Table joins.Trans_B2B_R12 as	Create Table Joins.Trans_B2B_R12 as Select *, se_trans_am as SUBM_AM_DCML from joins.Trans_B2B_R12_TMP;	sqlContext.sql(""" Create Table joins.Trans_B2B_R12 as

	real time 14,14 seconds **cpu time 14,13 seconds**	Select *, se_trans_am as SUBM_AM_DCML from joins.Trans_B2B_R12_TMP "") **CPU times: user 7,08 ms,** **sys: 3,13 ms,** **total: 10,2 ms** **Wall time: 23,2 s**	**57,255 seconds**	Select *, se_trans_am as SUBM_AM_DCML from joins.Trans_B2B_R12_TMP """) **85,77 seconds**
if/else Logic and Case statement	data trans_B2B_R12v1; set trans_B2B_R12 (where=(SE_TRANS_DT between "2021-01-01" and "2021-02-28")) ; if SUBM_AM_DCML < 0 then ROC = -1; else ROC = 1; format ROC_Band $20. ; if SUBM_AM_DCML >= 0 then ROC_Band = "Positive"; else if -99,99 <= SUBM_AM_DCML < 0 then ROC_Band = 'AUD -99,99 to 0'; else if -199,99 <= SUBM_AM_DCML < -99,99 then ROC_Band = 'AUD -199,99 to -100'; else if SUBM_AM_DCML < -199,99 then ROC_Band = 'AUD -200,00+'; run; **real time 16,19 seconds** **cpu time 16,19 seconds**	sqlContext.sql(""" create table joins.trans_B2B_R12v1 as select *, case when SUBM_AM_DCML < 0 then '-1' else '1' end as ROC, case when SUBM_AM_DCML >= 0 then 'Positive' when SUBM_AM_DCML >=-99,99 and SUBM_AM_DCML < 0 then 'AUD -99,99 to 0' when SUBM_AM_DCML >=-199,99 and SUBM_AM_DCML <-99,99 then 'AUD -199,99 to -100' when SUBM_AM_DCML <-199,99 then 'AUD -200,00+' end as ROC_Band from joins.trans_B2B_R12 where se_trans_dt >= '2021-01-01' and se_trans_dt <= '2021-02-28' """) **CPU times: user 38 ms,** **sys: 6,58 ms,** **total: 44,6 ms** **Wall time: 1min 16s**	create table Joins.Trans_B2B_R12v1 as select *, case when SUBM_AM_DCML<0 then'-1' else '1' end as ROC, case when SUBM_AM_DCML >=0 then'Positive' when SUBM_AM_DCML>=-99,99 and SUBM_AM_DCML<0 then 'AUD -99,99 to 0' when SUBM_AM_DCML>=-199,99 and SUBM_AM_DCML<99,99 then 'AUD -199,99 to -100' when SUBM_AM_DCML<-199,99 then 'AUD-200,00+' end as ROC_Band from Joins.Trans_B2B_R12 where se_trans_dt>='2021-01-01' and se_trans_dt<='2021-02-28'; **61,419 seconds**	sqlContext.sql(""" create table joins.trans_B2B_R12v1 as select *, case when SUBM_AM_DCML < 0 then '-1' else '1' end as ROC, case when SUBM_AM_DCML >= 0 then 'Positive' when SUBM_AM_DCML >=-99,99 and SUBM_AM_DCML < 0 then 'AUD -99,99 to 0' when SUBM_AM_DCML >=-199,99 and SUBM_AM_DCML < 99,99 then 'AUD -199,99 to -100' when SUBM_AM_DCML <-199,99 then 'AUD -200,00+' end as ROC_Band from joins.trans_B2B_R12 where se_trans_dt >= '2021-01-01' and se_trans_dt <= '2021-02-28' """) **135,49 seconds**
Group By	proc sql; Create table B2B_1 as Select SUM (SUBM_AM_DCML) as DBV_LCY, SUM (ROC) as ROCs, ROC_Band From trans_B2B_R12v1 Group by ROC_Band; quit; **real time 11,64 seconds** **cpu time 10,98 seconds**	sqlContext.sql("""Create table joins.B2B_1 as Select SUM (SUBM_AM_DCML) as DBV_LCY, SUM (ROC) as ROCs, ROC_Band From joins.trans_B2B_R12v1 Group by ROC_Band """) **CPU times: user 20,4 ms,** **sys: 3,56 ms,** **total: 24 ms** **Wall time: 1min 17s**	Create table joins.B2B_1 as Select SUM (SUBM_AM_DCML) as DBV_LCY, SUM (ROC) as ROCs, ROC_Band From joins.trans_B2B_R12v1 Group by ROC_Band; **30,779 seconds**	sqlContext.sql("""Create table joins.B2B_1 as Select SUM (SUBM_AM_DCML) as DBV_LCY, SUM (ROC) as ROCs, ROC_Band From joins.trans_B2B_R12v1 Group by ROC_Band """) **52,99 seconds**
Extract the data with no missing values.	libname joins "/idn/home/cbha"; proc sql; Create table joins.B2B_Final as Select * From B2B_1 where (DBV_LCY is not missing and ROCs is not missing); quit; **real time 0,05 seconds** **cpu time 0,00 seconds**	sqlContext.sql("""Create table joins.B2B_Final as Select * from joins.B2B_1 where DBV_LCY is not NULL and ROCs is not NULL """) **CPU times: user 2,84 ms,** **sys: 372 µs,** **total: 3,21 ms** **Wall time: 8,25 s**	Create table joins.B2B_Final as Select * from joins.B2B_1 where DBV_LCY is not NULL and ROCs is not NULL; **69,036 seconds**	sqlContext.sql("""Create table joins.B2B_Final as Select * from joins.B2B_1 where DBV_LCY is not NULL and ROCs is not NULL """) **0,444 seconds**
Extract the data in a new database	libname test "/idn/home/cbha"; proc sql; Create table Test.Trans_B2B_202102 as Select * From Trans_B2B_202102; quit; **real time 2:13,15** **cpu time 12,45 seconds**	sqlContext.sql(""" Create table Test.Trans_B2B_202102 as select * from joins.Trans_B2B_202102""") **CPU times: user 14,5 ms,** **sys: 257 µs,** **total: 14,7 ms** **Wall time: 5,54 s**	Create table Test.Trans_B2B_202102 as select * from joins.Trans_B2B_202102; **81,095 seconds**	sqlContext.sql(""" Create table Test.Trans_B2B_202102 as select * from joins.Trans_B2B_202102""") **53,17 seconds**
Export the data	proc export data = chan.B2B_Final outfile= "/idn/home/your_ads_id/final.xlsx" dbms = xlsx replace; **real time 0,09 seconds** **cpu time 0,01 seconds**	df=sqlContext.sql("""select * from Test.B2B_Final""") df2=df.toPandas() import pandas as pd df2.to_excel("/idn/home/cbha/Final.xlsx") **CPU times: user 38,2 ms,** **sys: 64,1 ms,** **total: 102 ms** **Wall time: 4,68 s**	insert overwrite local directory '/idn/home/cbha/join4' stored as textfile select * from Test.B2B_Final; **Time taken: 17,447 seconds**	df=sqlContext.sql("""select * from Test.B2B_Final""") df2=df.toPandas() import pandas as pd df2.to_excel("/idn/home/cbha/Final.xlsx") **3,83 seconds**

Create a database using macro	libname test "/idn/home/cbha";
	%let a=test;
	
	proc sql inobs=10;
	Create table &a..dataset as select * from rampcs.gms_transaction where se_subm_ctry_cd = '036' and se_trans_yr_mo IN ('202103','202102');
	quit;
	
	PROCEDURE SQL used:
	
	real time 9:58,11
	cpu time 0,15 seconds
	
	Total Time: 9,99 seconds
		try:
		db = 'database'
		location = '/idn/home/cbhan3'
		sqlContext.sql("""CREATE DATABASE IF NOT EXISTS {a} LOCATION
		'{b}/{a}.db"""".format(a=db,b=location))
		print("database {} created successfully".format(db))
		except Exception as e:
		print(e)
		sqlContext.sql("""create table {a}.new_table from cstonedb3.gms_transaction where se_subm_ctry_cd = '036' and se_trans_yr_mo IN ('202103','202102') limit 10"""".format(a=db))
		sqlContext.sql("""select * from {a}.new_table""".format(a=db)).show()
		
		CPU times: user 4,7 ms,
		sys: 163 µs,
		total: 4,87 ms
		Wall time: 9,51 s
			set hivevar:db1 = dbase_chan;
			
			set hivevar:location = maprfs:/idn/home/cbh;
			
			CREATE DATABASE IF NOT EXISTS ${hivevar:db1} LOCATION '${hivevar:location}/${hivevar:db1}.db';
			
			Create table ${hivevar:db1}.new_table as select * from cstonedb3.gms_transaction where se_subm_ctry_cd = '036' and se_trans_yr_mo IN ('202103','202102') limit 10;
			
			Time Taken: 232,86 seconds
				try:
				db = "database"
				location = "/idn/home/cbhan3"
				sqlContext.sql("""CREATE DATABASE IF NOT EXISTS {a} LOCATION
				'{b}/{a}.db"""".format(a=db,b=location))
				print("database {} created successfully".format(db))
				except Exception as e:
				print(e)
				sqlContext.sql("""create table {a}.new_table from cstonedb3.gms_transaction where se_subm_ctry_cd = '036' and se_trans_yr_mo IN ('202103','202102') limit 10"""".format(a=db))
				sqlContext.sql("""select * from {a}.new_table""".format(a=db)).show()
				
				9,26 seconds

Create a table with a left join using macro	libname test"/idn/home/cbh";
	
	%let a =test;
	%let c=RAMPCS;
	
	proc sql ;
	CREATE TABLE &a..trans_inpos_pyspark AS (Select b.subm_se10, b.se_subm_ctry_cd,
	b.se_subm_srce_sys_id, b.se_trans_am, b.se_trans_dt,b.se_trans_yr_mo
	from &c..gms_merchant_char a
	left join &c..GMS_transaction b
	on a.se10=b.subm_se10
	where b.se_subm_ctry_cd = '036'
	and b.se_subm_srce_sys_id ne 'F21'
	and b.se_trans_yr_mo IN ('202103');
	quit;
	
	real time 14:51,85
	cpu time 13,56 seconds
		db1="test"
		db2="cstonedb3"
		try:
		sqlContext.sql("""CREATE TABLE {a}.trans_inpos_pyspark AS (Select b.subm_se10, b.se_subm_ctry_cd,
		b.se_subm_srce_sys_id, b.se_trans_am, b.se_trans_dt,b.se_trans_yr_mo
		from {b}.gms_merchant_char a
		left join {b}.GMS_transaction b
		on a.se10=b.subm_se10
		where b.se_subm_ctry_cd = '036'
		and b.se_subm_srce_sys_id != 'F21'
		and b.se_trans_yr_mo IN ('202103')
		limit 1000
		)"""".format(a=db1,b=db2))
		print("Table Created succesfully")
		except Exception as e:
		print(e)
		
		df = sqlContext.sql("select * from {}.trans_inpos_pyspark".format(db1))
		
		df.show()
		
		CPU times: user 23,2 ms,
			set hivevar:db1 = test;
			
			set hivevar:db2=cstonedb3;
			
			CREATE TABLE ${hivevar:db1}.trans_inpos_pyspark AS (Select b.subm_se10, b.se_subm_ctry_cd,
			b.se_subm_srce_sys_id, b.se_trans_am, b.se_trans_dt,b.se_trans_yr_mo
			from ${hivevar:db2}.gms_merchant_char a
			left join ${hivevar:db2}.gms_transaction b
			on a.se10=b.subm_se10
			where b.se_subm_ctry_cd = '036'
			and b.se_subm_srce_sys_id != 'F21'
			and b.se_trans_yr_mo IN ('202103')
			limit 1000;
			
			select * from ${hivevar:db1}.trans_inpos_pyspark;
			
			Time taken: 226,312 seconds
				db1="test"
				db2="cstonedb3"
				try:
				sqlContext.sql("""CREATE TABLE {a}.trans_inpos_pyspark AS (Select b.subm_se10, b.se_subm_ctry_cd,
				b.se_subm_srce_sys_id, b.se_trans_am, b.se_trans_dt,b.se_trans_yr_mo
				from {b}.gms_merchant_char a
				left join {b}.GMS_transaction b
				on a.se10=b.subm_se10
				where b.se_subm_ctry_cd = '036'
				and b.se_subm_srce_sys_id != 'F21'
				and b.se_trans_yr_mo IN ('202103')
				limit 1000
				)"""".format(a=db1,b=db2))
				print("Table Created succesfully")
				except Exception as e:
				print(e)
				
				df = sqlContext.sql("select * from {}.trans_inpos_pyspark".format(db1))
				
				df.show()
				
				37,22 seconds

		sys: 14.8 ms, **total: 38,1 ms** **Wall time: 1min 58s**		

Create table and perform operations like put, substring,upper, Z format, cats	data df_tranformed;
	set test.trans_inpos_pyspark;
	se_trans_am_new1=upcase(substr(put(se_trans_am,z6.),5,2));
	se_subm_ctry_cd1=cats(se_subm_ctry_cd);
	run;
	
	real time 13,67 seconds
	cpu time 13,61 seconds
		from pyspark.sql.functions import *
		from pyspark.sql.types import StringType,BooleanType,DateType
		
		df_tranformed =
		df.withColumn("se_trans_am_new1",upper(substring(lpad(col("se_trans_am").cast(StringType()),6,'0'),5,2)))
		df_tranformed = df_tranformed.withColumn('se_subm_ctry_cd1', trim(df_tranformed.se_subm_ctry_cd))
		df_tranformed.show()
		
		CPU times: user 33,9 ms,
		
		sys: 4,07 ms,
		
		total: 37,9 ms
		
		Wall time: 1,08 s
			set hivevar:db1=test;
			create table ${hivevar:db1}.df_transformed AS (SELECT *,
			trim(se_subm_ctry_cd) as se_subm_ctry_cd1,
			upper(substr(lpad(cast(se_trans_am as String),6,'0'),5,2)) as se_trans_am_new1
			FROM ${hivevar:db1}.trans_inpos_pyspark);
			
			Time taken: 127,176 seconds
				from pyspark.sql.functions import *
				from pyspark.sql.types import StringType,BooleanType,DateType
				
				df_tranformed =
				df.withColumn("se_trans_am_new1",upper(substring(lpad(col("se_trans_am").cast(StringType()),6,'0'),5,2)))
				df_tranformed = df_tranformed.withColumn('se_subm_ctry_cd1', trim(df_tranformed.se_subm_ctry_cd))
				df_tranformed.show()
				
				3,48 seconds

Create View using macro	Libname test "/idn/home/cbha"; proc sql; create view &a..vw_trans_inpos_pyspark_new as select subm_se10, se_subm_ctry_cd, se_subm_srce_sys_id, se_trans_am, se_trans_dt,se_trans_am_new1 from df_transformed where se_trans_yr_mo='202103' order by subm_se10; quit; **real time 0,01 seconds** **cpu time 0,00 seconds**	db1="test" try: df_tranformed.createOrReplaceTempView("df_table") sqlContext.sql("""CREATE TABLE IF NOT EXISTS {}.trans_inpos_pyspark_new AS SELECT * FROM df_table""".format(db1)) print("table for view created") except Exception as e: print(e) try: sqlContext.sql("""Create or replace view {a}.vw_trans_inpos_pyspark_new AS (select subm_se10, se_subm_ctry_cd, se_subm_srce_sys_id, se_trans_am, se_trans_dt, se_trans_am_new1 from {a}.trans_inpos_pyspark_new where se_trans_yr_mo='202103' order by subm_se10)""".format(a=db1)) print("View created successfully") except Exception as e: print(e) df_view = sqlContext.sql("Select * from {}.vw_trans_inpos_pyspark_new".format(db1)) df_view.show() **CPU times: user 5,52 ms,** **sys: 1,24 ms,** **total: 6,76 ms** **Wall time: 3,47 s**	set hivevar:db1=test; Create or replace view ${hivevar:db1}.vw_trans_inpos_pyspark_new AS (select subm_se10, se_subm_ctry_cd, se_subm_srce_sys_id, se_trans_am, se_trans_dt, se_trans_am_new1 from ${hivevar:db1}.df_transformed where se_trans_yr_mo='202103' order by subm_se10); **Time taken: 0,477 seconds**	db1="test" try: df_tranformed.createOrReplaceTempView("df_table") sqlContext.sql("""CREATE TABLE IF NOT EXISTS {}.trans_inpos_pyspark_new AS SELECT * FROM df_table""".format(db1)) print("table for view created") except Exception as e: print(e) try: sqlContext.sql("""Create or replace view {a}.vw_trans_inpos_pyspark_new AS (select subm_se10, se_subm_ctry_cd, se_subm_srce_sys_id, se_trans_am, se_trans_dt, se_trans_am_new1 from {a}.trans_inpos_pyspark_new where se_trans_yr_mo='202103' order by subm_se10)""".format(a=db1)) print("View created successfully") except Exception as e: print(e) df_view = sqlContext.sql("Select * from {}.vw_trans_inpos_pyspark_new".format(db1)) df_view.show() **25,50 seconds**
count distinct	proc sql; create table df_distinct_count as select count(distinct(subm_se10)) as subm_se10_new from vw_trans_inpos_pyspark_new ;quit; **real time 11,15 seconds** **cpu time 10,30 seconds**	from pyspark.sql.functions import countDistinct df_distinct_count = df_view.select(countDistinct("subm_se10")).toDF("subm_se10_new") df_distinct_count.show() **CPU times: user 16,8 ms,** **sys: 5,91 ms,** **total: 22,7 ms** **Wall time: 15,4 s**	SELECT COUNT(DISTINCT subm_se10) as subm_se10_new FROM ${hivevar:db1}.vw_trans_inpos_pyspark_new; **Time taken: 82,161 seconds**	from pyspark.sql.functions import countDistinct df_distinct_count = df_view.select(countDistinct("subm_se10")).toDF("subm_se10_new") df_distinct_count.show() **3,83 seconds**
Proc freq	proc freq data=vw_trans_inpos_pyspark_new; table se_subm_srce_sys_id/ missing; run; **real time 27,09 seconds** **cpu time 46,58 seconds**	from pyspark.sql.functions import count frequencies = df_view.groupBy('se_subm_srce_sys_id').agg(count('se_subm_srce_sys_id').alias('Frequency')).selectExpr("*", "100 * Frequency / sum(Frequency) over() Percent').selectExpr("*", "sum(Frequency) over(order by Frequency desc) Cumulative_Frequency', "sum(Percent) over(order by Frequency desc) Cumulative_Percent') frequencies.show() **CPU times: user 3,15 ms,** **sys: 1,92 ms,** **total: 5,08 ms** **Wall time: 2,89 s**	with tbl AS (SELECT x.se_subm_srce_sys_id,x.occurrences, 100*x.occurrences/SUM(x.occurrences) OVER() AS percent from (select se_subm_srce_sys_id, count(se_subm_srce_sys_id) as occurrences from ${hivevar:db1}.vw_trans_inpos_pyspark_new group by se_subm_srce_sys_id)x) SELECT tbl.se_subm_srce_sys_id ,tbl.occurrences as frequency` , tbl.prcent, sum(tbl.occurrences) over(order by tbl.occurrences desc) AS Cumulative_Frequency, sum(tbl.prcent) over(order by tbl.occurrences desc) Cumulative_Percent from tbl; **Time taken: 190,537 seconds**	from pyspark.sql.functions import count frequencies = df_view.groupBy('se_subm_srce_sys_id').agg(count('se_subm_srce_sys_id').alias('Frequency')).selectExpr("*", "100 * Frequency / sum(Frequency) over() Percent').selectExpr("*", "sum(Frequency) over(order by Frequency desc) Cumulative_Frequency', "sum(Percent) over(order by Frequency desc) Cumulative_Percent') frequencies.show() **3,85 seconds**
Proc means with N and N miss	proc means n nmiss mean min max sum data=vw_trans_inpos_pyspark_new; run; **real time 29,82 seconds** **cpu time 48,66 seconds**	from pyspark.sql.functions import avg mean=df_view.select(avg('se_subm_ctry_cd').alias("mean")) mean.show() from pyspark.sql.functions import col,isnan # It filters out all the rows which contain null, 'None', 'NULL', or empty string values and returns a count from a given column null_values = df_view.filter(df_view.subm_se10.contains('None')	\ df_view.subm_se10.contains('NULL')	\ (col("subm_se10") == '')	\ isnan(df_view.subm_se10)	\ df_view.subm_se10.isNull()).count() # Total count minus the count of null values gives the count of not null values. Not_Null_Values = df_view.count()-null_values data = [(null_values,Not_Null_Values)] #Creating a dataframe of null and not null counts df_new = sqlContext.createDataFrame(data,["Null_values","Not_Null_values"]) df_new.show() **CPU times: user 17,1 ms,** **sys: 4,07 ms,** **total: 21,2 ms** **Wall time: 8,42 s**	select avg(se_subm_ctry_cd) as mean from ${hivevar:db1}.vw_trans_inpos_pyspark_new; with table1 as (select sum(case when subm_se10 = '' then 1 when subm_se10 = "None" then 1 when subm_se10 is null then 1 when subm_se10 = "Null" then 1 else 0 end) null_count, count(*) as Total from ${hivevar:db1}.vw_trans_inpos_pyspark_new) select table1.null_count as null_values, (table1.total - table1.null_count) as Not_Null_values from table1; **Time taken: 80,354 seconds**	from pyspark.sql.functions import avg mean=df_view.select(avg('se_subm_ctry_cd').alias("mean")) mean.show() from pyspark.sql.functions import col,isnan # It filters out all the rows which contain null, 'None', 'NULL', or empty string values and returns a count from a given column null_values = df_view.filter(df_view.subm_se10.contains('None')	\ df_view.subm_se10.contains('NULL')	\ (col("subm_se10") == '')	\ isnan(df_view.subm_se10)	\ df_view.subm_se10.isNull()).count() # Total count minus the count of null values gives the count of not null values. Not_Null_Values = df_view.count()-null_values data = [(null_values,Not_Null_Values)] #Creating a dataframe of null and not null counts df_new = sqlContext.createDataFrame(data,["Null_values","Not_Null_values"]) df_new.show() **5,49 seconds**
Nodupkey with dupout	Data trans_inpos_pyspark2; set vw_trans_inpos_pyspark_new; run; proc sort nodupkey data=trans_inpos_pyspark2 dupout=df_duplicate; by subm_se10; run;	from pyspark.sql.window import Window from pyspark.sql.functions import col, row_number # creating a window function which will partition by the given column and and order by the given column w2 = Window.partitionBy("subm_se10").orderBy(col("subm_se10")) # Selecting all the rows which are duplicate and will be dropped from the main dataframe into a new dataframe df_duplicate = df_view.withColumn("row",row_number().over(w2)).filter(col("row") > 1).drop("row")	--x. (m)?+.+` is a regex which means selecting every column from the table except the row number column i.e. â€™m&€™ because it is used only for the transformation and has no significance in the result. set hive.support.quoted.identifiers=none; select x.`(m)?+.+` from (select e.*, row_number() over (partition by subm_se10 order by subm_se10) as m from ${hivevar:db1}.vw_trans_inpos_pyspark_new e)x where x.m > 1;	from pyspark.sql.window import Window from pyspark.sql.functions import col, row_number # creating a window function which will partition by the given column and and order by the given column w2 = Window.partitionBy("subm_se10").orderBy(col("subm_se10")) # Selecting all the rows which are duplicate and will be dropped from the main dataframe into a new dataframe df_duplicate = df_view.withColumn("row",row_number().over(w2)).filter(col("row") > 1).drop("row")

	real time 19,44 seconds
	cpu time 46,90 seconds
		# Selecting all the rows which are unique into a new dataframe.
		df_unique = df_view.withColumn("row",row_number().over(w2)).filter(col("row") == 1).drop("row")
		
		df_duplicate.show()
		df_unique.show()
			select x.* from (
			select a.*,
			row_number() over (partition by subm_se10 order by subm_se10) as rn
			from ${hivevar:db1}.vw_trans_inpos_pyspark_new a)x
			where x.rn = 1;
				# Selecting all the rows which are unique into a new dataframe.
				df_unique = df_view.withColumn("row",row_number().over(w2)).filter(col("row") == 1).drop("row")
				
				df_duplicate.show()
				df_unique.show()

		CPU times: user 4,68 ms, **sys: 2,93 ms,** **total: 7,61 ms,** **Wall time: 5,25 s**	**Time taken: 122,611 seconds** **Time taken: 116,222 seconds**	**2,24 seconds**
Sum(), **min(),** **max(),** **Count(*)**	proc sql; create table df_operations as select sum(se_trans_am) as total_Sum, count(*) as total_count, min(se_trans_am) as min, max(se_trans_am) as max, min(INPUT(se_trans_dt,yymmdd10.)) as min_date format=yymmdd10., max(INPUT(se_trans_dt,yymmdd10.)) as max_date format=yymmdd10. from trans_inpos_pyspark2; quit; **real time 0,08 seconds** **cpu time 0,08 seconds**	df_view.createOrReplaceTempView("df_operations") df_operations = sqlContext.sql("""with tbl as (select min(se_trans_dt) as min_date, max(se_trans_dt) as max_date, sum(se_trans_am) as total_Sum, min(se_trans_am) as min, max(se_trans_am) as max, count(*) as total_count from df_operations) select tbl.min,tbl.max,tbl.total_sum,tbl.total_count, from_unixtime(unix_timestamp(tbl.min_date ,'yyyy-MM-dd'), 'ddMMMyyyy') as min_date, from_unixtime(unix_timestamp(tbl.max_date ,'yyyy-MM-dd'), 'ddMMMyyyy') as max_date from tbl """) df_operations.show() **CPU times: user 2,19 ms,** **sys: 490 µs,** **total: 2,68 ms,** **Wall time: 526 ms**	with tbl as (select min(se_trans_dt) as min_date,max(se_trans_dt) as max_date, sum(se_trans_am) as total_Sum, min(se_trans_am) as min, max(se_trans_am) as max, count(*) as total_count from ${hivevar:db1}.vw_trans_inpos_pyspark_new) select tbl.min,tbl.max,tbl.total_sum,tbl.total_count, from_unixtime(unix_timestamp(tbl.min_date ,'yyyy-MM-dd'), 'ddMMMyyyy') as min_date,from_unixtime(unix_timestamp(tbl.max_date ,'yyyy-MM-dd'), 'ddMMMyyyy') as max_date from tbl; **Time taken: 147,002 seconds**	df_view.createOrReplaceTempView("df_operations") df_operations = sqlContext.sql("""with tbl as (select min(se_trans_dt) as min_date, max(se_trans_dt) as max_date, sum(se_trans_am) as total_Sum, min(se_trans_am) as min, max(se_trans_am) as max, count(*) as total_count from df_operations) select tbl.min,tbl.max,tbl.total_sum,tbl.total_count, from_unixtime(unix_timestamp(tbl.min_date ,'yyyy-MM-dd'), 'ddMMMyyyy') as min_date, from_unixtime(unix_timestamp(tbl.max_date ,'yyyy-MM-dd'), 'ddMMMyyyy') as max_date from tbl """) df_operations.show() **1,24 seconds**
Drop view	proc sql; drop view vw_trans_inpos_pyspark_new ;quit; **real time 0,00 seconds** **cpu time 0,00 seconds**	try: sqlContext.sql("""drop view {}.vw_trans_inpos_pyspark_new""".format(db1)) print("View dropped successfully") except Exception as e: print(e) **CPU times: user 1,82 ms,** **sys: 650 µs,** **total: 2,46 ms,** **Wall time: 235 ms**	drop view ${hivevar:db1}.vw_trans_inpos_pyspark_new; **Time taken: 7,291 seconds**	try: sqlContext.sql("""drop view {}.vw_trans_inpos_pyspark_new""".format(db1)) print("View dropped successfully") except Exception as e: print(e) **0,001 seconds**
Delete an observation	proc sql; delete from trans_inpos_pyspark where se_trans_dt== 2021-03-11 ;quit; **real time 2:03,57** **cpu time 22,23 seconds**	try: df = df.filter(df.se_trans_dt < '2021-03-11') df.createOrReplaceTempView("NewTable") sqlContext.sql("INSERT OVERWRITE TABLE {}.trans_inpos_pyspark select * from NewTable".format(db1)) print("rows deleted successfully") except Exception as e: print(e) **CPU times: user 3,21 ms,** **sys: 1,09 ms,** **total: 4,3 ms,** **Wall time: 78,8 ms**	Create table ${hivevar:db1}.trans_inpos_pyspark_temp like ${hivevar:db1}.trans_inpos_pyspark; INSERT INTO ${hivevar:db1}.trans_inpos_pyspark_temp select * FROM ${hivevar:db1}.trans_inpos_pyspark where se_trans_dt < '2021-03-11'; INSERT OVERWRITE TABLE ${hivevar:db1}.trans_inpos_pyspark select * from ${hivevar:db1}.trans_inpos_pyspark_temp; drop Table ${hivevar:db1}.trans_inpos_pyspark_temp; **Time taken: 27,176 seconds**	try: df = df.filter(df.se_trans_dt < '2021-03-11') df.createOrReplaceTempView("NewTable") sqlContext.sql("INSERT OVERWRITE TABLE {}.trans_inpos_pyspark select * from NewTable".format(db1)) print("rows deleted successfully") except Exception as e: print(e) **0,04 seconds**

Append the data	proc sql;
	CREATE TABLE chand.trans_inpos_pyspark3 AS (Select a.subm_se10, a.se_subm_ctry_cd,
	a.se_subm_srce_sys_id, a.se_trans_am, a.se_trans_dt,a.se_trans_yr_mo
	from &c.gms_transaction a
	left join &c.GMS_MERCHANT_CHAR b
	on a.subm_se10=b.se10
	where a.se_subm_ctry_cd = '036'
	and a.se_subm_srce_sys_id != 'F21'
	and a.se_trans_yr_mo IN ('202102');
	quit;
	
	proc append data= trans_inpos_pyspark3 base=trans_inpos_pyspark;
	
	run;
	
	real time 8,92 seconds
	cpu time 6,87 seconds
		df2 = sqlContext.sql("""Select a.subm_se10, a.se_subm_ctry_cd, a.se_subm_srce_sys_id, a.se_trans_am,
		a.se_trans_dt,a.se_trans_yr_mo
		from {b}.gms_transaction a
		left join {b}.GMS_MERCHANT_CHAR b
		on a.subm_se10=b.se10
		where a.se_subm_ctry_cd = '036'
		and a.se_subm_srce_sys_id != 'F21'
		and a.se_trans_yr_mo IN ('202102')""".format(b=db2))
		
		df_union = df.union(df2)
		df_union.show()
		
		CPU times: user 28,9 ms,
		
		sys: 13,1 ms,
		
		total: 42 ms,
		
		Wall time: 1min 34s
			set hivevar:db1=test;
			set hivevar:db2=cstonedb3;
			
			(select * from ${hivevar:db1}.trans_inpos_pyspark)
			union
			(Select a.subm_se10, a.se_subm_ctry_cd, a.se_subm_srce_sys_id, a.se_trans_am,
			a.se_trans_dt,a.se_trans_yr_mo
			from ${hivevar:db2}.gms_transaction a
			left join ${hivevar:db2}.GMS_MERCHANT_CHAR b
			on a.subm_se10=b.se10
			where a.se_subm_ctry_cd = '036'
			and a.se_subm_srce_sys_id != 'F21'
			and a.se_trans_yr_mo IN ('202102'));
			
			Time taken: 541,084 seconds
				df2 = sqlContext.sql("""Select a.subm_se10, a.se_subm_ctry_cd, a.se_subm_srce_sys_id, a.se_trans_am,
				a.se_trans_dt,a.se_trans_yr_mo
				from {b}.gms_transaction a
				left join {b}.GMS_MERCHANT_CHAR b
				on a.subm_se10=b.se10
				where a.se_subm_ctry_cd = '036'
				and a.se_subm_srce_sys_id != 'F21'
				and a.se_trans_yr_mo IN ('202102')""".format(b=db2))
				
				df_union = df.union(df2)
				df_union.show()
				
				144,2 seconds

Drop Tables	proc sql; drop table trans_inpos_pyspark ;quit; **real time 1,91 seconds** **cpu time 1,91 seconds**	db1="test" try: sqlContext.sql("drop table {}.trans_inpos_pyspark".format(db1)) print("main table deleted") except Exception as e: print(e) **CPU times: user 868 µs,** **sys: 1,15 ms,** **total: 2,02 ms,** **Wall time: 348 ms**	DROP TABLE ${hivevar:db1}.trans_inpos_pyspark; **Time taken: 0,189 seconds**	db1="test" try: sqlContext.sql("drop table {}.trans_inpos_pyspark".format(db1)) print("main table deleted") except Exception as e: print(e) **0,199 seconds**

No labels

4 Comments

Charles A Flynt

FYI - exporting to PDF doesn't provide readable output. Can someone fix that? Be good to have a version of this in PDF but current export option does not allow for horizontal scrolling, so anything that doesn't fit on one page is lost. This means the side by side conversion examples, the most valuable part of the document, do not translate to the PDF version.

Julia J Bao

this book was directly built in confluence due to the 3 programming tools comparison. PDF format not working, you will need to go to this confluence to read it

Charles A Flynt

Thanks for the reply.

I would note, it's actually difficult to read in this confluence format too, even on large monitors, due to having to scroll horizontally. Would be better in another format, which would also lend itself to better conversion to PDF as a side benefit. Hopefully someone will revise it someday to be more accessible. It seems a good resource, just unfortunately not well formatted.

Julia J Bao

Thanks for your feedback! Since RAMP has been decommissioned, SAS is no longer a programming language. So users like you would need to copy and paste the areas needed into own document. I completely understand it is quit an effort for end users to do so .