**NOTES OF AZURE DATA BRICKS**

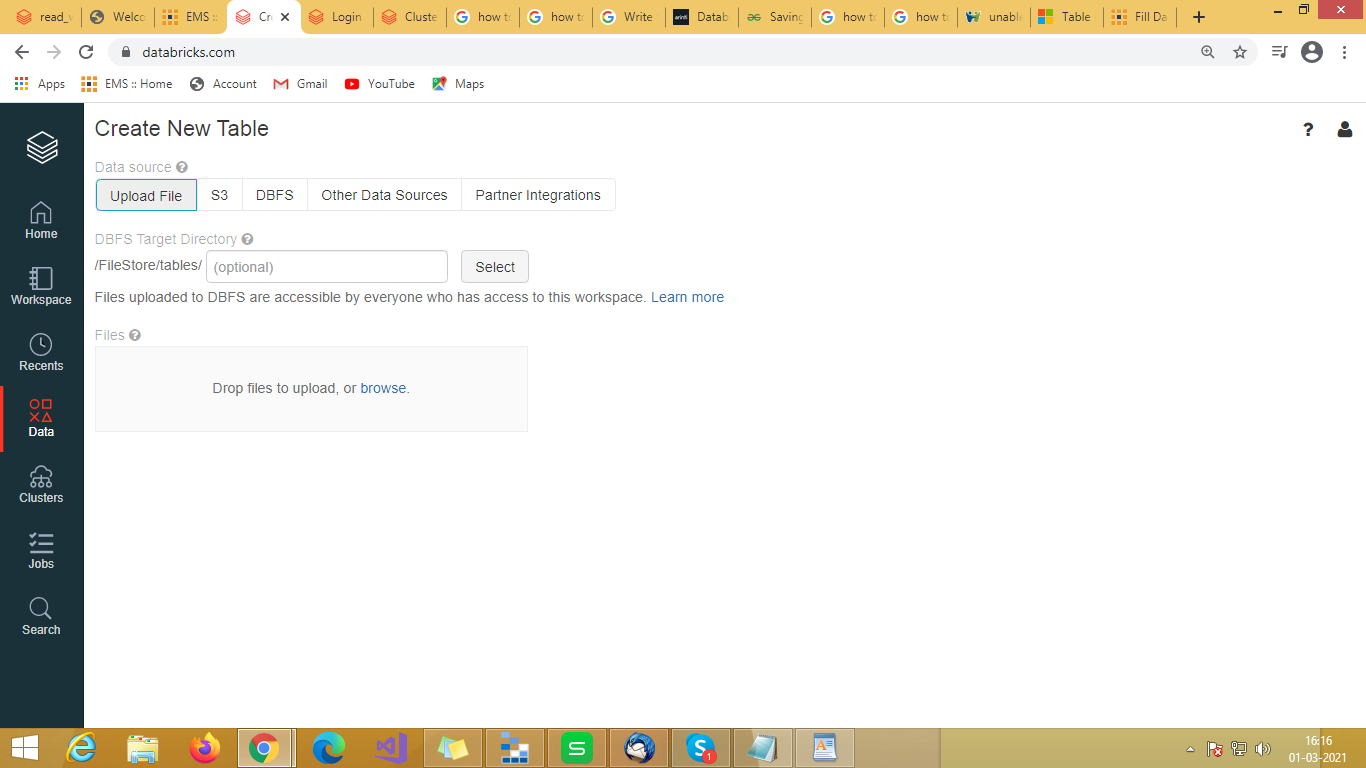
STEP 1: Create a Cluster

STEP 2: Create a NoteBook

STEP 3: Connect Cluster with NoteBook

Read CSV file

1. Upload the csv file in

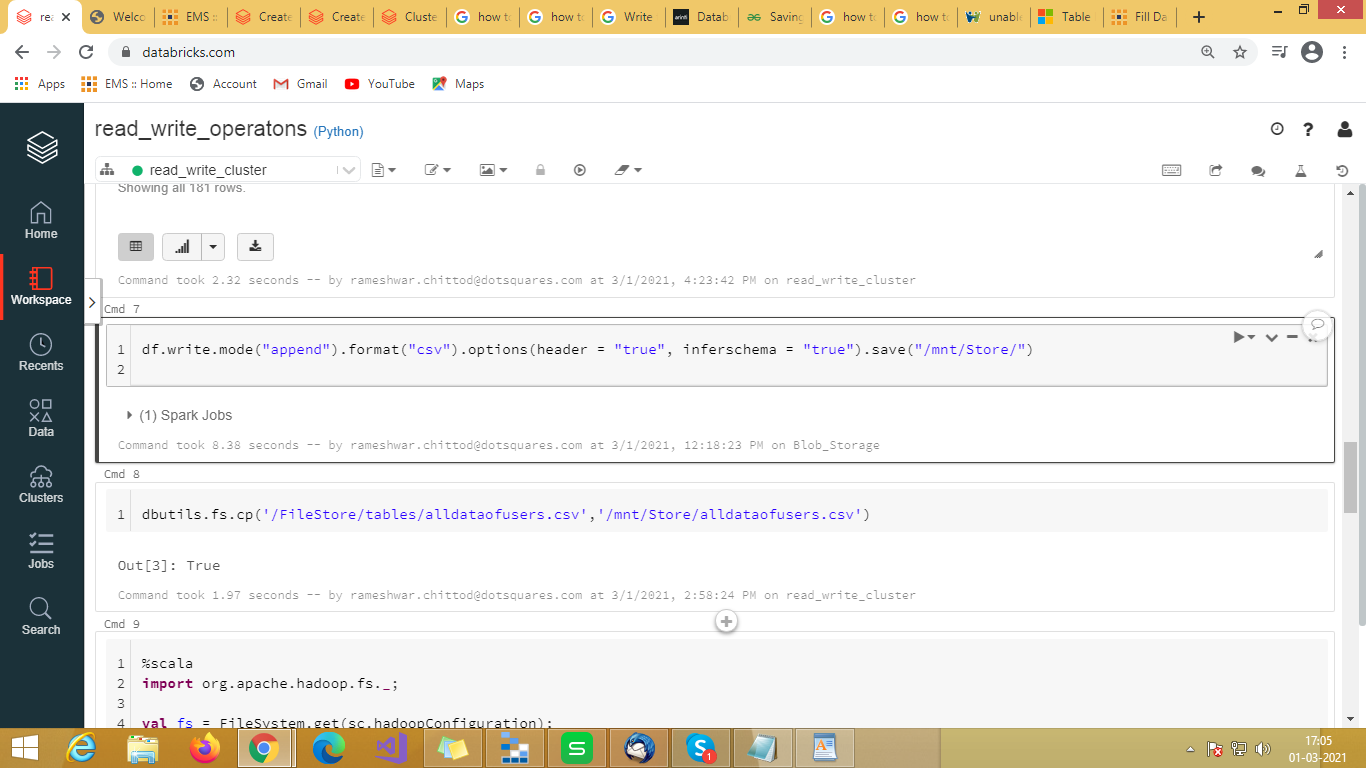


2.

%Python

df = spark.read.format("csv").options(header = "true", inferschema = "true").load("/FileStore/tables/alldataofusers.csv")

Display(df)



**NOTE**

⦁ In load we put the path of the file

⦁ in Format section we can write any format like :- csv, parquet, text, Delta, json etc.

⦁ the first line load the file in 'df' variable

⦁ second line display the result

**You can also read the nested json file**

df = spark.read.option("multiline", "true").json("/FileStore/tables/userjsondata.json")

from pyspark.sql.functions import explode, col

persons = df.select(explode("Sheet1").alias("Sheet"))

display(persons.select("Sheet.Age", "Sheet.First Name"))

**Join Operation**

df1 = spark.read.csv("PATH OF THE FILE 1")

df2 = spark.read.csv("PATH OF THE FILE 2")

df3 = df1.join(df2, df1.Primary\_key == df2.Foreign\_Key)

display(df3)

**Group Operation**

import pyspark.sql.function as f

pf = df.group by("Date").agg(

f.sum("Column-name").alias("total\_sum"),

f.count("Column-name").alias("total\_count"),

)

)

display(pf)

**Write File**

df = spark.read.format("csv").options(header = "true", inferschema = "true").load("/FileStore/tables/alldataofusers.csv")

df.write.mode("overwrite").format("csv").options(header = "true", inferschema = "true").save("/FileStore/tables/data/")

**NOTE:**

⦁ first line read file from the particular location

⦁ second step is used to write to file to given location .the given location is /FileStore/tables/data/")

⦁ the mode overwrite "mode("overwrite")." is used to create a new file and rewrite the file

**Append a File**

df = spark.read.format("csv").options(header = "true", inferschema = "true").load("/FileStore/tables/alldataofusers.csv")

df.write.mode("Append").format("csv").options(header = "true", inferschema = "true").save("/FileStore/tables/alldataofusers.csv")

**NOTE:**

⦁ append mode used to append a file OR insert a new Record in the same file

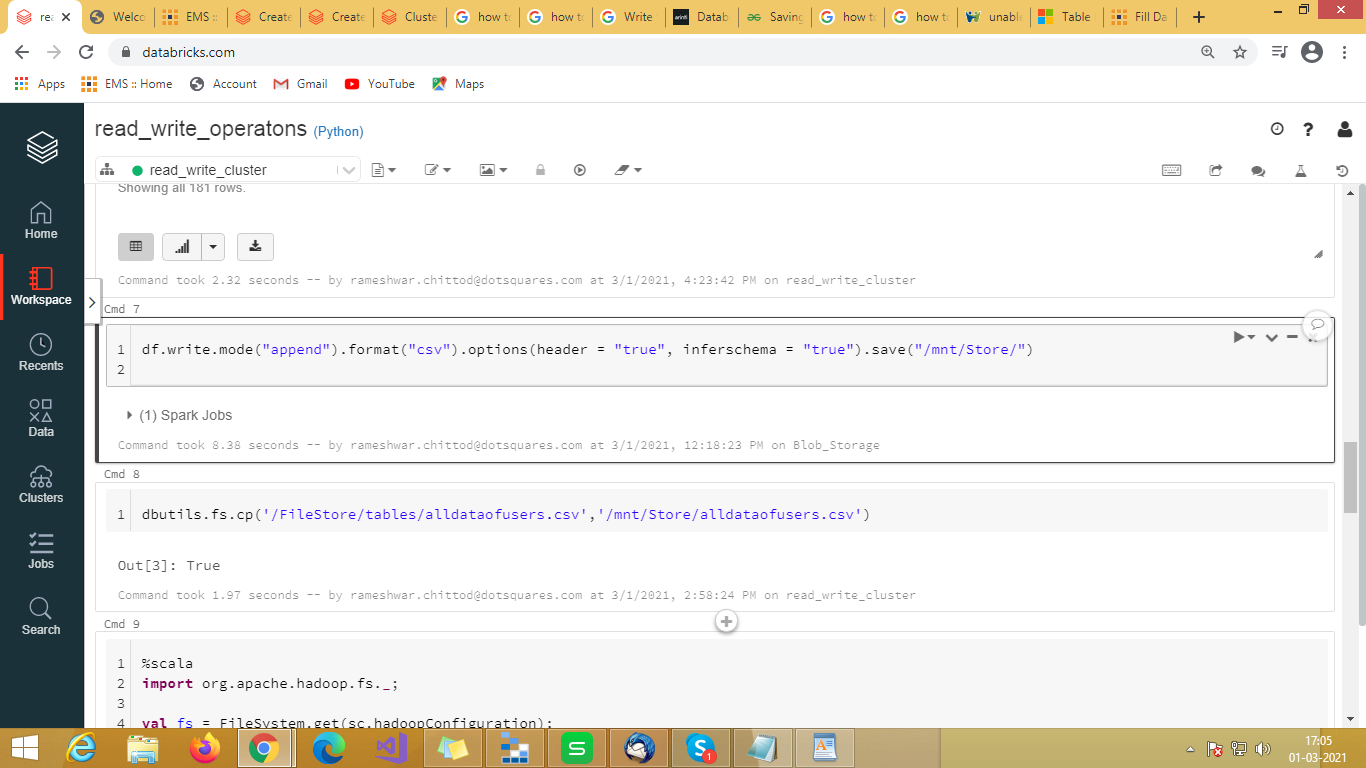
**COPY the file**

dbutils.fs.cp("/FileStore/tables/alldataofusers.csv" ,"/FileStore/tables/data/alldataofusers.csv")

**NOTE:**

⦁ /FileStore/tables/alldataofusers.csv location of fetching the file

⦁ ,"/FileStore/tables/data/alldataofusers.csv make a copy the the given path



**SAVE FILE**

df = spark.read.format("csv").options(header = "true", inferschema = "true").load("/FileStore/tables/alldataofusers.csv")

df.write.format("csv").saveAsTable("a.csv")

**OR**

df.write.mode("overwrite").format("csv").options(header = "true", inferschema = "true").save("/FileStore/tables/data/")

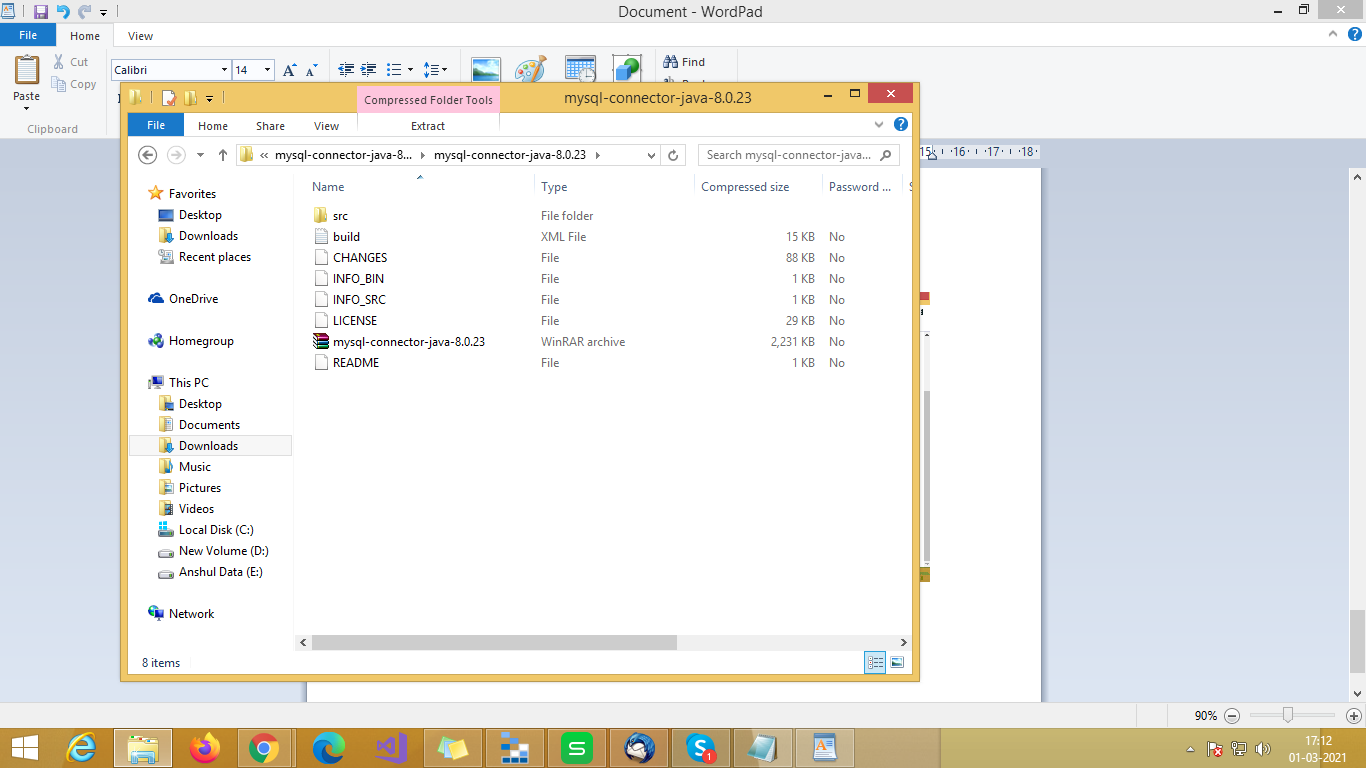
**Connect Sql database**

1. first You need to install jdbc driver of mysql in cluster

<https://dev.mysql.com/downloads/connector/j/>

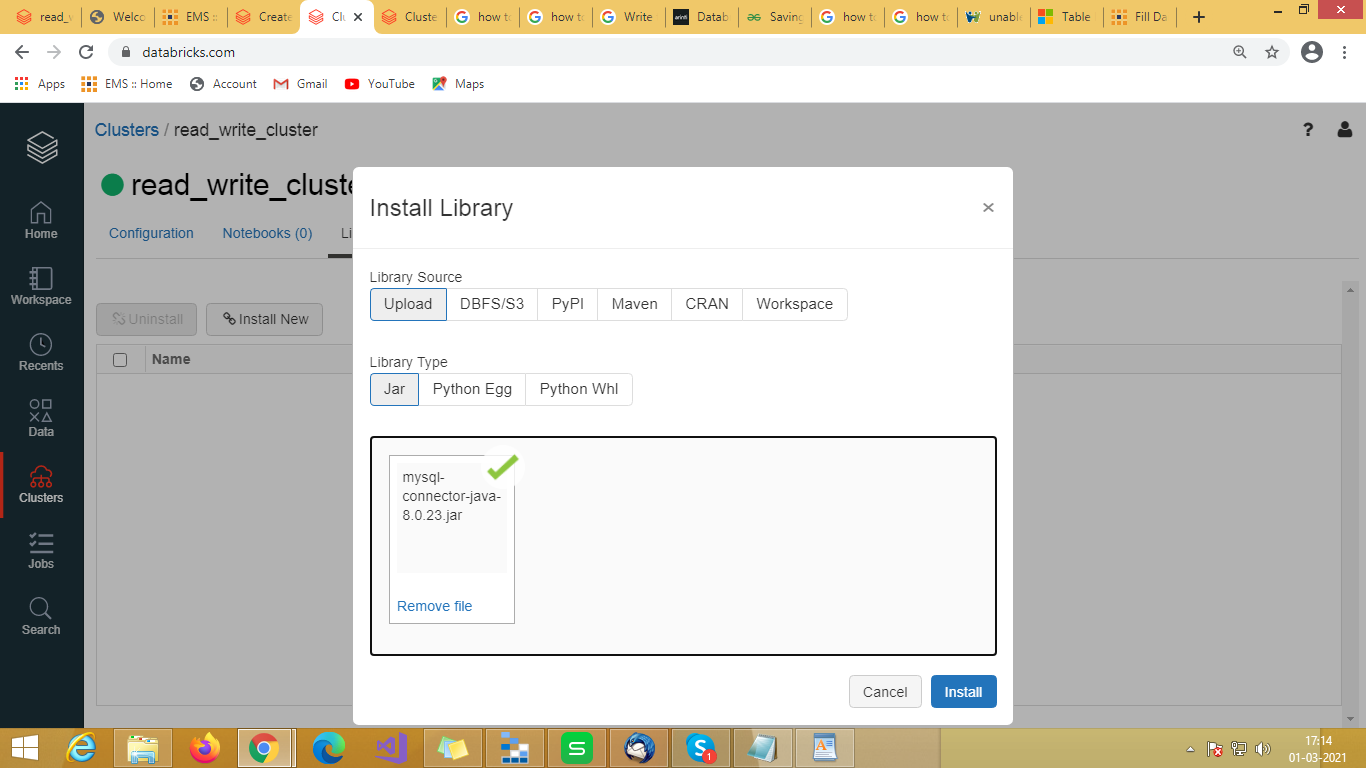
download the selected one

After download Extract the file



an upload the mysql-connector-java-8.0.23.jar file on the cluster

And install it



link : <https://docs.databricks.com/data/data-sources/sql-databases.html>

driver = "com.mysql.jdbc.Driver"

Url = "jdbc:mysql://<- HOSTNAME -->"

table = "DatabaseName.TableName"

UserName = ""

Password = ""

connectionProperties = {

"user" : UserName ,

"password" : Password ,

"driver" : "com.mysql.jdbc.Driver"

}

df = spark.read.format("jdbc")\

.option("driver", driver)\

.option("Url", Url)\

.option("dbtable", table)\

.option("user", UserName)\

.option("Password", Password)\

.load()

display(df)

**For save Table**

df.write.format("delta").saveAsTable("employee")

**For Write table into Sql database**

df = spark.read.format("delta").options(header = "true", inferschema = "true").load("file-path")

from pyspark.sql import \*

df1 = DataFrameWriter(df)

df1.jdbc(Url = Url, table = table, mode = "overwrite" properties = connectionProperties )

**Connection with sql server**

jdbcHostname = "ga-darwinsync-dev-warehouse.database.windows.net"

jdbcDatabase = "darwinsync-dev"

jdbcPort = 1433

jdbcUsername = "darwinsync\_dev"

jdbcPassword= "GA123!@#"

connectionProperti = {

"user" : jdbcUsername,

"password" : jdbcPassword,

"driver" : "com.microsoft.sqlserver.jdbc.SQLServerDriver"

}

jdbcUrl = "jdbc:sqlserver://{0}:{1};database={2}".format(jdbcHostname, jdbcPort, jdbcDatabase)

**Write data into sql server table**

df1 = DataFrameWriter(changedTypedf)

df1.jdbc(url = jdbcUrl, table = "demokkd", mode = "overwrite", properties = connectionProperti )

**Connection Between Blob Storage & DataBricks**

<https://docs.databricks.com/data/data-sources/azure/azure-storage.html>

containerName = "dataoutput"

storageAccountName = "stdotsquares"

dbutils.fs.mount(

source = "wasbs://containerName @storageAccountName.blob.core.windows.net",

mount\_point = "/mnt/storeData",

extra\_configs = {"fs.azure.sas.containerName .storageAccountName .blob.core.windows.net":"xWzDbS3icvjH1%2FBjbszeAZ0LVa7E9hp2l9OUc9dAa1s%3D"})

**OR**

%scala

val containerName = "dataoutput"

val storageAccountName = "stdotsquares"

val sas = "?sv=2019-12-12&st=2021-03-01T04%3A46%3A05Z&se=2021-03-02T04%3A46%3A05Z&sr=c&sp=racwdl&sig=xWzDbS3icvjH1%2FBjbszeAZ0LVa7E9hp2l9OUc9dAa1s%3D"

val config = "fs.azure.sas." + containerName+ "." + storageAccountName + ".blob.core.windows.net"

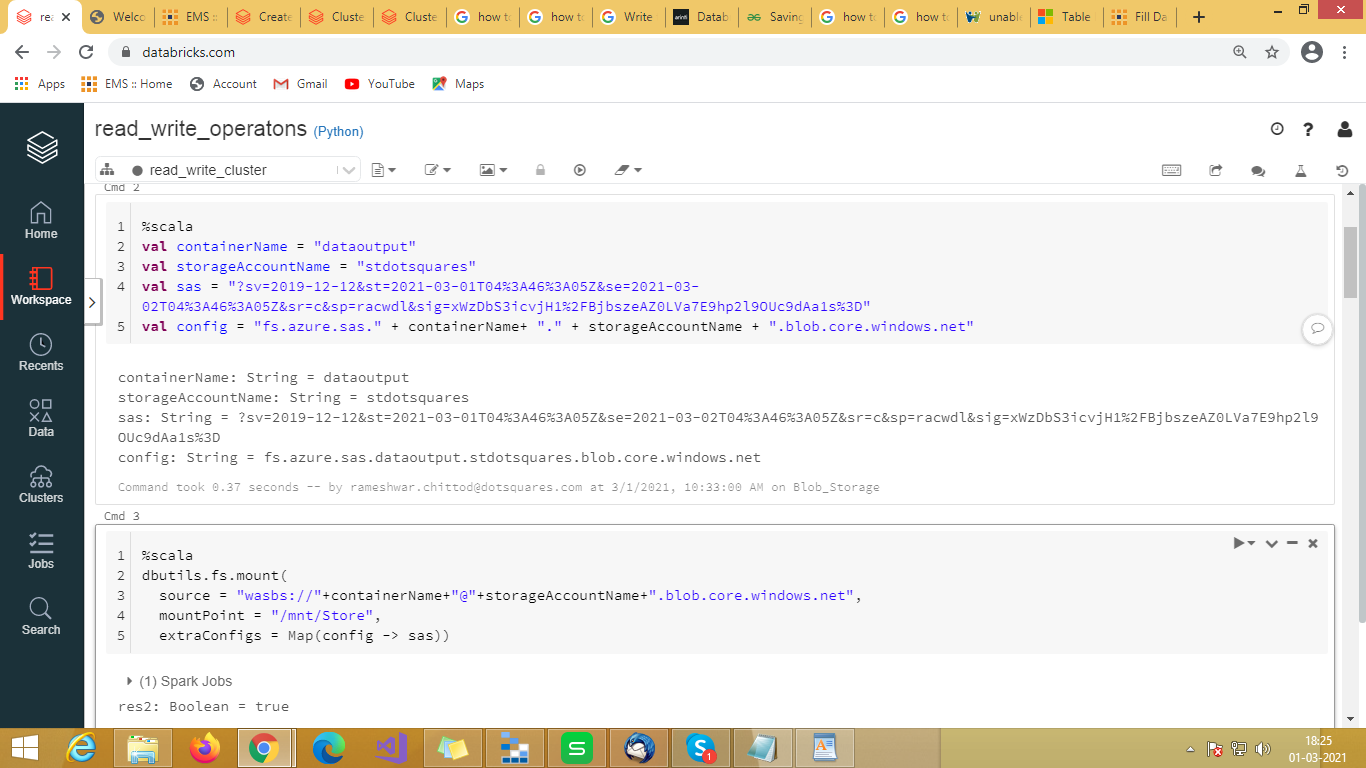
%scala

dbutils.fs.mount(

source = "[wasbs://"+containerName+"@"+storageAccountName+".blob.core.windows.net](mailto:wasbs://"+containerName+"@"+storageAccountName+".blob.core.windows.net)",

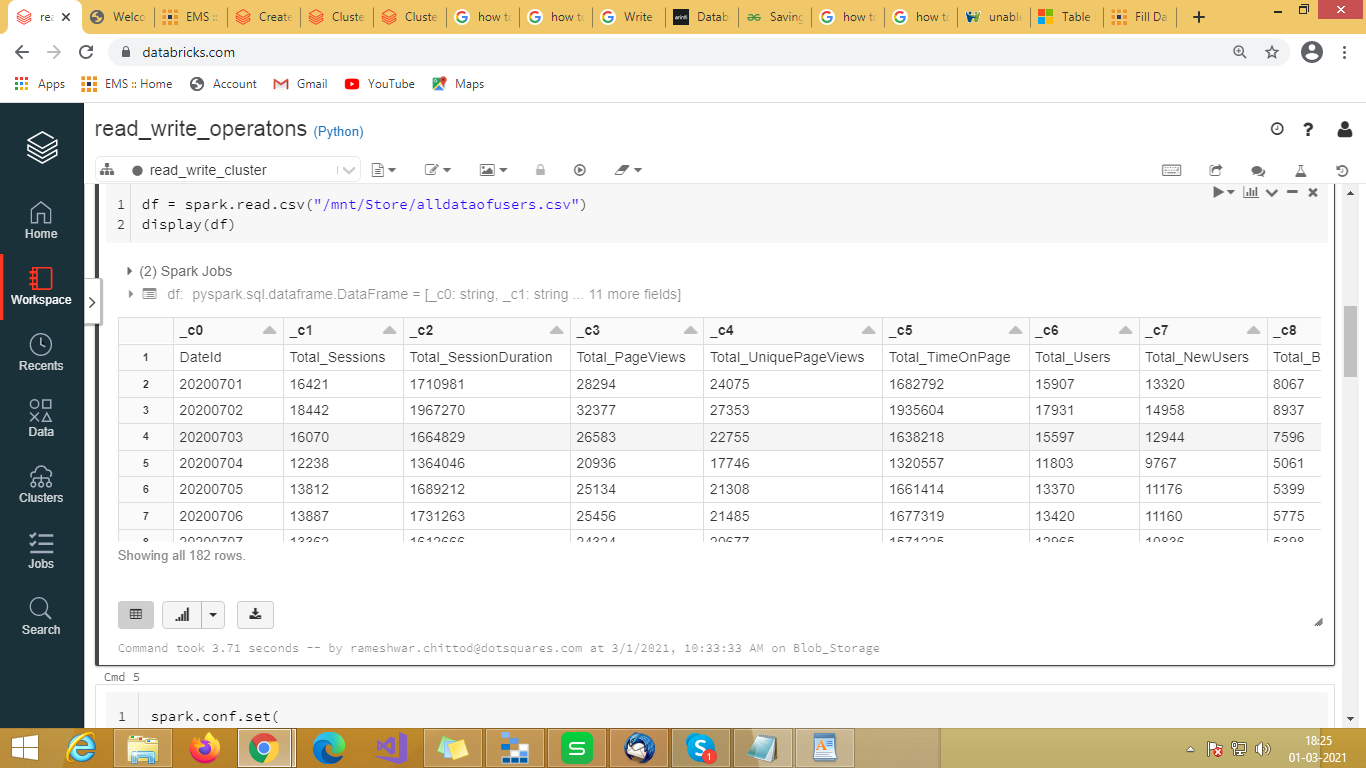
mountPoint = "/mnt/Store",

extraConfigs = Map(config -> sas))



df = spark.read.csv("/mnt/Store/alldataofusers.csv")

display(df)



**For Write in Blob Storage**

**For configuration**

spark.conf.set(

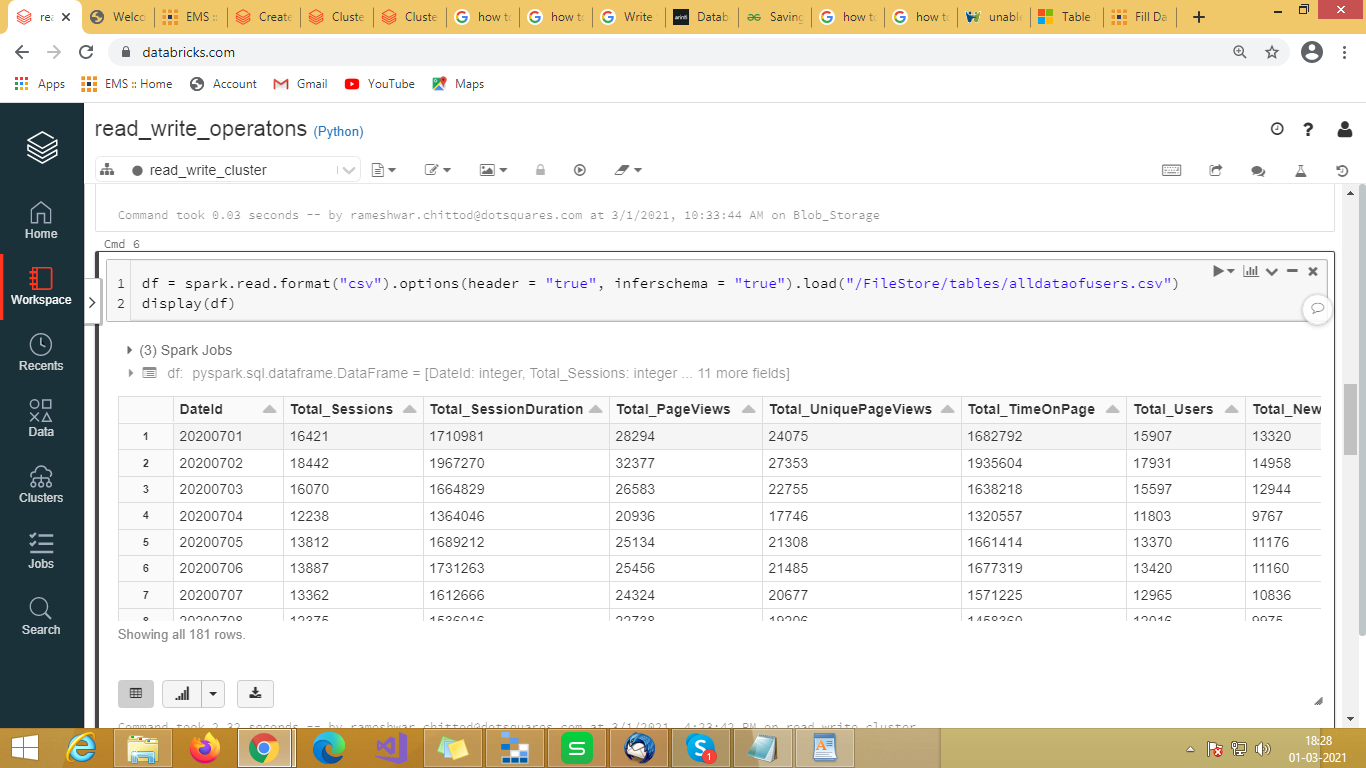
"fs.azure.sas.dataoutput.stdotsquares.blob.core.windows.net",

"xWzDbS3icvjH1%2FBjbszeAZ0LVa7E9hp2l9OUc9dAa1s%3D")

**Read any file from databricks database**

df = spark.read.format("csv").options(header = "true", inferschema = "true").load("/FileStore/tables/alldataofusers.csv")

display(df)



df.write.mode("overwrite").format("csv").options(header = "true", inferschema = "true").save("/mnt/Store/")

**OR**

df.write.mode("append").format("csv").options(header = "true", inferschema = "true").save("/mnt/Store/")

**OR**

you can make a copy of databricks file into blob storage

dbutils.fs.cp('/FileStore/tables/alldataofusers.csv','/mnt/Store/alldataofusers.csv')

**Read Multiple File From Blob Storage**

df = spark.read.csv(mount\_point +"/\*.csv")

**Rename the file that store in blob storage by save method**

%scala

import org.apache.hadoop.fs.\_;

val fs = FileSystem.get(sc.hadoopConfiguration);

val file = fs.globStatus(new Path("/mnt/Store/part-00000\*"))(0).getPath().getName();

fs.rename(new Path("/mnt/Store/" + file), new Path("/mnt/Store/alldataofuserswa.csv"));

**Check How Many file are there**

display(dbutils.fs.ls("dbfs:/mnt/Store/"))

**Remove file From blob Storage by the name**

dbutils.fs.rm("dbfs:/mnt/Store/alldataofusersw.csv")

**Remove Mounting point**

dbutils.fs.unmount("/mnt/Store");

**LINKS**

1. Connection with S3

<https://youtu.be/puwQawwl830>

1. EXTRACT DATA FROM GOOGLE ANALYTICS

<https://youtu.be/UVxkn8Ynbbs>

1. Create SQL Data Warehouse in Azure portal

<https://youtu.be/LixyZ4w_YDs>

1. Integrate Sql data Warehouse with Databricks

<https://youtu.be/U1otyIQhMZc>

1. azure data bricks pipeline

<https://youtu.be/njUiDmUyN6c>

1. call another notebook into notebook

<https://youtu.be/B1DyJScg0-k>

1. Connection with key-vault by using secrete scope of data bricks

<https://youtu.be/geCM32t_VWE>

**or**

[https://community.cloud.databricks.com/o=6361173\*\*\*\*\*\*\*\*#secrets/createScope](https://community.cloud.databricks.com/?o=6361173********" \l "secrets/createScope)

1. Trigger ADF

<https://youtu.be/uF3LOCVFHkw>

1. Cleaning and analyzing data

<https://youtu.be/-tZbkgTnGs4>

1. schedule data bricks notebook through jobs

<https://youtu.be/8e5vkoOblxo>

1. run data bricks jobs by python scripts <https://stackoverflow.com/questions/68868015/is-there-an-example-to-call-rest-api-from-ms-azure-databricks-notebook>