# 

# 

# Wunderflat Technical Challenge Document

### By Sai Venkata krishna Mammahi

# 

| Index | Page Numbers |
| --- | --- |
| SQL Solutions | 2-5 |
| Python Solutions | 6-14 |

# 

# 

# 

# 

# 

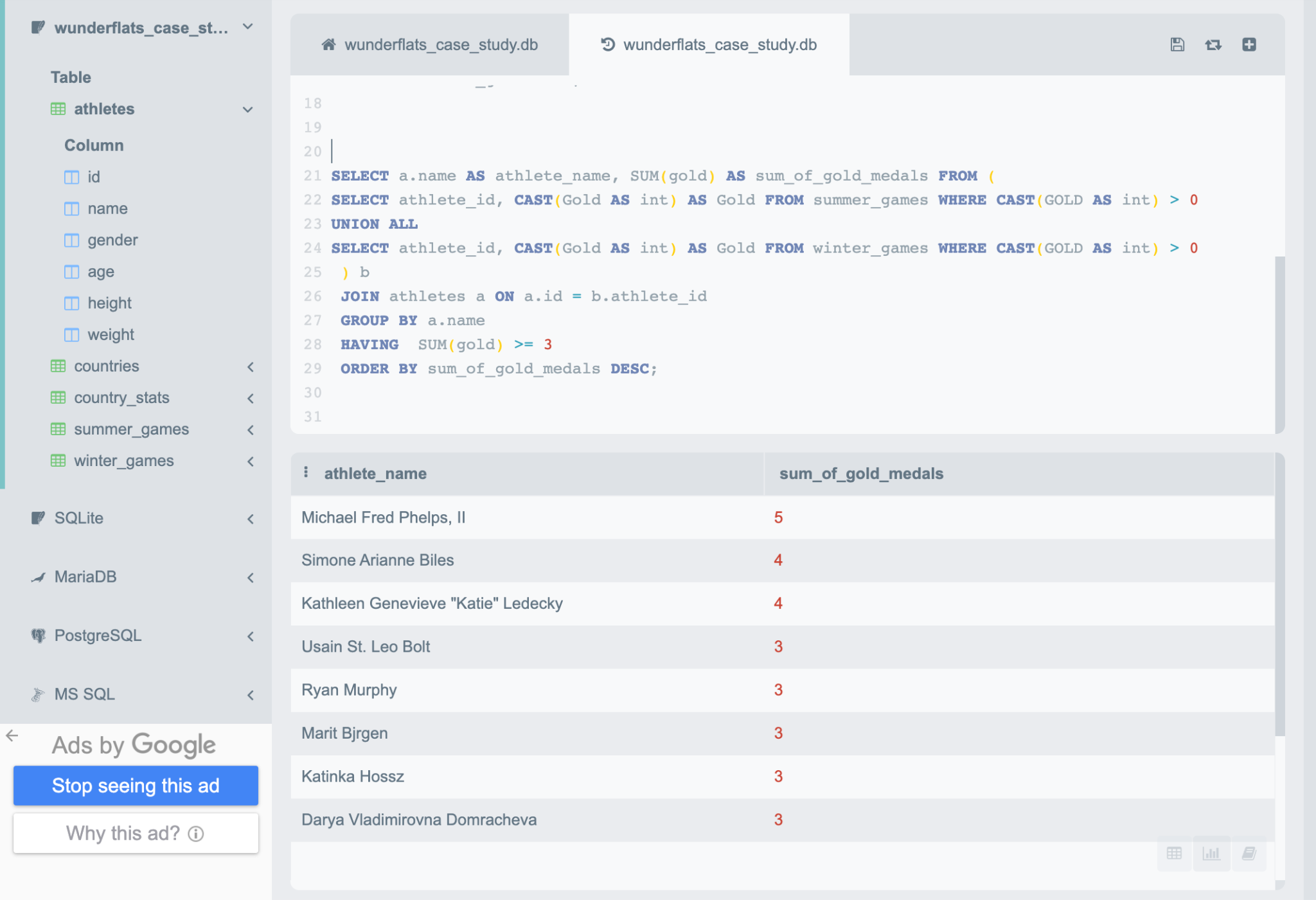
# SQL Questions:

1. **We want to list the names and gold medal count of all athletes with three or more gold medals over all games, starting with the ones with the most gold medals.**

**Solution 1:**

| select a.name as athlete\_name, SUM(gold) as sum\_of\_gold\_medals  from ( select athlete\_id, cast(Gold as int) as Gold from summer\_games  where cast(GOLD as int) > 0 union all select athlete\_id, cast(Gold as int) as Gold from winter\_games  where cast(GOLD as int) > 0 ) b join athletes a on a.id = b.athlete\_id group by a.name HAVING SUM(gold) >= 3 ORDER BY sum\_of\_gold\_medals DESC; |
| --- |

**Screenshot 1:**



**Explanation:**

“**three or more gold medals over all games**” Based on this statement i have filtered the athletes who won at least one or more gold medals from both summer and winter games and i did “UNION ALL” because we want both datasets data.

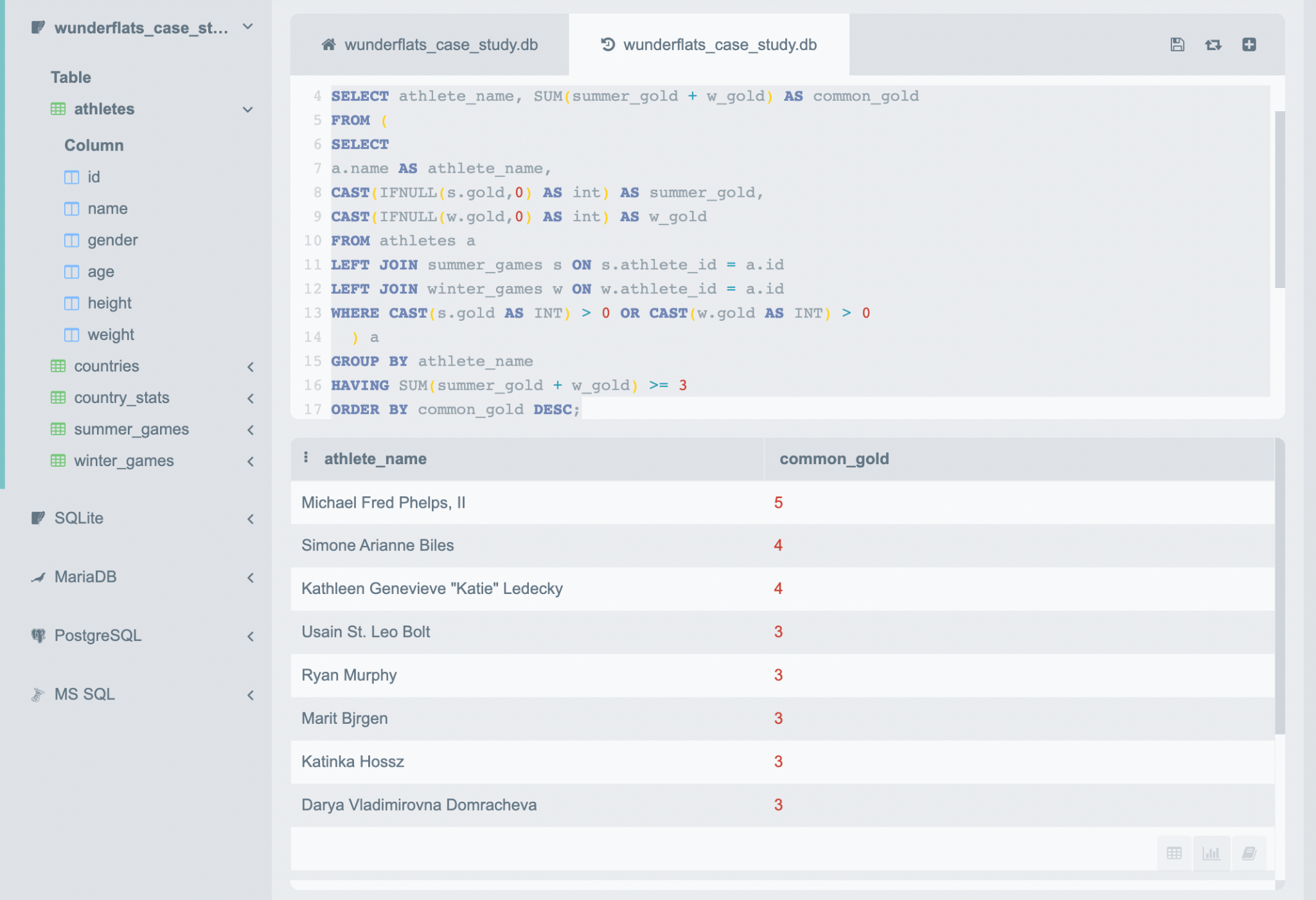
Then joined above data set with athlete table as per question “**We want to list the names and gold medal count of all athletes**”, So we require athlete name and his gold medal count, So from athletes table i got name, and from the above query i got gold medal data, and i summed up gold medal data by grouped on athlete name column and filtered the gold medal count should be 3 or more than 3.

As per “**starting with the ones with the most gold medal**” statement, we require data should be in DESC order based on gold\_medal count, so performed order by operation and sorted the data in DESC order using gold\_medal count column.

**Solution 2 :**

| **SELECT athlete\_name, SUM(summer\_gold + w\_gold) as common\_gold  from ( SELECT  a.name as athlete\_name, a.id, s.athlete\_id, w.athlete\_id, cast(IFNULL(s.gold,0) as int) as summer\_gold,  cast(IFNULL(w.gold,0) as int) as w\_gold from athletes a LEFT JOIN summer\_games s on s.athlete\_id = a.id LEFT JOIN winter\_games w on w.athlete\_id = a.id WHERE CAST(s.gold as INT) > 0 OR CAST(w.gold as INT) > 0   ) a GROUP BY athlete\_name HAVING SUM(summer\_gold + w\_gold) >= 3 ORDER BY common\_gold DESC;** |
| --- |

**Screen shot 2:**



**Explanation 2:**

I joined the athlete table with summer games and winter games to get the gold medal data from both tables. As you can see I performed left join on both tables because there are scenarios where athletes got gold medals in only one of the games, so if we perform inner join here then we will lose the gold medal data, due to that I have considered left join here.

In the Outer Query, added both medal count and did the sum aggregation on medal count and grouped the data by athlete name and filtered the data as per given condition where the athlete should have >=3 gold medals, and sorted the data in DESC order based on gold\_medal\_count.

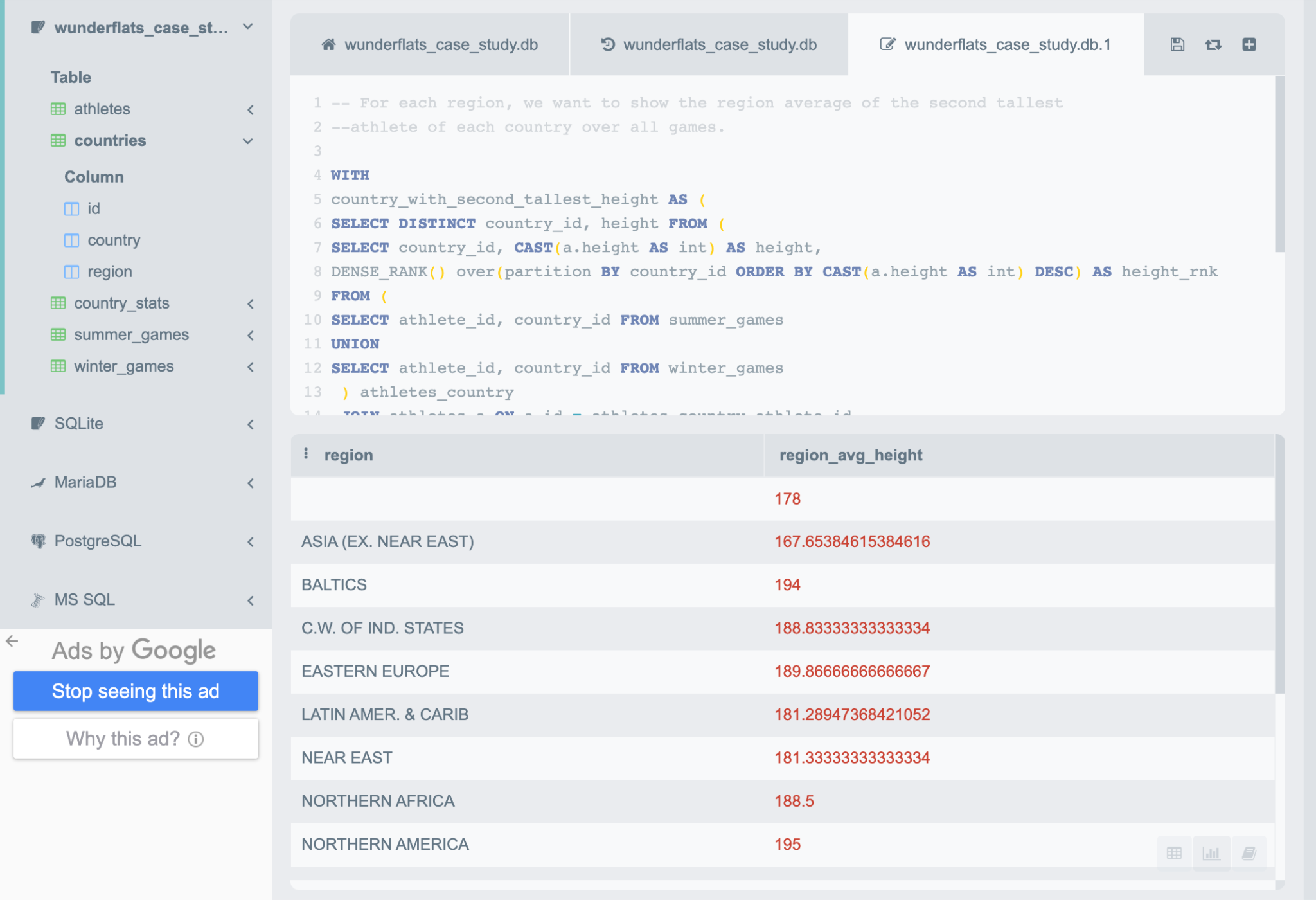
—------------------------------------------------------------------------------------------------------------------------

1. **For each region, we want to show the region average of the second tallest athlete of each country over all games.**

**Solution:**

| with  country\_with\_second\_tallest\_height as ( SELECT distinct country\_id, height FROM ( SELECT country\_id, cast(a.height as int) as height,  DENSE\_RANK() over(partition by country\_id order by cast(a.height as int) DESC) as height\_rnk from ( SELECT athlete\_id, country\_id from summer\_games UNION  SELECT athlete\_id, country\_id from winter\_games  ) athletes\_country   JOIN athletes a on a.id = athletes\_country.athlete\_id  ) country\_height   WHERE height\_rnk=2  )    SELECT region, AVG(height) as region\_avg\_height  FROM (  SELECT region,country, b.country\_id, b.height   from countries a  join country\_with\_second\_tallest\_height b on a.id = b.country\_id  order by region  )  GROUP BY region  ORDER BY region |
| --- |

**Screen shot :**



**Explanation:**

I have used CTE (Common Table Expression) for better readability.

As per “**the second tallest athlete of each country over all games”** Statement, I have taken athlete\_id and country\_id from both summer and winter games and i have just unioned the data, because we don’t require redundant data of athlete and his country.

Then i have joined above dataset with athlete to get the height of athletes, and then i have performed DENSE\_RANK() window function to get the ordered heights of each country\_id’s athlete. Once we got the height ranks, then i filtered the height\_rnk=2 as we require second tallest height of each country\_id.

Based on “**For each region, we want to show the region average”** Statement**,** In the last query, I have joined above dataset with countries table to get region,country and second\_tallest height info, once we got the region and heights info, i have performed the AVG() aggregation operation to get the avg height for each region.

# Python Solutions :

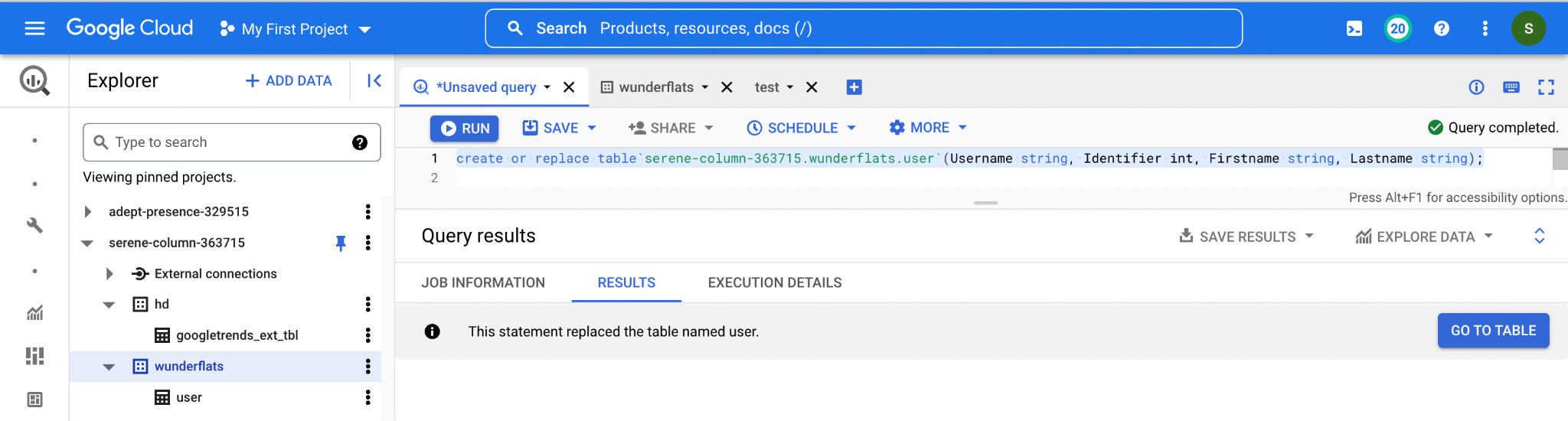
**Config 1 : config.yaml**

| file\_path : '/Users/saim/Documents/wunderflats/Data/username.csv'  file\_delimiter : ';'  bigquery\_table\_name : 'serene-column-363715.wunderflats.user'  google\_auth\_file\_path : '/Users/saim/Documents/wunderflats/Code/google\_auth.json' |
| --- |

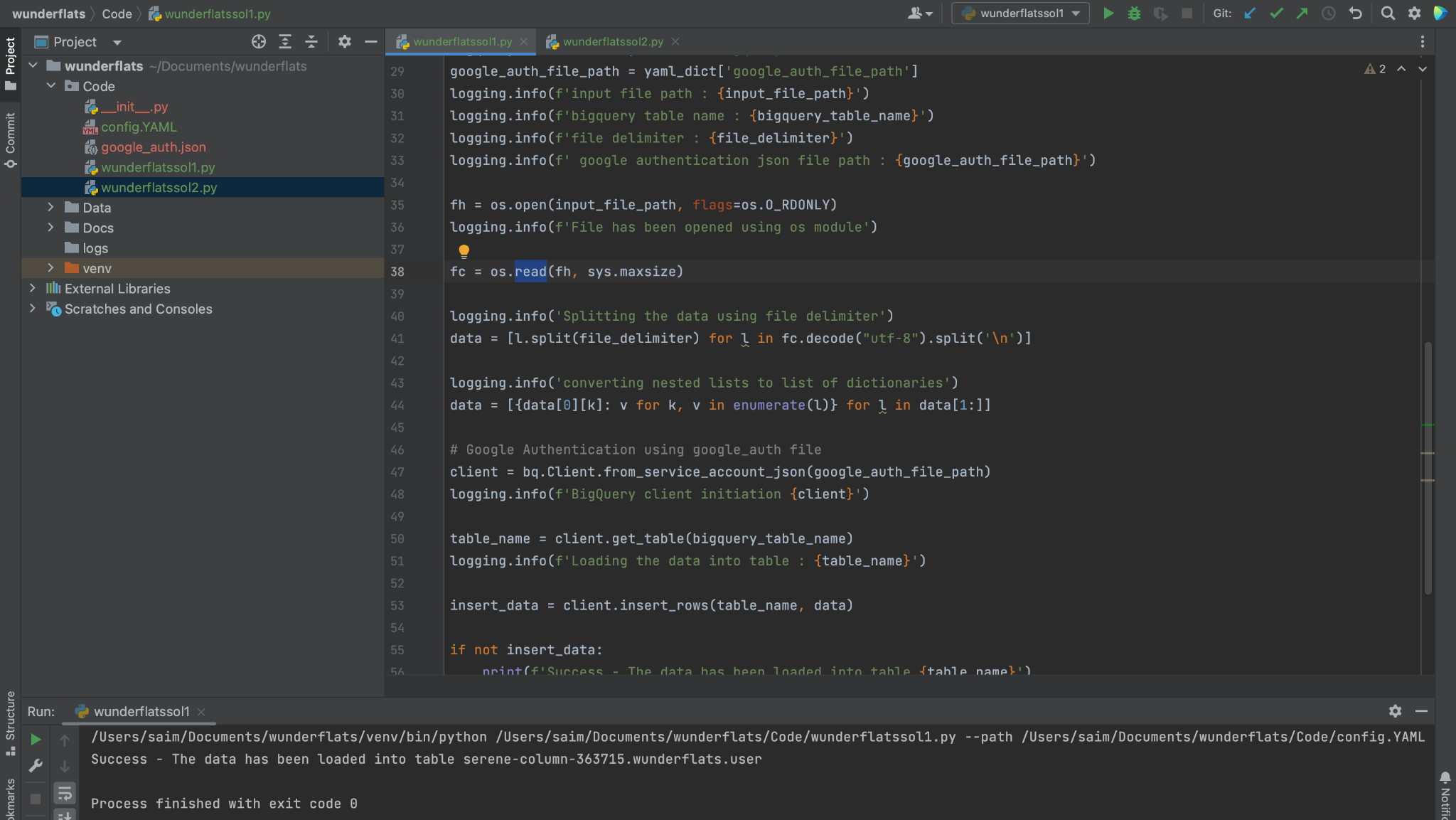
**Python Script 1: wunderflatssol1.py**

| **import os import yaml import google.cloud.bigquery as bq import argparse import logging import sys  # Create the parser parser = argparse.ArgumentParser(description='File path for the config File')  # Add the arguments parser.add\_argument('--path', type=str, required=True)  # Parsing the arguments args = parser.parse\_args()  # assigning the path argument to config\_path variable config\_path = args.path logging.info(f'YAML config file path {config\_path}')  # reading the yaml config file for the parameters with open(config\_path, 'r+') as stream:  yaml\_dict = yaml.safe\_load(stream=stream)  # assigning the config values to variable input\_file\_path = yaml\_dict['file\_path'] file\_delimiter = yaml\_dict['file\_delimiter'] bigquery\_table\_name = yaml\_dict['bigquery\_table\_name'] google\_auth\_file\_path = yaml\_dict['google\_auth\_file\_path'] logging.info(f'input file path : {input\_file\_path}') logging.info(f'bigquery table name : {bigquery\_table\_name}') logging.info(f'file delimiter : {file\_delimiter}') logging.info(f' google authentication json file path : {google\_auth\_file\_path}')  fh = os.open(input\_file\_path, flags=os.O\_RDONLY) logging.info(f'File has been opened using os module')  fc = os.read(fh, sys.maxsize)  logging.info('Splitting the data using file delimiter') data = [l.split(file\_delimiter) for l in fc.decode("utf-8").split('\n')]  logging.info('converting nested lists to list of dictionaries') data = [{data[0][k]: v for k, v in enumerate(l)} for l in data[1:]]  # Google Authentication using google\_auth file client = bq.Client.from\_service\_account\_json(google\_auth\_file\_path) logging.info(f'BigQuery client initiation {client}')  table\_name = client.get\_table(bigquery\_table\_name) logging.info(f'Loading the data into table : {table\_name}')  insert\_data = client.insert\_rows(table\_name, data)  if not insert\_data:  print(f'Success - The data has been loaded into table {table\_name}') else:  print("Failed - There is an issue with data loading, please look into it")** |
| --- |

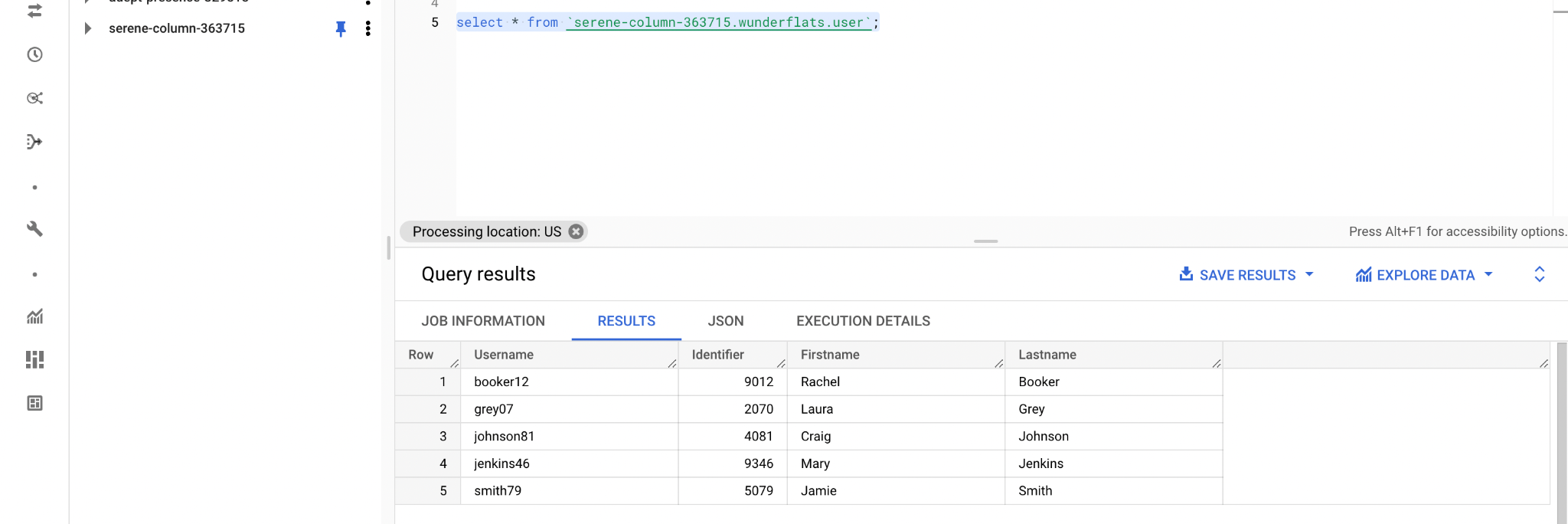
**Created a empty table in BigQuery: user**

****

**Project screen shot 1 :**

****

I have ran the program using pycharm, it ran successfully and data has been loaded in the table.

****

## **Python Code Alteration/Explanation/Assumptions:**

**Initial Code:**

| import os import google.cloud.bigquery as bq  fh = os.open('myfile.csv') fc = fh.read()  data = [l.split(",") for l in fc.split('\n')]  data = [{data[0][k]: v for k, v in enumerate(l)} for l in data[1:]]  c = bq.Client() t = c.get\_table("myproject.mydataset.mytable") e = c.insert\_rows(t, data) |
| --- |

1. Placing hard code values in proagramming languages is not a good practice, and also passing multiple arguments from shell or cron job is also not recomended.

Due to that i have created a “config.YAML” file, that contains all the values, and these values will be replacing the hard coded values which are mentioned in the above code.

**config.yaml**

| file\_path : '/Users/saim/Documents/wunderflats/Data/username.csv' file\_delimiter : ';' bigquery\_table\_name : 'serene-column-363715.wunderflats.user' google\_auth\_file\_path : '/Users/saim/Documents/wunderflats/Code/google\_auth.json' |
| --- |

1. As we discussed in the 1st point, passing the multiple arguments from cron job or command prompt/shell is also not recommended.

To overcome above constraint, i have designed the code such a way that we will pass only once argument from crontab such as “**--PATH <YAML FILE PATH>**”.

To parse the crontab python argument i have used argparse library, as it is recommended to parse python arguments rather than using sys.arg library.

| **import** os **import** yaml **import** google.cloud.bigquery **as** bq **import** argparse **import** logging **import** sys  *# Create the parser* parser = argparse.ArgumentParser(description='File path for the config File')  *# Add the arguments* parser.add\_argument('--path', type=str, required=**True**)  *# Parsing the arguments* args = parser.parse\_args()  *# assigning the path argument to config\_path variable* config\_path = args.path |
| --- |

1. In any code development implementation of logging would be more helpful in the debug the issues and tracking the code running status.

So i have used logging library to log the progress of code, which was missing in the initial code.

| **import** logging logging.info(f'input file path : {input\_file\_path}') logging.info(f'bigquery table name : {bigquery\_table\_name}') logging.info(f'file delimiter : {file\_delimiter}') logging.info(f' google authentication json file path : {google\_auth\_file\_path}') |
| --- |

1. Using the below code, i have extracted the values from config file and assign these values to replace hard coded values in the initial code.

| *# reading the yaml config file for the parameters* **with** open(config\_path, 'r+') **as** stream:  yaml\_dict = yaml.safe\_load(stream=stream)  *# assigning the config values to variable* input\_file\_path = yaml\_dict['file\_path'] file\_delimiter = yaml\_dict['file\_delimiter'] bigquery\_table\_name = yaml\_dict['bigquery\_table\_name'] google\_auth\_file\_path = yaml\_dict['google\_auth\_file\_path'] logging.info(f'input file path : {input\_file\_path}') logging.info(f'bigquery table name : {bigquery\_table\_name}') logging.info(f'file delimiter : {file\_delimiter}') logging.info(f' google authentication json file path : {google\_auth\_file\_path}') |
| --- |

1. The issue with current code is “**The os.open will always require 2 arguments i.e. File path and flags**”. Here flags represents mode of file operation, in my code i have considered os.RDONLY ( read only) as we are not performing any write operations on file.

So changed below code

fh = os.open('myfile.csv')

As this

| fh = os.open(input\_file\_path, flags=os.O\_RDONLY) logging.info(f'File has been opened using os module') |
| --- |

1. The below code in the initial code will work work as we can read the file with file open variable. As above “fh” variable will return int value. So i have read the file using the os module only. As os.read will take 2 arguments one is file open variable, and byte size as it reads and stores the data in bytes, so i specified sys.maxsize as byte number as file size is not static some times we will get bigger file.

So changed below code

| fc = fh.read() |
| --- |

As this

| fc = os.read(fh, sys.maxsize) |
| --- |

1. So in the 6th step, we are storing the data in bytes so i have tweaked the following little bit as we can not use split function on byte data, so i have decoded the data from byte to string using ‘UTF-8’

So changed below code

| data = [l.split(",") **for** l **in** fc.split('\n')] |
| --- |

As this

| data = [l.split(file\_delimiter) **for** l **in** fc.decode("utf-8").split('\n')] |
| --- |

1. I haven’t changed the following snippet

| data = [{data[0][k]: v **for** k, v **in** enumerate(l)} **for** l **in** data[1:]] |
| --- |

1. c = bq.Client() only works when have already added the google\_authentication file to environmental variables.

So, it might be an issue that in virtual environment, we haven’t added the google authentication file to environamental variables, so it could also be one of the errors.

So i have replaced the following code with my code, and i am creating a bigquery client using google\_auth file.

So changed below code

| c = bq.Client() |
| --- |

As this

| *# Google Authentication using google\_auth file* client = bq.Client.from\_service\_account\_json(google\_auth\_file\_path) logging.info(f'BigQuery client initiation {client}') |
| --- |

1. I havent changed the following code snippets.

| table\_name = client.get\_table(bigquery\_table\_name) logging.info(f'Loading the data into table : {table\_name}')  insert\_data = client.insert\_rows(table\_name, data) |
| --- |

1. I added the following code snippet to understand the status of data load.

| **if** **not** insert\_data:  print(f'Success - The data has been loaded into table {table\_name}') **else**:  print("Failed - There is an issue with data loading, please look into it") |
| --- |

1. There might be a chance, the virtual machine where we are running the above code doen’t have the python libraries so while handovering codebase to a colleague i will ask him to ensure the libraries are available in the virtual machine, if not i will ask him to run following commands before scheduling the above task in crontab.

Instructions :

| # Install following libraries in the virtual machine  pip install pandas numpy google-cloud-bigquery  # Open config.yaml file in virtual machine and replace the configuration values with actual values.  vi <path to config.yaml>/config.YAML   * Press i * replace the values * Press esc, and then type :wq!   # To schedule the python code in cron  type “crontab -e”   * Press i * Add following line, change cron scheduling and paths accordingly   \* \* \* \* \* cd <path to project> && $(where python3) <path to wunderflatssol1.py> —-path <path to YAML file> >> <mention the log file path>   * Press esc, and then type :wq! |
| --- |
| # Then go and check the values in BigQuery |

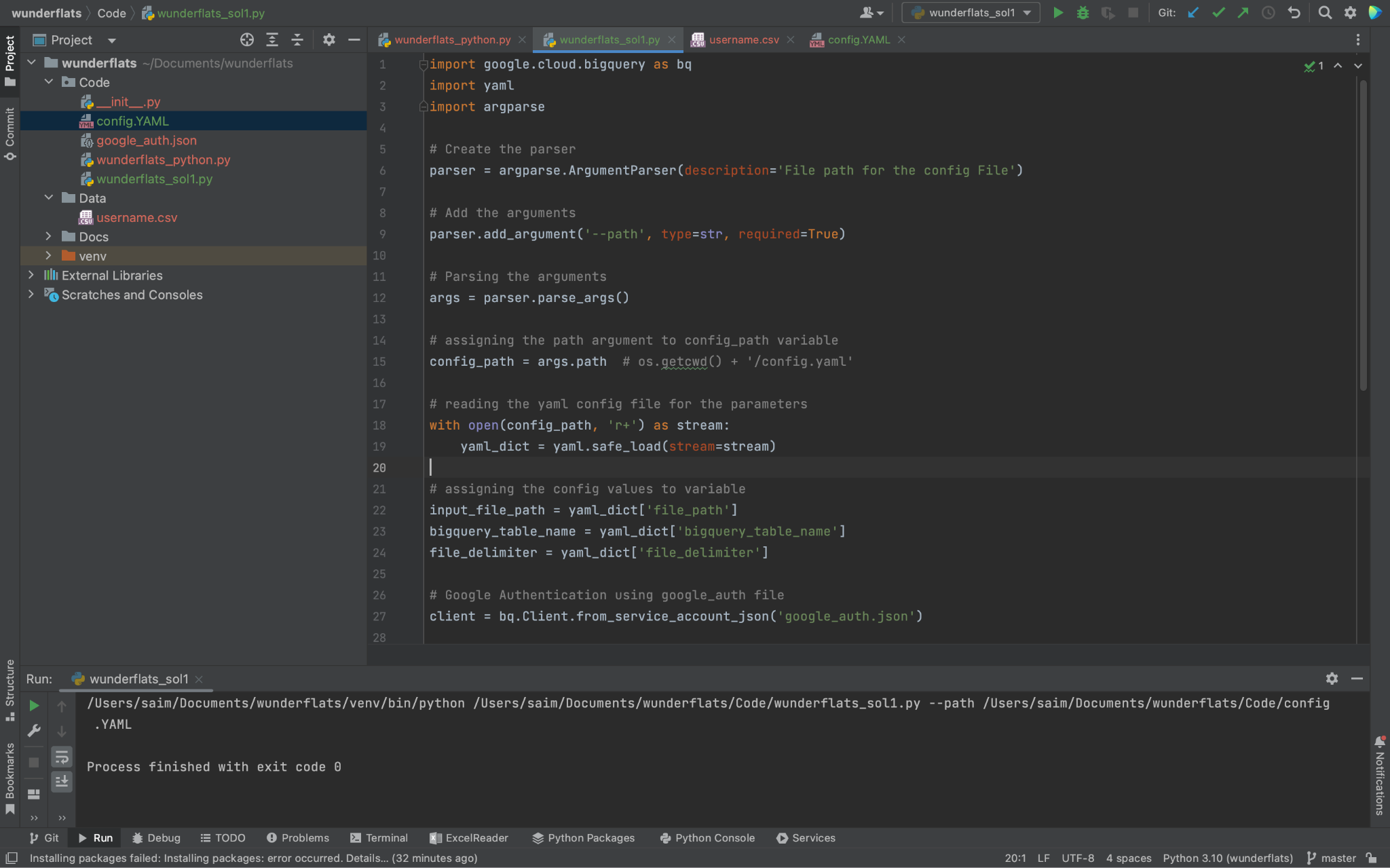
## 

## **Anothe Solution for Loading the data from Csv to BigQuery**

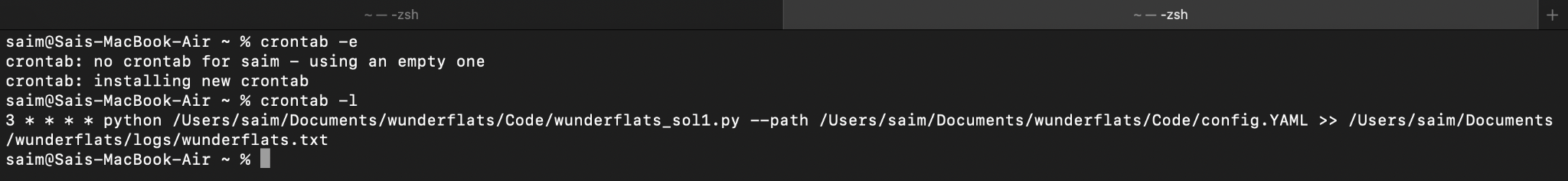
**Python Script : wunderflats1.py**

| import google.cloud.bigquery as bq  import yaml  import argparse  # Create the parser  parser = argparse.ArgumentParser(description='File path for the config File')  # Add the arguments  parser.add\_argument('--path', type=str, required=True)  # Parsing the arguments  args = parser.parse\_args()  # assigning the path argument to config\_path variable  config\_path = args.path # os.getcwd() + '/config.yaml'  # reading the yaml config file for the parameters  with open(config\_path, 'r+') as stream:  yaml\_dict = yaml.safe\_load(stream=stream)  # assigning the config values to variable  input\_file\_path = yaml\_dict['file\_path']  bigquery\_table\_name = yaml\_dict['bigquery\_table\_name']  file\_delimiter = yaml\_dict['file\_delimiter']  google\_auth\_file\_path = yaml\_dict['google\_auth\_file\_path']  # Google Authentication using google\_auth file  client = bq.Client.from\_service\_account\_json(google\_auth\_file\_path)  # Loading the configurations for data load  job\_config = bq.LoadJobConfig(  source\_format=bq.SourceFormat.CSV,  skip\_leading\_rows=1,  autodetect=True,  write\_disposition=bq.WriteDisposition.WRITE\_APPEND,  field\_delimiter=file\_delimiter,  allow\_quoted\_newlines=True  )  # Streaming the CSV data to BigQuery table using load\_table\_from\_file API  with open(file=input\_file\_path, mode='rb') as stream:  job = client.load\_table\_from\_file(stream, bigquery\_table\_name, job\_config=job\_config)  table = client.get\_table(bigquery\_table\_name)  print(f'Loaded num of rows {table.num\_rows} and columns {table.schema} and table name {bigquery\_table\_name}') |
| --- |

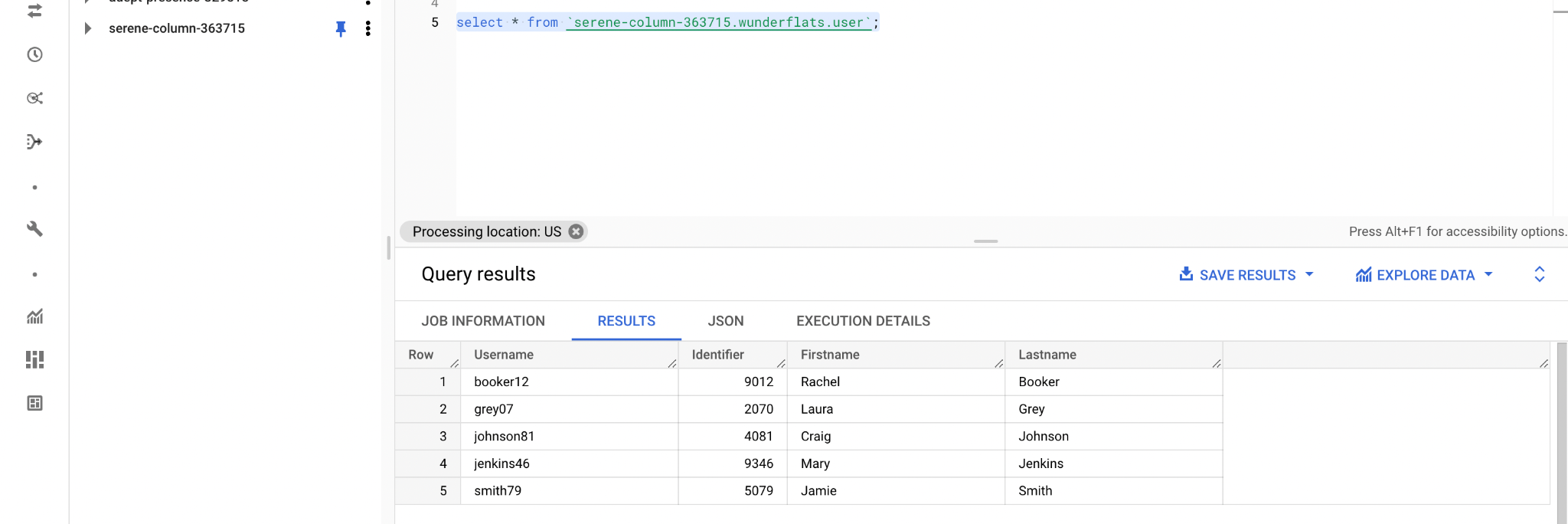
**Project screen shot 1 :**



**Testing the code with cron job :**

****

**After Cron job run:**

****