

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:

The screenshot shows a database management tool interface. On the left, a sidebar lists tables: athletes, countries, country_stats, summer_games, and winter_games. The main area displays the SQL query from the solution, with line numbers 18 to 31. Below the query, the results are shown in a table with two columns: athlete_name and sum_of_gold_medals. The results list athletes with 5, 4, 4, 3, 3, 3, 3, and 3 gold medals respectively.

athlete_name	sum_of_gold_medals
Michael Fred Phelps, II	5
Simone Arianne Biles	4
Kathleen Genevieve "Katie" Ledecky	4
Usain St. Leo Bolt	3
Ryan Murphy	3
Marit Bjergen	3
Katinka Hossz	3
Darya Vladimirovna Domracheva	3

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:



The screenshot shows a database interface with a table list on the left and a query editor on the right. The table list includes 'athletes' and 'summer_games'. The query editor shows the same SQL query as in the previous block. Below the query editor, the results of the query are displayed in a table format.

athlete_name	common_gold
Michael Fred Phelps, II	5
Simone Arianne Biles	4
Kathleen Genevieve "Katie" Ledecky	4
Usain St. Leo Bolt	3
Ryan Murphy	3
Marit Bjergen	3
Katinka Hossz	3
Darya Vladimirovna Domracheva	3

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.

2. 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 :

The screenshot shows a SQL IDE interface. On the left, a sidebar lists tables (athletes, countries, country_stats, summer_games, winter_games) and columns (id, country, region). The main area displays a SQL query and its results.

SQL Query:

```
1 -- For each region, we want to show the region average of the second tallest
2 --athlete of each country over all games.
3
4 WITH
5 country_with_second_tallest_height AS (
6 SELECT DISTINCT country_id, height FROM (
7 SELECT country_id, CAST(a.height AS int) AS height,
8 DENSE_RANK() over(partition BY country_id ORDER BY CAST(a.height AS int) DESC) AS height_rnk
9 FROM (
10 SELECT athlete_id, country_id FROM summer_games
11 UNION
12 SELECT athlete_id, country_id FROM winter_games
13 ) athletes_country
14 JOIN athletes ON a.id = athletes.country_athlete_id
```

Query Results:

region	region_avg_height
	178
ASIA (EX. NEAR EAST)	167.65384615384616
BALTICS	194
C.W. OF IND. STATES	188.83333333333334
EASTERN EUROPE	189.86666666666667
LATIN AMER. & CARIB	181.28947368421052
NEAR EAST	181.33333333333334
NORTHERN AFRICA	188.5
NORTHERN AMERICA	195

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 athletes 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(1)} 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")

```

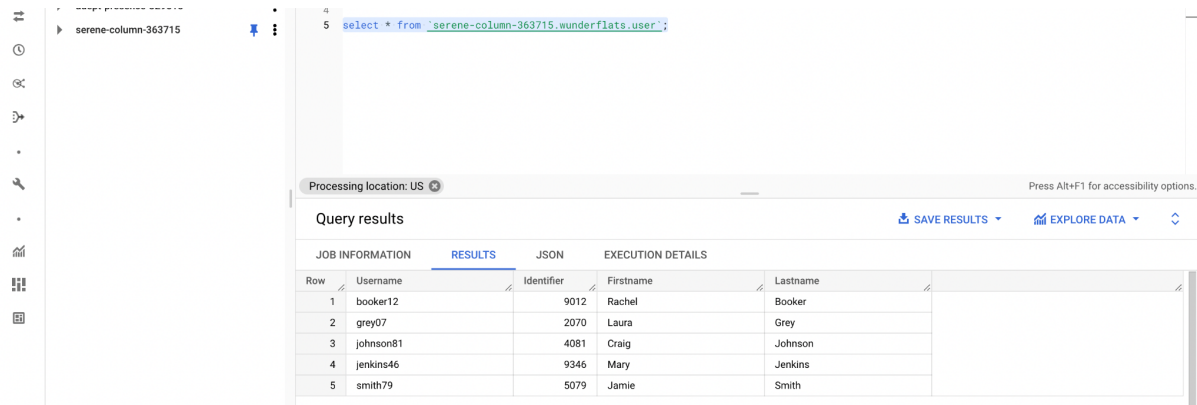
Created a empty table in BigQuery: user

The screenshot shows the Google Cloud console interface. On the left, the 'Explorer' pane shows the project structure with 'wunderflats' selected. The main area displays a query execution window. The query is: `create or replace table `serene-column-363715.wunderflats.user` (Username string, Identifier int, Firstname string, Lastname string);`. The query results section shows a message: 'This statement replaced the table named user.' and a 'GO TO TABLE' button.

Project screen shot 1 :

The screenshot shows a code editor with the file 'wunderflats.py'. The code is identical to the first block. The console output at the bottom shows the successful execution of the script, with the message: 'Success - The data has been loaded into table serene-column-363715.wunderflats.user'.

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



The screenshot shows the PyCharm IDE with a SQL query editor and the BigQuery results pane. The query is `select * from `serene-column-363715.wunderflats.user`;`. The results pane shows a table with 5 rows and 5 columns: Row, Username, Identifier, Firstname, and Lastname.

Row	Username	Identifier	Firstname	Lastname
1	booker12	9012	Rachel	Booker
2	grey07	2070	Laura	Grey
3	johnson81	4081	Craig	Johnson
4	jenkins46	9346	Mary	Jenkins
5	smith79	5079	Jamie	Smith

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 programming languages is not a good practice, and also passing multiple arguments from shell or cron job is also not recommended.

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'
```


2. 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
```

3. 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}')
```

4. 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}')
```

5. 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')
```

6. 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)
```

7. 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')]
```

8. I haven't changed the following snippet

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

9. `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}')
```

10. 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)
```

11. 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")
```

12. There might be a chance, the virtual machine where we are running the above code doesn'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)

# 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 :

```

1 import google.cloud.bigquery as bq
2 import yaml
3 import argparse
4
5 # Create the parser
6 parser = argparse.ArgumentParser(description='File path for the config File')
7
8 # Add the arguments
9 parser.add_argument('--path', type=str, required=True)
10
11 # Parsing the arguments
12 args = parser.parse_args()
13
14 # assigning the path argument to config_path variable
15 config_path = args.path # os.getcwd() + '/config.yaml'
16
17 # reading the yaml config file for the parameters
18 with open(config_path, 'r+') as stream:
19     yaml_dict = yaml.safe_load(stream)
20
21 # assigning the config values to variable
22 input_file_path = yaml_dict['file_path']
23 bigquery_table_name = yaml_dict['bigquery_table_name']
24 file_delimiter = yaml_dict['file_delimiter']
25
26 # Google Authentication using google_auth file
27 client = bq.Client.from_service_account_json('google_auth.json')
28

```

Run: wunderflats_sol1

/Users/saim/Documents/wunderflats/venv/bin/python /Users/saim/Documents/wunderflats/Code/wunderflats_sol1.py --path /Users/saim/Documents/wunderflats/Code/config.YAML

Process finished with exit code 0

Testing the code with cron job :

```

saim@Sais-MacBook-Air ~ % crontab -e
crontab: no crontab for saim - using an empty one
crontab: installing new crontab
saim@Sais-MacBook-Air ~ % crontab -l
3 * * * python /Users/saim/Documents/wunderflats/Code/wunderflats_sol1.py --path /Users/saim/Documents/wunderflats/Code/config.YAML >> /Users/saim/Documents/wunderflats/logs/wunderflats.txt
saim@Sais-MacBook-Air ~ %

```

After Cron job run:

serene-column-363715

5 select * from `serene-column-363715.wunderflats_user`;

Processing location: US

Query results

SAVE RESULTS EXPLORE DATA

JOB INFORMATION		RESULTS	JSON	EXECUTION DETAILS	
Row	Username	Identifier	Firstname	Lastname	
1	booker12	9012	Rachel	Booker	
2	grey07	2070	Laura	Grey	
3	johnson81	4081	Craig	Johnson	
4	jenkins46	9346	Mary	Jenkins	
5	smith79	5079	Jamie	Smith	