**Data Analyst and Health Data Fellow Screening Exercise**

**By Jessey Sitima**

**Exercise 1:**  **An SQL query that will display the duplicate records.**

**Query:**

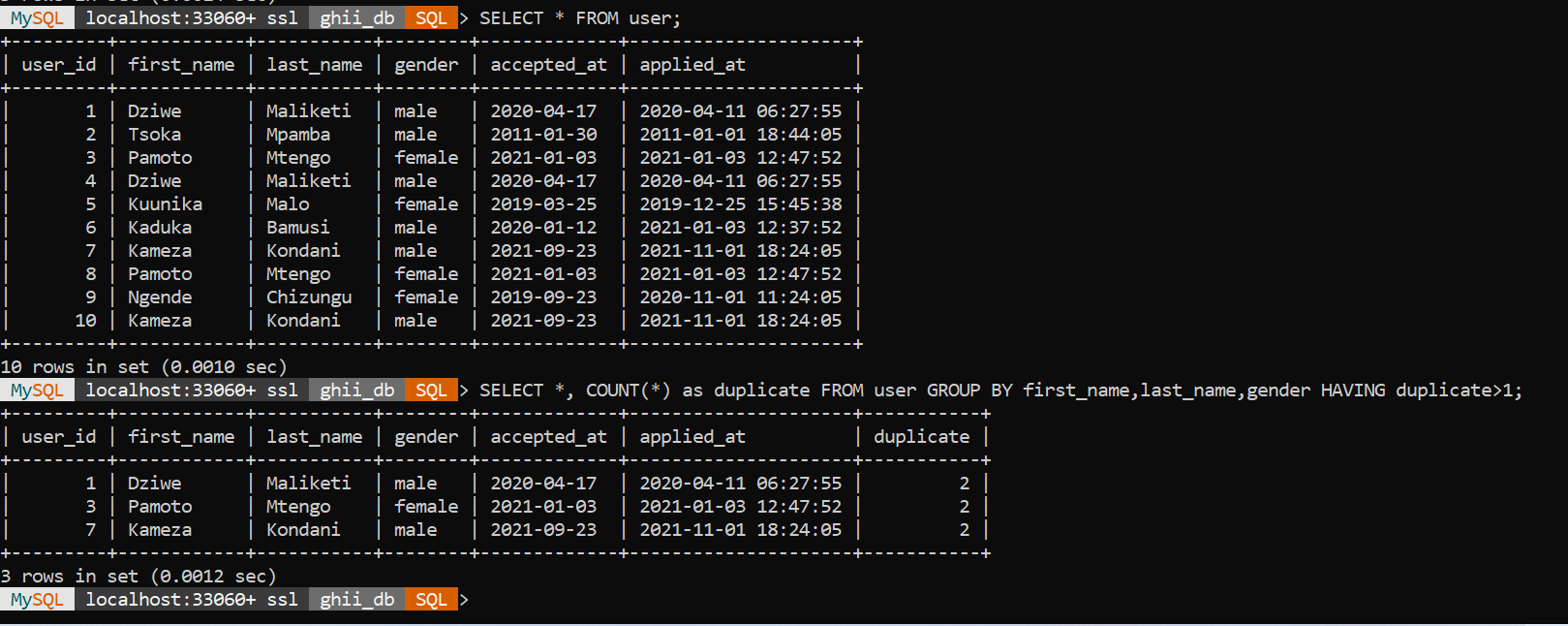
SELECT \*, COUNT(\*) as duplicate FROM user GROUP BY first\_name, last\_name, gender HAVING duplicate>1;

**Explanation:**

This SQL query selects all rows from the "user" table where there are duplicates in the combination of "first\_name," "last\_name," and "gender." It counts these duplicates and only returns rows where the count is greater than 1.

**Solution:**

The screenshot below shows the table before and after running the query;



**Exercise 2:**

1. **Two ways in which the query can be restructured (optimized) to run efficiently**
2. **Use of Common Table Expression (CTE) for the Subquery:**

Instead of using a subquery within the main query's FROM clause, we can rewrite the subquery as a Common Table Expression (CTE). CTEs are often easier to read and maintain, and they sometimes result in better query optimization. Using a CTE can make the query more readable and potentially improve its performance.

1. **Indexing:**

Ensure that the relevant columns used for joining and filtering the data are properly indexed. In this case, we can add indexes to the following columns:

* Index the "person\_id" column in the "Patient\_Visit" table since it's used for grouping and joining.
* Index the "person\_id" column in the "Diagnosis\_Stage" table.
* Proper indexing can significantly improve the query's performance by reducing the time it takes to access and retrieve the required data.

**(b) Query for the solution above:**

Making the Subquery as Common Table Expression CTE:

WITH MaxVisitDates AS (

SELECT person\_id, MAX(visit\_date) AS visit\_date

FROM Patient\_Visit

GROUP BY person\_id )

SELECT pd.person\_id, pd.first\_name, pd.last\_name, mvd.visit\_date

FROM Patient\_Demographic pd

INNER JOIN MaxVisitDates mvd ON pd.person\_id = mvd.person\_id

WHERE pd.person\_id IN (SELECT DISTINCT person\_id FROM Diagnosis\_Stage);

**Exercise 3: Data Duplication**

The below python script uses the panda library for reading data. The python script and the csv supporting docs can be found at <https://github.com/JesseySitima/GHII-Data-analyst-Exercise.git>

import pandas as pd

# task a: Read the CSV file

input\_file = "client\_purchases.csv"

df = pd.read\_csv(input\_file)

# task b: Find and identify duplicate records

duplicates = df[df.duplicated(subset=['First Name', 'Last Name', 'Email', 'Phone', 'Address', 'Zip Code', 'Gender', 'Date of Birth', 'Product', 'Quantity', 'Cost'])]

# task c: Remove the duplicate records

df = df.drop\_duplicates(subset=['First Name', 'Last Name', 'Email', 'Phone', 'Address', 'Zip Code', 'Gender', 'Date of Birth', 'Product', 'Quantity', 'Cost'], keep='first')

# task d: Export the cleaned CSV file

cleaned\_output\_file = "client\_purchases\_deduplicated.csv"

df.to\_csv(cleaned\_output\_file, index=False)

# task e: Identify unique clients and assign unique IDs

unique\_clients = df[['First Name', 'Last Name', 'Email', 'Phone', 'Address', 'Zip Code', 'Gender', 'Date of Birth']].drop\_duplicates().reset\_index(drop=True).reset\_index()

unique\_clients.rename(columns={'index': 'Client ID', 'First Name': 'First Name', 'Last Name': 'Last Name', 'Email': 'Email', 'Phone': 'Phone', 'Address': 'Address', 'Zip Code': 'Zip Code', 'Gender': 'Gender', 'Date of Birth': 'Date of Birth'}, inplace=True)

# Export the unique clients to a CSV file

unique\_clients\_output\_file = "clients\_unique.csv"

unique\_clients.to\_csv(unique\_clients\_output\_file, index=False)

**Exercise 4: Data Privacy and Security**

1. From the given deduplicated CVS, variables that qualify as Personal Identifiable Information are:

* First Name
* Last Name
* Email
* Phone
* Address
* Zip Code
* Gender
* Date of Birth

1. Below is a python script using the Faker library which generates realistic but fake data.

import pandas as pd

from faker import Faker

# Initialize the Faker generator

fake = Faker()

# Read the CSV file with PII

input\_file = "client\_purchases\_deduplicated.csv"

df = pd.read\_csv(input\_file)

# Anonymize sensitive data

df['First Name'] = [fake.first\_name() for \_ in range(len(df))]

df['Last Name'] = [fake.last\_name() for \_ in range(len(df))]

df['Email'] = [fake.email() for \_ in range(len(df))]

df['Phone'] = [fake.phone\_number() for \_ in range(len(df))]

df['Address'] = [fake.street\_address() for \_ in range(len(df))]

df['Zip Code'] = [fake.zipcode() for \_ in range(len(df))]

df['Gender'] = [fake.random\_element(elements=('Male', 'Female')) for \_ in range(len(df))]

df['Date of Birth'] = [fake.date\_of\_birth(minimum\_age=18, maximum\_age=90).strftime('%m/%d/%Y') for \_ in range(len(df))]

# Saving the anonymized data to a new CSV file

anonymized\_output\_file = "clients\_deidentified.csv"

df.to\_csv(anonymized\_output\_file, index=False)