Operational Manual

Funlab - Data Engineer Coding Challenge

05.05.2024

—

Satya Vivek Pabolu | Melbourne, Australia

Overview

SQL Problem

This operational manual provides step-by-step instructions for generating a login duration report from the input data provided in a SQL database. The report summarizes the periods when a user was logged into the system and calculates the duration of each login session.

Python Problem

This operational manual provides instructions on how to use a Python function to find the least repeating character in a given string. The function also prints out the count of each character in the string.

SQL Problem

Input Data Format

The input data is to be in a table format with two columns:

- TIMES: Timestamp indicating the time of login or logoff.
- STATUS: Indicates whether the event is a login ("on") or logoff ("off").

Procedure

Following steps to generate the login duration report:

Step 1: Connect to the Database

- Open your SQL client or interface.
- Connect to the database where the input data is stored.

Step 2: Write the SQL Query

Write the SQL query to generate the login duration report. The query consists of several parts:

- RankedLogins CTE: Assigns row numbers to each record in the input table.
- PairedLogins CTE: Pairs logon and logoff events.
- OrderedPairedLogins CTE: Orders logon and logoff pairs by LOG_ON time and assigns row numbers.

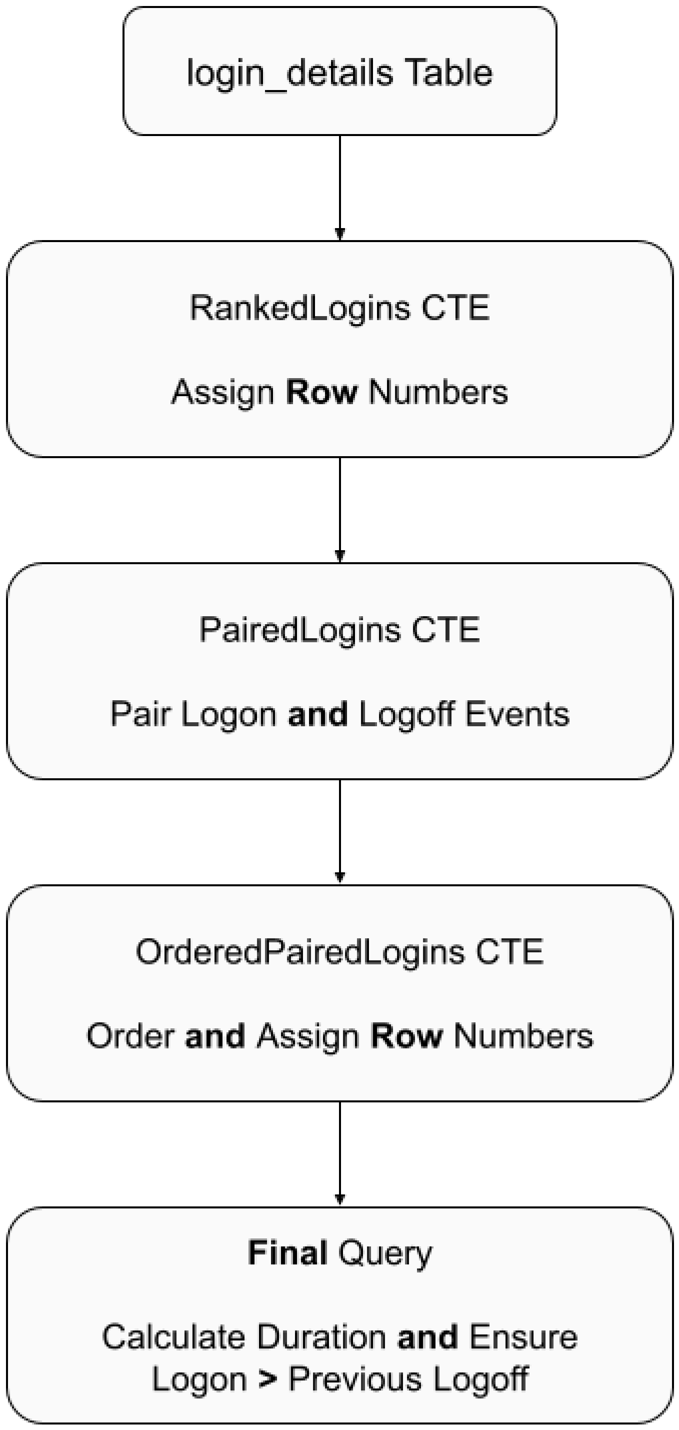
- Final Query: Calculates the duration between logon and logoff times and ensures that the next LOG_ON time is greater than the previous LOG_OFF time.
- **Complete SQL query**

```sql
-- Common Table Expression to assign row numbers to each record in
the input table

WITH RankedLogins AS (

    SELECT

        TIMES,

        STATUS,

        ROW_NUMBER() OVER (ORDER BY TIMES) AS rn

    FROM

        login_details

),

-- CTE to pair logon and logoff events

PairedLogins AS (

    SELECT

        l1.TIMES AS LOG_ON,

        MIN(l2.TIMES) AS LOG_OFF

    FROM

        RankedLogins l1

    LEFT JOIN

        RankedLogins l2 ON l1.rn < l2.rn AND l2.STATUS = 'off'

    WHERE

        l1.STATUS = 'on'

        AND (l2.TIMES IS NULL OR l2.TIMES > l1.TIMES)

    GROUP BY

        l1.TIMES

),
-- CTE to order logon and logoff pairs by LOG_ON time and assign row numbers
OrderedPairedLogins AS (
    SELECT
        LOG_ON,
        LOG_OFF,
        ROW_NUMBER() OVER (ORDER BY LOG_ON) AS rn
    FROM
        PairedLogins
)
-- Final query to calculate the duration between logon and logoff times
SELECT
    opl.LOG_ON,
    opl.LOG_OFF,
    DATEDIFF(MINUTE, opl.LOG_ON, opl.LOG_OFF) AS DURATION
FROM
    OrderedPairedLogins opl
-- Join condition to ensure that the LOG_ON time is greater than the previous LOG_OFF time
LEFT JOIN
    OrderedPairedLogins opl_prev ON opl.rn = opl_prev.rn + 1
WHERE
    opl_prev.LOG_OFF IS NULL OR opl.LOG_ON > opl_prev.LOG_OFF;
```

Step 3: Execute the Query

- Copy the SQL query and paste it into your SQL client or interface.
- Execute the query to generate the login duration report.

Flow Diagram


login_details Table
RankedLogins CTE
Assign **Row** Numbers
PairedLogins CTE
Pair Logon **and** Logoff Events
OrderedPairedLogins CTE
Order **and** Assign **Row** Numbers
Final Query
Calculate Duration **and** Ensure Logon **>** Previous Logoff

In this flow:

- login_details Table: The initial input table.
- RankedLogins CTE: Assigns row numbers to each record in the login_details table.
- PairedLogins CTE: Pairs logon and logoff events.
- OrderedPairedLogins CTE: Orders logon and logoff pairs by LOG_ON time and assigns row numbers.
- Final Query: Calculates the duration between logon and logoff times and ensures that the next LOG_ON time is greater than the previous LOG_OFF time.

Output

The output of the query will be a table representing different periods when the user was logged in, along with the corresponding login and logoff times and the duration of each login session.

OUTPUT Table

LOG_ON	**LOG_OFF**	**DURATION**
10:00:00.0000000	10:03:00.0000000	3
10:04:00.0000000	10:06:00.0000000	2
10:09:00.0000000	10:13:00.0000000	4
10:15:00.0000000	10:16:00.0000000	1

Conclusion

This operational manual provides an approach to generating a login duration report from input data stored in a SQL database. By following the outlined steps, users can efficiently extract meaningful insights about user activity within the system.

Python Problem

Function Overview

The function **least_repeating_character(***string***)** takes a string as input, counts the occurrences of each character, finds the character with the minimum count, and prints out the count of each character along with the least repeating character.

Usage

Input

- **string**: The input string for which you want to find the least repeating character.

Output

- The counts of each character in the input string.
- The least repeating character from the input string.

Instructions

Following steps to use the function:

- Open your Python environment (e.g., IDLE, Jupyter Notebook, etc.).
- Copy the function **least_repeating_character** into your Python environment.

```python
# Function Definition

def least_repeating_character(string):

    """

    Find the least repeating character in a given string and print
its count along with counts of all characters.


    Args:

    string (str): Input string.


    Returns:
```

```
    None: Prints the counts of each character and the least
repeating character.
    """
    # Create a dictionary to count occurrences of each character
    char_count = {}
    for char in string:
        if char in char_count:
            char_count[char] += 1
        else:
            char_count[char] = 1

    # Find the minimum count
    min_count = None
    least_repeating_char = None
    for char, count in char_count.items():
        if min_count is None or count < min_count:
            min_count = count
            least_repeating_char = char

    # Print out the count of each character
    print(char_count)

    # Print out any one of the least repeating characters
    print(least_repeating_char)

# Sample input
string = "aaabbbcccdddeeefffg"
```

```
# Call the function
least_repeating_character(string)
```

- Replace **string** with your desired input string.
- Execute the function.

Sample Output

For the sample input "**aaabbbcccdddeeefffg**", the output will be:

{'a': 3, 'b': 3, 'c': 3, 'd': 3, 'e': 3, 'f': 3, 'g': 1}

g

This output displays the counts of each character in the input string along with the least repeating character.

Conclusion

This operational manual provides a simple and effective way to find the least repeating character in a string using Python with only using loop and python data type; No in-built / 3rd party module. By following the instructions outlined above, users can quickly analyze their input strings and identify the character with the lowest frequency.