Task-1 SQL

In the attachment below, use each worksheet as a table in a relational database and write an SQL query that generates the output report

Input *(Data)*:

[ASDE Assignment](https://docs.google.com/spreadsheets/d/1lHYAKwfWEONislVTkkHu_zMoAWCOG1MlAt8nPoW0grY/edit#gid=0)

Output *(Report)*

Fetch top 3 departments along with their name and average monthly salary. Below is the format of the report.

|  |  |
| --- | --- |
| **DEPT\_NAME** | **AVG\_MONTHLY\_SALARY (USD)** |

Answer:

Assumptions:

* Assume the table containing employee data is named “Employee”.
* The “DEPT\_NAME” column is used to represent the department name.
* The “MONTHLY\_SALARY” column is used to represent the monthly salaries of employees.

Test-Case:

* Check if the query returns the top 3 departments with their average monthly salaries in descending order.
* Verify that the query handles ties in average monthly salary correctly.
* Check if the query handles cases where some departments may have no employees.

Comments:

* “AVG” function is used to calculate the average salary per month for each department and group the results by department name.
* “ORDER BY“ is used for ordering the result in descending order of average monthly salary.
* “LIMIT” using this we can limit output to top 3 departments.

Edge cases and debugging:

* Check for the empty dataset whether it handles correctly or not.
* In case of errors check if the table name in query matches with the table name mentioned in database.
* Also check if the column name mentioned in query are same as mentioned in database or not.

Query:

SELECT DEPT\_NAME, AVG(MONTHLY\_SALARY) AS AVG\_MONTHLY\_SALARY

FROM Employee

GROUP BY DEPT\_NAME

ORDER BY AVG\_MONTHLY\_SALARY DESC LIMIT 3;

## Task-2 Scripting

With the same attachment from Task-1, use each worksheet as a CSV file and write a script (Bash or Python) that generates the same report. Data is to be read from the CSV files not from a database.

Solution:

Assumptions:

To generate the same report form CSV file we need to import it first

Testcase:

* Verify the code reads data correctly from all csv files.
* if code calculates the avg monthly salary for each department correctly.
* Check if code generates the report in given format with top 3 department and their avg monthly salaries.

Comments:

* The code uses the ‘csv’ module to read data from the csv files.
* It calculates the average monthly salary for each department using a dictionary to store salaries and them sort the departments by avg salary in descending order.
* Lastly it prints the top 3 departments along with their average monthly salaries.

PYTHON code:

import csv

from collections import defaultdict

def calculate\_average\_salary(data):

department\_salaries = defaultdict(list)

for row in data:

department\_name = row[‘DEPT\_NAME’]

monthly\_salary = float(row[‘MONTHLY\_SALAER’])

department\_salaries[department\_name].append(monthly\_salary)

average\_salaries = {}

for department, salaries in department\_salaries.items():

average\_salary = sum(salaries)/len(salaries)

average\_salaries[department] = average\_salary

sorted\_departements=sorted(average\_salaries.item(),key=lambda x:x[1], reverse=True)

return sorted\_departements[:3]

#read data from CSV file:

data=[]

file\_names=[‘HR.csv’, ‘Finance.csv’, ‘IT.csv’, ‘ops.csv’, ‘sales.csv’]

for file\_name in file\_names:

with open(file\_name, mode= ‘r’, newline = ‘’) as file:

csv\_reader = csv.DictReader(file)

for row in csv\_reader:

data.append(row)

#calculate top 3 departements with their average monthly salaries

top\_departments=calculate\_average\_salary(data)

#print the report

print(“DEPT\_NAME”)

print(“AVG\_MONTHLY\_SALARY (USD)”)

for department, avg\_salary in top\_departments:

print(f “{department}\n{avg.salary:2f}”)

Task-3 Debugging

Given below is a Bash / Python script that performs following computation on an integer input (n):

1. If n is less than 10: Calculate its Square
   1. Example: 4 => 16
2. If n is between 10 and 20: Calculate the factorial of (n-10)
   1. Example: 15 => 120
3. If n is greater than 20: Calculate the sum of all integers between 1 and (n-20)
   1. Example: 25 => 15

The task is to identify the bugs in the script, fix them and share the new script. Only one of the two scripts required Bash or Python. **Hint**: You can correct the script by changing less than 5 characters.

Script (Bash)

|  |
| --- |
| #!/bin/bash N=$1 if [ $N -lt 10 ] then         OUT=$((N\*N)) elif [ $N -lt 20 ] then         OUT=1         LIM=$((N - 10))         for (( i=1; i<$LIM; i++ ))         do                 OUT=$((OUT \* i))         done else         LIM=$((N - 20))         OUT=$((LIM \* LIM))         OUT=$((OUT - LIM))         OUT=$((OUT / 2)) fi echo $OUT |

Script (Python)

|  |
| --- |
| def compute(n):     if n < 10:         out = n \*\* 2     elif n < 20:         out = 1         for i in range(1, n-10):             out \*= i     else:         lim = n - 20          out = lim \* lim          out = out - lim          out = out / 2      print(out)  n = int(input("Enter an integer: ")) compute(n) |

Changes required to make in code:

* In the factorial calculation (elif block), I fixed the range in the for loop to include 'n - 10' because the range should run up to 'n - 10'.
* In the sum calculation (else block), I improved the calculation using the formula for the sum of integers from 1 to N.
* Instead of printing the result inside the ‘compute’ function, I return the result and then print it outside the function

PYTHON CODE:

def compute(n):

if n < 10:

out =n\*\*2

elif n<20:

out=1

for I in range (1,n-10+1):

out\*=i

else:

lim=n-20

out=lim \*(lim+1)//2

return out

n=int(input(“Enter an integer:”))

result = compute(n)

print(result)

## Task-4 Bash

Use the attached sample TSV file: [sample.tsv](https://drive.google.com/open?id=1ynHzp19ARsC0cZV1mHPwvTDhB6NkH4lF)

Create a new TSV file (output.tsv) with:

1. The first column (Item Name) from the original file (sample.tsv)
2. Add an additional column in the end which has the line number of the item from the original file
3. Only include items whose name start with 'Xerox'
4. The rows are sorted by the item name. (ascending, alphabetical)
5. Once the file output.tsv is created, attach the o/p of the below command along with the generated file:

|  |
| --- |
| cat output.tsv | tee >(wc -l) | tee >(sha1sum) | tail -5 |

**Note:** You cannot use loops of any kind. Do this with a single line command. You can use the pipe operator.

Sample Output: (The first 3 lines of the output.tsv should look like this)

|  |
| --- |
| Xerox 1880 12784 Xerox 1880 12916 Xerox 1880 13334 |

Solution:

* ‘awk '/^Xerox/{print $0, NR}' sample.tsv: This part of the command uses awk to search for lines in sample.tsv where the Item Name starts with 'Xerox' (/^Xerox/). It then prints the entire line ($0) along with the line number (NR) as an additional column.
* 'sort -t$'\t' -k1,1 > output.tsv': After filtering and adding the line numbers, the output is piped to 'sort'. We use '-t$'\t'' to specify that the field separator is a tab, and '-k1,1' to sort based on the first column (Item Name) in ascending alphabetical order. The sorted result is then saved to 'output.tsv'.

BASH Command:

Awk ‘/^Xerox/{print $0,NR}’ sample.tsv |sort -t$‘\t’ -k1,1> output.tsv && head -3 output.tsv

Task-5 Bash

Write a command to print all the groups of the owner of all the files in the current directory.

**Note:** You cannot use loops of any kind. Do this with a single line command. You can use the pipe operator.

Sample Output:

(This only indicates the format, the actual output may change for your environment)

|  |
| --- |
| 1001(user-lenovo),27(sudo),136(vboxusers),141(libvirt),998(docker),1003(rvm) |

Solution:

* ‘find .-maxdepth 1-type f -exec stat -c“%U” {}\;’ : this part is used to finds all files (‘-type f’) in the current directory (not including subdirectories) and executes ‘stat’ to extract the owner’s username for each file.
* ’sort |uniq’ : it sorts the list of owner usernames and removes duplicates using ‘uniq’. This gives you a unique list of owner username.
* ‘xargs -I {} id -Gn{}:’ : it takes each unique username and runs the ‘id -Gn’ command to fetch the groups associated with that username.
* ‘tr ‘\n’ ‘,’ : it translates the newline characters into commas to format the output as requested.

BASH Command:

find . -maxdepth 1 -type f -exec stat -c “%U” \; |sort |uniq | xargs -I {} id -Gn {} |tr ‘\n’ ‘,’