**Interview Question Answers**

SQL & Python

04 October 2022

# Document Information

## Table of Contents

[Document Information 2](#_Toc115857704)

[Table of Contents 2](#_Toc115857705)

[Revision History 2](#_Toc115857706)

[Introduction 3](#_Toc115857707)

[Question 1 - Python (5 marks) 4](#_Toc115857708)

[Question 2 - Python (7.5 Marks) 5](#_Toc115857709)

[Question 3 - SQL (5 Marks) 6](#_Toc115857710)

[Question 4 - SQL (7.5 Marks) 7](#_Toc115857711)

[Question 5 - Hybrid (10 Marks) 8](#_Toc115857712)

## Revision History

|  |  |  |
| --- | --- | --- |
| **Version** | **Details** | **Date** |
| 0.01 | Draft Creation | 05/10/22 |
| 0.02 | Solution Review | 07/10/22 |

# Introduction

Below is a set of questions that will test your skillsets of both when approaching both Python and SQL problems.

Unless specified otherwise,

* You are free to use any python packages you see fit
* You can use any flavour of SQL you choose

The aim will be too complete is **two** or **three** of the questions, you can choose any 2/3 but you must choose **1 SQL** question and **1 python**. Optionally we'd like you to choose the hybrid one two as the question can be solved using either SQL/Python

The Idea of these questions are not to quiz you on your quickfire ability to remember syntax for specific languages, we are more interested in how you approach solving certain problems and would prefer that you accompany each solution with description as to why you solved it in that manner.

Aim to finish your solutions in **~ 30 minutes**

# Question 1 - Python (5 marks)

Below is a function that attempts to calculate a players expected rating score as part of an upcoming chess tournament’s ranking algorithm, this algorithm is based off one part of [ELO rating system](https://en.wikipedia.org/wiki/Elo_rating_system) by Arpad Elo.



The formula that this python function attempts to replicate can be mathematically expressed as:

Where:

RB = The players opponent

RA = The player whom the score Is being estimated for

Your task is re-writing this function with the aim of improving quality and robustness.

### Answer 1:

Following is the solution which contributes to robustness and quality by

* Reducing number of temporary variables
* Wrapping the function body in "try" block to handle errors

def function\_1(x:float, y:float):

try:

return 1 / (1 + pow(10, ((y - x) / 400)))

except Exception as e:

print('unexpected error encountered')

return

# Question 2 - Python (7.5 Marks)

Given a single string variable `input\_sentance`, create a function which returns a dictionary that counts the occurrence of words in any given input (exclude special characters where you deem necessary).

In your answer provide justification on how you approached this question and outline any error handling or robustness you included.

* Please note this solution should be completed using python3 & that you only have access to the **re** package

**Example:**



### Answer 2:

Following is the solution which contributes to robustness and error handling by

* Checking the input type for robustness
* To handle the edge case of empty string the function returns an empty HashMap
* In case string contains no words (only non-punctuations) an empty HashMap is returned

import re

def function\_2(input\_sentance:str):

if type(input\_sentance) != str: *# checking if the input type is string*

print('invalid input type')

return

*# Replacing every non letter and non-whitespace char with an empty string*

input\_sentance = re.sub(r'[^\w\s]', '', input\_sentance)

input\_array = input\_sentance.split() *# splitting the sentence into a list of words*

counts = {} *# hash map to keep counts of words*

for word in input\_array:

*# reading the current count of word from HashMap, if that word is not in the hashmap we get 0*

currentCount = counts.get(word, 0)

counts[word] = currentCount + 1 *# incrementing count of word by 1, if the word did no exist in the* hashmap it creates a new entry

return counts

# Question 3 - SQL (5 Marks)

Consider the following tables:

NAB coding challenge.



Each record in the `sales` table represents the total ($) sale amount a clerk made on any particular transaction. A clerk can make multiple sales per day. `clerk\_id` contains the `id` of the associated clerk.

Each record in the `clerks` table represents an employee’s information `id` & name.

It’s the end of the month (*March 1999*), so you would like to decide the employee of the month based on whomever has the highest total sales amount for this month only. However, we want to ensure that the full name is included in the results set

Write a SQL Query (use any flavour of SQL) that best represents this result and explain why you chose this as the best output.

### 

### Answer 3:

Below is the SQL solution using a subquery to aggregate the monthly sales from *sales* table for given month and joining the top record with *clerks* table to fetch the clerk’s full name.

By aggregating the sales record first and only joining the resultant record will increase the time and robustness of the solution.

select

clerks.id,

clerks.firstName,

clerks.lastName,

Total\_Sales\_by\_clerk.sum\_monthly\_sale

from clerks

inner join

select TOP 1 \* from

(  
 SELECT clerk\_id, SUM(sale\_total) AS sum\_monthly\_sale

FROM sales

WHERE DATE\_FORMAT(sale\_date, '%Y-%m-01') = '1999-03-01'

GROUP BY clerk\_id

ORDER BY sum\_monthly\_sale

) as Total\_Sales\_by\_Clerk

on Total\_Sales\_by\_Clerk.clerk\_id = clerks.id

# Question 4 - SQL (7.5 Marks)

Consider the following tables:



“Happy” is a small party business that has attempted to build their own database to store employee information as they grow. This attempt resulted in the creation of the `happy\_org` table shown above.

After using the database, they have found limitations and would like you to assist them in redesigning it. Some of the problems they face include but are not limited to.

* If updated, Happy has no way to track historical entries. Previous “Job Title”, “Salary”, “Name”
* Some employee’s names are too long and need to be abbreviated.
* Two of their employees share the same name and cause confusion for the admin team.

Describe and/or show how you would redesign this database to create a more robust solution for Happy Org.

### Answer 4:

Below are the changes made in the database to accommodate the business's current and future needs:

* Create Employee dimension table by normalising it and introduce two versions of employee name (namely name\_full that shows the full extended version and name\_short is the abbr. version).
* Employee ID is the unique key that will identify each employee from another even if the names are same.
* Create Type2 Fact table for Employement records that captures the changes in job title and salary with effective from and expiry datetime. The latest record will have a null effective\_to value.

|  |  |
| --- | --- |
| **employee\_dim** | |
| **id** | **INTEGER NOT NULL UNIQUE** |
| name\_full | VARCHAR(50) NOT NULL |
| name\_short | VARCHAR(20) NOT NULL |

|  |  |
| --- | --- |
| **employement\_records\_fact** | |
| **record\_id** | **INTEGER NOT NULL UNIQUE** |
| employee\_id | INTEGER NOT NULL |
| job\_title | VARCHAR(30) NOT NULL |
| salary | BIGINT NOT NULL |
| effective\_from | DATETIME NOT NULL |
| effective\_to | DATETIME |

* Additionally create a view of Current Employee facts with all the attributes built on the base fact table only showing the *effective\_to = NULL* records.
* Additionally create another view that captures all the historical records of the employee with all its attributes

 

Following are the quires to create above views:

create view employee\_current\_view as (

select

f.employee\_id,

d.name\_full,

d.name\_short,

f.job\_title as current\_job\_title,

f.salary as current\_salary,

f.effective\_from as last\_updated\_on

from employment\_records\_fact F

join employee\_dim d

on d.id = f.employee\_id

where f.effective\_to is null

)

create view employee\_history\_view as (

select

f.employee\_id,

d.name\_full,

d.name\_short,

f.job\_title,

f.salary,

f.effective\_from,

f.effective\_to,

(if isnull(f.effective\_to) then "Y" else "N" end) as is\_latest

from employment\_records\_fact F

join employee\_dim d

on d.id = f.employee\_id

where f.effective\_to is null

)

# Question 5 - Hybrid (10 Marks)

Consider that you have a set of data that captures metrics on both past and currently running business-critical SQL queries.

**[query\_ID | status| start\_datetime | end\_datetime]**

Your objective in this task is to build a solution that can detect if a currently running query is running longer than expected.

* Both `start\_datetime` & `end\_datetime` will be in a UNIX epoch time format
* The `query\_id` is the unique identifier of a query
* The `status` column will be either `completed` or `active`. You can assume that all `active` events are currently running.
* All `active` queries will have no value in the `end\_datetime` field until it moves into `completed`

**You may use either Python or any flavour of SQL in this task (Assume the data is either a DataFrame or a SQL Table, we really want to see how you approach the problem over your SQL or Python skills)**

### Answer 5:

Following is the solution is designed by

* Using 90th percentile of query completion time for last 30 days as threshold.
* For given dataset the above threshold will be calculated once.
* Checking all active quires against the threshold and filtering out the quires which is taking longer than expected.

import pandas as pd

from datetime import datetime, timedelta

def function\_5(df:pd.DataFrame):

\_30DaysAgo = datetime.now() - timedelta(days= 30) *# a variable to hold datetime of exactly 30 days ago*

dfLast30days = df[ df['start\_datetime'] > \_30DaysAgo ] *# filtering the queries that started within the last 30 days*

df = dfLast30days *# overwriting the df variable with the filtered datframe*

*# taking only the completed queries*

completed = df[ df['status'] == 'completed' ]

*# taking the 90th percentile of query completion time taken as threshold*

thresh = (completed['end\_datetime'] - completed['start\_datetime']).quantile(0.90)

*# all active queries*

active = df[ df['status'] == 'active' ]

*# all active queries taking longer than the threshold*

abnormal = active [ thresh < (datetime.now() - active['start\_datetime'])]

return abnormal