RPA Developer Technical Test

Teleperformance
each interaction matters

Applicant's name: Diego Eusse Naranjo

Personal ID: C.C 1037655233

January 26th, 2022

PART 1 – SQL PROGRAMMING

SQL

I. Which SQL statement is used to extract data from a database?

a) OPEN
b) EXTRACT
c) GET
d) SELECT

II. Which SQL statement is used to delete data from a database?

a) COLLAPSE
b) REMOVE
c) DELETE
d) SUBTRACT

III. With SQL, how do you select a column named "FirstName" from a table named "People"?

a) SELECT FirstName FROM People
b) SELECT People.FirstName
c) EXTRACT FirstName FROM People

IV. With SQL, how do you select all the records from a table named "People" where the value of the column "FirstName" is "Peter"?

a) SELECT [all] FROM People WHERE FirstName='Peter'
b) SELECT * FROM People WHERE FirstName='Peter'
c) SELECT * FROM People WHERE FirstName<>'Peter'

V. Explain the following query (functioning, requirements, and output) and describe each of its elements:

```
CREATE [ UNIQUE ] INDEX index
ON table (campo [ASC|DESC][, campo [ASC|DESC], ...])
[WITH { PRIMARY | DISALLOW NULL | IGNORE NULL }]
```

This query allows to create a new unique index on an existing table, which cannot have duplicated values within the key columns: the word 'UNIQUE' prohibits duplicated values in the indexed field(s).

The first line calls for the creation of the index and defines its name and the table over which the index is going to be created. In the given example, the index is to be called 'index' and will be created over the 'table' table.

Then, the table field(s) that will define the unique index, must be declared within the parentheses, specifying the order of the indexes, ASC is set by default.

Finally, the las line of the query, which is optional, the WITH clause allows to enforce data validation rules. The options are:

- **PRIMARY:** Designates the indexed field(s) as the primary key(s). As the primary key must be UNIQUE, the UNIQUE clause can be omitted.
- **DISALLOW NULL:** Prohibit null entries in the indexed field(s) of new records.
- **IGNORE NULL:** Prevents records with null values in the indexed field(s) from being included in the index.

The output of the query is the creation of the unique index, following the instructions given by the query and explained above.

VI. What would be the query in SQL Server to:

1. Create TABLE1 and TABLE2.

```
CREATE DATABASE Teleperformance;


USE Teleperformance;


CREATE TABLE Author
(
 [ID] INTEGER IDENTITY(1,1) PRIMARY KEY,
 [Name] VARCHAR(50),
 [Address] VARCHAR(128),
 [Phone] VARCHAR(15),
);
```

```
INSERT INTO Author (Name, Address, Phone)
VALUES
('N1', 'A1', 'P1'),
('N2', 'A2', 'P2'),
('N3', 'A3', 'P3'),
('N4', 'A4', 'P4');

CREATE TABLE Book (
  [ID] INT IDENTITY(1,1) PRIMARY KEY,
  [Author_ID] SMALLINT,
  [Title] VARCHAR(50),
  [Date] VARCHAR(50),
  [Pages] VARCHAR(50)
);

INSERT INTO Book (Author_ID, Title, Date, Pages)
VALUES
('1', 'T1', 'D1', 'Pa1'),
('2', 'T2', 'D2', 'Pa2'),
('3', 'T3', 'D3', 'Pa3'),
('2', 'T4', 'D4', 'Pa4'),
('4', 'T5', 'D5', 'Pa5');
```

2. Query an output similar to TABLE3.

TABLE1: Author

ID	Name	Address	Phone
1	N1	A1	P1
2	N2	A2	P2
3	N3	A3	P3
4	N4	A4	P4

TABLE2: Book

ID	Author_ID	Title	Date	Pages
1	1	T1	D1	Pa1
2	2	T2	D2	Pa2
3	3	T3	D3	Pa3
4	2	T4	D4	Pa4
5	4	T5	D5	Pa5

TABLE3

Name	Books per author
N1	1
N2	2
N3	1
N4	1

```
SELECT Author.Name, COUNT(*) AS 'Books per author'
FROM Author, Book
WHERE Author.ID = Book.Author_ID
GROUP BY Author.Name;
```

PART 2 – PYTHON PROGRAMMING

PYTHON

Python code is also sent as annex.

I. Below is the source code for a function called 'get_sql_string'.

```
def get_sql_string(stores):
    store_names = [x.split(', ')[0] for x in stores]
    store_names = [x.replace(' ', '_') for x in store_names]
    store_regions = [x.split(',')[1] for x in stores]
    locations = store_names + store_regions
    columns = ['sales_' + x.lower() for x in locations]
    return ', '.join(columns)
```

1. There's an error in line 4. What is it and how will you correct it?

Answer/: The split method is separating the x string just taking the comma into account and the blank space is being left within the strings, so store_regions is storing the name of the locations after a blank space. For Example: [' Colombia', ' Peru'] . In the end, this causes the algorithm to return an incorrect string. For example: 'sales_ colombia, sales_ peru', which could make some trouble within SQL.

This can be solved by adding the blank space within the split string parameter, as follows:

```
def get_sql_string(stores):
    store_names = [x.split(', ')[0] for x in stores]
    store_names = [x.replace(' ', '_') for x in store_names]
    store_regions = [x.split(', ')[1] for x in stores]
    locations = store_names + store_regions
    columns = ['sales_' + x.lower() for x in locations]
    return ', '.join(columns)
```

2. Assuming this bug was fixed, what would be returned if the following command was executed:

```
my_stores = ['Fulham Palace Rd, Hammersmith',
             'Crown St, Reading',
             'Leavesden Green, Watford']

get_sql_string(my_stores)
```

Paste a snippet of your output.

Answer/:
'sales_teleperformance_ltda, sales_teleperformance_sa, sales_colombia, sales_peru'

2. Assuming this bug was fixed, what would be returned if the following command was executed:

```
In [24]: my_stores = ['Fulham Palace Rd, Hammersmith', 'Crown St, Reading', 'Leavesden Green, Watford']
         get_sql_string(my_stores)

Out[24]: 'sales_fulham_palace_rd, sales_crown_st, sales_leavesden_green, sales_hammersmith, sales_reading, sales_watford'
```

3. Write a python function that:
   - Accepts a list of strings as input,
   - Drops the strings including numbers or special characters,
   - Prints the top 10 string(s) with maximum length sorted A-Z.

   Test it by including a list of items that would meet the requirements asked above. Paste a snippet of your code and output.

Answer/:

```
def string_process(strings):
    filtered_strings = [x for x in strings if x.isalpha()] #Takes only the alphabetic strings
    filtered_strings.sort(key=len, reverse=True)        #Sorts the list according to the strings length,
starting from the longest
    filtered_strings = filtered_strings[0:10]           #Takes the top 10 longest string(s)
    filtered_strings.sort(key=str.lower)                    #Sorts the strings alphabetically A-Z
    return filtered_strings
```

Testing:

```
text = "Hello, My NaMe 1$ D1360 &  I, wAnt 2 W0Rk 4 Teleperformance . I'm aN ArtiFicial
InteLligence SpecialliSt fr0m Medellin , Antioquia . I hav3 experience w1thin Data Processing &
Automation ."

string_list = text.split(' ')
string_process(string_list)
```

```python
def string_process(strings):
    filtered_strings = [x for x in strings if x.isalpha()] #Takes only the alphabetic strings
    filtered_strings.sort(key=len, reverse=True)           #Sorts the list according to the strings length, starting from the lor
    filtered_strings = filtered_strings[0:10]              #Takes the top 10 longest string(s)
    filtered_strings.sort(key=str.lower)                            #Sorts the strings alphabetically A-Z
    return filtered_strings
```

```python
text = "Hello, My NaMe 1$ D1360 &  I, wAnt 2 W0Rk 4 Teleperformance . I'm aN ArtiFicial InteLligence SpecialliSt fr0m Medellin ,
string_list = text.split(' ')
string_process(string_list)
```

```
['Antioquia',
 'ArtiFicial',
 'Automation',
 'experience',
 'InteLligence',
 'Medellin',
 'NaMe',
 'Processing',
 'SpecialliSt',
 'Teleperformance']
```

II. We oversee performing a scoring process daily, over the thousands of customers we contact every day per campaign. We assign a weight to different characteristics (based on predefined parameters), calculate the score of this customer (only on the day it enters our database), and save it in a column named initial_score.

Create a python function that uses this value to perform daily reductions based on the following, and saves the reduced score in column reduced_score:

- Customers between 0 and 15 days (column 'Days') will reduce only 10 points per day.
- Customers between 16 and 30 days (column 'Days') will reduce 50 points per day.
- Score in customers with more than 30 days (column 'Days') will be immediately reduced to zero.
- Maximum initial_score will always be 1,200 points.
- Once the reduced_score reaches 0, it will print a message saying: "Customer ID#___ has reached score 0 in ___ days". Use data in columns 'Days' and 'CustomerID' to complete it.

To develop this exercise, a simulation dataframe was created, using the following code:

```
import datetime
import pandas as pd
import random

df = pd.DataFrame(columns={"CustomerID", "InitialDate", "Days", "InitialScore"})
start_date = datetime.datetime(2021, 12, 20)
end_date = datetime.datetime.now()

days_between_dates = (end_date - start_date).days
print(days_between_dates)
data = []
```

```
for i in range(100):
    random_number_of_days = random.randrange(days_between_dates)
    random_date = start_date + datetime.timedelta(days=random_number_of_days)
    initial_score = random.randint(600, 1200)
    reduced_score = random.randint(0, initial_score)
    data.append([i, random_date,(end_date - random_date).days, initial_score, reduced_score])

df = pd.DataFrame(data, columns = ["CustomerID", "InitialDate", "Days", "InitialScore",
"ReducedScore"])
df
```

To perform the daily reductions, the following function is proposed:

```
def reduce_by_days(df):
    for i in range(0, len(df)):
        df.loc[i, ("Days")] = (datetime.datetime.now() - df.loc[i, ("InitialDate")]).days
        days = df.loc[i, ("Days")]
        value = 10 if days < 15 else (50 if days < 30 else df.loc[i, ("InitialScore")])
        df.loc[i, ("ReducedScore")] = df.loc[i, ("ReducedScore")] + value if days <= 30 else value
        if df.loc[i, ("InitialScore")] <= df.loc[i, ("ReducedScore")]:
            print("Customer ID " + str(df.loc[i, ("CustomerID")]) + " has reached score 0 in " +
str(df.loc[i, ("Days")]) + " days")
    return df
df2 = reduce_by_days(df)
df2
```

A snippet of the thrown results is shown below:

```
Customer ID 1 has reached score 0 in 31 days
Customer ID 5 has reached score 0 in 30 days
Customer ID 11 has reached score 0 in 30 days
Customer ID 26 has reached score 0 in 30 days
Customer ID 38 has reached score 0 in 31 days
Customer ID 39 has reached score 0 in 31 days
Customer ID 50 has reached score 0 in 22 days
Customer ID 75 has reached score 0 in 31 days
Customer ID 80 has reached score 0 in 22 days
Customer ID 81 has reached score 0 in 30 days
Customer ID 86 has reached score 0 in 31 days
```

	CustomerID	InitialDate	Days	InitialScore	ReducedScore
0	0	2021-12-29	27	1161	90
1	1	2021-12-25	31	889	889
2	2	2022-01-07	18	1097	303
3	3	2022-01-15	10	959	885
4	4	2022-01-04	21	766	196

III. Classes give us the ability to create more complicated data structures that contain arbitrary content. Follow these steps to create a Class for Triangles:

1. Create a class, Triangle. Arguments in its __init__() method: self, angle1, angle2, and angle3.
2. Create a variable named number_of_sides and set it equal to 3.
3. Create a method named check_angles. It should return True if the sum of self.angle1, self.angle2, and self.angle3 is equal 180, and False otherwise.
4. Create a variable named my_triangle and set it equal to a new instance of your Triangle class. Pass it three angles that sum to 180 (e.g. 90, 30, 60).
5. Print out my_triangle.number_of_sides and print out my_triangle.check_angles().

```
class Triangle:

  def __init__(self, angle1, angle2, angle3):
    self.number_of_sides = 3
    self.angle1 = angle1
    self.angle2 = angle2
    self.angle3 = angle3

  def check_angles(self):
    if(self.angle1 + self.angle2 + self.angle3 == 180):
      return True
    else:
      return False
```

```
my_triangle = Triangle(40, 60, 80)

print(my_triangle.number_of_sides)

print(my_triangle.check_angles())
```

```
In [74]: class Triangle:

             def __init__(self, angle1, angle2, angle3):
                 self.number_of_sides = 3
                 self.angle1 = angle1
                 self.angle2 = angle2
                 self.angle3 = angle3

             def check_angles(self):
                 if(self.angle1 + self.angle2 + self.angle3 == 180):
                     return True
                 else:
                     return False

In [75]: my_triangle = Triangle(40, 60, 80)
         print(my_triangle.number_of_sides)
         print(my_triangle.check_angles())

         3
         True
```

IV. Suppose the following four csv files are available to perform data analysis on the transaction history of one of our retail customers:

transactions.csv *(2 years of data)*
customer_id
store_id
date_id
ticket_id
product_id
unit
sales

stores.csv
store_id
store_name
state
banner

customer.csv
customer_id
age
gender
household_size

products.csv
product_id
product_name
brand
supplier
department (food, non-food etc)
category (water, chocolate, clothes etc)

As dataframes are not provided, some random testing .csv files were created. I start by reading them:

```
transactions_df = pd.read_csv("transactions.csv", sep = ";")
stores_df = pd.read_csv("stores.csv", sep = ";")
customer_df = pd.read_csv("customer.csv", sep = ";")
products_df = pd.read_csv("products.csv", sep = ";")
```

1. Write a code to sort customer table by customer_id (ascending order) and remove duplicates.

```
customer_df = pd.read_csv("customer.csv", sep = ";")
sorted_customers = customer_df.sort_values(by=['customer_id'], ascending=True)
dropped_customers = sorted_customers.drop_duplicates(subset=['customer_id'])
dropped_customers
```

2. Create a table named transaction_cube merging all tables.

```
import pandas as pd

transactions_df = pd.read_csv("transactions.csv", sep = ";")
stores_df = pd.read_csv("stores.csv", sep = ";")
customer_df = pd.read_csv("customer.csv", sep = ";")
products_df = pd.read_csv("products.csv", sep = ";")

df1 = transactions_df.merge(stores_df, on='store_id')
df2 = df1.merge(customer_df, on='customer_id')
transaction_cube = df2.merge(products_df, on='product_id')
transaction_cube
```

3. Create a table containing customer_id (s) existing in the transactions table but not in the customer table

```
import pandas as pd

transactions_df = pd.read_csv("transactions.csv", sep = ";")
customer_df = pd.read_csv("customer.csv", sep = ";")
transaction_customers = transactions_df [['customer_id']]
customer_ids = customer_df[['customer_id']]

filtered_customers =
transaction_customers.loc[~transaction_customers['customer_id'].isin(customer_ids
['customer_id'])].copy()
filtered_customers
```

4. Create a table named customer_summary with the following variables:
    - Customer_id
    - Banner
    - Category
    - Department
    - Total_sales
    - Total_units
    - Average_ticket

- Last_visit ---

To develop this assignment, the following code was developed:

```
import pandas as pd

customer_summary = transaction_cube.copy()
customer_summary = customer_summary[['customer_id', 'banner', 'category', 'department',
'sales', 'unit', 'ticket_id', 'date_id']]
customer_summary = customer_summary.groupby(['customer_id', 'banner', 'category',
'department']).agg({'sales':'sum', 'unit':'sum', 'ticket_id':'mean', 'date_id':'max'})
customer_summary.columns = ['Total_sales', 'Total_units', 'Average_ticket', 'Last_visit']
customer_summary
```

5. Create a table named customer_metrics with the following variables:
    - Customer_id
    - Month_year
    - Banner
    - Category
    - Department
    - Total_sales
    - Total_units
    - Visits_count (# tickets)
    - Visits_count_customer (# tickets regardless of product, category, or department)
    - Products_count
    - Products_count_customer (# products regardless of banner)
    - Median_Price
    - Distinct_Stores

    To develop this assignment, the following code was developed:

```
import datetime

customer_metrics = transaction_cube.copy()
customer_metrics = customer_metrics[['customer_id', 'banner', 'category', 'department',
'sales', 'unit', 'ticket_id', 'date_id', 'product_id', 'store_id']]
customer_metrics['date_id'] = customer_metrics['date_id'].map(lambda x:
datetime.datetime.utcfromtimestamp(x))
array = customer_metrics['date_id'].map(lambda x: str(x.month) + '_' + str(x.year))
customer_metrics['month_year'] = array
customer_metrics = customer_metrics.groupby(['customer_id', 'banner', 'category',
'department', 'store_id', 'month_year']).agg({'sales':'sum', 'unit':'sum', 'ticket_id':'count',
'product_id':'count', 'sales':'mean'})
df1 = customer_metrics.groupby(level=[0, 1, 4, 5]).agg({'ticket_id':'count'})
df1.columns = ['Visits_count_customer']
customer_metrics = customer_metrics.join(df1)
df2 = customer_metrics.groupby(level=[0, 2, 3, 4, 5]).agg({'product_id':'count'})
df2.columns = ['Products_count_customer']
customer_metrics = customer_metrics.join(df2)
customer_metrics.columns = ['Total_sales', 'Total_units', 'Visits_count', 'Products_count',
'Visits_count_customer', 'Products_count_customer']

customer_metrics
```

6. Considering the four tables provided and any additional field/table derived from the last analysis (you will have available two years of historical data), how will you measure customer loyalty segmentation in terms of shopping? How will you approach your idea and what methodology will you choose to segment the customers and validate your results?
   Only explain yourself, no code required.

Ans/:

To measure customer loyalty, several procedures are available. In my opinion, some fields give special information that must be taken as priority. In the last table, *customer_metrics*, some fields like 'Visits_count_customer' or just 'Visits_count' allow us to meet the customers with more visits to the instalations, within the given time, depending on whether we need to analyze the visits to a certain store, filtering by department, category or product or not. Sorting these fields would help us to find the customers with more visits to each store.

We could also take the Last_visit field from the *customer_metrics* table. This field could help us to find if a given customer is still active or if he/she has deserted.

New metrics could be created, as an example, a ratio between the 'Visits_count' and a determined period of time (weeks, months, years) could be found and sorted, to find the most rated-frequent customers, or well, a ratio between the the 'Products_count' and the 'Visits_count'. These two new metrics could tell us how often a customer visits us and how many products he buys per visit, in average.

Other fields could also be analyzed, such as 'Total_sales' or 'Products_count_customer', which can give us information about what the customer buys but taking into account that a high value in this field does not necessarily mean that the customer is loyal to our company: The customer could made a big purchase in a single visit.

To approach my idea, i would mainly use plots and unsupervised learning, when needed. In this way, I could first examine the dispersion of the data and define possible patterns among it. If needed, unsupervised machine-learning methods like clustering could help me segment different customer groups according to their characteristics. After having trained a classification model, results could be validated, to measure the model behavior.