**BUNGEE TECH ASSESMENT**

**SQL**

**By Abhigyan Dutta**

Creating a table first

Note: I am using PostgreSQL for this assessment.

The link for these queries: <https://drive.google.com/file/d/1dIcmEjzk2Ycp8e4j9CA88o62hCUd4cZT/view?usp=sharing>

CREATE TABLE Sales (

Sale\_date DATE,

Country VARCHAR(50),

Category VARCHAR(50),

Product VARCHAR(50),

Sales INT

);

INSERT INTO Sales (Sale\_date, Country, Category, Product, Sales)

VALUES

('2021-01-01', 'USA', 'Retail', 'UUID1', 10000),

('2021-01-01', 'USA', 'Healthcare', 'UUID2', 2000),

('2021-01-01', 'Canada', 'Retail', 'UUID3', 30000),

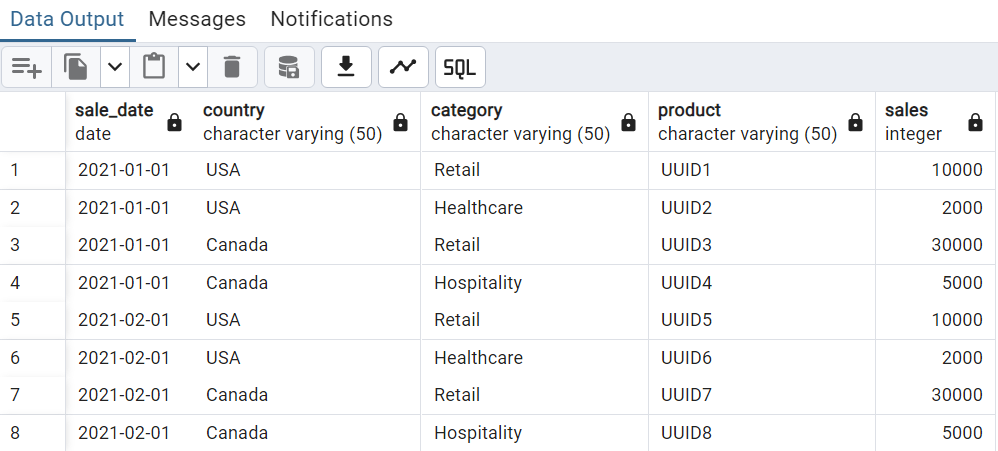
('2021-01-01', 'Canada', 'Hospitality', 'UUID4', 5000),

('2021-02-01', 'USA', 'Retail', 'UUID5', 10000),

('2021-02-01', 'USA', 'Healthcare', 'UUID6', 2000),

('2021-02-01', 'Canada', 'Retail', 'UUID7', 30000),

('2021-02-01', 'Canada', 'Hospitality', 'UUID8', 5000);



Q1. Top 3 categories within every country based on total sales in the current year.

WITH SalesSummary AS (

    SELECT

        Country,

        Category,

        SUM(Sales) AS Total\_Sales

    FROM

        Sales

    WHERE

        EXTRACT(YEAR FROM Sale\_date) = 2021

    GROUP BY

        Country, Category

),

RankedSales AS (

    SELECT

        Country,

        Category,

        Total\_Sales,

        RANK() OVER (PARTITION BY Country ORDER BY Total\_Sales DESC) AS Rank

    FROM

        SalesSummary

)

SELECT

    Country,

    Category,

    Total\_Sales

FROM

    RankedSales

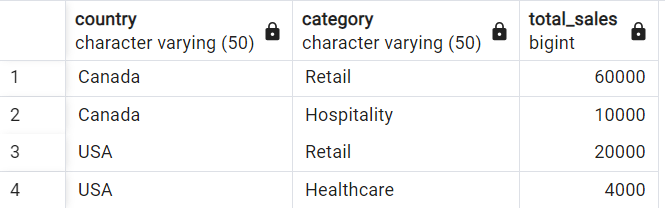
WHERE

    Rank <= 3

ORDER BY

    Country, Total\_Sales DESC;

Output:



Note: Since there are only two categories per country in the sample data, it lists those categories as the top ones. If there were more categories, the query would filter out only the top 3 based on total sales.

Q2. Total sales of products sold in both Feb & Jan, Total sales of products sold ONLY in Jan, Total sales of products sold ONLY in Feb.

WITH JanSales AS (

    SELECT Product, Sales

    FROM Sales

    WHERE EXTRACT(MONTH FROM Sale\_date) = 1 AND EXTRACT(YEAR FROM Sale\_date) = 2021

),

FebSales AS (

    SELECT Product, Sales

    FROM Sales

    WHERE EXTRACT(MONTH FROM Sale\_date) = 2 AND EXTRACT(YEAR FROM Sale\_date) = 2021

),

BothJanFeb AS (

    SELECT COALESCE(JanSales.Product, FebSales.Product) AS Product,

           COALESCE(JanSales.Sales, 0) + COALESCE(FebSales.Sales, 0) AS Sales

    FROM JanSales

    INNER JOIN FebSales ON JanSales.Product = FebSales.Product

),

OnlyJan AS (

    SELECT SUM(Sales) AS jan\_sales

    FROM JanSales

    WHERE Product NOT IN (SELECT Product FROM FebSales)

),

OnlyFeb AS (

    SELECT SUM(Sales) AS feb\_sales

    FROM FebSales

    WHERE Product NOT IN (SELECT Product FROM JanSales)

),

BothSales AS (

    SELECT SUM(Sales) AS jan\_feb\_sales

    FROM BothJanFeb

)

SELECT

    COALESCE(BothSales.jan\_feb\_sales, 0) AS jan\_feb\_sales,

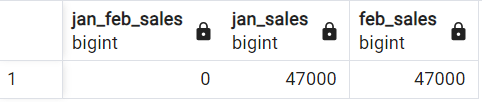
    COALESCE(OnlyJan.jan\_sales, 0) AS jan\_sales,

    COALESCE(OnlyFeb.feb\_sales, 0) AS feb\_sales

FROM

    BothSales, OnlyJan, OnlyFeb;

Output:



Q3. In the query written in question Q1 what are the partitions and indexes you would create for best performance?

To improve the query performance in question Q1, here are the key partitions and indexes:

**Partition by Sale\_date (e.g., by year)**: This limits data scanned when filtering for specific years, like 2021.

**Index on Country and Category**: Helps speed up filtering and grouping by Country and Category, making the sales summary faster.

**Index on Total\_Sales** (for ranking): Speeds up ranking categories by sales within each country, making it easier to sort by sales in the top categories.

These changes help the database quickly filter, group, and rank the needed data.

Table: Employee

+--------------+---------+

| Column Name | Type |

+--------------+---------+

| id | int |

| name | varchar |

| departmentId | int |

+--------------+---------+

id is the primary key column for this table.

departmentId is a foreign key of the ID from the Department table.

Each row of this table indicates the ID, name, and salary of an employee. It also contains the ID of their department.

Table: Salary

+--------------+---------+

| Column Name | Type |

+--------------+---------+

| employeeid | int |

| salary | varchar |

+--------------+---------+

Table: Department

+-------------+---------+

| Column Name | Type |

+-------------+---------+

| id | int |

| name | varchar |

+-------------+---------+

id is the primary key column for this table.

Each row of this table indicates the ID of a department and its name.

Q1: Employees with same last name and different departments.

SELECT

    e1.name AS employee1\_name,

    d1.name AS department1\_name,

    e2.name AS employee2\_name,

    d2.name AS department2\_name

FROM

    Employee e1

JOIN

    Employee e2 ON SPLIT\_PART(e1.name, ' ', -1) = SPLIT\_PART(e2.name, ' ', -1)

                AND e1.departmentId <> e2.departmentId

                AND e1.id < e2.id

JOIN

    Department d1 ON e1.departmentId = d1.id

JOIN

    Department d2 ON e2.departmentId = d2.id;

Output: Since there is no dataset, no output

Q2: Top 3 salaries department wise

output format :: (department, employee\_name, salary)

SELECT

    d.name AS department,

    e.name AS employee\_name,

    s.salary

FROM

    Employee e

JOIN

    Salary s ON e.id = s.employeeid

JOIN

    Department d ON e.departmentId = d.id

JOIN (

    SELECT

        employeeid,

        departmentId,

        DENSE\_RANK() OVER (PARTITION BY departmentId ORDER BY CAST(salary AS NUMERIC) DESC) AS rank

    FROM

        Employee

    JOIN

        Salary ON Employee.id = Salary.employeeid

) ranked\_salaries ON e.id = ranked\_salaries.employeeid

WHERE

    ranked\_salaries.rank <= 3

ORDER BY

    d.name, CAST(s.salary AS NUMERIC) DESC;

Output: Since there is no dataset, no output

Q3: Salaries that are lower than average salary of department.

output format :: (department, employee\_name, salary)

SELECT

    d.name AS department,

    e.name AS employee\_name,

    s.salary

FROM

    Employee e

JOIN

    Salary s ON e.id = s.employeeid

JOIN

    Department d ON e.departmentId = d.id

WHERE

    s.salary::NUMERIC < (

        SELECT AVG(s2.salary::NUMERIC)

        FROM Employee e2

        JOIN Salary s2 ON e2.id = s2.employeeid

        WHERE e2.departmentId = e.departmentId

    );

Output: Since there is no dataset, no output

**PYTHON**

Note: I am using juyter notebook for this assessment.

The link for this notebook: <https://drive.google.com/file/d/1qxphx0Gqz5dB9Sm68xyazcxauufpSey0/view?usp=sharing>

Q1: Average Price per SKU.

import pandas as pd

pd.set\_option('display.max\_columns', None)

data = pd.read\_csv('MAIN DE.csv')

# Clean the PRICE column and calculate average price per SKU

data['PRICE'] = data['PRICE'].replace(r'[\$,]', '', regex=True).astype(float)

average\_price\_per\_sku = data.groupby('SKU')['PRICE'].mean()

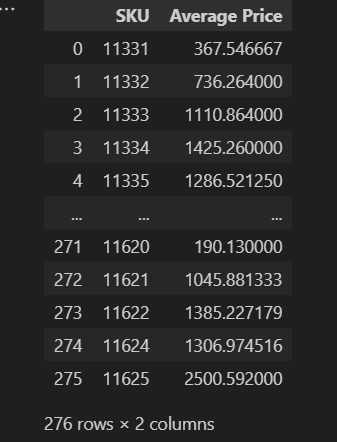
average\_price\_per\_sku\_df = pd.DataFrame(average\_price\_per\_sku).reset\_index()

average\_price\_per\_sku\_df.columns = ['SKU', 'Average Price']

display(average\_price\_per\_sku\_df)

pd.reset\_option('display.max\_columns')

Output:



Q2: Country wise Number of unique products being sold, descending order of unique products

# Calculate the number of unique products per country, sorted in descending order

country\_unique\_products = data.groupby('COUNTRY')['SKU'].nunique().sort\_values(ascending=False)

country\_unique\_products\_df = pd.DataFrame(country\_unique\_products).reset\_index()

country\_unique\_products\_df.columns = ['Country', 'Unique Products']

display(country\_unique\_products\_df)

Output:





Q3: Convert the given CSV to parquet, with two new columns

Column 1: &#39;currency&#39; , populate the currency column with an appropriate value.

Column 2: &#39;unit of measure&#39; , populate column with an appropriate value.

Dtypes for Price column should be Float, rest all string.

To convert the CSV to parquet we have to install pyarrow

!pip install pyarrow

Code:

import pandas as pd

from tabulate import tabulate

data = pd.read\_csv('MAIN DE.csv')

# Convert the PRICE column to float, removing any currency symbols

data['PRICE'] = data['PRICE'].replace('[\$,]', '', regex=True).astype(float)

# Add new column 'currency' with a default value

data['currency'] = 'USD'

def convert\_to\_ml(capacity):

    if pd.isna(capacity):

        return None

    elif 'L' in capacity:

        return float(capacity.replace('L', '')) \* 1000  # Convert liters to milliliters

    elif 'ml' in capacity:

        return float(capacity.replace('ml', ''))

    else:

        return None

data['CAPACITY'] = data['CAPACITY'].astype(str)

data['CAPACITY'] = data['CAPACITY'].apply(convert\_to\_ml)

data['unit of measure'] = 'ml'

for col in data.columns:

    if col not in ['PRICE', 'CAPACITY']:

        data[col] = data[col].astype(str)

# Save the DataFrame as a Parquet file

data.to\_parquet('MAIN\_DE.parquet', engine='pyarrow')

print(tabulate(data, headers='keys', tablefmt='pretty', showindex=False))

Sample Output:

Note: Since the whole output doesn’t fit here I am attaching a sample output.

