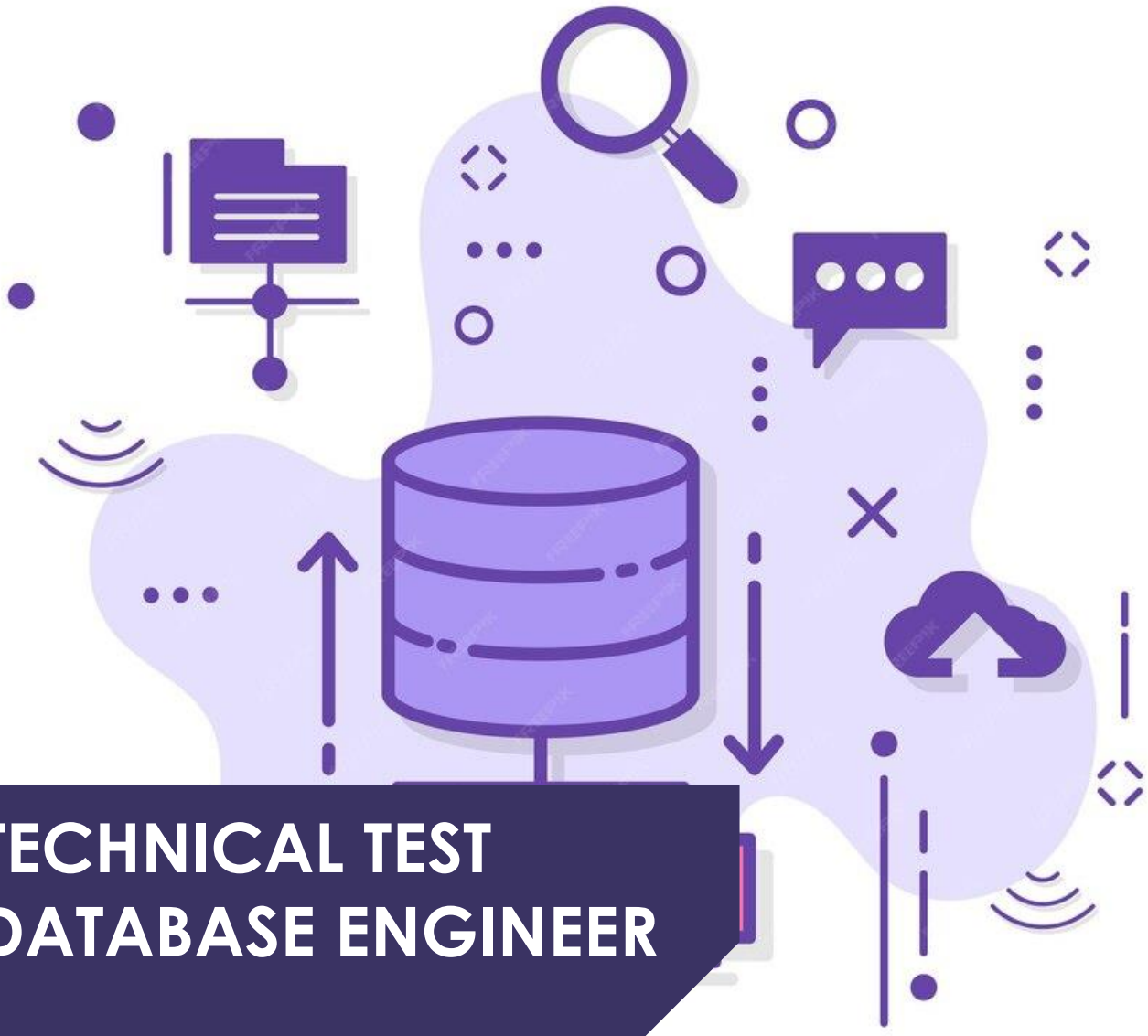


TECHNICAL TEST DATABASE ENGINEER

PT. Sukses Solusindo Digital

Name Candidate: Mukhlis Aryanto

Email Candidate: aryantomukhlis@gmail.com

Phone Candidate: 083869756993

Github Project: https://github.com/mukhlisaryanto/DBE_PT_Sukses_Solusindo_Digital

TABLE OF CONTENTS

PREPARATION 3

Environment 3

DATA WAREHOUSE DESIGN 4

Star Schema Design 4

Run Query To Create Data Warehouse in console psql 5

ETL PROCESS 6

Extract Load Data using python script 6

Run Source code Extract and Load Data 6

Run Query Transform Data in console psql 8

Data Validation 9

DATA ANALYSIS SQL QUERIES 10

Total sales amount per product category 10

Top 3 customers based on total purchase amount. 10

Monthly sales trend for the first quarter of 2024. 11

TOTAL QUANTITY SOLD PER STORE. 11

PERFORMANCE OPTIMIZATION 12

Indexing 12

Partitioning 12

Denormalization 12

Query Optimization 13

REPORTING 14

SQL Query for reporting 14

Run the python code to export query to excel format and visualize using matplotlib 14

Visualize 15

PREPARATION

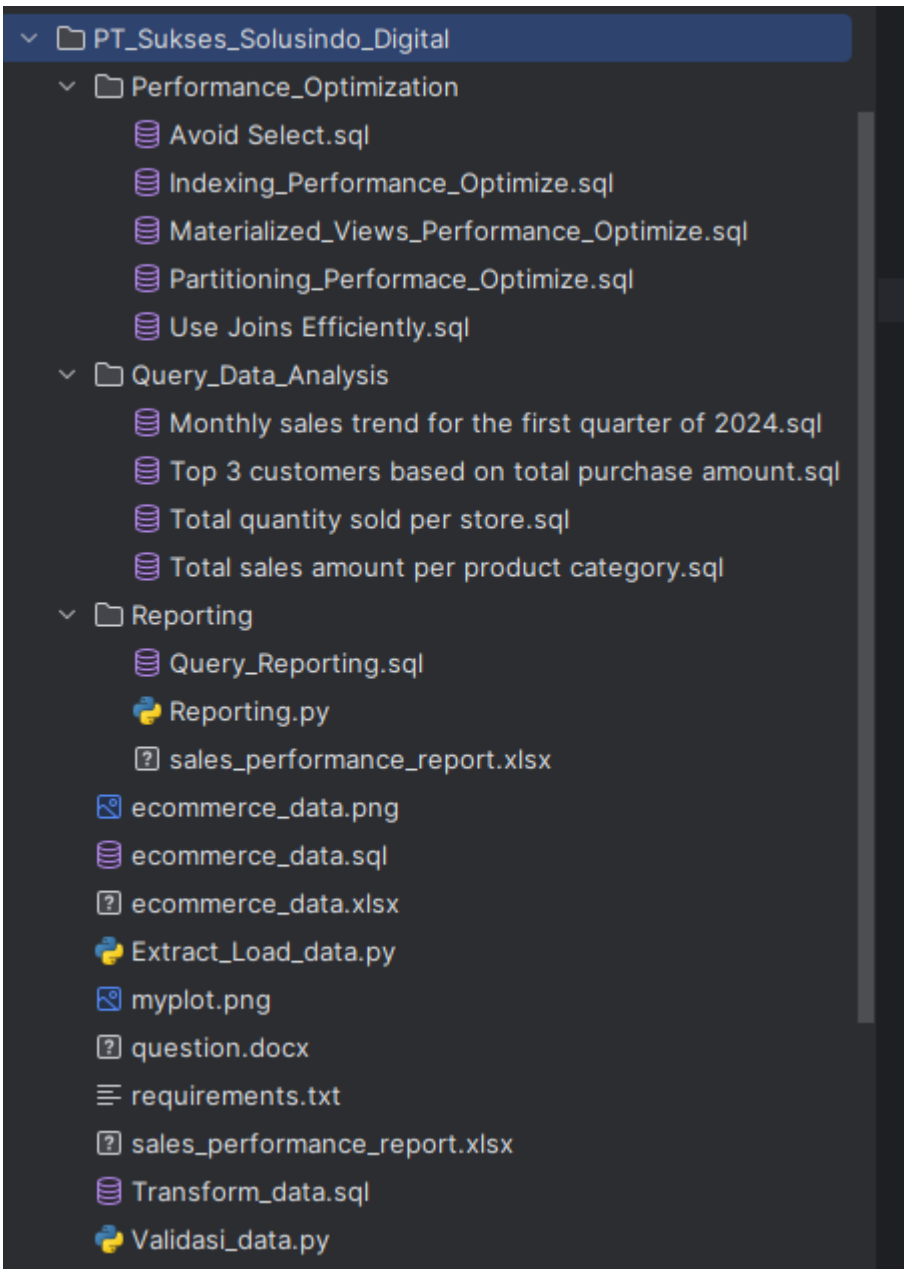
Environment

DBMS : PostgreSQL version 16 & PgAdmin version 8.2

Code Editor : PyCharm version 2024.1.1

Python : 3.13.3

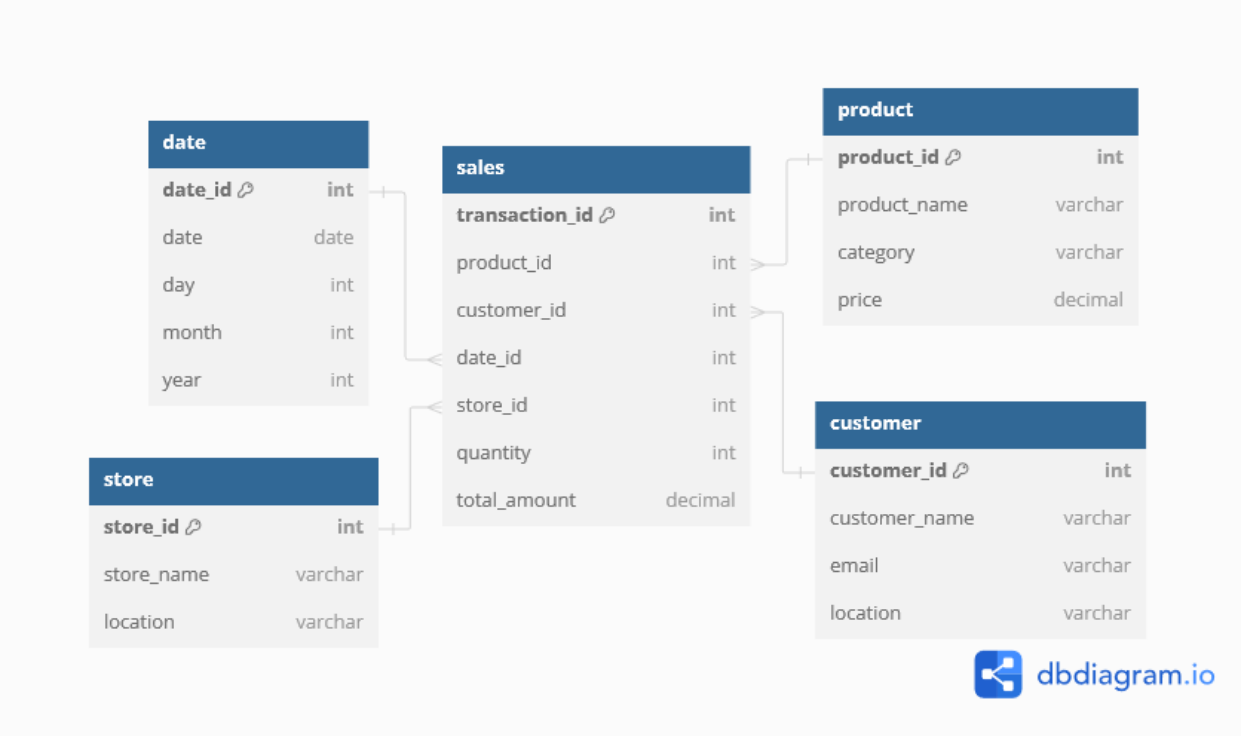
All files into one folder **DBE_PT_Sukses_Solusindo_Digital**



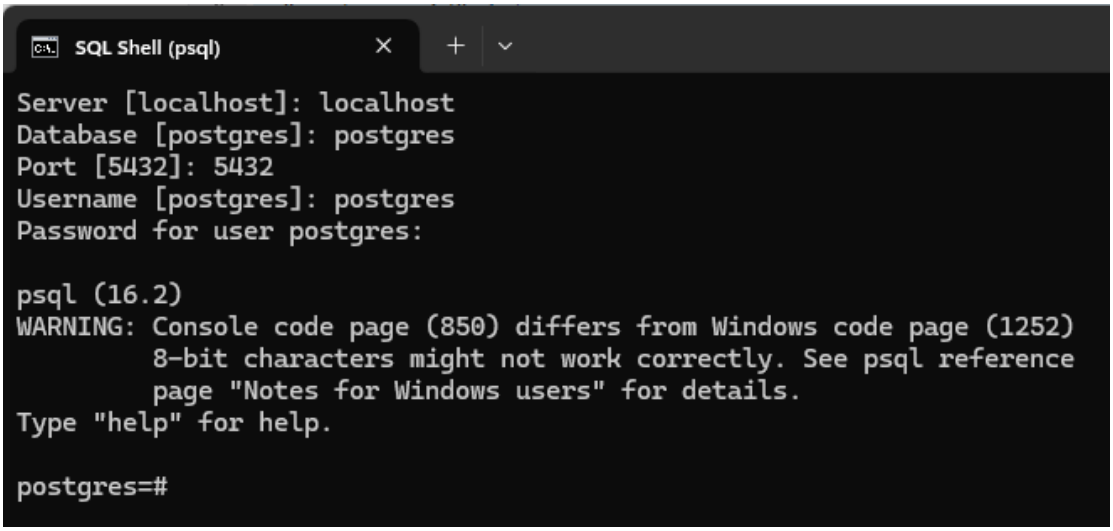
DATA WAREHOUSE DESIGN

Star Schema Design

Design a star schema for an e-commerce platform using the provided raw data. Identify the fact and dimension tables.



Open SQL Shell (psql)



My Password: root

Run Query To Create Data Warehouse in console psql

Namefile : ecommerce_data.sql

```
CREATE DATABASE ecommerce_data;

\c ecommerce_data

CREATE TABLE  IF NOT EXISTS "sales" (
  "transaction_id" int PRIMARY KEY,
  "product_id" int,
  "customer_id" int,
  "date_id" int,
  "store_id" int,
  "quantity" int,
  "total_amount" decimal
);

CREATE TABLE  IF NOT EXISTS "product" (
  "product_id" int PRIMARY KEY,
  "product_name" varchar,
  "category" varchar,
  "price" decimal
);

CREATE TABLE  IF NOT EXISTS "customer" (
  "customer_id" int PRIMARY KEY,
  "customer_name" varchar,
  "email" varchar,
  "location" varchar
);

CREATE TABLE  IF NOT EXISTS "date" (
  "date_id" int PRIMARY KEY,
  "date" date,
  "day" int,
  "month" int,
  "year" int
);

CREATE TABLE  IF NOT EXISTS "store" (
  "store_id" int PRIMARY KEY,
  "store_name" varchar,
  "location" varchar
);

ALTER TABLE "sales" ADD FOREIGN KEY ("product_id") REFERENCES "product"
("product_id");
ALTER TABLE "sales" ADD FOREIGN KEY ("customer_id") REFERENCES "customer"
("customer_id");
ALTER TABLE "sales" ADD FOREIGN KEY ("date_id") REFERENCES "date" ("date_id");
ALTER TABLE "sales" ADD FOREIGN KEY ("store_id") REFERENCES "store" ("store_id");
```

ETL PROCESS

Extract Load Data using python script

Install dependencies library in terminal

```
Pip install openpyxl
pip install psycopg2
pip install pandas
```

Run Source code Extract and Load Data

Namefile : Extract_Load_data.py

```
import pandas as pd
import psycopg2

# Baca file Excel
excel_file = r"./ecommerce_data.xlsx"
sheets = ['product', 'customer', 'date', 'store', 'sales']

# Membaca setiap sheet ke dalam DataFrame
dfs = {sheet: pd.read_excel(excel_file, sheet_name=sheet) for sheet in sheets}

# Koneksi ke PostgreSQL
try:
    conn = psycopg2.connect(
        dbname="ecommerce_data",
        user="postgres",
        password="root",
        host="localhost",
        port="5432"
    )
    cursor = conn.cursor()
    print("Koneksi ke database berhasil.")
except Exception as e:
    print(f"Error saat menghubungkan ke database: {e}")

# Fungsi untuk menentukan tipe data SQL berdasarkan tipe data pandas
def get_sql_type(dtype):
    if pd.api.types.is_integer_dtype(dtype):
        return 'INT'
    elif pd.api.types.is_float_dtype(dtype):
        return 'DECIMAL'
    elif pd.api.types.is_bool_dtype(dtype):
        return 'BOOLEAN'
    elif pd.api.types.is_datetime64_any_dtype(dtype):
        return 'DATE'
    else:
        return 'VARCHAR'

# Fungsi untuk memuat DataFrame ke tabel PostgreSQL
def load_to_postgres(df, table_name):
    csv_file = f'{table_name}.csv'
    # Simpan DataFrame sebagai CSV sementara
    df.to_csv(csv_file, index=False, header=False)
    print(f"DataFrame disimpan sebagai {csv_file}")
```

```
    # Muat data ke PostgreSQL
    try:
        with open(csv_file, 'r') as f:
            cursor.copy_expert(f"COPY {table_name} FROM STDIN WITH CSV", f)
        conn.commit()
        print(f"Data dimuat ke tabel {table_name}")
    except Exception as e:
        print(f"Error saat memuat data ke tabel {table_name}: {e}")

# Membuat tabel sementara dan memuat data
for sheet, df in dfs.items():
    temp_table = f"{sheet}_temp"
    df.columns = map(str.lower, df.columns)  # Pastikan kolom dalam huruf kecil
    # Buat tabel sementara (sesuaikan tipe data jika diperlukan)
    cursor.execute(f"DROP TABLE IF EXISTS {temp_table}")
    create_table_sql = f"""
    CREATE TABLE {temp_table} (
        {', '.join([f'{col} {get_sql_type(df[col].dtype)}' for col in
df.columns])}
    )
    """
    print(f"Membuat tabel: {create_table_sql}")
    cursor.execute(create_table_sql)
    conn.commit()

    # Muat data ke tabel sementara
    load_to_postgres(df, temp_table)

# Tutup koneksi
cursor.close()
conn.close()
print("Proses selesai.")
```

Result:

```
Run    Extract_Load_data
        CREATE TABLE store_temp (
            store_id INT, store_name VARCHAR, location VARCHAR
        )

DataFrame disimpan sebagai store_temp.csv
Data dimuat ke tabel store_temp
Membuat tabel:
    CREATE TABLE sales_temp (
        transaction_id INT, product_id INT, customer_id INT, date_id INT, store_id INT, quantity INT, total_amount INT
    )

DataFrame disimpan sebagai sales_temp.csv
Data dimuat ke tabel sales_temp
Proses selesai.

Process finished with exit code 0
```

Run Query Transform Data in console psql

Namefile : Transform_data.sql

```
-- Buat tabel akhir jika belum ada
CREATE TABLE IF NOT EXISTS product (
  product_id INT PRIMARY KEY,
  product_name VARCHAR,
  category VARCHAR,
  price DECIMAL
);

CREATE TABLE IF NOT EXISTS customer (
  customer_id INT PRIMARY KEY,
  customer_name VARCHAR,
  email VARCHAR,
  location VARCHAR
);

CREATE TABLE IF NOT EXISTS date (
  date_id INT PRIMARY KEY,
  date DATE,
  day INT,
  month INT,
  year INT
);

CREATE TABLE IF NOT EXISTS store (
  store_id INT PRIMARY KEY,
  store_name VARCHAR,
  location VARCHAR
);

CREATE TABLE IF NOT EXISTS sales (
  transaction_id INT PRIMARY KEY,
  product_id INT,
  customer_id INT,
  date_id INT,
  store_id INT,
  quantity INT,
  total_amount DECIMAL
);

-- Pindahkan data dari tabel sementara ke tabel akhir
INSERT INTO product (product_id, product_name, category, price)
SELECT product_id, product_name, category, price FROM product_temp;

INSERT INTO customer (customer_id, customer_name, email, location)
SELECT customer_id, customer_name, email, location FROM customer_temp;

INSERT INTO date (date_id, date, day, month, year)
SELECT date_id, CAST("date" AS DATE), day, month, year FROM date_temp;

INSERT INTO store (store_id, store_name, location)
SELECT store_id, store_name, location FROM store_temp;

INSERT INTO sales (transaction_id, product_id, customer_id, date_id, store_id,
quantity, total_amount)
SELECT transaction_id, product_id, customer_id, date_id, store_id, quantity,
```

```
total_amount FROM sales_temp;

-- Hapus tabel sementara
DROP TABLE IF EXISTS product_temp;
DROP TABLE IF EXISTS customer_temp;
DROP TABLE IF EXISTS date_temp;
DROP TABLE IF EXISTS store_temp;
DROP TABLE IF EXISTS sales_temp;
```

Data Validation

```
import psycopg2

# Koneksi ke PostgreSQL
conn = psycopg2.connect(
    dbname="ecommerce_data", user="postgres", password="root",host="localhost",
    port="5432"
)
cursor = conn.cursor()
# Fungsi untuk menjalankan query validasi
def validate_query(query):
    cursor.execute(query)
    return cursor.fetchone()[0]

# 1. Periksa Jumlah Baris
print("Jumlah baris di tabel product:", validate_query("SELECT COUNT(*) FROM
product"))
print("Jumlah baris di tabel customer:", validate_query("SELECT COUNT(*) FROM
customer"))
print("Jumlah baris di tabel date:", validate_query("SELECT COUNT(*) FROM date"))
print("Jumlah baris di tabel store:", validate_query("SELECT COUNT(*) FROM
store"))
print("Jumlah baris di tabel sales:", validate_query("SELECT COUNT(*) FROM
sales"))

# 2. Periksa Data Sampel
def sample_data(table_name, limit=5):
    cursor.execute(f"SELECT * FROM {table_name} LIMIT {limit}")
    return cursor.fetchall()
print("Sampel data dari tabel product:")
for row in sample_data("product"):
    print(row)
print("Sampel data dari tabel customer:")
for row in sample_data("customer"):
    print(row)
print("Sampel data dari tabel date:")
for row in sample_data("date"):
    print(row)
print("Sampel data dari tabel store:")
for row in sample_data("store"):
    print(row)
print("Sampel data dari tabel sales:")
for row in sample_data("sales"):
    print(row)

# Tutup koneksi
cursor.close()
conn.close()
```

DATA ANALYSIS SQL QUERIES

Total sales amount per product category

```
SELECT p.category, SUM(s.total_amount) AS total_sales_amount
FROM sales s
JOIN product p ON s.product_id = p.product_id
GROUP BY p.category
ORDER BY total_sales_amount DESC;
```

Output:

```
ecommerce_data=# SELECT p.category, SUM(s.total_amount) AS total_sales_amount
ecommerce_data-# FROM sales s
ecommerce_data-# JOIN product p ON s.product_id = p.product_id
ecommerce_data-# GROUP BY p.category
ecommerce_data-# ORDER BY total_sales_amount DESC;
  category   | total_sales_amount
-------------+--------------------
 Electronics |           11360000
 Furniture   |             750000
(2 rows)
```

- Using a join to combine sales tables with product tables based on product_id.
- Group data based on category and calculate the total_amount total for each category.

Top 3 customers based on total purchase amount.

```
SELECT c.customer_name, SUM(s.total_amount) AS total_purchase_amount
FROM sales s
JOIN customer c ON s.customer_id = c.customer_id
GROUP BY c.customer_name
ORDER BY total_purchase_amount DESC
LIMIT 3;
```

Output:

```
ecommerce_data=# SELECT c.customer_name, SUM(s.total_amount) AS total_purchase_amount
ecommerce_data-# FROM sales s
ecommerce_data-# JOIN customer c ON s.customer_id = c.customer_id
ecommerce_data-# GROUP BY c.customer_name
ecommerce_data-# ORDER BY total_purchase_amount DESC
ecommerce_data-# LIMIT 3;
 customer_name | total_purchase_amount
---------------+-----------------------
 Ethan Harris  |               2600000
 Bob Johnson   |               2150000
 John Doe      |               1800000
(3 rows)
```

- Use a join to combine sales tables with customer tables based on customer_id.
- Group data based on customer_name and calculate total total_amount for each customer.
- Sort the results based on total_purchase_amount in the decreased order and limit the results only the top 3.

Monthly sales trend for the first quarter of 2024.

```
SELECT D.MONTH, SUM(S.TOTAL_AMOUNT) AS MONTHLY_SALES_AMOUNT
FROM SALES S
JOIN DATE D ON S.DATE_ID = D.DATE_ID
WHERE D.YEAR = 2024 AND D.MONTH IN (1, 2, 3)
GROUP BY D.MONTH
ORDER BY D.MONTH;
```

OUTPUT:

```
ecommerce_data=# SELECT d.month, SUM(s.total_amount) AS monthly_sales_amount
ecommerce_data-# FROM sales s
ecommerce_data-# JOIN date d ON s.date_id = d.date_id
ecommerce_data-# WHERE d.year = 2024 AND d.month IN (1, 2, 3)
ecommerce_data-# GROUP BY d.month
ecommerce_data-# ORDER BY d.month;
 month | monthly_sales_amount
-------+----------------------
     1 |             12110000
(1 row)
```

- Use a join to combine sales tables with date tables based on date_id.
- Filter data for 2024 and January to March.
- Group data based on month and calculate the total_amount total for each month.

TOTAL QUANTITY SOLD PER STORE.

```
SELECT ST.STORE_NAME, SUM(S.QUANTITY) AS TOTAL_QUANTITY_SOLD
FROM SALES S
JOIN STORE ST ON S.STORE_ID = ST.STORE_ID
GROUP BY ST.STORE_NAME
ORDER BY TOTAL_QUANTITY_SOLD DESC;
```

OUTPUT:

```
ecommerce_data=# SELECT st.store_name, SUM(s.quantity) AS total_quantity_sold
ecommerce_data-# FROM sales s
ecommerce_data-# JOIN store st ON s.store_id = st.store_id
ecommerce_data-# GROUP BY st.store_name
ecommerce_data-# ORDER BY total_quantity_sold DESC;
 store_name | total_quantity_sold
------------+---------------------
 Store B    |                  12
 Store C    |                   6
 Store A    |                   6
(3 rows)
```

- Use a join to combine sales tables with store tables based on store_id.
- Group data based on store_name and calculate the total quantity for each store.

PERFORMANCE OPTIMIZATION

Indexing

Indexes can significantly speed up query execution by allowing the database to quickly locate and retrieve the required data without scanning the entire table.

Create Indexes: Create indexes on columns that are frequently used in `WHERE`, `JOIN`, `ORDER BY`, and `GROUP BY` clauses.

```
CREATE INDEX IF NOT EXISTS idx_sales_product_id ON sales(product_id);
CREATE INDEX IF NOT EXISTS idx_sales_customer_id ON sales(customer_id);
CREATE INDEX IF NOT EXISTS idx_sales_date_id ON sales(date_id);
CREATE INDEX IF NOT EXISTS idx_sales_store_id ON sales(store_id);
```

Partitioning

Partitioning splits a large table into smaller, more manageable pieces, which can improve query performance by allowing the database to scan only relevant partitions.

Range Partitioning: Partition tables by date range to optimize queries that filter by date.

```
CREATE TABLE sales_partitioned (
    transaction_id INT,
    product_id INT,
    customer_id INT,
    date_id INT,
    store_id INT,
    quantity INT,
    total_amount DECIMAL
) PARTITION BY RANGE (date_id);

CREATE TABLE sales_2024 PARTITION OF sales_partitioned FOR VALUES FROM (20240101)
TO (20241231);
```

Denormalization

Denormalization involves combining tables to reduce the number of joins, which can speed up query execution for read-heavy operations.

Materialized Views: Use materialized views to store precomputed results of complex queries.

```
CREATE MATERIALIZED VIEW mv_sales_summary AS
SELECT product_id, customer_id, date_id, store_id, SUM(total_amount) AS
total_sales
FROM sales
GROUP BY product_id, customer_id, date_id, store_id;
```

Query Optimization

Optimize SQL queries to ensure efficient execution plans.

**Avoid SELECT:* Select only the columns you need.

```
-- Instead of this:
SELECT * FROM sales WHERE product_id = 1;

-- Use this:
SELECT transaction_id, total_amount FROM sales WHERE product_id = 1;
```

Use Joins Efficiently: Ensure proper use of join types and conditions.

```
SELECT s.transaction_id, p.product_name
FROM sales s
JOIN product p ON s.product_id = p.product_id
WHERE s.date_id BETWEEN '2024-01-01' AND '2024-03-31';
```

REPORTING

Create a report that shows the sales performance (total amount and quantity sold) by store and product category for the given data.

SQL Query for reporting

Namefile : Query_Reporting.sql

```
SELECT
    st.store_name,
    p.category,
    SUM(s.total_amount) AS total_sales_amount,
    SUM(s.quantity) AS total_quantity_sold
FROM sales s
JOIN product p ON s.product_id = p.product_id
JOIN store st ON s.store_id = st.store_id
GROUP BY
    st.store_name, p.category
ORDER BY
    st.store_name, p.category;
```

Run the python code to export query to excel format and visualize using matplotlib

Namefile : Reporting.py

```
import pandas as pd
import psycopg2

# Koneksi ke PostgreSQL
conn = psycopg2.connect(
    dbname="ecommerce_data",
    user="postgres",
    password="root",
    host="localhost",
    port="5432"
)
cursor = conn.cursor()

# Menjalankan Query untuk Laporan
query = """
SELECT
    st.store_name,
    p.category,
    SUM(s.total_amount) AS total_sales_amount,
    SUM(s.quantity) AS total_quantity_sold
FROM sales s
JOIN product p ON s.product_id = p.product_id
JOIN store st ON s.store_id = st.store_id
GROUP BY
    st.store_name, p.category
ORDER BY
    st.store_name, p.category;
```

```
"""

# Mengambil Data ke DataFrame
sales_report_df = pd.read_sql_query(query, conn)

# Tutup Koneksi
cursor.close()
conn.close()

# Tampilkan DataFrame
print(sales_report_df)

# Simpan DataFrame ke Excel (opsional)
sales_report_df.to_excel("sales_performance_report.xlsx", index=False)

import matplotlib.pyplot as plt

# Membuat Pivot Table untuk Visualisasi
pivot_df = sales_report_df.pivot(index='store_name', columns='category',
values='total_sales_amount')

# Membuat Bar Plot
pivot_df.plot(kind='bar', figsize=(12, 8))
plt.title('Sales Performance by Store and Product Category')
plt.xlabel('Store Name')
plt.ylabel('Total Sales Amount')
plt.legend(title='Product Category')
plt.tight_layout()
plt.show()
```

Visualize