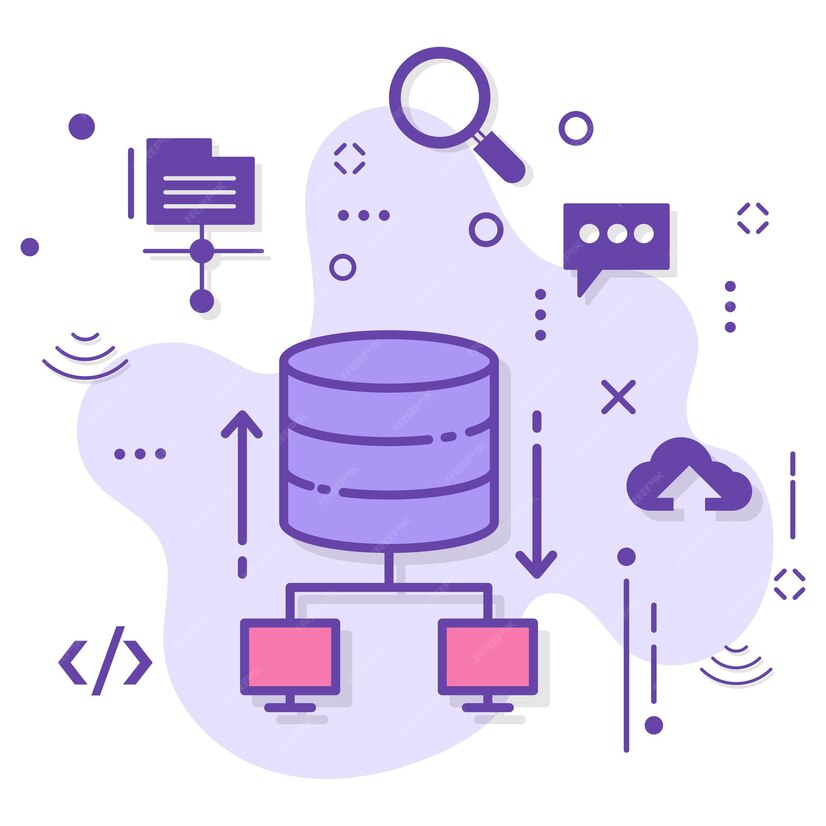
|  |  |  |
| --- | --- | --- |
| PT. Sukses Solusindo Digital |  |  |
| Name Candidate: Mukhlis Aryanto  Email Candidate: aryantomukhlis@gmail.com | Phone Candidate: 083869756993  Github Project: <https://github.com/mukhlisaryanto/DBE_PT_Sukses_Solusindo_Digital> |  |



|  |
| --- |
| TECHNICAL TEST DATABASE ENGINEER |

TABLE OF CONTENTS

[PREPARATION 3](#_Toc171730498)

[Environment 3](#_Toc171730499)

[DATA WAREHOUSE DESIGN 4](#_Toc171730500)

[Star Schema Design 4](#_Toc171730501)

[Run Query To Create Data Warehouse in console psql 5](#_Toc171730502)

[ETL PROCESS 6](#_Toc171730503)

[Extract Load Data using python script 6](#_Toc171730504)

[Run Source code Extract and Load Data 6](#_Toc171730505)

[Run Query Transform Data in console psql 8](#_Toc171730506)

[Data Validation 9](#_Toc171730507)

[DATA ANALYSIS SQL QUERIES 10](#_Toc171730508)

[Total sales amount per product category 10](#_Toc171730509)

[Top 3 customers based on total purchase amount. 10](#_Toc171730510)

[Monthly sales trend for the first quarter of 2024. 11](#_Toc171730511)

[TOTAL QUANTITY SOLD PER STORE. 11](#_Toc171730512)

[PERFORMANCE OPTIMIZATION 12](#_Toc171730513)

[Indexing 12](#_Toc171730514)

[Partitioning 12](#_Toc171730515)

[Denormalization 12](#_Toc171730516)

[Query Optimization 13](#_Toc171730517)

[REPORTING 14](#_Toc171730518)

[SQL Query for reporting 14](#_Toc171730519)

[Run the python code to export query to excel format and visualize using matplotlib 14](#_Toc171730520)

[Visualize 15](#_Toc171730521)

# PREPARATION

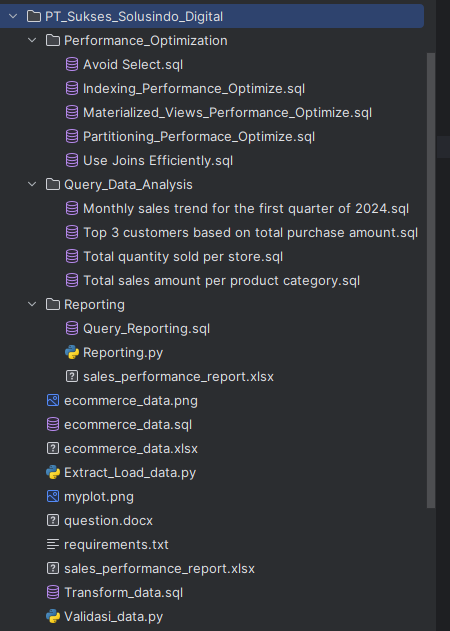
## Environment

DBMS : PostgreSQL version 16 & PgAdmin version 8.2

Code Editor : PyCharm version 2024.1.1

Python : 3.13.3

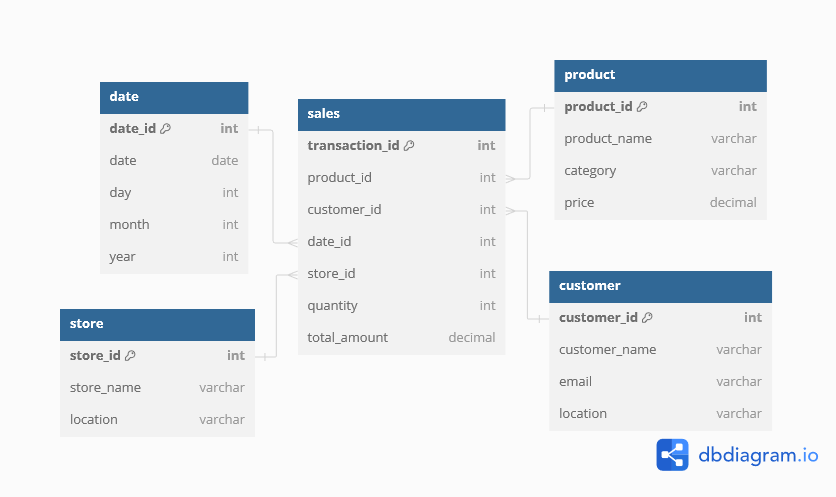
All files into one folder **DBE\_PT\_Sukses\_Solusindo\_Digital**



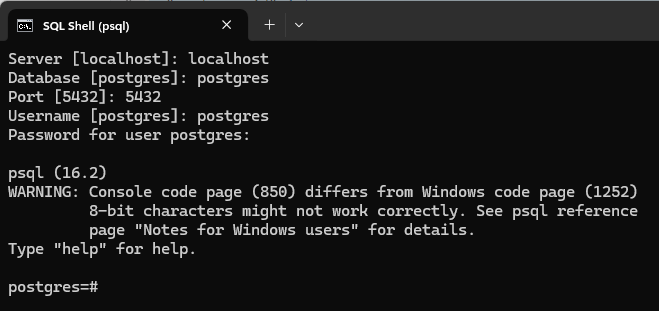
# DATA WAREHOUSE DESIGN

## 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

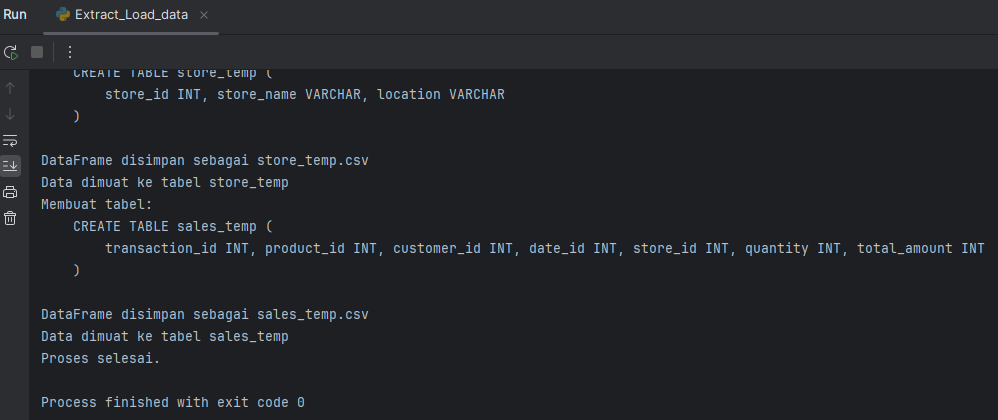
Install dependencies library in terminal

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 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

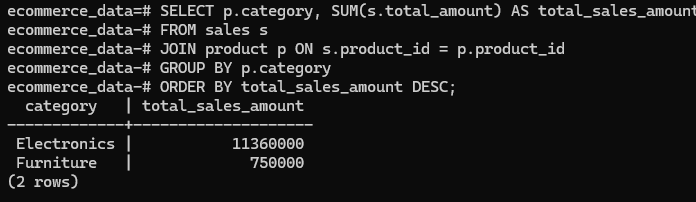
import psycopg2  
  
# Koneksi ke PostgreSQL  
conn = psycopg2.connect(  
 dbname="ecommerce\_data", user="postgres", password="root",host="localhost",  
 port="5432"  
)  
cursor = conn.cursor()  
# Fungsi untuk menjalankan query validasi  
def validate\_query(query):  
 cursor.execute(query)  
 return 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:

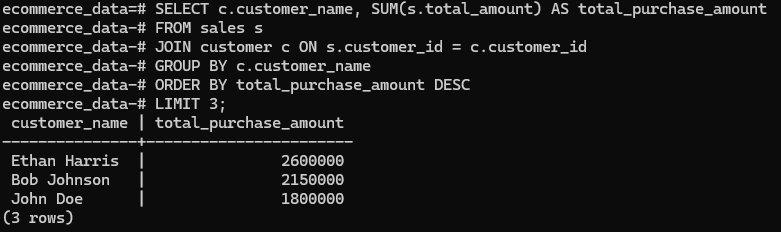


* Using a join to combine sales tables with product tables based on product\_id.
* Group data based on category and calculate the total\_amount total for each category.

## 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:

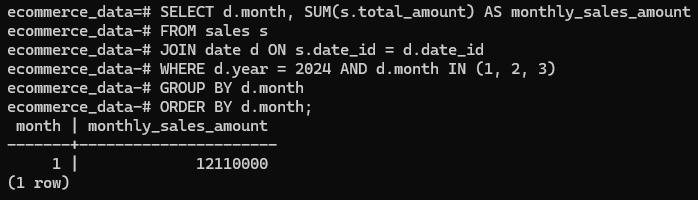


* Use a join to combine sales tables with customer tables based on customer\_id.
* Group data based on customer\_name and calculate total total\_amount for each customer.
* Sort the results based on total\_purchase\_amount in the decreased order and limit the results only the top 3.

## 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:

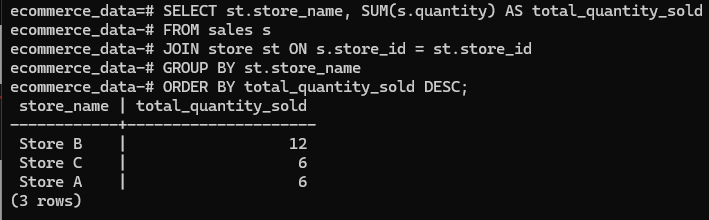


* Use a join to combine sales tables with date tables based on date\_id.
* Filter data for 2024 and January to March.
* Group data based on month and calculate the total\_amount total for each month.

## 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:



* 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

