In [1]:

**import** atoti **as** tt

Welcome to Atoti 0.8.12!

By using this community edition, you agree with the license available at https://doc s.atoti.io/latest/eula.html.

Browse the official documentation at https://docs.atoti.io. Join the community at [https://www.atoti.io/register.](http://www.atoti.io/register)

Atoti collects telemetry data, which is used to help understand how to improve the p roduct.

If you don't wish to send usage data, you can request a trial license at https://ww w.atoti.io/evaluation-license-request.

You can hide this message by setting the `ATOTI\_HIDE\_EULA\_MESSAGE` environment varia ble to True.

|  |  |  |
| --- | --- | --- |
| In | [2]: | session **=** tt**.**Session() |
|  |  |  |
| In | [4]: | dbname **=** "sakila" uname **=** "postgres"  passwd **=** "postgres" |
|  |  | connSql **=** f"postgresql://localhost:55432/{dbname}?user={uname}&password={passwd}" print(connSql) |

postgresql://localhost:55432/sakila?user=postgres&password=postgres

In [25]:

host **=** "localhost" port **=** "55432"

*# Membuat string koneksi*

connSql **=** f"postgresql://{uname}:{passwd}@{host}:{port}/{dbname}" print(connSql)

*# Membuat engine SQLAlchemy*

engine **=** create\_engine(connSql)

postgresql://postgres:postgres@localhost:55432/sakila

In [5]:

products**=**session**.**read\_sql( "select \* from actor;", url**=**connSql,

table\_name**=** "actor", keys**=**{"actor\_id"})

In [6]:

products**.**head()

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Out[6]: | **actor\_id** | **first\_name** | **last\_name** | **last\_update** |
|  | **20** | LUCILLE | TRACY | 2006-02-15 04:34:33 |
|  | **7** | GRACE | MOSTEL | 2006-02-15 04:34:33 |
|  | **30** | SANDRA | PECK | 2006-02-15 04:34:33 |
|  | **23** | SANDRA | KILMER | 2006-02-15 04:34:33 |
|  | **57** | JUDE | CRUISE | 2006-02-15 04:34:33 |

In [34]:

**import** pandas **as** pd

**from** sqlalchemy **import** create\_engine

# 1.Kategori Film Terpopuler dari Masing-masing Negara:

In [39]:

*# Mendefinisikan query untuk mendapatkan kategori film terpopuler dari setiap negar*

query **=** """

SELECT country.country, category.name as category, COUNT(category.name) as category FROM customer

JOIN address ON customer.address\_id = address.address\_id JOIN city ON address.city\_id = city.city\_id

JOIN country ON city.country\_id = country.country\_id

JOIN rental ON customer.customer\_id = rental.customer\_id

JOIN inventory ON rental.inventory\_id = inventory.inventory\_id JOIN film ON inventory.film\_id = film.film\_id

JOIN film\_category ON film.film\_id = film\_category.film\_id

JOIN category ON film\_category.category\_id = category.category\_id GROUP BY country.country, category.name

ORDER BY country.country, category\_count DESC; """

*# Membaca hasil query ke dalam DataFrame*

df **=** pd**.**read\_sql(query, engine)

*# Mendapatkan kategori terpopuler dari setiap negara*

popular\_genres **=** df**.**groupby('country')**.**first()**.**reset\_index() popular\_genres **=** popular\_genres[['country', 'category']]

*# Menampilkan hasil*

popular\_genres

|  |  |  |  |
| --- | --- | --- | --- |
| Out[39]: |  | **country** | **category** |
|  | **0** | Afghanistan | Comedy |
|  | **1** | Algeria | Sci-Fi |
|  | **2** | American Samoa | Sports |
|  | **3** | Angola | Animation |
|  | **4** | Anguilla | Travel |
|  | **...** | ... | ... |
|  | **103** | Vietnam | New |

**104** Virgin Islands, U.S. Animation

**105** Yemen Action

**106** Yugoslavia Animation

**107** Zambia Animation

108 rows × 2 columns

penjelasan langkah-langkah yang dilakukan oleh kode di atas:

1. **Pendefinisian Query SQL**: Query SQL yang diberikan bertujuan untuk mengambil data kategori film terpopuler dari setiap negara. Query tersebut menggabungkan beberapa tabel, seperti tabel customer , address , city , country , rental , inventory ,

film , film\_category , dan category . Query tersebut melakukan JOIN antar tabel berdasarkan hubungan antar kunci primer dan kunci asing di antara tabel-tabel tersebut.

1. **Eksekusi Query dan Pembacaan ke DataFrame**: Setelah query SQL didefinisikan, selanjutnya query dieksekusi menggunakan fungsi pd.read\_sql() . Hasil dari query kemudian dibaca ke dalam DataFrame Pandas yang disimpan dalam variabel df .

untuk fungsi SQL dalam kueri tersebut:

1. **SELECT**: Kita memilih kolom-kolom country.country , category.name (yang di- alias sebagai category ), dan juga menggunakan fungsi agregat

COUNT(category.name) untuk menghitung jumlah kategori film yang ada.

1. **FROM**: Data diambil dari tabel customer , yang bergabung dengan tabel-tabel lainnya untuk mendapatkan informasi yang dibutuhkan.
2. **JOIN**: Dalam total ada 7 JOIN yang dilakukan dengan tabel-tabel berikut:

customer , address , city , dan country : Ini dilakukan untuk mendapatkan informasi tentang negara dari alamat pelanggan.

rental dan inventory : Digunakan untuk mengaitkan transaksi sewa dengan inventaris film.

film : Digunakan untuk mendapatkan informasi tentang film.

film\_category dan category : Untuk mengaitkan kategori film dengan film itu sendiri.

1. **GROUP BY**: Digunakan untuk mengelompokkan hasil berdasarkan nilai-nilai tertentu

dari satu atau beberapa kolom. Di sini, kita mengelompokkan hasil berdasarkan

country.country (negara) dan category.name (kategori film) untuk menghitung jumlah masing-masing kategori film di setiap negara.

1. **COUNT(category.name)**: Fungsi agregat ini digunakan untuk menghitung jumlah kategori film yang muncul dalam setiap kelompok (negara dan kategori).
2. **ORDER BY**: Digunakan untuk mengurutkan hasil berdasarkan kolom tertentu. Di sini, kita mengurutkan hasil berdasarkan nama negara ( country.country ) secara ascending (default), dan jumlah kategori film terbanyak ( category\_count ) dalam setiap negara secara descending ( DESC ).

Setelah menjalankan kueri SQL menggunakan pd.read\_sql() , hasilnya dibaca ke dalam DataFrame Pandas yang disimpan dalam variabel df . Selanjutnya, kita menggunakan metode groupby() untuk mengambil kategori terpopuler dari setiap negara dengan memilih baris pertama dari setiap kelompok negara, dan kemudian menampilkan hasilnya. 3. **Menampilkan Hasil**: Hasil akhir dari kategori film terpopuler dari setiap negara kemudian ditampilkan dengan menggunakan perintah print(popular\_genres) .

Kode tersebut melakukan pengambilan dan pengolahan data untuk mendapatkan kategori film terpopuler dari setiap negara dalam database yang digunakan.

# Sebutkan jumlah customer masing-masing negara!

In [102…

*#kueri SQL untuk menghitung jumlah customer masing-masing negara*

query **=** '''

SELECT c.country, COUNT(\*) AS total\_customers FROM customer cu

JOIN address a ON cu.address\_id = a.address\_id JOIN city ci ON a.city\_id = ci.city\_id

JOIN country c ON ci.country\_id = c.country\_id GROUP BY c.country

'''

*# Jalankan kueri dan baca hasilnya ke dalam DataFrame Pandas*

df\_customers\_by\_country **=** pd**.**read\_sql\_query(query, connSql)

*# Tampilkan DataFrame*

df\_customers\_by\_country

**import** psycopg2

**import** pandas **as** pd

*# Establishing a connection to the database*

conn **=** psycopg2**.**connect( dbname**=**"sakila",

user**=**"postgres",

password**=**"postgres", host**=**"localhost",

port**=**"55432"

)

*# SQL query*

sql\_query **=** """

SELECT country, COUNT(customer\_id) AS total\_customers FROM customer

JOIN address ON customer.address\_id = address.address\_id JOIN city ON address.city\_id = city.city\_id

JOIN country ON city.country\_id = country.country\_id GROUP BY country

ORDER BY total\_customers DESC;

"""

*# Reading data into pandas DataFrame*

df **=** pd**.**read\_sql\_query(sql\_query, conn)

*# Closing the connection*

conn**.**close()

*# Displaying the DataFrame as a table*

print(df)

|  |  |  |
| --- | --- | --- |
|  | country | total\_customers |
| 0 | India | 60 |
| 1 | China | 53 |
| 2 | United States | 36 |
| 3 | Japan | 31 |
| 4 | Mexico | 30 |
| .. | ... | ... |
| 103 | Anguilla | 1 |
| 104 | Brunei | 1 |
| 105 | Tonga | 1 |
| 106 | Nepal | 1 |
| 107 | New Zealand | 1 |

[108 rows x 2 columns]

C:\Users\ASUS\AppData\Local\Temp\ipykernel\_11288\2515987948.py:40: UserWarning: pand as only supports SQLAlchemy connectable (engine/connection) or database string URI o r sqlite3 DBAPI2 connection. Other DBAPI2 objects are not tested. Please consider us ing SQLAlchemy.

df = pd.read\_sql\_query(sql\_query, conn)

alur code sebagai berikut:

* 1. **Pendefinisian Query SQL**: Pertama-tama, sebuah query SQL didefinisikan untuk menghitung jumlah pelanggan (customer) dari setiap negara. Query tersebut melakukan

JOIN antara tabel , city , dan country berdasarkan

address

customer ,

hubungan antara kunci primer dan kunci asing.

* 1. **Eksekusi Query dan Pembacaan ke DataFrame**: Setelah query SQL didefinisikan, langkah selanjutnya adalah mengeksekusi query tersebut menggunakan fungsi

pd.read\_sql\_query() . Hasil dari query kemudian dibaca ke dalam DataFrame Pandas yang disimpan dalam variabel df\_customers\_by\_country .

fungsi SQL yang digunakan dalam kueri tersebut:

1. **SELECT**: Fungsi ini digunakan untuk memilih kolom-kolom dari tabel yang ingin ditampilkan dalam hasil query. Pada kasus ini, kita memilih kolom country dari tabel

country , dan juga menggunakan fungsi agregat COUNT(\*) untuk menghitung jumlah baris yang dihasilkan setelah pengelompokan.

1. **FROM**: Fungsi ini menentukan tabel-tabel mana yang digunakan dalam query. Di sini, kita menggunakan tabel customer , address , city , dan country sebagai sumber data.
2. **JOIN**: Fungsi ini digunakan untuk menggabungkan data dari beberapa tabel berdasarkan kunci-kunci yang sesuai. Kita menggunakan klausa JOIN untuk menghubungkan tabel customer dengan tabel address , city , dan country sesuai dengan relasi kunci yang telah ditentukan.
3. **ON**: Klausa ini digunakan untuk menentukan kriteria penggabungan antara dua tabel. Pada kasus ini, kita menggunakan klausa ON untuk menentukan relasi antara kunci- kunci dalam tabel yang di-join.
4. **GROUP BY**: Fungsi ini digunakan untuk mengelompokkan hasil query berdasarkan nilai- nilai tertentu dari satu atau beberapa kolom. Di sini, kita mengelompokkan hasil berdasarkan nilai kolom country .
5. **COUNT(\*)**: Fungsi agregat ini digunakan untuk menghitung jumlah baris yang dihasilkan setelah pengelompokan. Dalam konteks ini, kita menghitung jumlah baris yang mewakili jumlah customer untuk setiap negara.
6. **Tampilkan DataFrame**: DataFrame yang berisi jumlah pelanggan masing-masing negara kemudian ditampilkan dengan menggunakan perintah

print(df\_customers\_by\_country) .

Dengan demikian, kode ini melakukan pengambilan dan pengolahan data untuk menghitung jumlah pelanggan dari setiap negara dalam database yang digunakan. Hasilnya adalah DataFrame yang berisi dua kolom: nama negara dan jumlah pelanggan di negara tersebut.

# 5 Aktor yang Paling Banyak Bermain Film

In [101…

*#kueri SQL untuk mencari 5 Aktor yang Paling Banyak Bermain Film*

query **=** ''' SELECT

CONCAT(a.first\_name,' ', a.last\_name) AS full\_name, COUNT(\*) AS total\_films

FROM actor a

JOIN film\_actor fa ON a.actor\_id = fa.actor\_id GROUP BY a.actor\_id, full\_name

ORDER BY total\_films DESC LIMIT 5;

'''

*# Jalankan kueri dan baca hasilnya ke dalam DataFrame Pandas*

jumlah\_customer\_pernegara **=** pd**.**read\_sql\_query(query, connSql)

*# Tampilkan DataFrame*

jumlah\_customer\_pernegara

|  |  |  |  |
| --- | --- | --- | --- |
| Out[101… |  | **full\_name** | **total\_films** |
|  | **0** | GINA DEGENERES | 42 |
|  | **1** | WALTER TORN | 41 |
|  | **2** | MARY KEITEL | 40 |
|  | **3** | MATTHEW CARREY | 39 |
|  | **4** | SANDRA KILMER | 37 |

Alur dari kueri SQL sebagai berikut:

* 1. **FROM Clause**: Kueri dimulai dengan mengambil data dari tabel actor dan

film\_actor . Ini dilakukan dengan menggabungkan tabel-tabel ini menggunakan klausa JOIN berdasarkan kolom actor\_id .

* 1. **SELECT Clause**: Kueri ini memilih dua kolom untuk ditampilkan dalam hasil kueri:

CONCAT(a.first\_name,' ', a.last\_name) AS full\_name : Menggabungkan kolom first\_name dan last\_name dari tabel actor menjadi satu kolom bernama full\_name .

COUNT(\*) AS total\_films : Menghitung jumlah baris atau film yang dimainkan oleh setiap aktor.

* 1. **GROUP BY Clause**: Karena kita menggunakan fungsi agregat ( COUNT ) dalam klausa

SELECT, kita perlu mengelompokkan data dengan kolom yang tidak diagregasi. Kita mengelompokkan data berdasarkan actor\_id (ID aktor) dan full\_name (nama lengkap aktor).

* 1. **ORDER BY Clause**: Hasil kueri akan diurutkan berdasarkan total\_films (jumlah film yang dimainkan oleh aktor) secara menurun ( DESC ).
  2. **LIMIT Clause**: Karena kita hanya tertarik pada lima aktor teratas yang paling banyak bermain dalam film, kita menggunakan klausa LIMIT 5 untuk membatasi hasil hanya

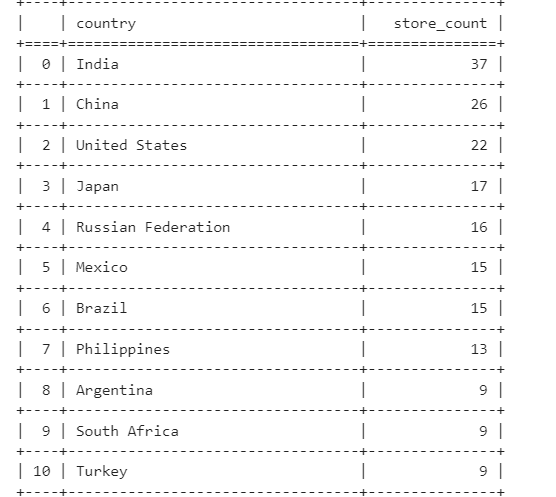
untuk lima baris pertama.

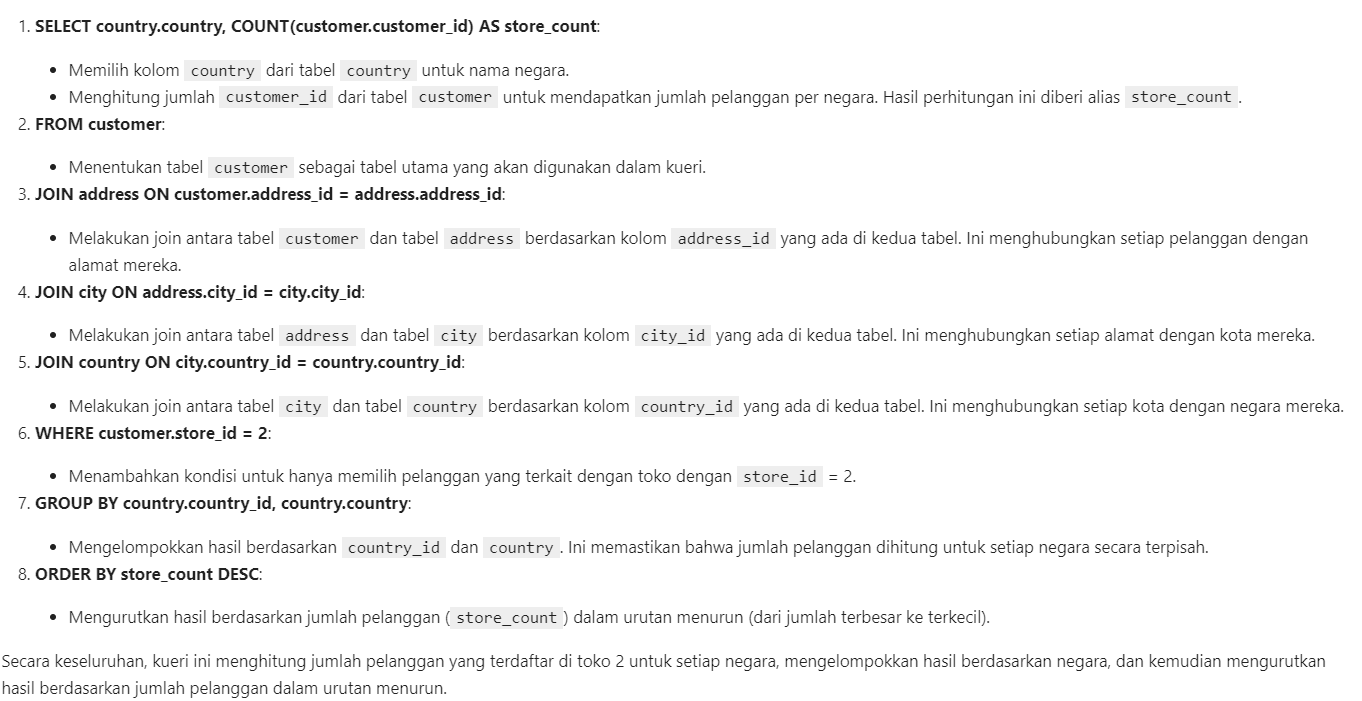
Jadi, hasil kueri ini akan memberikan lima aktor teratas yang paling banyak bermain dalam film, diurutkan berdasarkan jumlah film yang dimainkan oleh mereka.

# Jumlah Store 1 di Masing-masing Negara:



Output:





# Jumlah Store 2 di Masing-masing Negara:

# Menulis query SQL untuk menghitung jumlah store 2 di masing-masing negara

query = """

SELECT country.country, COUNT(customer.customer\_id) AS store\_count

FROM customer

JOIN address ON customer.address\_id = address.address\_id

JOIN city ON address.city\_id = city.city\_id

JOIN country ON city.country\_id = country.country\_id

WHERE customer.store\_id = 2

GROUP BY country.country\_id, country.country

ORDER BY store\_count DESC;

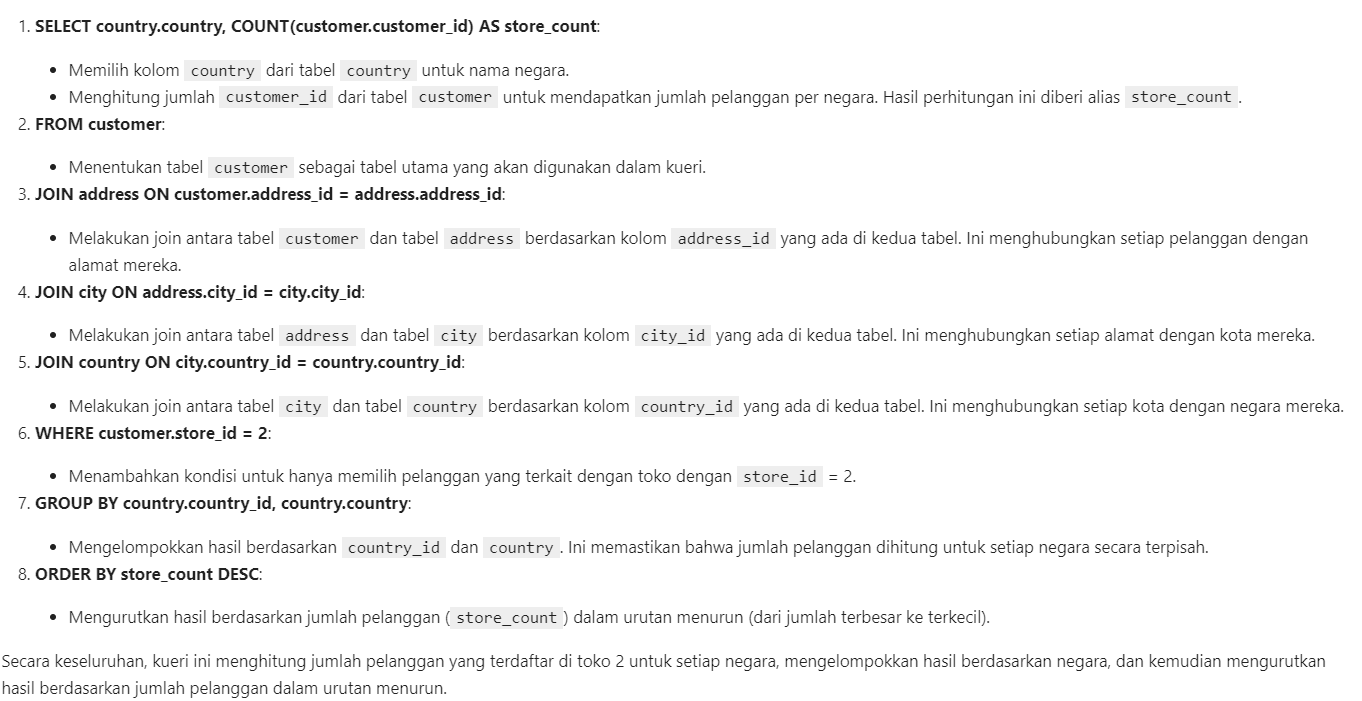
**“””**

In [44]:

**Output:**

****

Alur dari kueri SQL sebagai berikut:



# 6. Jumlah Revenue Store 1 di Setiap Negara:

In [45]:

*#kueri SQL untuk menghitung Jumlah Revenue Store 1 di Setiap Negara:*

query **=** '''

SELECT c.country, SUM(p.amount) AS total\_revenue\_store\_1 FROM payment p

JOIN rental r ON p.rental\_id = r.rental\_id

JOIN inventory i ON r.inventory\_id = i.inventory\_id JOIN store s ON i.store\_id = s.store\_id

JOIN address a ON s.address\_id = a.address\_id JOIN city ci ON a.city\_id = ci.city\_id

JOIN country c ON ci.country\_id = c.country\_id WHERE s.store\_id = 1

GROUP BY c.country '''

*# Jalankan kueri dan baca hasilnya ke dalam DataFrame Pandas*

jumlah\_revenue\_store\_1**=** pd**.**read\_sql\_query(query, connSql)

*# Tampilkan DataFrame*

jumlah\_revenue\_store\_1

Out[45]:

**country total\_revenue\_store\_1 0** Canada 33689.74

Alur kueri SQL sebagai berikut:

1. **FROM Clause**: Kueri dimulai dengan mengambil data dari tabel ,

rental

payment ,

, store , address , city , dan country . Ini dilakukan dengan

inventory

menggabungkan tabel-tabel ini menggunakan klausa JOIN berdasarkan kunci hubung yang sesuai. Misalnya, JOIN rental r ON p.rental\_id = r.rental\_id menggabungkan tabel payment dengan rental berdasarkan kolom rental\_id .

1. **SELECT Clause**: Kueri ini memilih dua kolom untuk ditampilkan dalam hasil kueri:

c.country : Kolom ini menunjukkan nama negara.

SUM(p.amount) AS total\_revenue\_store\_1 : Menghitung jumlah pendapatan dari setiap transaksi pembayaran ( amount ) di toko dengan ID 1 (store\_id = 1) di setiap negara.

1. **WHERE Clause**: Klausa WHERE digunakan untuk memfilter baris berdasarkan kondisi tertentu. Di sini, kita membatasi hasil hanya pada toko dengan ID 1 (store\_id = 1).
2. **GROUP BY Clause**: Karena kita menggunakan fungsi agregat ( SUM ) dalam klausa SELECT, kita perlu mengelompokkan data dengan kolom yang tidak diagregasi. Kita mengelompokkan data berdasarkan nama negara ( c.country ).

Jadi, hasil kueri ini akan memberikan total pendapatan dari toko dengan ID 1 di setiap negara.

# 7. Jumlah Revenue Store 2 di Setiap Negara:

In [46]:

*#kueri SQL untuk menghitung Jumlah Revenue Store 2 di Setiap Negara:*

query **=** '''

SELECT c.country, SUM(p.amount) AS total\_revenue\_store\_2 FROM payment p

JOIN rental r ON p.rental\_id = r.rental\_id

JOIN inventory i ON r.inventory\_id = i.inventory\_id JOIN store s ON i.store\_id = s.store\_id

JOIN address a ON s.address\_id = a.address\_id JOIN city ci ON a.city\_id = ci.city\_id

JOIN country c ON ci.country\_id = c.country\_id WHERE s.store\_id = 2

GROUP BY c.country

'''

*# Jalankan kueri dan baca hasilnya ke dalam DataFrame Pandas*

jumlah\_revenue\_store\_2 **=** pd**.**read\_sql\_query(query, connSql)

*# Tampilkan DataFrame*

jumlah\_revenue\_store\_2

Out[46]:

**country total\_revenue\_store\_2**

**0** Australia 33726.77

Alur kueri SQL sebagai berikut:

1. **FROM Clause**: Kueri dimulai dengan mengambil data dari tabel ,

rental

payment ,

, store , address , city , dan country . Ini dilakukan dengan

inventory

menggabungkan tabel-tabel ini menggunakan klausa JOIN berdasarkan kunci hubung yang sesuai. Misalnya, JOIN rental r ON p.rental\_id = r.rental\_id menggabungkan tabel payment dengan rental berdasarkan kolom rental\_id .

1. **SELECT Clause**: Kueri ini memilih dua kolom untuk ditampilkan dalam hasil kueri:

c.country : Kolom ini menunjukkan nama negara.

SUM(p.amount) AS total\_revenue\_store\_2 : Menghitung jumlah pendapatan dari setiap transaksi pembayaran ( amount ) di toko dengan ID 2 (store\_id = 2) di setiap negara.

1. **WHERE Clause**: Klausa WHERE digunakan untuk memfilter baris berdasarkan kondisi tertentu. Di sini, kita membatasi hasil hanya pada toko dengan ID 2 (store\_id = 2).
2. **GROUP BY Clause**: Karena kita menggunakan fungsi agregat ( SUM ) dalam klausa SELECT, kita perlu mengelompokkan data dengan kolom yang tidak diagregasi. Kita mengelompokkan data berdasarkan nama negara ( c.country ).

Jadi, hasil kueri ini akan memberikan total pendapatan dari toko dengan ID 2 di setiap negara.

In [90]:

**import** pandas **as** pd

**from** sqlalchemy **import** create\_engine

*# Konfigurasi koneksi ke database PostgreSQL*

dbname **=** "sakila" uname **=** "postgres"

passwd **=** "postgres" host **=** "localhost" port **=** "55432"

*# Membuat string koneksi*

connSql **=** f"postgresql://{uname}:{passwd}@{host}:{port}/{dbname}"

*# Membuat engine SQLAlchemy*

engine **=** create\_engine(connSql)

*# Mendefinisikan query untuk mendapatkan jumlah revenue Store 2 dari setiap negara*

query **=** """ SELECT

country.country AS country,

SUM(payment.amount) AS revenue

FROM

payment

JOIN rental ON payment.rental\_id = rental.rental\_id

JOIN customer ON rental.customer\_id = customer.customer\_id JOIN address ON customer.address\_id = address.address\_id

JOIN city ON address.city\_id = city.city\_id

JOIN country ON city.country\_id = country.country\_id WHERE

customer.store\_id = 2 GROUP BY

country.country ORDER BY

revenue DESC;

"""

*# Membaca hasil query ke dalam DataFrame*

df **=** pd**.**read\_sql(query, engine)

*# Menampilkan DataFrame*

print(df)

|  |  |  |
| --- | --- | --- |
|  | country | revenue |
| 0 | China | 2770.07 |
| 1 | India | 2410.17 |
| 2 | Mexico | 1741.86 |
| 3 | United States | 1701.01 |
| 4 | Japan | 1512.43 |
| .. | ... | ... |
| 76 | Israel | 84.83 |
| 77 | Kazakstan | 77.77 |
| 78 | American Samoa | 71.80 |
| 79 | Switzerland | 58.86 |
| 80 | Sudan | 57.81 |

[81 rows x 2 columns]

Alur dari kueri SQL sebagai berikut:

1. **FROM Clause**: Kueri dimulai dengan mengambil data dari tabel ,

rental

payment ,

, , dan country . Ini dilakukan dengan menggabungkan

address

city

customer ,

tabel-tabel ini menggunakan klausa JOIN berdasarkan kunci hubung yang sesuai. Misalnya, JOIN rental ON payment.rental\_id = rental.rental\_id menggabungkan tabel payment dengan rental berdasarkan kolom rental\_id .

1. **SELECT Clause**: Kueri ini memilih dua kolom untuk ditampilkan dalam hasil kueri:

country.country AS country : Kolom ini menunjukkan nama negara.

SUM(payment.amount) AS revenue : Menghitung total pendapatan ( amount ) dari setiap transaksi pembayaran ( payment ) yang dilakukan oleh pelanggan dari toko dengan ID 2 ( customer.store\_id = 2 ) di setiap negara.

1. **WHERE Clause**: Klausa WHERE digunakan untuk memfilter baris berdasarkan kondisi tertentu. Di sini, kita membatasi hasil hanya pada pelanggan yang terkait dengan toko dengan ID 2 ( customer.store\_id = 2 ).
2. **GROUP BY Clause**: Karena kita menggunakan fungsi agregat ( SUM ) dalam klausa SELECT, kita perlu mengelompokkan data dengan kolom yang tidak diagregasi. Kita mengelompokkan data berdasarkan nama negara ( country.country ).
3. **ORDER BY Clause**: Hasil kueri akan diurutkan berdasarkan total pendapatan ( revenue ) secara menurun ( DESC ).

Jadi, hasil kueri ini akan memberikan total pendapatan dari toko dengan ID 2 di setiap negara, diurutkan dari total pendapatan tertinggi hingga terendah.

# Jumlah Revenue Masing-masing Store Setiap Minggunya:

In [89]:

**import** psycopg2

**import** pandas **as** pd

**import** matplotlib.pyplot **as** plt

*# Establishing a connection to the database*

conn **=** psycopg2**.**connect( dbname**=**"sakila",

user**=**"postgres",

password**=**"postgres", host**=**"localhost",

port**=**"55432"

)

*# SQL query to get weekly revenue for each store*

sql\_query **=** """ SELECT

store.store\_id,

EXTRACT(WEEK FROM payment.payment\_date) AS week, EXTRACT(YEAR FROM payment.payment\_date) AS year, SUM(payment.amount) AS weekly\_revenue

FROM

payment

C:\Users\ASUS\AppData\Local\Temp\ipykernel\_11288\3250296664.py:36: UserWarning: pand as only supports SQLAlchemy connectable (engine/connection) or database string URI o r sqlite3 DBAPI2 connection. Other DBAPI2 objects are not tested. Please consider us ing SQLAlchemy.

df = pd.read\_sql\_query(sql\_query, conn)

JOIN

rental ON payment.rental\_id = rental.rental\_id

JOIN

inventory ON rental.inventory\_id = inventory.inventory\_id

JOIN

store ON inventory.store\_id = store.store\_id GROUP BY

store.store\_id, year, week ORDER BY

store.store\_id, year, week;

"""

*# Reading data into pandas DataFrame*

df **=** pd**.**read\_sql\_query(sql\_query, conn)

*# Closing the connection*

conn**.**close()

*# Ensure 'year' and 'week' are integers*

df['year'] **=** df['year']**.**astype(int)

df['week'] **=** df['week']**.**astype(int)

*# Calculate week\_date directly*

df['week\_date'] **=** pd**.**to\_datetime(df['year']**.**astype(str) **+** df['week']**.**astype(str) **+**

*# Plotting the weekly revenue for each store*

stores **=** df['store\_id']**.**unique()

**for** store\_id **in** stores:

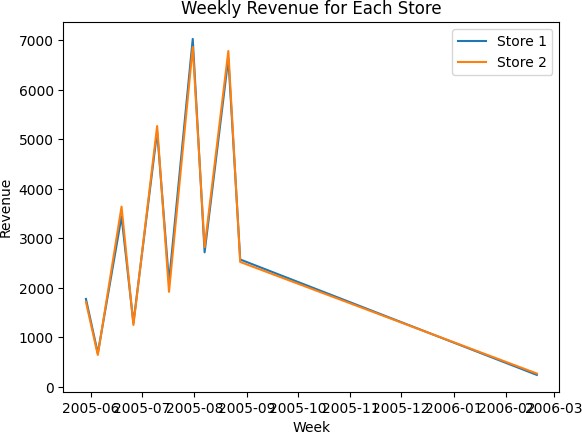
store\_data **=** df[df['store\_id'] **==** store\_id]

plt**.**plot(store\_data['week\_date'], store\_data['weekly\_revenue'], label**=**f'Store {

plt**.**title('Weekly Revenue for Each Store') plt**.**xlabel('Week')

plt**.**ylabel('Revenue') plt**.**legend()

plt**.**show() df



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Out[89]: | **store\_id** | **week** | **year** | **weekly\_revenue** | **week\_date** |
|  | **0** 1 | 21 | 2005 | 1777.87 | 2005-05-29 |
|  | **1** 1 | 22 | 2005 | 682.37 | 2005-06-05 |
|  | **2** 1 | 24 | 2005 | 3442.77 | 2005-06-19 |
|  | **3** 1 | 25 | 2005 | 1294.01 | 2005-06-26 |
|  | **4** 1 | 27 | 2005 | 5159.84 | 2005-07-10 |
|  | **5** 1 | 28 | 2005 | 2123.09 | 2005-07-17 |
|  | **6** 1 | 30 | 2005 | 7030.71 | 2005-07-31 |
|  | **7** 1 | 31 | 2005 | 2719.64 | 2005-08-07 |
|  | **8** 1 | 33 | 2005 | 6641.48 | 2005-08-21 |
|  | **9** 1 | 34 | 2005 | 2574.86 | 2005-08-28 |
|  | **10** 1 | 7 | 2006 | 243.10 | 2006-02-19 |
|  | **11** 2 | 21 | 2005 | 1717.77 | 2005-05-29 |
|  | **12** 2 | 22 | 2005 | 646.42 | 2005-06-05 |
|  | **13** 2 | 24 | 2005 | 3643.17 | 2005-06-19 |
|  | **14** 2 | 25 | 2005 | 1251.93 | 2005-06-26 |
|  | **15** 2 | 27 | 2005 | 5270.19 | 2005-07-10 |
|  | **16** 2 | 28 | 2005 | 1921.34 | 2005-07-17 |
|  | **17** 2 | 30 | 2005 | 6868.72 | 2005-07-31 |
|  | **18** 2 | 31 | 2005 | 2824.22 | 2005-08-07 |
|  | **19** 2 | 33 | 2005 | 6787.04 | 2005-08-21 |
|  | **20** 2 | 34 | 2005 | 2524.89 | 2005-08-28 |
|  | **21** 2 | 7 | 2006 | 271.08 | 2006-02-19 |

Alur kueri SQL sebagai berikut:

* 1. **FROM Clause**: Kueri dimulai dengan mengambil data dari tabel payment , yang berisi informasi tentang setiap transaksi pembayaran.
  2. **JOIN Clauses**: Kueri menggunakan beberapa klausa JOIN untuk menggabungkan tabel-

tabel lain yang terkait dengan transaksi pembayaran:

JOIN rental ON payment.rental\_id = rental.rental\_id : Menggabungkan tabel rental , yang berisi informasi tentang setiap transaksi penyewaan film, berdasarkan kolom rental\_id .

JOIN inventory ON rental.inventory\_id = inventory.inventory\_id : Menggabungkan tabel inventory , yang berisi informasi tentang setiap barang

inventaris film yang tersedia untuk disewakan, berdasarkan kolom inventory\_id .

JOIN store ON inventory.store\_id = store.store\_id : Menggabungkan tabel store , yang berisi informasi tentang setiap toko yang menyediakan layanan sewa film, berdasarkan kolom store\_id .

* 1. **SELECT Clause**: Kueri ini memilih beberapa kolom untuk ditampilkan dalam hasil kueri:

store.store\_id : Kolom ini menunjukkan ID dari setiap toko.

EXTRACT(WEEK FROM payment.payment\_date) AS week : Kolom ini mengekstrak informasi minggu (week) dari tanggal pembayaran.

EXTRACT(YEAR FROM payment.payment\_date) AS year : Kolom ini

mengekstrak informasi tahun (year) dari tanggal pembayaran.

SUM(payment.amount) AS weekly\_revenue : Kolom ini menghitung total pendapatan mingguan (weekly revenue) untuk setiap toko.

* 1. **GROUP BY Clause**: Kueri menggunakan klausa GROUP BY untuk mengelompokkan hasil berdasarkan ID toko, tahun, dan minggu.
  2. **ORDER BY Clause**: Hasil kueri akan diurutkan berdasarkan ID toko, tahun, dan minggu untuk menjaga urutan data yang benar.
  3. **Manipulasi Data**: Setelah mendapatkan data, baris yang terkait dengan minggu dan

tahun dikonversi menjadi tanggal dengan bantuan kolom 'week' dan 'year'. Ini membantu dalam visualisasi data.

Jadi, hasil kueri ini memberikan informasi tentang pendapatan mingguan untuk setiap toko dalam basis data Sakila, yang dapat digunakan untuk menganalisis kinerja penjualan setiap toko dari waktu ke waktu.

# Lamanya Peminjaman Masing-masing Customer:

In [87]:

**import** psycopg2

**import** pandas **as** pd

*# Establishing a connection to the database*

conn **=** psycopg2**.**connect( dbname**=**"sakila",

user**=**"postgres",

password**=**"postgres", host**=**"localhost",

port**=**"55432"

)

*# SQL query to calculate the rental duration for each customer and group them*

sql\_query **=** """ SELECT

customer\_id, CASE

WHEN rental\_duration BETWEEN 2 AND 4 THEN '2 - 4 days'

WHEN rental\_duration BETWEEN 5 AND 7 THEN '5 - 7 days'

WHEN rental\_duration BETWEEN 8 AND 9 THEN '8 - 9 days'

WHEN rental\_duration BETWEEN 10 AND 14 THEN '10 - 14 days' ELSE 'More than 14 days'

END AS rental\_duration\_group,

COUNT(\*) AS customer\_count

FROM

(

SELECT

rental.customer\_id,

EXTRACT(DAY FROM rental.return\_date - rental.rental\_date) AS rental

FROM

rental

) AS subquery GROUP BY

customer\_id,

rental\_duration\_group ORDER BY

customer\_id,

rental\_duration\_group;

"""

*# Reading data into pandas DataFrame*

df **=** pd**.**read\_sql\_query(sql\_query, conn)

*# Closing the connection*

conn**.**close()

*# Displaying the DataFrame*

df

C:\Users\ASUS\AppData\Local\Temp\ipykernel\_11288\3017912100.py:42: UserWarning: pand as only supports SQLAlchemy connectable (engine/connection) or database string URI o r sqlite3 DBAPI2 connection. Other DBAPI2 objects are not tested. Please consider us ing SQLAlchemy.

df = pd.read\_sql\_query(sql\_query, conn)

|  |  |  |  |
| --- | --- | --- | --- |
| Out[87]: | **customer\_id** | **rental\_duration\_group** | **customer\_count** |
|  | **0** 1 | 2 - 4 days | 11 |
|  | **1** 1 | 5 - 7 days | 7 |
|  | **2** 1 | 8 - 9 days | 6 |
|  | **3** 1 | More than 14 days | 8 |
|  | **4** 2 | 2 - 4 days | 9 |
|  | **...** ... | ... | ... |
|  | **2381** 598 | 8 - 9 days | 5 |
|  | **2382** 599 | 2 - 4 days | 4 |
|  | **2383** 599 | 5 - 7 days | 8 |
|  | **2384** 599 | 8 - 9 days | 4 |
|  | **2385** 599 | More than 14 days | 3 |

2386 rows × 3 columns

Alur dari kueri SQL sebagai berikut:

* 1. **FROM Clause**: Kueri dimulai dengan mengambil data dari tabel rental , yang berisi informasi tentang setiap transaksi penyewaan film.
  2. **Subquery**: Kueri ini menggunakan subquery untuk menghitung durasi penyewaan untuk setiap pelanggan. Subquery ini mengambil data dari tabel rental dan menghitung durasi penyewaan dengan mengurangi tanggal pengembalian

( return\_date ) dari tanggal penyewaan ( rental\_date ). Hasilnya adalah jumlah hari yang dibutuhkan pelanggan untuk mengembalikan film setelah menyewanya.

* 1. **SELECT Clause**: Kueri ini memilih beberapa kolom untuk ditampilkan dalam hasil kueri:

customer\_id : ID pelanggan yang melakukan transaksi penyewaan.

CASE ... END AS rental\_duration\_group : Klausa CASE digunakan untuk mengelompokkan durasi penyewaan ke dalam kategori yang lebih bermakna. Misalnya, durasi penyewaan antara 2 dan 4 hari akan masuk ke dalam kategori '2 - 4 days'.

COUNT(\*) AS customer\_count : Menghitung jumlah pelanggan untuk setiap kategori durasi penyewaan.

* 1. **GROUP BY Clause**: Kueri ini menggunakan klausa GROUP BY untuk mengelompokkan

hasil berdasarkan ID pelanggan dan kategori durasi penyewaan.

* 1. **ORDER BY Clause**: Hasil kueri akan diurutkan berdasarkan ID pelanggan dan kategori durasi penyewaan untuk menjaga urutan data yang benar.

Jadi, hasil kueri ini memberikan informasi tentang jumlah pelanggan yang melakukan transaksi penyewaan film dalam berbagai kategori durasi penyewaan. Kategori durasi penyewaan ini membantu untuk menganalisis kebiasaan penyewaan pelanggan dan pola peminjaman film.

# 10. Revenue Masing-masing Kategori Film:

In [103…

**import** psycopg2

**import** pandas **as** pd

**import** matplotlib.pyplot **as** plt

*# Establishing a connection to the database*

conn **=** psycopg2**.**connect( dbname**=**"sakila",

user**=**"postgres",

password**=**"postgres", host**=**"localhost",

port**=**"55432"

)

*# SQL query to calculate revenue for each film category*

sql\_query **=** """

C:\Users\ASUS\AppData\Local\Temp\ipykernel\_11288\2306563887.py:38: UserWarning: pand as only supports SQLAlchemy connectable (engine/connection) or database string URI o r sqlite3 DBAPI2 connection. Other DBAPI2 objects are not tested. Please consider us ing SQLAlchemy.

df = pd.read\_sql\_query(sql\_query, conn)

SELECT

c.name AS category,

SUM(p.amount) AS total\_revenue

FROM

film\_category fc

JOIN

film f ON fc.film\_id = f.film\_id

JOIN

category c ON fc.category\_id = c.category\_id

JOIN

inventory i ON f.film\_id = i.film\_id

JOIN

rental r ON i.inventory\_id = r.inventory\_id

JOIN

payment p ON r.rental\_id = p.rental\_id GROUP BY

c.name ORDER BY

total\_revenue DESC;

"""

*# Reading data into pandas DataFrame*

df **=** pd**.**read\_sql\_query(sql\_query, conn)

*# Closing the connection*

conn**.**close()

*# Plotting the revenue for each film category*

plt**.**figure(figsize**=**(10, 6))

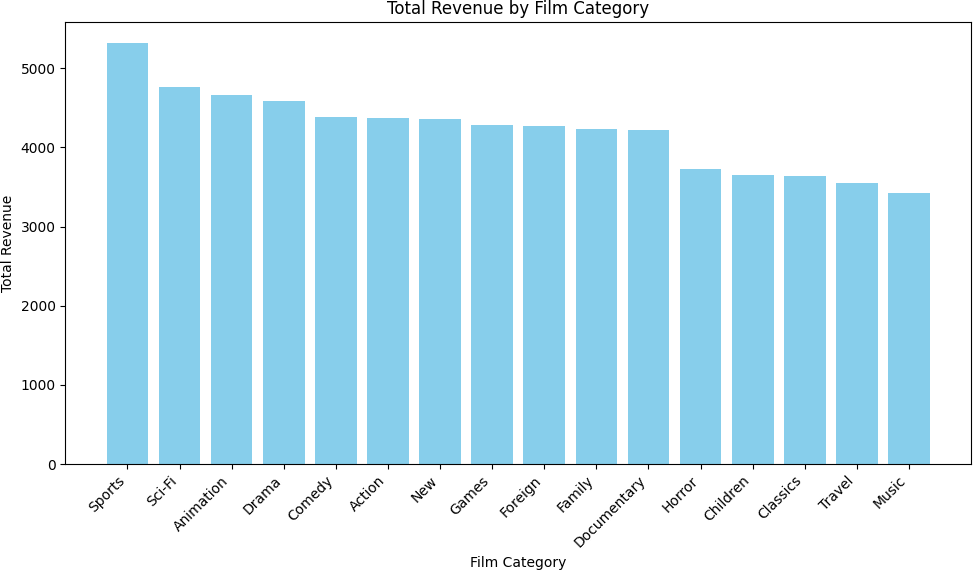
plt**.**bar(df['category'], df['total\_revenue'], color**=**'skyblue') plt**.**xlabel('Film Category')

plt**.**ylabel('Total Revenue')

plt**.**title('Total Revenue by Film Category') plt**.**xticks(rotation**=**45, ha**=**'right')

plt**.**tight\_layout() plt**.**show()

df



|  |  |  |  |
| --- | --- | --- | --- |
| Out[103… |  | **category** | **total\_revenue** |
|  | **0** | Sports | 5314.21 |
|  | **1** | Sci-Fi | 4756.98 |
|  | **2** | Animation | 4656.30 |
|  | **3** | Drama | 4587.39 |
|  | **4** | Comedy | 4383.58 |
|  | **5** | Action | 4375.85 |
|  | **6** | New | 4352.61 |
|  | **7** | Games | 4281.33 |
|  | **8** | Foreign | 4270.67 |
|  | **9** | Family | 4235.03 |
|  | **10** | Documentary | 4217.52 |
|  | **11** | Horror | 3722.54 |
|  | **12** | Children | 3655.55 |
|  | **13** | Classics | 3639.59 |
|  | **14** | Travel | 3549.64 |
|  | **15** | Music | 3417.72 |

Alur dari kueri SQL sebagai berikut:

1. **FROM Clause**: Kueri dimulai dengan mengambil data dari tabel film\_category , yang berisi informasi tentang kategori film. Tabel ini digabungkan dengan tabel film untuk mendapatkan informasi lebih lanjut tentang film-film dalam setiap kategori. Selanjutnya, tabel category digabungkan untuk mendapatkan nama kategori berdasarkan ID kategori.
2. **JOIN Clauses**: Kueri menggunakan beberapa klausa JOIN untuk menggabungkan tabel- tabel terkait:

JOIN film ON film\_category.film\_id = film.film\_id : Menggabungkan tabel film dengan film\_category berdasarkan ID film.

JOIN category ON film\_category.category\_id = category.category\_id : Menggabungkan tabel category dengan film\_category berdasarkan ID kategori.

JOIN inventory ON film.film\_id = inventory.film\_id : Menggabungkan tabel inventory dengan film untuk mendapatkan informasi tentang inventaris film.

JOIN rental ON inventory.inventory\_id = rental.inventory\_id : Menggabungkan tabel rental dengan inventory untuk mendapatkan informasi tentang transaksi penyewaan film.

JOIN payment ON rental.rental\_id = payment.rental\_id : Menggabungkan tabel payment dengan rental untuk mendapatkan informasi tentang transaksi pembayaran film.

1. **SELECT Clause**: Kueri ini memilih dua kolom untuk ditampilkan dalam hasil kueri:

c.name AS category : Kolom ini menunjukkan nama kategori film.

SUM(p.amount) AS total\_revenue : Menghitung total pendapatan ( amount ) dari setiap transaksi pembayaran ( payment ) yang terkait dengan film-film dalam setiap kategori.

1. **GROUP BY Clause**: Kueri ini menggunakan klausa GROUP BY untuk mengelompokkan hasil berdasarkan nama kategori film ( c.name ). Ini memungkinkan kita untuk menghitung total pendapatan untuk setiap kategori film.
2. **ORDER BY Clause**: Hasil kueri akan diurutkan berdasarkan total pendapatan

( total\_revenue ) secara menurun ( DESC ). Ini akan membantu dalam menampilkan kategori film dengan pendapatan tertinggi di bagian atas.

Jadi, hasil kueri ini memberikan informasi tentang total pendapatan yang dihasilkan oleh setiap kategori film, yang dapat digunakan untuk menganalisis performa relatif dari kategori film dalam basis data Sakila.

In [68]:

**import** atoti **as** tt

In [69]:

sessions **=** tt**.**Session()

Closing existing "Unnamed" session to create the new one.

In [70]:

dbname **=** "northwind" uname **=** "postgres" passwd **=** "postgres"

connSql **=** f"postgresql://localhost:55432/{dbname}?user={uname}&password={passwd}" print(connSql)

postgresql://localhost:55432/northwind?user=postgres&password=postgres

# Mencoba membaca beberapa kolom

In [71]:

data **=** sessions**.**read\_sql(

"select \* from public.Categories;", url**=**connSql,

table\_name**=**"Categories", keys**=**{"category\_id"},)

In [72]:

data**.**head()

Out[72]:

**category\_id**

**category\_name description picture**

**5** Grains/Cereals Breads, crackers, pasta, and cereal

**7** Produce Dried fruit and bean curd

**3** Confections Desserts, candies, and sweet breads

**1** Beverages Soft drinks, coffees, teas, beers, and ales

**2** Condiments Sweet and savory sauces, relishes, spreads, an...

# Import dan Membaca kolom database

In [73]:

**import** pandas **as** pd

**from** sqlalchemy **import** create\_engine

**import** sqlite3

In [85]:

**from** sqlalchemy **import** create\_engine, MetaData

*# Membuat koneksi dengan database PostgreSQL*

engine **=** create\_engine(connSql)

*# Membuat objek MetaData*

metadata **=** MetaData()

*# Memuat metadata dari database*

metadata**.**reflect(bind**=**engine)

*# Mendapatkan daftar semua tabel* tables **=** metadata**.**tables**.**keys() tables

Out[85]:

In [108…

**import** pandas **as** pd

**from** sqlalchemy **import** create\_engine, text *# Buat koneksi dengan database PostgreSQL* engine **=** create\_engine(connSql)

*# Buat kueri SQL untuk mendapatkan total penjualan per customer, per tahun, dan per*

query **=** """

SELECT c.customer\_id,

EXTRACT(YEAR FROM o.order\_date) AS order\_year, p.category\_id,

SUM(od.quantity \* od.unit\_price) AS total\_sales FROM orders o

JOIN customers c ON o.customer\_id = c.customer\_id JOIN order\_details od ON o.order\_id = od.order\_id JOIN products p ON od.product\_id = p.product\_id

GROUP BY c.customer\_id, order\_year, p.category\_id; """

*# Jalankan kueri dan simpan hasilnya ke dalam DataFrame*

rekap\_customer **=** pd**.**read\_sql\_query(query, engine)

*# Tampilkan DataFrame*

rekap\_customer

dict\_keys(['territories', 'region', 'order\_details', 'orders', 'customers', 'emplo yees', 'shippers', 'products', 'categories', 'suppliers', 'employee\_territories', 'us\_states', 'customer\_demographics', 'customer\_customer\_demo'])

# Sebutkan total jumlah penjualan per customer, per tahun, dan per kategori produk!

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Out[108… | **customer\_id** | **order\_year** | **category\_id** | **total\_sales** |
|  | **0** BERGS | 1996.0 | 1 | 229.199993 |
|  | **1** KOENE | 1997.0 | 4 | 4642.000000 |
|  | **2** ERNSH | 1997.0 | 3 | 10736.199951 |
|  | **3** WELLI | 1998.0 | 1 | 783.000000 |
|  | **4** ROMEY | 1996.0 | 3 | 7.300000 |
|  | **...** ... | ... | ... | ... |
|  | **1071** WANDK | 1997.0 | 3 | 510.399998 |
|  | **1072** WHITC | 1997.0 | 6 | 2136.999981 |
|  | **1073** SAVEA | 1998.0 | 3 | 4636.650017 |
|  | **1074** FRANR | 1998.0 | 1 | 360.000000 |
|  | **1075** FRANS | 1998.0 | 8 | 500.000000 |

1076 rows × 4 columns

Alur kueri yang dilakukan melalui agregasi data penjualan dari tabel ,

customers

orders ,

order\_details , dan products berdasarkan customer, tahun pesanan ( order\_year ),

dan kategori produk ( category\_id ). Kueri tersebut kemudian menghitung total penjualan dengan mengalikan jumlah produk yang dibeli ( od.quantity ) dengan harga satuan

produk ( od.unit\_price ), dan menjumlahkannya untuk setiap kelompok customer, tahun, dan kategori produk.

Berikut adalah alur kueri secara lebih detail:

* 1. Kueri menghubungkan tabel orders , customers , order\_details , dan products menggunakan klausa JOIN untuk memungkinkan akses ke informasi yang diperlukan.
  2. Kueri menggunakan klausa GROUP BY untuk mengelompokkan data berdasarkan

customer\_id , order\_year (tahun dari tanggal pesanan), dan category\_id (kategori produk).

* 1. Dalam klausa SELECT, kueri menggunakan fungsi EXTRACT untuk mengekstraksi tahun dari order\_date dan menghitung total penjualan dengan mengalikan quantity

dengan unit\_price dari setiap produk ( SUM(od.quantity \* od.unit\_price) ).

* 1. Hasil kueri disimpan dalam DataFrame rekap\_customer menggunakan

pd.read\_sql\_query .

Jadi, kueri tersebut memberikan rangkuman (rekapitulasi) total penjualan per customer, per tahun, dan per kategori produk.

# Sebutkan tiga employee dengan penjualan terbanyak!

In [113…

**import** pandas **as** pd

**from** sqlalchemy **import** create\_engine, text *# Buat koneksi dengan database PostgreSQL* engine **=** create\_engine(connSql)

*# Buat kueri SQL untuk mendapatkan tiga employee dengan penjualan terbanyak*

query **=** """

SELECT CONCAT(e.first\_name, ' ', e.last\_name) AS full\_name FROM employees e

JOIN orders o ON e.employee\_id = o.employee\_id

JOIN order\_details od ON o.order\_id = od.order\_id GROUP BY e.employee\_id, full\_name

ORDER BY SUM(od.quantity \* od.unit\_price) DESC LIMIT 3;

"""

*# Jalankan kueri dan simpan hasilnya ke dalam DataFrame*

tiga\_terbanyak **=** pd**.**read\_sql\_query(query, engine)

*# Tampilkan DataFrame*

tiga\_terbanyak

|  |  |  |
| --- | --- | --- |
| Out[113… |  | **full\_name** |
|  | **0** | Margaret Peacock |
|  | **1** | Janet Leverling |
|  | **2** | Nancy Davolio |

penjelasan alur dari kueri SQL:

* 1. **FROM clause**: Kueri dimulai dengan mengambil data dari tabel employees , orders , dan order\_details . Ini dilakukan menggunakan klausa JOIN untuk menggabungkan data dari tabel-tabel ini berdasarkan relasi antara mereka. Misalnya, JOIN orders o ON e.employee\_id = o.employee\_id menggabungkan data dari tabel employees dan orders berdasarkan kolom employee\_id .
  2. **SELECT clause**: Kita memilih kolom first\_name dan last\_name dari tabel

employees . Namun, karena kita ingin menampilkan nama lengkap, kita menggunakan fungsi CONCAT untuk menggabungkan nama depan dan belakang menjadi satu kolom bernama full\_name . Ini dilakukan dengan menambahkan spasi di antara nama depan dan belakang menggunakan tanda kutip spasi: CONCAT(e.first\_name, ' ',

e.last\_name) AS full\_name .

* 1. **GROUP BY clause**: Karena kita menggunakan fungsi agregasi ( SUM ) dalam klausa SELECT, kita perlu mengelompokkan data berdasarkan kolom yang tidak diagregasi. Dalam hal ini, kita ingin mengelompokkan data berdasarkan employee\_id (untuk memastikan setiap karyawan hanya muncul satu kali dalam hasil) dan full\_name (nama lengkap).
  2. **ORDER BY clause**: Kita ingin mengurutkan hasil berdasarkan total penjualan dari yang tertinggi ke yang terendah. Oleh karena itu, kita menggunakan klausa ORDER BY

SUM(od.quantity \* od.unit\_price) DESC untuk mengurutkan hasil berdasarkan jumlah penjualan secara menurun ( DESC ).

* 1. **LIMIT clause**: Akhirnya, kita hanya ingin menampilkan tiga baris pertama dari hasil (yaitu, tiga karyawan dengan penjualan terbanyak). Kita mencapai ini dengan

menggunakan klausa LIMIT 3 .

Dengan demikian, kueri tersebut menghasilkan daftar tiga karyawan dengan penjualan

terbanyak, diurutkan berdasarkan total penjualan mereka, dan hanya menampilkan nama lengkap mereka.

# Sebutkan employee dengan penjualan terbanyak per produk dan per tahun!

In [115…

**import** pandas **as** pd

**from** sqlalchemy **import** create\_engine, text

*# Buat koneksi dengan database PostgreSQL*

engine **=** create\_engine(connSql)

*# Buat kueri SQL untuk mendapatkan employee dengan penjualan terbanyak per produk d*

query **=** """ SELECT

CONCAT(e.first\_name, ' ', e.last\_name) AS full\_name, EXTRACT(YEAR FROM o.order\_date) AS order\_year,

p.product\_id,

p.product\_name,

SUM(od.quantity \* od.unit\_price) AS total\_sales

FROM

employees e

JOIN orders o ON e.employee\_id = o.employee\_id

JOIN order\_details od ON o.order\_id = od.order\_id JOIN products p ON od.product\_id = p.product\_id

GROUP BY

full\_name, order\_year,

p.product\_id, p.product\_name

ORDER BY

order\_year,

p.product\_id,

total\_sales DESC;

"""

*# Jalankan kueri dan simpan hasilnya ke dalam DataFrame*

terbanyak\_employe **=** pd**.**read\_sql\_query(query, engine)

*# Tampilkan DataFrame*

terbanyak\_employe

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Out[115… |  | **full\_name** | **order\_year** | **product\_id** | **product\_name** | **total\_sales** |
|  | **0** | Nancy Davolio | 1996.0 | 1 | Chai | 647.999983 |
|  | **1** | Michael Suyama | 1996.0 | 1 | Chai | 503.999987 |
| **2** | | Margaret Peacock | 1996.0 | 1 | Chai | 475.199987 |
| **3** | | Laura Callahan | 1996.0 | 1 | Chai | 172.799995 |
| **4** | | Nancy Davolio | 1996.0 | 2 | Chang | 1139.999986 |
| **...** | | ... | ... | ... | ... | ... |

**1148** Nancy Davolio 1998.0 77 Original Frankfurter grüne

Soße

1092.000000

**1149** Michael Suyama

1998.0

77

Original Frankfurter grüne

Soße

793.000000

**1150** Andrew Fuller 1998.0 77 Original Frankfurter grüne

Soße

754.000000

**1151**

Steven Buchanan

1998.0

77

Original Frankfurter grüne

Soße

650.000000

**1152** Laura Callahan 1998.0 77 Original Frankfurter grüne

Soße

559.000000

1153 rows × 5 columns

Alur dari kueri SQL sebagai berikut:

* 1. **FROM Clause**: Kueri dimulai dengan mengambil data dari tabel employees , orders ,

order\_details , dan products . Ini dilakukan dengan menggabungkan tabel-tabel ini menggunakan klausa JOIN berdasarkan kunci hubung yang sesuai. Misalnya, JOIN orders o ON e.employee\_id = o.employee\_id menggabungkan tabel

employees dengan orders berdasarkan kolom employee\_id .

* 1. **SELECT Clause**: Kita memilih beberapa kolom untuk ditampilkan dalam hasil kueri:

CONCAT(e.first\_name, ' ', e.last\_name) AS full\_name : Ini

menggabungkan kolom first\_name dan last\_name dari tabel employees menjadi satu kolom bernama full\_name .

EXTRACT(YEAR FROM o.order\_date) AS order\_year : Ini mengekstrak tahun dari kolom order\_date dari tabel orders dan memberikan alias order\_year .

p.product\_id dan p.product\_name : Kolom ini mengidentifikasi produk yang

dijual.

SUM(od.quantity \* od.unit\_price) AS total\_sales : Ini menghitung total penjualan untuk setiap kombinasi employee, tahun, dan produk dengan

mengalikan jumlah barang yang dibeli ( od.quantity ) dengan harga satuan barang ( od.unit\_price ) untuk setiap pesanan dan menjumlahkannya.

* 1. **GROUP BY Clause**: Karena kita menggunakan fungsi agregat ( SUM ) dalam klausa SELECT, kita perlu mengelompokkan data dengan kolom-kolom yang tidak diagregasi. Kita mengelompokkan data berdasarkan full\_name , order\_year , p.product\_id , dan p.product\_name .
  2. **ORDER BY Clause**: Hasil kueri akan diurutkan berdasarkan order\_year ,

p.product\_id , dan total penjualan ( total\_sales ) secara menurun ( DESC ),

sehingga memberikan kita employee dengan penjualan terbanyak per produk dan per tahun.

Jadi, hasil kueri ini akan memberikan daftar employee dengan penjualan terbanyak untuk setiap produk dan tahun yang ada dalam basis data.

# Sebutkan negara dengan penjualan terbanyak! Visualisasikan!

In [97]:

**from** sqlalchemy **import** create\_engine

**import** matplotlib.pyplot **as** plt

*# Konfigurasi koneksi ke database PostgreSQL*

dbname **=** "northwind" uname **=** "postgres" passwd **=** "postgres" host **=** "localhost"

port **=** "55432"

*# Membuat string koneksi*

connSql **=** f"postgresql://{uname}:{passwd}@{host}:{port}/{dbname}" print(connSql)

*# Membuat engine SQLAlchemy*

engine **=** create\_engine(connSql)

*# Mendefinisikan query untuk mendapatkan total penjualan per negara*

query **=** """ SELECT

c.country,

SUM(od.unit\_price \* od.quantity) AS total\_sales FROM

orders o JOIN

customers c ON o.customer\_id = c.customer\_id JOIN

order\_details od ON o.order\_id = od.order\_id GROUP BY

c.country ORDER BY

total\_sales DESC; """

*# Membaca hasil query ke dalam DataFrame # Membaca hasil query ke dalam DataFrame* data **=** pd**.**read\_sql(query, engine)

*# Visualisasi data*

plt**.**figure(figsize**=**(14, 8))

plt**.**bar(df['country'], df['total\_sales'], color**=**'skyblue') plt**.**title('Total Sales per Country')

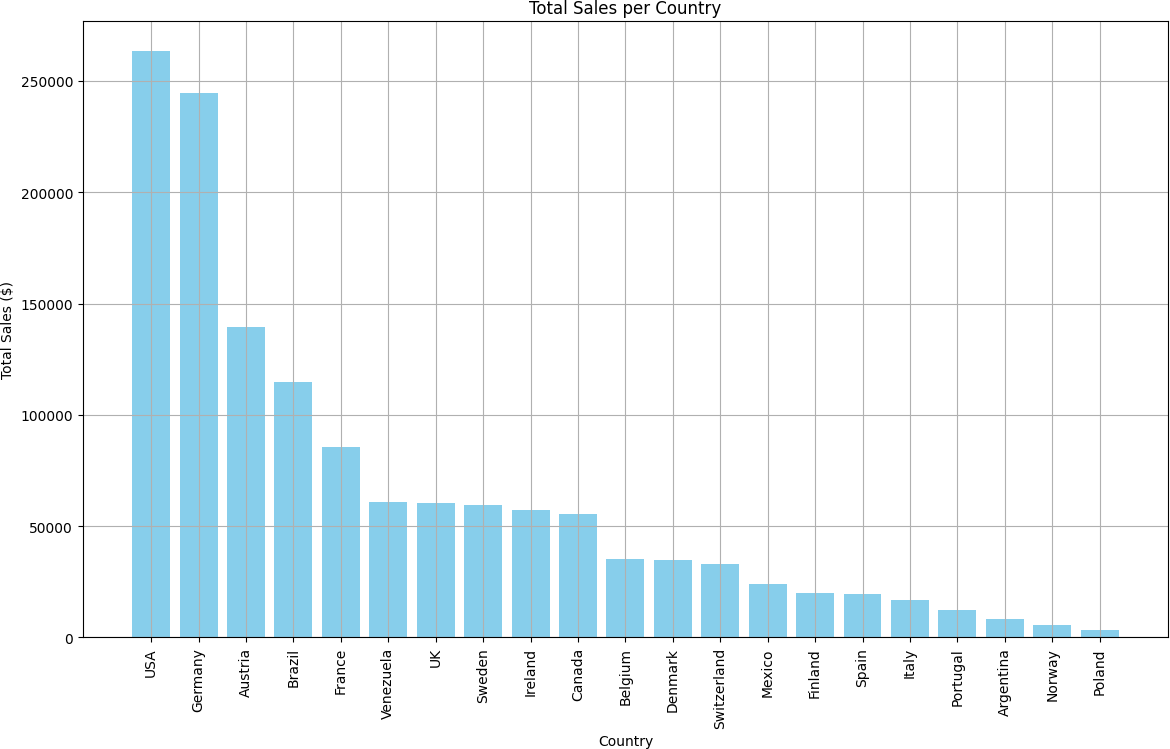
plt**.**xlabel('Country')

plt**.**ylabel('Total Sales ($)') plt**.**xticks(rotation**=**90)

plt**.**grid(**True**) plt**.**show()

*# Menampilkan DataFrame*

data

postgresql://postgres:postgres@localhost:55432/northwind

|  |  |  |  |
| --- | --- | --- | --- |
| Out[97]: |  | **country** | **total\_sales** |
|  | **0** | USA | 263566.980017 |
|  | **1** | Germany | 244640.629969 |
|  | **2** | Austria | 139496.629867 |
|  | **3** | Brazil | 114968.480040 |
|  | **4** | France | 85498.760092 |
| **5** Venezuela 60814.889832 | | | |
| **6** | | UK | 60616.509948 |
| **7** Sweden 59523.699950 | | | |
| **8** | | Ireland | 57317.390162 |
| **9** Canada 55334.100187 | | | |
| **10** | | Belgium | 35134.980321 |

**11** Denmark 34782.250007

**12** Switzerland 32919.499998

**13** Mexico 24073.449984

**14** Finland 19778.450103

**15** Spain 19431.889992

**16** Italy 16705.149927

**17** Portugal 12468.650044

**18** Argentina 8119.099991

**19** Norway 5735.150015

**20** Poland 3531.949996

Alur dari kueri SQL sebagai berikut:

* 1. **FROM Clause**: Kueri dimulai dengan mengambil data dari tabel orders , customers , dan order\_details . Ini dilakukan dengan menggabungkan tabel-tabel ini

menggunakan klausa JOIN berdasarkan kunci hubung yang sesuai. Misalnya, JOIN

customers c ON o.customer\_id = c.customer\_id menggabungkan tabel

orders dengan customers berdasarkan kolom customer\_id .

* 1. **SELECT Clause**: Kita memilih dua kolom untuk ditampilkan dalam hasil kueri:

c.country : Kolom ini menunjukkan negara tempat pelanggan berada.

SUM(od.unit\_price \* od.quantity) AS total\_sales : Ini menghitung total penjualan untuk setiap negara dengan mengalikan harga satuan barang

( od.unit\_price ) dengan jumlah barang yang dibeli ( od.quantity ) untuk setiap pesanan dan menjumlahkannya.

* 1. **GROUP BY Clause**: Karena kita menggunakan fungsi agregat ( SUM ) dalam klausa SELECT, kita perlu mengelompokkan data dengan kolom yang tidak diagregasi. Kita mengelompokkan data berdasarkan c.country , yaitu negara.
  2. **ORDER BY Clause**: Hasil kueri akan diurutkan berdasarkan total penjualan

( total\_sales ) secara menurun ( DESC ), sehingga memberikan kita total penjualan per negara yang terbesar.

Jadi, hasil kueri ini akan memberikan total penjualan per negara, yang kemudian diurutkan dari yang tertinggi ke terendah. Dalam kode yang diberikan, hasil kueri ini kemudian

disajikan dalam bentuk bar chart menggunakan Matplotlib.

# Sebutkan total penjualan dan rata-rata penjualan bulanan berdasarkan employee dan tahun!

In [121…

**import** pandas **as** pd

**from** sqlalchemy **import** create\_engine, text *# Buat koneksi dengan database PostgreSQL* engine **=** create\_engine(connSql)

*# Buat kueri SQL untuk mendapatkan total penjualan dan rata-rata penjualan bulanan*

query **=** """ SELECT

CONCAT(e.first\_name, ' ', e.last\_name) AS full\_name, EXTRACT(YEAR FROM o.order\_date) AS order\_year,

EXTRACT(MONTH FROM o.order\_date) AS order\_month, SUM(od.quantity \* od.unit\_price) AS total\_sales,

AVG(od.quantity \* od.unit\_price) AS average\_monthly\_sales FROM employees e

JOIN orders o ON e.employee\_id = o.employee\_id

JOIN order\_details od ON o.order\_id = od.order\_id

GROUP BY e.employee\_id, e.first\_name, e.last\_name, order\_year, order\_month ORDER BY e.employee\_id, order\_year, order\_month;

"""

*# Jalankan kueri dan simpan hasilnya ke dalam DataFrame*

penjualan\_employe **=** pd**.**read\_sql\_query(query, engine)

*# Tampilkan DataFrame*

penjualan\_employe

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Out[121… | **full\_name** | **order\_year** | **order\_month** | **total\_sales** | **average\_monthly\_sales** |
|  | **0** Nancy Davolio | 1996.0 | 7.0 | 2018.599993 | 672.866664 |
|  | **1** Nancy Davolio | 1996.0 | 8.0 | 6007.100028 | 500.591669 |
|  | **2** Nancy Davolio | 1996.0 | 9.0 | 6883.700181 | 529.515399 |
|  | **3** Nancy Davolio | 1996.0 | 10.0 | 4061.399942 | 507.674993 |
|  | **4** Nancy Davolio | 1996.0 | 11.0 | 10261.200030 | 932.836366 |
|  | **...** ... | ... | ... | ... | ... |
|  | **187** Anne Dodsworth | 1997.0 | 12.0 | 1941.500000 | 388.300000 |
|  | **188** Anne Dodsworth | 1998.0 | 1.0 | 5627.139982 | 432.856922 |
|  | **189** Anne Dodsworth | 1998.0 | 2.0 | 19325.510010 | 1756.864546 |
|  | **190** Anne Dodsworth | 1998.0 | 3.0 | 7566.599998 | 630.550000 |
|  | **191** Anne Dodsworth | 1998.0 | 4.0 | 9501.499993 | 950.149999 |

192 rows × 5 columns

Alur dari kueri SQL sebagai berikut:

1. **FROM Clause**: Kueri dimulai dengan mengambil data dari tabel employees , orders , dan order\_details . Ini dilakukan dengan menggabungkan tabel-tabel ini

menggunakan klausa JOIN berdasarkan kunci hubung yang sesuai. Misalnya, JOIN orders o ON e.employee\_id = o.employee\_id menggabungkan tabel

employees dengan orders berdasarkan kolom employee\_id .

1. **SELECT Clause**: Kueri ini memilih beberapa kolom untuk ditampilkan dalam hasil kueri:

CONCAT(e.first\_name, ' ', e.last\_name) AS full\_name : Menggabungkan kolom first\_name dan last\_name dari tabel employees menjadi satu kolom bernama full\_name .

EXTRACT(YEAR FROM o.order\_date) AS order\_year : Mengekstrak tahun dari kolom order\_date dari tabel orders dan memberikan alias order\_year .

EXTRACT(MONTH FROM o.order\_date) AS order\_month : Mengekstrak bulan

dari kolom order\_date dari tabel orders dan memberikan alias

order\_month .

SUM(od.quantity \* od.unit\_price) AS total\_sales : Menghitung total penjualan untuk setiap employee, tahun, dan bulan dengan mengalikan harga satuan barang ( od.unit\_price ) dengan jumlah barang yang dibeli

( od.quantity ) untuk setiap pesanan dan menjumlahkannya.

AVG(od.quantity \* od.unit\_price) AS average\_monthly\_sales :

Menghitung rata-rata penjualan bulanan untuk setiap employee, tahun, dan bulan

dengan menghitung rata-rata dari hasil perkalian harga satuan barang dengan jumlah barang yang dibeli.

1. **GROUP BY Clause**: Karena kita menggunakan fungsi agregat ( SUM dan AVG ) dalam klausa SELECT, kita perlu mengelompokkan data dengan kolom yang tidak diagregasi. Kita mengelompokkan data berdasarkan e.employee\_id , e.first\_name ,

e.last\_name , order\_year , dan order\_month .

1. **ORDER BY Clause**: Hasil kueri akan diurutkan berdasarkan e.employee\_id ,

order\_year , dan order\_month .

Jadi, hasil kueri ini akan memberikan total penjualan dan rata-rata penjualan bulanan untuk setiap employee, tahun, dan bulan yang ada dalam basis data.

# 6. Sebutkan total penjualan dan diskon per produk dan per bulan!

In [127…

**import** pandas **as** pd

**from** sqlalchemy **import** create\_engine, text *# Buat koneksi dengan database PostgreSQL* engine **=** create\_engine(connSql)

*# Buat kueri SQL untuk mendapatkan total penjualan dan diskon per produk dan per bu*

query **=** """ SELECT

p.product\_name,

EXTRACT(MONTH FROM o.order\_date) AS order\_month, SUM(od.quantity \* od.unit\_price) AS total\_sales, SUM(od.quantity \* od.discount) AS total\_discount

FROM

orders o

JOIN

order\_details od ON o.order\_id = od.order\_id

JOIN

products p ON od.product\_id = p.product\_id GROUP BY

p.product\_name, order\_month

ORDER BY

order\_month,

p.product\_name;

"""

*# Jalankan kueri dan simpan hasilnya ke dalam DataFrame*

produk **=** pd**.**read\_sql\_query(query, engine)

*# Tampilkan DataFrame*

produk

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Out[127… | **product\_name** | **order\_month** | **total\_sales** | **total\_discount** |
|  | **0** Alice Mutton | 1.0 | 4898.400074 | 21.50 |
|  | **1** Aniseed Syrup | 1.0 | 1190.000000 | 0.00 |
|  | **2** Boston Crab Meat | 1.0 | 2880.399946 | 21.70 |
|  | **3** Camembert Pierrot | 1.0 | 6936.000046 | 12.00 |
|  | **4** Carnarvon Tigers | 1.0 | 3437.500000 | 6.25 |
|  | **...** ... | ... | ... | ... |
|  | **771** Uncle Bob's Organic Dried Pears | 12.0 | 1380.000000 | 2.45 |
|  | **772** Valkoinen suklaa | 12.0 | 195.000000 | 0.00 |
|  | **773** Vegie-spread | 12.0 | 1878.600021 | 1.60 |
|  | **774** Wimmers gute Semmelknödel | 12.0 | 1961.750011 | 7.00 |
|  | **775** Zaanse koeken | 12.0 | 285.000000 | 0.00 |

776 rows × 4 columns

Alur kueri SQL sebagai berikut:

1. **FROM Clause**: Kueri dimulai dengan mengambil data dari tabel orders ,

order\_details , dan products . Ini dilakukan dengan menggabungkan tabel-tabel ini menggunakan klausa JOIN berdasarkan kunci hubung yang sesuai. Misalnya, JOIN order\_details od ON o.order\_id = od.order\_id menggabungkan tabel

orders dengan order\_details berdasarkan kolom order\_id .

1. **SELECT Clause**: Kueri ini memilih beberapa kolom untuk ditampilkan dalam hasil kueri:

p.product\_name : Kolom ini menunjukkan nama produk.

EXTRACT(MONTH FROM o.order\_date) AS order\_month : Mengambil bulan dari tanggal pesanan menggunakan fungsi EXTRACT dan memberikan alias

order\_month .

SUM(od.quantity \* od.unit\_price) AS total\_sales : Menghitung total

penjualan untuk setiap produk dan bulan dengan mengalikan harga satuan barang ( od.unit\_price ) dengan jumlah barang yang dibeli ( od.quantity ) untuk

setiap pesanan dan menjumlahkannya.

SUM(od.quantity \* od.discount) AS total\_discount : Menghitung total diskon yang diberikan untuk setiap produk dan bulan dengan mengalikan jumlah barang yang dibeli dengan diskon yang diberikan ( od.discount ) untuk setiap pesanan dan menjumlahkannya.

1. **GROUP BY Clause**: Karena kita menggunakan fungsi agregat ( SUM ) dalam klausa SELECT, kita perlu mengelompokkan data dengan kolom yang tidak diagregasi. Kita

mengelompokkan data berdasarkan p.product\_name (nama produk) dan

order\_month (bulan dari tanggal pesanan).

1. **ORDER BY Clause**: Hasil kueri akan diurutkan berdasarkan bulan ( order\_month ) dan nama produk ( p.product\_name ).

Jadi, hasil kueri ini akan memberikan total penjualan dan diskon per produk per bulan, diurutkan berdasarkan bulan dan nama produk.

# 7. Sebutkan total penjualan, banyaknya produk, dan jumlah produk terjual untuk masing-masing pemesanan (order)?

In [128…

**import** pandas **as** pd

**from** sqlalchemy **import** create\_engine, text *# Buat koneksi dengan database PostgreSQL* engine **=** create\_engine(connSql)

*# Buat kueri SQL untuk mendapatkan total penjualan, banyaknya produk, dan jumlah pr*

query **=** """ SELECT

o.order\_id,

COUNT(od.product\_id) AS total\_products\_ordered, SUM(od.quantity) AS total\_products\_sold,

SUM(od.quantity \* od.unit\_price) AS total\_sales FROM

orders o JOIN

order\_details od ON o.order\_id = od.order\_id GROUP BY

o.order\_id ORDER BY

o.order\_id; """

*# Jalankan kueri dan simpan hasilnya ke dalam DataFrame*

rekap\_penjualan **=** pd**.**read\_sql\_query(query, engine)

*# Tampilkan DataFrame*

rekap\_penjualan

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Out[128… | **order\_id** | **total\_products\_ordered** | **total\_products\_sold** | **total\_sales** |
|  | **0** 10248 | 3 | 27 | 439.999998 |
|  | **1** 10249 | 2 | 49 | 1863.400064 |
|  | **2** 10250 | 3 | 60 | 1813.000040 |
|  | **3** 10251 | 3 | 41 | 670.799986 |
|  | **4** 10252 | 3 | 105 | 3730.000153 |
|  | **...** ... | ... | ... | ... |
|  | **825** 11073 | 2 | 30 | 300.000000 |
|  | **826** 11074 | 1 | 14 | 244.300011 |
|  | **827** 11075 | 3 | 42 | 586.000000 |
|  | **828** 11076 | 3 | 50 | 1056.999998 |
|  | **829** 11077 | 25 | 72 | 1374.600000 |

830 rows × 4 columns

Alur kueri SQL sebagai berikut:

1. **FROM Clause**: Kueri dimulai dengan mengambil data dari tabel orders dan

order\_details . Ini dilakukan dengan menggabungkan tabel-tabel ini menggunakan klausa JOIN berdasarkan kunci hubung yang sesuai. Misalnya, JOIN order\_details od ON o.order\_id = od.order\_id menggabungkan tabel orders dengan

order\_details berdasarkan kolom order\_id .

1. **SELECT Clause**: Kueri ini memilih beberapa kolom untuk ditampilkan dalam hasil kueri:

o.order\_id : Kolom ini menunjukkan ID pemesanan.

COUNT(od.product\_id) AS total\_products\_ordered : Menghitung jumlah produk yang dipesan dalam setiap pemesanan dengan menghitung jumlah entri dalam kolom product\_id dari tabel order\_details .

SUM(od.quantity) AS total\_products\_sold : Menghitung jumlah produk yang terjual dalam setiap pemesanan dengan menjumlahkan nilai dalam kolom

quantity dari tabel order\_details .

SUM(od.quantity \* od.unit\_price) AS total\_sales : Menghitung total

penjualan dalam setiap pemesanan dengan mengalikan jumlah produk yang terjual ( od.quantity ) dengan harga satuan produk ( od.unit\_price ) dari tabel

order\_details dan menjumlahkannya.

1. **GROUP BY Clause**: Karena kita menggunakan fungsi agregat ( COUNT dan SUM ) dalam klausa SELECT, kita perlu mengelompokkan data dengan kolom yang tidak diagregasi.

Kita mengelompokkan data berdasarkan o.order\_id (ID pemesanan).

1. **ORDER BY Clause**: Hasil kueri akan diurutkan berdasarkan o.order\_id (ID pemesanan).

Jadi, hasil kueri ini akan memberikan total penjualan, banyaknya produk, dan jumlah produk terjual untuk setiap pemesanan, diurutkan berdasarkan ID pemesanan.