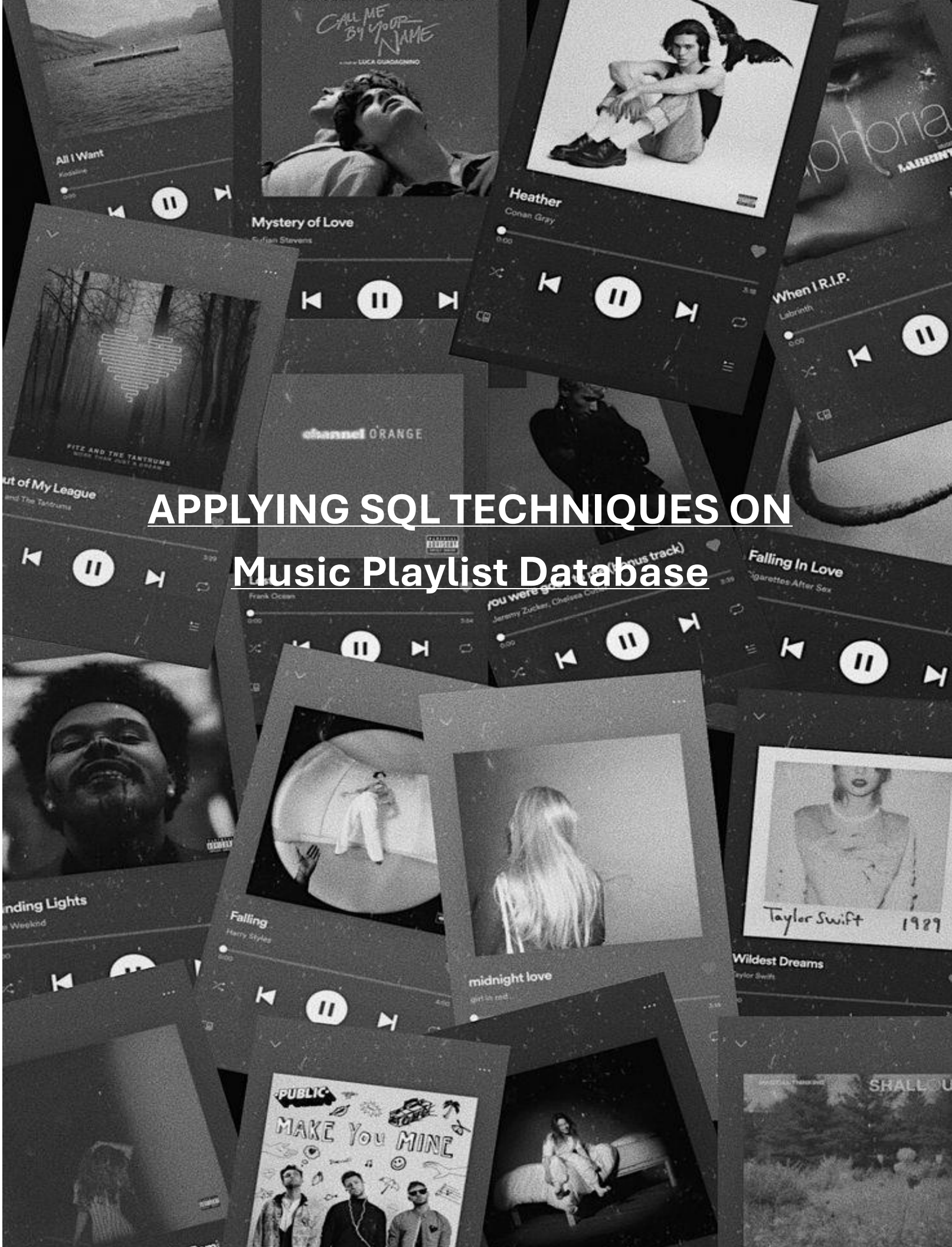
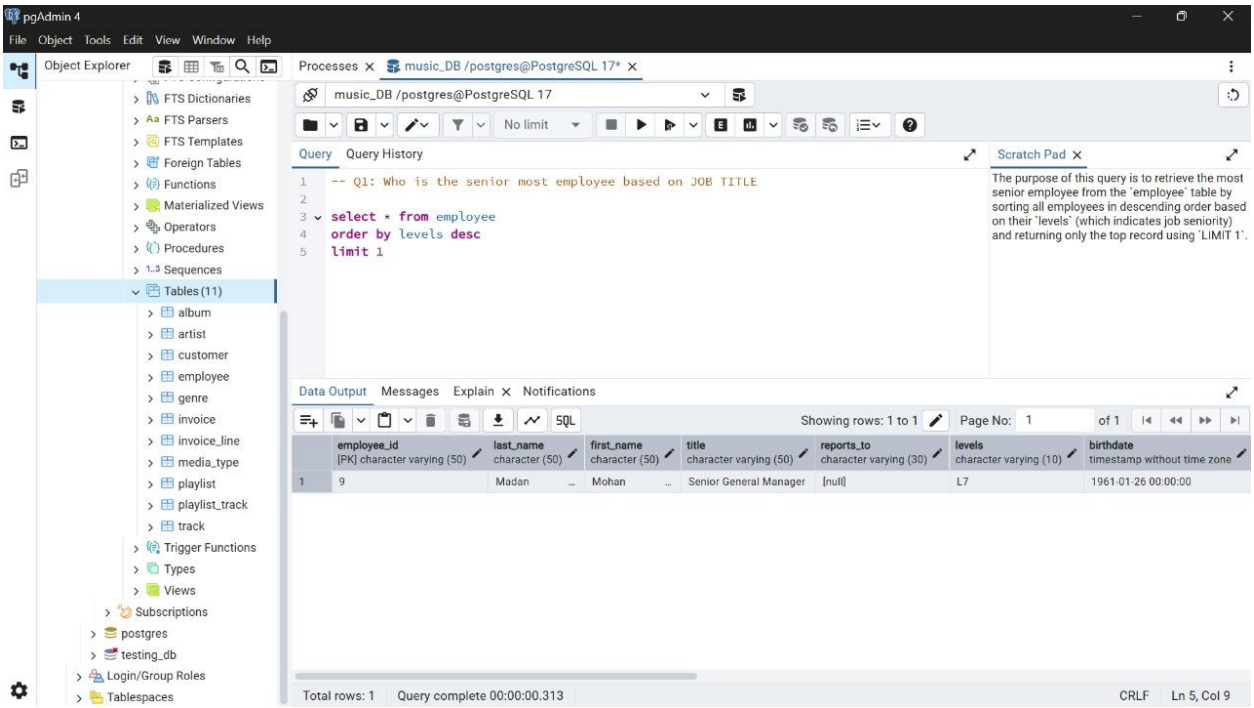
APPLYING SQL TECHNIQUES ON Music Playlist Database



Questions along with their Output, Syntax and reasoning...

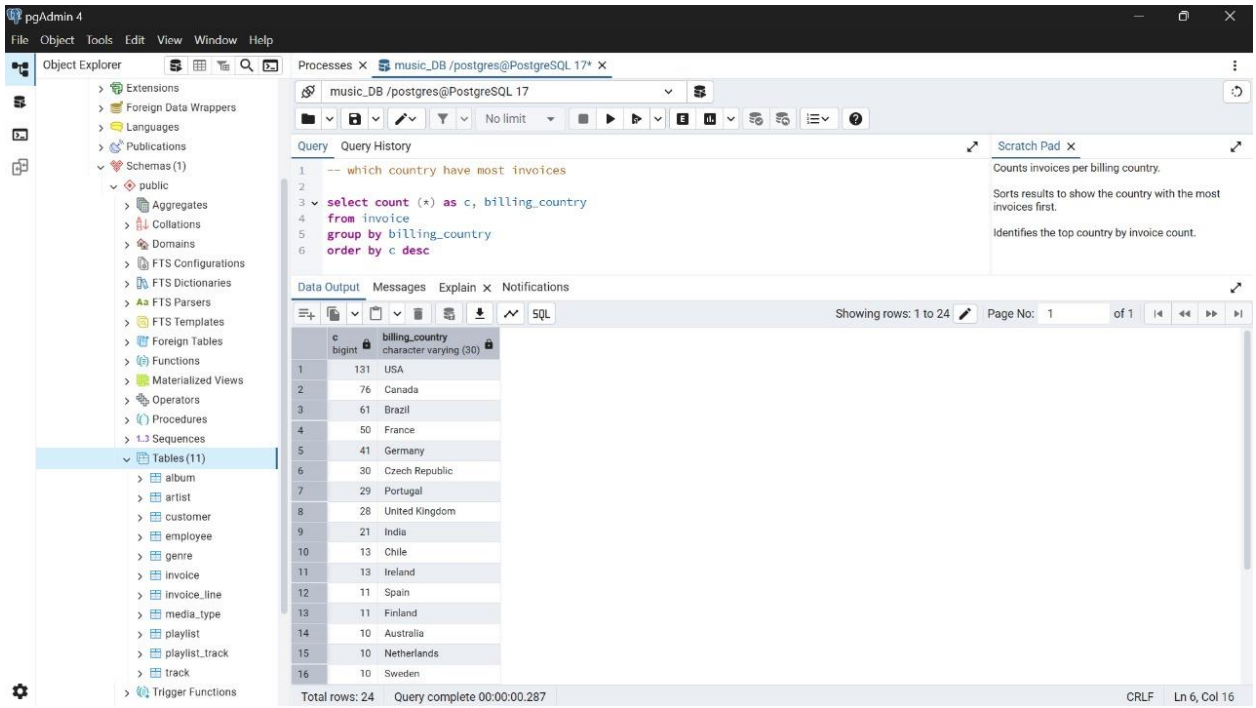
1. Who is the senior most employee based on job title?



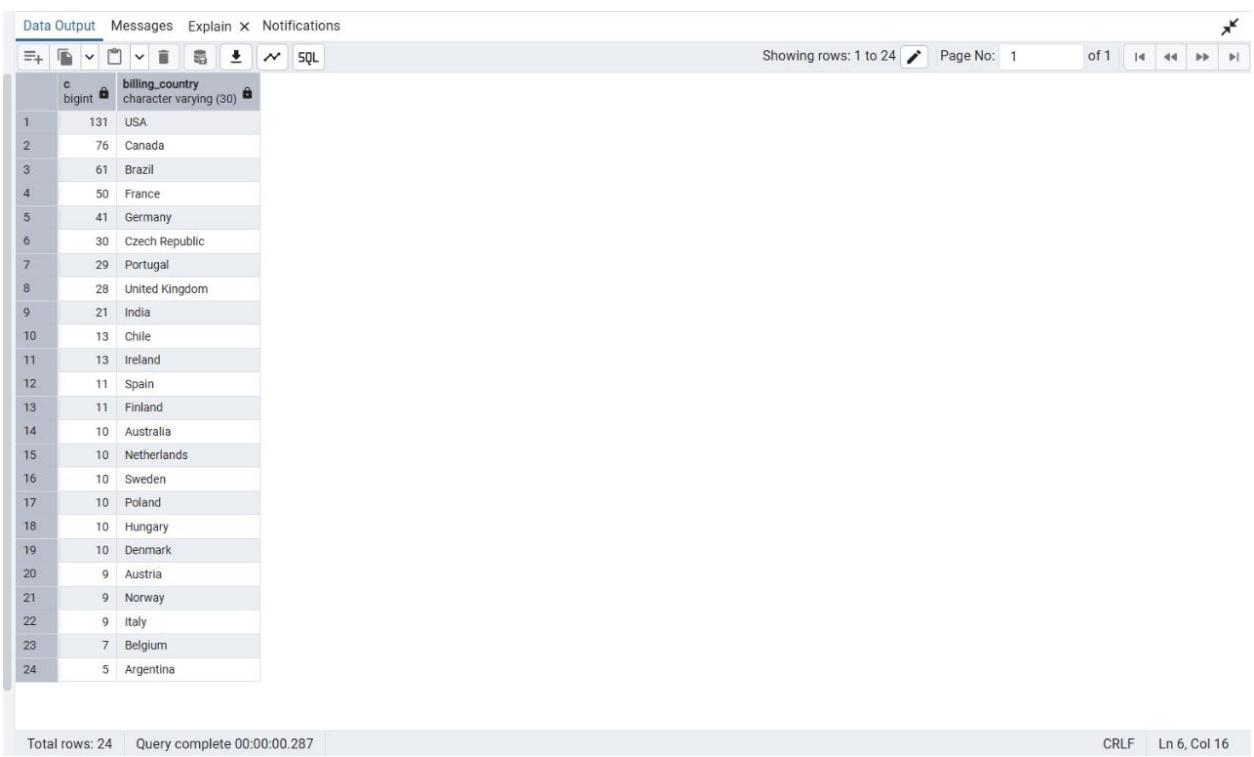
Explanation

- **SELECT * FROM employee**
  → Fetches all columns from the employee table.
- **ORDER BY levels DESC**
  → Sorts the records in **descending order** of the levels column.
  → Assumes that a **higher level (e.g., L7)** indicates a more senior position.
- **LIMIT 1**
  → Restricts the result to **only the top 1 record** after sorting.
  → This ensures only the **most senior employee** is shown.

2. Which countries have the most invoices?

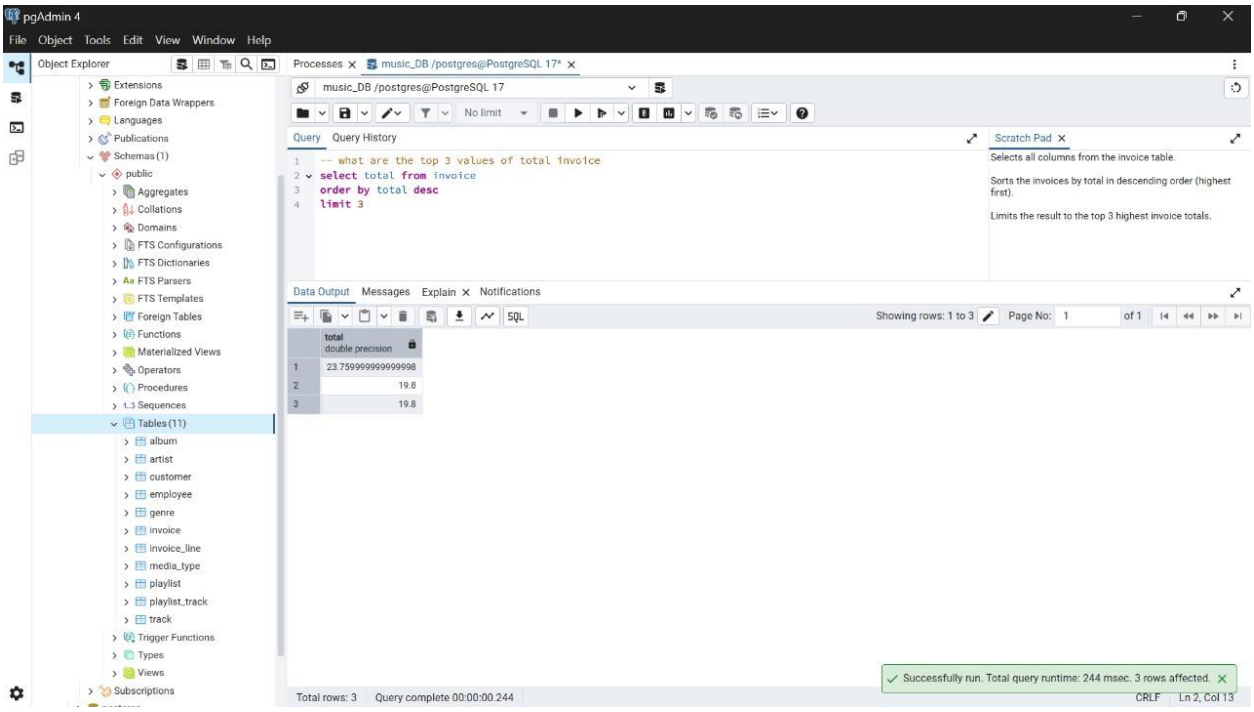


Output



	c bigint	billing_country character varying (30)
1	131	USA
2	76	Canada
3	61	Brazil
4	50	France
5	41	Germany
6	30	Czech Republic
7	29	Portugal
8	28	United Kingdom
9	21	India
10	13	Chile
11	13	Ireland
12	11	Spain
13	11	Finland
14	10	Australia
15	10	Netherlands
16	10	Sweden
17	10	Poland
18	10	Hungary
19	10	Denmark
20	9	Austria
21	9	Norway
22	9	Italy
23	7	Belgium
24	5	Argentina

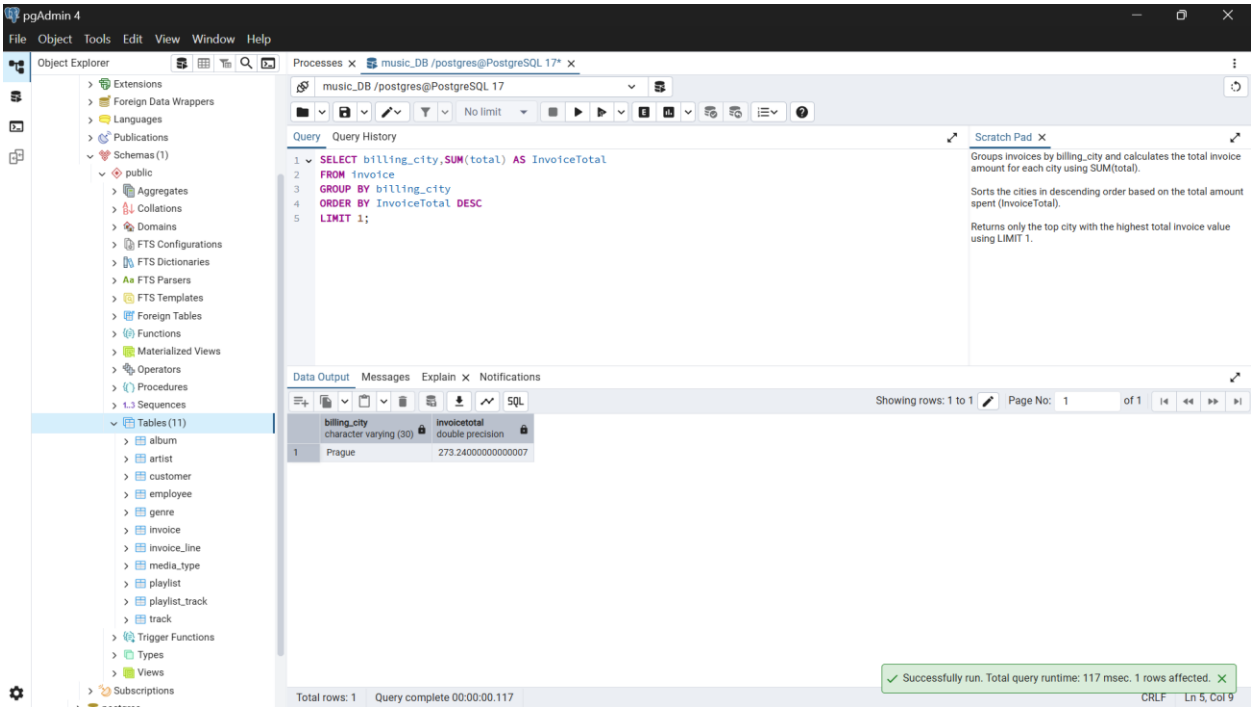
3. What are the top 3 values of total invoice?



Explanation

- **SELECT total FROM invoice**
  → Retrieves the total amount from all invoices.
- **ORDER BY total DESC**
  → Sorts the totals in **descending order** (highest to lowest).
- **LIMIT 3**
  → Returns only the **top 3 highest invoice totals**.

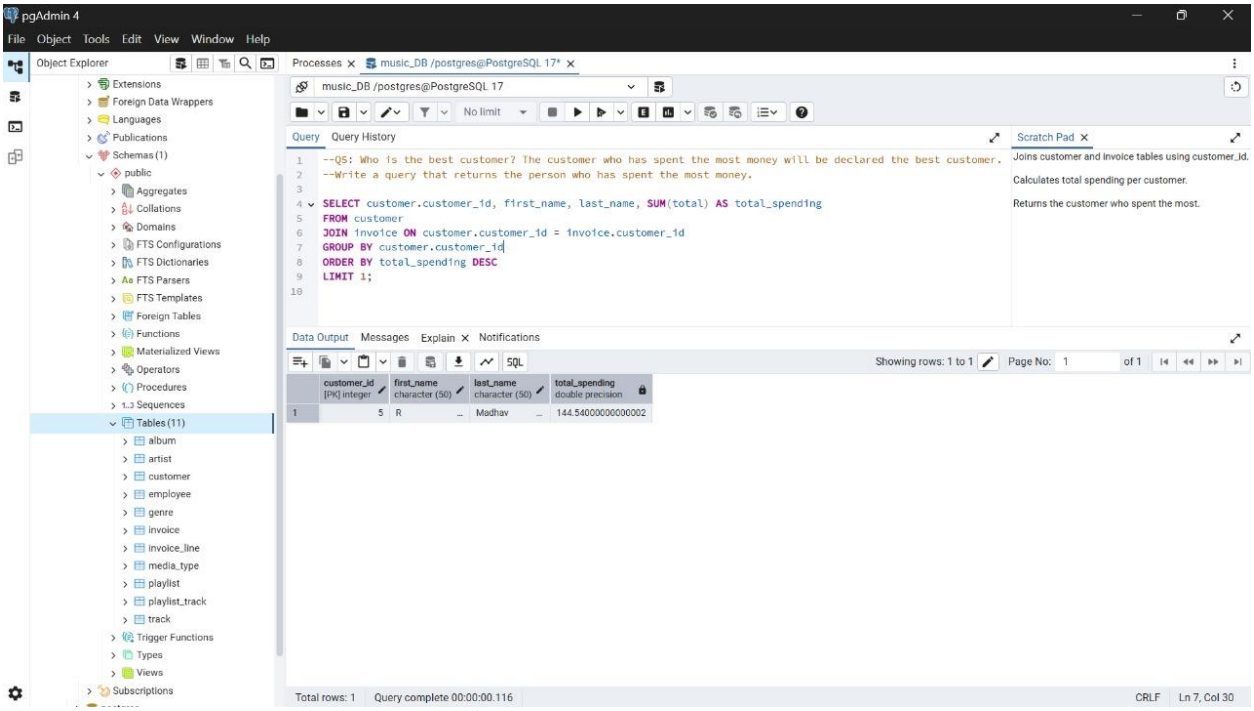
4. **Which city has the best customers? We would like to throw a promotional Music Festival in the city we made the most money. Write a query that returns one city that has the highest sum of invoice totals. Return both the city name & sum of all invoice totals.**



Explanation

- **Groups invoices by billing_city** and calculates the **total invoice amount** for each city using SUM(total).
- **Sorts the cities** in descending order based on the total amount spent (InvoiceTotal).
- **Returns only the top city** with the highest total invoice value using LIMIT 1.

5. **Who is the best customer? The customer who has spent the most money will be declared the best customer. Write a query that returns the person who has spent the most money.**

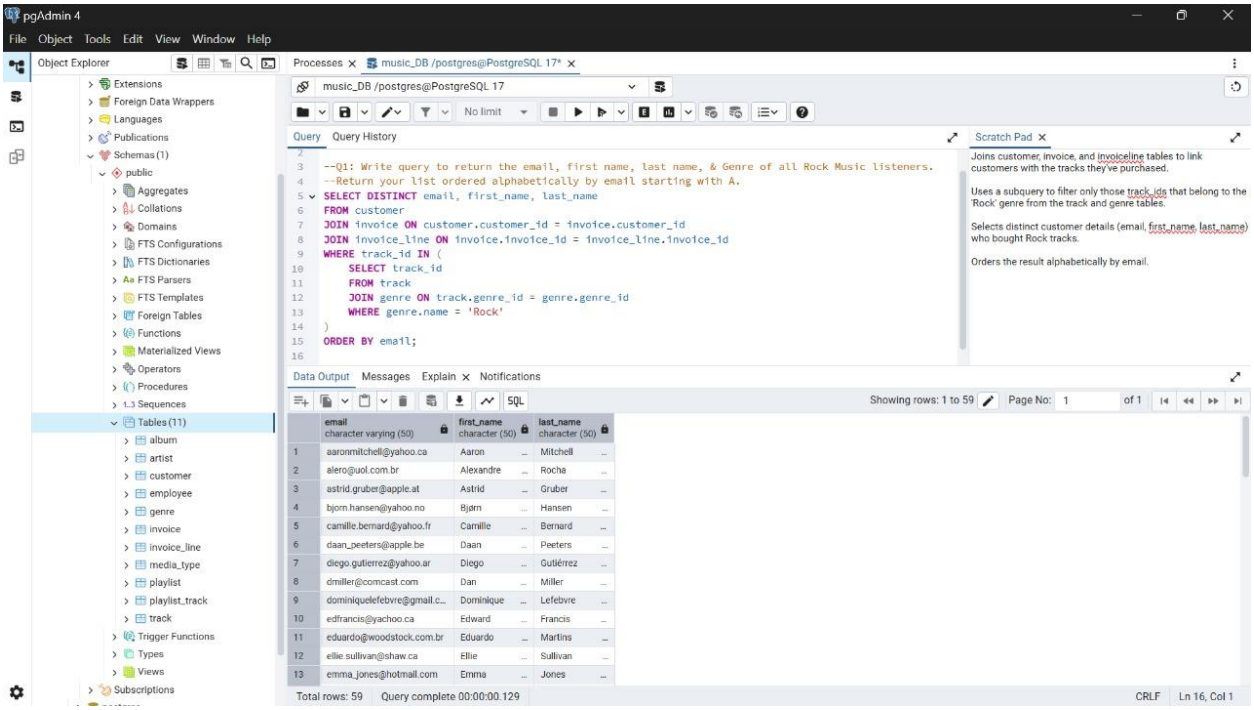


Explanation

- **SELECT customer.customer_id, first_name, last_name, SUM(total) AS total_spending**
  → Retrieves the customer's ID, first name, last name, and their **total spending** by summing up all invoice amounts.
- **FROM customer**
  → Data is being selected from the **customer** table.
- **JOIN invoice ON customer.customer_id = invoice.customer_id**
  → Performs an **inner join** between customer and invoice tables using the customer_id column.
  → This links each customer to their respective invoices.
- **GROUP BY customer.customer_id**
  → Groups all invoice records by customer to calculate **total spending per customer**.
- **ORDER BY total_spending DESC**
  → Sorts the results in **descending order** so the highest spender comes first.

- **LIMIT 1**
  → Returns **only the top customer** who has spent the most.

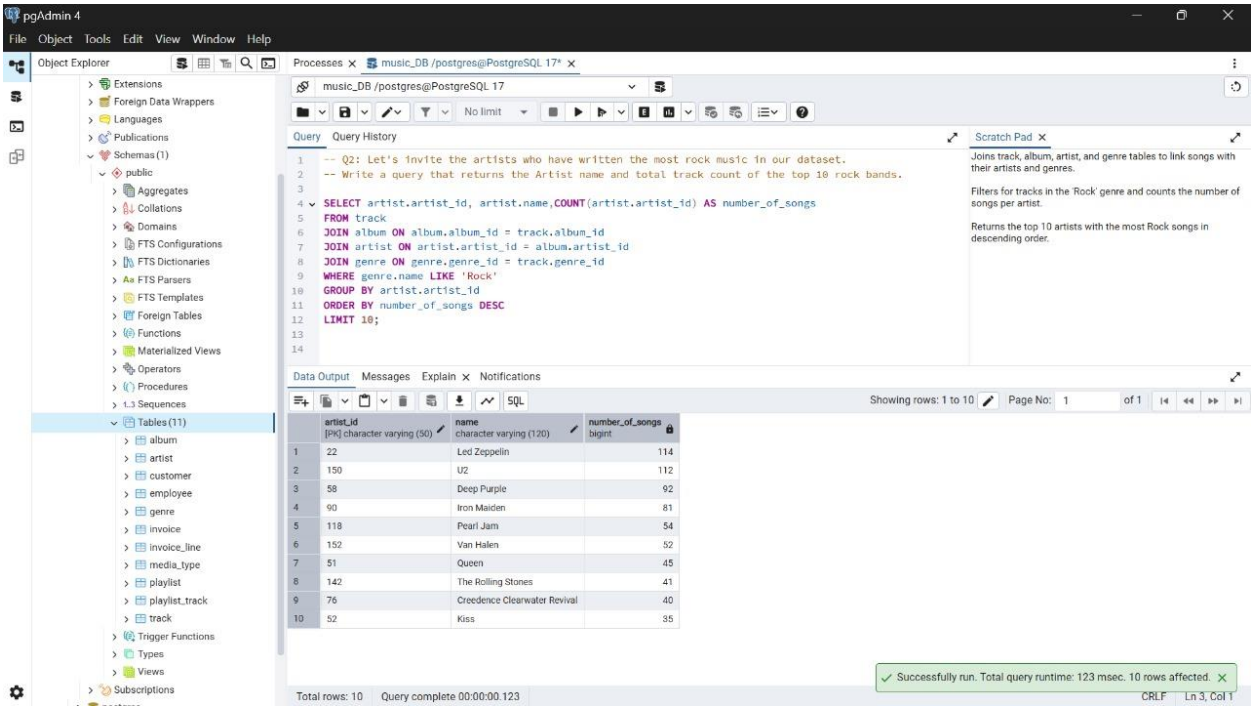
6. Write a query to return the email, first name, last name, & genre of all Rock music listeners. Return your list ordered alphabetically by email starting with A.



Explanation

- **Joins** the track, album, artist, and genre tables to connect each track to its respective artist and genre.
- **Filters for Rock genre tracks** and **counts the number of Rock songs** per artist using COUNT().
- **Returns the top 10 artists** who have the most Rock songs, sorted in **descending order** by song count.

7. Let's invite the artists who have written the most Rock music in our dataset. Write a query that returns the artist name and total track count of the top 10 Rock bands.



- **SELECT artist.artist_id, artist.name, COUNT(artist.artist_id) AS number_of_songs**
  → Retrieves each **artist's ID**, **name**, and the **total number of Rock songs** associated with them.
- **FROM track**
  → Starts from the track table which contains information about each song.
- **JOIN album ON album.album_id = track.album_id**
  → Links each track to its respective album using album_id.
- **JOIN artist ON artist.artist_id = album.artist_id**
  → Links each album to its artist using artist_id.
- **JOIN genre ON genre.genre_id = track.genre_id**
  → Links each track to its genre.
- **WHERE genre.name LIKE 'Rock'**
  → Filters the tracks to include **only those with the genre 'Rock'**.
- **GROUP BY artist.artist_id**
  → Groups the data **by artist**, so that the number of Rock songs per artist can be counted.

- **ORDER BY number_of_songs DESC**
  → Sorts the result by **number of Rock songs**, highest first.
- **LIMIT 10**
  → Returns only the **top 10 artists** with the most Rock songs.