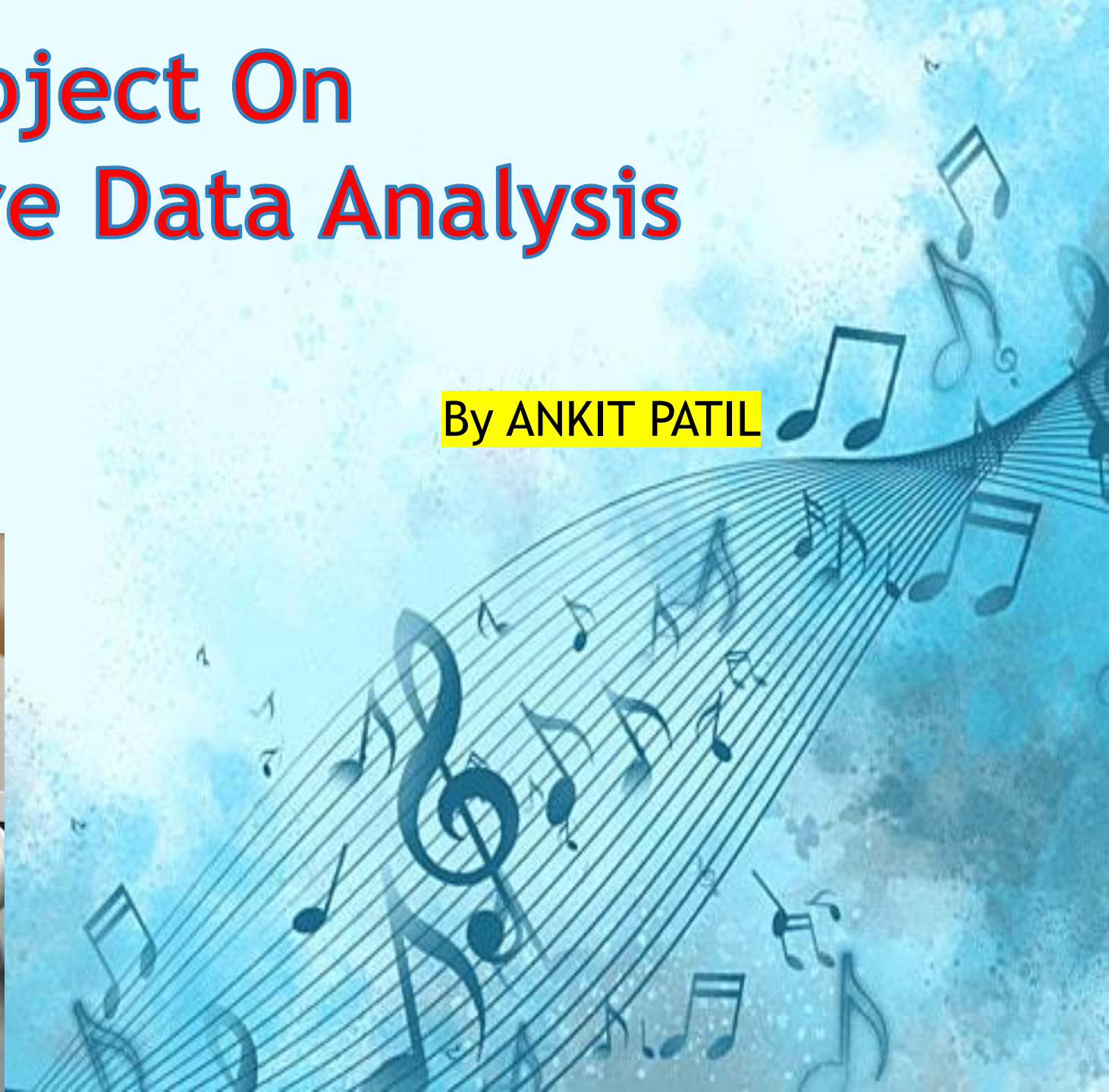


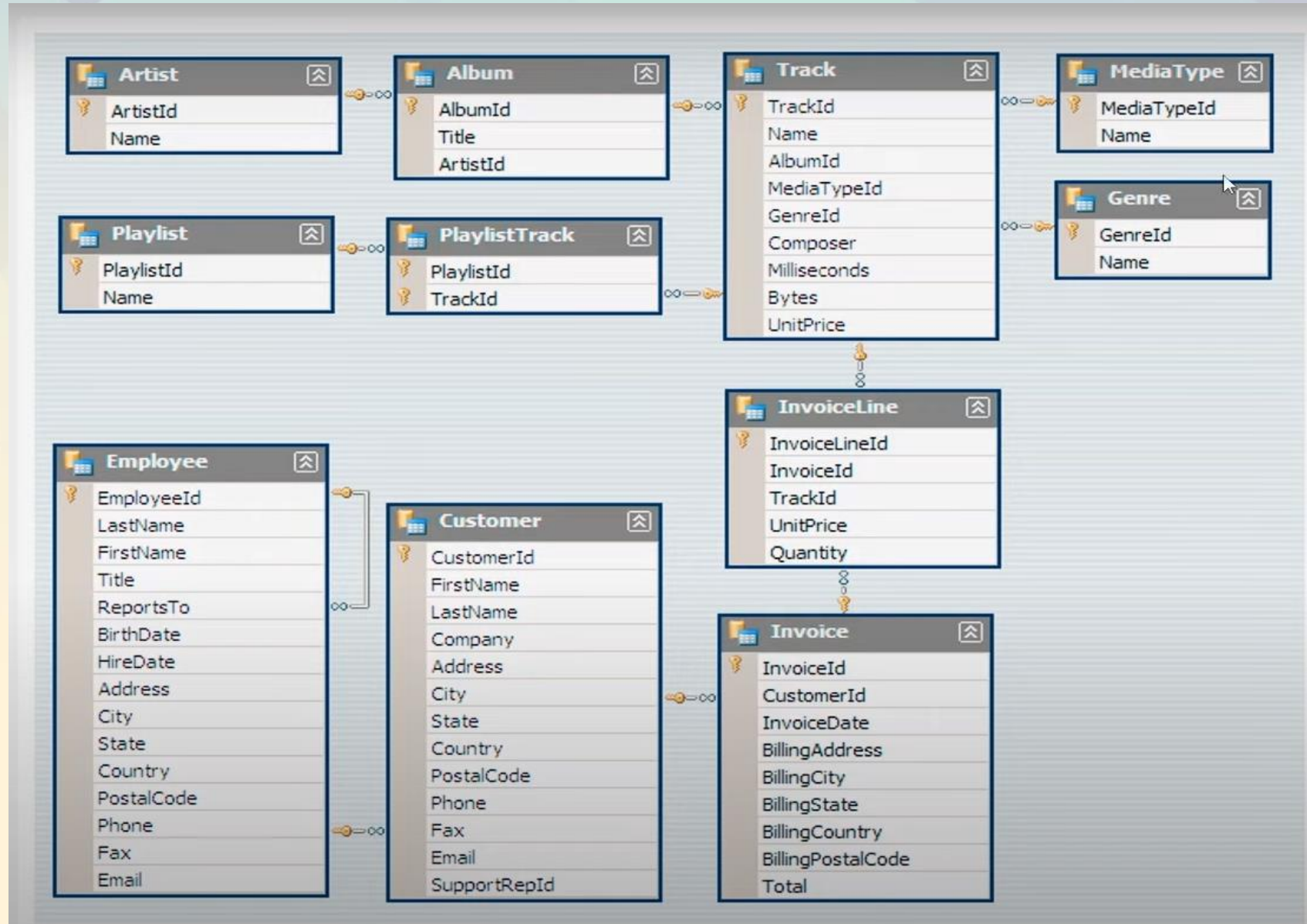
Project On Music Store Data Analysis

By ANKIT PATIL



1) MUSIC PLAYLIST DATABASE SCHEMA

SCHEMA OF THE DATABASE -

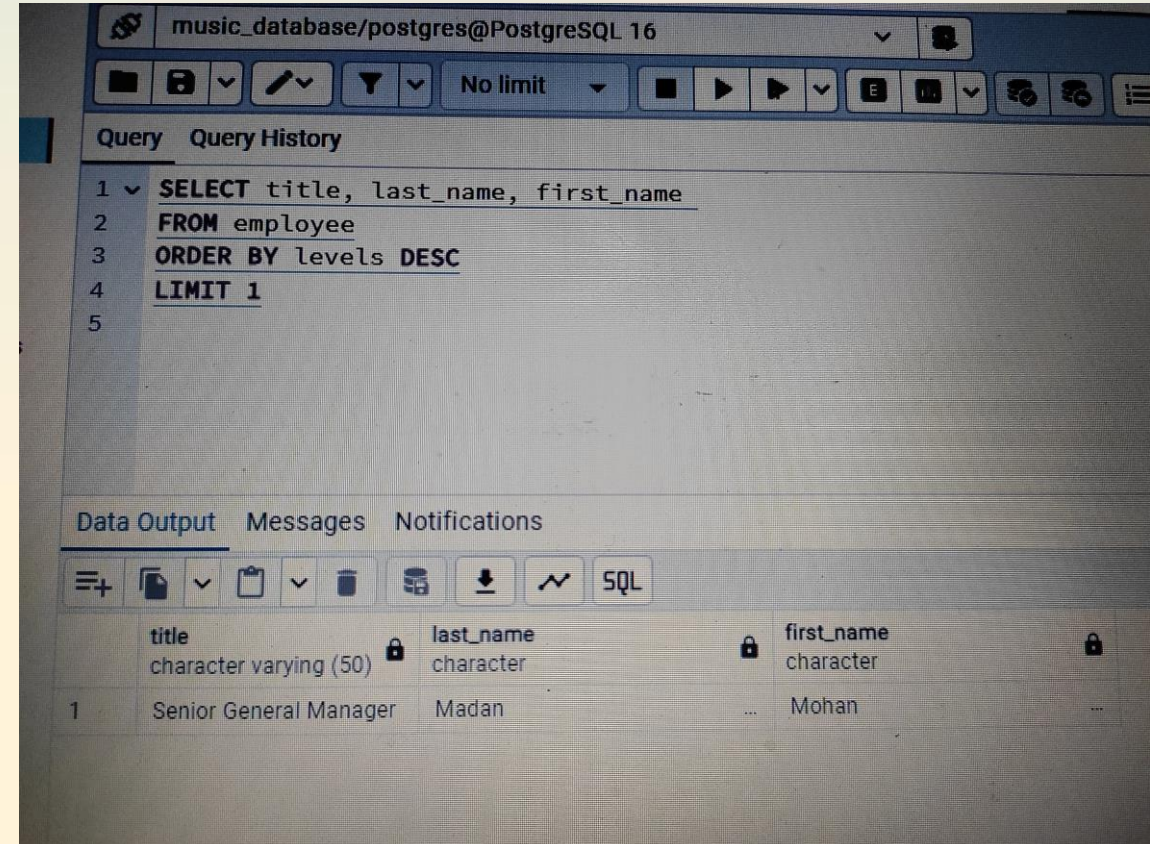


Here We Solve The Questions On Music Dataset Using SQL

Q1: Who is the senior most employee based on job title?

SQL Query

```
SELECT title, last_name, first_name  
FROM employee  
ORDER BY levels DESC  
LIMIT 1
```



The screenshot shows a PostgreSQL SQL client interface. The title bar indicates the connection is to 'music_database/postgres@PostgreSQL 16'. The 'Query' tab is active, displaying the following SQL query:

```
1 SELECT title, last_name, first_name  
2 FROM employee  
3 ORDER BY levels DESC  
4 LIMIT 1  
5
```

Below the query editor, the 'Data Output' tab is active, showing the result of the query. The result is a single row with the following data:

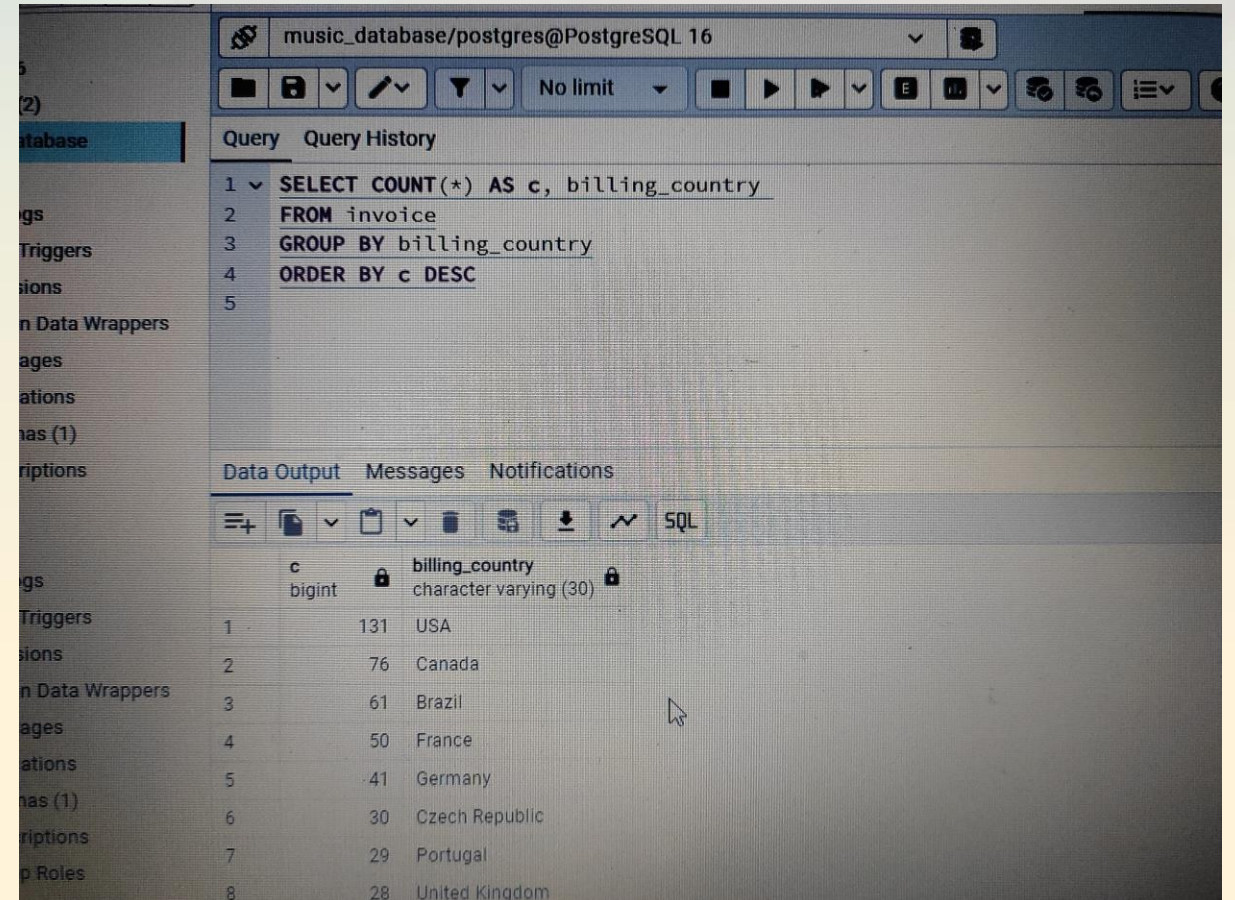
	title	last_name	first_name
1	Senior General Manager	Madan	Mohan

The interface also includes a 'Query History' tab, a 'Messages' tab, and a 'Notifications' tab. The 'Data Output' tab has a toolbar with icons for various actions like copy, paste, and download.

Q2: Which countries have the most Invoices?

SQL Query

```
SELECT COUNT(*) AS c, billing_country
FROM invoice
GROUP BY billing_country
ORDER BY c DESC
```



The screenshot shows a PostgreSQL database interface with the query editor and results pane. The query executed is:

```
1 SELECT COUNT(*) AS c, billing_country
2 FROM invoice
3 GROUP BY billing_country
4 ORDER BY c DESC
5
```

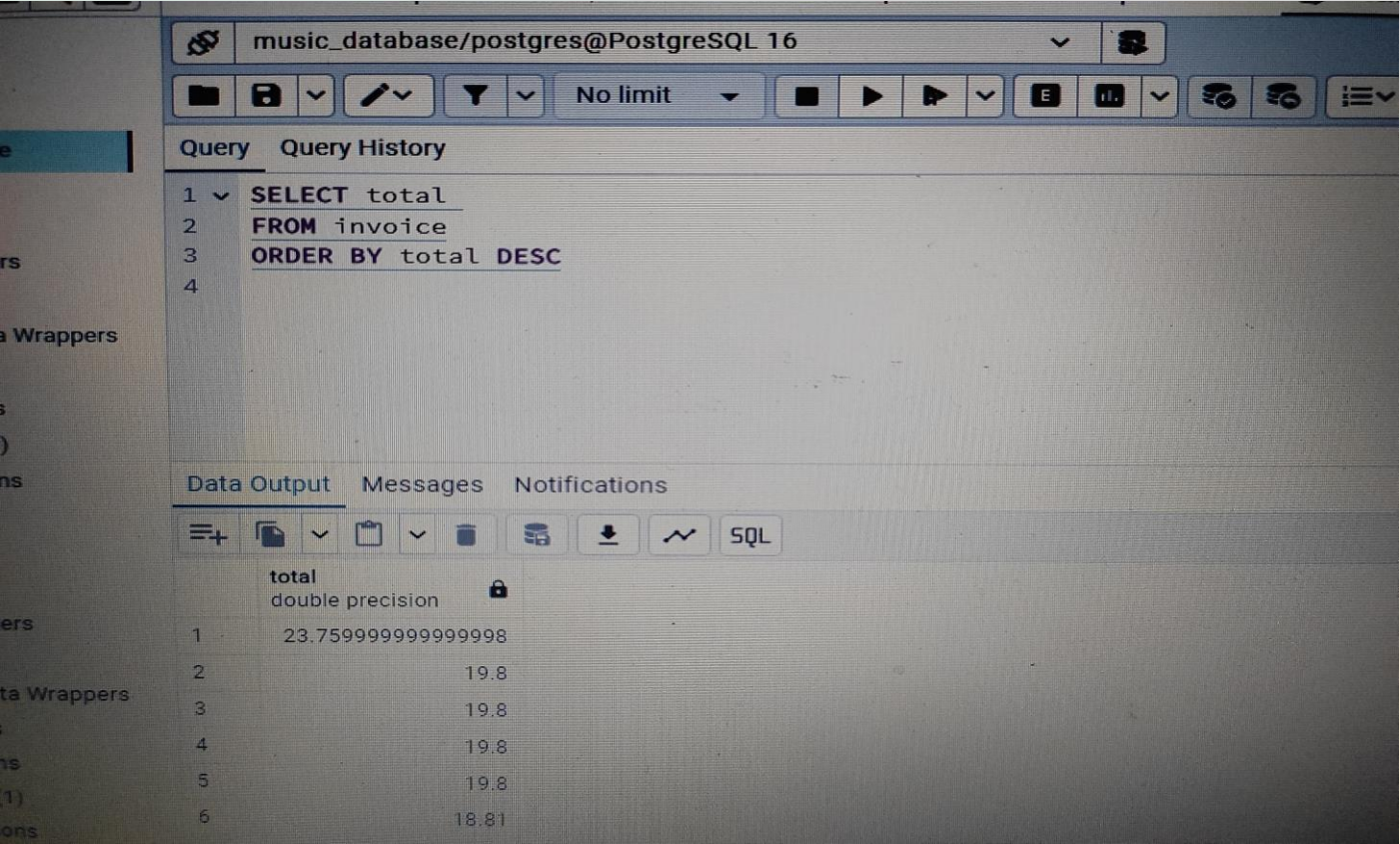
The results pane displays the following data:

	c	billing_country
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

Q3: What are top 3 values of total invoice?

SQL Query

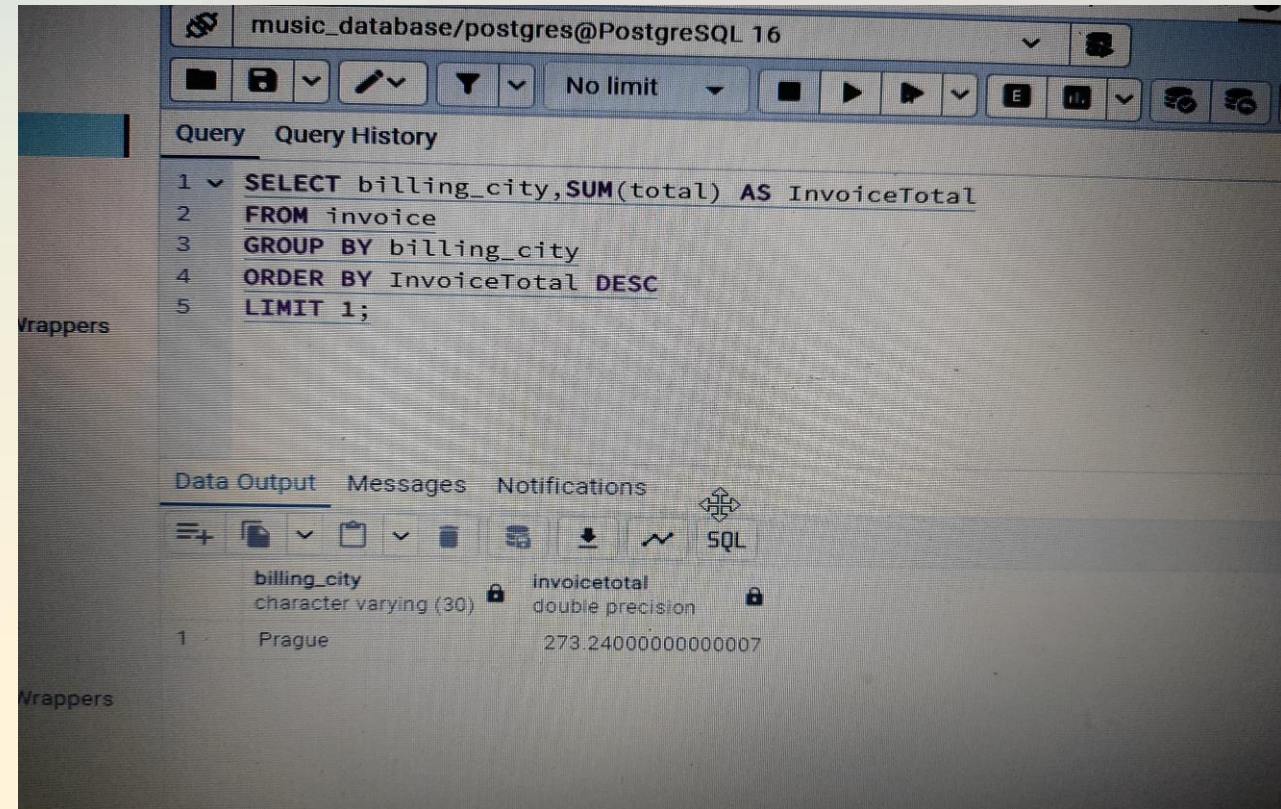
```
SELECT total
FROM invoice
ORDER BY total DESC
```



Q4: 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

SQL Query

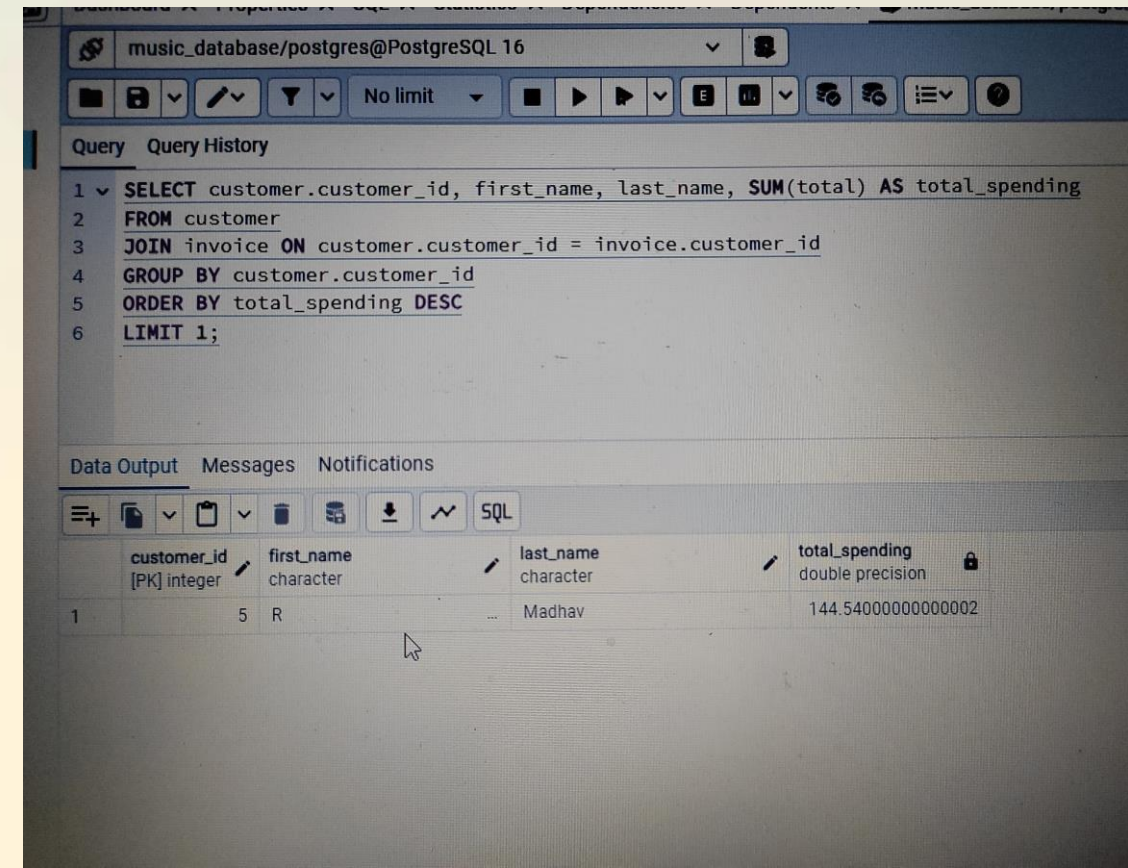
```
SELECT billing_city,SUM(total) AS InvoiceTotal
FROM invoice
GROUP BY billing_city
ORDER BY Invoice Total DES
CLIMIT 1;
```



Q5: 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.

SQL Query

```
SELECT customer.customer_id, first_name,  
last_name, SUM(total) AS total_spending  
FROM customer  
JOIN invoice ON customer.customer_id = invoice.customer_id  
GROUP BY customer.customer_id  
ORDER BY total_spending DESC  
LIMIT 1;
```



The screenshot shows a PostgreSQL query editor interface. The top bar indicates the database is 'music_database/postgres@PostgreSQL 16'. The query editor displays the following SQL query:

```
1 SELECT customer.customer_id, first_name, last_name, SUM(total) AS total_spending  
2 FROM customer  
3 JOIN invoice ON customer.customer_id = invoice.customer_id  
4 GROUP BY customer.customer_id  
5 ORDER BY total_spending DESC  
6 LIMIT 1;
```

Below the query editor, the 'Data Output' tab is active, showing the results of the query. The results are displayed in a table with the following columns: customer_id, first_name, last_name, and total_spending. The first row shows the results for the customer with the highest total spending.

	customer_id [PK] integer	first_name character	last_name character	total_spending double precision
1	5	R	Madhav	144.54000000000002

Q6: Write 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.

SQL Query

```
SELECT DISTINCT email,first_name,
last_name
FROM customer
JOIN invoice ON customer.customer_id
= invoice.customer_id
JOIN invoiceline ON invoice.invoice_id =
invoiceline.invoice_id
WHERE track_id IN(SELECT track_id
FROM track
JOIN genre ON track.genre_id =
genre.genre_id
WHERE genre.name LIKE 'Rock')
ORDER BY email;
```

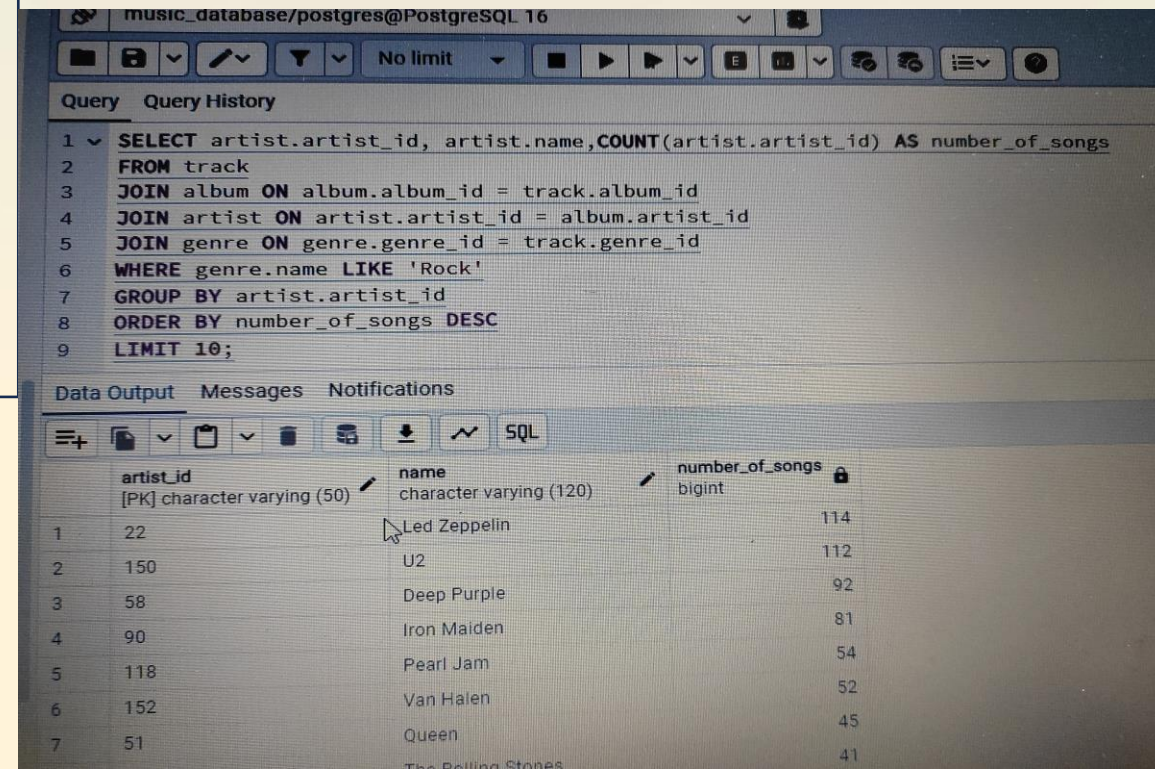
The screenshot shows a database management interface. On the left is a sidebar with a tree view of database objects: FTS Configurations, FTS Dictionaries, FTS Parsers, FTS Templates, Foreign Tables, Functions, Materialized Views, Operators, Procedures, Sequences, Tables (11), Trigger Functions, and Types. The 'Tables (11)' folder is expanded, showing a list of tables: album, artist, customer, employee, genre, invoice, invoice_line, media_type, playlist, playlist_track, and track. The 'customer' table is selected. The main area displays the SQL query in a text editor, with a toolbar above it containing icons for file operations, query execution, and other functions. Below the query editor are tabs for 'Query', 'Query History', 'Data output', 'Messages', and 'Notifications'. The 'Data output' tab is active, showing a table with the query results. The table has four columns: 'email' (character varying (50)), 'first_name' (character (50)), 'last_name' (character (50)), and an unlabeled column. The results are ordered by email, starting with 'aaronmitchell@yahoo...'. The first row is highlighted in blue.

	email character varying (50)	first_name character (50)	last_name character (50)
1	aaronmitchell@yahoo...	Aaron	Mitchell
2	alero@uol.com.br	Alexandre	Rocha
3	astrid.gruber@apple.at	Astrid	Gruber
4	bjorn.hansen@yahoo....	Bjorn	Hansen
5	camille.bernard@yah...	Camille	Bernard
6	dean.nastore@apple...	Dean	Nastore

Q7: 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.

SQL Query

```
SELECT artist.artist_id, artist.name, COUNT(artist.artist_id) AS
number_of_songs
FROM track
JOIN album ON album.album_id = track.album_id
JOIN artist ON artist.artist_id = album.artist_id
JOIN genre ON genre.genre_id = track.genre_id
WHERE genre.name LIKE 'Rock'
GROUP BY artist.artist_id
ORDER BY number_of_songs DESC
LIMIT 10;
```



The screenshot shows a PostgreSQL database interface with the following query and results:

```
1 SELECT artist.artist_id, artist.name, COUNT(artist.artist_id) AS number_of_songs
2 FROM track
3 JOIN album ON album.album_id = track.album_id
4 JOIN artist ON artist.artist_id = album.artist_id
5 JOIN genre ON genre.genre_id = track.genre_id
6 WHERE genre.name LIKE 'Rock'
7 GROUP BY artist.artist_id
8 ORDER BY number_of_songs DESC
9 LIMIT 10;
```

	artist_id [PK] character varying (50)	name character varying (120)	number_of_songs bigint
1	22	Led Zeppelin	114
2	150	U2	112
3	58	Deep Purple	92
4	90	Iron Maiden	81
5	118	Pearl Jam	54
6	152	Van Halen	52
7	51	Queen	45
		The Rolling Stones	41

Q8: Return all the track names that have a song length longer than the average song length. Return the Name and Milliseconds for each track. Order by the song length with the longest songs listed first.

SQL Query

```
SELECT name,milliseconds
FROM track
WHERE milliseconds > ( SELECT
AVG(milliseconds) AS avg_track_length FROM
track )
ORDER BY milliseconds DESC;
```

The screenshot shows a SQL IDE interface. On the left is a sidebar with a tree view of database objects including 'Configurations', 'Dictionaries', 'Parsers', 'Templates', 'Sign Tables', 'ctions', 'erialized Views', 'rators', 'cedures', 'uences', 'es (11)', 'lbum', 'rtist', 'ustomer', 'mployee', 'enre', 'voice', 'voice_line', 'edia_type', 'laylist', 'laylist_track', 'ack', 'ger Functions', and 'es'. The main area is divided into two panes. The top pane, titled 'Query', contains the following SQL code:

```
2 -- Return the Name and Milliseconds for each track. Order by the song L
3 -- listed first.
4
5 SELECT name,milliseconds
6 FROM track
7 WHERE milliseconds > (
8     SELECT AVG(milliseconds) AS avg_track_length
9     FROM track)
10 ORDER BY milliseconds DESC;
11
```

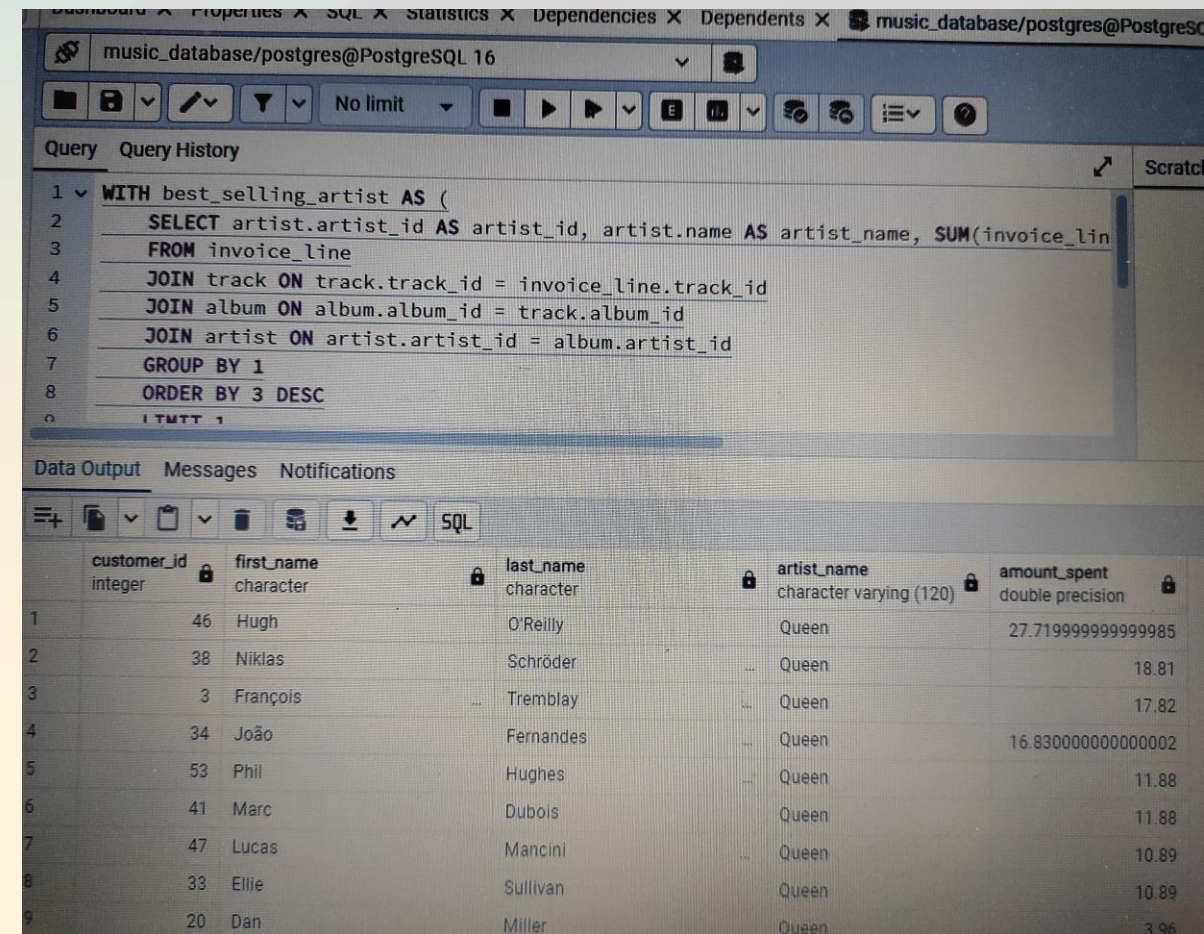
The bottom pane, titled 'Data output', shows the result of the query. It has a table with one column, 'avg_track_length', and one row with the value '393599.2121039109'. The status bar at the bottom indicates 'Total rows: 1 of 1' and 'Query complete 00:00:00.275'.

	avg_track_length
1	393599.2121039109

Q9: Find how much amount spent by each customer on artists? Write a query to return customer name, artist name and total spent

SQL Query

```
WITH best_selling_artist AS (SELECT artist.artist_id AS
artist_id, artist.name AS artist_name,
SUM(invoice_line.unit_price*invoice_line.quantity) AS
total_sales
FROM invoice_line
JOIN track ON track.track_id = invoice_line.track_id
JOIN album ON album.album_id = track.album_id
JOIN artist ON artist.artist_id = album.artist_id
GROUP BY 1
ORDER BY 3 DESC
LIMIT 1
)
```



The screenshot shows a PostgreSQL query editor with the following SQL query:

```
1 WITH best_selling_artist AS (
2   SELECT artist.artist_id AS artist_id, artist.name AS artist_name, SUM(invoice_line
3     FROM invoice_line
4     JOIN track ON track.track_id = invoice_line.track_id
5     JOIN album ON album.album_id = track.album_id
6     JOIN artist ON artist.artist_id = album.artist_id
7     GROUP BY 1
8     ORDER BY 3 DESC
9 )
10 LIMIT 1
```

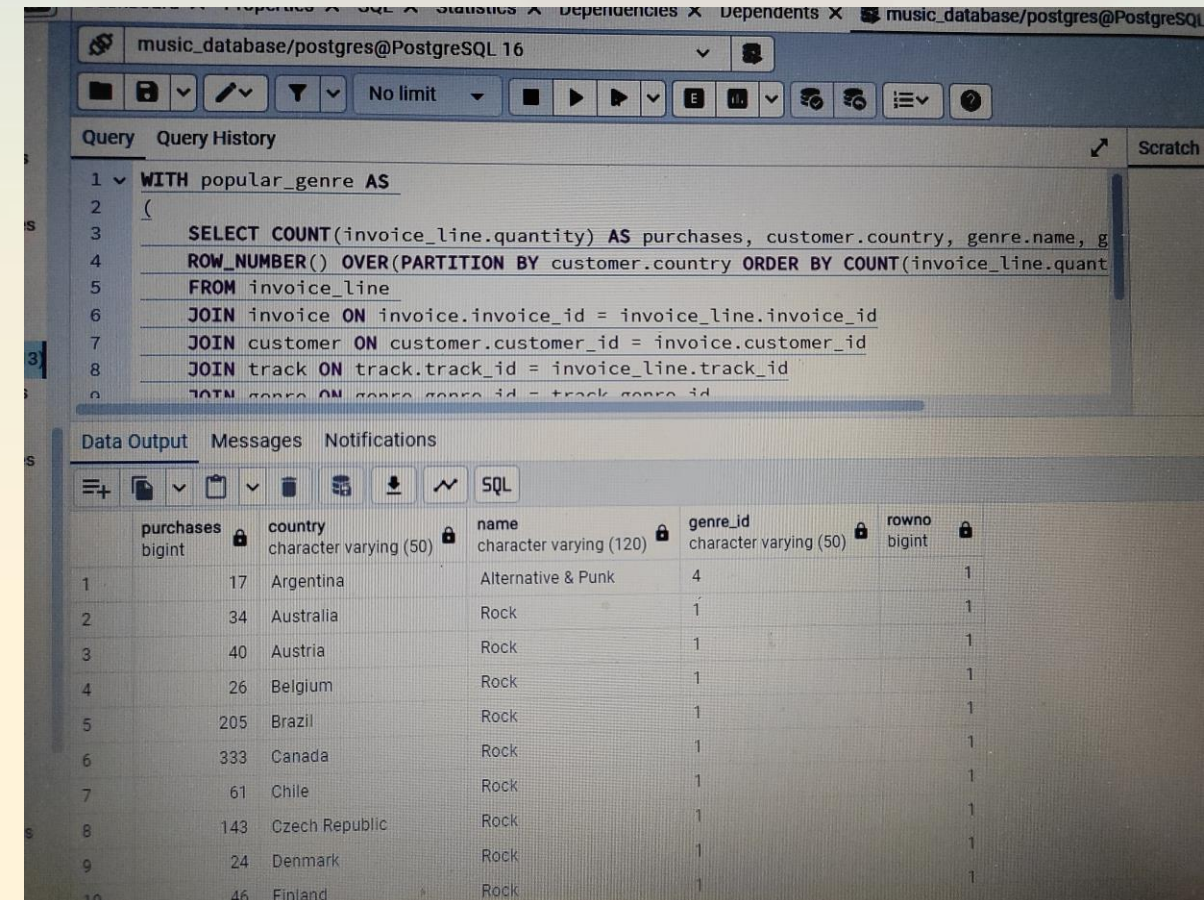
The results are displayed in a table with the following columns: customer_id, first_name, last_name, artist_name, and amount_spent.

customer_id	first_name	last_name	artist_name	amount_spent
46	Hugh	O'Reilly	Queen	27.719999999999985
38	Niklas	Schröder	Queen	18.81
3	François	Tremblay	Queen	17.82
34	João	Fernandes	Queen	16.830000000000002
53	Phil	Hughes	Queen	11.88
41	Marc	Dubois	Queen	11.88
47	Lucas	Mancini	Queen	10.89
33	Ellie	Sullivan	Queen	10.89
20	Dan	Miller	Queen	3.96

Q10: We want to find out the most popular music Genre for each country. We determine the most popular genre as the genre with the highest amount of purchases. Write a query that returns each country along with the top Genre. For countries where the maximum number of purchases is shared return all Genres.

SQL Query

```
WITH popular_genre AS
(
SELECT COUNT(invoice_line.quantity) AS purchases,
customer.country, genre.name, genre.genre_id,
ROW_NUMBER() OVER(PARTITION BY customer.country
ORDER BY COUNT(invoice_line.quantity) DESC) AS RowNo
FROM invoice_line
JOIN invoice ON invoice.invoice_id = invoice_line.invoice_id
JOIN customer ON customer.customer_id =
invoice.customer_id
JOIN track ON track.track_id = invoice_line.track_id
JOIN genre ON genre.genre_id = track.genre_id
GROUP BY 2,3,4
ORDER BY 2 ASC, 1 DESC)SELECT * FROM popular_genre
WHERE RowNo <= 1
```



The screenshot shows a PostgreSQL query editor with the following SQL query:

```
WITH popular_genre AS
(
SELECT COUNT(invoice_line.quantity) AS purchases, customer.country, genre.name, g
ROW_NUMBER() OVER(PARTITION BY customer.country ORDER BY COUNT(invoice_line.quant
FROM invoice_line
JOIN invoice ON invoice.invoice_id = invoice_line.invoice_id
JOIN customer ON customer.customer_id = invoice.customer_id
JOIN track ON track.track_id = invoice_line.track_id
JOIN genre ON genre.genre_id = track.genre_id
```

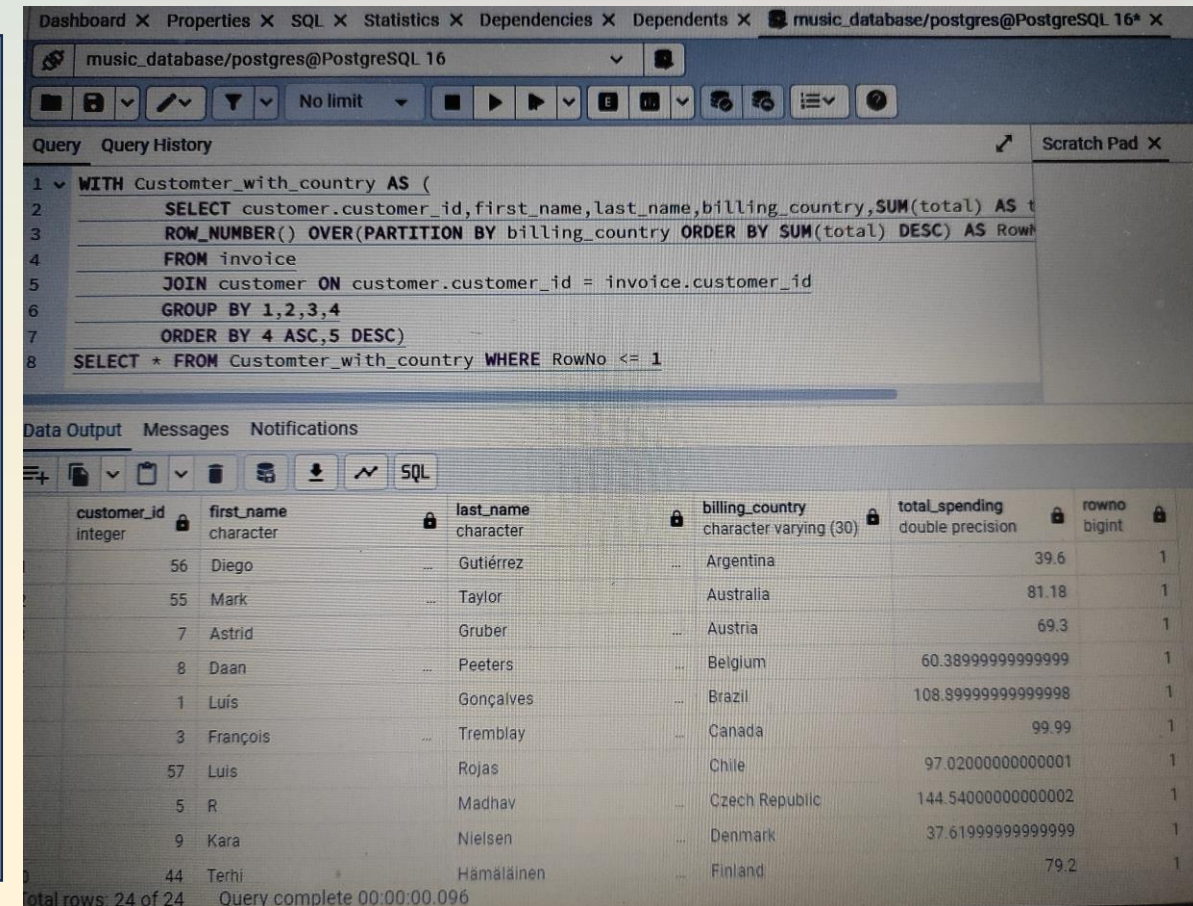
The results are displayed in a table with the following columns: purchases (bigint), country (character varying (50)), name (character varying (120)), genre_id (character varying (50)), and rowno (bigint). The table shows the top genre for each country, ordered by country name.

	purchases bigint	country character varying (50)	name character varying (120)	genre_id character varying (50)	rowno bigint
1	17	Argentina	Alternative & Punk	4	1
2	34	Australia	Rock	1	1
3	40	Austria	Rock	1	1
4	26	Belgium	Rock	1	1
5	205	Brazil	Rock	1	1
6	333	Canada	Rock	1	1
7	61	Chile	Rock	1	1
8	143	Czech Republic	Rock	1	1
9	24	Denmark	Rock	1	1
10	46	Finland	Rock	1	1

Q11: Write a query that determines the customer that has spent the most on music for each country. Write a query that returns the country along with the top customer and how much they spent. For countries where the top amount spent is shared, provide all customers who spent this amount.

SQL Query

```
WITH Customer_with_country AS (  
SELECT  
customer.customer_id, first_name, last_name, billing_country  
, SUM(total) AS total_spending, ROW_NUMBER()  
OVER(PARTITION BY billing_country  
ORDER BY SUM(total) DESC) AS RowNo FROM invoice  
JOIN customer ON customer.customer_id =  
invoice.customer_id  
GROUP BY 1,2,3,4  
ORDER BY 4 ASC, 5 DESC)  
SELECT * FROM Customer_with_country  
WHERE RowNo <= 1
```



The screenshot shows a PostgreSQL SQL client interface with the following components:

- Query Editor:** Displays the SQL query used to generate the results.
- Data Output:** A table showing the results of the query, sorted by total spending in descending order within each country.

Query:

```
1 WITH Customer_with_country AS (  
2     SELECT customer.customer_id, first_name, last_name, billing_country, SUM(total) AS t  
3     ROW_NUMBER() OVER(PARTITION BY billing_country ORDER BY SUM(total) DESC) AS RowNo  
4     FROM invoice  
5     JOIN customer ON customer.customer_id = invoice.customer_id  
6     GROUP BY 1,2,3,4  
7     ORDER BY 4 ASC, 5 DESC)  
8 SELECT * FROM Customer_with_country WHERE RowNo <= 1
```

Data Output:

customer_id	first_name	last_name	billing_country	total_spending	rowno
56	Diego	Gutiérrez	Argentina	39.6	1
55	Mark	Taylor	Australia	81.18	1
7	Astrid	Gruber	Austria	69.3	1
8	Daan	Peeters	Belgium	60.38999999999999	1
1	Luís	Gonçalves	Brazil	108.89999999999998	1
3	François	Tremblay	Canada	99.99	1
57	Luis	Rojas	Chile	97.02000000000001	1
5	R	Madhav	Czech Republic	144.54000000000002	1
9	Kara	Nielsen	Denmark	37.61999999999999	1
44	Terhi	Hämäläinen	Finland	79.2	1

Total rows: 24 of 24 Query complete 00:00:00.096

THANK YOU

