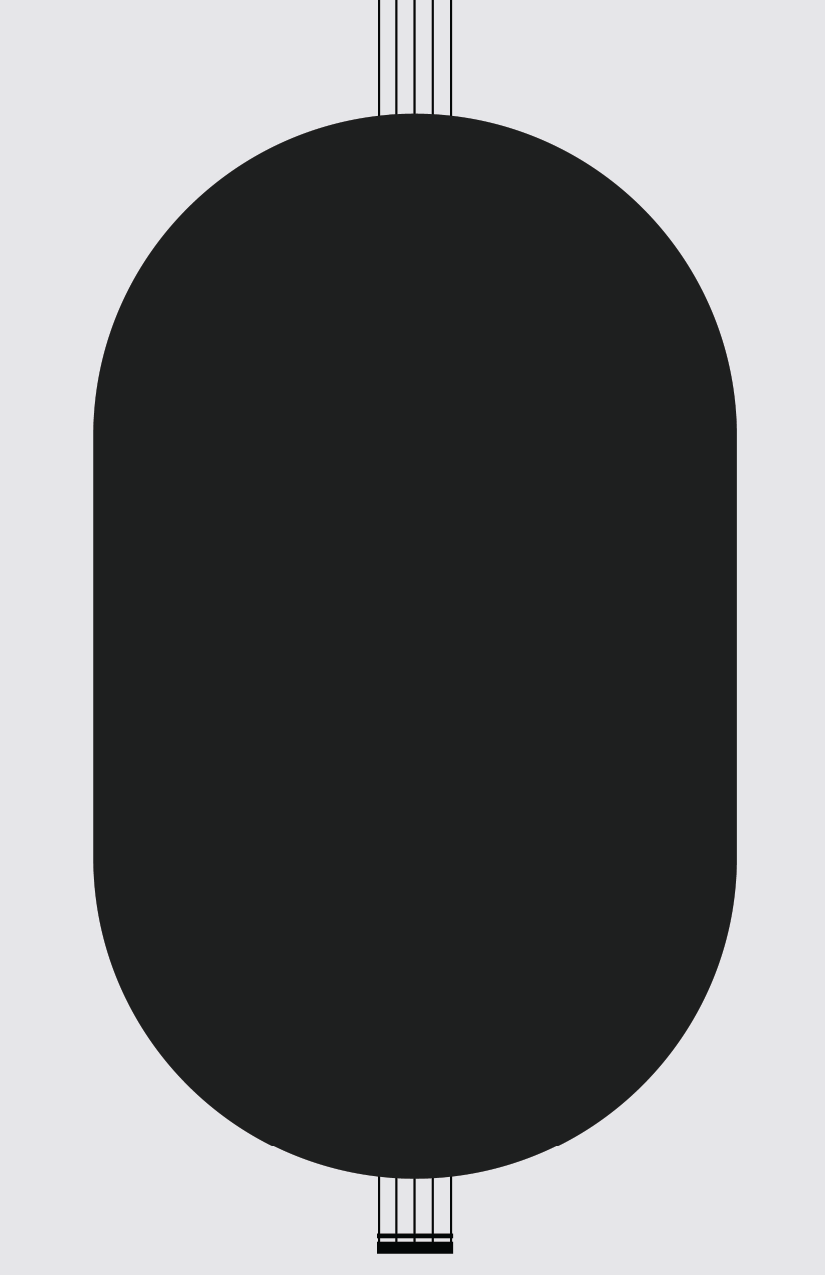
**** Jiosaavan

Music and Podcasts

****

**SQL project by –** **Bhushan Prabhakar Dhawas**

**Saavan Media limited in association with Jio**



## 

## **About This App -**

JioSaavn is India’s leading FREE music app, offering an extensive and exclusive library of 8 crore+ songs across 16 languages.

**Date founded:** **2007**

**Founders :**[**Rishi Malhotra**](https://www.google.com/search?sa=X&sca_esv=4812ae3c1a5b5c28&sca_upv=1&rlz=1C1RXQR_enIN1060IN1064&biw=1536&bih=826&sxsrf=ADLYWILXtimFuKvr9qVKXaDsgrLdR8FYXQ:1727590667769&q=Rishi+Malhotra&si=ACC90nwLLwns5sISZcdzuISy7t-NHozt8Cbt6G3WNQfC9ekAgMZcvnM-DHwc-np0pqrYwCOAWDHB7c9wVVKc1Kih4ELAqsuP7Err-LvfldPO6jSbeS2ihqcj0jiTX4LB6XZlZv4A8ZvualXRTPYH5xtBUMjnRODetVuFNp-AwX9JFQLVJX9i6vZcitCzPCCJfpQ6g-tNVFG-BXRcZYEVVk6gnN827Uv2Q9ewsGl_qFTD1mP5DuSt9XfpZvjoZrWZ0SgIsm-0bPz_Et3R70tcgt6560CDE2r8lg%3D%3D&ved=2ahUKEwjK_MbgwOeIAxVGcmwGHZ2GCAYQmxMoAHoECDYQAg) **,**[**Vinodh Bhat**](https://www.google.com/search?sa=X&sca_esv=4812ae3c1a5b5c28&sca_upv=1&rlz=1C1RXQR_enIN1060IN1064&biw=1536&bih=826&sxsrf=ADLYWILXtimFuKvr9qVKXaDsgrLdR8FYXQ:1727590667769&q=Vinodh+Bhat&si=ACC90nxYhNno81_TzuVO0e1EieRzQwXK2DrwuMdAl3IyVcpNQzdfp2709WhnSAMeFyzzfX0fDQXE7PY-QyFMxqQfKA-uq9uUDUKhy3RNmvHasR1diYMG7ZUPVeW3DKgGzHirGWD06Dc1Oc_ZhX9ZlPtJJorSQYgHhw10Nz3A24fsYo2iXQi_3R_z_oTmIJtmQjqHO2DRaJFJkrFrRfASDsLDvWlMKV6atzLlg8DXl_reMxiKBEsGyyfhKItUsX1drV1VX64UhfiKAg1NK9sZpMDmCYG-U-ZpDA%3D%3D&ved=2ahUKEwjK_MbgwOeIAxVGcmwGHZ2GCAYQmxMoAXoECDYQAw)**,**

[**Paramdeep Singh**](https://www.google.com/search?sa=X&sca_esv=4812ae3c1a5b5c28&sca_upv=1&rlz=1C1RXQR_enIN1060IN1064&biw=1536&bih=826&sxsrf=ADLYWILXtimFuKvr9qVKXaDsgrLdR8FYXQ:1727590667769&q=Paramdeep+Singh&si=ACC90nwLLwns5sISZcdzuISy7t-NHozt8Cbt6G3WNQfC9ekAgPtnK-wykP4m-tooCYB4ysaTAP4KxwEI6OCF1SaIbFbaeMRCrdwFM3wBhpe3FkaNEXB10DakwXwuTUcvuuhIS9iTW8Mdb5csDb4LgOmOLr4ecmNscf6mLZUkgXljy0_Y6djSsUiu4w2TJgYqESbgAbpF1-_fQHJQg8Yvn76ZVCrEzFaZ44uDxlH71gkItQ-bSmAT3DqCH1PQSFGDFS0E6X0UlQDmbIPNSy8Fe1y__NNu1OTp2Q%3D%3D&ved=2ahUKEwjK_MbgwOeIAxVGcmwGHZ2GCAYQmxMoAnoECDYQBA)

**Developer :  Bodvod Network (2006–2018) ;**

**Digital18 (2018–present);**

**Platform(s):** [**Android**](https://www.google.com/search?sa=X&sca_esv=4812ae3c1a5b5c28&sca_upv=1&rlz=1C1RXQR_enIN1060IN1064&biw=1536&bih=826&sxsrf=ADLYWILXtimFuKvr9qVKXaDsgrLdR8FYXQ:1727590667769&q=Android&stick=H4sIAAAAAAAAAONgVuLSz9U3MCqvKEkvX8TK7piXUpSfmQIAyAsAJhgAAAA&ved=2ahUKEwjK_MbgwOeIAxVGcmwGHZ2GCAYQmxMoAHoECDQQAg)**;**[**iOS**](https://www.google.com/search?sa=X&sca_esv=4812ae3c1a5b5c28&sca_upv=1&rlz=1C1RXQR_enIN1060IN1064&biw=1536&bih=826&sxsrf=ADLYWILXtimFuKvr9qVKXaDsgrLdR8FYXQ:1727590667769&q=iOS&stick=H4sIAAAAAAAAAONgVuLSz9U3MC5PyjE0WcTKnOkfDABeKS5iFAAAAA&ved=2ahUKEwjK_MbgwOeIAxVGcmwGHZ2GCAYQmxMoAXoECDQQAw)**;**[**watchOS**](https://www.google.com/search?sa=X&sca_esv=4812ae3c1a5b5c28&sca_upv=1&rlz=1C1RXQR_enIN1060IN1064&biw=1536&bih=826&sxsrf=ADLYWILXtimFuKvr9qVKXaDsgrLdR8FYXQ:1727590667769&q=watchOS&stick=H4sIAAAAAAAAAONgVuLWz9U3MDQszi0oq1zEyl6eWJKc4R8MAHyENH4ZAAAA&ved=2ahUKEwjK_MbgwOeIAxVGcmwGHZ2GCAYQmxMoAnoECDQQBA)**;**[**Alexa**](https://www.google.com/search?sa=X&sca_esv=4812ae3c1a5b5c28&sca_upv=1&rlz=1C1RXQR_enIN1060IN1064&biw=1536&bih=826&sxsrf=ADLYWILXtimFuKvr9qVKXaDsgrLdR8FYXQ:1727590667769&q=Alexa&stick=H4sIAAAAAAAAAONgVuLVT9c3NEw2NbbIKzFLWcTK6piTWpEIAOAdKTEZAAAA&ved=2ahUKEwjK_MbgwOeIAxVGcmwGHZ2GCAYQmxMoA3oECDQQBQ)**;**

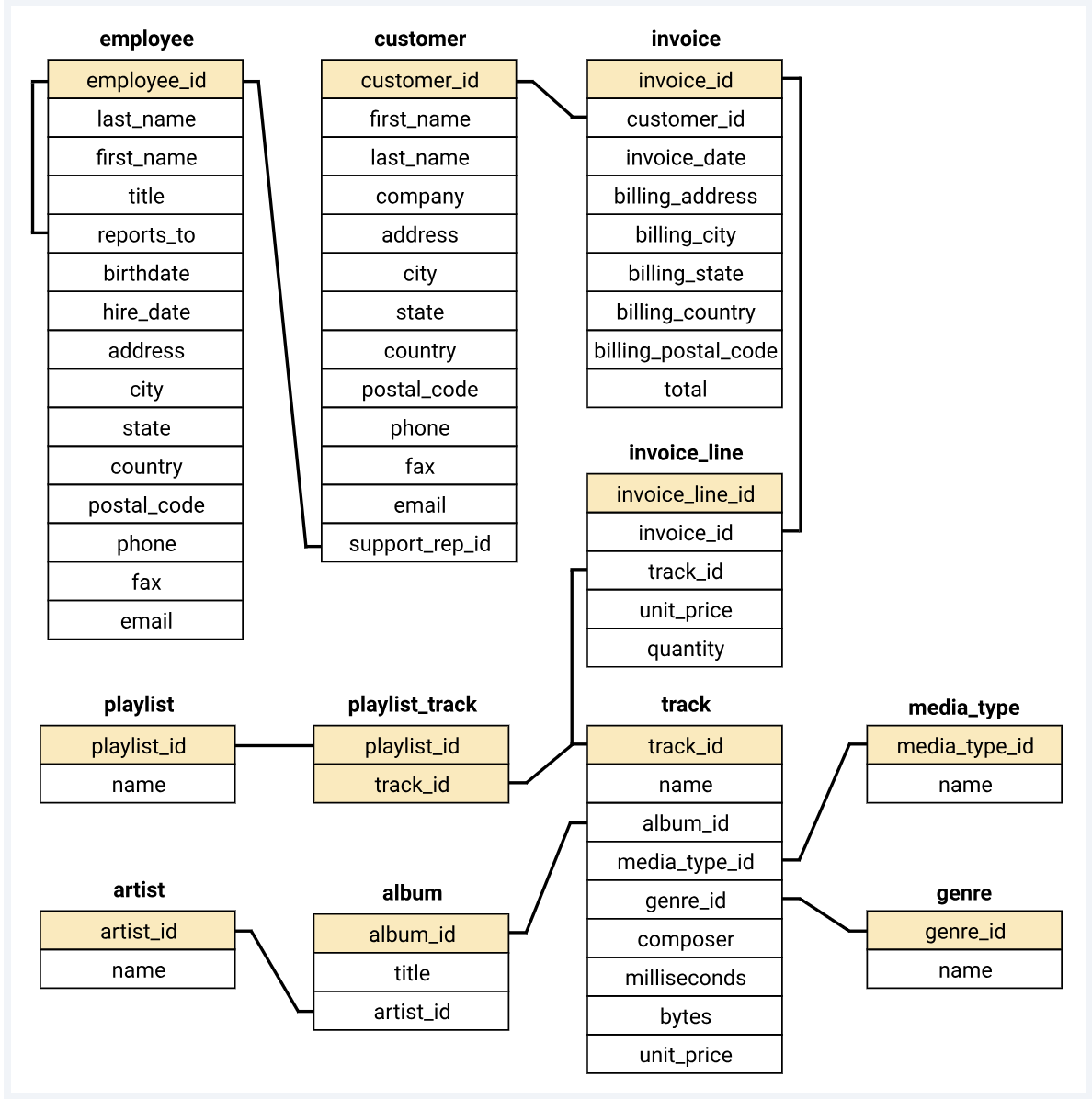
[**Google Home**](https://www.google.com/search?sa=X&sca_esv=4812ae3c1a5b5c28&sca_upv=1&rlz=1C1RXQR_enIN1060IN1064&biw=1536&bih=826&sxsrf=ADLYWILXtimFuKvr9qVKXaDsgrLdR8FYXQ:1727590667769&q=Google+Home&stick=H4sIAAAAAAAAAONgVuLVT9c3NEzOs6zIzi1IW8TK7Z6fn56TquCRn5sKAM_rbKkfAAAA&ved=2ahUKEwjK_MbgwOeIAxVGcmwGHZ2GCAYQmxMoBHoECDQQBg)**;**[**Windows**](https://www.google.com/search?sa=X&sca_esv=4812ae3c1a5b5c28&sca_upv=1&rlz=1C1RXQR_enIN1060IN1064&biw=1536&bih=826&sxsrf=ADLYWILXtimFuKvr9qVKXaDsgrLdR8FYXQ:1727590667769&q=Windows&stick=H4sIAAAAAAAAAONgVuLQz9U3MCmKt1jEyh6emZeSX14MAFTjqsQWAAAA&ved=2ahUKEwjK_MbgwOeIAxVGcmwGHZ2GCAYQmxMoBXoECDQQBw)

**Developer : Saavan media limited .**

**Category : Music and podcast.**

**Data taken from – KAGGLE.COM**

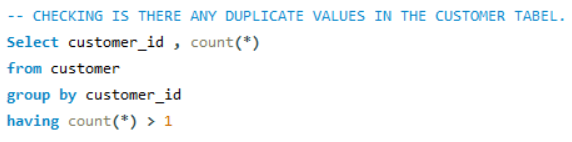
Schema Diagram

****

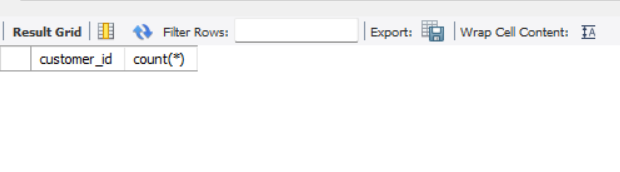
**Question Set 1 – Easy**

**Que- 1. Check is there any duplicate in the customer tabel.**

**Ans –** I generated a query using the **COUNT()** clause which count all the values from the tabel. Later I group the value on the basis of customer\_id , and then applied a filter after **GROUP BY** clause of with the help of **HAVING** clause showing is there any count value more than 1 time or not .

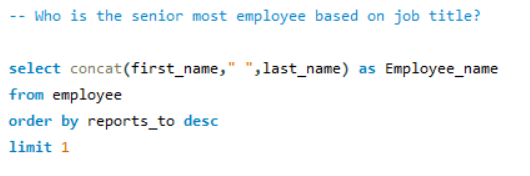
****

This proves that we don’t have any duplicate values in the customer tabel.

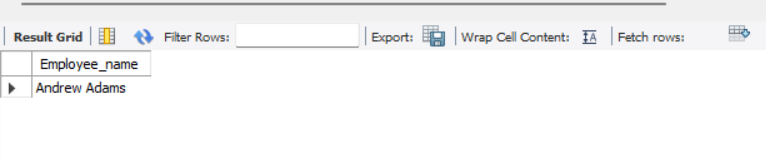
****

**Que-2.Who is the senior most employee based on job title?**

**Ans-** For this problem , I have used the **CONCAT()** function as it will concatenate the first\_name and last\_name and will give the employee full\_name from the employee tabel. As reports\_to column shows which employee report to which employee, I have used the **ORDER BY** clause and order the report\_to column in **desc** pattern and put the **LIMIT** of 1.

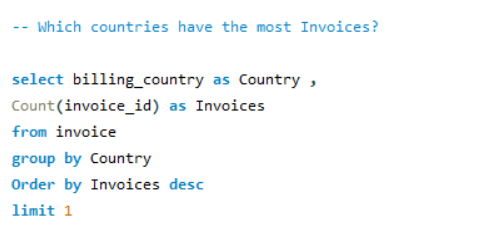
****

This shows the senior most employee in the company to whom other employee report to.

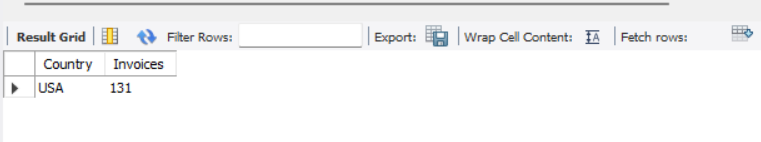
****

**Que-3. Which country have the most Invoice?**

**Ans –** For this problem , I have used the invoice tabel as it contains the invoice\_id and billing\_country names. I did **COUNT()** of invoice\_id and **ALIASING** as Invoices and same done for billing\_country as Country. Later I **group** the country and order Invoices in **desc** order , this will show the count of invoices in descending order for different countries.



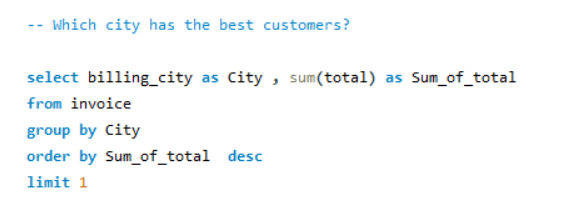
By applying the **limit** of **1** to the written query we will get the country with most invoices.



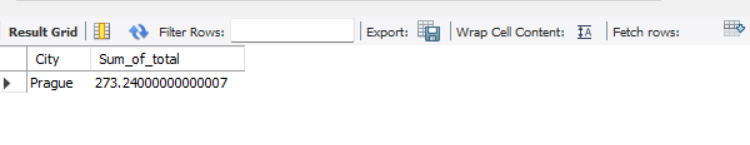
**Que-4.Which city has the best customers?**

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.

**Ans-** We want the highest sum of invoice totals, for this invoice tabel is used. The billing\_city is **ALIASING** as City and as we want sum of total , I have used **SUM()** function to get the Sum\_of\_total. Later I group it by City and **ORDER BY** Sum\_of\_total in **descending** order.

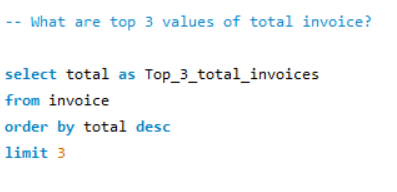
****

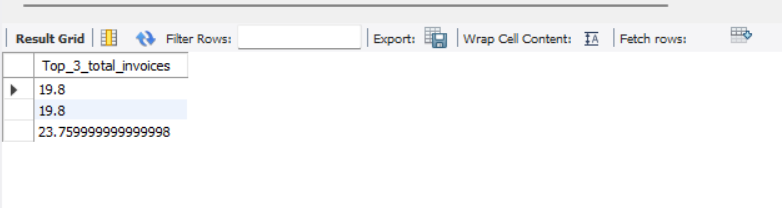
By appyling the **limit** of **1** we will get the City who has the best customer.

****

**Que-5. What are top 3 values of total invoice?**

**Ans -** For this problem invoice tabel is used. Total column is **ORDER BY** in **descending** order and **LIMIT 3** is taken to get the top 3 values of total invoice.

****

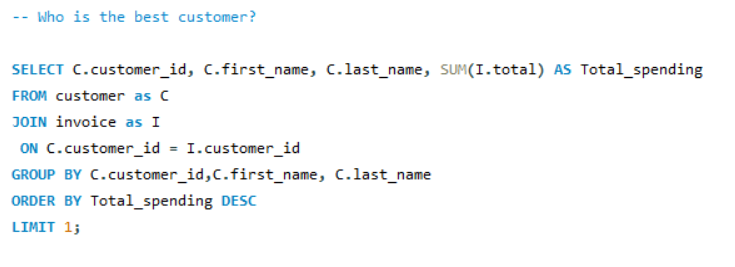
**Result -**

**Que-6. Who is the best customer?**

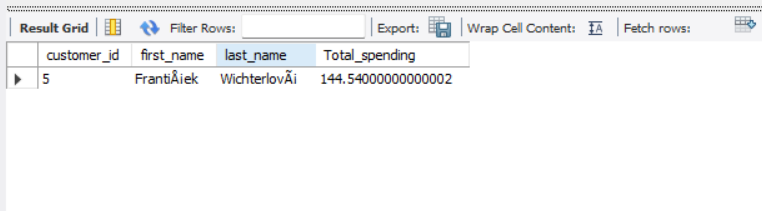
The customer who has spent the most money will be declared the best customer. Write a query that returns the person who has spent the most money.

**Ans –** For this problem I have used two tabels as coulumns customer\_id , first\_name , last\_name are present in the customer tabel and total column is present in invoice tabel . These two tabels are join with the help of **INNER JOIN** and joined **ON** the basis of common column customer\_id.

By joining two tabels we will get Total\_spending ,customer\_id,first\_name and last\_name.

****

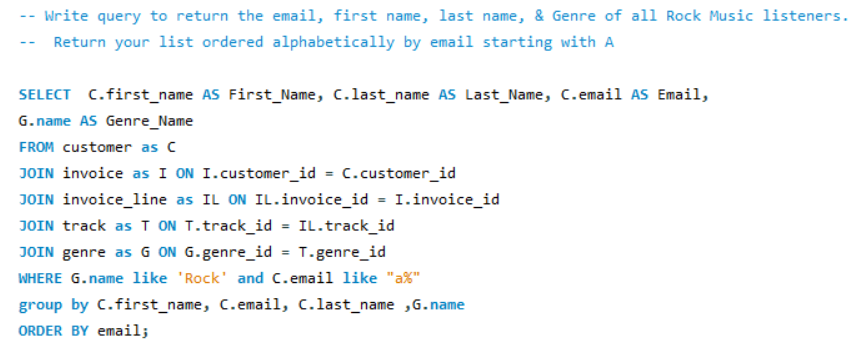
As we got the columns together now we have to find the best customer by putting **ORDER By** on Total\_spending in descending order and putting **limit** 1.

****

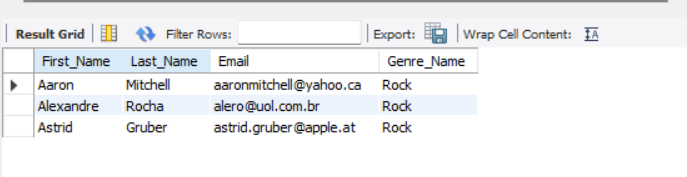
**Question Set 2 – Moderate**

**Que-1. 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.**

**Ans-** This SQL query retrieves customer information from a database, including their first name, last name, email, and the genre of music they purchased. It **joins** multiple tables: customer, invoice, invoice\_line, track, and genre. The query **filters** customers who purchased tracks from the **"Rock"** genre and whose email contains **"a%"** using a **LIKE clause**. It **groups** the results by the customer's first name, email, last name, and genre name and orders the output by email. The query provides customer details based on their rock genre purchases and a specific email pattern.

****

**Result -**

****

**Que-2.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 .**

**Ans -** This SQL query retrieves the top 10 artists with the most rock tracks in a music database.

Here's how it works:

1. Selects artist names and the count of their tracks.

2. From the **artist table (A),** it **joins the album table (B)** based on matching **artist IDs**.

3. **Joins** the **track table (T)** based on matching **album IDs**.

4. **Joins** the **genre table (G)** to **filter** for the genre with the name **"Rock."**

5. The **WHERE** clause restricts the results to only tracks of the **"Rock" genre**.

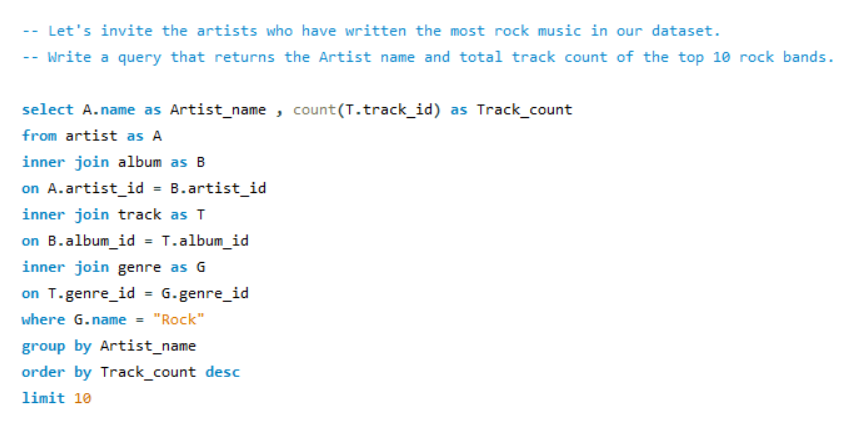
6. **Groups** the results by the artist name.

7. Uses the **COUNT()** function to count the number of tracks for each artist.

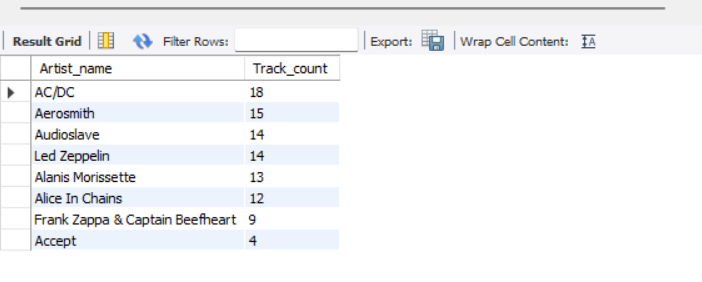
8. **Orders** the results by track count in descending order **(DESC).**

9. **Limits** the output to the top 10 artists.

10. The result displays the artist name and their rock track count.



**Result-**



**Que-3.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.**

**Ans –**

This SQL query retrieves a list of songs longer than the average song length, sorted by their duration in descending order. Here's a breakdown of how it works:

1. The **SELECT** clause retrieves the name of the song and its milliseconds value, renaming them as Song\_name and Song\_length\_in\_milliseconds, respectively.

2. The **WHERE clause** filters the tracks, only selecting songs whose length in milliseconds is greater than the average length of all songs.

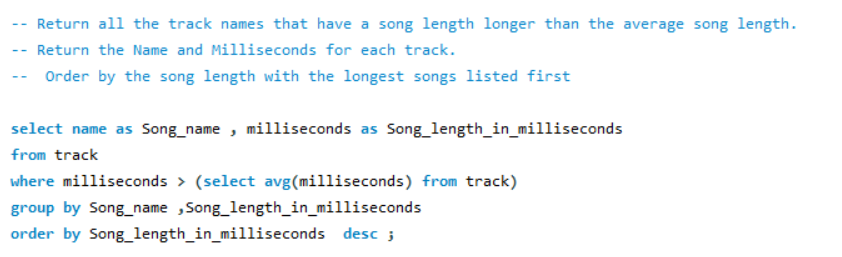
3. The **subquery** (select avg(milliseconds) from track) computes the average song length in milliseconds.

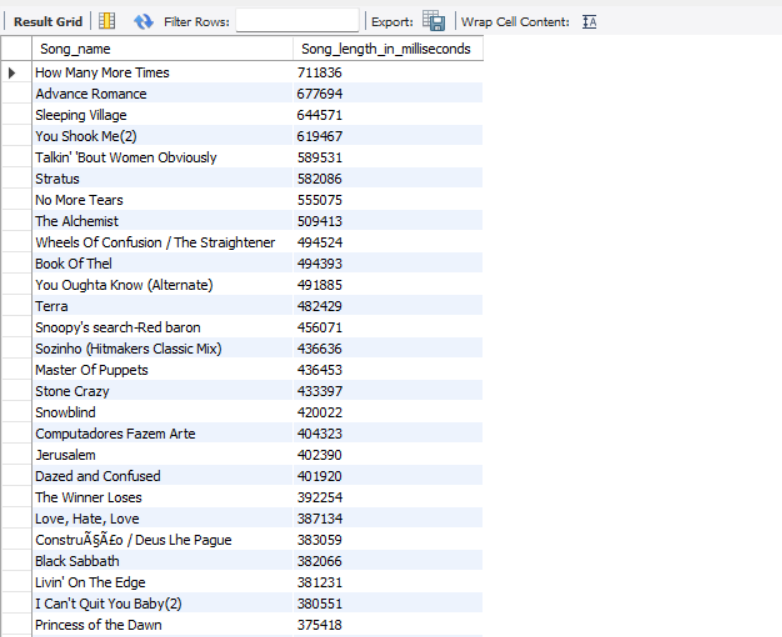
4. The **outer query** compares each song’s length to the **average value** computed by the subquery.

5. The **GROUP BY** clause groups the results by both Song\_name and Song\_length\_in\_milliseconds to ensure no duplicate records.

6. The **ORDER BY** clause sorts the results in descending order based on song length.

7. The final result is a list of songs whose lengths exceed the average, displayed in descending order of their length.

****

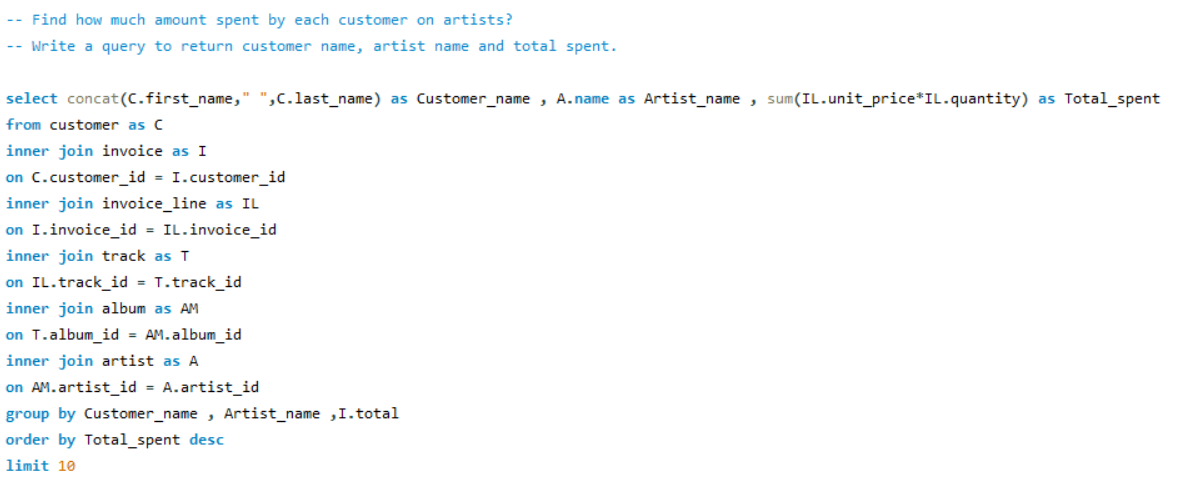
**Result-**

**Question Set 3 – HARD**

**Que-1.Find how much amount spent by each customer on artists?**

**Write a query to return customer name, artist name and total spent.**

**Ans –**



This SQL query retrieves the top 10 customers by the total amount they spent on tracks by various artists.

Here's how it works:

1. **Customer Name Concatenation**: The query selects the concatenation of the first and last name from the customer table as Customer\_name.

2. **Artist Name Selection**: It selects the artist's name (A.name) from the artist table.

3. **Total Amount Spent Calculation**: The total amount spent is calculated by summing the product of IL.unit\_price and IL.quantity from the invoice\_line table for each customer.

4. **Inner Join Operations**:

The **customer table** is **joined** with the invoice table using **customer\_id** to link customers with their purchases.

The **invoice\_line** table is **joined** to link invoices to individual tracks purchased.

The **track table** is **joined** using **track\_id** to associate line items with tracks.

The **album table** is joined to associate tracks with their respective albums.

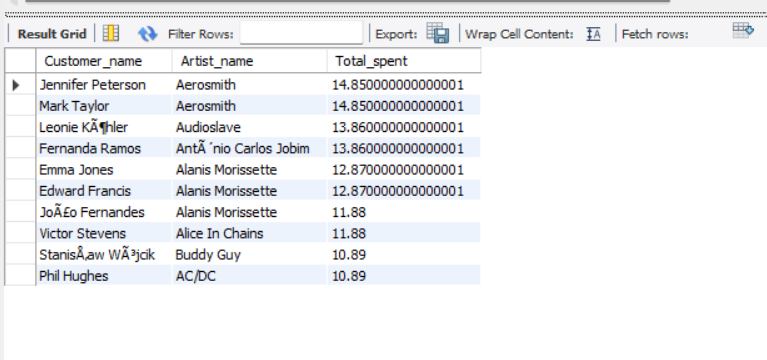
The **artist table** is joined to get the artist information for each album.

5. **Grouping**: The results are grouped by Customer\_name, Artist\_name, and I.total to summarize the total spent per customer for each artist.

6. **Sorting**: The results are sorted by Total\_spent in descending order to show the highest spenders first.

7. **Limiting**: The query limits the output to the top 10 customers with the highest total spent.

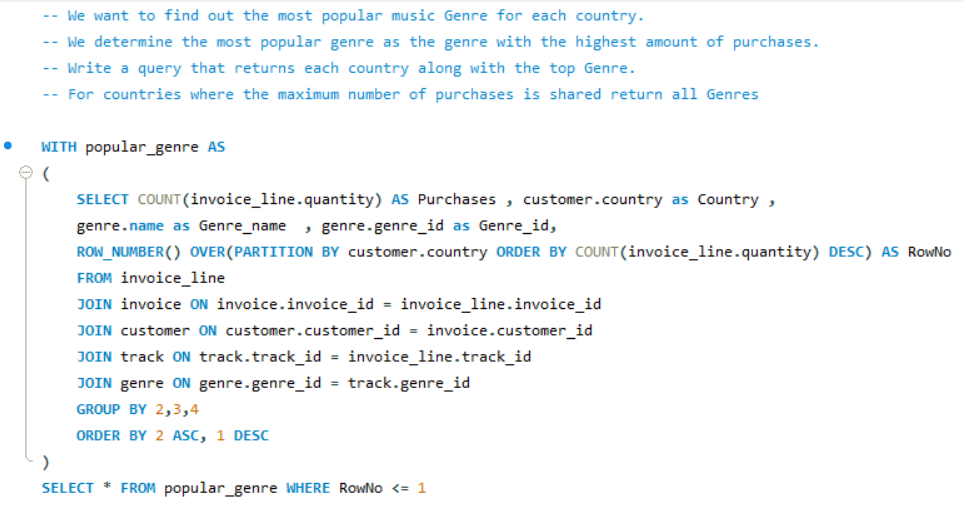
This returns the top 10 customers who spent the most on tracks by specific artists.

**Result-**

**Que-2.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.**

**Ans –**



This SQL query uses a Common Table Expression (CTE) to find the most popular genre for each country.

Here's how it works:

1. **CTE** (WITH populat\_genre AS): A CTE named populat\_genre is created to calculate the total number of purchases for each genre in each country.

2. **COUNT**(invoice\_line.quantity): The number of purchases for each genre is calculated by counting the quantity from the invoice\_line table.

3. **GROUP BY** Country and Genre: The data is grouped by customer.country, genre.name, and genre.genre\_id.

4. **ROW\_NUMBER Function**: The ROW\_NUMBER() function assigns a rank to genres in each country, ordered by the highest number of purchases.

5. **JOINs**:

invoice\_line is joined with invoice to associate each line item with a customer.

**customer** is joined to get the **country**.

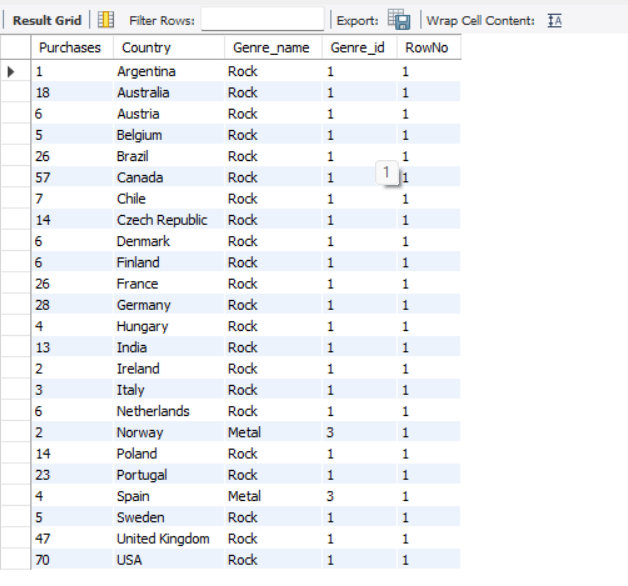
**track** is joined to link each track with its **genre**.

**genre** is joined to retrieve **genre** information.

6. **Selecting Top Genre**: The outer query selects from the CTE and filters for rows where RowNo <= 1, which retrieves the most popular genre for each country.

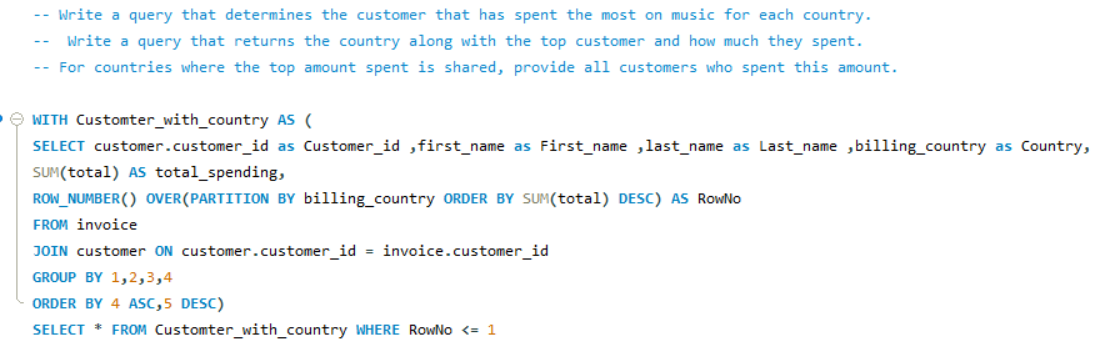
This approach provides the top genre by purchase count in each country.

**Result-**



**Que-3.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.**

**Ans –**

This SQL query uses a Common Table Expression (CTE) to find the top-spending customer from each country.

Here’s a breakdown of how it works:

1. The query begins with a **WITH clause**, defining a CTE named Customer\_with\_country.

2. **Inside the CTE**, it selects the customer’s customer\_id, first\_name, last\_name, and billing\_country (renamed as Country).

3. It also calculates the total spending for each customer by summing the total field from the invoice table and renaming it as total\_spending.

4. The **ROW\_NUMBER()** function is used to assign a unique row number (RowNo) to each customer, partitioned by their billing\_country.

5. The **OVER(PARTITION BY** billing\_country **ORDER BY SUM**(total) **DESC**) ensures that the row numbers are generated separately for each country and sorted by the customer's total spending in descending order, with the highest spender getting row number 1.

6. It performs an **INNER JOIN** between the invoice and customer tables, connecting the customer data via customer\_id.

7. The query **groups** the results by customer\_id, first\_name, last\_name, and billing\_country, ensuring that each customer’s spending is aggregated correctly.

8. The **GROUP BY** clause uses positional references (1, 2, 3, 4) for customer\_id, first\_name, last\_name, and billing\_country.

9. The **ORDER BY** clause sorts the data first by Country (position 4, in ascending order), and then by total\_spending (position 5, in descending order).

10. After defining the CTE, the query selects all columns (\*) from the Customer\_with\_country CTE.

11. The final **WHERE** clause filters the CTE results, selecting only rows where RowNo <= 1.

12. This ensures that only the highest-spending customer (with RowNo = 1) from each country is selected.

13. The result will list each country's top-spending customer based on their total invoice spending.

14. By using CTE, the query simplifies processing by calculating row numbers first, then applying filtering in the final step.

**Result -**

