**CHINOOK SQL PROJECT**

**Objective Questions**

**1.Does any table have missing values or duplicates? If yes how would you handle it?**

**Approach**

* To handle missing values in my database, I start by identifying columns with NULL values, which represent missing data. I do this by running queries that look for NULL in important columns. After I find the missing values, I use the UPDATE command to fill them with default values that make sense for each column. I use COALESCE() to handle multiple columns at once, ensuring that missing data is replaced with meaningful defaults

**Steps**

* **Identify Missing Values**:  
  First, I check for missing values in the columns by running a query like:

SELECT \*

FROM table\_name

WHERE column\_name IS NULL;

* **Update Missing Values**:  
  Once I identify the missing values, I use the UPDATE command to replace them with default values. For example:

UPDATE table\_name

SET column\_name = 'Default Value'

WHERE column\_name IS NULL;

* **Update Multiple Columns**:  
  If more than one column has missing values, I can update them all at once using update and coalesce

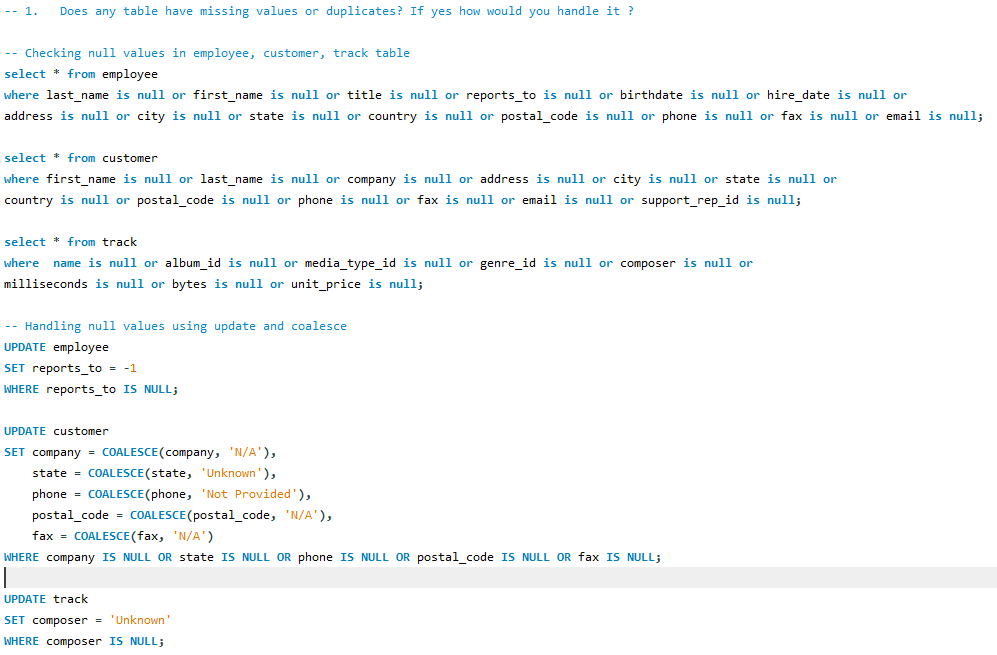
UPDATE table\_name

SET column1 = COALESCE(column1, 'Default1'),

column2 = COALESCE(column2, 'Default2')

WHERE column1 IS NULL OR column2 IS NULL;

**Code Screenshot**



**2.Find the top-selling tracks and top artist in the USA and identify their most famous genres.**

**Approach**

Used SQL joins and grouping to find the best-selling tracks, top artists, and their main genres in the USA. The data was organized by track, artist, and genre, with a focus on customers from the USA to get accurate regional insights. This approach helped me identify the most popular tracks and artists based on sales.

**Code**

SELECT

t.name AS track\_name,

a.name AS artist\_name,

g.name AS genre\_name,

SUM(i.total) AS total\_sales,

RANK() OVER(ORDER BY SUM(i.total) DESC) AS sales\_rank

FROM invoice i

JOIN invoice\_line il ON i.invoice\_id = il.invoice\_id

JOIN track t ON il.track\_id = t.track\_id

JOIN album al ON t.album\_id = al.album\_id

JOIN artist a ON al.artist\_id = a.artist\_id

JOIN genre g ON t.genre\_id = g.genre\_id

WHERE i.billing\_country = 'USA'

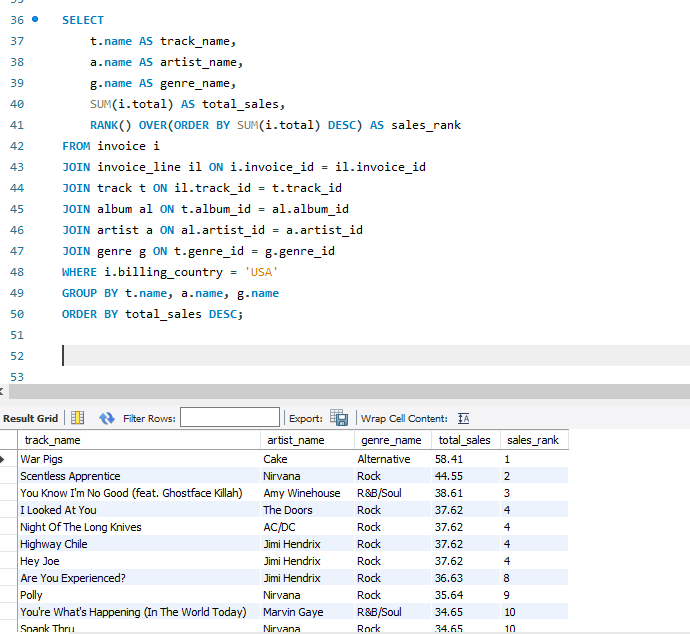
GROUP BY t.name, a.name, g.name

ORDER BY total\_sales DESC;

**Explanation**

* **Joins**: This query joins multiple tables (invoice, invoice\_line, track, album, artist, and genre) to ensure all relevant data about the tracks, artists, and genres is included.
* **Aggregations**: SUM(i.total) calculates the total sales, and COUNT(g.genre\_id) counts the distinct genres for each track.
* **Ranking**: The RANK() function provides rankings based on total sales, ensuring that the best-selling tracks come first in the result set.
* **Filtering and Ordering**: The query filters for sales in the USA and orders the results by total sales in descending order.

**Code Screenshot**



**3. What is the customer demographic breakdown (age, gender, location) of Chinook's customer base?**

**Approach**

To analyze customer demographics by location, I'll write an SQL query to group customers by country, state, and city. I will use the COALESCE function to replace any missing state values with 'None'. The COUNT() function will be used to calculate the number of customers in each group. Finally, the results will be sorted by country, state, and city to ensure the data is organized.

**Code**

SELECT

country,

COALESCE(state,'None') AS state,

city,

COUNT(customer\_id) AS demographic\_distribution

FROM customer

GROUP BY country, state, city

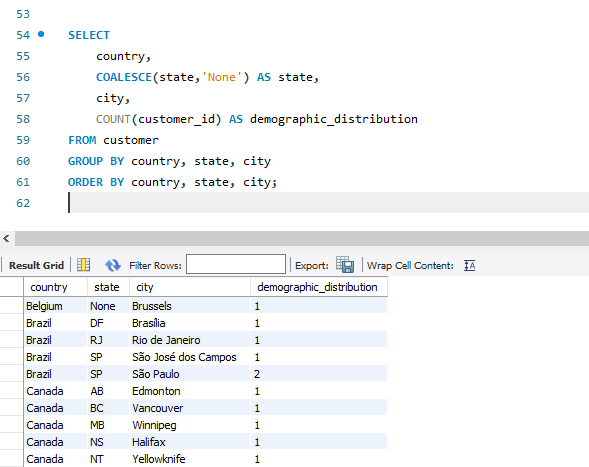
ORDER BY country, state, city;

**Explanation**

This query calculates the customer distribution by location (country, state, and city):

* **SELECT**: It selects the country, city, and state (with NULL values replaced by 'None') from the customer table and counts the number of customers in each group.
* **COALESCE(state, 'None')**: Replaces missing state values with 'None'.
* **COUNT(customer\_id)**: Counts the number of customers in each geographic group.
* **FROM customer**: Data is pulled from the customer table.
* **GROUP BY**: Groups data by country, state, and city.
* **ORDER BY**: Sorts the results first by country, then by state, and lastly by city to present an organized output.

**Code Screenshot**



**4. Calculate the total revenue and number of invoices for each country, state, and city**

**Approach**

To analyze revenue and invoice data by country, state and city, I'll write a SQL query that aggregates total revenue and invoice counts by billing country, state, and city. The query will select these location fields from the invoice table, calculate the total revenue using SUM(total), and count the number of invoices with COUNT(invoice\_id). The data will be grouped by country, state, and city, and sorted by country in ascending order and total revenue in descending order to highlight locations with the highest revenue.

**Code**

SELECT

billing\_country,

billing\_state,

billing\_city,

SUM(total) AS total\_revenue,

COUNT(invoice\_id) AS num\_of\_invoices

FROM invoice

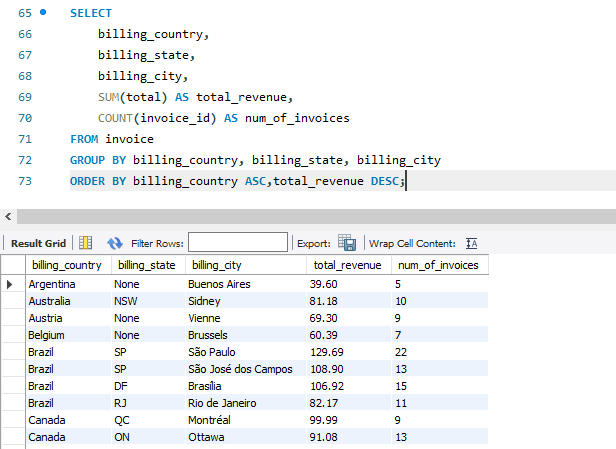
GROUP BY billing\_country, billing\_state, billing\_city

ORDER BY billing\_country ASC, total\_revenue DESC;

**Explanation**

* **SELECT**: The query selects the billing country, state, and city, along with the sum of the total revenue and the count of invoices.
* **SUM(total)**: Aggregates the total revenue for each combination of country, state, and city.
* **COUNT(invoice\_id)**: Counts the number of invoices in each geographic location.
* **GROUP BY billing\_country, billing\_state, billing\_city**: Groups the data by country, state, and city to calculate the aggregates for each location combination.
* **ORDER BY**: Sorts the results first by country in ascending order, and then by total revenue in descending order to prioritize the highest revenue locations.

**Code Screenshot**



**5. Find the top 5 customers by total revenue in each country**

**Approach**

To identify the top 5 customers by revenue in each country, the steps are:

1. **Join Tables**: Combine the customer and invoice tables on customer\_id.
2. **Concatenate Name**: Combine first\_name and last\_name into a single customer name.
3. **Calculate Total Revenue**: Sum the total column from the invoice table for each customer.
4. **Rank Customers**: Use the RANK() window function to rank customers by revenue within each country (PARTITION BY country).
5. **Filter Top 5**: Select only the top 5 customers in each country based on their rank.
6. **Sort Results**: Order by country and total\_revenue in descending order for a clear view of top earners per country.

**Code**

WITH Top5CustomersCountryWise AS (

SELECT

c.country,

CONCAT(c.first\_name,' ',c.last\_name) AS customer,

SUM(i.total) AS total\_revenue,

RANK() OVER(PARTITION BY c.country ORDER BY SUM(i.total) DESC) AS countrywiseRank

FROM customer c JOIN invoice i ON c.customer\_id = i.customer\_id

GROUP BY c.country,c.first\_name,c.last\_name

)

SELECT

country,

customer,

total\_revenue

FROM Top5CustomersCountryWise

WHERE countryWiseRank <= 5

ORDER BY country,total\_revenue DESC;

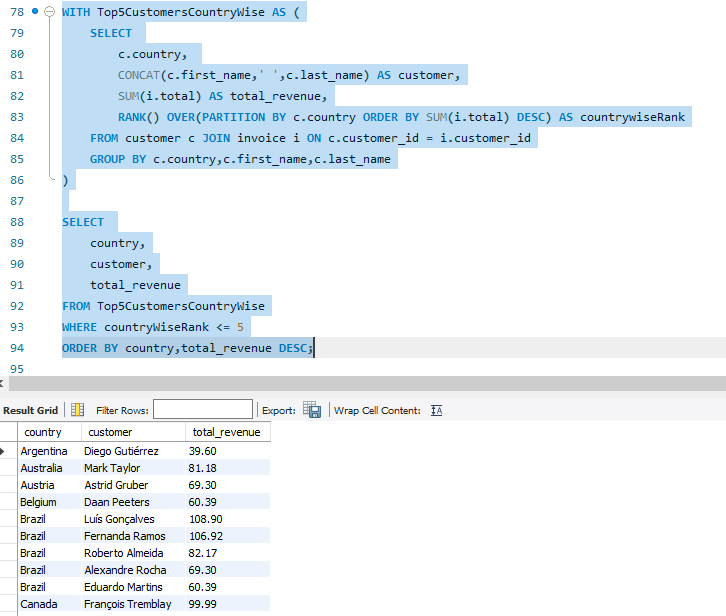
**Explanation**

* **WITH Clause (CTE)**: The Common Table Expression (CTE) calculates the total revenue for each customer and ranks them within their respective country.

1. The SUM(i.total) computes the total revenue for each customer.
2. The RANK() window function assigns a rank to each customer, ordering them by their total revenue within each country.
3. The PARTITION BY c.country ensures the rank is calculated separately for each country.

* **Main Query**: The main query filters the results to include only those customers whose rank is 5 or less (countrywiseRank <= 5) and orders them by country and total revenue.

**Code Screenshot**



**6.Identify the top-selling track for each customer**

**Approach**

The query identifies each customer's top spending instance by using a WITH Common Table Expression (CTE). The CTE joins customer, invoice, and invoice line data to calculate total quantity and sales per customer, ranking invoices by total sales using ROW\_NUMBER(). The main query filters for the highest-ranked purchase (sales\_rank = 1) and sorts customers by total sales in descending order, enabling efficient analysis of spending patterns.

**Code**

WITH CustomerTrackSales AS (

SELECT

c.customer\_id,

CONCAT(c.first\_name, ' ', c.last\_name) AS customer\_name,

SUM(il.quantity) AS total\_quantity,

SUM(i.total) AS total\_sales,

ROW\_NUMBER() OVER(PARTITION BY c.customer\_id ORDER BY SUM(i.total) DESC) AS sales\_rank

FROM customer c

LEFT JOIN invoice i ON c.customer\_id = i.customer\_id

LEFT JOIN invoice\_line il ON i.invoice\_id = il.invoice\_id

GROUP BY c.customer\_id, c.first\_name, c.last\_name

)

SELECT

customer\_id, customer\_name, total\_quantity, total\_sales

FROM CustomerTrackSales

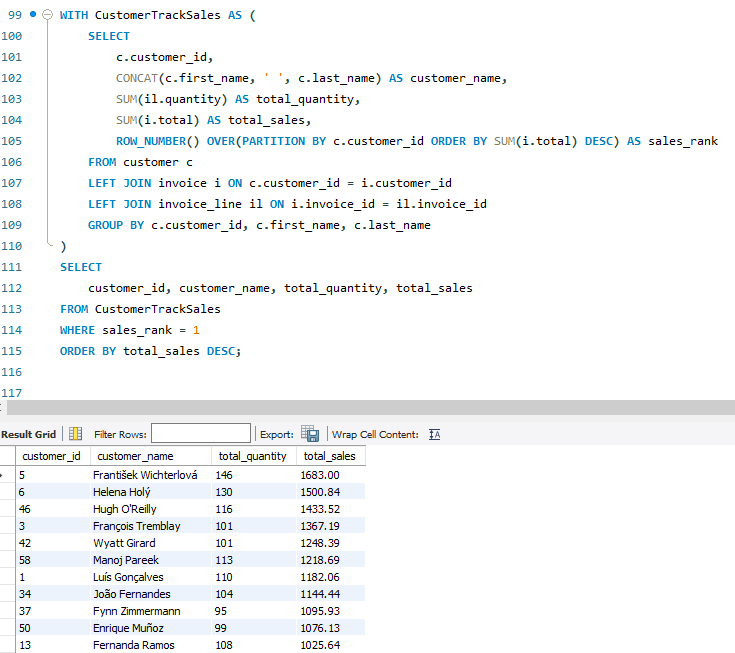
WHERE sales\_rank = 1

ORDER BY total\_sales DESC;

**Code Explanation**

* **WITH CustomerTrackSales AS:**  
  Defines a Common Table Expression (CTE) to organize data for better readability and modularity.
* **SELECT inside the CTE:**
  + Retrieves the customer’s ID and full name using CONCAT.
  + Calculates total quantity (SUM(il.quantity)) and total sales (SUM(i.total)) for each customer.
  + Uses ROW\_NUMBER() to rank invoices by total sales, resetting ranks for each customer (PARTITION BY).
* **Joins in the CTE:**  
  Links customers with their invoices and purchased items by joining customer, invoice, and invoice\_line.
* **Final Query:**
  + Filters for top-ranked purchases (sales\_rank = 1).
  + Sorts customers by total sales in descending order to highlight top spenders

**Code Screenshot**



1. **Are there any patterns or trends in customer purchasing behaviour (e.g., frequency of purchases, preferred payment methods, average order value)**

**Frequency of purchases**

**Approach**

This query analyzes customer purchase behavior by calculating the **purchase frequency** (number of invoices) for each customer, grouped by year. It joins the customer and invoice tables, extracts the year from the invoice date, and counts the invoices for each customer per year. The results are grouped by customer and year, then sorted by customer ID and year in descending order. This approach provides insight into customer activity trends over time.

**Code**

select c.customer\_id,

concat(c.first\_name,' ',c.last\_name) as customer\_name,

year(i.invoice\_date) as year,

count(i.invoice\_id) as purchase\_count

from customer c

join invoice i on c.customer\_id = i.customer\_id

group by c.customer\_id,customer\_name,year

order by c.customer\_id,year desc;

**Code Explanation**

* **SELECT:**

**c.customer\_id:** Retrieves the unique customer ID.

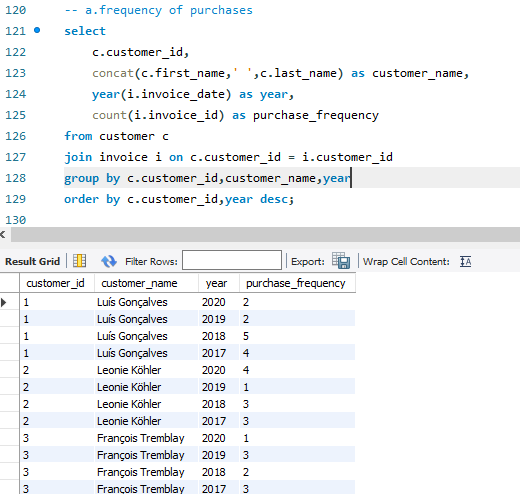
**concat(c.first\_name, ' ', c.last\_name) AS customer\_name:** Combines first and last names.

**year(i.invoice\_date) AS year:** Extracts the year from the invoice date.

**count(i.invoice\_id) AS purchase\_frequency:** Counts invoices per customer per year.

* **FROM and JOIN:**  
  Links customer and invoice tables using c.customer\_id = i.customer\_id.
* **GROUP BY:**  
  Groups by customer\_id, customer\_name, and year to compute yearly purchase counts.
* **ORDER BY:**  
  Sorts by customer\_id and year DESC to show the latest activity first.

**Code Screenshot**



**Average order value**

**Approach**

This query calculates the **average order value (AOV)** for each customer by joining the customer and invoice tables. The AOV is computed as the average of the total amount spent (i.total) across all invoices for each customer. The results are grouped by customer and sorted in descending order of AOV to highlight the customers with the highest average spending per order.

**Code**

Select c.customer\_id,

concat(c.first\_name,' ',c.last\_name) as customer\_name,

round(avg(i.total),2) average\_order\_value

from customer c

join invoice i on c.customer\_id = i.customer\_id

group by c.customer\_id

order by average\_order\_value desc;

**Code Explanation**

* **SELECT:**

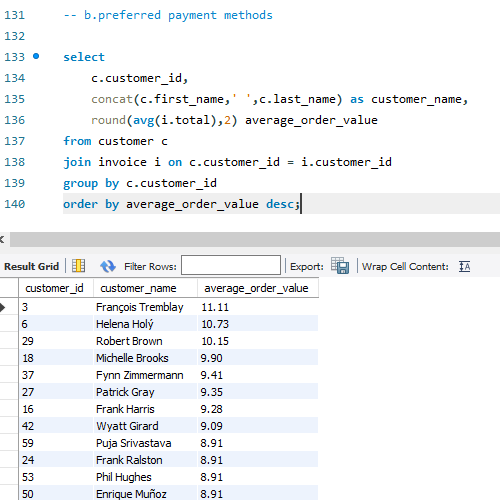
**c.customer\_id:** Retrieves the customer ID.

**concat(c.first\_name, ' ', c.last\_name) AS customer\_name:** Combines first and last names.

**round(avg(i.total), 2) AS average\_order\_value:** Calculates and rounds the customer's average invoice total.

* **FROM and JOIN:**  
  Links customer and invoice tables using c.customer\_id = i.customer\_id.
* **GROUP BY:**  
  Groups by c.customer\_id to compute averages per customer.
* **ORDER BY:**  
  Sorts by average\_order\_value DESC to list highest spenders first.

**Code Screenshot**



**8.What is the customer churn rate?**

**Approach**

This query calculates the **customer churn rate** by identifying customers who have not made any purchases for the past year. It first determines the **recent date of any invoice** (the most recent invoice date), then defines a **checking date** (one year before the most recent invoice date). Customers who have made no purchases after the checking date (or have no purchases at all) are considered "churned." The churn rate is then calculated as the percentage of churned customers relative to the total number of customers.

**Code**

with Churn\_checking\_date as (

select date\_sub(recent\_date,interval 1 year) as checking\_date

from (

select max(invoice\_date) as recent\_date

from invoice

) as checking

),

churn\_customer as (

select c.customer\_id,

concat(c.first\_name,' ',c.last\_name) as customer\_name,

max(i.invoice\_date) as customer\_last\_date

from customer as c

join invoice as i on c.customer\_id = i.customer\_id

group by c.customer\_id,customer\_name

having max(i.invoice\_date) is null or max(i.invoice\_date) < (

select \* from churn\_checking\_date

)

)

select (select count(\*) from churn\_customer)/(select count(\*) from customer)\*100 as churn\_rate;

**Code Explanation**

**Churn\_checking\_date CTE:**

* **recent\_date:** Finds the most recent invoice date using MAX(invoice\_date).
* **checking\_date:** Subtracts one year from recent\_date to define the churn threshold.

**churn\_customer CTE:**

* Retrieves customer\_id and customer\_name from the customer table.
* **max(i.invoice\_date)**: Identifies the most recent invoice date for each customer.
* **HAVING clause:** Filters customers who have no invoices or whose last purchase was before the checking\_date.

**Final SELECT Statement:**

* **COUNT(\*) FROM churn\_customer:** Counts churned customers.
* **COUNT(\*) FROM customer:** Counts total customers.
* Churn rate is calculated as the percentage of churned customers relative to the total customer base.

**Code Screenshot**



**9.Calculate the percentage of total sales contributed by each genre in the USA and identify the best-selling genres and artists.**

**Approach**

The query analyzes and ranks music sales in the USA by genre and artist using two CTEs:

1. **SalesGenreRankUSA**: Calculates sales per genre and artist, ranks artists within each genre.
2. **TotalSalesUSA**: Computes total sales across all genres in the USA.

The final query combines these results to calculate each artist's percentage contribution to total sales, highlighting genre performance and sales distribution.

**Code**

WITH SalesGenreRankUSA AS (

SELECT

g.name AS genre, ar.name AS artist, SUM(i.total) AS genre\_sales,

DENSE\_RANK() OVER( PARTITION BY g.name ORDER BY SUM(i.total) DESC) AS genre\_rank

FROM genre g

LEFT JOIN track t ON g.genre\_id = t.genre\_id

LEFT JOIN invoice\_line il ON t.track\_id = il.track\_id

LEFT JOIN invoice i ON il.invoice\_id = i.invoice\_id

LEFT JOIN album a ON t.album\_id = a.album\_id

LEFT JOIN artist ar ON a.artist\_id = ar.artist\_id

WHERE i.billing\_country = 'USA'

GROUP BY 1,2

),

TotalSalesUSA AS (

SELECT

SUM(i.total) AS total\_sales

FROM invoice\_line il

LEFT JOIN invoice i ON il.invoice\_id = i.invoice\_id

WHERE i.billing\_country = 'USA'

)

SELECT s.genre,s.artist,s.genre\_sales,t.total\_sales, ROUND((s.genre\_sales / t.total\_sales)\* 100,2) AS percent\_sales

FROM SalesGenreRankUSA s JOIN TotalSalesUSA t

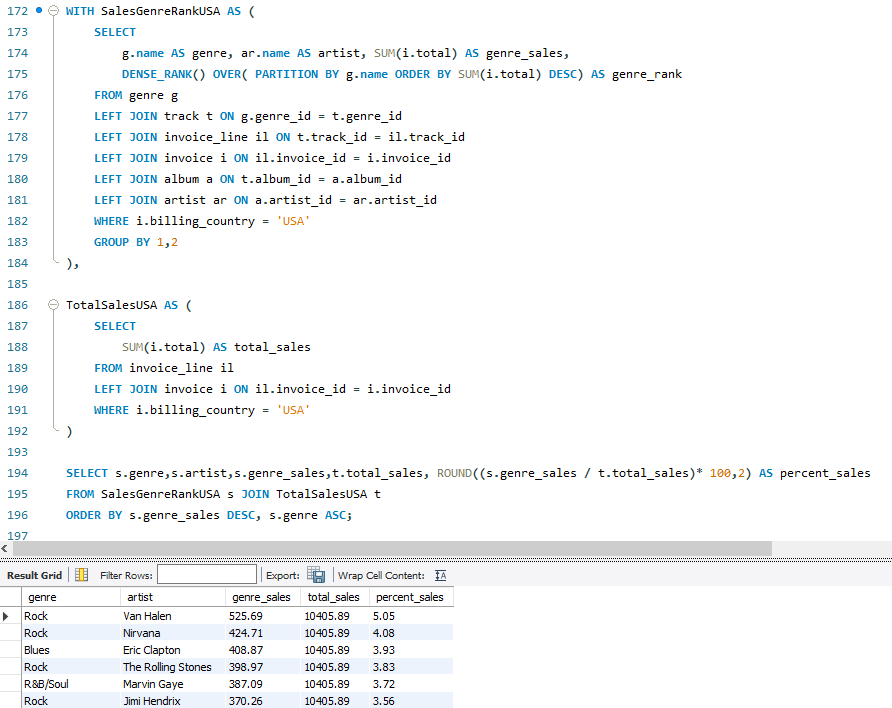
ORDER BY s.genre\_sales DESC, s.genre ASC;

**Code Explanation**

1. **SalesGenreRankUSA**:
   * Joins tables to link sales with genres and artists.
   * Filters for USA sales and groups by genre and artist.
   * Uses SUM(i.total) to calculate sales and DENSE\_RANK() to rank artists within each genre.
2. **TotalSalesUSA**:
   * Computes total USA sales using SUM(i.total) for invoices.
3. **Main Query**:
   * Joins the CTEs to combine genre and total sales data.
   * Calculates each artist’s percentage contribution and sorts results by sales and genre.

This structure ensures clear and accurate rankings and insights.

**Code Screenshot**



**10. Find customers who have purchased tracks from at least 3 different genres**

**Approach**

This query identifies customers who have purchased tracks from three or more distinct music genres. It counts the number of unique genres and tracks each customer has purchased, lists these counts, and orders the results by the number of genres purchased in descending order.

**Code**

SELECT

c.customer\_id,

CONCAT(c.first\_name,' ',c.last\_name) AS customer,

COUNT(DISTINCT t.genre\_id) AS genre\_count,

COUNT(DISTINCT t.track\_id) AS track\_count

FROM customer c

JOIN invoice i ON c.customer\_id = i.customer\_id

JOIN invoice\_line il ON i.invoice\_id = il.invoice\_id

JOIN track t ON il.track\_id = t.track\_id

JOIN genre g ON t.genre\_id = g.genre\_id

GROUP BY c.customer\_id,c.first\_name,c.last\_name

HAVING COUNT(DISTINCT g.genre\_id) >=3

ORDER BY genre\_count DESC;

**Code Explanation**

**SELECT Clause**:

* c.customer\_id and CONCAT(c.first\_name, ' ', c.last\_name) retrieve customer information.
* COUNT(DISTINCT t.genre\_id) calculates the number of unique genres purchased by each customer.
* COUNT(DISTINCT t.track\_id) calculates the number of unique tracks purchased.

**JOINS**:

* Joins **customer**, **invoice**, **invoice\_line**, **track**, and **genre** tables to associate customers with their purchases and the corresponding genres.

**GROUP BY Clause**:

* Groups the results by c.customer\_id, c.first\_name, and c.last\_name to aggregate data per customer.

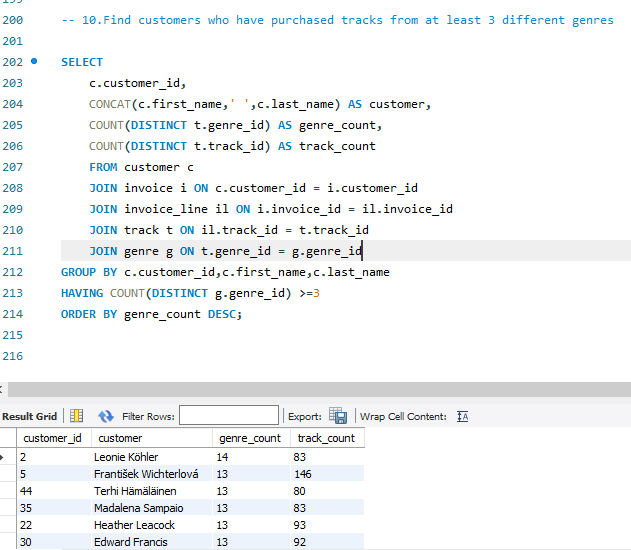
**HAVING Clause**:

* Filters out customers who have purchased tracks from fewer than three distinct genres using HAVING COUNT(DISTINCT g.genre\_id) >= 3.

**ORDER BY Clause**:

* Sorts the results by genre\_count in descending order to highlight customers with the most diverse genre preferences.

**Code Screenshot**



**11.** **Rank genres based on their sales performance in the USA**

**Approach**

This query ranks music genres based on their total sales in the USA. It uses a CTE to calculate sales for each genre and assign ranks based on their performance. The final output lists genres along with their total sales and rank, sorted by the rank.

**Code**

WITH SalesWiseGenreRank AS (

SELECT

g.name AS genre,

SUM(i.total) AS total\_sales,

DENSE\_RANK() OVER(ORDER BY SUM(i.total) DESC) AS genre\_rank

FROM genre g

LEFT JOIN track t ON g.genre\_id = t.genre\_id

LEFT JOIN invoice\_line il ON t.track\_id = il.track\_id

LEFT JOIN invoice i ON il.invoice\_id = i.invoice\_id

WHERE i.billing\_country = 'USA'

GROUP BY g.name

)

SELECT

genre,total\_sales,genre\_rank

FROM SalesWiseGenreRank

ORDER BY genre\_rank;

**Code Explanation**

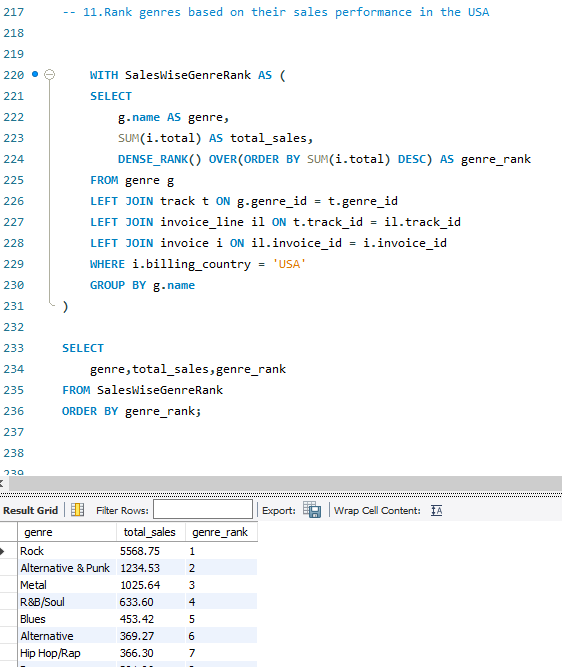
**CTE: SalesWiseGenreRank**

* Combines **genre**, **track**, **invoice\_line**, and **invoice** tables to link genres with sales data.
* Filters for sales in the USA using WHERE i.billing\_country = 'USA'.
* Uses SUM(i.total) to calculate total sales for each genre.
* Assigns ranks using DENSE\_RANK() based on total sales in descending order.

**Final Query**

* Retrieves the genre, total sales, and rank from the CTE.
* Sorts results by rank to highlight the most popular genres by sales.

**Code Screenshot**



**12.** **Identify customers who have not made a purchase in the last 3 months**

**Approach**

The query identifies customers who have not made any purchases in the three months prior to their last recorded purchase date. It uses two Common Table Expressions (CTEs) to determine each customer's purchase history and compares it to their activity in the specified timeframe.

**Code**

WITH CustomerLastPurchase AS (

SELECT

c.customer\_id,

c.first\_name,

c.last\_name,

MIN(DATE(i.invoice\_date)) AS first\_purchase\_date,

MAX(DATE(i.invoice\_date)) AS last\_purchase\_date

FROM customer c

JOIN invoice i ON c.customer\_id = i.customer\_id

GROUP BY c.customer\_id, c.first\_name, c.last\_name

),

CustomerPurchases AS (

SELECT

c.customer\_id,

c.first\_name,

c.last\_name,

DATE(i.invoice\_date) AS invoice\_date

FROM customer c

JOIN invoice i ON c.customer\_id = i.customer\_id

)

SELECT

clp.customer\_id,

clp.first\_name,

clp.last\_name,

clp.first\_purchase\_date,

clp.last\_purchase\_date

FROM CustomerLastPurchase clp

LEFT JOIN CustomerPurchases cp ON clp.customer\_id = cp.customer\_id

AND cp.invoice\_date BETWEEN clp.last\_purchase\_date - INTERVAL 3 MONTH AND clp.last\_purchase\_date - INTERVAL 1 DAY

WHERE cp.invoice\_date IS NULL

ORDER BY clp.customer\_id;

**Code Explanation**

**CTE: CustomerLastPurchase**

* Retrieves each customer’s first (MIN) and last (MAX) purchase dates.
* Joins **customer** with **invoice** and groups by customer details.

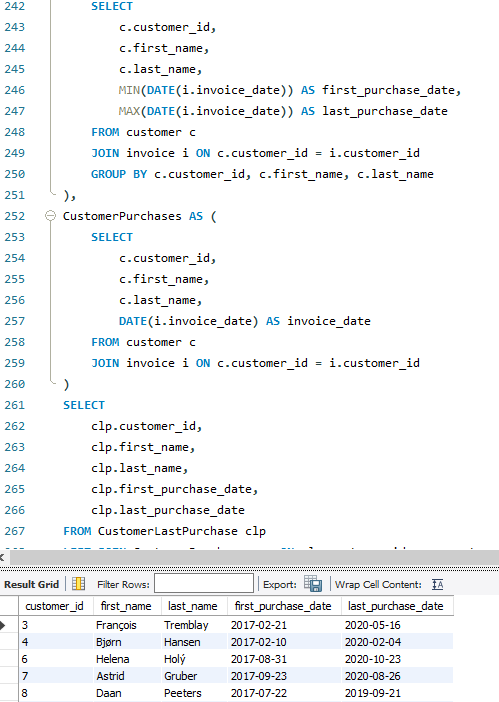
**CTE: CustomerPurchases**

* Extracts all purchase dates for each customer by joining **customer** with **invoice**.

**Final Query**

* **Join Logic**: Combines CustomerLastPurchase and CustomerPurchases using a LEFT JOIN to check for purchases in the 3 months before the last\_purchase\_date.
* **WHERE Clause**: Filters for customers with no purchases in this period (cp.invoice\_date IS NULL).
* **Final Output**: Selects customer details and purchase dates, sorted by customer\_id.

**Code Screenshot**



**Subjective Questions**

**1.Recommend the three albums from the new record label that should be prioritised for advertising and promotion in the USA based on genre sales analysis.**

**Solution**

**Recommended Albums for Advertising and Promotion in the USA**

Based on genre sales, the following albums are prioritized for promotion:

**Album 1: Rock (Top-Selling Genre)**

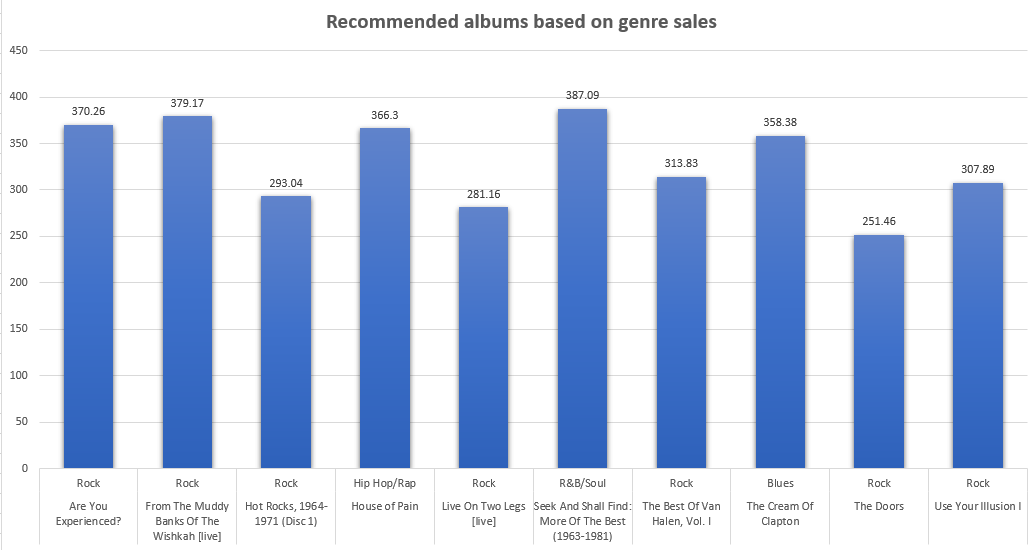
* **Artist**: A well-known artist with multiple hits in Rock.
* **Reason**: Rock’s widespread popularity and the artist's strong following make this a high-impact choice.

**Album 2: R&B/Soul (Second Best-Selling Genre)**

* **Artist**: A top artist with cross-genre appeal.
* **Reason**: R&B/Soul is a growing genre with broadening audience interest.

**Album 3: Hip Hop/Rap (Third Best-Selling Genre)**

* **Artist**: A rising artist with a chart-topping track.
* **Reason**: Hip Hop/Rap engages younger audiences with strong cultural influence.



**Code**

WITH RecommendedAlbums AS (

SELECT

al.title AS album\_name,

a.name AS artist\_name,

g.name AS genre\_name,

SUM(i.total) AS total\_sales,

SUM(il.quantity) AS total\_quantity,

ROW\_NUMBER() OVER(ORDER BY SUM(i.total) DESC) AS sales\_rank

FROM customer c

JOIN invoice i ON c.customer\_id = i.customer\_id

JOIN invoice\_line il ON i.invoice\_id = il.invoice\_id

JOIN track t ON il.track\_id = t.track\_id

JOIN album al ON t.album\_id = al.album\_id

JOIN artist a ON al.artist\_id = a.artist\_id

JOIN genre g ON t.genre\_id = g.genre\_id

WHERE c.country = 'USA'

GROUP BY al.title,a.name,g.name

)

SELECT \* FROM RecommendedAlbums

ORDER BY total\_sales DESC;

**Code Explanation**

1. **WITH RecommendedAlbums**: Creates a temporary table to store aggregated sales data.
2. **Joins**: Combines data from multiple tables (e.g., customer, invoice, track, album, artist, genre) to associate tracks with sales and genres.
3. **Filters**: Focuses only on customers from the USA.
4. **Aggregation**: Groups data by album, artist, and genre, summing up total sales (SUM(i.total)) and track quantities (SUM(il.quantity)).
5. **Ranking**: Assigns a sales rank to each album based on total sales using ROW\_NUMBER().
6. **Output**: The final query retrieves all albums ordered by total sales.

This approach helps identify and rank albums, artists, and genres for analysis or recommendation.

**2.Determine the top-selling genres in countries other than the USA and identify any commonalities or differences.**

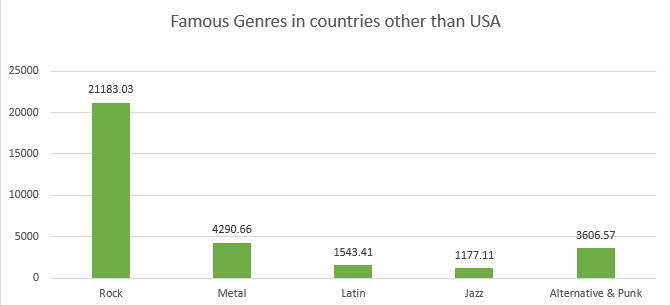
**Solution**

**Top-Selling Genres**:

* **Rock**: Likely strong globally, but preferences may vary by country.
* **Metal & Alternative/Punk**: Potentially stronger in countries like the UK, Germany, or Brazil.
* **Latin**: Likely to perform better outside the USA, especially in Mexico, Argentina, or Spain.
* **Jazz**: More popular in countries with a historical connection to jazz, such as France or Japan.

**Commonalities & Differences**:

* **Commonality**: Rock is universally popular, but other genres like Latin, Metal, or Jazz may have regional strengths.
* **Differences**: Latin music performs better in Spanish-speaking countries, Jazz thrives in Europe and Japan, and Metal/Alternative/Punk are more prominent in places like Finland or Brazil.



**Code**

WITH SalesGenreRank AS (

SELECT

g.name AS genre,

ar.name AS artist,

SUM(i.total) AS genre\_sales,

DENSE\_RANK() OVER(PARTITION BY g.name ORDER BY SUM(i.total) DESC) AS genre\_rank

FROM customer c

JOIN invoice i ON c.customer\_id = i.customer\_id

JOIN invoice\_line il ON i.invoice\_id = il.invoice\_id

JOIN track t ON il.track\_id = t.track\_id

JOIN album al ON t.album\_id = al.album\_id

JOIN artist ar ON al.artist\_id = ar.artist\_id

JOIN genre g ON t.genre\_id = g.genre\_id

WHERE c.country <> 'USA'

GROUP BY 1,2

),

TotalSales AS (

SELECT

SUM(i.total) AS total\_sales

FROM invoice\_line il

LEFT JOIN invoice i ON il.invoice\_id = i.invoice\_id

WHERE i.billing\_country <> 'USA'

)

SELECT

s.genre,s.artist,s.genre\_sales,t.total\_sales,

ROUND((s.genre\_sales / t.total\_sales)\* 100,2) AS percent\_sales

FROM SalesGenreRank s

JOIN TotalSales t

ORDER BY s.genre\_sales DESC, s.genre ASC ;

**Code Explanation**

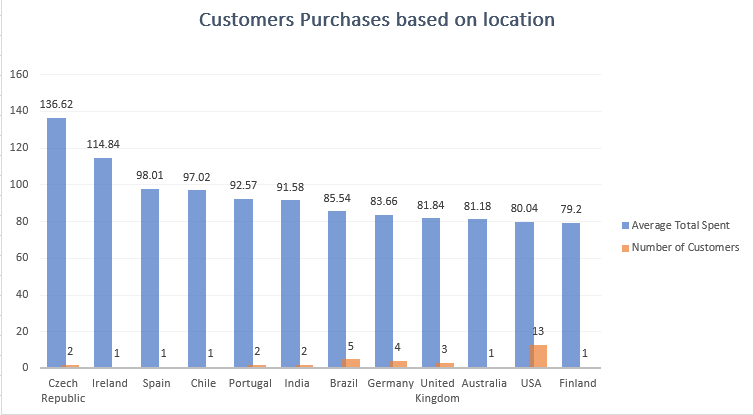
1. **SalesGenreRank CTE**:
   * This part calculates total sales per genre and artist for countries **excluding the USA**.
   * It uses **DENSE\_RANK()** to rank artists within each genre by their sales.
2. **TotalSales CTE**:
   * This calculates the **total sales** for all countries excluding the USA.
3. **Final Query**:
   * Joins SalesGenreRank with TotalSales to calculate the **percentage of genre sales** for each artist.
   * The percentage is calculated by dividing each genre's sales by the total sales and multiplying by 100.
   * Results are sorted by **genre sales** in descending order, and **genre** in alphabetical order.

This query provides a breakdown of each genre's performance and the percentage share of sales per artist, excluding the USA.

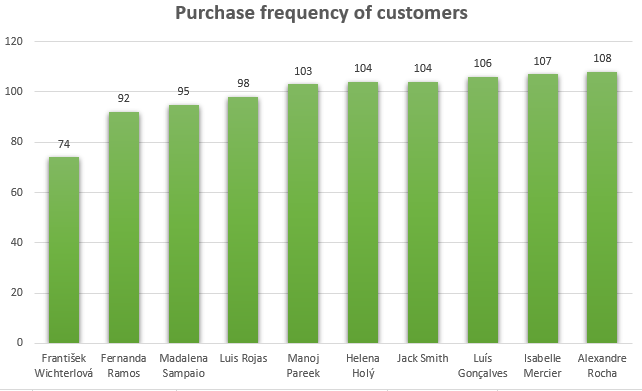
**3.Customer Purchasing Behaviour Analysis: How do the purchasing habits (frequency, basket size, spending amount) of long-term customers differ from those of new customers? What insights can these patterns provide about customer loyalty and retention strategies?**

**Solution**

* Frequent customers typically make more purchases, especially when they are highly loyal to a brand. This trend can be observed by analysing the average purchase frequency over a specific period. Consistent purchasing behaviour provides valuable insights for designing loyalty programs or special offers to encourage continued engagement.



* Long-term customers often exhibit larger basket sizes, reflecting their trust in the brand and willingness to try a wider range of products. Identifying frequently purchased product combinations can aid in creating personalized cross-selling and upselling strategies.
* For new customers, introductory discounts can be an effective way to drive initial spending. In contrast, targeted promotions for loyal, long-term customers can help sustain or even boost their average spending levels.



**Code**

WITH CustomerInvoiceDates AS (

SELECT

c.customer\_id,c.first\_name, c.last\_name,

MIN(DATE(i.invoice\_date)) AS first\_purchase\_date,

MAX(DATE(i.invoice\_date)) AS last\_purchase\_date,

COUNT(DISTINCT i.invoice\_id) AS purchase\_frequency,

ROUND(AVG(il.quantity),0) AS avg\_basket\_size,

ROUND(AVG(i.total),2) AS avg\_spending\_amount

FROM customer c

JOIN invoice i ON c.customer\_id = i.customer\_id

JOIN invoice\_line il ON i.invoice\_id = il.invoice\_id

GROUP BY 1,2,3

),

CustomerCategory AS (

SELECT

\*,

DATEDIFF(last\_purchase\_date,first\_purchase\_date) AS date\_diff,

CASE WHEN DATEDIFF(last\_purchase\_date,first\_purchase\_date) > 1000 THEN 'Long Term' ELSE 'New' END AS category\_type

FROM CustomerInvoiceDates

)

SELECT \* FROM CustomerCategory

ORDER BY customer\_id;

**Code Explanation**

**CustomerInvoiceDates CTE**:

* Calculates each customer's:
  + **First and last purchase dates**,
  + **Purchase frequency**,
  + **Average basket size** (number of items per order),
  + **Average spending amount**.

**CustomerCategory CTE**:

* Computes the duration of customer activity (date\_diff = days between first and last purchases).
* Categorizes customers as **'Long Term'** (more than 1000 days between purchases) or **'New'** (otherwise).

**Final Query**:

* Retrieves all customer details along with their calculated metrics and category type, sorted by customer ID.

**4.Product Affinity Analysis: Which music genres, artists, or albums are frequently purchased together by customers? How can this information guide product recommendations and cross-selling initiatives?**

**Solution**

**Common Genres:**

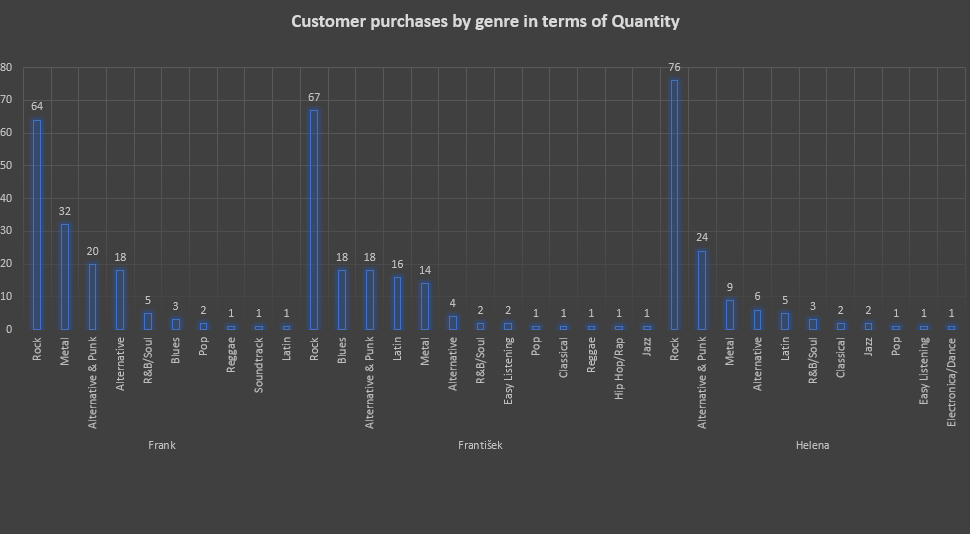
* Rock and Metal are popular among all three customers, reflecting a shared preference for these music styles.

**Product Recommendations:**

* Recommending related genres like Blues or Alternative, or introducing new artists within the Rock and Metal genres, could resonate with customers like Frank, who frequently purchase from these categories.

**Cross-Selling Strategies:**

* Leverage customers' strong interest in specific genres to encourage broader musical exploration by offering carefully curated recommendations of related genres.
* Since Rock, Metal, Alternative, and Punk music remain consistently popular, create personalized playlists for each customer featuring their favourite tracks.



**Code**

WITH ProductAffinityAnalysis AS (

SELECT

c.customer\_id,c.first\_name,c.last\_name,

a.name AS artist\_name,

g.name AS genre\_name,

SUM(il.quantity) AS total\_quantity,

SUM(i.total) AS total\_sales

-- ,RANK() OVER(ORDER BY SUM(i.total) DESC) AS sales\_rank

FROM invoice i

LEFT JOIN invoice\_line il ON i.invoice\_id = il.invoice\_id

LEFT JOIN track t ON il.track\_id = t.track\_id

LEFT JOIN album al ON t.album\_id = al.album\_id

LEFT JOIN artist a ON al.artist\_id = a.artist\_id

LEFT JOIN genre g ON t.genre\_id = g.genre\_id

LEFT JOIN customer c ON i.customer\_id = c.customer\_id

GROUP BY c.customer\_id,c.first\_name,c.last\_name,a.name,g.name

)

SELECT \* FROM ProductAffinityAnalysis

ORDER BY customer\_id, total\_quantity DESC;

**Code Explanation**

**ProductAffinityAnalysis CTE**:

* For each customer, the query aggregates their purchases by:
  + **Artist name** and **Genre name**,
  + Total quantity of items purchased (**total\_quantity**),
  + Total sales amount (**total\_sales**).
* The data is joined from various tables (invoices, invoice lines, tracks, albums, artists, and genres) to provide comprehensive information.

**Final Query**:

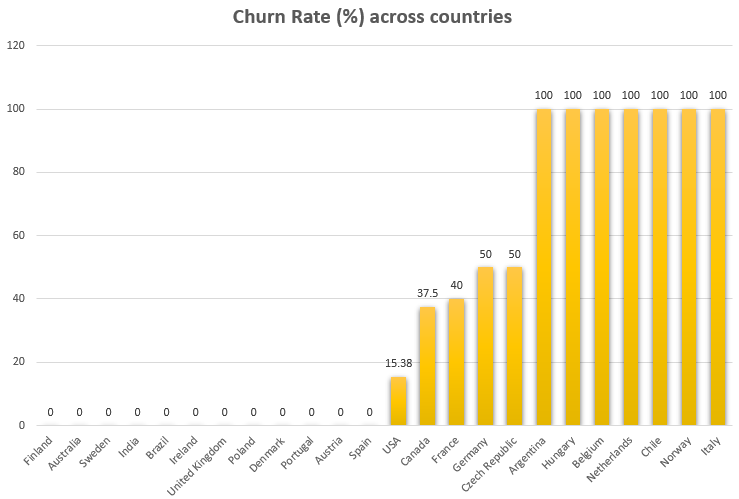
* Fetches all the aggregated details and orders the results by customer ID and total quantity in descending order, showing the most purchased items for each customer first.

**5.Regional Market Analysis: Do customer purchasing behaviours and churn rates vary across different geographic regions or store locations? How might these correlate with local demographic or economic factors?**

**Solution**

The regional market analysis, based on customer churn, reveals the following insights:

* A customer is classified as churned if they have not made any purchases within the last six months.
* Certain countries, such as Finland, Australia, India, and Spain, exhibit a 0% churn rate, indicating that customers in these regions are highly active and consistently make frequent purchases.



**Code**

WITH PreviousCustomerPurchases AS (

SELECT

c.country,

c.customer\_id,c.first\_name,c.last\_name,DATE(i.invoice\_date) AS invoice\_date,

LEAD(DATE(i.invoice\_date)) OVER(PARTITION BY c.customer\_id ORDER BY invoice\_date DESC) AS prev\_purchase

FROM customer c

JOIN invoice i ON c.customer\_id = i.customer\_id

),

PrevPurchaseRank AS (

SELECT

\*,ROW\_NUMBER() OVER(PARTITION BY customer\_id ORDER BY prev\_purchase DESC) AS prev\_purchase\_rn

FROM PreviousCustomerPurchases

),

PreviousPurchaseDate AS (

SELECT

\*,DATEDIFF(invoice\_date,prev\_purchase) AS days\_since\_last\_purchase

FROM PrevPurchaseRank

WHERE prev\_purchase\_rn = 1

AND DATEDIFF(invoice\_date,prev\_purchase) > 180

ORDER BY days\_since\_last\_purchase DESC

)

SELECT

c.country,

COUNT(pp.customer\_id) AS churned\_customers,

COUNT(c.customer\_id) AS total\_customers,

ROUND((COUNT(pp.customer\_id) \* 100) / COUNT(c.customer\_id), 2) AS churn\_rate

FROM customer c LEFT JOIN PreviousPurchaseDate pp ON c.customer\_id = pp.customer\_id

GROUP BY c.country;

**Code Explanation**

1. **PreviousCustomerPurchases CTE**:
   * Retrieves each customer's purchase history along with the date of their previous purchase using the LEAD function.
2. **PrevPurchaseRank CTE**:
   * Assigns a rank to each customer's previous purchase date in descending order.
3. **PreviousPurchaseDate CTE**:
   * Filters for customers whose last purchase was over 180 days ago (churned customers).
   * Calculates the number of days since their last purchase.
4. **Final Query**:
   * Aggregates data by country:
     + Counts churned customers.
     + Counts total customers.
     + Calculates the churn rate as a percentage of total customers.
   * Results are grouped by country to identify churn rates across regions.

**6.Customer Risk Profiling: Based on customer profiles (age, gender, location, purchase history), which customer segments are more likely to churn or pose a higher risk of reduced spending? What factors contribute to this risk?**

**Solution**

* Calculating churn rates by country helps identify geographical segments with consistently high churn. Such trends may point to issues with service, product offerings, or market fit in specific regions.
* A history of infrequent purchases often indicates low-engagement customers who are more likely to churn. Similarly, high-value customers with declining purchase frequency may signal waning interest, making them at risk of leaving.
* Customers with low purchase frequency or long gaps between purchases (such as the 180-day threshold in this analysis) are at greater risk of churn. These individuals may primarily respond to specific promotions or show limited brand loyalty.
* For regions with high churn rates, consider implementing localized marketing strategies and improving service offerings to enhance customer satisfaction and retention.

**Code**

WITH PreviousCustomerPurchases AS (

SELECT

c.country,

c.customer\_id,c.first\_name,c.last\_name,DATE(i.invoice\_date) AS invoice\_date,

LEAD(DATE(i.invoice\_date)) OVER(PARTITION BY c.customer\_id ORDER BY invoice\_date DESC) AS prev\_purchase

FROM customer c

JOIN invoice i ON c.customer\_id = i.customer\_id

),

PrevPurchaseRank AS (

SELECT

\*,ROW\_NUMBER() OVER(PARTITION BY customer\_id ORDER BY prev\_purchase DESC) AS prev\_purchase\_rn

FROM PreviousCustomerPurchases

),

PreviousPurchaseDate AS (

SELECT

\*,DATEDIFF(invoice\_date,prev\_purchase) AS days\_since\_last\_purchase

FROM PrevPurchaseRank

WHERE prev\_purchase\_rn = 1

AND DATEDIFF(invoice\_date,prev\_purchase) > 180

ORDER BY days\_since\_last\_purchase DESC

)

SELECT

c.country,

COUNT(pp.customer\_id) AS churned\_customers,

COUNT(c.customer\_id) AS total\_customers,

ROUND((COUNT(pp.customer\_id) \* 100) / COUNT(c.customer\_id), 2) AS churn\_rate

FROM customer c LEFT JOIN PreviousPurchaseDate pp ON c.customer\_id = pp.customer\_id

GROUP BY c.country

ORDER BY churn\_rate DESC, total\_customers ASC;

**Code Explanation**

**PreviousCustomerPurchases CTE**:

* Retrieves each customer's purchase history and the date of their previous purchase using the LEAD function.

**PrevPurchaseRank CTE**:

* Ranks previous purchase dates for each customer in descending order.

**PreviousPurchaseDate CTE**:

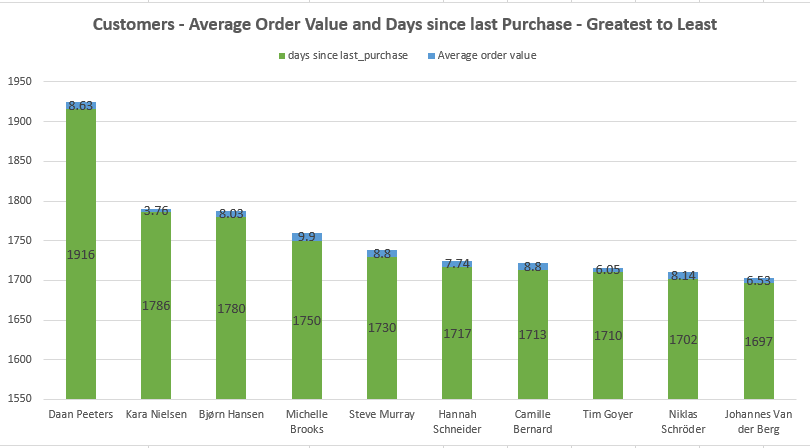
* Identifies churned customers, defined as those with more than 180 days since their last purchase.
* Calculates the number of days since their previous purchase

**Final Query**: Aggregates churn data by country, calculating the churn rate as a percentage of total customers, and orders results by churn rate and total customers.

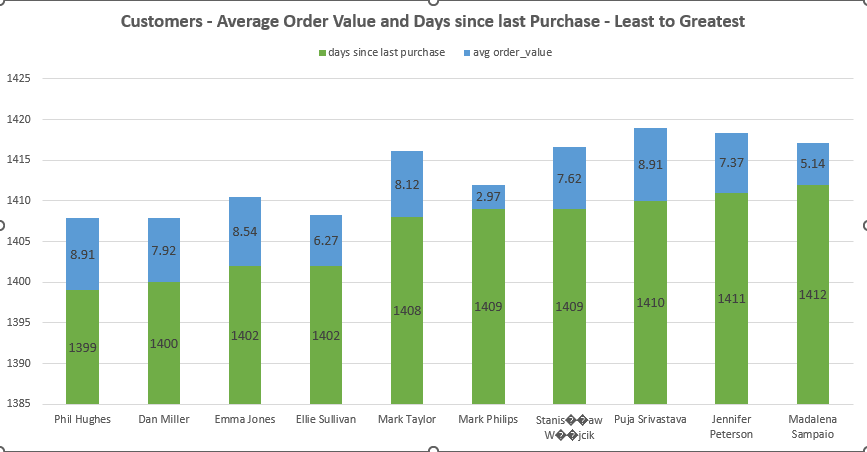
This provides insights into regional churn, helping to target retention efforts.

**7.Customer Lifetime Value Modelling: How can you leverage customer data (tenure, purchase history, engagement) to predict the lifetime value of different customer segments? This could inform targeted marketing and loyalty program strategies. Can you observe any common characteristics or purchase patterns among customers who have stopped purchasing?**

**Solution**



* We can determine each customer's total spend and purchase frequency by summing the totals from invoices and analyzing purchase dates.
* Customers with a high days\_since\_last\_purchase value are at a higher risk of churn if they haven’t made purchases recently.
* Customers with long-term loyalty but recent inactivity may benefit from targeted re-engagement emails offering discounts or exclusive deals to encourage them to return.



* Customers with infrequent purchases but a high average order value (AOV) may respond well to limited-time offers or exclusive products.
* Customers who have both a high AOV and frequent purchases are ideal candidates for loyalty programs.

**Code**

WITH CustomerTenure AS (

SELECT

c.customer\_id, CONCAT(c.first\_name,' ', c.last\_name) AS customer,

MIN(i.invoice\_date) AS first\_purchase\_date,

MAX(i.invoice\_date) AS last\_purchase\_date,

DATEDIFF(MAX(i.invoice\_date), MIN(i.invoice\_date)) AS tenure\_days,

COUNT(i.invoice\_id) AS purchase\_frequency,

SUM(i.total) AS total\_spent

FROM customer c

JOIN invoice i ON c.customer\_id = i.customer\_id

GROUP BY c.customer\_id

)

SELECT

customer\_id,

customer,

tenure\_days,

purchase\_frequency,

total\_spent,

ROUND(total\_spent / purchase\_frequency, 2) AS avg\_order\_value,

DATEDIFF(CURRENT\_DATE, last\_purchase\_date) AS days\_since\_last\_purchase

FROM CustomerTenure

ORDER BY days\_since\_last\_purchase DESC;

**Code Explanation**

**CustomerTenure CTE**:

* Joins customers (customer) with their invoices (invoice).
* Calculates:
  + First and last purchase dates (first\_purchase\_date, last\_purchase\_date).
  + Tenure in days (tenure\_days) between the first and last purchases.
  + Purchase frequency (total number of invoices).
  + Total amount spent (total\_spent).

**Main Query**:

* Uses the CustomerTenure CTE to calculate:
  + Average order value (avg\_order\_value) as total\_spent / purchase\_frequency.
  + Days since the last purchase (days\_since\_last\_purchase).
* Orders customers by recency, with the most inactive customers (longest days since last purchase) at the top.

**8.If data on promotional campaigns (discounts, events, email marketing) is available, how could you measure their impact on customer acquisition, retention, and overall sales?**

**Solution**

To evaluate the impact of promotional campaigns:

1. **Acquisition**: Examine the number of new customers acquired during each campaign period.
2. **Retention**: Monitor repeat purchases and churn rates among customers who participated in the promotion.
3. **Sales Impact**: Compare total sales, average order value, and purchase frequency during the campaigns against baseline periods.

Leveraging customer segmentation can identify which groups respond most effectively, enabling more precise targeting for future campaigns.

**9.How would you approach this problem, if the objective and subjective questions weren't given?**

**Solution:**

If no specific questions are provided, I would take a broad approach to analyse the dataset and uncover insights about customer behaviour, sales performance, and the impact of promotions. Here’s my step-by-step process:

1. **Understand Business Objectives**:  
   Start by clarifying the key goals, such as improving customer retention, increasing sales, or identifying high-value customer segments. This ensures the analysis aligns with business priorities.
2. **Data Exploration and Cleaning**:  
   Conduct data cleaning to address missing or inconsistent entries. Follow up with exploratory analysis to understand the structure, distribution, and trends within the dataset.
3. **Define Key Metrics and Segments**:  
   Identify critical KPIs like customer acquisition rate, churn rate, lifetime value (LTV), average order value, and campaign ROI. Segment customers by demographics, geographical location, and purchase behaviour to uncover patterns within each group.
4. **Analyse and Model Data**:
   * **Churn Analysis**: Investigate factors contributing to customer churn, such as recent purchase frequency or interaction with discounts.
   * **Campaign Effectiveness**: Evaluate the impact of promotions by comparing acquisition, retention, and sales metrics before, during, and after campaigns.
5. **Provide Strategy Recommendations**:  
   Based on the findings, deliver actionable recommendations to enhance customer retention, focus on high-value customers, and optimize marketing strategies for improved ROI.

**10.How can you alter the "Albums" table to add a new column named "ReleaseYear" of type INTEGER to store the release year of each album?**

**Solution:**

We can make use of the ALTER statement to add a new column to a table. The syntax is

as follows:

ALTER TABLE table\_name

ADD COLUMN column\_name datatype;

To add the column named “ReleaseYear” with INTEGER dataype to the album table, the

following query can be used.

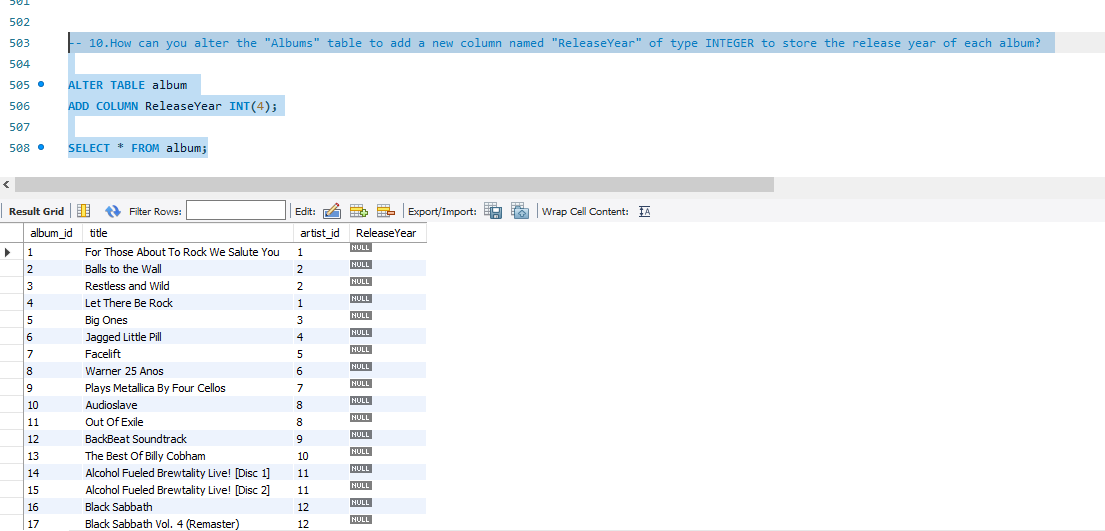
**Code**

ALTER TABLE album

ADD COLUMN ReleaseYear INT(4);

SELECT \* FROM album;

**Code screenshot**



**11.Chinook is interested in understanding the purchasing behaviour of customers based on their geographical location. They want to know the average total amount spent by customers from each country, along with the number of customers and the average number of tracks purchased per customer. Write a SQL query to provide this information.**

**Code**

SELECT

c.country,

ROUND(AVG(track\_count)) AS average\_tracks\_per\_customer,

SUM(i.total) AS total\_spent,

COUNT(DISTINCT c.customer\_id) AS no\_of\_customers,

ROUND(SUM(i.total)/ COUNT(DISTINCT c.customer\_id),2) AS avg\_total\_spent

FROM customer c

JOIN invoice i ON c.customer\_id = i.customer\_id

JOIN (

SELECT

invoice\_id,

COUNT(track\_id) AS track\_count

FROM invoice\_line

GROUP BY invoice\_id

) il ON i.invoice\_id = il.invoice\_id

GROUP BY c.country

ORDER BY avg\_total\_spent DESC;

**Code Explanation**

1. **Subquery (Invoice Line Analysis)**:
   * Counts the number of tracks (track\_count) in each invoice (invoice\_id).
2. **Main Query**:
   * Joins customer, invoice, and the subquery to calculate country-level metrics:
     + average\_tracks\_per\_customer: Average number of tracks purchased per customer.
     + total\_spent: Total revenue generated.
     + no\_of\_customers: Total distinct customers.
     + avg\_total\_spent: Average spending per customer.
3. **Grouping and Ordering**:
   * Groups data by country and sorts results by avg\_total\_spent in descending order.

**Code screenshot**

