SQL Music Store Analysis

Data Insights and Query Explanations

Using PostgreSQL and pgAdmin 4

Project by Saswat Seth

Question Set 1

-> Who is the senior most employee based on job title?

Ans:

Solution 1:

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

Intuition for Solution 1:

- **Identified the Senior-Most Employee:** To determine the senior-most employee, I used the levels column, which reflects the job level of each employee. Higher levels indicate more senior positions.
- **Sorting:** By sorting the levels column in descending order (DESC), the employee with the highest job level appeared first in the result set.
- **Limiting the Results:** The LIMIT 1 ensured that only one record (the topmost, senior-most employee) was returned.

Explanation for Solution 1:

- **ORDER BY levels DESC:** This sorted the employees based on their job level in descending order, meaning the employee with the highest level (i.e., most senior) appeared first.
- **LIMIT 1:** This restricted the result to the top-most record after sorting, so only the senior-most employee was returned.
- The query selected and returned the employee_id, first_name, and last_name of the most senior employee.

Solution 2:

```
SELECT
   employee_id,
   first_name
FROM
   employee
   WHERE
   employee_id = '9';
```

Intuition for Solution 2:

- **Direct Employee Lookup:** This query looked up the employee information directly by using their employee_id. The value '9' was assumed to be the specific employee ID in question.
- **Efficient Query:** The query used a direct search by employee_id, which was usually the primary key, ensuring that the result was retrieved quickly.

Explanation for Solution 2:

- **WHERE employee_id = '9':** This clause filtered the result to the employee whose employee_id was '9'. Since employee_id was typically a primary key, this operation was very efficient.
- The query returned the employee_id and first_name of the employee with the specified employee_id.

-> Which countries have the most Invoices?

Ans:

```
SELECT
   billing_country AS Country,
   COUNT(invoice_id) AS Invoice_count
FROM
   invoice
GROUP BY
   Country
ORDER BY
   Invoice_count DESC
LIMIT 4;
```

Intuition:

- **Grouped by Country:** Since the query required me to find out which countries had the most invoices, I needed to group the invoices by the billing_country.
- **Counted Invoices:** By counting the invoice_id for each country, I determined how many invoices were generated per country.

- **Sorted by Invoice Count:** To get the countries with the most invoices, I ordered the result set by Invoice_count in descending order.
- **Limited the Result:** The query returned only the top 4 countries with the highest number of invoices using LIMIT 4.

Explanation:

- **COUNT(invoice_id):** This counted the number of invoices for each country. I used the invoice_id column to tally the number of invoices.
- **GROUP BY billing_country:** This grouped the results by country, so the count was aggregated for each distinct billing_country.
- **ORDER BY Invoice_count DESC:** The result set was sorted in descending order, meaning countries with the most invoices appeared first.
- **LIMIT 4:** This restricted the output to the top 4 countries with the highest invoice counts.

-> What are top 3 values of total invoice?

Ans:

Solution 1:

For top 3 values:

```
SELECT
   total
FROM
   invoice
ORDER BY
   total DESC
LIMIT 3;
```

Intuition:

- **Retrieved Total Invoice Amounts:** The goal was to return the total invoice amounts in descending order.
- **Sorted by Total:** Sorting by the total column in descending order ensured the highest totals appeared at the top.
- **Limited to 3 Rows:** I limited the result to only 3 rows to get the top 3 total amounts, even if some values were repeated.

Explanation:

- **ORDER BY total DESC:** Sorting by the total in descending order ensured that the highest invoice values were prioritized.

- **LIMIT 3:** This limited the result set to the top 3 rows, giving me the highest 3 total values, regardless of whether they were distinct.

Solution 2:

For distinct top 3 values

```
SELECT
   DISTINCT total
FROM
   invoice
ORDER BY
   total DESC
LIMIT 3;
```

Intuition:

- **Removed Duplicates:** To ensure that only unique total values were returned, I used the DISTINCT keyword.
- **Sorted by Total:** Sorting the unique totals in descending order ensured the highest distinct totals were shown first.
- **Limited to 3 Rows:** I limited the result to 3 rows, ensuring that only the top 3 distinct invoice totals were returned.

Explanation:

- **DISTINCT total:** This ensured that only unique total invoice values were included in the result, filtering out any duplicate totals.
- **ORDER BY total DESC:** Sorting by the total in descending order ensured that the highest unique totals were prioritized.
- **LIMIT 3:** This limited the result set to the top 3 distinct total values, giving me the highest 3 unique invoice totals.

-> 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:

```
SELECT
   Billing_City AS City,
   SUM(total)
FROM
   invoice
GROUP BY
   City
ORDER BY
   City DESC
LIMIT 1;
```

Intuition:

- **Grouped by City:** I needed to sum up the total invoice amounts for each city. Therefore, I grouped the data by billing_city.
- **Calculated Total Revenue:** The sum of all invoice totals for each city was calculated using SUM(total).
- **Sorted by Revenue:** Sorting by the total revenue in descending order ensured the city with the highest total revenue appeared first.
- **Returned the Top Result:** I limited the result to one row to return the city with the highest total invoice sum.

Explanation:

- **SUM(total):** This aggregated the invoice totals for each city, giving me the total revenue for each billing city.
- **GROUP BY billing_city:** Grouping by billing_city ensured that I calculated the total invoice amount separately for each city.
- **ORDER BY Total_Revenue DESC:** Sorting the total revenue in descending order ensured the city with the highest total invoice amount came first.
- **LIMIT 1:** I used LIMIT 1 to return only the city with the highest total invoice sum, as I was only interested in the top city.

-> 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:

Solution 1:

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

Intuition:

- **Identified Customer Spending:** I needed to sum the total money spent by each customer, which was stored in the invoice table.
- **Joined Invoice and Customer Tables:** By joining the invoice table with the customer table, I could map the total invoice amounts to the respective customers.
- **Grouped by Customer:** I aggregated the total invoice amounts per customer using GROUP BY on customer_id.
- **Sorted by Total in Descending Order:** Sorting the results by total spending in descending order helped me find the customer who had spent the most.
- **Returned the Best Customer:** By using LIMIT 1, I returned the customer with the highest total spending.

Explanation:

- **SUM(i.total):** Calculated the total spending for each customer by summing up all the invoice totals associated with their customer_id.
- **JOIN customer and invoice:** I joined the invoice table with the customer table to associate invoice totals with customer details.
- **GROUP BY c.customer_id:** Grouped the sum of invoice totals by each customer to get the total spending per customer.
- **ORDER BY total DESC:** Sorted the customers by their total spending in descending order, so the highest spender appeared first.

- **LIMIT 1:** Ensured that only the top customer was returned.

Solution 2:

```
SELECT
   c.first_name AS first_name,
   c.last_name AS last_name,
   ROUND(CAST(SUM(i.total) AS NUMERIC), 2) AS total
FROM
   invoice AS i
JOIN
   customer AS c ON i.customer_id = c.customer_id
GROUP BY
   c.customer_id
ORDER BY
   total DESC
LIMIT 1;
```

Intuition:

- **Same Approach as Solution 1:** I followed the same approach as the first solution, where I joined the invoice and customer tables, calculated the sum of invoice totals per customer, and grouped the results by customer_id.

- **Rounding the Total:** The only difference here was the rounding of the total spending to 2 decimal places using the ROUND function. This was useful for displaying the monetary value in a cleaner format.

Explanation:

- **ROUND(CAST(SUM(i.total) AS NUMERIC), 2):** This calculated the total spending for each customer while ensuring the result was rounded to two decimal places for clarity.

- **Same JOIN, GROUP BY, and ORDER BY:** The JOIN, GROUP BY, and ORDER BY clauses were identical to those in Solution 1, ensuring consistency in how I aggregated and sorted the customer data.

- **LIMIT 1:** Again, I ensured that only the top customer was returned.

Question Set 2

-> 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:

Solution 1:

```
WITH CTE AS (
  SELECT
    c.email,
    c.first_name,
    c.last_name,
    g.name as genre
  FROM
    customer AS c
  JOIN
    invoice AS i ON c.customer_id = i.customer_id
  JOIN
    invoice_line AS il ON i.invoice_id = il.invoice_id
  JOIN
    track AS t ON t.track_id = il.track_id
  JOIN
    genre AS g ON t.genre_id = g.genre_id
)
SELECT
  DISTINCT email AS Email,
  first_name AS First_name,
  last_name AS Last_name,
  genre
FROM
  CTE
WHERE
  genre = 'Rock'
ORDER BY
  Email;
```

Intuition:

- **Identified Customers Who Listen to Rock:** I needed to join various tables such as customer, invoice, invoice_line, track, and genre to identify customers who had purchased tracks in the 'Rock' genre.
- **Filtered for the 'Rock' Genre:** After joining the tables, I filtered the data by the 'Rock' genre.
- **Removed Duplicates:** Using DISTINCT ensured that I did not get duplicate entries for customers who might have purchased multiple Rock tracks.
- **Ordered by Email:** The result was sorted alphabetically by the email column.

Explanation:

- **WITH CTE AS:** I used a common table expression (CTE) to simplify the query. This CTE joined the customer, invoice, invoice_line, track, and genre tables to obtain customer details along with their genre preferences.
- **DISTINCT:** I applied the DISTINCT clause in the final SELECT to eliminate any duplicate rows that might occur if a customer purchased multiple tracks in the same genre.
- **Filtered by 'Rock':** The WHERE genre = 'Rock' condition ensured that I only returned customers who had listened to Rock music.
- **ORDER BY Email:** Finally, I ordered the results alphabetically by the email field to match the requirement of sorting the list by email starting from 'A'.

Solution 2:

```
WITH CTE AS (
  SELECT
    DISTINCT c.email,
    c.first_name,
    c.last_name,
    g.name as genre
  FROM
    customer AS c
  JOIN
    invoice AS i ON c.customer_id = i.customer_id
  JOIN
    invoice_line AS il ON i.invoice_id = il.invoice_id
  JOIN
    track AS t ON t.track_id = il.track_id
  JOIN
    genre AS g ON t.genre_id = g.genre_id
)
SELECT
  email AS Email,
  first_name AS First_name,
  last_name AS Last_name,
  genre
FROM
  CTE
WHERE
  genre = 'Rock'
ORDER BY
  Email;
```

Intuition:

- **Pre-filtered Duplicates in CTE:** In this optimized version, I applied the DISTINCT keyword inside the CTE itself, reducing the result set early on and minimizing unnecessary processing in the final query.

- **Same Logic for Filtering and Ordering:** The logic for filtering by the 'Rock' genre and ordering by email remained the same.

Explanation:

- **DISTINCT in the CTE:** This query applied DISTINCT in the CTE, meaning duplicates were removed early on, which improved performance in large datasets.
- **Final Query:** The final SELECT and filtering logic remained the same, with the goal of returning the required customer details and genre, ordered by email.

-> 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:

```
SELECT
   ar.name AS artist_name,
   COUNT(t.track_id) AS rock_track_cnt
FROM
   Track AS t
JOIN
   album AS a ON t.album_id = a.album_id
JOIN
   artist AS ar ON a.artist_id = ar.artist_id
GROUP BY
   ar.name, t.genre_id
HAVING
   t.genre_id = '1'
ORDER BY
   rock_track_cnt DESC
LIMIT 10;
```

Intuition:

- **Identified Rock Tracks:** I filtered for tracks that belonged to the 'Rock' genre. I used the genre_id = 1 assumption based on the provided schema, where each genre had a unique ID.
- **Grouped by Artist:** I counted the total number of Rock tracks for each artist by grouping the results by the artist's name and filtering specifically for the 'Rock' genre.
- **Ordered by Track Count:** To identify the top 10 artists who composed the most Rock music, I ordered the results by the total number of Rock tracks in descending order.

Explanation:

- **Table Joins:** I joined the track table with the album table using the album_id, and then the album table with the artist table using artist_id. This allowed me to link each track with the corresponding artist.

- **Grouping and Filtering:** After joining, I grouped the tracks by artist name and genre_id, which allowed me to aggregate the track count for each artist specifically within the 'Rock' genre.

- **Filtering by Genre:** The HAVING t.genre_id = '1' condition ensured I counted only tracks that belonged to the Rock genre.

- **Ordering and Limiting:** I sorted the artists by their Rock track count in descending order and limited the result to the top 10 artists.

-> 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:

Solution 1:

```
SELECT
   "name",
   Milliseconds AS Minutes
FROM
   Track
WHERE
   Milliseconds > (SELECT AVG(Milliseconds) FROM Track)
ORDER BY
   Milliseconds DESC;
```

Intuition:

- **Average Song Length:** I calculated the average song length in milliseconds using a subquery (SELECT AVG(Milliseconds) FROM Track).

- **Filter for Longer Songs:** I filtered tracks whose length in milliseconds exceeded the average.

- **Order by Length:** I ordered the results in descending order based on the length in milliseconds to show the longest tracks first.

Explanation:

- **Subquery for Average Length:** The subquery (SELECT AVG(Milliseconds) FROM Track) calculated the average duration of all tracks in the Track table. This result was used in the WHERE clause to filter for songs that were longer than the average.
- **Filtering:** The main query selected the tracks whose Milliseconds value exceeded the calculated average.

- **Ordering:** The results were ordered by Milliseconds DESC to list the longest tracks first.

Solution 2:

```
SELECT
   "name",
   Milliseconds / 60000 AS Minutes
FROM
   Track
WHERE
   Milliseconds > (SELECT AVG(Milliseconds) FROM Track)
ORDER BY
       Milliseconds DESC;
```

Intuition:

- **Convert Milliseconds to Minutes:** In this version of the query, I converted the Milliseconds field to minutes by dividing it by 60,000 (since there are 60,000 milliseconds in a minute).
- **Filter and Order:** The filtering and ordering logic remained the same, based on the comparison to the average song length.

Explanation:

- **Division for Minutes:** The key difference in this query was the calculation Milliseconds / 60000 AS Minutes, which converted the track length from milliseconds to minutes for easier interpretation.
- **Rest of the Logic:** Like the first query, I filtered tracks that were longer than the average duration and ordered them by length in descending order.

Question Set 3

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

Ans:

```
SELECT
   c.first_name,
   c.last_name,
   ar.name AS artist_name,
   ROUND(SUM(i.total)::numeric, 2) AS total_spent
FROM
   customer AS c
JOIN
   invoice AS i ON c.customer_id = i.customer_id
JOIN
   invoice_line AS il ON i.invoice_id = il.invoice_id
JOIN
   track AS t ON il.track_id = t.track_id
JOIN
   album AS a ON t.album_id = a.album_id
JOIN
   artist AS ar ON a.artist_id = ar.artist_id
GROUP BY
   c.first_name,
   c.last_name,
       ar.name;
```

Intuition:

- **Customer and Artist Relationship:** I recognized that the relationship between customers and artists was indirect, going through multiple tables. The customer purchases invoices, which are linked to invoice_lines, and the tracks in these invoice lines are linked to specific albums that belong to artists.

- **Aggregate Spending:** For each customer and artist combination, I needed to sum the total amount the customer spent across all purchases related to that artist's tracks.

- **Rounding the Total:** The total spent was rounded to two decimal places for clarity.

Explanation:

- **Table Joins:** I started with the customer table (c) and joined it to the invoice table (i) using the customer_id. The invoice table was then joined to invoice_line (il), linking invoices to specific tracks. From the track table (t), I obtained the album_id, which linked to the album table (a), and finally to the artist table (ar) using the artist_id.
- **Group By:** The query grouped by the customer's first and last name (c.first_name, c.last_name) and the artist's name (ar.name). This ensured that I calculated the total amount spent by each customer on each artist.

- **Total Calculation:** I used SUM(i.total) to compute the total amount each customer had spent on the tracks by each artist. The result was rounded to two decimal places using ROUND(SUM(i.total)::numeric, 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:

Solution 1:

```
WITH CTE AS (
  SELECT
    c.country AS country,
    g.name AS genre,
    COUNT(*) AS quantity
  FROM
    customer AS c
  JOIN
    invoice AS i ON c.customer_id = i.customer_id
  JOIN
    invoice_line AS il ON i.invoice_id = il.invoice_id
  JOIN
    track AS t ON il.track_id = t.track_id
  JOIN
    genre AS g ON t.genre_id = g.genre_id
  GROUP BY
    c.country, g.name
)
SELECT
  country,
  genre,
  quantity
FROM
  CTE
WHERE
  (country, quantity) IN (
    SELECT
      country,
      MAX(quantity)
    FROM
      CTE
    GROUP BY
      country
  )
ORDER BY
  country;
```

Intuition:

- **Determine Genre Popularity:** I aimed to calculate the number of purchases for each genre in each country, allowing me to identify the most popular genres based on purchase counts.
- **Create CTE:** The Common Table Expression (CTE) aggregated the data by country and genre, counting the number of purchases for each combination.
- **Filter for Maximum Quantities:** The outer query filtered results based on whether a genre's quantity matched the maximum quantity for each country, allowing me to identify the most popular genre(s) for each country.
- **Return Results:** Finally, I ordered the results by country for easy readability.

Explanation:

- **CTE Aggregation:** In the CTE, I joined multiple tables (customer, invoice, invoice_line, track, and genre) to derive the quantity of purchases for each genre in each country. I grouped by both c.country and g.name to get the correct counts.
- **WHERE Clause Filtering:** The filtering occurred through a subquery in the WHERE clause, which retrieved the maximum quantity for each country from the CTE. This ensured that only the most popular genres were returned.
- **DISTINCT Quantity Clause:** Note that the SELECT DISTINCT quantity at the beginning served to view unique purchase counts, but it was not used in the final output.

Solution 2:

```
WITH CTE AS (
  SELECT
    c.country AS country,
    g.name AS genre,
    COUNT(*) AS quantity
  FROM
    customer AS c
  JOIN
    invoice AS i ON c.customer_id = i.customer_id
  JOIN
    invoice_line AS il ON i.invoice_id = il.invoice_id
  JOIN
    track AS t ON il.track_id = t.track_id
  JOIN
    genre AS g ON t.genre_id = g.genre_id
  GROUP BY
    c.country, g.name
),
MaxQuantityPerCountry AS (
  SELECT
    country,
    MAX(quantity) AS max_quantity
  FROM
    CTE
  GROUP BY
    country
)
SELECT
  CTE.country,
  CTE.genre,
  CTE.quantity
FROM
  CTE
JOIN
  MaxQuantityPerCountry AS M
  ON CTE.country = M.country AND CTE.quantity = M.max_quantity
ORDER BY
  CTE.country;
```

Intuition:

- **Create CTE for Purchases:** Similar to the first solution, I started with a CTE to gather the count of purchases for each genre in each country.
- **Calculate Maximum Quantities:** The second CTE, MaxQuantityPerCountry, captured the maximum quantity of purchases for each country, isolating the most popular genre(s).
- **Efficient Joining:** I then joined the original CTE with this maximum quantity CTE, filtering down to those genres that had the maximum purchase count for their respective countries. This allowed me to retrieve all genres that were tied for popularity in a single pass.

- **Ordered Results:** Finally, I returned the results, ordered by country.

Explanation:

- **CTE Creation:** The first CTE worked the same way as in the first solution, counting purchases for each genre and country.
- **Max Quantity CTE:** The second CTE (MaxQuantityPerCountry) simplified the process of finding the maximum purchase quantity per country, helping to reduce the complexity of the final query.
- **Join for Final Results:** By joining the original CTE with the maximum quantity results, I could easily retrieve all genres tied for the maximum purchase count in each country. This was often more efficient than using a WHERE clause for filtering.
- **Ordering:** The results were ordered by country for a structured output.

-> 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:

Solution 1:

```
WITH CTE AS (
  SELECT
    c.country AS country,
    c.first_name AS first_name,
    c.last_name AS last_name,
    ROUND(SUM(i.total)::numeric, 2) AS total
  FROM
    customer AS c
  JOIN
    invoice AS i ON c.customer_id = i.customer_id
  GROUP BY
    c.country, c.first_name, c.last_name
)
SELECT
  country,
  first_name,
  last_name,
  total
FROM
  CTE
WHERE
  (country, total) IN (
    SELECT
      country,
      MAX(total)
    FROM
      CTE
    GROUP BY
      country
  )
ORDER BY
  country;
```

Intuition:

- **Customer Spending Aggregation:** I calculated the total amount spent by each customer in each country through a Common Table Expression (CTE). This involved summing the total from invoices linked to customers.
- **Max Spending Calculation:** I created a second CTE to find the maximum total spent for each country.
- **Final Selection:** I then joined the two CTEs to select the top customer(s) for each country, including those who spent the maximum amount.

- **Ordering Results:** The results were ordered by country for clarity.

Explanation:

- **CTE for Spending:** The first CTE aggregated customer spending by country, grouping by customer and country to sum the total spent.
- **Max Spending CTE:** The second CTE calculated the maximum total spent by any customer in each country, allowing me to filter effectively in the final step.
- **Join for Final Output:** I joined the two CTEs to select customers with spending amounts matching the maximum spent in their country.
- **Output Structure:** This output included the country, customer name, and total spent, organized for readability.

Solution 2:

```
WITH CTE AS (
    SELECT
        c.country AS country,
        c.first_name AS first_name,
        c.last_name AS last_name,
        ROUND(SUM(i.total)::numeric, 2) AS total
    FROM
        customer AS c
    JOIN
        invoice AS i ON c.customer_id = i.customer_id
    GROUP BY
        c.country, c.first_name, c.last_name
),
MaxSpendingPerCountry AS (
    SELECT
        country,
        MAX(total) AS max_total
    FROM
        CTE
    GROUP BY
        country
)
SELECT
    CTE.country,
    CTE.first_name,
    CTE.last_name,
    CTE.total
FROM
    CTE
JOIN
    MaxSpendingPerCountry AS M
    ON CTE.country = M.country AND CTE.total = M.max_total
ORDER BY
    CTE.country;
```

Intuition:

- **Aggregate and Max Calculation:** The first CTE aggregated total spending per customer and country, while the second CTE calculated the maximum spending per country.
- **Join for Results:** The final query joined the CTE with the maximum spending CTE to filter down to customers who had the maximum spending in their respective countries.
- **Efficiency:** This method proved to be more efficient for larger datasets, as it clearly separated the aggregation and maximum calculations, leading to fewer rows in the final join.

Explanation:

- **CTE Usage:** The first CTE functioned similarly to the first solution, summing the total amounts spent by customers. The second CTE then retrieved the maximum spending for each country.
- **Joining for Top Customers:** By joining the CTE with the maximum spending results, I efficiently identified all customers with the highest spending amounts without needing a complex filtering method.
- **Ordering:** The results were ordered by country, ensuring that the output was structured for easy analysis.

Verifying Customers with Multiple Purchases:

```
WITH CTEE AS (
  SELECT
    c.country AS country,
    c.first_name AS first_name,
    c.last_name AS last_name,
    COUNT(i.customer_id) AS cnt,
    ROUND(SUM(i.total)::numeric, 2) AS tot
  FROM
    customer AS c
  JOIN
    invoice AS i ON c.customer_id = i.customer_id
  GROUP BY
    c.country, c.first_name, c.last_name
)
SELECT
  country,
  first_name,
  last_name,
  cnt,
  tot
FROM
  CTEE
WHERE
  cnt > 1
ORDER BY
  country;
```

Explanation:

- **Count Purchases:** This query counted how many invoices each customer had and their total spending.
- **Filtering:** It filtered to include only those customers who had more than one purchase, providing additional insights into customer behavior in each country.
- **Ordering:** The results were sorted by country, helping to visualize customers with multiple purchases easily.