Analysis of Music Store Database with PostgreSQL

This presentation explores the analysis of a music store database using PostgreSQL. We'll cover topics from database design and SQL querying to data visualization and performance optimization.

by Madhur Dhavre

Introduction to the Music Store Database

Overview

This database stores information about a music store, including customer data, inventory, sales, and financial transactions. This database provides valuable insights into the business operations and customer behavior.

Key Entities

The database consists of various tables representing key entities such as customers, products, orders, employees, and suppliers.

Data Modeling and Schema Design

Relational Model

The database is designed using a relational model, ensuring data integrity and efficient query performance.

Normalization

Tables are normalized to eliminate data redundancy and ensure consistency.

Data Types

Appropriate data types are chosen for each column to ensure data accuracy and efficiency.

Connecting to the Database with PostgreSQL

PostgreSQL Client

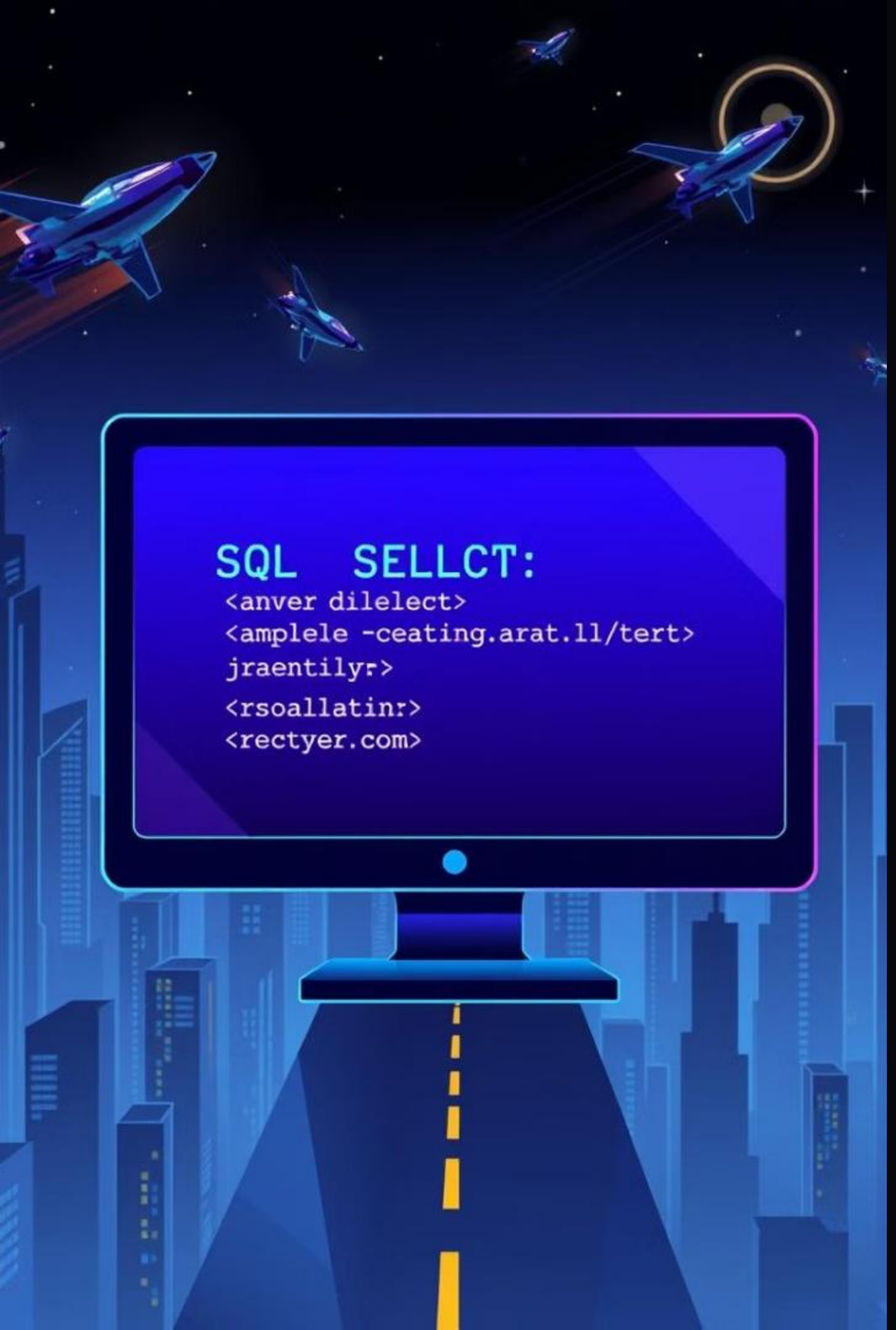
Various client tools, such as pgAdmin, can be used to connect to the database.

SQL Commands

Use SQL commands to connect, create, and manage database objects.

Security Measures

Implement security measures like user authentication and access control.



Querying the Database: Basic SQL Statements

1. **SELECT**
   Retrieve data from one or more tables.
2. **FROM**
   Specify the table(s) to query.
3. **WHERE**
   Filter the results based on conditions.
4. **ORDER BY**
   Sort the results in ascending or descending order.

Who is the senior most employee based on job title?

```
select * from employee
order by levels desc
limit 1
```

The query identifies and displays the **three largest invoice totals** in the `invoice` table.

	employee_id [PK] character varying (50)	last_name character (50)	first_name character (50)	title character varying (50)
1	9	Madan ...	Mohan ...	Senior General Manager

Key Features of the Query:

1. Column Selection:
   - Retrieves the total column from the invoice table, which represents the total amount for each invoice.
2. Sorting:
   - Orders the rows in descending order (DESC) based on the total column, so the highest invoice amounts appear first.
3. Limiting Results:
   - Uses LIMIT 3 to restrict the output to the top 3 invoices with the highest totals.

Which countries have the most Invoices?

Key Features of the Query:

1. Count Invoices:
   - COUNT(*) calculates the total number of invoices for each billing_country.
2. Alias:
   - billing_country AS c assigns the alias c to the billing_country column for easier reference.
3. Grouping:
   - GROUP BY billing_country groups the data by the billing_country, aggregating the count for each country.
4. Sorting:
   - ORDER BY c DESC sorts the results alphabetically in descending order based on the country name (c).
5. Limiting Results:
   - LIMIT 1 restricts the output to the top entry after sorting.

```
SELECT COUNT(*),billing_country as c
FROM invoice
GROUP BY billing_country
order by c desc limit 1
```

The query determines the alphabetically last country (billing_country) and the total number of invoices associated with it.

	count bigint	c character varying (30)
1	131	USA

What are top 3 values of total invoice?

```
select total
from invoice
order by total
desc limit 3
```

The query identifies and displays the **three largest invoice totals** in the invoice table, sorted from highest to lowest.

	total double precision
1	23.759999999999998
2	19.8
3	19.8

Key Features of the Query:

1. Column Selection:
   - Retrieves the total column from the invoice table, representing invoice amounts.
2. Sorting:
   - ORDER BY total DESC: Sorts the invoice totals in descending order, showing the highest values first.
3. Limiting Results:
   - LIMIT 3: Restricts the output to the top 3 invoices with the largest totals.

Optimizing Database Performance

1

Query Optimization

Analyze and improve query performance.

2

Indexing

Create indexes to speed up data retrieval.

3

Database Tuning

Adjust database settings for optimal performance.

Write a query that returns one city that has the highest sum of invoice totals.

```
select sum(total) as invoice_total,billing_city
from invoice
group by billing_city
order by invoice_total desc limit 1
```

The query identifies the billing city with the highest total invoice amount and displays that city along with its total invoice value.

	invoice_total double precision	billing_city character varying (30)
1	273.24000000000007	Prague

Key Features of the Query:

1. Aggregation:
   - SUM(total) calculates the total invoice amount (invoice_total) for each billing_city.
2. Grouping:
   - GROUP BY billing_city groups the invoices by billing_city, aggregating the total for each city.
3. Sorting:
   - ORDER BY invoice_total DESC sorts the cities by their total invoice amounts in descending order, prioritizing the city with the highest total.
4. Limiting Results:
   - LIMIT 1 restricts the output to the city with the highest total invoice amount.

Write a query that returns the person who has spent the most money.

Key Features of the Query:

1. Column Selection:
   - Retrieves customer_id, first_name, last_name, and the total spending (total_spending) for each customer.
2. Join Operation:
   - Joins the customer table with the invoice table using the customer_id to link customers with their invoices.
3. Aggregation:
   - SUM(total) calculates the total spending for each customer across all their invoices.
4. Grouping:
   - GROUP BY customer.customer_id groups data by customer_id to calculate spending for each individual customer.
5. Sorting:
   - ORDER BY total_spending DESC sorts customers by their total spending in descending order, highlighting the top spender.
6. Limiting Results:
   - LIMIT 1 restricts the output to the customer with the highest total spending.

```sql
select customer.customer_id,first_name,last_name,sum(total) as total_spending
from customer join invoice
on customer.customer_id = invoice.customer_id
GROUP BY customer.customer_id
ORDER BY total_spending DESC
limit 1
```

The query identifies the **highest-spending customer**, displaying their ID, name, and total spending.

	customer_id [PK] integer	first_name character (50)	last_name character (50)	total_spending double precision
1	5	R	Madhav	144.54000000000002

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

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

	customer_id [PK] integer	first_name character (50)	last_name character (50)	total_spending double precision
1	5	R ...	Madhav ...	144.54000000000002

The key features :

1. Distinct Selection: It selects unique combinations of email, first_name, and last_name from the customer table.
2. Joins:
   - Joins the customer table with the invoice table using custome
   - Joins the invoice table with the invoice_line table using invoic
3. Subquery: A subquery retrieves track_id values from the track table, where the genre is "Rock" (using the genre table).
4. Filtering: The main query filters the invoice_line records to include only those that have a track_id matching the subquery's results.
5. Ordering: Results are ordered by email in ascending order.

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.

Key Features of the Query:

1. Join Operations:
   - Joins the track table with the album table on album_id.
   - Joins the album table with the artist table on artist_id.
   - Joins the track table with the genre table on genre_id.
2. Filtering: The query filters for tracks that belong to the "Rock" genre.
3. Grouping: The results are grouped by artist.artist_id, so each artist appears once.
4. Aggregation: It counts the number of songs for each artist using COUNT(artist.artist_id) and aliases it as number_of_songs.
5. Ordering: The results are ordered by number_of_songs in descending order, showing the artists with the most "Rock" songs at the top.
6. Limiting: The query returns only the top 10 artists based on the number of "Rock" songs.

This query aims to find the top 10 artists with the most "Rock" genre tracks.

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

	artist_id [PK] character varying (50)	name character varying (120)	number_of_songs bigint
1	22	Led Zeppelin	114
2	150	U2	112
3	58	Deep Purple	92
4	90	Iron Maiden	81
5	118	Pearl Jam	54
6	152	Van Halen	52
7	51	Queen	45
8	142	The Rolling Stones	41
9	76	Creedence Clearwater Revival	40
10	52	Kiss	35

Advanced SQL Techniques: Joins, Aggregations, and Subqueries

1

Joins

Combine data from multiple tables based on relationships.

2

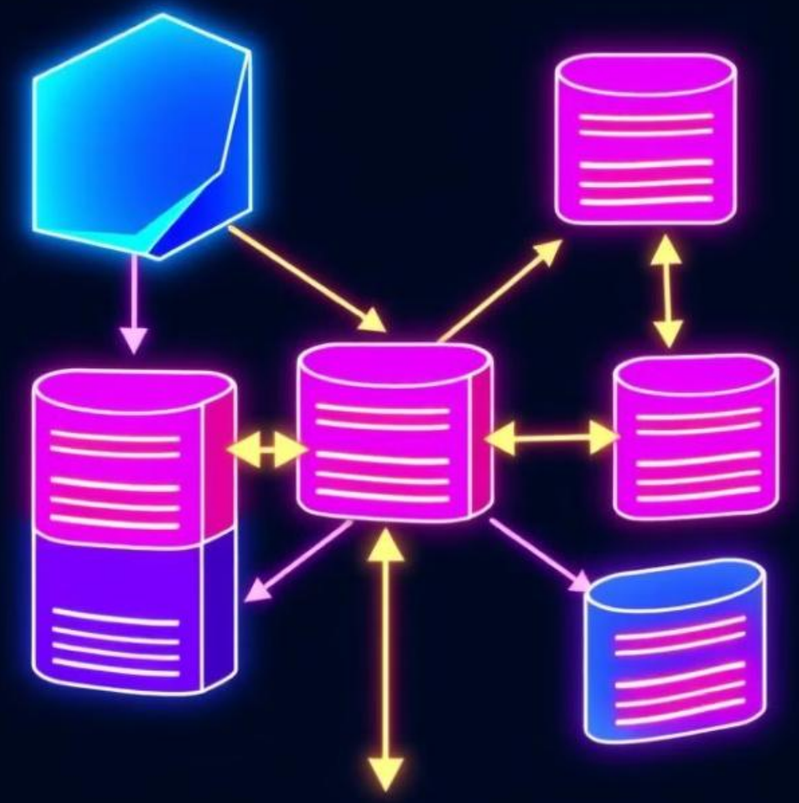
Aggregations

Perform calculations like SUM, AVG, COUNT, and MAX.

3

Subqueries

Nested queries to filter results based on conditions from other queries.



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

```sql
select name,milliseconds
from track
where milliseconds >
   (select avg(milliseconds) as avg_lenth
from track)
order by milliseconds desc;
```

The query retrieves tracks longer than the average duration, ordered by length in descending order.

	name character varying (150)	milliseconds integer
1	Occupation / Precipice	5286953
2	Through a Looking Glass	5088838
3	Greetings from Earth, Pt. 1	2960293
4	The Man With Nine Lives	2956998
5	Battlestar Galactica, Pt. 2	2956081

The key features of this SQL query are:

1. Main Query:
   - Selects the name (track name) and milliseconds (duration) from the track table.
   - Filters tracks to include only those where the milliseconds (duration) is greater than the average track duration in the track table.
2. Subquery:
   - The subquery calculates the average duration (avg(milliseconds)) of all tracks in the track table and aliases it as avg_length.
3. Filtering:
   - The main query includes only those tracks where the duration (milliseconds) is greater than the average calculated by the subquery.
4. Ordering:
   - The results are ordered by milliseconds in descending order, showing the longest tracks first.

Find how much amount spent by each customer on artists?

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

Key Features:

1. **CTE for Best-Selling Artist**:
   - Extracts the top-selling artist using `SUM` and `LIMIT 1`.
   - Simplifies the main query by pre-calculating the artist's information.
2. **Join Operations**:
   - Links customer data (`customer` table) with invoices, invoice lines, and track inform
   - Ensures only tracks from the best-selling artist are included by joining with the `best_selling_artist` CTE.
3. **Grouping and Sorting**:
   - Groups results by customer to calculate their total spending on the artist' s track
   - Orders by the highest spending (`amount_spent`) to highlight the top customers.

The query outputs:

1. **Customer Details**: `customer_id`, `first_name`, `last_name`.
2. **Artist Name**: The best-selling artist's name.
3. **Total Spending**: The total amount the customer spent on tracks by the best-selling arti

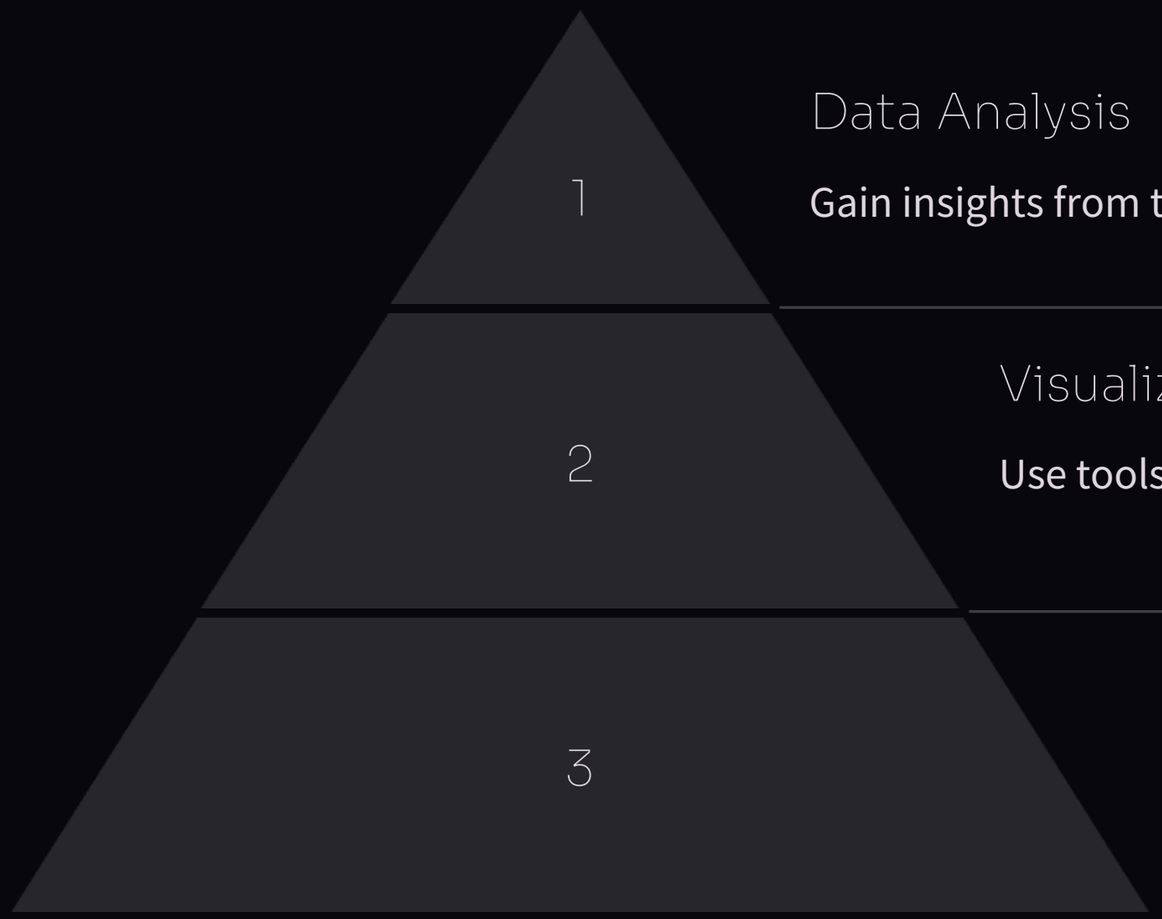
Results are sorted by the total spending, showing the top-spending customers first.

Method; Common Table Expression (CTE):

```sql
WITH best_selling_artist AS (
    SELECT artist.artist_id AS artist_id, artist.name AS artist_name,
    SUM(invoice_line.unit_price*invoice_line.quantity) AS total_sales
    FROM invoice_line
    JOIN track ON track.track_id = invoice_line.track_id
    JOIN album ON album.album_id = track.album_id
    JOIN artist ON artist.artist_id = album.artist_id
    GROUP BY 1
    ORDER BY 3 DESC
    LIMIT 1
)
SELECT c.customer_id, c.first_name, c.last_name, bsa.artist_name,
SUM(il.unit_price*il.quantity) AS amount_spent
FROM invoice i
JOIN customer c ON c.customer_id = i.customer_id
JOIN invoice_line il ON il.invoice_id = i.invoice_id
JOIN track t ON t.track_id = il.track_id
JOIN album alb ON alb.album_id = t.album_id
JOIN best_selling_artist bsa ON bsa.artist_id = alb.artist_id
GROUP BY 1,2,3,4
ORDER BY 5 DESC;
```

	customer_id integer	first_name character (50)	last_name character (50)	artist_name character varying (120)	amount_spent double precision
1	46	Hugh	O'Reilly	Queen	27.719999999999985
2	38	Niklas	Schröder	Queen	18.81
3	3	François	Tremblay	Queen	17.82
4	34	João	Fernandes	Queen	16.830000000000002
5	53	Phil	Hughes	Queen	11.88

Data Visualization and Reporting



Data Analysis

Gain insights from the data.

Visualization Tools

Use tools like Tableau, Power BI, or Python libraries to visualize data.

Reports

Create reports to communicate findings to stakeholders.

We want to find out the top 5 most popular music Genre for each country.

Write a query that returns each country along with the top Genre. For countries where the maximum number of purchases is shared return all Genres.

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

The CTE "popular_genre" calculates the popularity of genres in each country by linking purchases to genres and countries. It aggregates data using COUNT (invoice_line.quantity) and assigns a ranking (RowNo) to genres within each country based on the number of purchases. The results are grouped by customer. country, genre.name and genre.genre_id to calculate genre popularity for each country. The main query filters the CTE to find the most popular genre for each country using the WHERE RowNo <= 1 condition.

	purchases bigint	country character varying (50)	name character varying (120)	genre_id character varying (50)	rowno bigint
1	17	Argentina	Alternative & Punk	4	1
2	34	Australia	Rock	1	1
3	40	Austria	Rock	1	1
4	26	Belgium	Rock	1	1
5	205	Brazil	Rock	1	1

The query outputs:

1. The total number of purchases (purchases) for the most popular genre.
2. The country (country) where the genre is most popular.
3. The name and ID of the genre (name, genre_id).

Key Features:

- ROW_NUMBER(): Used to rank genres within each country.
- PARTITION BY: Ensures the ranking is reset for each country.
- WHERE RowNo <= 1: Ensures only the top-ranked genre for each country is in

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.

Method :1 RECURSIVE

```sql
WITH RECURSIVE
    customter_with_country AS (
        SELECT customer.customer_id,first_name,last_name,billing_country,SUM(total) AS total_spending
        FROM invoice
        JOIN customer ON customer.customer_id = invoice.customer_id
        GROUP BY 1,2,3,4
        ORDER BY 2,3 DESC),

    country_max_spending AS(
        SELECT billing_country,MAX(total_spending) AS max_spending
        FROM customter_with_country
        GROUP BY billing_country)

SELECT cc.billing_country, cc.total_spending, cc.first_name, cc.last_name, cc.customer_id
FROM customter_with_country cc
JOIN country_max_spending ms
ON cc.billing_country = ms.billing_country
WHERE cc.total_spending = ms.max_spending
ORDER BY 1;
```

The CTE "customter_with_country" calculates the total spending of each customer grouped by their country, using customer and invoice tables. It then groups results by customer and country, and orders the results by their names in descending order. The CTE "country_max_spending" finds the maximum total spending for each country, selecting the maximum spending value from the customter_with_country CTE.

The main query combines the two CTEs to match each country's highest spending amount with the corresponding customer, filtering results using WHERE clause. The final output is ordered by billing_country.

	billing_country character varying (30)	total_spending double precision	first_name character (50)	last_name character (50)	customer_id integer
1	Argentina	39.6	Diego	Gutiérrez	56
2	Australia	81.18	Mark	Taylor	55
3	Austria	69.3	Astrid	Gruber	7
4	Belgium	60.38999999999999	Daan	Peeters	8
5	Brazil	108.89999999999998	Luís	Gonçalves	1
6	Canada	99.99	François	Tremblay	3
7	Chile	97.02000000000001	Luis	Rojas	57
8	Czech Republic	144.54000000000002	R	Madhav	5
9	Denmark	37.61999999999999	Kara	Nielsen	9
10	Finland	79.2	Terhi	Hämäläinen	44

The query outputs:

1. The billing_country of the customer.
2. Their total_spending, first_name, last_name, and customer_id.

Each row represents the highest-spending customer in each country sorted alphabetically by the country name.

Key Features:

- MAX(): Finds the highest spending amount per country.
- Joins: Matches the maximum spending amount with the corresponding customer.
- CTEs: Simplify the query structure and make intermediate steps clearer.

Method 2 ; Common Table Expression (CTE):

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

The Common Table Expression (CTE) is used to calculate the total spending of each customer grouped by their country. It joins customer and invoice tables, aggregates data using SUM(total), assigns a ranking (RowNo) to customers within each country, and groups results by customer_id, first_name, last_name, and billing_country.

The main query filters the CTE to find the top spender in each country, using the WHERE RowNo <= 1 condition to include only the highest spending customer in the result.

	customer_id integer	first_name character (50)	last_name character (50)	billing_country character varying (30)	total_spending double precision	rowno bigint
1	56	Diego	Gutiérrez	Argentina	39.6	1
2	55	Mark	Taylor	Australia	81.18	1
3	7	Astrid	Gruber	Austria	69.3	1
4	8	Daan	Peeters	Belgium	60.38999999999999	1
5	1	Luís	Gonçalves	Brazil	108.89999999999998	1
6	3	François	Tremblay	Canada	99.99	1
7	57	Luis	Rojas	Chile	97.02000000000001	1
8	5	R	Madhav	Czech Republic	144.54000000000002	1
9	9	Kara	Nielsen	Denmark	37.61999999999999	1
10	44	Terhi	Hämäläinen	Finland	79.2	1

The query outputs:

1. The customer_id, first_name, and last_name of the highest spender in each country.
2. Their billing_country and total spending (total_spending).

Key Features:

- ROW_NUMBER(): Ranks customers based on spending within each cour
- PARTITION BY: Ensures the ranking is specific to each country.
- WHERE RowNo <= 1: Selects only the highest spender for each country.

Conclusion and Next Steps

1

Insights

The analysis provides valuable insights into customer behavior, product performance, and business trends.

2

Decision Making

The information can inform strategic decisions for the music store.

3

Future Analysis

Further analysis can be conducted to explore new opportunities and improve operations.