**Question Set 1 - Easy**

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

SELECT title, last\_name, first\_name

FROM employee

ORDER BY levels DESC

LIMIT 1

**Q2: Which countries have the most Invoices?**

SELECT COUNT(\*) AS c, billing\_country

FROM invoice

GROUP BY billing\_country

ORDER BY c DESC

**Q3: What are top 3 values of total invoice?**

SELECT total

FROM invoice

ORDER BY total DESC

**Q4: Which city has the best customers? We would like to throw a promotional Music Festival in the city we made the most money. Write a query that returns one city that has the highest sum of invoice totals. Return both the city name & sum of all invoice totals**

SELECT billing\_city,SUM(total) AS InvoiceTotal

FROM invoice

GROUP BY billing\_city

ORDER BY InvoiceTotal DESC

LIMIT 1;

**Question 5: 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.**

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;

**Question set 2- Moderate\_level**

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

SELECT DISTINCT email AS Email,first\_name AS FirstName, last\_name AS LastName, genre.name AS Name

FROM customer

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

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

JOIN track ON track.track\_id = invoice\_line.track\_id

JOIN genre ON genre.genre\_id = track.genre\_id

WHERE genre.name LIKE 'Rock'

ORDER BY email;

We use multiple **JOINs** in this query because the required data is spread across different tables. Let's break it down:

**Why So Many JOINs?**

1. **Customer Table** (customer)
   * Has customer details like email, first\_name, and last\_name.
   * But it **doesn’t** have purchase history.
2. **Invoice Table** (invoice)
   * Links customers to their purchases (customer\_id).
   * But it **doesn’t** have track details.
3. **Invoice Line Table** (invoice\_line)
   * Breaks down each invoice into individual tracks (track\_id).
   * But it **doesn’t** know the genre.
4. **Track Table** (track)
   * Contains track\_id and links each track to its **genre** (genre\_id).
   * But it **doesn’t** have genre names.
5. **Genre Table** (genre)
   * Has genre\_id and name (like 'Rock').

**Final Reason**

Since a **customer** → buys an **invoice** → contains multiple **tracks** → each **track** has a **genre**, we need all these joins to connect the dots and find customers who bought **Rock** music.

Without these joins, the database wouldn’t know how customers are linked to the genre of the tracks they purchased! 🎶

1. **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**

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, artist.name -- Add artist.name here

ORDER BY number\_of\_songs DESC

LIMIT 10;

This query finds the **top 10 artists** with the most **Rock** songs in the database.

**Step-by-Step Breakdown**

1. **Selecting Data:**
   * artist.artist\_id, artist.name → Fetches the artist's ID and name.
   * COUNT(artist.artist\_id) AS number\_of\_songs → Counts the number of rock songs per artist.
2. **Joining Tables:**
   * track → Has song details, including album info.
   * album → Connects each song to an artist via album\_id.
   * artist → Provides artist names.
   * genre → Helps filter only **Rock** songs.
3. **Filtering:**
   * WHERE genre.name LIKE 'Rock' → Includes only songs in the Rock genre.
4. **Grouping:**
   * GROUP BY artist.artist\_id, artist.name → Groups the data by artist to count their songs correctly.
5. **Sorting & Limiting:**
   * ORDER BY number\_of\_songs DESC → Sorts artists by the number of Rock songs (most to least).
   * LIMIT 10 → Shows only the **top 10 artists** with the most Rock songs.

**Why So Many Joins?**

* **Songs are in track**, but they don’t directly store artist names.
* **Each song belongs to an album**, which has an **artist\_id**.
* **Artists are in the artist table** but need to be linked through albums.
* **Genres are stored separately**, so we join to filter only Rock songs.

1. **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.**

SELECT name, milliseconds

FROM track

WHERE milliseconds > (

SELECT AVG (milliseconds) AS avg\_track\_length

FROM track)

ORDER BY milliseconds DESC;

**What Does This Query Do?**

This query **finds songs that are longer than the average track length** and sorts them from longest to shortest.

**Step-by-Step Breakdown:**

1. **Selecting Data:**
   * name, milliseconds → Retrieves the **track name** and its **length** (in milliseconds).
2. **Filtering with a Subquery:**
   * WHERE milliseconds > (
     + **Subquery**: SELECT AVG(milliseconds) FROM track → Calculates the **average track length**.
     + Filters only **tracks longer than the average length**.
3. **Sorting:**
   * ORDER BY milliseconds DESC → Sorts the results from **longest to shortest** track.

**Why Use a Subquery?**

* We need to **compare each track’s length** to the **average track length**.
* Since **AVG(milliseconds)** returns a single value, a subquery is the best way to get and use it in filtering.

**Use Case:**

* Find **exceptionally long songs**.
* Identify **epic-length tracks** for playlists.
* Analyze **track length trends** in a music database. 🎵📊

**Question Set 3 - Advance**

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

SELECT c.customer\_id,

c.first\_name,

c.last\_name,

ar.name AS 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 a ON a.album\_id = t.album\_id

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

WHERE ar.artist\_id = (

SELECT artist.artist\_id

FROM invoice\_line il

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

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

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

GROUP BY artist.artist\_id

ORDER BY SUM(il.unit\_price \* il.quantity) DESC

LIMIT 1

)

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

ORDER BY amount\_spent DESC;

* This SQL query **finds customers who spent money on the best-selling artist** and **calculates how much they spent**.
* **How the Query Works (Step by Step)**

**1️. Find the Best-Selling Artist (Subquery)**

WHERE ar.artist\_id = (

SELECT ar.artist\_id

FROM artist ar

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

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

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

GROUP BY ar.artist\_id

ORDER BY SUM(il.unit\_price \* il.quantity) DESC

LIMIT 1

)

* **This part finds the artist who has sold the most music.**
* It adds up all **sales (unit price × quantity) for each artist**.
* **Orders them from highest to lowest** sales.
* **Picks the top-selling artist** using LIMIT 1.

**2️. Find Customers Who Bought That Artist’s Music**

SELECT c.customer\_id,

c.first\_name,

c.last\_name,

ar.name AS artist\_name,

SUM(il.unit\_price \* il.quantity) AS amount\_spent

* **Gets customer details** (customer\_id, first\_name, last\_name).
* **Gets the artist’s name** (artist\_name).
* **Calculates total spending** (SUM(il.unit\_price \* quantity)).

**3️. Connect All Necessary Tables**

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

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

* **Connects customers to invoices** (who made purchases).
* **Connects invoices to invoice lines** (which tracks they bought).
* **Connects tracks to albums and albums to artists** (who made the music).

**4️. Only Keep Customers Who Bought the Best-Selling Artist’s Music**

WHERE ar.artist\_id = (subquery)

* **Filters results** to include **only** purchases from the **top-selling artist**.

**5️. Group & Sort the Results**

GROUP BY c.customer\_id, ar.name

ORDER BY amount\_spent DESC;

* Groups by customer ID and artist name (to sum up spending per customer).
* Orders by amount spent (highest first) so the biggest spenders appear at the top.
* **Why Do We Use So Many Joins?**

Music sales data is spread across multiple tables:  
🎵 Tracks → Albums → Artists  
🛒 Invoice → Invoice Line → Tracks  
👤 Customers → Invoices

* We need joins to connect customers with the artists they bought from.
* **Summary**

✔ Find the best-selling artist  
✔ Find customers who bought their music  
✔ Calculate how much each customer spent  
✔ Show results from highest spenders to lowest

This query helps understand customer spending habits for the most popular artist.

1. **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.**

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

* This query finds the most popular music genre in each country based on the number of purchases.

**Steps to Solve This Problem**

**🔹 Step 1: Identify Key Tables**

We need the following information:

* Customer’s country → from the customer table
* Track genre → from the genre table
* Track purchases → from the invoice\_line table
* Link customers to purchases → using the invoice table

**Step 2: Count Purchases for Each Genre in Each Country**

**COUNT(invoice\_line.quantity) AS purchases**

* Counts the total number of times each genre was purchased in each country.
* Groups data by country and genre to summarize total purchases.

**Step 3: Rank Genres Within Each Country**

**ROW\_NUMBER() OVER(PARTITION BY customer.country ORDER BY COUNT(invoice\_line.quantity) DESC) AS RowNo**

* PARTITION BY customer.country → Separates rankings for each country.
* ORDER BY COUNT(invoice\_line.quantity) DESC → Sorts genres from most purchased to least.
* Assigns a ranking (RowNo) where RowNo = 1 means the most purchased genre in that country.

**Step 4: Store This Data in a Temporary Table (popular\_genre)**

* **WITH popular\_genre AS (...)**

Creates a temporary dataset that holds purchase counts and rankings for each genre per country.

**Step 5: Select Only the Most Popular Genre for Each Country**

* **SELECT \* FROM popular\_genre WHERE RowNo <= 1**

Filters the data so that only the top-ranked (RowNo = 1) genre per country is included in the final results.

* **📌 Final Summary of Steps**
* **✅ Step 1: Identify necessary tables and relationships.  
  ✅ Step 2: Count total purchases for each genre in each country.  
  ✅ Step 3: Use ROW\_NUMBER() to rank genres within each country.  
  ✅ Step 4: Store results in a temporary table (popular\_genre).  
  ✅ Step 5: Select only the top genre (RowNo = 1) for each country.**
* ensures we get one genre per country.

1. **Q3: 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**

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

**Steps to Solve This Problem**

**🔹 Step 1: Identify Key Tables**

To determine the top spender in each country, we need:

**customer table** → To get customer details (first name, last name, and country).

**invoice table** → To get purchase amounts (total) and link them to customers.

**🔹 Step 2: Calculate Total Spending for Each Customer**

SUM(total) AS total\_spending

* This adds up the total purchases (SUM(total)) made by each customer.
* Groups data by customer\_id to get the total spending per customer.

**🔹 Step 3: Rank Customers Within Each Country**

ROW\_NUMBER() OVER(PARTITION BY billing\_country ORDER BY SUM(total) DESC) AS RowNo

* PARTITION BY billing\_country → Separates rankings for each country.
* ORDER BY SUM(total) DESC → Sorts customers from highest to lowest spending.
* Assigns a ranking (RowNo) where RowNo = 1 means the highest spender in that country.

**🔹 Step 4: Store This Data in a Temporary Table (Customter\_with\_country)**

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

)

* Creates a temporary dataset (Customter\_with\_country) that stores:
  + Each customer’s total spending.
  + Their **ranking** (RowNo) within their country.

**🔹 Step 5: Select Only the Highest-Spending Customer in Each Country**

SELECT \* FROM Customter\_with\_country WHERE RowNo <= 1;

* **Filters the data** so that **only the top-ranked (RowNo = 1) customer per country** is included in the final results.

**📌 Final Summary of Steps**

✅ **Step 1:** Identify necessary tables and relationships.  
✅ **Step 2:** Calculate total spending per customer (SUM(total)).  
✅ **Step 3:** Use ROW\_NUMBER() to rank customers within each country.  
✅ **Step 4:** Store results in a temporary table (Customter\_with\_country).  
✅ **Step 5:** Select only the highest spender (RowNo = 1) for each country.