https://www.kaggle.com/datasets/tonygordonjr/spotify-dataset-2023?utm_source=chatgpt.com
https://www.kaggle.com/datasets/maharshipandya/-spotify-tracks-dataset?utm_source=chatgpt.com
https://www.kaggle.com/datasets/solomonameh/spotify-music-dataset?utm_source=chatgpt.com
https://www.kaggle.com/datasets/salvatorerastelli/spotify-and-youtube?utm_source=chatgpt.com
https://www.kaggle.com/datasets/amitanshjoshi/spotify-1million-tracks?utm_source=chatgpt.com
https://www.kaggle.com/datasets/nelgiriyewithana/top-spotify-songs-2023?utm_source=chatgpt.com

1. Show all columns from `spotify_songs`.
2. Display only `track_name`, `artist_name`, and `genre`.
3. Count the total number of songs.
4. Show distinct genres available.
5. Find all songs released in **2020**.
6. Get all songs by **Ed Sheeran**.
7. Show the top 5 most popular songs.
8. Find the longest track by `duration_ms`.
9. Find the shortest track by `duration_ms`.
10. Get songs with popularity greater than 80.
11. List all tracks released before **2015**.
12. Show all songs sorted by popularity (descending).
13. Count how many songs are in the **Pop** genre.
14. Find the average duration of songs.
15. Show the first 10 songs with the highest tempo.

---

## 🔹 Intermediate Level (16–35)

16. Show the number of songs per genre.
17. Find the top 3 artists with the most songs.
18. Get the average popularity of each genre.
19. List artists who have more than 50 songs.
20. Show the 5 least popular songs.
21. Find the average tempo per genre.
22. Show all songs where `danceability > 0.8`.
23. Show songs with popularity between 60 and 80.
24. Count the number of songs per release year.
25. Find the genre with the highest average popularity.
26. Get the top 5 artists with the highest average song popularity.
27. Show all songs where duration is greater than **5 minutes** (300,000 ms).
28. Find the year with the most songs released.
29. Get all songs with **valence > 0.7 and energy > 0.7**.
30. Show the average danceability of songs by Adele.
31. Find artists who released songs in more than 3 different years.
32. Show songs where duration is greater than the **average duration** of all tracks.
33. List the number of albums per artist.
34. Find the artist with the longest total duration of songs.
35. Show all songs where `track_name` contains the word "Love".

---

## 🔹 Advanced Level (36–50)

36. Rank songs in each genre by popularity using **ROW_NUMBER()**.
37. Use a **CTE** to calculate the average popularity per genre, then select genres above the overall average.
38. Show songs with popularity above their genre’s average (use a subquery).
39. Find the year with the highest **average danceability**.
40. Compare each song’s duration to the previous song by the same artist (use **LAG()**).
41. Get the top 10% of songs by popularity (use **NTILE(10)** or **PERCENT_RANK()**).
42. Find the most energetic song for each artist (use GROUP BY + MAX).
43. Calculate a rolling 5-year average of popularity (if DB supports **window functions**).
44. Show the longest track per genre.
45. Find the average popularity of songs released after 2015 compared to before 2015.
46. Create a pivot table: number of songs per genre per year.
47. Find the correlation between `energy` and `danceability` using SQL (if DB supports CORR function).
48. Show artists who appear in both Pop and Rock genres (use **INTERSECT/EXISTS**).
49. Create a view for the top 100 most popular songs.
50. Find the difference in average popularity between Pop and R&B songs.

## 🔹 Ranking Functions (ROW_NUMBER, RANK, DENSE_RANK)

1. Assign a **row number** to songs ordered by popularity.
2. Rank songs by duration (longest first).
3. Use `DENSE_RANK()` to rank songs by tempo.
4. Rank songs **within each genre** by popularity.
5. Find the **most popular song per artist** using `ROW_NUMBER()`.

6. Show the top 3 songs per artist by popularity.

7. Find the longest track per album using `RANK()`.

8. For each year, rank songs by danceability.

9. Find the 2nd most popular song for each genre.

10. Show songs ranked by valence within each artist.

---

## 🔹 Aggregates with Window Functions

11. Show each song with the **average popularity** of its genre.

12. Display each song's popularity compared to the **average popularity of all songs**.

13. Find each artist's song popularity compared to their **own average popularity**.

14. Show total number of songs per artist using `COUNT() OVER()`.

15. Show total number of songs per genre using `COUNT() OVER(PARTITION BY genre)`.

16. Show each song's duration and the **total duration of songs by that artist**.

17. Find each song's danceability compared to the **max danceability of its genre**.

18. Show min, max, and average tempo for each genre.

19. Show each song's popularity share in its artist's catalog (song_popularity / SUM(popularity)).

20. Show cumulative popularity of songs per artist using `SUM() OVER(ORDER BY popularity)`.

---

## 🔹 LAG / LEAD

21. Show each song and the **previous song's popularity** using `LAG()`.

22. Show each song and the **next song's popularity** using `LEAD()`.

23. Find difference in popularity between each song and the previous one (same artist).

24. Find difference in duration between consecutive songs (ordered by release_year).

25. Show tempo difference between each song and the next one within the same genre.

26. Show change in popularity between consecutive years for each artist.

27. Compare danceability of current vs. previous track per album.

28. Show difference in valence between consecutive songs (same genre).

29. Find if a song is more popular than the previous release by the same artist.

30. Show the release year and previous year's average popularity per artist.

---

## 🔹 FIRST_VALUE / LAST_VALUE / NTH_VALUE

31. Show the **first song released** by each artist.

32. Show the **last song released** by each artist.

33. Show the first popular song (highest popularity) per genre.

34. Show the last song (lowest popularity) per genre.

35. Use `NTH_VALUE` to get the 3rd most popular song per artist.

36. Use `FIRST_VALUE` to show the earliest album of each artist.

37. Use `LAST_VALUE` to show the latest album of each artist.

38. Show first tempo track and last tempo track per genre.

39. Show first 2020 release and last 2020 release per artist.

40. Use `NTH_VALUE` to fetch the 5th longest track overall.

---

## 🔹 Advanced Analytics (NTILE, Moving Averages, Percentile)

41. Divide songs into **quartiles** by popularity using `NTILE(4)`.
42. Divide songs into 10 groups based on tempo (`NTILE(10)`).
43. Find the top 10% songs by energy using `PERCENT_RANK()`.
44. Compute cumulative count of songs by artist ordered by release year.
45. Compute running total of popularity per genre.
46. Compute running average duration per artist.
47. Show rolling 3-song average popularity using window frame (`ROWS BETWEEN 2 PRECEDING AND CURRENT ROW`).
48. Show moving average tempo over 5 songs (ordered by release year).
49. Show the percentile rank of each song's popularity within its genre.
50. Find the median popularity per genre using `PERCENTILE_CONT(0.5)`.

# CTE + CASE SQL Questions

### Basic (1–10)

1. Write a CTE to calculate song duration in minutes. Use a `CASE` to label songs as **"Short" (<3 min), "Medium" (3–5 min), or "Long" (>5 min)**.
2. Use a CTE to classify songs by popularity:
    - `CASE WHEN popularity >= 80 THEN 'Hit' WHEN popularity >= 50 THEN 'Average' ELSE 'Low' END`.
3. Create a CTE that counts songs by genre and use `CASE` to label:
    - `Pop`, `Non-Pop`.

4. Use a CTE to calculate the **average danceability per song**, then `CASE` classify: High (>0.7) or Low (<=0.7).

5. Write a CTE that returns all tracks released before 2015, then use `CASE` to mark them as “Old Songs”.

6. Use a CTE to calculate average tempo per genre and use `CASE` to mark `Fast (>120)` or `Slow`.

7. Use a CTE to filter only Ed Sheeran’s songs, then classify by popularity into High/Low.

8. Create a CTE that selects songs from 2020 and use `CASE` to label them as “Pandemic Release”.

9. Use a CTE to calculate the average popularity per year. In the main query, mark each year as “Above Avg” or “Below Avg”.

10. Write a CTE that shows album-level total duration. Add a `CASE` to classify albums as Short (<20 min), Medium (20–40 min), or Long (>40 min).

---

## Intermediate (11–20)

11. Use a CTE to calculate the average popularity per artist. Then use `CASE` to mark them as “Top Artist” (>80) or “Other”.

12. Write a CTE to calculate total songs per release year. Add a `CASE` to classify years as: “Classic (<2010)”, “Modern (2010–2019)”, “Recent (2020+)”.

13. Use a CTE to calculate average energy per genre. Add a `CASE` to classify: “Energetic” if >0.6, else “Calm”.

14. Use a CTE to calculate track tempo categories: `CASE` → “Chill (<90)”, “Normal (90–120)”, “Fast (120–150)”, “Extreme (>150)”.

15. Write a CTE that calculates the number of songs per artist. Use `CASE` to label: “Rising (<5 songs)”, “Active (5–20 songs)”, “Prolific (>20 songs)”.

16. Create a CTE that calculates average valence (mood) per genre. Use `CASE` to mark as “Happy (>0.6)” or “Sad”.

17. Use a CTE to calculate the longest track per artist. Add a `CASE` to check if the track is longer than 7 minutes = “Extended Play”.

18. Write a CTE that finds the average popularity of songs per album. Add a `CASE` to label albums as “Successful (>70)” or “Flop”.

19. Use a CTE to calculate the total number of Pop vs Non-Pop songs. Use `CASE` for classification.

20. Write a CTE that calculates average popularity per decade. Use `CASE` to group: 1980s, 1990s, 2000s, 2010s, 2020s.

---

## Advanced (21–30)

21. Use a CTE with a `CASE` to classify songs as “Hit” (popularity >80) or “Flop”, then calculate hit ratio per artist.

22. Use a CTE that finds the difference between a song’s popularity and its artist’s average popularity. Use `CASE` to mark “Above Artist Avg” or “Below Artist Avg”.

23. Write a recursive CTE to calculate cumulative songs per year. Add a `CASE` to classify: “Growth” or “Decline” compared to the previous year.

24. Use a CTE with window functions: calculate each song’s popularity rank per genre. Add a `CASE` to label ranks 1–3 as “Top 3”.

25. Create a CTE that calculates the average tempo per artist. Use `CASE` to classify artists as “High BPM” (>120) or “Low BPM”.

26. Write a CTE that groups by artist and calculates total song duration. Use `CASE` to mark artists with >1 hour of songs as “Long Play Artists”.

27. Use a CTE to calculate the percentage of Pop songs per year. Add a `CASE` to label years as “Pop Dominated” (>50% Pop) or “Mixed”.

28. Write a CTE that shows popularity trend by comparing current year’s average vs previous year. Use `CASE` to mark “Improved” or “Declined”.

29. Use a CTE to find the artist with the highest energy per year. Add a `CASE` to label them as “Energy King”.

30. Write a CTE that calculates quartiles of popularity using `NTILE(4)`. Use `CASE` to label songs as Q1, Q2, Q3, Q4.

# 30 Advanced SQL Join Questions

## 1. Multi-Table Joins

1. Join `spotify_songs` with `artists` to display song names with their artist names.
2. Join `spotify_songs` with `albums` to show song names with album names.
3. Join all three tables to display `track_name, artist_name, album_name`.
4. Show all songs with their artist country (using join between `spotify_songs` and `artists`).
5. List albums with their total number of songs (JOIN + GROUP BY).

---

## 2. Self Joins

6. Use a self-join on `spotify_songs` to find pairs of songs from the same artist released in the same year.
7. Use a self-join to compare the longest and shortest songs by the same artist.
8. Find song pairs from the same album where one song has higher popularity than the other.
9. Find artists who released songs with the same tempo (±5 BPM) using self-join.
10. Use self-join to list song names that have identical duration.

---

## 3. Subqueries with Joins

11. Show songs that belong to the album with the **maximum number of songs** (subquery + join).
12. Find artists whose **average popularity** is greater than the **overall average popularity**.
13. List all songs from the artist who has the **most songs in the dataset**.

14. Find albums released in the same year as the album with the **highest average popularity**.

15. Show songs from the genre that has the **highest average danceability**.

---

## 4. CTE + Joins

16. Use a CTE to calculate average popularity per artist, then join with `artists` to show “Top 10 Artists”.

17. Write a CTE that finds the most popular album per year, then join with `albums` to show album details.

18. Create a CTE of songs classified as “Hit” (popularity > 80), then join with `artists` to list hitmakers.

19. Use a CTE to find total song duration per artist, then join with `artists` table to show artist names.

20. Create a CTE that calculates number of songs per genre per year, then join with `spotify_songs` to compare.

---

## 5. Advanced Join Scenarios

21. Perform a LEFT JOIN to show all artists and their songs, including artists with no songs.

22. Perform a RIGHT JOIN to show all albums and their songs, including albums with no tracks.

23. Perform a FULL OUTER JOIN to show all artists and albums (even if some don’t match).

24. Find artists who released albums but have no songs in the dataset (anti-join).

25. Show albums with songs in multiple genres (JOIN + GROUP BY + HAVING).

---

## 6. Complex Analytics with Joins

26. Find the **most popular song per artist** using JOIN + window function.

27. List artists with more than 1 album, and show their album names using JOIN.

28. Find the **top 3 most popular songs per country** (JOIN spotify_songs + artists + window function).

29. Join songs and albums to find the **year with the most album releases**.

30. Use a JOIN with NTILE() to divide artists into 4 quartiles based on total popularity.