

(CS 425-02) DATABASE ORGANISATION

Project Deliverable - 3

Group Members Details

Name	CWID	EMAIL
DEVESH PATEL	A20548274	dpatel219@hawk.iit.edu
PURVIT ASHESH PATEL	A20551053	ppatel180@hawk.iit.edu
VISHWASHREE CHANNAA REDDY	A20556543	vishwashreech@hawk.iit.edu

Equal Contribution

1) Shows how many songs each artist has.

Explanation: This query counts and lists the total number of songs attributed to each artist.

Query:

```
SELECT a.ArtistName, COUNT(sa.SongID) AS TotalSongs
```

```
FROM artist a
```

```
JOIN SongArtist sa ON a.ArtistID = sa.ArtistID
```

```
GROUP BY a.ArtistName;
```

The screenshot shows a database query editor with a toolbar at the top. The query text is as follows:

```
433 ('Free', 0.00),
434 ('Premium', 9.99),
435 ('Family', 14.99);
436
437 • SELECT a.ArtistName, COUNT(sa.SongID) AS TotalSongs
438 FROM artist a
439 JOIN SongArtist sa ON a.ArtistID = sa.ArtistID
440 GROUP BY a.ArtistName;
441
```

Below the query editor is the 'Result Grid' section, which displays the results of the query in a table format:

ArtistName	TotalSongs
Taylor Swift	2
The Beatles	2
Drake	2
Billie Eilish	2
Ed Sheeran	2
Ariana Grande	1
Elton John	1
Beyoncé	1
Bob Dylan	1
Kanye West	1

At the bottom of the screenshot is the 'Output' section, which shows the execution details of the query:

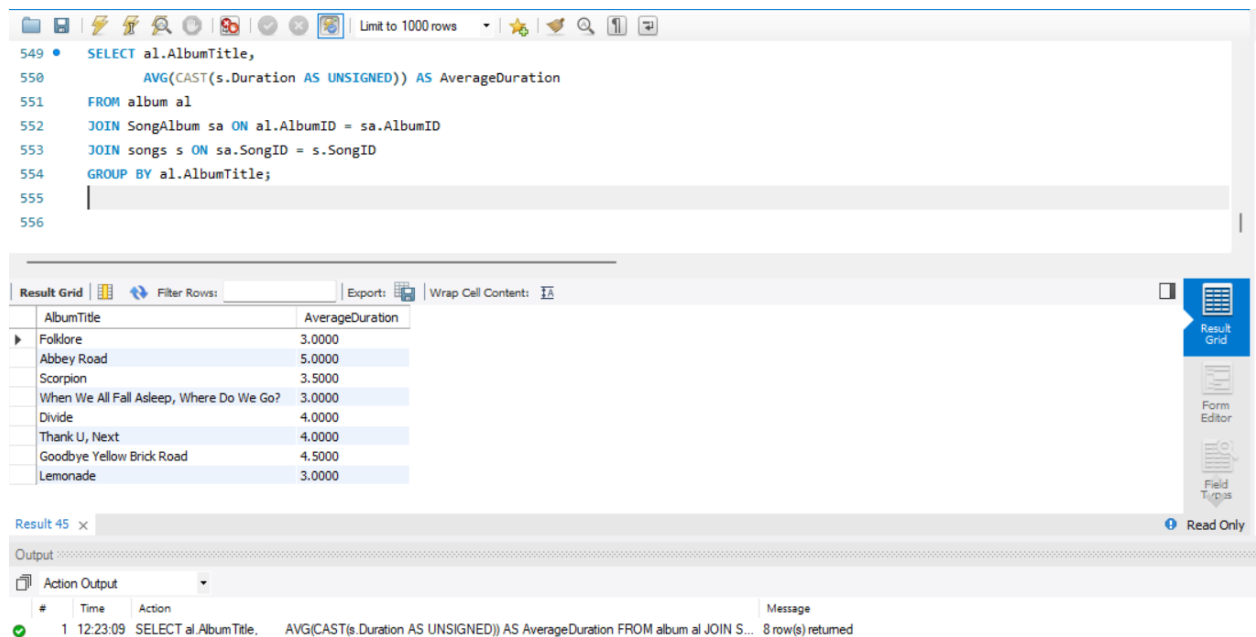
#	Time	Action	Message
1	12:01:47	SELECT a.ArtistName, COUNT(sa.SongID) AS TotalSongs FROM artist a JOIN SongArtist sa ON a.ArtistID = sa...	10 row(s) returned

2) Average Duration of Albums

Explanation: This query results the calculation and displays the mean duration of songs across each album in a music database

Query:

```
SELECT al.AlbumTitle,  
       AVG(CAST(s.Duration AS UNSIGNED)) AS AverageDuration  
FROM album al  
JOIN SongAlbum sa ON al.AlbumID = sa.AlbumID  
JOIN songs s ON sa.SongID = s.SongID  
GROUP BY al.AlbumTitle;
```



The screenshot shows a database query editor with the following SQL query:

```
549 • SELECT al.AlbumTitle,  
550       AVG(CAST(s.Duration AS UNSIGNED)) AS AverageDuration  
551 FROM album al  
552 JOIN SongAlbum sa ON al.AlbumID = sa.AlbumID  
553 JOIN songs s ON sa.SongID = s.SongID  
554 GROUP BY al.AlbumTitle;  
555  
556
```

Below the query editor is a "Result Grid" showing the results of the query. The grid has two columns: "AlbumTitle" and "AverageDuration". The results are as follows:

AlbumTitle	AverageDuration
Folklore	3.0000
Abbey Road	5.0000
Scorpion	3.5000
When We All Fall Asleep, Where Do We Go?	3.0000
Divide	4.0000
Thank U, Next	4.0000
Goodbye Yellow Brick Road	4.5000
Lemonade	3.0000

At the bottom of the screenshot, there is an "Action Output" section showing the execution of the query. It indicates that the query was executed at 12:23:09 and returned 8 rows.

3) List all genres with the number of albums associated with each genre.

Explanation: The objective of identifying each music genre available in a database and counting how many albums are categorized under each genre.

Query:

```
SELECT g.GenreName, COUNT(ag.AlbumID) AS AlbumCount
```

```
FROM genre g
```

```
LEFT JOIN AlbumGenre ag ON g.GenreID = ag.GenreID
```

```
GROUP BY g.GenreID;
```

The screenshot shows a database query editor interface. The top section contains the SQL query, which is a multi-part query. The bottom section displays the results of the query in a table format. The table has two columns: 'GenreName' and 'AlbumCount'. The results show 15 rows of data, including genres like Rock, Hip-Hop, Indie, Electronic, Country, R&B, Jazz, Classical, Metal, Folk, and Blues.

```
443 FROM artist a
444 JOIN SongArtist sa ON a.ArtistID = sa.ArtistID
445 GROUP BY a.ArtistName;
446
447 • SELECT g.GenreName, COUNT(ag.AlbumID) AS AlbumCount
448 FROM genre g
449 LEFT JOIN AlbumGenre ag ON g.GenreID = ag.GenreID
450 GROUP BY g.GenreID;
451
```

GenreName	AlbumCount
Rock	3
Hip-Hop	3
Indie	3
Electronic	3
Country	0
R&B	0
Jazz	0
Classical	0
Metal	0
Folk	0
Blues	0

Result 27 x

Output

#	Time	Action	Message
1	12:04:44	SELECT g.GenreName, COUNT(ag.AlbumID) AS AlbumCount FROM genre g LEFT JOIN AlbumGenre ag ON g....	15 row(s) returned

4) List all artists who released albums in the last 5 years.

Explanation: This query uses for identifying and listing artists who have released albums within the past five years.

Query:

```
SELECT DISTINCT a.ArtistName
```

```
FROM artist a
```

```
JOIN ArtistAlbum aa ON a.ArtistID = aa.ArtistID
```

```
JOIN album al ON aa.AlbumID = al.AlbumID
```

```
WHERE al.ReleaseYear >= YEAR(CURDATE()) - 5;
```

The screenshot displays a SQL query editor and its results. The query editor contains the following SQL code:

```
445 GROUP BY a.ArtistName;
446
447 • SELECT g.GenreName, COUNT(ag.AlbumID) AS AlbumCount
448 FROM genre g
449 LEFT JOIN AlbumGenre ag ON g.GenreID = ag.GenreID
450 GROUP BY g.GenreID;
451
452 • SELECT DISTINCT a.ArtistName
453 FROM artist a
454 JOIN ArtistAlbum aa ON a.ArtistID = aa.ArtistID
455 JOIN album al ON aa.AlbumID = al.AlbumID
456 WHERE al.ReleaseYear >= YEAR(CURDATE()) - 5;
457
```

The results viewer shows a table with the following data:

ArtistName
The Beatles
Drake
Taylor Swift
Elton John

The results viewer also shows a message: "4 row(s) returned".

5) Calculate the average song duration by genre.

Explanation: This query describes the process of calculating the mean duration of songs within each genre in a music database.

Query:

```
SELECT g.GenreName, AVG(TIME_TO_SEC(s.Duration)) AS AverageDurationSeconds
```

```
FROM genre g
```

```
JOIN SongGenre sg ON g.GenreID = sg.GenreID
```

```
JOIN songs s ON sg.SongID = s.SongID
```

```
GROUP BY g.GenreID;
```

The screenshot shows a database query editor with a SQL query and its results. The query is as follows:

```
451
452 • SELECT DISTINCT a.ArtistName
453 FROM artist a
454 JOIN ArtistAlbum aa ON a.ArtistID = aa.ArtistID
455 JOIN album al ON aa.AlbumID = al.AlbumID
456 WHERE al.ReleaseYear >= YEAR(CURDATE()) - 5;
457
458 • SELECT g.GenreName, AVG(TIME_TO_SEC(s.Duration)) AS AverageDurationSeconds
459 FROM genre g
460 JOIN SongGenre sg ON g.GenreID = sg.GenreID
461 JOIN songs s ON sg.SongID = s.SongID
462 GROUP BY g.GenreID;
463
```

The results are displayed in a table with the following data:

GenreName	AverageDurationSeconds
Pop	15740.0000
Rock	17540.0000
Hip-Hop	14860.0000
Indie	15280.0000
Electronic	15160.0000

The interface also shows an 'Action Output' section at the bottom, indicating that the query was executed successfully and returned 5 rows.

6) Find the user with the most playlists.

Explanation: The query individual user who has created the highest number of playlists in a database, indicating the most active or engaged user in terms of playlist creation.

Query:

```
SELECT u.Username, COUNT(p.PlaylistID) AS PlaylistCount
```

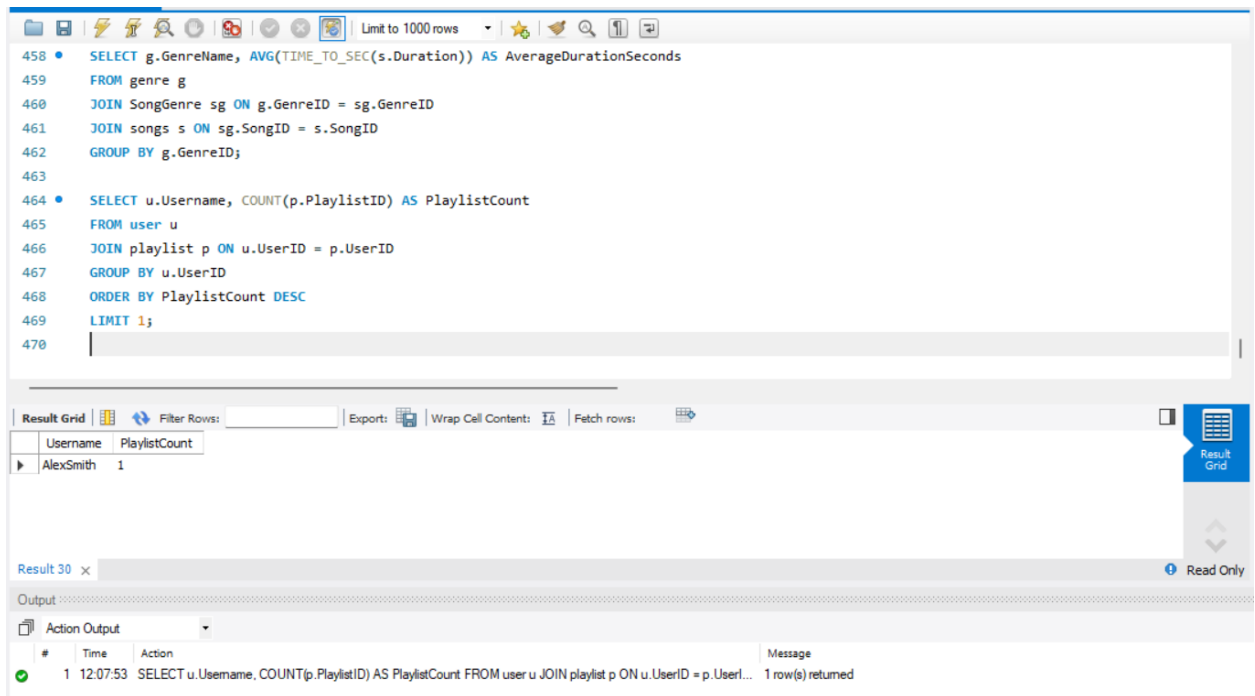
```
FROM user u
```

```
JOIN playlist p ON u.UserID = p.UserID
```

```
GROUP BY u.UserID
```

```
ORDER BY PlaylistCount DESC
```

```
LIMIT 1;
```



The screenshot displays a SQL query editor and its results. The query editor contains the following SQL code:

```
458 • SELECT g.GenreName, AVG(TIME_TO_SEC(s.Duration)) AS AverageDurationSeconds
459 FROM genre g
460 JOIN SongGenre sg ON g.GenreID = sg.GenreID
461 JOIN songs s ON sg.SongID = s.SongID
462 GROUP BY g.GenreID;
463
464 • SELECT u.Username, COUNT(p.PlaylistID) AS PlaylistCount
465 FROM user u
466 JOIN playlist p ON u.UserID = p.UserID
467 GROUP BY u.UserID
468 ORDER BY PlaylistCount DESC
469 LIMIT 1;
470
```

The results viewer shows a single row in the Result Grid:

Username	PlaylistCount
AlexSmith	1

The output pane shows the execution message:

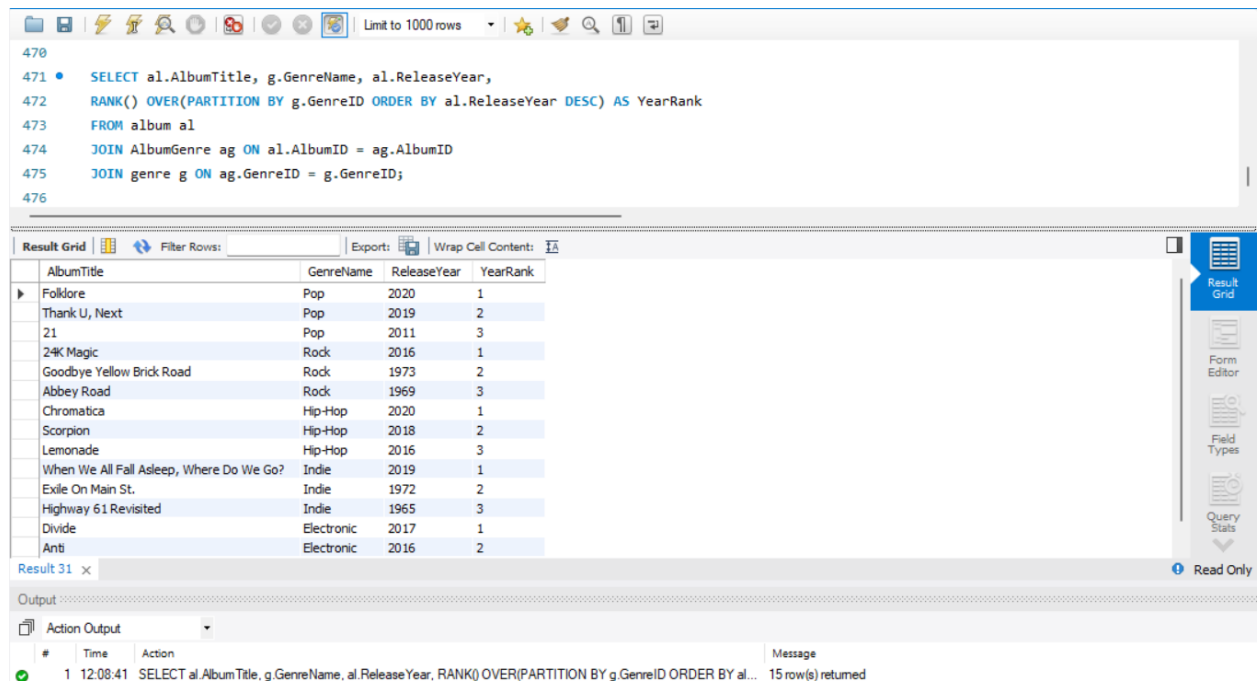
```
1 12:07:53 SELECT u.Username, COUNT(p.PlaylistID) AS PlaylistCount FROM user u JOIN playlist p ON u.UserID = p.UserID... 1 row(s) returned
```

7) Rank albums by release year within each genre.

Explanation: This query involves ordering albums based on their release years within each distinct genre, providing a way to see how albums are temporally distributed across various musical styles.

Query:

```
SELECT al.AlbumTitle, g.GenreName, al.ReleaseYear,  
  
RANK() OVER(PARTITION BY g.GenreID ORDER BY al.ReleaseYear DESC) AS  
YearRank  
  
FROM album al  
  
JOIN AlbumGenre ag ON al.AlbumID = ag.AlbumID  
  
JOIN genre g ON ag.GenreID = g.GenreID;
```



The screenshot displays a SQL query editor with a toolbar at the top. The query is as follows:

```
470  
471 • SELECT al.AlbumTitle, g.GenreName, al.ReleaseYear,  
472 RANK() OVER(PARTITION BY g.GenreID ORDER BY al.ReleaseYear DESC) AS YearRank  
473 FROM album al  
474 JOIN AlbumGenre ag ON al.AlbumID = ag.AlbumID  
475 JOIN genre g ON ag.GenreID = g.GenreID;  
476
```

Below the query editor, the 'Result Grid' is visible, showing the results of the query. The grid has four columns: AlbumTitle, GenreName, ReleaseYear, and YearRank. The results are as follows:

AlbumTitle	GenreName	ReleaseYear	YearRank
Folklore	Pop	2020	1
Thank U, Next	Pop	2019	2
21	Pop	2011	3
24K Magic	Rock	2016	1
Goodbye Yellow Brick Road	Rock	1973	2
Abbey Road	Rock	1969	3
Chromatica	Hip-Hop	2020	1
Scorpion	Hip-Hop	2018	2
Lemonade	Hip-Hop	2016	3
When We All Fall Asleep, Where Do We Go?	Indie	2019	1
Exile On Main St.	Indie	1972	2
Highway 61 Revisited	Indie	1965	3
Divide	Electronic	2017	1
Anti	Electronic	2016	2

At the bottom of the screenshot, the 'Action Output' section shows a message: '1 12:08:41 SELECT al.AlbumTitle, g.GenreName, al.ReleaseYear, RANK() OVER(PARTITION BY g.GenreID ORDER BY al... 15 row(s) returned'.

8) Rank artists by the number of playlists their songs appear in.

Explanation: This query involves determining the popularity or visibility of artists based on the frequency of their songs' appearances in playlists. By counting how many playlists include at least one song by an artist and then ranking these artists accordingly.

Query:

```
SELECT a.ArtistName, COUNT(DISTINCT sp.PlaylistID) AS PlaylistAppearances
```

```
FROM artist a
```

```
JOIN SongArtist sa ON a.ArtistID = sa.ArtistID
```

```
JOIN SongPlaylist sp ON sa.SongID = sp.SongID
```

```
GROUP BY a.ArtistID
```

```
ORDER BY PlaylistAppearances DESC;
```

The screenshot shows a database query editor interface. The top section contains the SQL query, which is a SELECT statement with JOINs and a GROUP BY clause. The bottom section displays the results in a table format. The table has two columns: ArtistName and PlaylistAppearances. The results are sorted in descending order of PlaylistAppearances. The first two rows are The Beatles and Ed Sheeran, both with 2 appearances. The remaining eight rows (Taylor Swift, Drake, Billie Eilish, Ariana Grande, Elton John, Beyoncé, Bob Dylan, and Kanye West) all have 1 appearance.

```
476
477 • SELECT a.ArtistName, COUNT(DISTINCT sp.PlaylistID) AS PlaylistAppearances
478 FROM artist a
479 JOIN SongArtist sa ON a.ArtistID = sa.ArtistID
480 JOIN SongPlaylist sp ON sa.SongID = sp.SongID
481 GROUP BY a.ArtistID
482 ORDER BY PlaylistAppearances DESC;
483
```

ArtistName	PlaylistAppearances
The Beatles	2
Ed Sheeran	2
Taylor Swift	1
Drake	1
Billie Eilish	1
Ariana Grande	1
Elton John	1
Beyoncé	1
Bob Dylan	1
Kanye West	1

Result 33 x Read Only

Output

#	Time	Action	Message
1	12:09:55	SELECT a.ArtistName, COUNT(DISTINCT sp.PlaylistID) AS PlaylistAppearances FROM artist a JOIN SongArtist ...	10 row(s) returned

9) Artist's First and Last Album Release Difference

Explanation: This query calculates the time span between the first and last albums released by each artist, offering insights into the length of their active recording careers.

Query:

```
SELECT a.ArtistName,  
       MIN(al.ReleaseYear) AS FirstAlbumYear,  
       MAX(al.ReleaseYear) AS LastAlbumYear,  
       MAX(al.ReleaseYear) - MIN(al.ReleaseYear) AS CareerSpanYears  
FROM artist a  
JOIN ArtistAlbum aa ON a.ArtistID = aa.ArtistID  
JOIN album al ON aa.AlbumID = al.AlbumID  
GROUP BY a.ArtistName  
HAVING CareerSpanYears > 0  
ORDER BY CareerSpanYears DESC;
```

The screenshot shows a SQL query editor with the following query:

```
484 • SELECT a.ArtistName,  
485       MIN(al.ReleaseYear) AS FirstAlbumYear,  
486       MAX(al.ReleaseYear) AS LastAlbumYear,  
487       MAX(al.ReleaseYear) - MIN(al.ReleaseYear) AS CareerSpanYears  
488 FROM artist a  
489 JOIN ArtistAlbum aa ON a.ArtistID = aa.ArtistID  
490 JOIN album al ON aa.AlbumID = al.AlbumID  
491 GROUP BY a.ArtistName  
492 HAVING CareerSpanYears > 0  
493 ORDER BY CareerSpanYears DESC;  
494
```

Below the query editor is a results grid showing the output of the query. The grid has columns: ArtistName, FirstAlbumYear, LastAlbumYear, and CareerSpanYears. The results are sorted by CareerSpanYears in descending order.

ArtistName	FirstAlbumYear	LastAlbumYear	CareerSpanYears
Taylor Swift	1969	2020	51
Elton John	1972	2020	48
Ed Sheeran	1965	2010	45
Billie Eilish	1973	2016	43
Ariana Grande	2011	2016	5
Drake	2017	2019	2
The Beatles	2018	2019	1

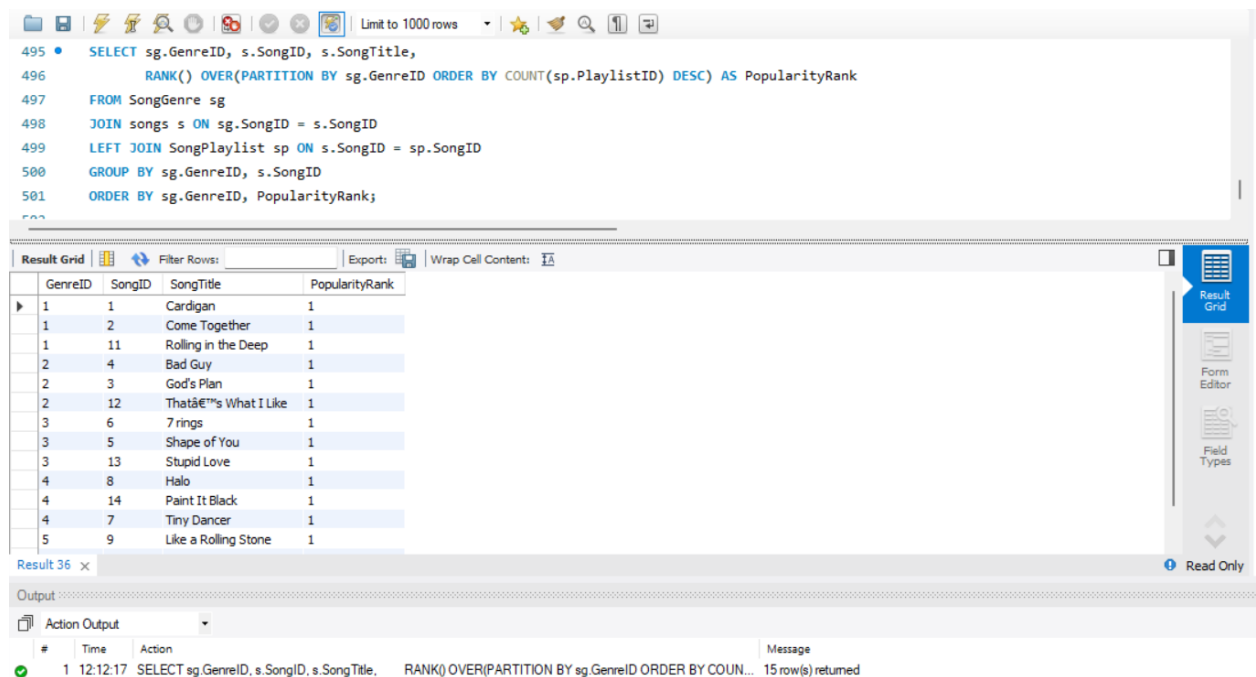
At the bottom of the screenshot, there is an "Action Output" section showing the execution of the query. It indicates that the query was executed at 12:11:23 and returned 7 rows.

10) Ranking Songs by Popularity Within Each Genre

Explanation: This process involves evaluating and ranking songs based on their popularity within their respective genres. Popularity can be determined by various metrics such as the number of times a song is played, featured in playlists, or downloaded.

Query:

```
SELECT sg.GenreID, s.SongID, s.SongTitle,  
       RANK() OVER(PARTITION BY sg.GenreID ORDER BY COUNT(sp.PlaylistID)  
DESC) AS PopularityRank  
FROM SongGenre sg  
JOIN songs s ON sg.SongID = s.SongID  
LEFT JOIN SongPlaylist sp ON s.SongID = sp.SongID  
GROUP BY sg.GenreID, s.SongID  
ORDER BY sg.GenreID, PopularityRank;
```



The screenshot displays a database query editor interface. The top section shows the SQL query being executed, which ranks songs by popularity within each genre. The results are displayed in a table with columns: GenreID, SongID, SongTitle, and PopularityRank. The table shows 15 rows of data, with songs ranked by their popularity within each genre. The bottom section shows the query execution log, indicating that the query was executed successfully and returned 15 rows.

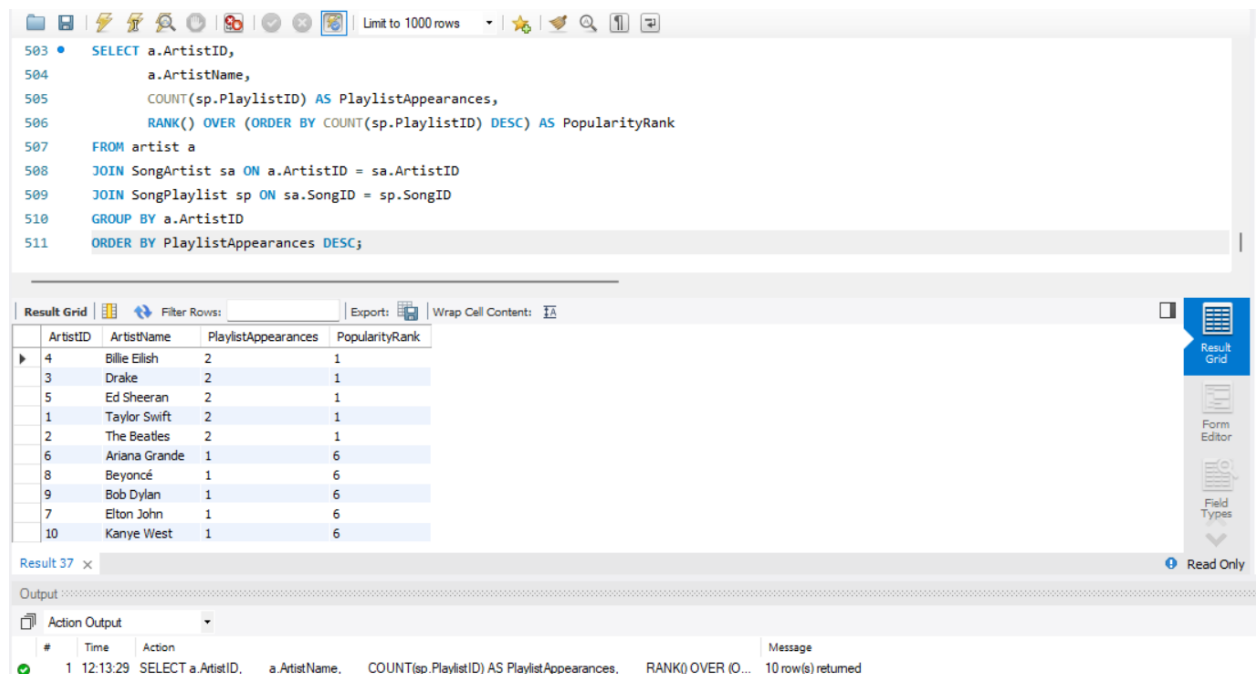
GenreID	SongID	SongTitle	PopularityRank
1	1	Cardigan	1
1	2	Come Together	1
1	11	Rolling in the Deep	1
2	4	Bad Guy	1
2	3	God's Plan	1
2	12	That's What I Like	1
3	6	7 rings	1
3	5	Shape of You	1
3	13	Stupid Love	1
4	8	Halo	1
4	14	Paint It Black	1
4	7	Tiny Dancer	1
5	9	Like a Rolling Stone	1

11) Artist Ranking by Number of Songs in Playlists

Explanation: This query approach ranks artists based on how frequently their songs are included in playlists, serving as a measure of their popularity and listener engagement within a music streaming platform. By counting the total number of playlist appearances for each artist's songs and then ranking the artists from most to least appearances.

Query:

```
SELECT a.ArtistID,  
       a.ArtistName,  
       COUNT(sp.PlaylistID) AS PlaylistAppearances,  
       RANK() OVER (ORDER BY COUNT(sp.PlaylistID) DESC) AS PopularityRank  
FROM artist a  
JOIN SongArtist sa ON a.ArtistID = sa.ArtistID  
JOIN SongPlaylist sp ON sa.SongID = sp.SongID  
GROUP BY a.ArtistID  
ORDER BY PlaylistAppearances DESC;
```



The screenshot displays a SQL query editor with the following query:

```
503 • SELECT a.ArtistID,  
504       a.ArtistName,  
505       COUNT(sp.PlaylistID) AS PlaylistAppearances,  
506       RANK() OVER (ORDER BY COUNT(sp.PlaylistID) DESC) AS PopularityRank  
507 FROM artist a  
508 JOIN SongArtist sa ON a.ArtistID = sa.ArtistID  
509 JOIN SongPlaylist sp ON sa.SongID = sp.SongID  
510 GROUP BY a.ArtistID  
511 ORDER BY PlaylistAppearances DESC;
```

Below the query editor, the "Result Grid" shows the results of the query. The grid has four columns: ArtistID, ArtistName, PlaylistAppearances, and PopularityRank. The results are as follows:

ArtistID	ArtistName	PlaylistAppearances	PopularityRank
4	Billie Eilish	2	1
3	Drake	2	1
5	Ed Sheeran	2	1
1	Taylor Swift	2	1
2	The Beatles	2	1
6	Ariana Grande	1	6
8	Beyoncé	1	6
9	Bob Dylan	1	6
7	Elton John	1	6
10	Kanye West	1	6

The bottom of the screenshot shows the "Output" section with a message: "1 12:13:29 SELECT a.ArtistID, a.ArtistName, COUNT(sp.PlaylistID) AS PlaylistAppearances, RANK() OVER (O... 10 row(s) returned".

12) Next and Previous Song Duration for Each Song

Explanation: This query analysis involves comparing the duration of each song in a database with the durations of the song that comes immediately before (previous) and after (next) it when sorted by a specified criterion, such as release date or alphabetical order. By applying window functions like LEAD and LAG, one can retrieve the duration of adjacent songs, providing context for understanding a song's length relative to others in the collection.

Query:

```
SELECT SongTitle,  
       Duration,  
       LEAD(Duration, 1) OVER (ORDER BY Duration) AS NextSongDuration,  
       LAG(Duration, 1) OVER (ORDER BY Duration) AS PreviousSongDuration  
FROM songs;
```

The screenshot shows a database query editor with the following SQL query:

```
513 • SELECT SongTitle,  
514         Duration,  
515         LEAD(Duration, 1) OVER (ORDER BY Duration) AS NextSongDuration,  
516         LAG(Duration, 1) OVER (ORDER BY Duration) AS PreviousSongDuration  
517 FROM songs;
```

The query results are displayed in a table with the following columns: SongTitle, Duration, NextSongDuration, and PreviousSongDuration. The results are sorted by Duration in ascending order.

SongTitle	Duration	NextSongDuration	PreviousSongDuration
Work	3:21	3:24	NULL
That's What I Like	3:24	3:34	3:21
Halo	3:34	3:41	3:24
Shape of You	3:41	3:50	3:34
Come Together	3:50	3:53	3:41
Cardigan	3:53	3:59	3:50
Tiny Dancer	3:59	4:19	3:53
7 rings	4:19	4:23	3:59
Stupid Love	4:23	4:32	4:19
Power	4:32	4:45	4:23
Like a Rolling Stone	4:45	5:11	4:32
Paint It Black	5:11	5:23	4:45
God's Plan	5:23	5:24	5:11
Rolling in the Deep	5:24	5:50	5:23
Bad Guy	5:50	NULL	5:24

The screenshot also shows the query execution status at the bottom, indicating that the query was successful and returned 15 rows.

13) First and Last Album Released Each Year

Explanation: This query identifies the first and last albums released in each year within a music database. It involves sorting albums by their release dates for every year and then selecting the earliest and latest release as representatives of the year's musical bookends.

Query:

```
SELECT ReleaseYear,
```

```
    FIRST_VALUE(AlbumTitle) OVER (PARTITION BY ReleaseYear ORDER BY  
    ReleaseYear ASC) AS FirstAlbum,
```

```
    LAST_VALUE(AlbumTitle) OVER (PARTITION BY ReleaseYear ORDER BY  
    ReleaseYear ASC ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED  
    FOLLOWING) AS LastAlbum
```

```
FROM album
```

```
GROUP BY ReleaseYear, AlbumTitle;
```

The screenshot shows a SQL query editor with the following query:

```
519 • SELECT ReleaseYear,  
520     FIRST_VALUE(AlbumTitle) OVER (PARTITION BY ReleaseYear ORDER BY ReleaseYear ASC) AS FirstAlbum,  
521     LAST_VALUE(AlbumTitle) OVER (PARTITION BY ReleaseYear ORDER BY ReleaseYear ASC ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING),  
522 FROM album  
523 GROUP BY ReleaseYear, AlbumTitle;  
524
```

Below the query editor is a results grid showing the output of the query. The grid has four columns: ReleaseYear, FirstAlbum, and LastAlbum. The data is as follows:

ReleaseYear	FirstAlbum	LastAlbum
1965	Highway 61 Revisited	Highway 61 Revisited
1969	Abbey Road	Abbey Road
1972	Exile On Main St.	Exile On Main St.
1973	Goodbye Yellow Brick Road	Goodbye Yellow Brick Road
2010	My Beautiful Dark Twisted Fantasy	My Beautiful Dark Twisted Fantasy
2011	21	21
2016	Lemonade	Anti
2016	Lemonade	Anti
2016	Lemonade	Anti
2017	Divide	Divide
2018	Scorpion	Scorpion
2019	When We All Fall Asleep, Where ...	Thank U, Next
2019	When We All Fall Asleep, Where ...	Thank U, Next
2020	Folklore	Chromatica
2020	Folklore	Chromatica

The results grid also shows a status bar at the bottom indicating "Result 40 x" and "Read Only".

Below the results grid is an "Action Output" section showing the execution of the query:

#	Time	Action	Message
1	12:16:12	SELECT ReleaseYear, FIRST_VALUE(AlbumTitle) OVER (PARTITION BY ReleaseYear ORDER BY Relea...	15 row(s) returned

14) Analysis of Genre Trends Over Time

Explanation: This query involves examining how the popularity of different music genres has changed over time within a music database or industry at large. By tracking the number of albums or songs released in each genre annually, one can identify trends, such as the rise or decline of specific genres, shifts in musical tastes, and the emergence of new genres.

Query:

```
SELECT ReleaseYear, GenreName,  
  
FIRST_VALUE(GenreName) OVER (PARTITION BY ReleaseYear ORDER BY GenreAppearance  
ASC) AS FirstGenreAppearance,  
  
LAST_VALUE(GenreName) OVER (PARTITION BY ReleaseYear ORDER BY GenreAppearance  
DESC ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING) AS  
LastGenreAppearance  
  
FROM (  
  
    SELECT s.ReleaseYear, g.GenreName,  
  
           ROW_NUMBER() OVER (PARTITION BY s.ReleaseYear, g.GenreName ORDER BY  
           s.ReleaseYear) AS GenreAppearance  
  
    FROM songs s  
  
    JOIN SongGenre sg ON s.SongID = sg.SongID  
  
    JOIN genre g ON sg.GenreID = g.GenreID  
  
) AS GenreTrends  
  
GROUP BY ReleaseYear, GenreName  
  
ORDER BY ReleaseYear, GenreName;
```

The screenshot displays a database query editor with the following SQL query:

```
525 SELECT ReleaseYear, GenreName,  
526 FIRST_VALUE(GenreName) OVER (PARTITION BY ReleaseYear ORDER BY GenreAppearance ASC) AS FirstGenreAppearance,  
527 LAST_VALUE(GenreName) OVER (PARTITION BY ReleaseYear ORDER BY GenreAppearance DESC ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING) AS LastGenreAppearance,  
528 FROM (  
529     SELECT s.ReleaseYear, g.GenreName,  
530            ROW_NUMBER() OVER (PARTITION BY s.ReleaseYear, g.GenreName ORDER BY s.ReleaseYear) AS GenreAppearance  
531     FROM songs s  
532     JOIN SongGenre sg ON s.SongID = sg.SongID  
533     JOIN genre g ON sg.GenreID = g.GenreID  
534 ) AS GenreTrends  
535 GROUP BY ReleaseYear, GenreName  
536 ORDER BY ReleaseYear, GenreName;
```

The results are shown in a table with the following columns: ReleaseYear, GenreName, FirstGenreAppearance, and LastGenreAppearance. The table contains 15 rows of data, showing the progression of genres over time.

ReleaseYear	GenreName	FirstGenreAppearance	LastGenreAppearance
1965	Electronic	Electronic	Electronic
1969	Pop	Pop	Pop
1972	Indie	Indie	Indie
1973	Indie	Indie	Indie
2010	Electronic	Electronic	Electronic
2011	Pop	Pop	Pop
2016	Electronic	Electronic	Rock
2016	Indie	Electronic	Rock
2016	Rock	Electronic	Rock
2017	Hip-Hop	Hip-Hop	Hip-Hop
2018	Rock	Rock	Rock
2019	Hip-Hop	Hip-Hop	Rock
2019	Rock	Hip-Hop	Rock

The output section shows the execution of the query, indicating that 15 rows were returned.

15) Artist Contribution to Genres

Explanation: This query analysis quantifies and highlights the contribution of individual artists to specific music genres. It involves counting the number of songs or albums an artist has released within each genre, thereby measuring their influence or dominance in those categories.

Query:

```
SELECT a.ArtistName,
       g.GenreName,
       COUNT(*) AS SongsInGenre,
       COUNT(*) * 100.0 / SUM(COUNT(*)) OVER (PARTITION BY g.GenreName) AS
PercentageOfGenre
FROM SongArtist sa
JOIN artist a ON sa.ArtistID = a.ArtistID
JOIN SongGenre sg ON sa.SongID = sg.SongID
JOIN genre g ON sg.GenreID = g.GenreID
GROUP BY a.ArtistName, g.GenreName
ORDER BY g.GenreName, PercentageOfGenre DESC;
```

The screenshot shows a database query editor with the following SQL query:

```
537
538 • SELECT a.ArtistName,
539         g.GenreName,
540         COUNT(*) AS SongsInGenre,
541         COUNT(*) * 100.0 / SUM(COUNT(*)) OVER (PARTITION BY g.GenreName) AS PercentageOfGenre
542 FROM SongArtist sa
543 JOIN artist a ON sa.ArtistID = a.ArtistID
544 JOIN SongGenre sg ON sa.SongID = sg.SongID
545 JOIN genre g ON sg.GenreID = g.GenreID
546 GROUP BY a.ArtistName, g.GenreName
547 ORDER BY g.GenreName, PercentageOfGenre DESC;
548
```

The results are displayed in a grid with the following columns: ArtistName, GenreName, SongsInGenre, and PercentageOfGenre.

ArtistName	GenreName	SongsInGenre	PercentageOfGenre
Ed Sheeran	Electronic	2	66.66667
Kanye West	Electronic	1	33.33333
Beyoncé	Hip-Hop	1	33.33333
Drake	Hip-Hop	1	33.33333
The Beatles	Hip-Hop	1	33.33333
Billie Eilish	Indie	2	66.66667
Bob Dylan	Indie	1	33.33333
Taylor Swift	Pop	2	66.66667
Ariana Grande	Pop	1	33.33333
Drake	Rock	1	33.33333
Elton John	Rock	1	33.33333
The Beatles	Rock	1	33.33333

The interface also includes a toolbar at the top with icons for various actions, a sidebar on the right with options like 'Result Grid', 'Form Editor', 'Field Types', and 'Query Data', and a status bar at the bottom showing 'Result 43' and 'Read Only'.