Workshop Tasks

Using the schema above, write SQL queries to answer the following questions:

1. **Basic Selection**: Retrieve the titles and release dates of all songs released in 2022, ordered by release date (newest first).

SELECT title, release_date

FROM songs

WHERE release_date BETWEEN '2022-01-01' AND '2022-12-31'

ORDER BY release_date DESC

```sql
SELECT title, release_date
FROM songs
WHERE release_date BETWEEN '2022-01-01' AND '2022-12-31'
ORDER BY release_date DESC
```

title	release_date
Anti-Hero	2022-10-21
Tití Me Preguntó	2022-05-06
Me Porto Bonito	2022-05-06

2. **Filtering**: Find all songs with a popularity score greater than 80 and a duration less than 4 minutes (240 seconds).

SELECT title, popularity_score, duration_seconds

FROM songs

WHERE popularity_score > 80 AND duration_seconds < 240

```
1 SELECT title, popularity_score, duration_sec
2 FROM songs
3 WHERE popularity_score > 80 AND duration_sec
```

title	popularity_score	duration
Anti-Hero	92	200
Cruel Summer	88	178
Blinding Lights	95	200
Die For You	90	232
Dynamite	86	199
Butter	89	164
Bad Guy	91	194
Shape of You	94	233
God's Plan	93	198
Don't Start Now	92	183
Levitating	94	203
Here Comes The Sun	85	186
Me Porto Bonito	89	178
Easy On Me	90	224
Thank U Next	89	207
7 Rings	88	178
Paint It Black	82	202
HUMBLE.	86	177
All The Stars	84	232
Poker Face	83	200
Shallow	92	217
Mr. Brightside	95	224
Somebody Told Me	83	197
Save Your Tears	91	215
Ocean Eyes	81	200
Shivers	86	207
One Dance	92	173
New Rules	88	209
Yesterday	86	125
Dakiti	88	205
Rolling in the Deep	90	228

3. **Pattern Matching**: List all artists whose names start with "The".

SELECT artist_name

FROM artists

WHERE artist_name LIKE 'The%'

```sql
SELECT artist_name
FROM artists
WHERE artist_name LIKE 'The%'
```

artist_name
The Weeknd
The Beatles
The Rolling Stones
The Killers

4. **Multiple Conditions**: Find all premium customers who joined in 2022.

SELECT customer_id, join_date, premium_member

FROM customers

WHERE premium_member = 1 AND join_date BETWEEN '2022-01-01' AND '2022-12-31'

```sql
SELECT customer_id, join_date, premium_member
FROM customers
WHERE premium_member = 1 AND join_date BETWEEN '2022-01-01' AND '2022-12-31'
```

customer_id	join_date	premium_member
4	2022-01-17	1
7	2022-09-03	1
11	2022-03-15	1
14	2022-07-07	1
18	2022-11-08	1

5. **Calculations and Aliasing**: Calculate the total duration (in minutes) of all songs in the database and display the result with an appropriate column name.

SELECT SUM(duration_seconds)/60 AS 'Total Duration of All Songs (Minutes)'

FROM songs

```sql
SELECT SUM(duration_seconds)/60 AS 'Total Duration of All Songs (Minutes)'
FROM songs
```

Total Duration of All Songs (Minutes)
141

6. **Advanced Filtering**: Find the top 5 most expensive song purchases in the database.

SELECT song_id, price

FROM purchases

ORDER BY price DESC

LIMIT 5

```sql
SELECT song_id, price
FROM purchases
ORDER BY price DESC
LIMIT 5
```

song_id	price
40	1.49
1	1.29
14	1.29
28	1.29
10	1.29

7. **Using Multiple Tables Separately**: First, find all song_ids from songs with a popularity score above 90. Then, use those song_ids to find purchases of those songs.

SELECT song_id, popularity_score

FROM songs

WHERE popularity_score >90

```sql
SELECT song_id, popularity_score
FROM songs
WHERE popularity_score >90
```

song_id	popularity_score
1	92
3	95
7	91
9	94
11	93
13	92
14	94
19	91
28	92
29	95
32	91
36	92

```sql
SELECT *
FROM purchases
WHERE song_id IN (1, 3,7, 9, 11, 13, 14, 19, 28, 29, 32, 36)
ORDER BY song_id
```

```sql
SELECT *
FROM purchases
WHERE song_id IN (1, 3,7, 9, 11, 13, 14, 19, 28, 29, 32, 36)
ORDER BY song_id
```

purchase_id	customer_id	song_id	purchase_date	price
1	4	1	2023-01-15	1.29
2	7	3	2023-02-02	0.99
4	12	7	2022-12-10	0.99
8	2	9	2023-02-18	0.99
26	9	11	2022-11-30	1.29
20	19	13	2022-11-20	1.29
3	1	14	2023-01-20	1.29
13	9	14	2022-11-15	0.99
12	1	19	2022-12-28	1.29
5	9	28	2023-01-05	1.29
10	6	29	2023-01-30	1.29
32	3	36	2023-02-15	1.29

8. **Range Checking**: Find all purchases made between January 1, 2023 and March 31, 2023.

```sql
SELECT *
FROM  purchases
WHERE purchase_date BETWEEN '2023-01-01' AND '2023-03-31'
ORDER BY purchase_date
```

```sql
SELECT *
FROM  purchases
WHERE purchase_date BETWEEN '2023-01-01' AND '2023-03-31'
ORDER BY purchase_date
```

purchase_id	customer_id	song_id	purchase_date	price
5	9	28	2023-01-05	1.29
22	14	31	2023-01-08	1.29
27	16	6	2023-01-12	0.99
1	4	1	2023-01-15	1.29
39	8	34	2023-01-18	0.99
3	1	14	2023-01-20	1.29
31	8	15	2023-01-22	0.99
15	16	8	2023-01-25	0.99
35	20	18	2023-01-28	0.99
10	6	29	2023-01-30	1.29
2	7	3	2023-02-02	0.99
11	14	5	2023-02-05	0.99
28	7	33	2023-02-08	1.29
19	2	24	2023-02-10	0.99
40	3	37	2023-02-12	1.29
32	3	36	2023-02-15	1.29
8	2	9	2023-02-18	0.99
24	1	27	2023-02-22	1.29
36	5	22	2023-02-25	1.29
16	7	21	2023-02-28	1.29
21	6	35	2023-03-01	0.99
33	15	4	2023-03-05	0.99
6	18	17	2023-03-08	0.99
14	4	26	2023-03-12	1.29
29	12	25	2023-03-15	0.99
17	12	2	2023-03-18	0.99
37	17	12	2023-03-20	0.99
9	19	23	2023-03-22	0.99
25	4	20	2023-03-28	0.99

9. **Advanced Filtering with ORDER BY**: Identify the songs with the highest popularity scores (above 90).

SELECT title, popularity_score

FROM songs

WHERE popularity_score >90

ORDER BY popularity_score DESC

Run SQLite

```
1 SELECT title, popularity_score
2 FROM songs
3 WHERE popularity_score >90
4 ORDER BY popularity_score DESC
```

title	popularity_score
Blinding Lights	95
Mr. Brightside	95
Shape of You	94
Levitating	94
God's Plan	93
Anti-Hero	92
Don't Start Now	92
Shallow	92
One Dance	92
Bad Guy	91
Hello	91
Save Your Tears	91

Discussion Questions

1. In our database design, we separated purchases and streams into different tables. What are the benefits of this approach versus having a single "user_interactions" table?

Having separate tables helps reduce data redundancy and eliminate NULL values for columns that would only be relevant to either a purchase or a stream (not both). It also allows for clearer relationships and overall smaller table sizes for faster processing. Lastly, it makes sense from a business perspective to separate purchases and streams in order to better answer more nuanced questions.

2. Based on the provided data model, what business questions could music executives answer using SQL queries that we haven't covered in our exercise?

Other business questions to answer could include:

- Total revenue
- Revenue trends over time
- Which artists or songs drove revenue
- Which customers buy the most songs
- Which customers stream the most
- Popular genres
- Overlapping genres between customers
- Artist popularity

3. How would you extend this schema to track more detailed user behavior, such as when users skip songs or how much of a song they listen to before skipping?

You could add new columns for the following: stream start timestamp, stream end timestamp, stream duration, and if the song was listened all the way to completion or not.

4. For tasks 1-3, how could you combine them into a single, more complex query that finds popular short songs by artists whose names start with "The"?

You could join the tables “songs” and “artists” on artist_id, then filter by artists’ whose names start with “The,” song duration <240 seconds, and popularity score >80.

```
SELECT title, artist_name, duration_seconds, popularity_score

FROM songs JOIN artists ON artists.artist_id = songs.artist_id

WHERE popularity_score > 80 AND duration_seconds < 240 AND artist_name LIKE 'The%'
```

```sql
SELECT title, artist_name, duration_seconds, popularity_score
FROM songs JOIN artists ON artists.artist_id = songs.artist_id
WHERE popularity_score > 80 AND duration_seconds < 240 AND artist_name LIKE 'The%'
```

title	artist_name	duration_seconds	popularity_score
Blinding Lights	The Weeknd	200	95
Die For You	The Weeknd	232	90
Here Comes The Sun	The Beatles	186	85
Paint It Black	The Rolling Stones	202	82
Mr. Brightside	The Killers	224	95
Somebody Told Me	The Killers	197	83
Save Your Tears	The Weeknd	215	91
Yesterday	The Beatles	125	86