Music Streaming SQL- Project

Developed By : Shashi Maurya.

YOP : 2025.

Education : BCA.

Institute : QSpiders.

📖 Project Overview

A comprehensive SQL-based analytics platform that analyzes music streaming data to derive business insights. This project demonstrates real-world data analysis using Oracle SQL.

🎯 Key Features

- **User behavior & engagement analysis**
- **Content performance tracking**
- **Genre-wise market analysis**
- **Premium vs free user comparison**
- **Retention & trend analysis**

📊 Dataset

- **6 normalized tables**
- **Realistic streaming data**

🛠️ Tech Stack

- **Oracle SQL**
- **Advanced SQL Queries**
- **Business Intelligence**

🚀 Business Value

- Helps music companies understand user preferences, optimize content strategy, and improve revenue through data-driven decisions.

TABLES CREATION.

```
SQL> CREATE TABLE users (
  2      user_id NUMBER PRIMARY KEY,
  3      username VARCHAR2(50) UNIQUE NOT NULL,
  4      email VARCHAR2(100) UNIQUE NOT NULL,
  5      country VARCHAR2(50),
  6      premium_member CHAR(1) DEFAULT 'N'
  7  );

Table created.

SQL>
SQL> CREATE TABLE artists (
  2      artist_id NUMBER PRIMARY KEY,
  3      artist_name VARCHAR2(100) NOT NULL,
  4      genre VARCHAR2(50),
  5      monthly_listeners NUMBER DEFAULT 0
  6  );

Table created.

SQL>
SQL> CREATE TABLE songs (
  2      song_id NUMBER PRIMARY KEY,
  3      song_title VARCHAR2(100) NOT NULL,
  4      artist_id NUMBER NOT NULL,
  5      duration_seconds NUMBER,
  6      genre VARCHAR2(50),
  7      total_streams NUMBER DEFAULT 0,
  8      FOREIGN KEY (artist_id) REFERENCES artists(artist_id)
  9  );

Table created.

SQL>
SQL> CREATE TABLE playlists (
  2      playlist_id NUMBER PRIMARY KEY,
  3      playlist_name VARCHAR2(100) NOT NULL,
  4      user_id NUMBER NOT NULL,
  5      created_date DATE DEFAULT SYSDATE,
  6      FOREIGN KEY (user_id) REFERENCES users(user_id)
  7  );

Table created.



SQL>
SQL> CREATE TABLE playlist_songs (
  2      playlist_id NUMBER,
  3      song_id NUMBER,
  4      added_date DATE DEFAULT SYSDATE,
  5      PRIMARY KEY (playlist_id, song_id),
  6      FOREIGN KEY (playlist_id) REFERENCES playlists(playlist_id),
  7      FOREIGN KEY (song_id) REFERENCES songs(song_id)
  8  );

Table created.

SQL>
SQL> CREATE TABLE streaming_history (
  2      stream_id NUMBER PRIMARY KEY,
  3      user_id NUMBER NOT NULL,
  4      song_id NUMBER NOT NULL,
  5      stream_date DATE DEFAULT SYSDATE,
  6      duration_played NUMBER,
  7      completed CHAR(1) DEFAULT 'N',
  8      FOREIGN KEY (user_id) REFERENCES users(user_id),
  9      FOREIGN KEY (song_id) REFERENCES songs(song_id)
 10  );

Table created.
```

INSERT DATA.

```
SQL> -- Insert artists
SQL> INSERT INTO artists VALUES (1, 'Arijit Singh', 'Bollywood', 50000000);

1 row created.

SQL> INSERT INTO artists VALUES (2, 'Taylor Swift', 'Pop', 80000000);

1 row created.

SQL> INSERT INTO artists VALUES (3, 'A.R. Rahman', 'Soundtrack', 45000000);

1 row created.

SQL> INSERT INTO artists VALUES (4, 'The Weeknd', 'R&B', 70000000);
Enter value for b:
old   1: INSERT INTO artists VALUES (4, 'The Weeknd', 'R&B', 70000000)
new   1: INSERT INTO artists VALUES (4, 'The Weeknd', 'R', 70000000)

1 row created.

SQL> -- Insert songs
SQL> INSERT INTO songs VALUES (101, 'Tum Hi Ho', 1, 290, 'Bollywood', 150000000);

1 row created.

SQL> INSERT INTO songs VALUES (102, 'Chahun Main Ya Naa', 1, 330, 'Bollywood', 120000000);

1 row created.

SQL> INSERT INTO songs VALUES (103, 'Anti-Hero', 2, 200, 'Pop', 300000000);

1 row created.

SQL> INSERT INTO songs VALUES (104, 'Jai Ho', 3, 320, 'Soundtrack', 250000000);

1 row created.

SQL> INSERT INTO songs VALUES (105, 'Blinding Lights', 4, 220, 'R&B', 2000000000);
Enter value for b:
old   1: INSERT INTO songs VALUES (105, 'Blinding Lights', 4, 220, 'R&B', 2000000000)
new   1: INSERT INTO songs VALUES (105, 'Blinding Lights', 4, 220, 'R', 2000000000)

1 row created.

SQL> -- Insert users
SQL> INSERT INTO users VALUES (1001, 'amit_music', 'amit@email.com', 'India', 'Y');

1 row created.
```

```
SQL> -- Insert users
SQL> INSERT INTO users VALUES (1001, 'amit_music', 'amit@email.com', 'India', 'Y');

1 row created.

SQL> INSERT INTO users VALUES (1002, 'priya_swift', 'priya@email.com', 'India', 'N');

1 row created.

SQL> INSERT INTO users VALUES (1003, 'rahul_weeknd', 'rahul@email.com', 'USA', 'Y');

1 row created.

SQL>
SQL> -- Insert playlists
SQL> INSERT INTO playlists VALUES (5001, 'My Fav Bollywood', 1001, SYSDATE);

1 row created.

SQL> INSERT INTO playlists VALUES (5002, 'Workout Mix', 1002, SYSDATE);

1 row created.

SQL>
SQL> -- Insert playlist songs
SQL> INSERT INTO playlist_songs VALUES (5001, 101, SYSDATE);

1 row created.

SQL> INSERT INTO playlist_songs VALUES (5001, 102, SYSDATE);

1 row created.

SQL> INSERT INTO playlist_songs VALUES (5002, 105, SYSDATE);
```

```
SQL> -- Insert users
SQL> INSERT INTO users VALUES (1001, 'amit_music', 'amit@email.com', 'India', 'Y');

1 row created.

SQL> INSERT INTO users VALUES (1002, 'priya_swift', 'priya@email.com', 'India', 'N');

1 row created.

SQL> INSERT INTO users VALUES (1003, 'rahul_weeknd', 'rahul@email.com', 'USA', 'Y');

1 row created.

SQL>
SQL> -- Insert playlists
SQL> INSERT INTO playlists VALUES (5001, 'My Fav Bollywood', 1001, SYSDATE);

1 row created.

SQL> INSERT INTO playlists VALUES (5002, 'Workout Mix', 1002, SYSDATE);

1 row created.

SQL>
SQL> -- Insert playlist songs
SQL> INSERT INTO playlist_songs VALUES (5001, 101, SYSDATE);

1 row created.

SQL> INSERT INTO playlist_songs VALUES (5001, 102, SYSDATE);

1 row created.

SQL> INSERT INTO playlist_songs VALUES (5002, 105, SYSDATE);

1 row created.

SQL>
SQL> -- Insert streaming history
SQL> INSERT INTO streaming_history VALUES (6001, 1001, 101, SYSDATE, 290, 'Y');

1 row created.

SQL> INSERT INTO streaming_history VALUES (6002, 1001, 102, SYSDATE-1, 200, 'N');

1 row created.

SQL> INSERT INTO streaming_history VALUES (6003, 1002, 103, SYSDATE-2, 200, 'Y');
```

```
1 row created.

SQL> INSERT INTO streaming_history VALUES (6003, 1002, 103, SYSDATE-2, 200, 'Y');

1 row created.

SQL> INSERT INTO streaming_history VALUES (6004, 1003, 105, SYSDATE, 220, 'Y');

1 row created.

SQL> INSERT INTO streaming_history VALUES (6005, 1001, 105, SYSDATE, 220, 'Y');

1 row created.

SQL>
SQL> COMMIT;

Commit complete.
```

QUESTION SLOVING.

```
SQL> -- Top 5 most streamed songs
SQL> SELECT
  2      s.song_title,
  3      a.artist_name,
  4      s.total_streams,
  5      ROUND(s.duration_seconds/60, 2) as duration_minutes
  6  FROM songs s
  7  JOIN artists a ON s.artist_id = a.artist_id
  8  ORDER BY s.total_streams DESC
  9  FETCH FIRST 5 ROWS ONLY;
FETCH FIRST 5 ROWS ONLY
*
```

```
SQL>
SQL> -- Artist popularity ranking
SQL> SELECT
  2     artist_name,
  3     genre,
  4     monthly_listeners,
  5     (SELECT COUNT(*) FROM songs WHERE artist_id = a.artist_id) as total_songs,
  6     (SELECT SUM(total_streams) FROM songs WHERE artist_id = a.artist_id) as total_streams
  7  FROM artists a
  8  ORDER BY monthly_listeners DESC;
```

```
SQL>
SQL> -- User listening statistics
SQL> SELECT
  2      u.username,
  3      u.premium_member,
  4      COUNT(sh.stream_id) as total_streams,
  5      COUNT(DISTINCT sh.song_id) as unique_songs_played,
  6      SUM(sh.duration_played) as total_seconds_listened
  7  FROM users u
  8  LEFT JOIN streaming_history sh ON u.user_id = sh.user_id
  9  GROUP BY u.user_id, u.username, u.premium_member
 10  ORDER BY total_seconds_listened DESC;
```

```
SQL> SET PAGES 100 LINES 100;
SQL> -- Genre-wise market analysis
SQL> SELECT
  2      genre,
  3      COUNT(*) as total_songs,
  4      SUM(total_streams) as total_streams,
  5      ROUND(AVG(total_streams), 0) as avg_streams_per_song,
  6      ROUND(SUM(total_streams) * 100.0 / (SELECT SUM(total_streams) FROM songs), 2) as market_sha
re_percent
  7  FROM songs
  8  GROUP BY genre
  9  ORDER BY total_streams DESC;
```

```
SQL>
SQL> -- Premium vs Free user comparison
SQL> SELECT
  2      premium_member,
  3      COUNT(DISTINCT u.user_id) as user_count,
  4      ROUND(AVG(stream_count), 0) as avg_streams_per_user,
  5      ROUND(AVG(total_seconds)/3600, 2) as avg_hours_per_user
  6  FROM users u
  7  JOIN (
  8      SELECT user_id, COUNT(*) as stream_count, SUM(duration_played) as total_seconds
  9      FROM streaming_history
 10      GROUP BY user_id
 11  ) sh ON u.user_id = sh.user_id
 12  GROUP BY premium_member;
```

```
SQL>
SQL> -- User engagement ranking
SQL> SELECT
  2      u.username,
  3      u.country,
  4      u.premium_member,
  5      COUNT(sh.stream_id) as total_streams,
  6      COUNT(DISTINCT sh.song_id) as unique_songs,
  7      COUNT(DISTINCT p.playlist_id) as playlists_created,
  8      RANK() OVER (ORDER BY COUNT(sh.stream_id) DESC) as engagement_rank
  9  FROM users u
 10  LEFT JOIN streaming_history sh ON u.user_id = sh.user_id
 11  LEFT JOIN playlists p ON u.user_id = p.user_id
 12  GROUP BY u.user_id, u.username, u.country, u.premium_member
 13  ORDER BY engagement_rank;
```

```
SQL> -- Most popular songs by genre
SQL> SELECT
  2      genre,
  3      song_title,
  4      artist_name,
  5      total_streams,
  6      RANK() OVER (PARTITION BY genre ORDER BY total_streams DESC) as genre_rank
  7  FROM songs s
  8  JOIN artists a ON s.artist_id = a.artist_id
  9  WHERE genre IS NOT NULL
 10  QUALIFY RANK() OVER (PARTITION BY genre ORDER BY total_streams DESC) <= 3
 11  ORDER BY genre, genre_rank;
QUALIFY RANK() OVER (PARTITION BY genre ORDER BY total_streams DESC) <= 3
*
```

```
SQL>
SQL> -- Users who might like similar songs (basic recommendation)
SQL> SELECT DISTINCT
  2      s2.song_title,
  3      s2.artist_name,
  4      s2.genre,
  5      s2.total_streams
  6  FROM streaming_history sh1
  7  JOIN streaming_history sh2 ON sh1.user_id = sh2.user_id AND sh1.song_id != sh2.song_id
  8  JOIN songs s1 ON sh1.song_id = s1.song_id
  9  JOIN songs s2 ON sh2.song_id = s2.song_id
 10  JOIN artists a ON s2.artist_id = a.artist_id
 11  WHERE sh1.user_id = 1001  -- Amit's ID
 12  AND s2.song_id NOT IN (
 13      SELECT song_id FROM streaming_history WHERE user_id = 1001
 14  )
 15  ORDER BY s2.total_streams DESC
 16  FETCH FIRST 10 ROWS ONLY;
FETCH FIRST 10 ROWS ONLY
```

```
SQL>
SQL> -- Artist cross-promotion opportunities
SQL> SELECT
  2      a1.artist_name as artist_A,
  3      a2.artist_name as artist_B,
  4      COUNT(DISTINCT u.user_id) as common_listeners,
  5      ROUND(COUNT(DISTINCT u.user_id) * 100.0 / (
  6          SELECT COUNT(*) FROM user_followers WHERE artist_id = a1.artist_id
  7      ), 2) as overlap_percent
  8  FROM user_followers uf1
  9  JOIN user_followers uf2 ON uf1.user_id = uf2.user_id AND uf1.artist_id != uf2.artist_id
 10  JOIN artists a1 ON uf1.artist_id = a1.artist_id
 11  JOIN artists a2 ON uf2.artist_id = a2.artist_id
 12  WHERE a1.artist_id = 1  -- Arijit Singh
 13  GROUP BY a1.artist_name, a2.artist_name, a1.artist_id
 14  ORDER BY common_listeners DESC
 15  FETCH FIRST 5 ROWS ONLY;
FETCH FIRST 5 ROWS ONLY
```

```
SQL> -- Daily streaming trends (last 7 days)
SQL> SELECT
  2      TRUNC(stream_date) as streaming_date,
  3      COUNT(*) as total_streams,
  4      COUNT(DISTINCT user_id) as daily_active_users,
  5      ROUND(AVG(duration_played), 0) as avg_session_duration
  6  FROM streaming_history
  7  WHERE stream_date >= SYSDATE - 7
  8  GROUP BY TRUNC(stream_date)
  9  ORDER BY streaming_date DESC;
```

```
SQL>
SQL> -- User retention analysis
SQL> SELECT
  2      'Last 7 days' as period,
  3      COUNT(DISTINCT user_id) as active_users,
  4      (SELECT COUNT(*) FROM users) as total_users,
  5      ROUND(COUNT(DISTINCT user_id) * 100.0 / (SELECT COUNT(*) FROM users), 2) as retention_rate
  6  FROM streaming_history
  7  WHERE stream_date >= SYSDATE - 7
  8
```

```
SQL> SELECT
  2      'Last 30 days' as period,
  3      COUNT(DISTINCT user_id) as active_users,
  4      (SELECT COUNT(*) FROM users) as total_users,
  5      ROUND(COUNT(DISTINCT user_id) * 100.0 / (SELECT COUNT(*) FROM users), 2) as retention_rate
  6  FROM streaming_history
  7  WHERE stream_date >= SYSDATE - 30;
    (SELECT COUNT(*) FROM users) as total_users,
```

```
SQL>
SQL> -- Content performance by duration
SQL> SELECT
  2      CASE
  3          WHEN duration_seconds < 180 THEN 'Short (<3 min)'
  4          WHEN duration_seconds BETWEEN 180 AND 300 THEN 'Medium (3-5 min)'
  5          ELSE 'Long (>5 min)'
  6      END as duration_category,
  7      COUNT(*) as song_count,
  8      ROUND(AVG(total_streams), 0) as avg_streams,
  9      SUM(total_streams) as total_streams
 10  FROM songs
 11  GROUP BY
 12      CASE
 13          WHEN duration_seconds < 180 THEN 'Short (<3 min)'
 14          WHEN duration_seconds BETWEEN 180 AND 300 THEN 'Medium (3-5 min)'
 15          ELSE 'Long (>5 min)'
 16      END
 17  ORDER BY total_streams DESC;
```