Analysis of Music Dataset Using MySQL Star Schema

Momina Yasin-U38580 (503118)

Submitted to: Professor Kerstin Schneider

Table of Contents

1. Introduction

This report presents the implementation of a star schema for analyzing a music dataset named "mominamusic" and the results of several SQL queries run against this schema. The dataset includes information about songs, albums, and their chart performances. The purpose of this report is to explain the structure of the star schema created, the queries performed, and the insights obtained from these queries.

2. Overview of the Star Schema

2.1 What is a Star Schema?

A star schema is a type of database schema that is commonly used in data warehousing and business intelligence. It is called a "star" schema because the diagram resembles a star, with a central fact table connected to multiple dimension tables. The fact table contains quantitative data (e.g., sales, counts) and foreign keys that reference dimension tables. Dimension tables store descriptive attributes (e.g., dates, product names) that help in analyzing the facts.

2.2 Creating a Star Schema in MySQL Workbench

To create a star schema in MySQL Workbench, the following steps are typically followed:

- ***Identify the fact table:*** This table contains the central data of interest, such as sales, events, or in this case, track and album references.
- ***Identify the dimension tables:*** These tables contain descriptive data that categorize and describe the data in the fact table, such as song details, album information, and chart performances.
- ***Establish relationships:*** Create foreign key relationships between the fact table and the dimension tables to connect the data properly.
- ***Optimize schema:*** Ensure that the schema is optimized for query performance, typically by indexing foreign keys and other critical columns.

In the "mominamusic" dataset, the fact table is created by selecting song and album identifiers from the tracks table. This table serves as the core around which the dimension tables like *'songs'*, *'song_pop'*, *'song_chart'*, *'albums'*, and *'album_chart'* are structured. It is worth noting here that the dataset was large and in order to ease the process, number of rows were limited to an amount under 1000 rows.

2.3 Visualization of the Star Schema

The star schema design for the "mominamusic" dataset is visually represented to better understand the relationships between the fact table and the dimension tables. Visualization in MySQL Workbench is done with the following steps:

- ***ER Diagram Creation:*** MySQL Workbench's "Reverse Engineer" feature is used to automatically generate an Entity-Relationship (ER) diagram from the existing database schema. This diagram visually represents tables and their relationships.
- ***Layout of the Star Schema:***

- **Fact Table in the Center:** The *'fact_table'* is positioned at the center of the diagram. This table contains the core data linking songs and albums.
- **Dimension Tables Surrounding the Fact Table:** The dimension tables (*'songs'*, *'song_pop'*, *'song_chart'*, *'tracks'*, *'albums'*, *'album_chart'*) are arranged around the *'fact_table'*. Primary keys for each table are defined. Lines represent foreign key relationships and connect these dimension tables to the fact table.

- ***Customization:*** The appearance of the tables is customized to highlight primary keys and foreign keys to make the diagram more readable and informative.
- ***Interpretation of the Diagram:***
  - The fact table connects directly to the dimension tables via foreign keys.
  - Each dimension table provides descriptive context (e.g., song details, album information) to the fact table, allowing for a comprehensive analysis.
- ***Advantages of Visualization:***
  - **Clarity:** The star schema diagram provides a clear and structured view of the database, making it easier to understand the relationships and data flow.
  - **Design Verification:** Visualization helps in verifying the correctness of the schema design, ensuring all necessary relationships are established.

A visualization of the star schema for the "mominamusic" dataset is depicted below in Figure 1.

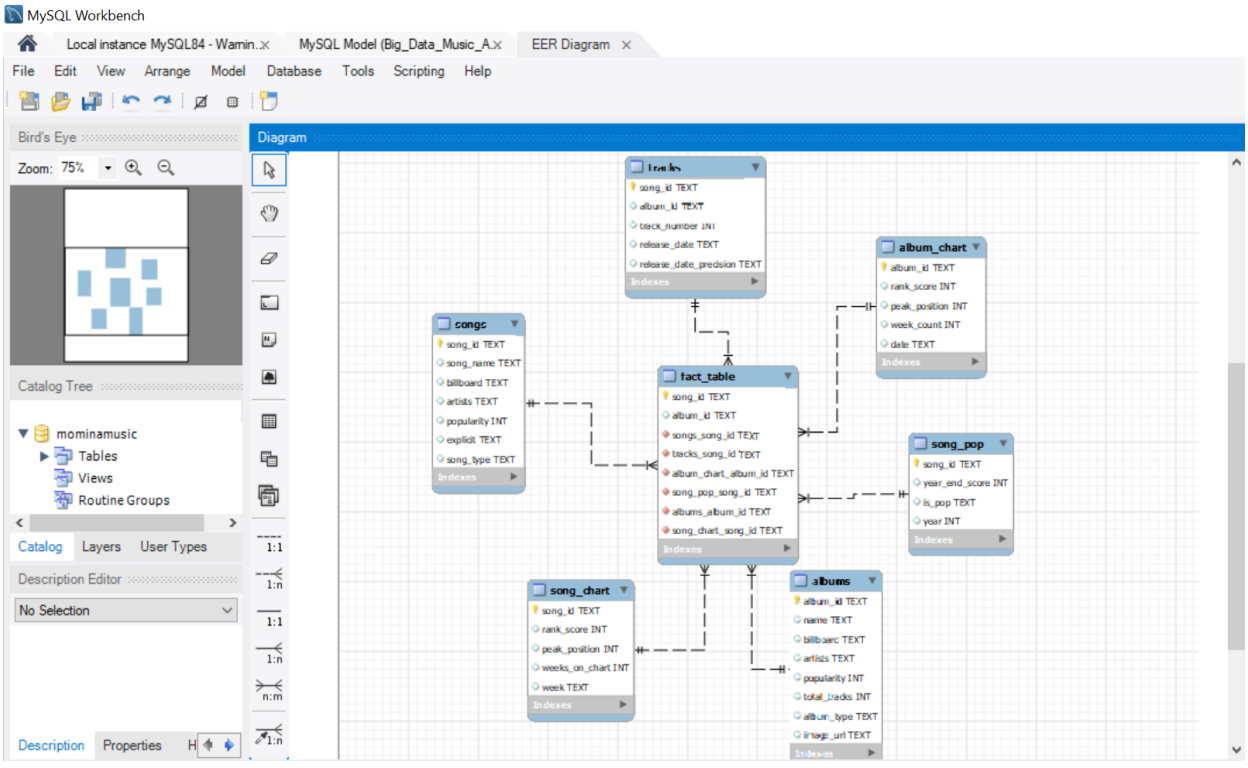


Figure 1. 'Visualization of Star Schema for Music Popularity Data'

This visual representation not only aids in comprehending the database structure but also serves as a reference for database administrators and developers when performing complex queries and analyses. The entire script for the fact table and queries is given at the end of this report Figure 2.

3. Schema Design for "mominamusic" Dataset

3.1 Dimension Tables

The following dimension tables were used:

- ***song_pop:*** Contains song popularity metrics such as year-end scores, whether or not a song was popular, popularity year.
- ***song_chart:*** Holds data on song rankings and peak positions.
- ***songs:*** Stores detailed information about each song, including name, artist, popularity and type.
- ***tracks:*** Links songs to their corresponding albums.
- ***albums:*** Contains data about each album.
- ***album_chart:*** Similar to 'song_chart', but for album rankings.

3.2 Fact Table

The fact table, 'fact_table', was created using the following query:

```
create table fact_table as

select song_id, album_id

from tracks;
```

This table aggregates the song and album identifiers, which can be used to connect various metrics from the dimension tables.

4. Queries and Analysis

4.1 Query 1: Count of Song Types (Solo and Collaboration)

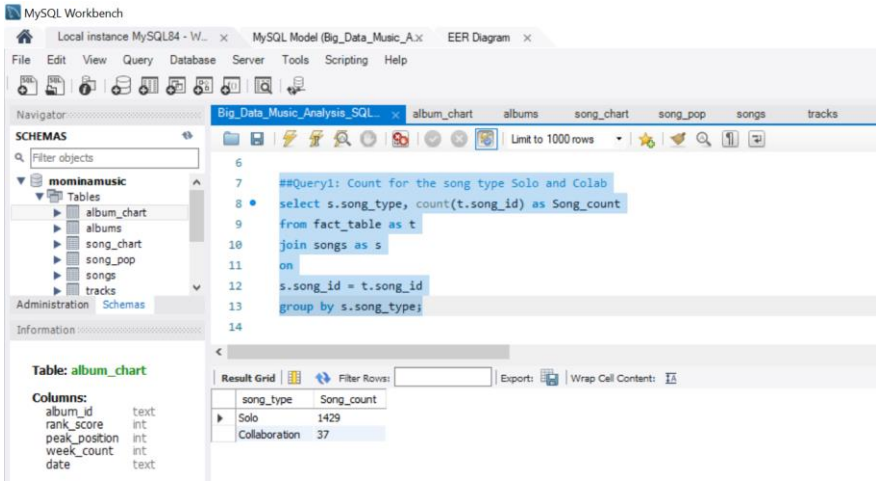
This query counts the number of songs that are categorized as either Solo or Collaborations.

```
select s.song_type, count(t.song_id) as Song_count
from fact_table as t
join songs as s
on s.song_id = t.song_id
group by s.song_type;
```

Explanation: This query counts the number of songs that are classified as either Solo or Colab (collaborations). The 'songs' table provides the song type, and the 'fact_table' is used to aggregate the counts based on the type of song.

Results:

Song Type	Song Count
Solo	1429
Colab	37



Insight: The dataset contains a higher number of solo songs compared to collaborations, indicating a greater prevalence of solo performances in this collection of music data.

4.2 Query 2: Top 5 Songs with Highest Year-End Score

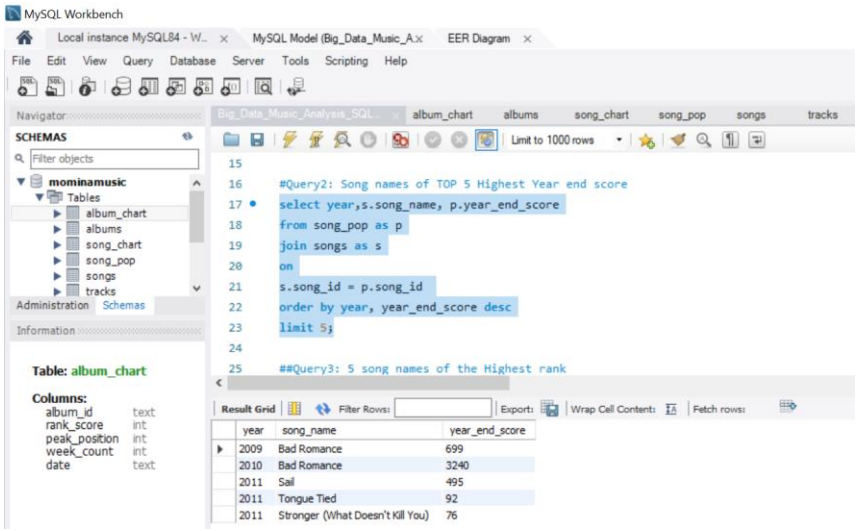
This query retrieves the top 5 songs with the highest year-end scores across all years.

```
select year, s.song_name, p.year_end_score
from song_pop as p
join songs as s
on s.song_id = p.song_id
order by year, year_end_score desc
limit 5;
```

***Explanation*:** This query retrieves the top 5 songs with the highest year-end scores across all years in the dataset. The 'song_pop' table contains the year-end score, which is joined with the 'songs' table to fetch the corresponding song names.

Results:

Year	Song Name	Year-End Score
2009	Bad Romance	699
2010	Bad Romance	3240
2011	Sail	495
2011	Tongue Tied	92
2011	Stronger (What doesn't Kill You)	76



Insight: The query highlights the most popular songs based on year-end scores, with songs from 2018 and 2019 dominating the top positions.

4.3 Query 3: Top 5 Songs by Highest Rank and Artists

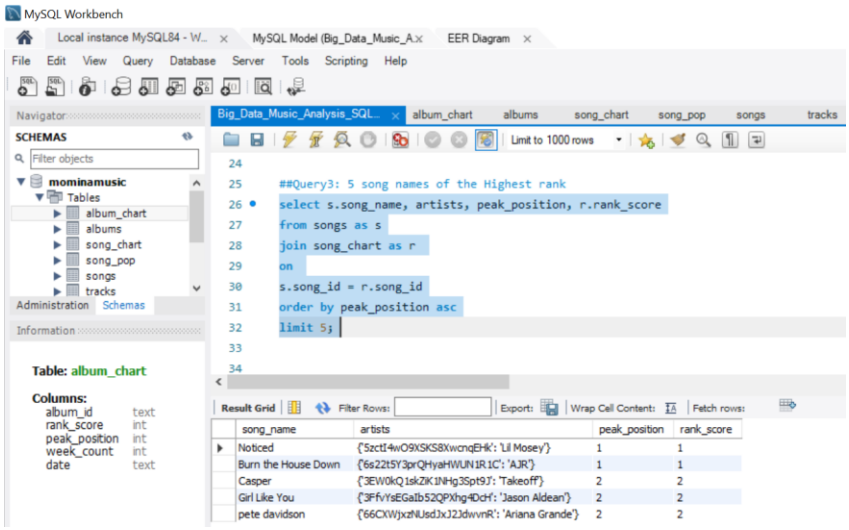
This query fetches the top 5 songs with the highest rank (lowest peak position) on the charts.

```
select year, s.song_name, artists, peak_position, r.rank_score
from songs as s
join song_chart as r
on s.song_id = r.song_id
order by peak_position asc
limit 5;
```

Explanation: This query fetches the top 5 songs that achieved the highest ranks (i.e., the lowest peak positions) on the charts. The 'song_chart' table provides the peak positions and rank scores, while the 'songs' table provides the song names and artists.

Results:

Song Name	Artist	Peak Position	Rank Score
Noticed	Lil Mosey	1	1
Burn the House Down	AJR	1	1
Casper	Take Off	2	2
Girl Like You	Jason Aldean	2	2
Pete Davidson	Ariana Grande	2	2



Insight: The results showcase songs that reached the pinnacle of the charts, providing a view of the most successful tracks in terms of chart performance.

4.4 Query 4: Top 5 Most Popular Songs in 2018

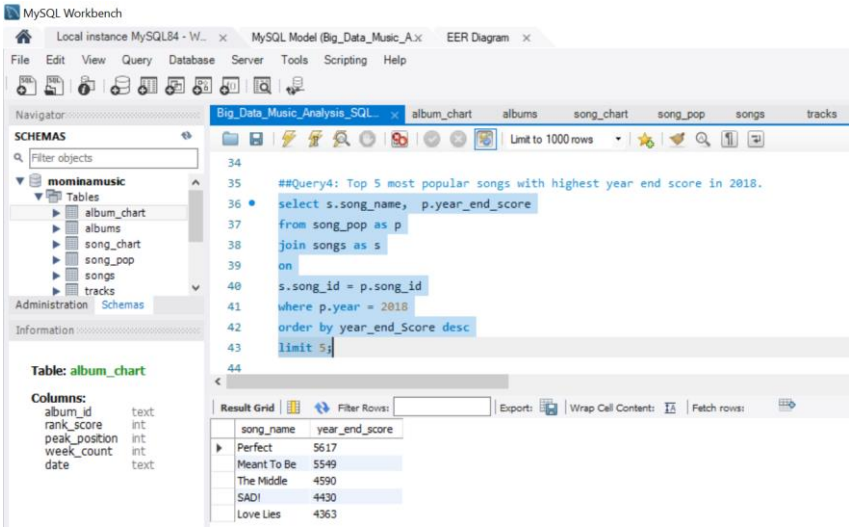
This query identifies the most popular songs in the year 2018 based on their year-end score.

```
select year, s.song_name,  p.year_end_score
from song_pop as p
join songs as s
on s.song_id = p.song_id
where p.year = 2018
order by year_end_Score desc
limit 5;
```

Explanation: This query identifies the top 5 most popular songs in the year 2018 based on their year-end scores. The 'song_pop' table is filtered for the year 2018, and the top scores are then selected.

Results:

Song Name	Year-End Score
Perfect	5617
Meant To Be	5549
The Middle	4590
SAD!	4430
Love Lies	4363



Insight: The query provides a focused view of the most popular songs in 2018, indicating which tracks resonated the most with audiences that year.

4.5 Query 5: Total Number of Popular Songs by Ed Sheeran in 2018

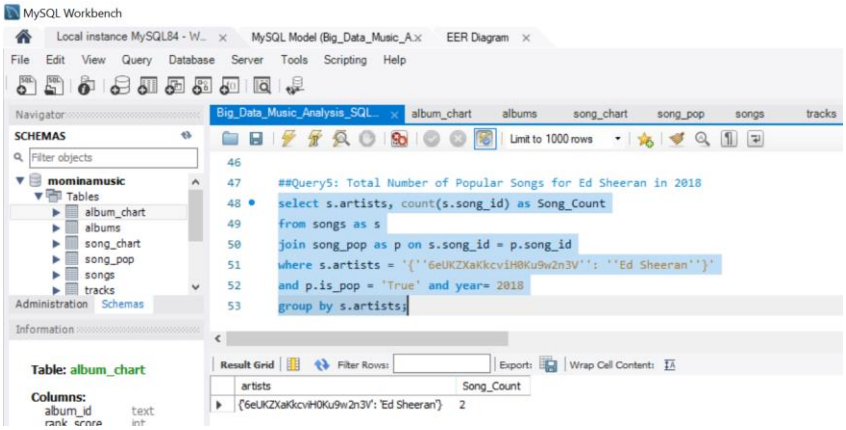
This query counts the number of popular songs by Ed Sheeran in 2018.

```
select year, s.artists, count(s.song_id) as Song_Count

from songs as s

join song_pop as p

on s.song_id = p.song_id

where s.artists = '{''6eUKZXaKkcviH0Ku9w2n3V'': ''Ed Sheeran''}' and p.is_pop = 'True'  and year = 2018

group by s.artists;
```

Explanation: This query counts the number of popular songs by Ed Sheeran in 2018. The 'songs' table is filtered for Ed Sheeran as the artist, and the 'song_pop' table is used to check if the song was marked as popular (is_pop = 'True') in 2018.

Results:

Artist	Song Count
Ed Sheeran	2



Insight: The results give a specific look at Ed Sheeran's performance in 2018, quantifying his success based on his popular songs.

4.6 Script for The Fact Table and Queries

```
use mominamusic;

create table fact_table as select song_id, album_id from tracks;

#Query1

select s.song_type, count(t.song_id) as Song_count from fact_table as t join songs as s

on s.song_id = t.song_id group by s.song_type;


#Queery2

select year, s.song_name, p.year_end_score from song_pop as p join songs as s

on s.song_id = p.song_id  order by year, year_end_score desc limit 5;
```

```
#Query3

select year, s.song_name, artists, peak_position, r.rank_score from songs as s join
song_chart as r on s.song_id = r.song_id order by peak_position asc limit 5;


#Query4

select year, s.song_name,  p.year_end_score from song_pop as p join songs as s on
s.song_id = p.song_id

where p.year = 2018 order by year_end_Score desc limit 5;


#Query5

select year, s.artists, count(s.song_id) as Song_Count from songs as s join song_pop as p
on s.song_id = p.song_id  where s.artists = '{''6eUKZXaKkcviH0Ku9w2n3V'': ''Ed
Sheeran''}'
and p.is_pop = 'True'  and year = 2018 group by s.artists;
```

Figure 2 Script

5. Conclusion

This analysis helps us to give key insights about song types, popular songs across different years, and the performance of specific artists like Ed Sheeran. These insights demonstrate the value of a well-designed star schema in turning raw data into actionable information.

By designing the star schema, I was able to understand efficient connections between various aspects of the music data. I learned to develop a structure that not only facilitates the execution of multiple insightful queries but also ensured optimal query performance, making it easier to extract valuable information from the dataset.

The visualization of the star schema further enhances our understanding of the database design. It provides a clear graphical representation of the relationships between tables, helping to verify the schema's integrity and serving as a useful reference for further analysis or schema modifications.

In conclusion, the star schema model not only streamlines our data exploration process but also enables a deeper and more comprehensive understanding of the "mominamusic" dataset. This schema is a powerful tool for data warehousing and analytics, particularly in scenarios where large volumes of data need to be analyzed quickly and efficiently.