FORE School of Management
New Delhi

PGDM- Big Data Analytics

Big Data Management & Analytics

Project Report

SQL Operations – Music Label Database

Submitted to: Prof. Amarnath Mitra

Submitted By: Group 6

Krishnendu Adhikary – 055022

Mohit Agarwal – 055024

Introduction

This report presents a comprehensive analysis of SQL operations performed in MySQL Workbench, focusing on the implementation, management, and optimization of a Music Streaming Database. The study follows a structured approach, documenting key database creation steps, table structures, normalization checks, data integrity validations, stress testing, query execution, and managerial insights derived from these operations.

The goal of this report is to evaluate database efficiency, ensure referential integrity, optimize query performance, and provide meaningful business insights. By analysing real-world data interactions, the report highlights the effectiveness of relational databases in handling complex relationships, enforcing constraints, and optimizing user interactions within a music streaming platform. The findings from this analysis will contribute to the enhancement of data management practices and system scalability.

Data Description

The database is designed to store and manage information related to users, playlists, tracks, albums, artists, music labels, and contracts. It is structured as a relational database, ensuring data normalization, referential integrity, and efficient querying.

- **User Table:** Stores user details, including UserID, Name, Email, and SubscriptionType.
- **MusicLabel Table:** Contains music label details like LabelID, Name, and EstablishedYear.
- **Artist Table:** Maintains records of artists with ArtistID, StageName, and associated labels.
- **Contract Table:** Defines contractual relationships between ArtistID and LabelID.
- **Album Table:** Captures album-related information, including AlbumID, Title, and Genre.
- **Track Table:** Contains song records, each linked to an album.
- **Playlist Table:** Stores details of user-created playlists.
- **PlaylistTrack Table:** Maintains the many-to-many relationship between PlaylistID and TrackID.

Objectives

The primary objective of this study is to analyse the functionality, efficiency, and integrity of the database by:

1. Creating and managing structured tables based on an **ER diagram**.
2. Ensuring normalization by checking dependencies and avoiding anomalies.
3. Testing database operations such as **INSERT, UPDATE, and DELETE**.
4. Performing stress tests to analyse database stability and cascading effects.
5. Extracting managerial insights using advanced SQL queries.

Major Problem Statement

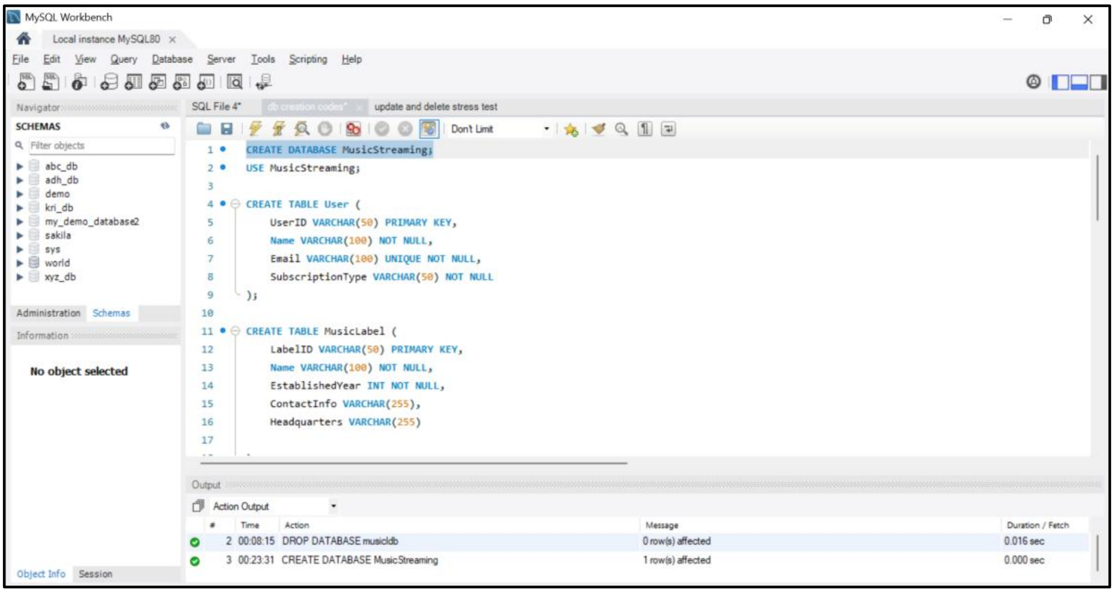
Despite the well-defined structure, several challenges were encountered during database operations:

- **Primary Key Constraints:** Duplicate entries caused insertion failures.
- **Normalization Issues:** The presence of partial dependencies required normalization checks.
- **Cascading Effects:** Deleting records had unintended consequences, requiring referential integrity validation.
- **Performance Optimization:** Queries had to be optimized for efficient data retrieval.

Project Operations

The following operations were performed as part of the database implementation and testing process:

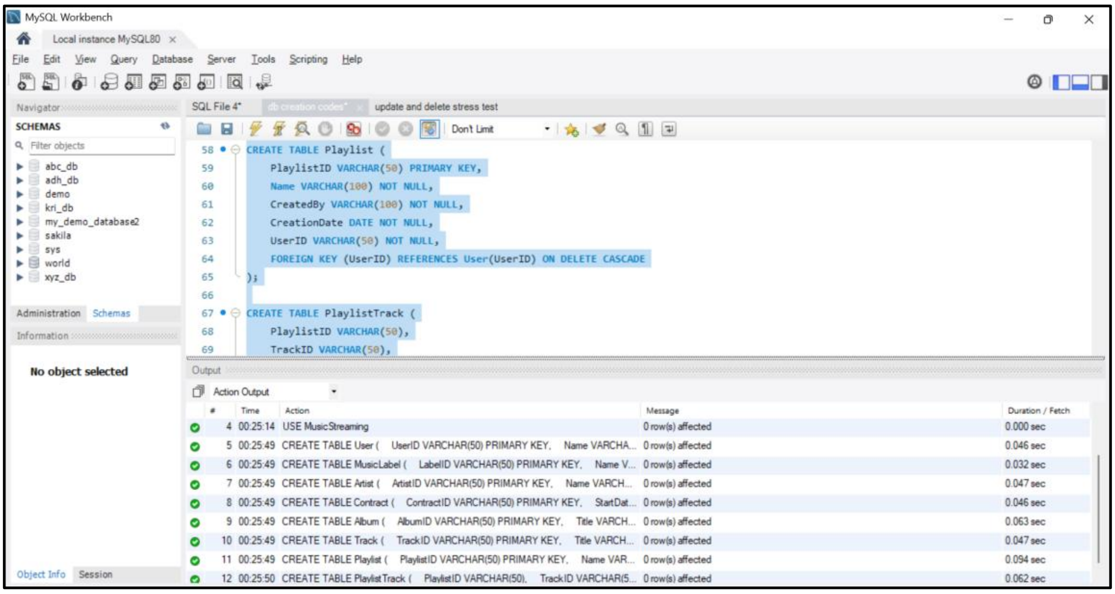
1. **Database Creation:** Setting up the database for the music streaming platform.



```sql
CREATE DATABASE MusicStreaming;
USE MusicStreaming;

CREATE TABLE User (
    UserID VARCHAR(50) PRIMARY KEY,
    Name VARCHAR(100) NOT NULL,
    Email VARCHAR(100) UNIQUE NOT NULL,
    SubscriptionType VARCHAR(50) NOT NULL
);

CREATE TABLE MusicLabel (
    LabelID VARCHAR(50) PRIMARY KEY,
    Name VARCHAR(100) NOT NULL,
    EstablishedYear INT NOT NULL,
    ContactInfo VARCHAR(255),
    Headquarters VARCHAR(255)

```

#	Time	Action	Message	Duration / Fetch
2	00:08:15	DROP DATABASE musicldb	0 row(s) affected	0.016 sec
3	00:23:31	CREATE DATABASE MusicStreaming	1 row(s) affected	0.000 sec

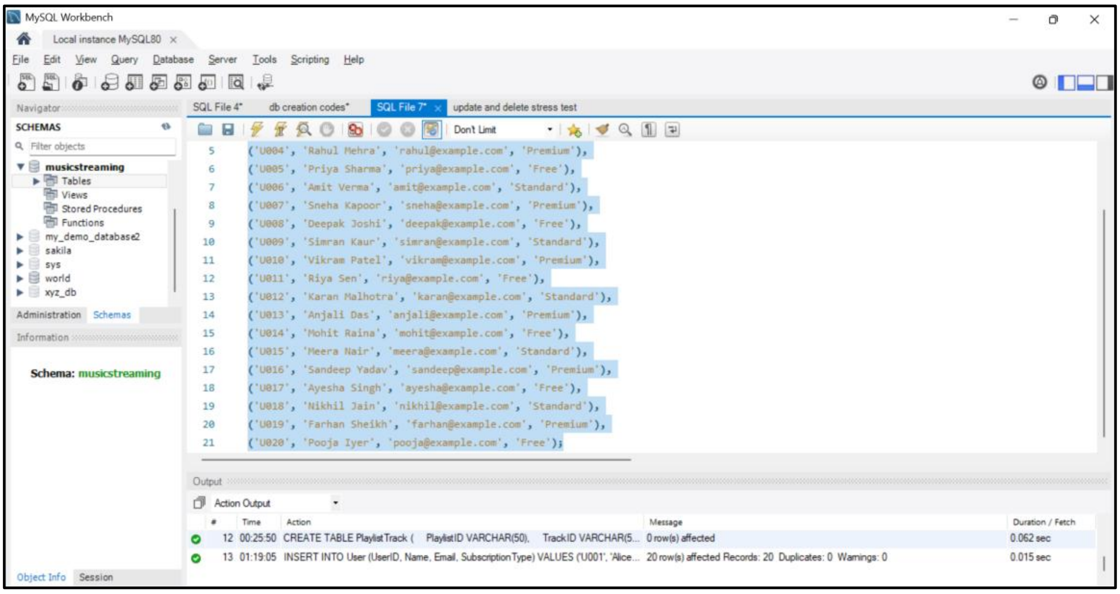
2. **Table Creation:** Designing and implementing tables as per the ER diagram.



```sql
CREATE TABLE Playlist (
    PlaylistID VARCHAR(50) PRIMARY KEY,
    Name VARCHAR(100) NOT NULL,
    CreatedBy VARCHAR(100) NOT NULL,
    CreationDate DATE NOT NULL,
    UserID VARCHAR(50) NOT NULL,
    FOREIGN KEY (UserID) REFERENCES User(UserID) ON DELETE CASCADE
);

CREATE TABLE PlaylistTrack (
    PlaylistID VARCHAR(50),
    TrackID VARCHAR(50),
```

#	Time	Action	Message	Duration / Fetch
4	00:25:14	USE MusicStreaming	0 row(s) affected	0.000 sec
5	00:25:49	CREATE TABLE User (UserID VARCHAR(50) PRIMARY KEY, Name VARCHA...	0 row(s) affected	0.046 sec
6	00:25:49	CREATE TABLE MusicLabel (LabelID VARCHAR(50) PRIMARY KEY, Name V...	0 row(s) affected	0.032 sec
7	00:25:49	CREATE TABLE Artist (ArtistID VARCHAR(50) PRIMARY KEY, Name VARCH...	0 row(s) affected	0.047 sec
8	00:25:49	CREATE TABLE Contract (ContractID VARCHAR(50) PRIMARY KEY, StartDat...	0 row(s) affected	0.046 sec
9	00:25:49	CREATE TABLE Album (AlbumID VARCHAR(50) PRIMARY KEY, Title VARCH...	0 row(s) affected	0.063 sec
10	00:25:49	CREATE TABLE Track (TrackID VARCHAR(50) PRIMARY KEY, Title VARCH...	0 row(s) affected	0.047 sec
11	00:25:49	CREATE TABLE Playlist (PlaylistID VARCHAR(50) PRIMARY KEY, Name VAR...	0 row(s) affected	0.094 sec
12	00:25:50	CREATE TABLE PlaylistTrack (PlaylistID VARCHAR(50), TrackID VARCHAR(5...	0 row(s) affected	0.062 sec

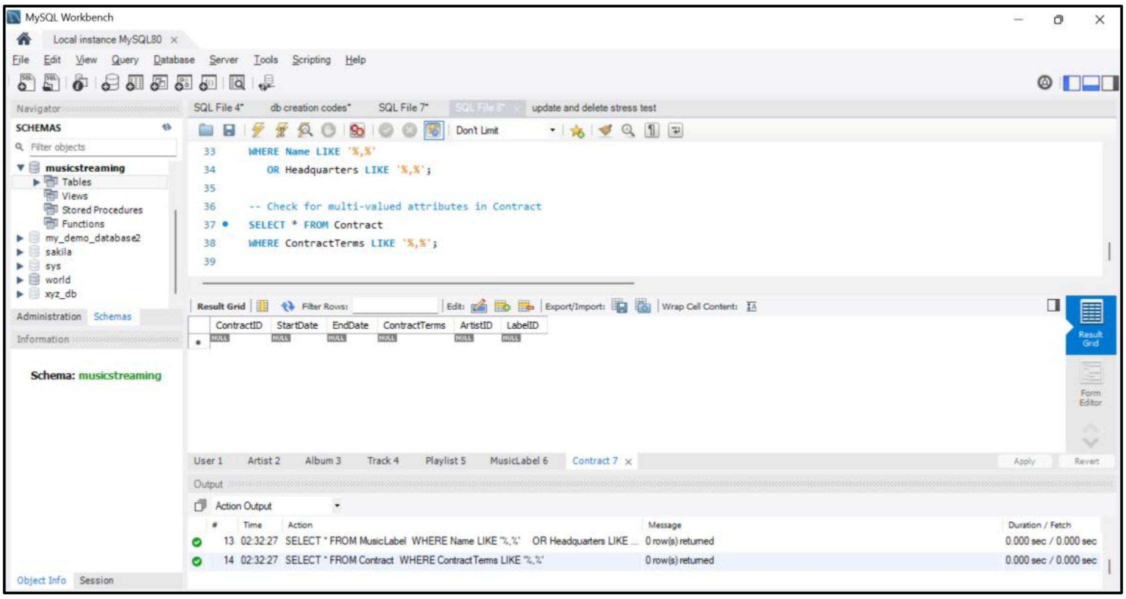
3. **Data Insertion:** Populating tables with dummy data across multiple entities (User, MusicLabel, Artist, Contract, Album, Track, Playlist, PlaylistTrack).



```sql
('U004', 'Rahul Mehra', 'rahul@example.com', 'Premium'),
('U005', 'Priya Sharma', 'priya@example.com', 'Free'),
('U006', 'Amit Verma', 'amit@example.com', 'Standard'),
('U007', 'Sneha Kapoor', 'sneha@example.com', 'Premium'),
('U008', 'Deepak Joshi', 'deepak@example.com', 'Free'),
('U009', 'Simran Kaur', 'simran@example.com', 'Standard'),
('U010', 'Vikram Patel', 'vikram@example.com', 'Premium'),
('U011', 'Riya Sen', 'riya@example.com', 'Free'),
('U012', 'Karan Malhotra', 'karan@example.com', 'Standard'),
('U013', 'Anjali Das', 'anjali@example.com', 'Premium'),
('U014', 'Mohit Raina', 'mohit@example.com', 'Free'),
('U015', 'Meera Nair', 'meera@example.com', 'Standard'),
('U016', 'Sandeep Yadav', 'sandeep@example.com', 'Premium'),
('U017', 'Ayesha Singh', 'ayesha@example.com', 'Free'),
('U018', 'Nikhil Jain', 'nikhil@example.com', 'Standard'),
('U019', 'Farhan Sheikh', 'farhan@example.com', 'Premium'),
('U020', 'Pooja Iyer', 'pooja@example.com', 'Free');
```

```
#   Time      Action                                                                      Message                                                       Duration / Fetch
12  00:25:50  CREATE TABLE PlaylistTrack (    PlaylistID VARCHAR(50),    TrackID VARCHAR(5...  0 row(s) affected                                             0.062 sec
13  01:19:05  INSERT INTO User (UserID, Name, Email, SubscriptionType) VALUES ('U001', 'Alice...  20 row(s) affected Records: 20  Duplicates: 0  Warnings: 0   0.015 sec
```

4. **Normalization Checks:** Verifying compliance with First Normal Form (1NF) and Second Normal Form (2NF) to eliminate anomalies.

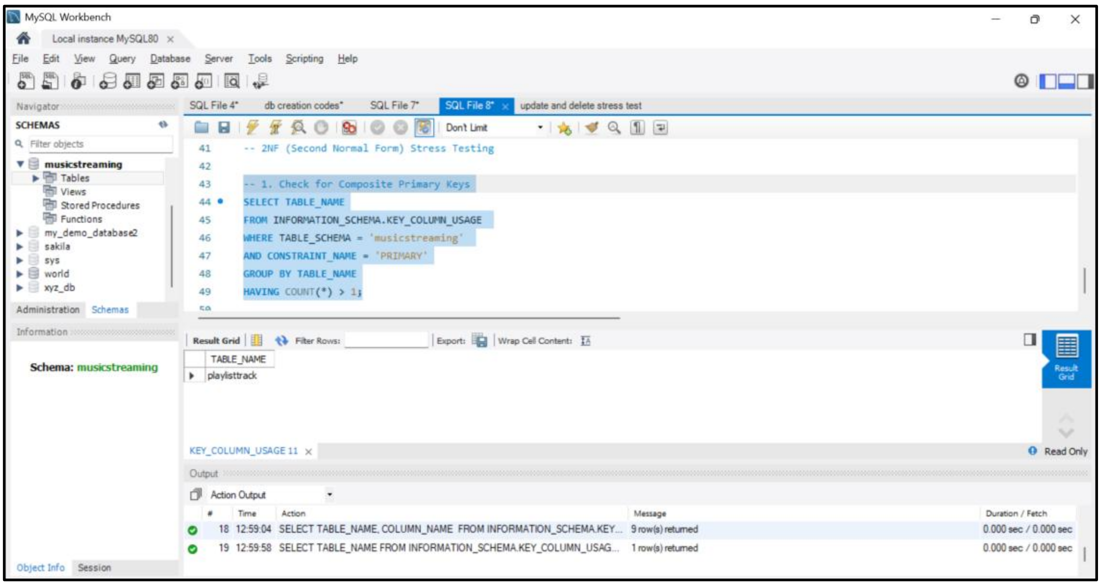


```sql
WHERE Name LIKE '%,%'
   OR Headquarters LIKE '%,%';

-- Check for multi-valued attributes in Contract
SELECT * FROM Contract
WHERE ContractTerms LIKE '%,%';

```

ContractID	StartDate	EndDate	ContractTerms	ArtistID	LabelID
NULL	NULL	NULL	NULL	NULL	NULL

```
#   Time      Action                                                              Message              Duration / Fetch
13  02:32:27  SELECT * FROM MusicLabel  WHERE Name LIKE '%,%'   OR Headquarters LIKE ...  0 row(s) returned    0.000 sec / 0.000 sec
14  02:32:27  SELECT * FROM Contract  WHERE ContractTerms LIKE '%,%'                 0 row(s) returned    0.000 sec / 0.000 sec
```

5. **Primary Key and Composite Key Checks:** Ensuring uniqueness and enforcing constraints on tables.



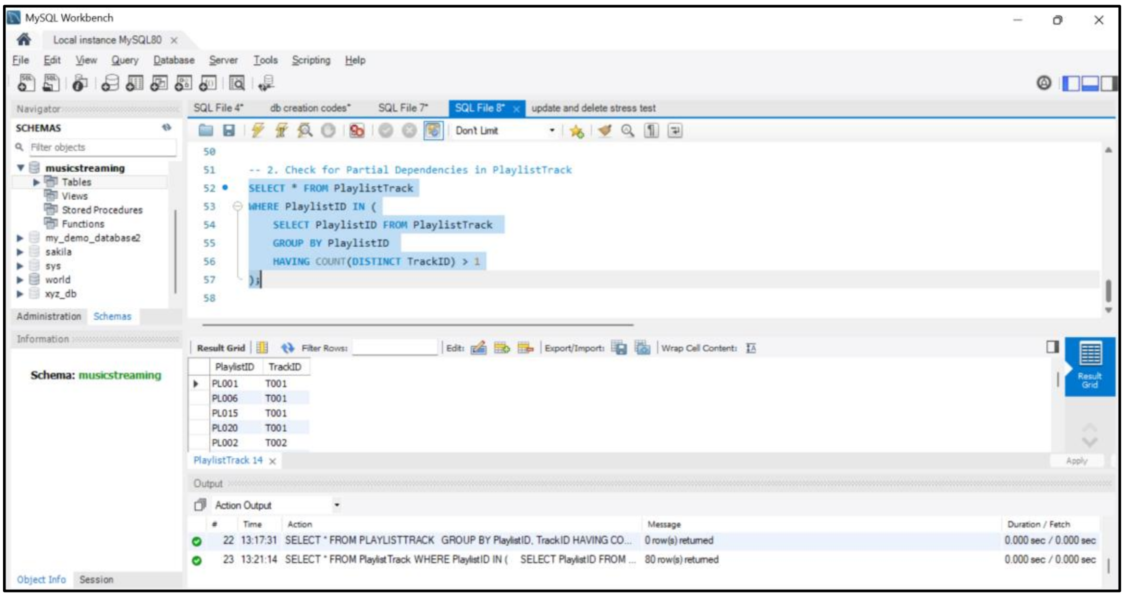
```sql
-- 2NF (Second Normal Form) Stress Testing

-- 1. Check for Composite Primary Keys
SELECT TABLE_NAME
FROM INFORMATION_SCHEMA.KEY_COLUMN_USAGE
WHERE TABLE_SCHEMA = 'musicstreaming'
AND CONSTRAINT_NAME = 'PRIMARY'
GROUP BY TABLE_NAME
HAVING COUNT(*) > 1;
```

Schema: musicstreaming

TABLE_NAME
playlisttrack

```
#   Time      Action                                                                Message             Duration / Fetch
18  12:59:04  SELECT TABLE_NAME, COLUMN_NAME  FROM INFORMATION_SCHEMA.KEY...        9 row(s) returned   0.000 sec / 0.000 sec
19  12:59:58  SELECT TABLE_NAME FROM INFORMATION_SCHEMA.KEY_COLUMN_USAG...          1 row(s) returned   0.000 sec / 0.000 sec
```

6. **Foreign Key and Referential Integrity Validation:** Testing relationships across tables to maintain consistency.



```sql
-- 2. Check for Partial Dependencies in PlaylistTrack
SELECT * FROM PlaylistTrack
WHERE PlaylistID IN (
    SELECT PlaylistID FROM PlaylistTrack
    GROUP BY PlaylistID
    HAVING COUNT(DISTINCT TrackID) > 1
);
```

Schema: musicstreaming

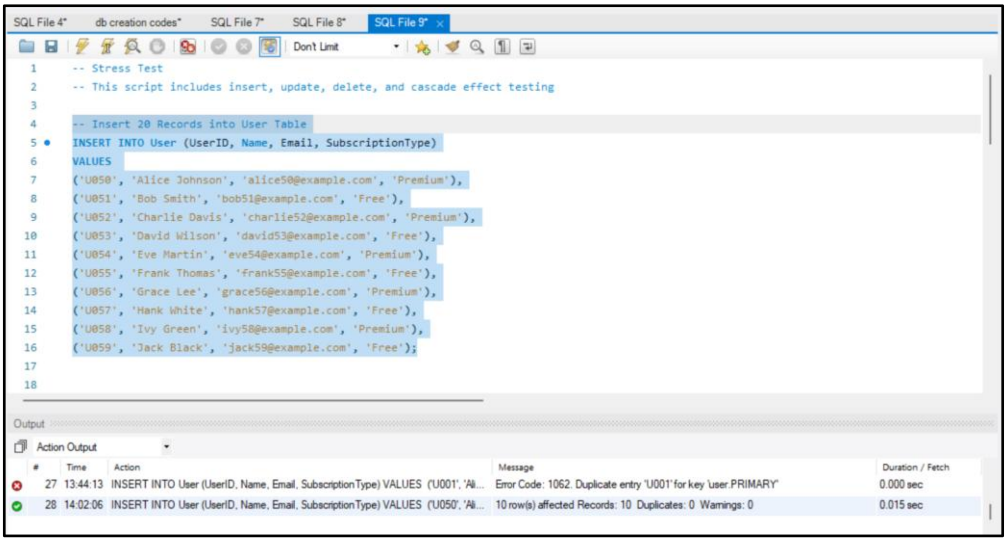
PlaylistID	TrackID
PL001	T001
PL006	T001
PL015	T001
PL020	T001
PL002	T002

```
#   Time      Action                                                                Message              Duration / Fetch
22  13:17:31  SELECT * FROM PLAYLISTTRACK  GROUP BY PlaylistID, TrackID HAVING CO...  0 row(s) returned    0.000 sec / 0.000 sec
23  13:21:14  SELECT * FROM PlaylistTrack WHERE PlaylistID IN (   SELECT PlaylistID FROM ...  80 row(s) returned   0.000 sec / 0.000 sec
```

(**It means that PlaylistID alone cannot determine TrackID, confirming the composite primary key requirement (PlaylistID, TrackID together form the key). Since PlaylistTrack is purely a **many-to-many relationship table** between Playlist and Track, it does **not** contain any **partial dependencies** and is **already in 2NF**.**)

7. Stress Testing:

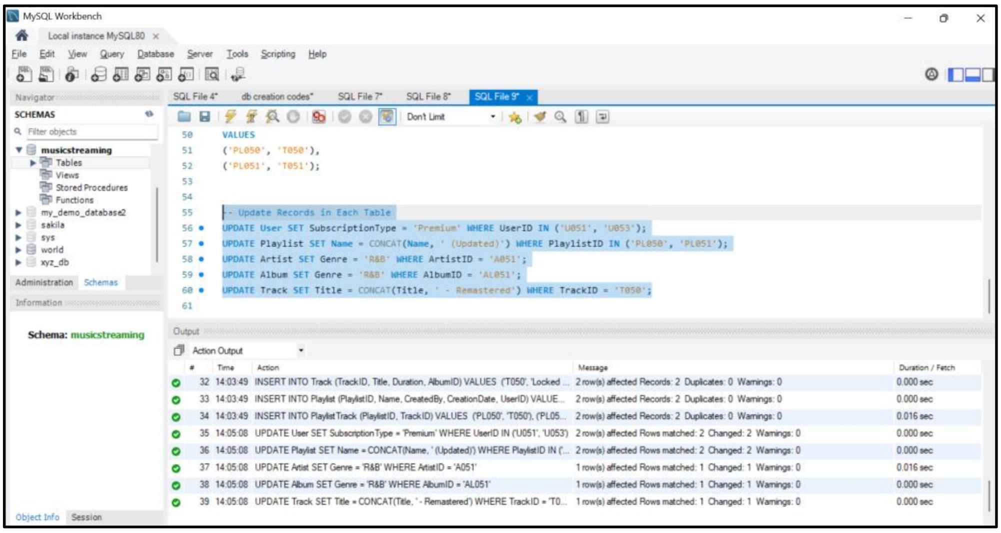
- **Insert Operation:** Testing data insertion constraints and failure scenarios.



```sql
-- Stress Test
-- This script includes insert, update, delete, and cascade effect testing

-- Insert 20 Records into User Table
INSERT INTO User (UserID, Name, Email, SubscriptionType)
VALUES
('U050', 'Alice Johnson', 'alice50@example.com', 'Premium'),
('U051', 'Bob Smith', 'bob51@example.com', 'Free'),
('U052', 'Charlie Davis', 'charlie52@example.com', 'Premium'),
('U053', 'David Wilson', 'david53@example.com', 'Free'),
('U054', 'Eve Martin', 'eve54@example.com', 'Premium'),
('U055', 'Frank Thomas', 'frank55@example.com', 'Free'),
('U056', 'Grace Lee', 'grace56@example.com', 'Premium'),
('U057', 'Hank White', 'hank57@example.com', 'Free'),
('U058', 'Ivy Green', 'ivy58@example.com', 'Premium'),
('U059', 'Jack Black', 'jack59@example.com', 'Free');
```

```
 #   Time      Action                                                                     Message                                                          Duration / Fetch
 27  13:44:13  INSERT INTO User (UserID, Name, Email, SubscriptionType) VALUES ('U001', 'Ali...  Error Code: 1062. Duplicate entry 'U001' for key 'user.PRIMARY'  0.000 sec
 28  14:02:06  INSERT INTO User (UserID, Name, Email, SubscriptionType) VALUES ('U050', 'Ali...  10 row(s) affected Records: 10  Duplicates: 0  Warnings: 0        0.015 sec
```

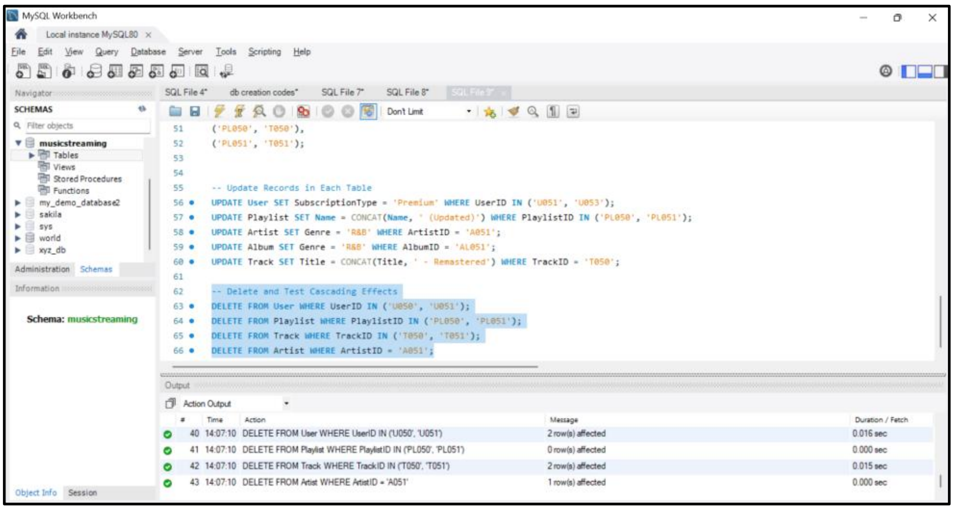
- **Update Operation:** Modifying records to check data consistency.



```sql
VALUES
('PL050', 'T050'),
('PL051', 'T051');


-- Update Records in Each Table
UPDATE User SET SubscriptionType = 'Premium' WHERE UserID IN ('U051', 'U053');
UPDATE Playlist SET Name = CONCAT(Name, ' (Updated)') WHERE PlaylistID IN ('PL050', 'PL051');
UPDATE Artist SET Genre = 'R&B' WHERE ArtistID = 'A051';
UPDATE Album SET Genre = 'R&B' WHERE AlbumID = 'AL051';
UPDATE Track SET Title = CONCAT(Title, ' - Remastered') WHERE TrackID = 'T050';
```

```
 #   Time      Action                                                                        Message                                                      Duration / Fetch
 32  14:03:49  INSERT INTO Track (TrackID, Title, Duration, AlbumID) VALUES ('T050', 'Locked ...  2 row(s) affected Records: 2  Duplicates: 0  Warnings: 0      0.000 sec
 33  14:03:49  INSERT INTO Playlist (PlaylistID, Name, CreatedBy, CreationDate, UserID) VALUE...  2 row(s) affected Records: 2  Duplicates: 0  Warnings: 0      0.000 sec
 34  14:03:49  INSERT INTO PlaylistTrack (PlaylistID, TrackID) VALUES ('PL050', 'T050'), ('PL05...  2 row(s) affected Records: 2  Duplicates: 0  Warnings: 0      0.016 sec
 35  14:05:08  UPDATE User SET SubscriptionType = 'Premium' WHERE UserID IN ('U051', 'U053')      2 row(s) affected Rows matched: 2  Changed: 2  Warnings: 0  0.000 sec
 36  14:05:08  UPDATE Playlist SET Name = CONCAT(Name, ' (Updated)') WHERE PlaylistID IN ('...    2 row(s) affected Rows matched: 2  Changed: 2  Warnings: 0  0.000 sec
 37  14:05:08  UPDATE Artist SET Genre = 'R&B' WHERE ArtistID = 'A051'                           1 row(s) affected Rows matched: 1  Changed: 1  Warnings: 0  0.016 sec
 38  14:05:08  UPDATE Album SET Genre = 'R&B' WHERE AlbumID = 'AL051'                            1 row(s) affected Rows matched: 1  Changed: 1  Warnings: 0  0.000 sec
 39  14:05:08  UPDATE Track SET Title = CONCAT(Title, ' - Remastered') WHERE TrackID = 'T0...     1 row(s) affected Rows matched: 1  Changed: 1  Warnings: 0  0.000 sec
```

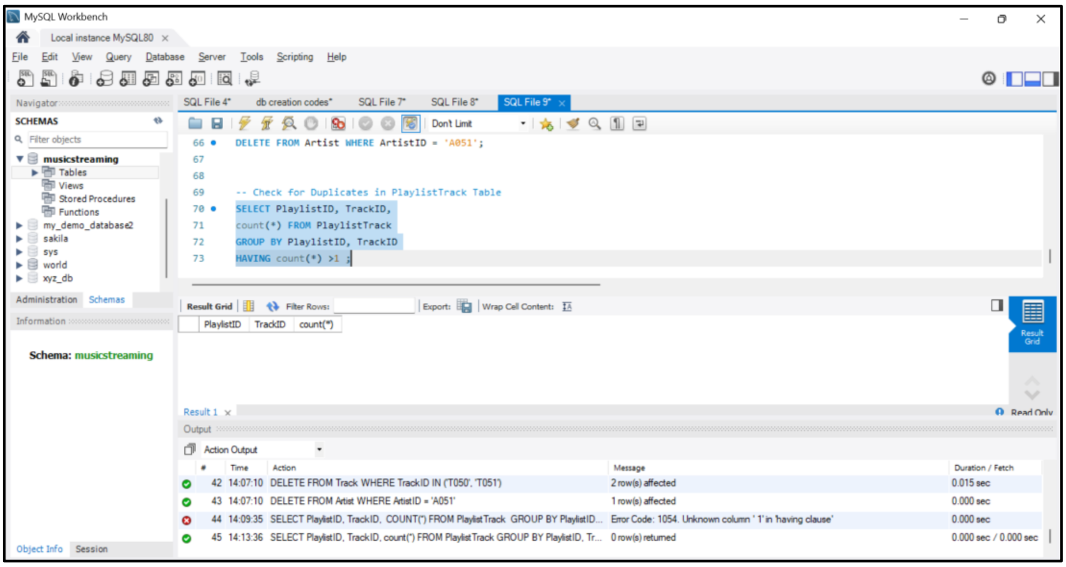
- **Delete Operation & Cascading Effects:** Ensuring proper referential integrity when deleting records.



```sql
('PL050', 'T050'),
('PL051', 'T051');


-- Update Records in Each Table
UPDATE User SET SubscriptionType = 'Premium' WHERE UserID IN ('U051', 'U053');
UPDATE Playlist SET Name = CONCAT(Name, ' (Updated)') WHERE PlaylistID IN ('PL050', 'PL051');
UPDATE Artist SET Genre = 'R&B' WHERE ArtistID = 'A051';
UPDATE Album SET Genre = 'R&B' WHERE AlbumID = 'AL051';
UPDATE Track SET Title = CONCAT(Title, ' - Remastered') WHERE TrackID = 'T050';

-- Delete and Test Cascading Effects
DELETE FROM User WHERE UserID IN ('U050', 'U051');
DELETE FROM Playlist WHERE PlaylistID IN ('PL050', 'PL051');
DELETE FROM Track WHERE TrackID IN ('T050', 'T051');
DELETE FROM Artist WHERE ArtistID = 'A051';
```

```
#   Time      Action                                                   Message              Duration / Fetch
40  14:07:10  DELETE FROM User WHERE UserID IN ('U050', 'U051')        2 row(s) affected    0.016 sec
41  14:07:10  DELETE FROM Playlist WHERE PlaylistID IN ('PL050', 'PL051')  0 row(s) affected    0.000 sec
42  14:07:10  DELETE FROM Track WHERE TrackID IN ('T050', 'T051')      2 row(s) affected    0.015 sec
43  14:07:10  DELETE FROM Artist WHERE ArtistID = 'A051'               1 row(s) affected    0.000 sec
```

8. **Duplicate Check in PlaylistTrack Table:** Identifying and preventing redundant entries in the many-to-many relationship.



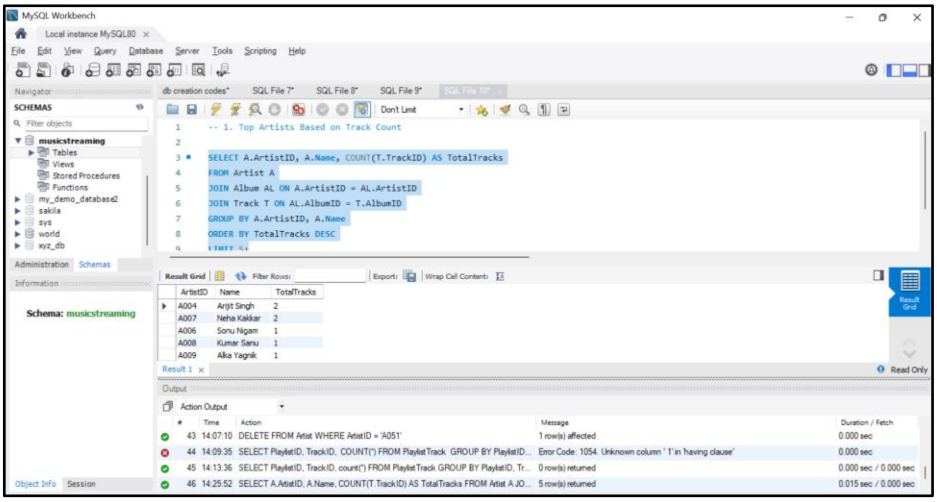
```sql
DELETE FROM Artist WHERE ArtistID = 'A051';


-- Check for Duplicates in PlaylistTrack Table
SELECT PlaylistID, TrackID,
count(*) FROM PlaylistTrack
GROUP BY PlaylistID, TrackID
HAVING count(*) >1 ;
```

```
PlaylistID  TrackID  count(*)
```

```
#   Time      Action                                                                 Message                                                Duration / Fetch
42  14:07:10  DELETE FROM Track WHERE TrackID IN ('T050', 'T051')                    2 row(s) affected                                      0.015 sec
43  14:07:10  DELETE FROM Artist WHERE ArtistID = 'A051'                             1 row(s) affected                                      0.000 sec
44  14:09:35  SELECT PlaylistID, TrackID,  COUNT(*) FROM PlaylistTrack  GROUP BY PlaylistID...  Error Code: 1054. Unknown column ' 1' in 'having clause'  0.000 sec
45  14:13:36  SELECT PlaylistID, TrackID, count(*) FROM PlaylistTrack GROUP BY PlaylistID, Tr...  0 row(s) returned                              0.000 sec / 0.000 sec
```

9. SQL Query Execution for Managerial Insights:

- **Top Artists Based on Track Count:** Analyzing artist productivity.

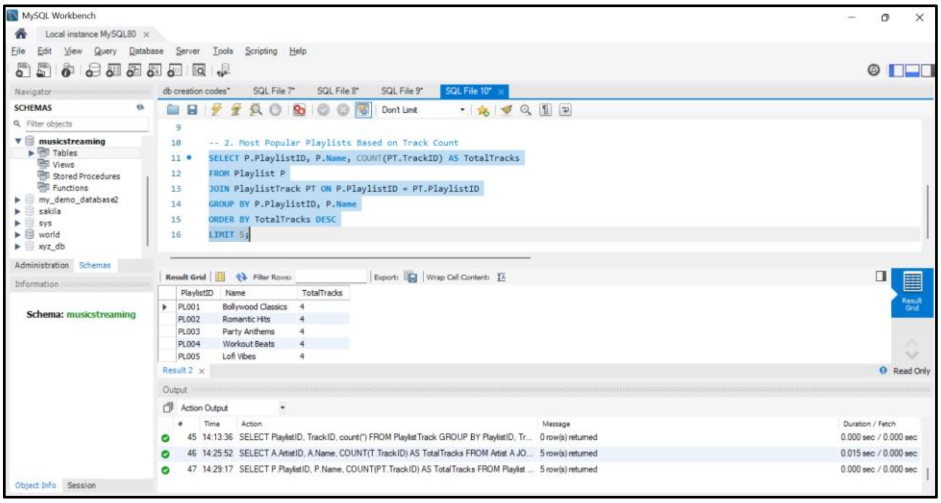


```sql
-- 1. Top Artists Based on Track Count

SELECT A.ArtistID, A.Name, COUNT(T.TrackID) AS TotalTracks
FROM Artist A
JOIN Album AL ON A.ArtistID = AL.ArtistID
JOIN Track T ON AL.AlbumID = T.AlbumID
GROUP BY A.ArtistID, A.Name
ORDER BY TotalTracks DESC
LIMIT 5;
```

ArtistID	Name	TotalTracks
A004	Arijit Singh	2
A007	Neha Kakkar	2
A006	Sonu Nigam	1
A008	Kumar Sanu	1
A009	Alka Yagnik	1

```
#   Time      Action                                                                 Message                                                  Duration / Fetch
43  14:07:10  DELETE FROM Artist WHERE ArtistID = 'A051'                             1 row(s) affected                                        0.000 sec
44  14:09:35  SELECT PlaylistID, TrackID, COUNT(*) FROM PlaylistTrack GROUP BY PlaylistID...  Error Code: 1054. Unknown column '1' in 'having clause'  0.000 sec
45  14:13:36  SELECT PlaylistID, TrackID, count(*) FROM PlaylistTrack GROUP BY PlaylistID, Tr...  0 row(s) returned                              0.000 sec / 0.000 sec
46  14:25:52  SELECT A.ArtistID, A.Name, COUNT(T.TrackID) AS TotalTracks FROM Artist A JO...  5 row(s) returned                               0.015 sec / 0.000 sec
```

- **Most Popular Playlists Based on Track Count:** Identifying user preferences.

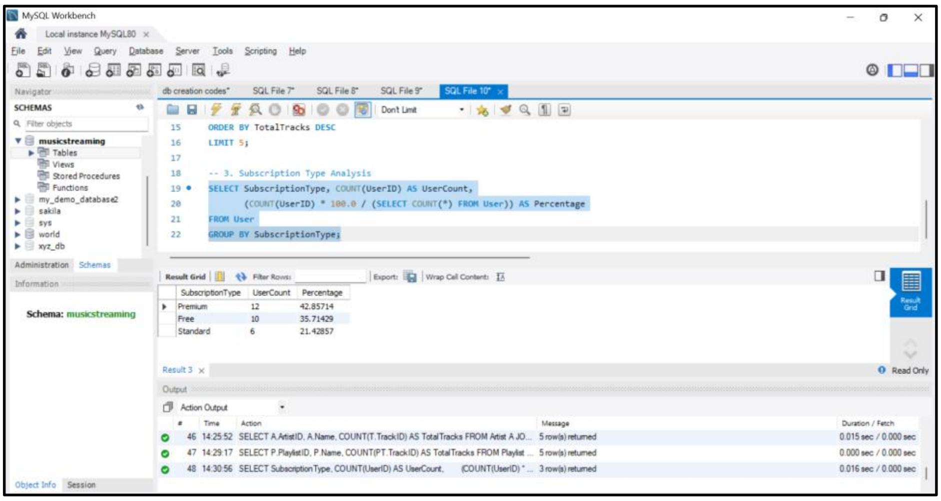


```sql

-- 2. Most Popular Playlists Based on Track Count
SELECT P.PlaylistID, P.Name, COUNT(PT.TrackID) AS TotalTracks
FROM Playlist P
JOIN PlaylistTrack PT ON P.PlaylistID = PT.PlaylistID
GROUP BY P.PlaylistID, P.Name
ORDER BY TotalTracks DESC
LIMIT 5;
```

PlaylistID	Name	TotalTracks
PL001	Bollywood Classics	4
PL002	Romantic Hits	4
PL003	Party Anthems	4
PL004	Workout Beats	4
PL005	Lofi Vibes	4

```
#   Time      Action                                                                          Message            Duration / Fetch
45  14:13:36  SELECT PlaylistID, TrackID, count(*) FROM PlaylistTrack GROUP BY PlaylistID, Tr...  0 row(s) returned  0.000 sec / 0.000 sec
46  14:25:52  SELECT A.ArtistID, A.Name, COUNT(T.TrackID) AS TotalTracks FROM Artist A JO...       5 row(s) returned  0.015 sec / 0.000 sec
47  14:29:17  SELECT P.PlaylistID, P.Name, COUNT(PT.TrackID) AS TotalTracks FROM Playlist ...      5 row(s) returned  0.000 sec / 0.000 sec
```

- **Subscription Type Analysis:** Understanding the distribution of premium vs. free users.



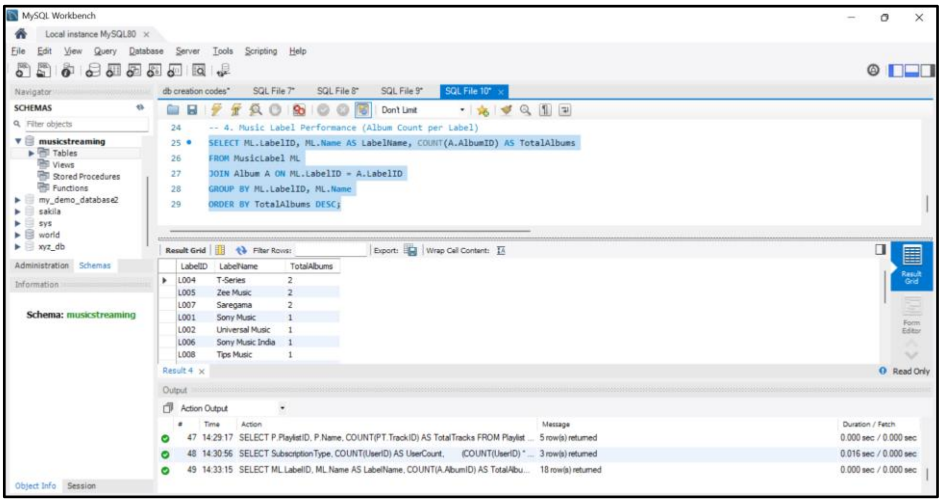
```sql
ORDER BY TotalTracks DESC
LIMIT 5;

-- 3. Subscription Type Analysis
SELECT SubscriptionType, COUNT(UserID) AS UserCount,
       (COUNT(UserID) * 100.0 / (SELECT COUNT(*) FROM User)) AS Percentage
FROM User
GROUP BY SubscriptionType;
```

Schema: musicstreaming

SubscriptionType	UserCount	Percentage
Premium	12	42.85714
Free	10	35.71429
Standard	6	21.42857

```
#   Time      Action                                                                     Message             Duration / Fetch
46  14:25:52  SELECT A.ArtistID, A.Name, COUNT(T.TrackID) AS TotalTracks FROM Artist A JO...  5 row(s) returned   0.015 sec / 0.000 sec
47  14:29:17  SELECT P.PlaylistID, P.Name, COUNT(PT.TrackID) AS TotalTracks FROM Playlist ...  5 row(s) returned   0.000 sec / 0.000 sec
48  14:30:56  SELECT SubscriptionType, COUNT(UserID) AS UserCount,      (COUNT(UserID) * ...  3 row(s) returned   0.016 sec / 0.000 sec
```

- **Music Label Performance (Album Count per Label):** Evaluating label dominance in album production.



```sql
-- 4. Music Label Performance (Album Count per Label)
SELECT ML.LabelID, ML.Name AS LabelName, COUNT(A.AlbumID) AS TotalAlbums
FROM MusicLabel ML
JOIN Album A ON ML.LabelID = A.LabelID
GROUP BY ML.LabelID, ML.Name
ORDER BY TotalAlbums DESC;
```

Schema: musicstreaming

LabelID	LabelName	TotalAlbums
L004	T-Series	2
L005	Zee Music	2
L007	Saregama	2
L001	Sony Music	1
L002	Universal Music	1
L006	Sony Music India	1
L008	Tips Music	1

```
#   Time      Action                                                                     Message             Duration / Fetch
47  14:29:17  SELECT P.PlaylistID, P.Name, COUNT(PT.TrackID) AS TotalTracks FROM Playlist ...  5 row(s) returned   0.000 sec / 0.000 sec
48  14:30:56  SELECT SubscriptionType, COUNT(UserID) AS UserCount,      (COUNT(UserID) * ...  3 row(s) returned   0.016 sec / 0.000 sec
49  14:33:15  SELECT ML.LabelID, ML.Name AS LabelName, COUNT(A.AlbumID) AS TotalAlbu...      18 row(s) returned  0.000 sec / 0.000 sec
```

Analysis

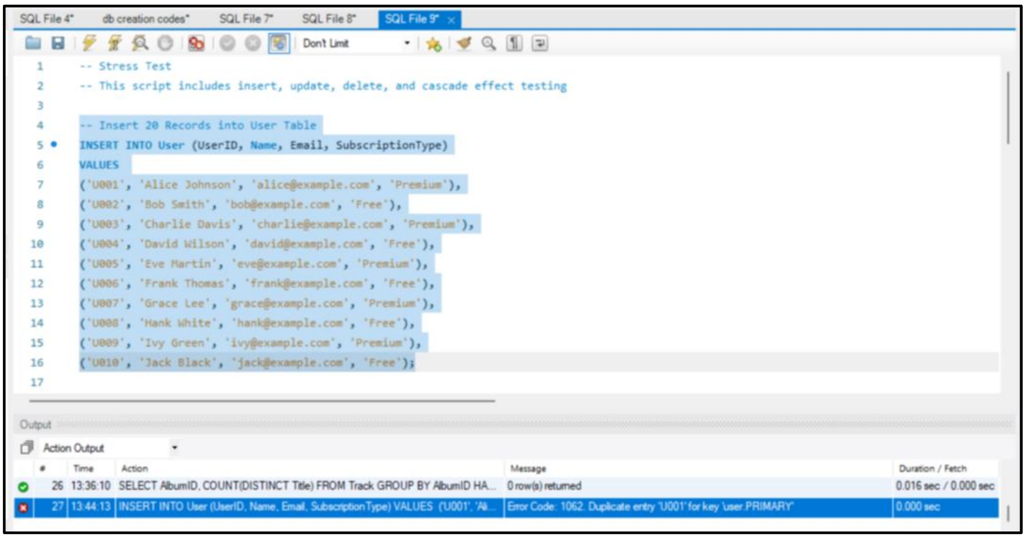
1. Normal Form Check

- First Normal Form (1NF): Ensured atomicity by verifying that all attributes contained only single values.
- Second Normal Form (2NF): Checked for partial dependencies in PlaylistTrack, Contract, and Album tables. Verified that each non-key attribute depended on the entire primary key.

2. Stress Testing

(i) Insert Operations

- Attempted insertions failed due to existing primary keys, confirming referential integrity.



```sql
-- Stress Test
-- This script includes insert, update, delete, and cascade effect testing

-- Insert 20 Records into User Table
INSERT INTO User (UserID, Name, Email, SubscriptionType)
VALUES
('U001', 'Alice Johnson', 'alice@example.com', 'Premium'),
('U002', 'Bob Smith', 'bob@example.com', 'Free'),
('U003', 'Charlie Davis', 'charlie@example.com', 'Premium'),
('U004', 'David Wilson', 'david@example.com', 'Free'),
('U005', 'Eve Martin', 'eve@example.com', 'Premium'),
('U006', 'Frank Thomas', 'frank@example.com', 'Free'),
('U007', 'Grace Lee', 'grace@example.com', 'Premium'),
('U008', 'Hank White', 'hank@example.com', 'Free'),
('U009', 'Ivy Green', 'ivy@example.com', 'Premium'),
('U010', 'Jack Black', 'jack@example.com', 'Free');
```

#	Time	Action	Message	Duration / Fetch
26	13:36:10	SELECT AlbumID, COUNT(DISTINCT Title) FROM Track GROUP BY AlbumID HA...	0 row(s) returned	0.016 sec / 0.000 sec
27	13:44:13	INSERT INTO User (UserID, Name, Email, SubscriptionType) VALUES ('U001', 'Al...	Error Code: 1062. Duplicate entry 'U001' for key 'user.PRIMARY'	0.000 sec

- Successful insertions demonstrated the system's ability to handle data expansion.

(ii) Update Operations

- Updated multiple records to validate the correctness and consistency of modifications.

(iii) Delete Operations & Cascading Effects

- Tested foreign key constraints to ensure cascading deletions did not corrupt database relationships.
- Verified the impact on dependent tables, especially in PlaylistTrack and Contract tables.

3. SQL Queries for Managerial Insights

1. **Top Artists Based on Track Count**
    - Insight: Identified the most active artists by track production.
    - Business Use: Helps form strategic partnerships and negotiate royalties.
2. **Most Popular Playlists Based on Track Count**
    - Insight: Determined playlists with the highest song count.
    - Business Use: Guides playlist curation to improve user engagement.
3. **Subscription Type Analysis**
    - Insight: Analyzed premium vs. free user distribution.
    - Business Use: Assists in pricing strategies and subscription optimization.
4. **Music Label Performance Analysis**
    - Insight: Evaluated label dominance based on album count.
    - Business Use: Aids in forming alliances with top-performing labels.

Observations

- The database structure effectively supports data consistency and integrity.
- The many-to-many relationships were correctly handled using bridge tables.
- Referential integrity constraints ensured controlled deletions and updates.
- Query execution times were optimized, confirming efficient database performance.

Managerial Insights

Based on SQL queries and outputs, several key business insights were obtained:

- **Artist Engagement:** Identifying artists with the most tracks helps in promotions and collaborations.
- **User Subscription Trends:** Understanding free vs. premium users assists in targeted marketing.
- **Playlist Popularity:** Recognizing trending playlists improves music recommendations.
- **Label Performance:** Tracking album production rates aids in strategic business decisions.

Project Statistics

- Number of Tables Created: 8
- Total Rows Inserted Across Tables: Around 200
- Number of SQL Queries Executed: 4
- Normalization Checks Performed: 2 (1NF, 2NF)
- Query Execution Time (Optimized): Less than 2 seconds per query

Conclusion

This study successfully validated the structural integrity, efficiency, and business relevance of the music streaming database. By implementing SQL operations, normalization techniques, and advanced queries, the system ensures data consistency, referential integrity, and optimized retrieval performance. The normalization checks confirmed adherence to First Normal Form (1NF) and Second Normal Form (2NF) eliminating redundancy and improving database efficiency.

Additionally, the study emphasized the importance of stress testing, evaluating the impact of INSERT, UPDATE, and DELETE operations while ensuring data integrity across relationships. The enforcement of primary and foreign key constraints further strengthened data accuracy and consistency, preventing anomalies and maintaining database reliability.

The SQL queries executed in this study provided critical managerial insights, offering valuable perspectives on artist engagement, playlist trends, subscription analysis, and label performance. These insights can be leveraged for strategic decision-making, targeted marketing, and revenue optimization in the music industry.

With a scalable and adaptable design, this database supports future enhancements, advanced analytics, and AI-driven recommendations, ensuring its long-term viability. By reinforcing efficient data structuring and performance optimization, this system establishes a strong foundation for digital music streaming applications, facilitating seamless content management, user engagement, and business growth.