DBMS PORJECT REPORT

PROJECT NAME: MUSIC STREAMING MANAGEMENT

TEAM MEMBER DETAILS:

NAME	SRN
T ANIRUDDHA	PES1UG23AM332
SUCHITH	PES1UG23AM321

OVERVIEW:

The **Music Streaming Management System** is a database project designed to manage users, artists, songs, and playlists efficiently. It allows users to browse, play, and organize music while maintaining data consistency and security. The system demonstrates the use of relational database concepts and SQL for managing and retrieving music-related information effectively.

User requirement specification:

1. Introduction

The Music Streaming Management System is developed to manage and organize digital music data efficiently. It provides a structured database for handling information related to users, songs, artists, albums, playlists, genres, and subscription plans. The system allows users to browse songs, create playlists, subscribe to different plans, and enjoy personalized music experiences. It ensures efficient data storage, quick retrieval, and maintenance of relationships among entities such as songs, artists, and users.

2. Purpose

The purpose of this system is to design a database that supports all core operations of a music streaming platform. It focuses on maintaining data consistency, enforcing relationships, and ensuring smooth management of user subscriptions, music libraries, and playlists.

3. Functional Requirements

1. User Management:
    - Store user details such as name, email, phone number, and subscription plan.
    - Each user should be able to subscribe or upgrade to different plans.
    - Maintain the date and type of subscription plan for every user.
2. Artist Management:
    - Maintain details of artists such as artist ID and name.
    - Map each artist to multiple songs and albums.
3. Song Management:
    - Store song information including song ID, title, duration, and song link.
    - Link songs to their respective artists, albums, and genres.
    - Allow retrieval of songs based on genre, artist, or album.
4. Album Management:
    - Maintain album details such as album ID, title, release date, cover art, and duration.
    - Connect albums with multiple artists and songs.
5. Genre Management:
    - Categorize songs based on genre.
    - Store genre ID and genre name for classification.
6. Playlist Management:
    - Users can create, edit, and delete playlists.
    - Each playlist stores details like playlist ID, name, status, total duration, and track count.
    - Support multiple songs within each playlist.
7. Subscription & Payment Plan Management:
    - Store payment plan details such as plan ID, type, and amount.
    - Record which user has purchased which plan and on what date.
    - Enable future tracking and renewal of plans.

4. Non-Functional Requirements

- Data Integrity: Ensure proper relationships between entities using primary and foreign keys.
- Scalability: Support a growing number of songs, artists, and users.
- Security: Protect user and payment data from unauthorized access.
- Performance: Ensure efficient query execution and data retrieval.
- Usability: The design should be simple and easy to integrate with a front-end interface.

5. Entities Identified

- USER: Stores user information and subscription details.
- SONGS: Contains song details and links to artist, album, and genre.
- ARTISTS: Maintains artist records.
- ALBUMS: Stores album information and related songs.
- GENRE: Represents different music genres.
- PLAYLISTS: Represents user-created playlists.
- PAYMENT_PLAN: Contains information about subscription plans and their cost.

6. Relationships

- One user can have many playlists.
- Each user subscribes to one payment plan.
- A song can belong to multiple genres.
- An artist can create multiple songs and albums.
- An album contains multiple songs.
- A playlist can include multiple songs and each song can belong to many playlists (many-to-many relationship).

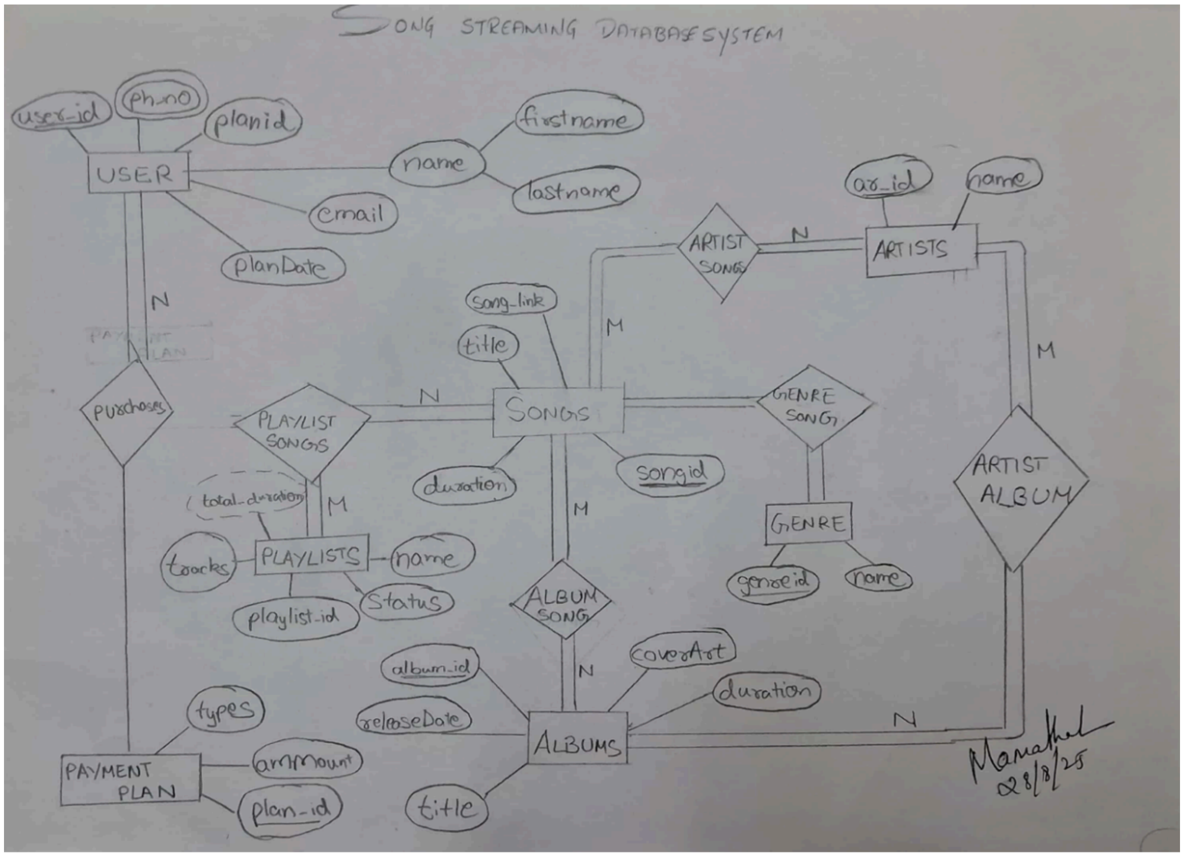
7. Expected Outcome

The system will provide a centralized and efficient database structure for managing users, songs, artists, playlists, albums, and subscriptions. It ensures reliable data handling, supports music streaming functionalities, and serves as a foundation for developing a full-fledged streaming application**.**

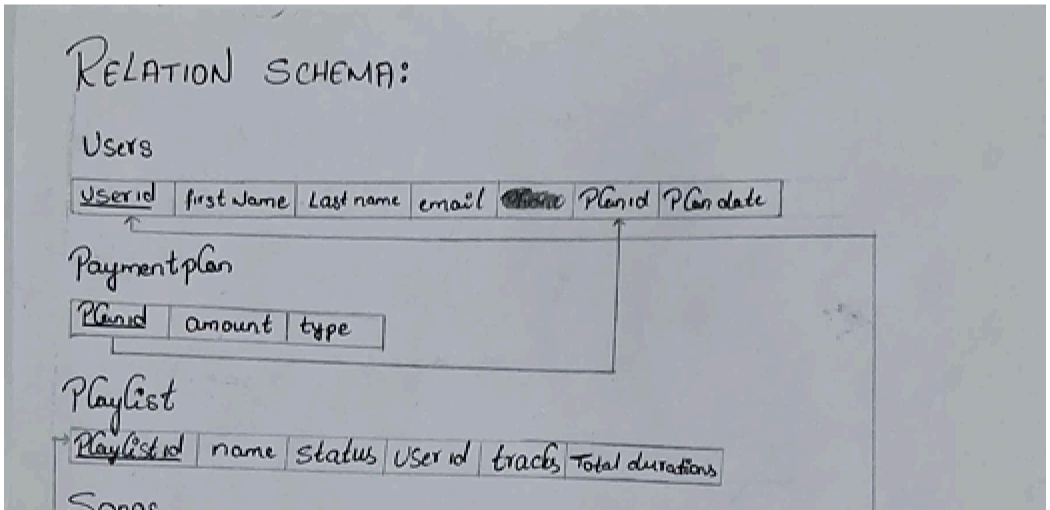
List of Software /Tools/Programming languages used:

- My sql
- Python
- Streamlit

ER DIAGRAM:



RELATION SCHEMA:



RELATION SCHEMA:

Users

Userid	first Name	Last name	email	[illegible]	Planid	Plan date

Paymentplan

Planid	amount	type

Playlist

Playlistid	name	status	User id	tracks	Total durations

DDL QUERIES:

TABLE CREATION QUERY:

```
-- ============================================

-- 🎵 MUSIC STREAMING DATABASE - DDL SCRIPT

-- ============================================


-- Drop existing tables (in dependency order)

DROP TABLE IF EXISTS `albumsong`;

DROP TABLE IF EXISTS `artistalbum`;

DROP TABLE IF EXISTS `artistsong`;

DROP TABLE IF EXISTS `genresong`;

DROP TABLE IF EXISTS `playlistsongs`;

DROP TABLE IF EXISTS `userphone`;

DROP TABLE IF EXISTS `albums`;

DROP TABLE IF EXISTS `artists`;

DROP TABLE IF EXISTS `genres`;

DROP TABLE IF EXISTS `paymentplan`;

DROP TABLE IF EXISTS `songs`;

DROP TABLE IF EXISTS `users`;

DROP TABLE IF EXISTS `playlists`;


-- ============================================

-- TABLE DEFINITIONS

-- ============================================


CREATE TABLE `albums` (
  `albumId` varchar(10) NOT NULL,
  `title` varchar(100) NOT NULL,
  `releaseDate` date DEFAULT NULL,
  `duration` int DEFAULT NULL,
  `coverArt` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`albumId`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

CREATE TABLE `artists` (
  `artistId` varchar(10) NOT NULL,
  `name` varchar(100) NOT NULL,
  PRIMARY KEY (`artistId`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

CREATE TABLE `songs` (
  `songId` varchar(10) NOT NULL,
  `title` varchar(100) NOT NULL,
  `releaseDate` date DEFAULT NULL,
  `duration` time DEFAULT NULL,
  `song_link` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`songId`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

CREATE TABLE `genres` (
  `genreId` varchar(10) NOT NULL,
  `name` varchar(50) NOT NULL,
```

```sql
  PRIMARY KEY (`genreId`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;


CREATE TABLE `paymentplan` (

  `planId` varchar(10) NOT NULL,

  `amount` decimal(10,2) NOT NULL,

  `type` varchar(50) DEFAULT NULL,

  PRIMARY KEY (`planId`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;


CREATE TABLE `users` (

  `userId` varchar(10) NOT NULL,

  `firstName` varchar(50) NOT NULL,

  `lastName` varchar(50) DEFAULT NULL,

  `email` varchar(100) NOT NULL,

  `planId` varchar(10) DEFAULT NULL,

  `paidDate` date DEFAULT NULL,

  PRIMARY KEY (`userId`),

  UNIQUE KEY `email` (`email`),

  KEY `planId` (`planId`),

  CONSTRAINT `users_ibfk_1` FOREIGN KEY (`planId`) REFERENCES `paymentplan`
(`planId`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;


CREATE TABLE `playlists` (

  `playlistId` varchar(10) NOT NULL,

  `name` varchar(100) NOT NULL,
  `status` varchar(20) DEFAULT NULL,

  `userId` varchar(10) NOT NULL,

  `tracks` int DEFAULT '0',

  `total_duration` int DEFAULT '0',

  PRIMARY KEY (`playlistId`),

  KEY `userId` (`userId`),

  CONSTRAINT `playlists_ibfk_1` FOREIGN KEY (`userId`) REFERENCES `users`
(`userId`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;


CREATE TABLE `albumsong` (

  `albumId` varchar(10) NOT NULL,

  `songId` varchar(10) NOT NULL,

  PRIMARY KEY (`albumId`,`songId`),

  KEY `songId` (`songId`),

  CONSTRAINT `albumsong_ibfk_1` FOREIGN KEY (`albumId`) REFERENCES `albums`
(`albumId`),

  CONSTRAINT `albumsong_ibfk_2` FOREIGN KEY (`songId`) REFERENCES `songs`
(`songId`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;


CREATE TABLE `artistalbum` (

  `artistId` varchar(10) NOT NULL,

  `albumId` varchar(10) NOT NULL,

  PRIMARY KEY (`artistId`,`albumId`),

  KEY `albumId` (`albumId`),

  CONSTRAINT `artistalbum_ibfk_1` FOREIGN KEY (`artistId`) REFERENCES `artists`
(`artistId`),
  CONSTRAINT `artistalbum_ibfk_2` FOREIGN KEY (`albumId`) REFERENCES `albums`
(`albumId`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

CREATE TABLE `artistsong` (
  `artistId` varchar(10) NOT NULL,
  `songId` varchar(10) NOT NULL,
  PRIMARY KEY (`artistId`,`songId`),
  KEY `songId` (`songId`),
  CONSTRAINT `artistsong_ibfk_1` FOREIGN KEY (`artistId`) REFERENCES `artists`
(`artistId`),
  CONSTRAINT `artistsong_ibfk_2` FOREIGN KEY (`songId`) REFERENCES `songs`
(`songId`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

CREATE TABLE `genresong` (
  `genreId` varchar(10) NOT NULL,
  `songId` varchar(10) NOT NULL,
  PRIMARY KEY (`genreId`,`songId`),
  KEY `songId` (`songId`),
  CONSTRAINT `genresong_ibfk_1` FOREIGN KEY (`genreId`) REFERENCES `genres`
(`genreId`),
  CONSTRAINT `genresong_ibfk_2` FOREIGN KEY (`songId`) REFERENCES `songs`
(`songId`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

CREATE TABLE `playlistsongs` (
  `playlistId` varchar(10) NOT NULL,
  `songId` varchar(10) NOT NULL,
  PRIMARY KEY (`playlistId`,`songId`),
  KEY `songId` (`songId`),
  CONSTRAINT `playlistsongs_ibfk_1` FOREIGN KEY (`playlistId`) REFERENCES
`playlists` (`playlistId`),
  CONSTRAINT `playlistsongs_ibfk_2` FOREIGN KEY (`songId`) REFERENCES `songs`
(`songId`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

CREATE TABLE `userphone` (
  `userId` varchar(10) NOT NULL,
  `phone` varchar(20) NOT NULL,
  PRIMARY KEY (`userId`,`phone`),
  CONSTRAINT `userphone_ibfk_1` FOREIGN KEY (`userId`) REFERENCES `users`
(`userId`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

-- =============================================
-- TRIGGERS
-- =============================================

DELIMITER ;;

CREATE TRIGGER `after_playlistsongs_insert`
AFTER INSERT ON `playlistsongs`
FOR EACH ROW
BEGIN
```

```sql
    UPDATE Playlists

    SET tracks = (

        SELECT COUNT(*)

        FROM PlaylistSongs

        WHERE playlistId = NEW.playlistId

    ),

    total_duration = (

        SELECT COALESCE(SUM(TIME_TO_SEC(S.duration)), 0)

        FROM PlaylistSongs PS

        JOIN Songs S ON PS.songId = S.songId

        WHERE PS.playlistId = NEW.playlistId

    )

    WHERE playlistId = NEW.playlistId;

END;;


CREATE TRIGGER `after_playlistsongs_delete`

AFTER DELETE ON `playlistsongs`

FOR EACH ROW

BEGIN

    UPDATE Playlists

    SET tracks = (

        SELECT COUNT(*)

        FROM PlaylistSongs

        WHERE playlistId = OLD.playlistId

    ),

    total_duration = (
        SELECT COALESCE(SUM(TIME_TO_SEC(S.duration)), 0)

        FROM PlaylistSongs PS

        JOIN Songs S ON PS.songId = S.songId

        WHERE PS.playlistId = OLD.playlistId

    )

    WHERE playlistId = OLD.playlistId;

END;;


DELIMITER ;


-- =============================================

-- END OF DDL SCRIPT

-- =============================================
```

DML QUERIES:

```
-- =============================================

-- 🎵 MUSIC STREAMING DATABASE - DML SCRIPT

-- =============================================


-- =============================================

-- INSERT INTO CORE TABLES

-- =============================================


-- Artists
```

```sql
INSERT INTO `artists` VALUES

('AR1','The Weeknd'),

('AR2','Taylor Swift'),

('AR3','Ed Sheeran'),

('AR4','Billie Eilish'),

('AR5','Drake');


-- Genres

INSERT INTO `genres` VALUES

('G1','Pop'),

('G2','Rock'),

('G3','Hip Hop'),

('G4','R&B'),

('G5','Electronic');


-- Payment Plans

INSERT INTO `paymentplan` VALUES

('P1',0.00,'Free'),

('P2',9.99,'Premium'),

('P3',14.99,'Family'),

('P4',4.99,'Student');


-- Albums

INSERT INTO `albums` VALUES

('AL1','After Hours','2020-03-20',3780,'https://coverart.com/1'),

('AL2','Midnights','2022-10-21',2640,'https://coverart.com/2'),
('AL3','Divide','2017-03-03',4620,'https://coverart.com/3'),

('AL4','Views','2016-04-29',4820,'https://coverart.com/4');


-- Songs

INSERT INTO `songs` VALUES

('S1','Blinding Lights','2019-11-29','00:03:20','https://spotify.com/track/1'),

('S2','Anti-Hero','2022-10-21','00:03:20','https://spotify.com/track/2'),

('S3','Shape of You','2017-01-06','00:03:53','https://spotify.com/track/3'),

('S4','Bad Guy','2019-03-29','00:03:14','https://spotify.com/track/4'),

('S5','Hotline Bling','2015-07-31','00:04:27','https://spotify.com/track/5'),

('S6','meeee','2025-10-23','00:05:12','https://google.com');


-- Users

INSERT INTO `users` VALUES

('U1','John','Smith','john@email.com','P2','2024-01-15'),

('U2','Emma','Johnson','emma@email.com','P1',NULL),

('U3','Mike','Brown','mike@email.com','P2','2024-02-01'),

('U4','Sarah','Davis','sarah@email.com','P3','2024-01-20'),

('U5','David','Wilson','david@email.com','P4','2024-03-10');


-- =============================================

-- INSERT INTO RELATIONSHIP TABLES

-- =============================================


-- Album ↔ Song

INSERT INTO `albumsong` VALUES
```

```sql
('AL1','S1'),

('AL2','S2'),

('AL3','S3'),

('AL1','S4'),

('AL4','S5');


-- Artist ↔ Album

INSERT INTO `artistalbum` VALUES

('AR1','AL1'),

('AR4','AL1'),

('AR2','AL2'),

('AR3','AL3'),

('AR5','AL4');


-- Artist ↔ Song

INSERT INTO `artistsong` VALUES

('AR1','S1'),

('AR2','S2'),

('AR3','S3'),

('AR4','S4'),

('AR5','S5');


-- Genre ↔ Song

INSERT INTO `genresong` VALUES

('G1','S1'),

('G4','S1'),
('G1','S2'),

('G1','S3'),

('G3','S5');


-- =============================================

-- INSERT INTO USER-RELATED TABLES

-- =============================================


-- Playlists

INSERT INTO `playlists` VALUES

('PL1','Workout Mix','Public','U1',2,433),

('PL2','Chill Vibes','Private','U2',3,706),

('PL3','Road Trip','Public','U3',4,927),

('PL4','Study Focus','Private','U4',3,720),

('PL5','Party Hits','Public','U5',2,387),

('PL6','KILL YOURSELF','PRIVATE','U1',1,200);


-- Playlist ↔ Songs

INSERT INTO `playlistsongs` VALUES

('PL1','S1'),

('PL3','S1'),

('PL6','S1'),

('PL2','S2'),

('PL1','S3'),

('PL2','S4'),

('PL2','S6');
```

```sql
-- User Phone Numbers

INSERT INTO `userphone` VALUES

('U1','+1-555-0101'),

('U2','+1-555-0102'),

('U3','+1-555-0103'),

('U4','+1-555-0104'),

('U5','+1-555-0105');


-- ==============================================

-- END OF DML SCRIPT

-- ==============================================
```

1.INSERT QUERIES:

```sql
INSERT INTO `albums` VALUES

('AL1','After Hours','2020-03-20',3780,'https://coverart.com/1'),

('AL2','Midnights','2022-10-21',2640,'https://coverart.com/2'),

('AL3','Divide','2017-03-03',4620,'https://coverart.com/3'),

('AL4','Views','2016-04-29',4820,'https://coverart.com/4');


INSERT INTO `albumsong` VALUES

('AL1','S1'),

('AL2','S2'),
('AL3','S3'),

('AL1','S4'),

('AL4','S5');


INSERT INTO `artistalbum` VALUES

('AR1','AL1'),

('AR4','AL1'),

('AR2','AL2'),

('AR3','AL3'),

('AR5','AL4');


INSERT INTO `artists` VALUES

('AR1','The Weeknd'),

('AR2','Taylor Swift'),

('AR3','Ed Sheeran'),

('AR4','Billie Eilish'),

('AR5','Drake');


INSERT INTO `artistsong` VALUES

('AR1','S1'),

('AR2','S2'),

('AR3','S3'),

('AR4','S4'),

('AR5','S5');


INSERT INTO `genres` VALUES
```

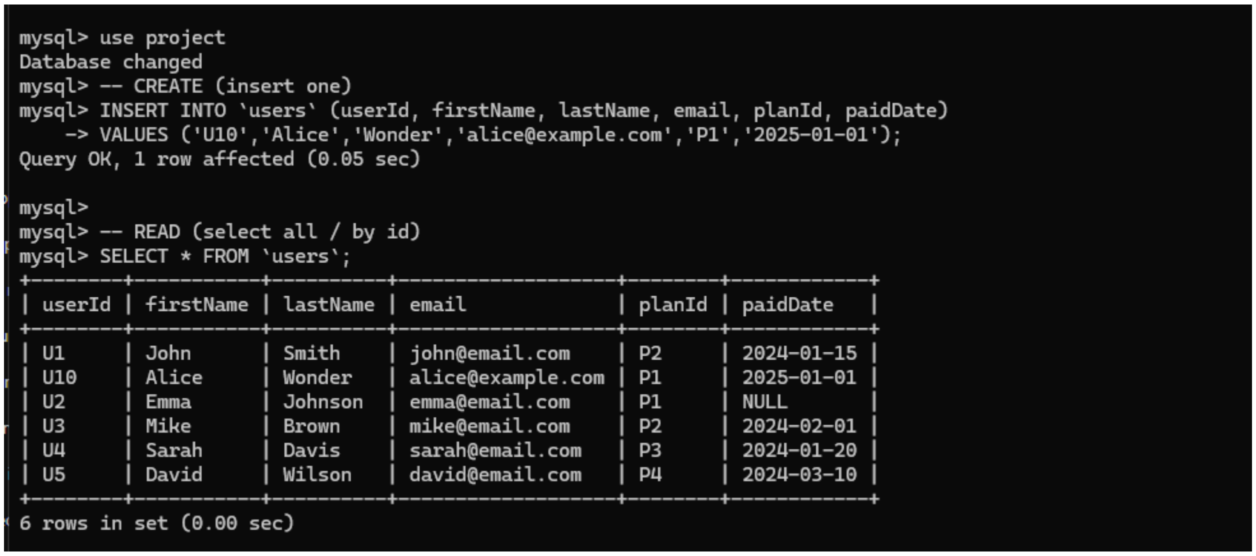
```sql
('G1','Pop'),

('G2','Rock'),

('G3','Hip Hop'),

('G4','R&B'),

('G5','Electronic');


INSERT INTO `genresong` VALUES

('G1','S1'),

('G4','S1'),

('G1','S2'),

('G1','S3'),

('G3','S5');


INSERT INTO `paymentplan` VALUES

('P1',0.00,'Free'),

('P2',9.99,'Premium'),

('P3',14.99,'Family'),

('P4',4.99,'Student');


INSERT INTO `playlists` VALUES

('PL1','Workout Mix','Public','U1',2,433),

('PL2','Chill Vibes','Private','U2',3,706),

('PL3','Road Trip','Public','U3',4,927),

('PL4','Study Focus','Private','U4',3,720),

('PL5','Party Hits','Public','U5',2,387),

('PL6','KILL YOURSELF','PRIVATE','U1',1,200);
```

```sql
INSERT INTO `playlistsongs` VALUES

('PL1','S1'),

('PL3','S1'),

('PL6','S1'),

('PL2','S2'),

('PL1','S3'),

('PL2','S4'),

('PL2','S6');


INSERT INTO `songs` VALUES

('S1','Blinding Lights','2019-11-29','00:03:20','https://spotify.com/track/1'),

('S2','Anti-Hero','2022-10-21','00:03:20','https://spotify.com/track/2'),

('S3','Shape of You','2017-01-06','00:03:53','https://spotify.com/track/3'),

('S4','Bad Guy','2019-03-29','00:03:14','https://spotify.com/track/4'),

('S5','Hotline Bling','2015-07-31','00:04:27','https://spotify.com/track/5'),

('S6','meeee','2025-10-23','00:05:12','https://google.com');


INSERT INTO `userphone` VALUES

('U1','+1-555-0101'),

('U2','+1-555-0102'),

('U3','+1-555-0103'),

('U4','+1-555-0104'),

('U5','+1-555-0105');


INSERT INTO `users` VALUES
```

```
('U1','John','Smith','john@email.com','P2','2024-01-15'),

('U2','Emma','Johnson','emma@email.com','P1',NULL),

('U3','Mike','Brown','mike@email.com','P2','2024-02-01'),

('U4','Sarah','Davis','sarah@email.com','P3','2024-01-20'),

('U5','David','Wilson','david@email.com','P4','2024-03-10');
```

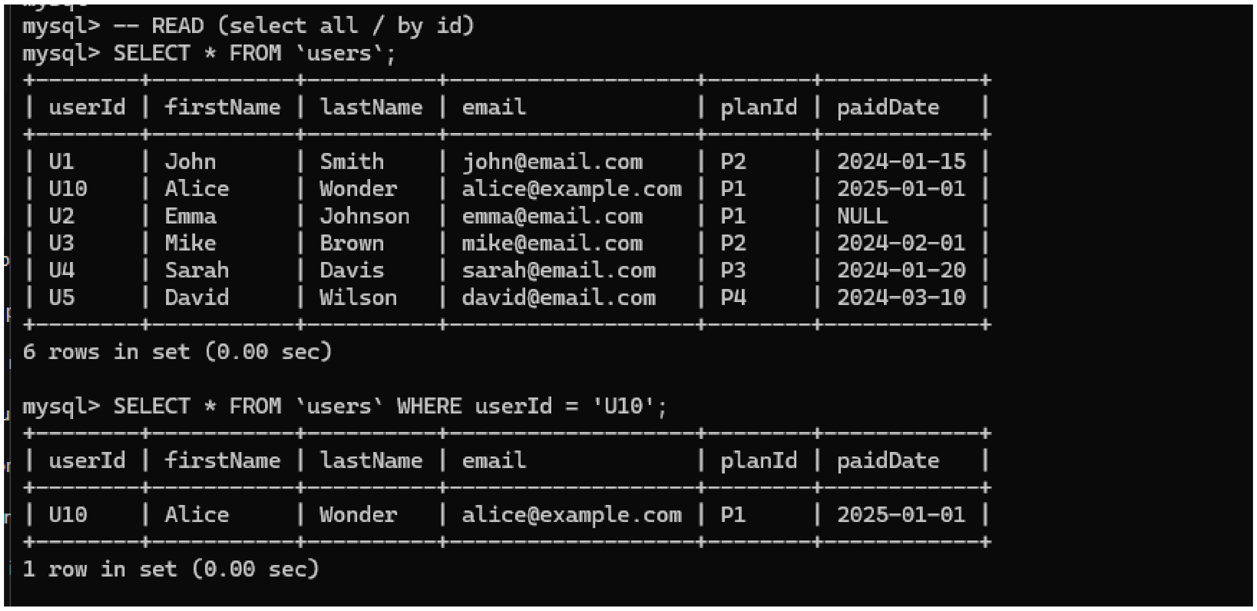
2.CRUD OPERATIONS:

- **Operations on user table**
- **create:**



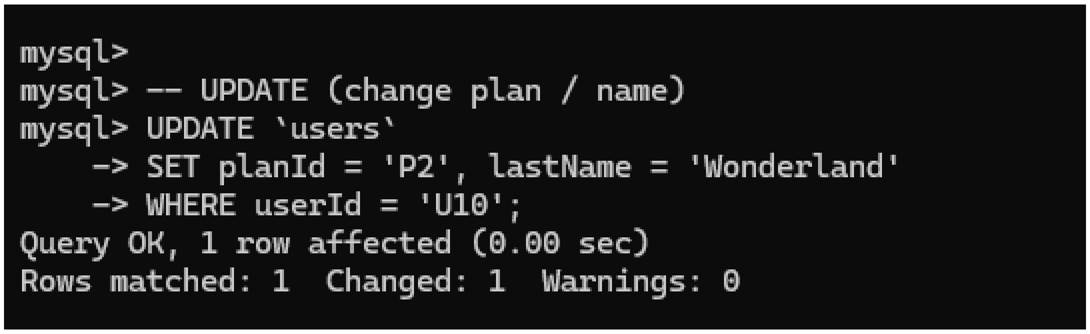
```
mysql> use project
Database changed
mysql> -- CREATE (insert one)
mysql> INSERT INTO `users` (userId, firstName, lastName, email, planId, paidDate)
    -> VALUES ('U10','Alice','Wonder','alice@example.com','P1','2025-01-01');
Query OK, 1 row affected (0.05 sec)

mysql>
mysql> -- READ (select all / by id)
mysql> SELECT * FROM `users`;
+--------+-----------+----------+-------------------+--------+------------+
| userId | firstName | lastName | email             | planId | paidDate   |
+--------+-----------+----------+-------------------+--------+------------+
| U1     | John      | Smith    | john@email.com    | P2     | 2024-01-15 |
| U10    | Alice     | Wonder   | alice@example.com | P1     | 2025-01-01 |
| U2     | Emma      | Johnson  | emma@email.com    | P1     | NULL       |
| U3     | Mike      | Brown    | mike@email.com    | P2     | 2024-02-01 |
| U4     | Sarah     | Davis    | sarah@email.com   | P3     | 2024-01-20 |
| U5     | David     | Wilson   | david@email.com   | P4     | 2024-03-10 |
+--------+-----------+----------+-------------------+--------+------------+
6 rows in set (0.00 sec)
```

Read:



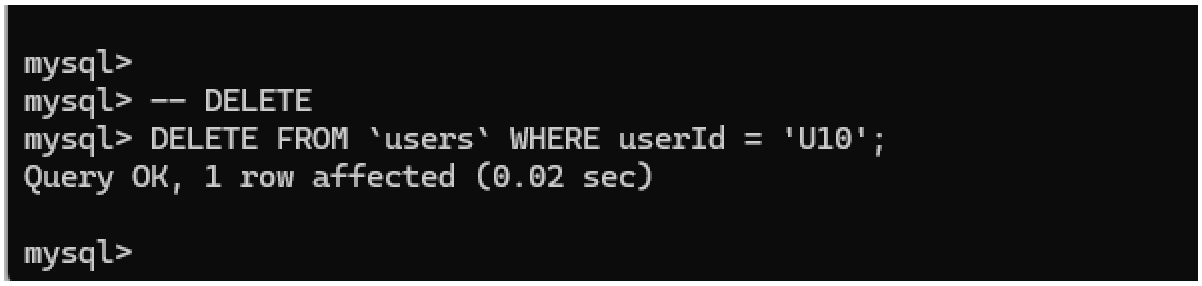
```
mysql> -- READ (select all / by id)
mysql> SELECT * FROM `users`;
+--------+-----------+----------+-------------------+--------+------------+
| userId | firstName | lastName | email             | planId | paidDate   |
+--------+-----------+----------+-------------------+--------+------------+
| U1     | John      | Smith    | john@email.com    | P2     | 2024-01-15 |
| U10    | Alice     | Wonder   | alice@example.com | P1     | 2025-01-01 |
| U2     | Emma      | Johnson  | emma@email.com    | P1     | NULL       |
| U3     | Mike      | Brown    | mike@email.com    | P2     | 2024-02-01 |
| U4     | Sarah     | Davis    | sarah@email.com   | P3     | 2024-01-20 |
| U5     | David     | Wilson   | david@email.com   | P4     | 2024-03-10 |
+--------+-----------+----------+-------------------+--------+------------+
6 rows in set (0.00 sec)

mysql> SELECT * FROM `users` WHERE userId = 'U10';
+--------+-----------+----------+-------------------+--------+------------+
| userId | firstName | lastName | email             | planId | paidDate   |
+--------+-----------+----------+-------------------+--------+------------+
| U10    | Alice     | Wonder   | alice@example.com | P1     | 2025-01-01 |
+--------+-----------+----------+-------------------+--------+------------+
1 row in set (0.00 sec)
```

Update:



```
mysql>
mysql> -- UPDATE (change plan / name)
mysql> UPDATE `users`
    -> SET planId = 'P2', lastName = 'Wonderland'
    -> WHERE userId = 'U10';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

Delete:



```
mysql>
mysql> -- DELETE
mysql> DELETE FROM `users` WHERE userId = 'U10';
Query OK, 1 row affected (0.02 sec)

mysql>
```

Procedures used in the sql

```
DROP PROCEDURE IF EXISTS update_playlist_stats;

DELIMITER $$

CREATE PROCEDURE update_playlist_stats(IN p_playlistId VARCHAR(10))

BEGIN

    UPDATE playlists

    SET tracks = (

        SELECT COUNT(*)

        FROM playlistsongs

        WHERE playlistId = p_playlistId

    ),

    total_duration = (

        SELECT COALESCE(SUM(TIME_TO_SEC(s.duration)), 0)

        FROM playlistsongs ps

        JOIN songs s ON ps.songId = s.songId

        WHERE ps.playlistId = p_playlistId

    )

    WHERE playlistId = p_playlistId;

END$$

DELIMITER ;
```

Triggers:

```sql
DELIMITER $$


DROP TRIGGER IF EXISTS after_playlistsongs_insert$$

CREATE TRIGGER after_playlistsongs_insert

AFTER INSERT ON playlistsongs

FOR EACH ROW

BEGIN

    UPDATE playlists

    SET tracks = (

        SELECT COUNT(*)

        FROM playlistsongs

        WHERE playlistId = NEW.playlistId

    ),

    total_duration = (

        SELECT COALESCE(SUM(TIME_TO_SEC(s.duration)), 0)

        FROM playlistsongs ps

        JOIN songs s ON ps.songId = s.songId

        WHERE ps.playlistId = NEW.playlistId

    )

    WHERE playlistId = NEW.playlistId;

END$$


DROP TRIGGER IF EXISTS after_playlistsongs_delete$$

CREATE TRIGGER after_playlistsongs_delete

AFTER DELETE ON playlistsongs

FOR EACH ROW
```

```
BEGIN

    UPDATE playlists

    SET tracks = (

        SELECT COUNT(*)

        FROM playlistsongs

        WHERE playlistId = OLD.playlistId

    ),

    total_duration = (

        SELECT COALESCE(SUM(TIME_TO_SEC(s.duration)), 0)

        FROM playlistsongs ps

        JOIN songs s ON ps.songId = s.songId

        WHERE ps.playlistId = OLD.playlistId

    )

    WHERE playlistId = OLD.playlistId;

END$$


DELIMITER ;
```

Functions/procedures:

```
-- 1) FUNCTION: playlist_total_duration_seconds

DROP FUNCTION IF EXISTS playlist_total_duration_seconds;

DELIMITER $$

CREATE FUNCTION playlist_total_duration_seconds(p_playlistId VARCHAR(10))
```

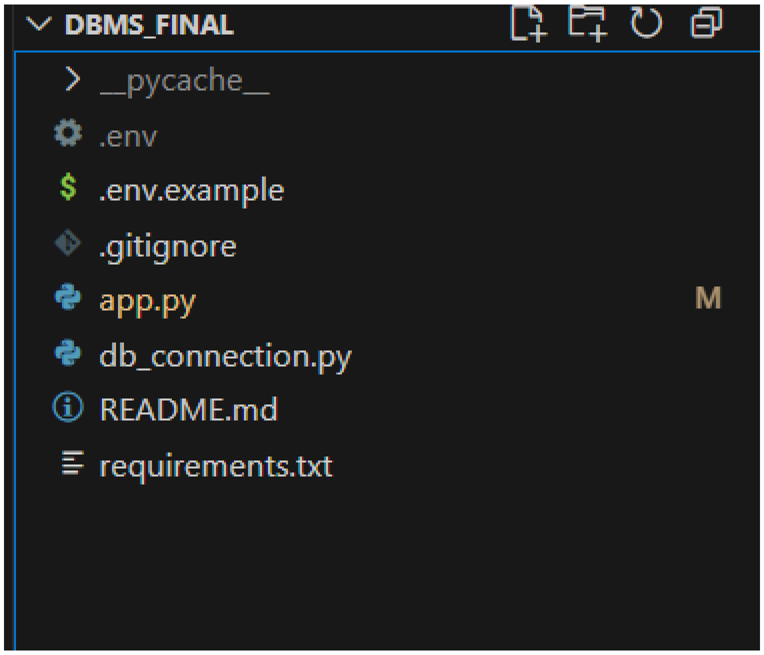
```sql
RETURNS INT

DETERMINISTIC

BEGIN

    DECLARE v_total INT DEFAULT NULL;

    -- Prefer stored total_duration if present

    SELECT total_duration INTO v_total

    FROM playlists

    WHERE playlistId = p_playlistId

    LIMIT 1;


    IF v_total IS NOT NULL AND v_total <> 0 THEN

        RETURN v_total;

    END IF;


    -- Fallback: compute from songs

    SELECT COALESCE(SUM(TIME_TO_SEC(s.duration)), 0)

    INTO v_total

    FROM playlistsongs ps

    JOIN songs s ON ps.songId = s.songId

    WHERE ps.playlistId = p_playlistId;


    RETURN IFNULL(v_total, 0);

END$$

DELIMITER ;


-- Usage (example):
```

```sql
-- SELECT playlist_total_duration_seconds('PL1');



-- 2) PROCEDURE: add_song_to_playlist

DROP PROCEDURE IF EXISTS add_song_to_playlist;

DELIMITER $$

CREATE PROCEDURE add_song_to_playlist(

    IN p_playlistId VARCHAR(10),

    IN p_songId VARCHAR(10),

    OUT p_added TINYINT               -- 1 if inserted, 0 if already exists or
error

)

BEGIN

    DECLARE EXIT HANDLER FOR SQLEXCEPTION

    BEGIN

        ROLLBACK;

        SET p_added = 0;

    END;


    START TRANSACTION;


    -- ensure playlist exists

    IF (SELECT COUNT(*) FROM playlists WHERE playlistId = p_playlistId) = 0 THEN

        SET p_added = 0;

        ROLLBACK;

        LEAVE proc_end;

    END IF;
    -- ensure song exists
    IF (SELECT COUNT(*) FROM songs WHERE songId = p_songId) = 0 THEN
        SET p_added = 0;
        ROLLBACK;
        LEAVE proc_end;
    END IF;

    -- do not add duplicate
    IF (SELECT COUNT(*) FROM playlistsongs WHERE playlistId = p_playlistId AND
songId = p_songId) > 0 THEN
        SET p_added = 0;
        COMMIT;
        LEAVE proc_end;
    END IF;

    -- insert
    INSERT INTO playlistsongs (playlistId, songId) VALUES (p_playlistId,
p_songId);

    -- update stats: prefer calling the existing procedure if present
    IF (SELECT COUNT(*) FROM information_schema.ROUTINES
        WHERE ROUTINE_SCHEMA = DATABASE() AND ROUTINE_NAME =
'update_playlist_stats') > 0 THEN
        CALL update_playlist_stats(p_playlistId);
    ELSE
        -- inline update if update_playlist_stats not available
        UPDATE playlists

        SET tracks = (

            SELECT COUNT(*) FROM playlistsongs WHERE playlistId = p_playlistId

        ),

        total_duration = (

            SELECT COALESCE(SUM(TIME_TO_SEC(s.duration)), 0)

            FROM playlistsongs ps JOIN songs s ON ps.songId = s.songId

            WHERE ps.playlistId = p_playlistId

        )

        WHERE playlistId = p_playlistId;

    END IF;


    SET p_added = 1;

    COMMIT;


    proc_end: BEGIN

        -- noop label block to allow LEAVE

    END;

END$$

DELIMITER ;


-- Usage (example):

-- CALL add_song_to_playlist('PL1','S3', @was_added);

-- SELECT @was_added;
```

```sql
-- 3) PROCEDURE: change_user_plan

DROP PROCEDURE IF EXISTS change_user_plan;

DELIMITER $$

CREATE PROCEDURE change_user_plan(

    IN p_userId VARCHAR(10),

    IN p_newPlanId VARCHAR(10),

    IN p_paidDate DATE,

    OUT p_ok TINYINT           -- 1 success, 0 failure

)

BEGIN

    DECLARE v_exists INT DEFAULT 0;

    DECLARE v_plan_exists INT DEFAULT 0;


    -- check user

    SELECT COUNT(*) INTO v_exists FROM users WHERE userId = p_userId;

    IF v_exists = 0 THEN

        SET p_ok = 0;

        LEAVE cp_end;

    END IF;


    -- if newPlanId not NULL, check that plan exists

    IF p_newPlanId IS NOT NULL THEN

        SELECT COUNT(*) INTO v_plan_exists FROM paymentplan WHERE planId =
p_newPlanId;

        IF v_plan_exists = 0 THEN

            SET p_ok = 0;

            LEAVE cp_end;
        END IF;

    END IF;


    -- perform update

    UPDATE users

    SET planId = p_newPlanId,

        paidDate = p_paidDate

    WHERE userId = p_userId;


    SET p_ok = 1;


    cp_end: BEGIN

        -- end label

    END;

END$$

DELIMITER ;


-- Usage (example):

-- CALL change_user_plan('U1','P3','2025-10-01', @ok);

-- SELECT @ok;
```

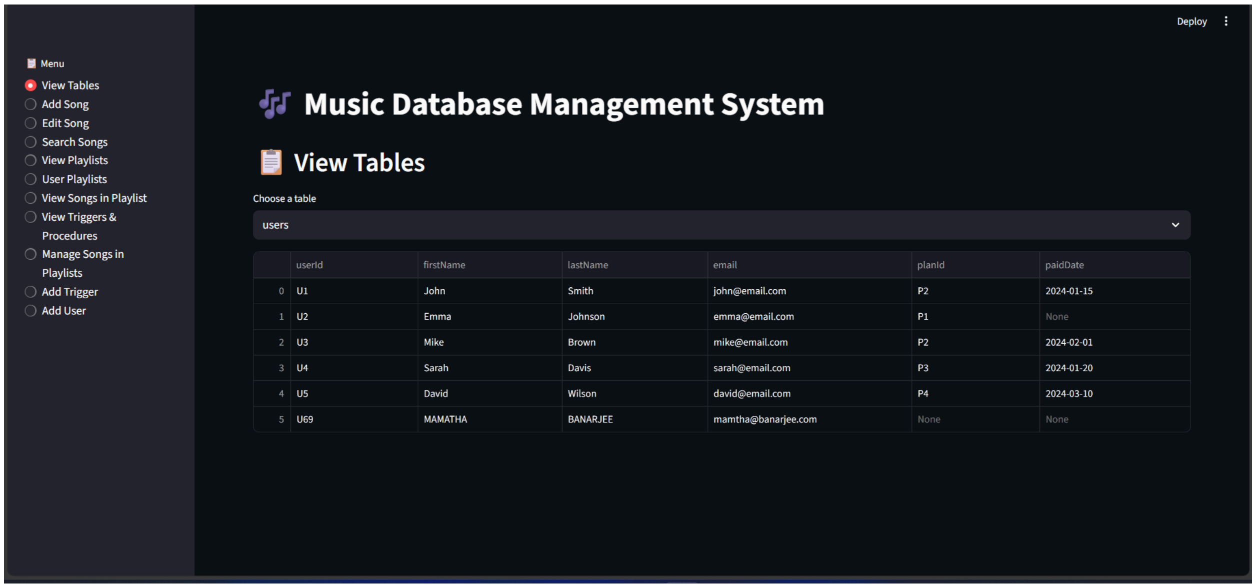
GUI:

FILE STRUCTURE:



```
DBMS_FINAL
  > __pycache__
  .env
  .env.example
  .gitignore
  app.py                M
  db_connection.py
  README.md
  requirements.txt
```

APP.py



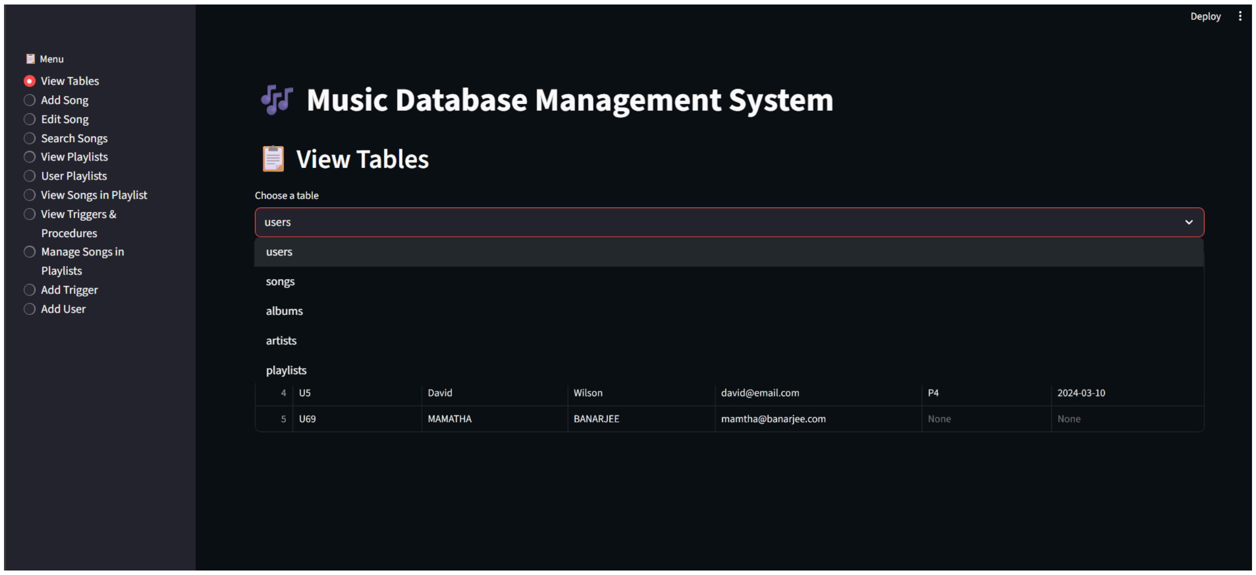
Music Database Management System

View Tables

Choose a table

users

	userId	firstName	lastName	email	planId	paidDate
0	U1	John	Smith	john@email.com	P2	2024-01-15
1	U2	Emma	Johnson	emma@email.com	P1	None
2	U3	Mike	Brown	mike@email.com	P2	2024-02-01
3	U4	Sarah	Davis	sarah@email.com	P3	2024-01-20
4	U5	David	Wilson	david@email.com	P4	2024-03-10
5	U69	MAMATHA	BANARJEE	mamtha@banarjee.com	None	None

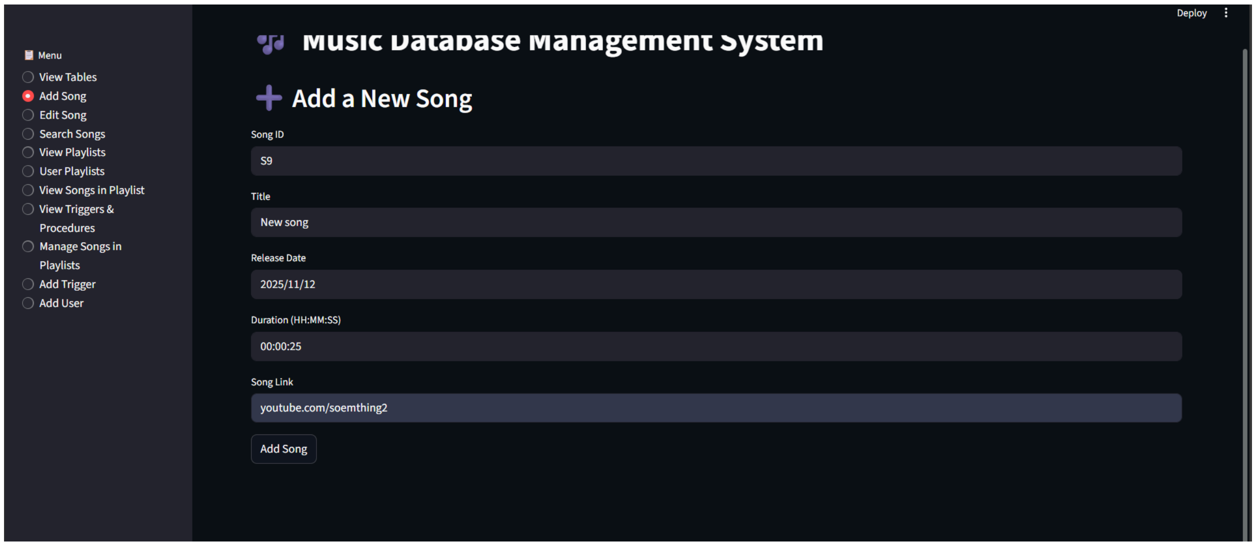


Music Database Management System

View Tables

Choose a table: users

4	U5	David	Wilson	david@email.com	P4	2024-03-10
5	U69	MAMATHA	BANARJEE	mamtha@banarjee.com	None	None



Music Database Management System

Add a New Song

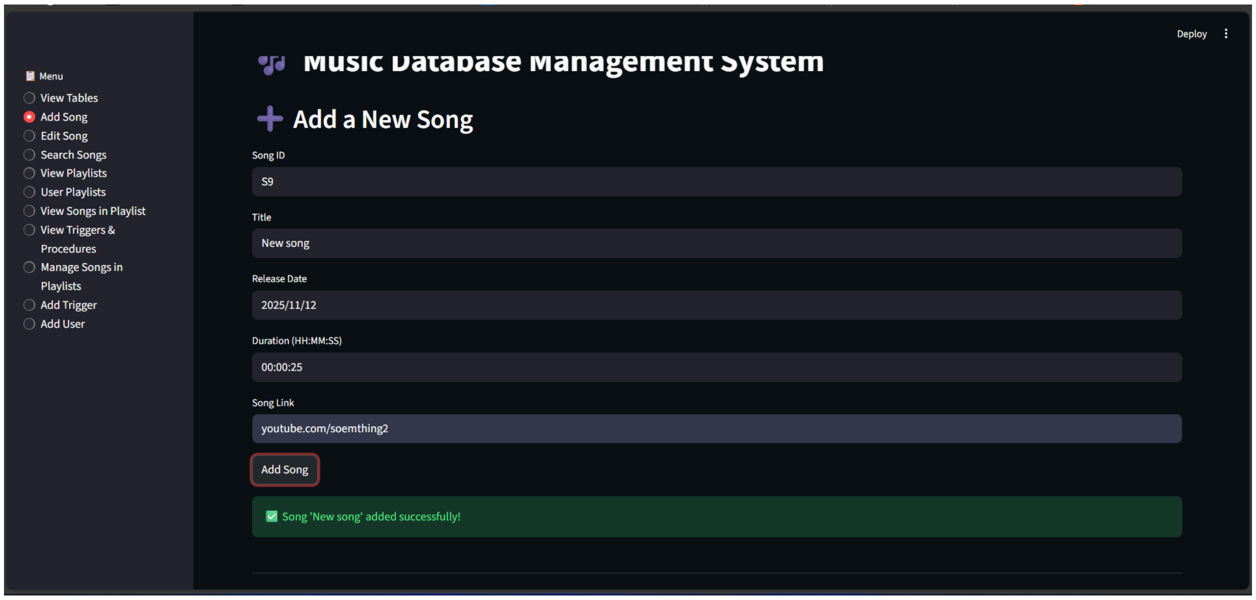
Song ID: S9

Title: New song

Release Date: 2025/11/12

Duration (HH:MM:SS): 00:00:25

Song Link: youtube.com/soemthing2



Music Database Management System

Add a New Song

Song ID: S9

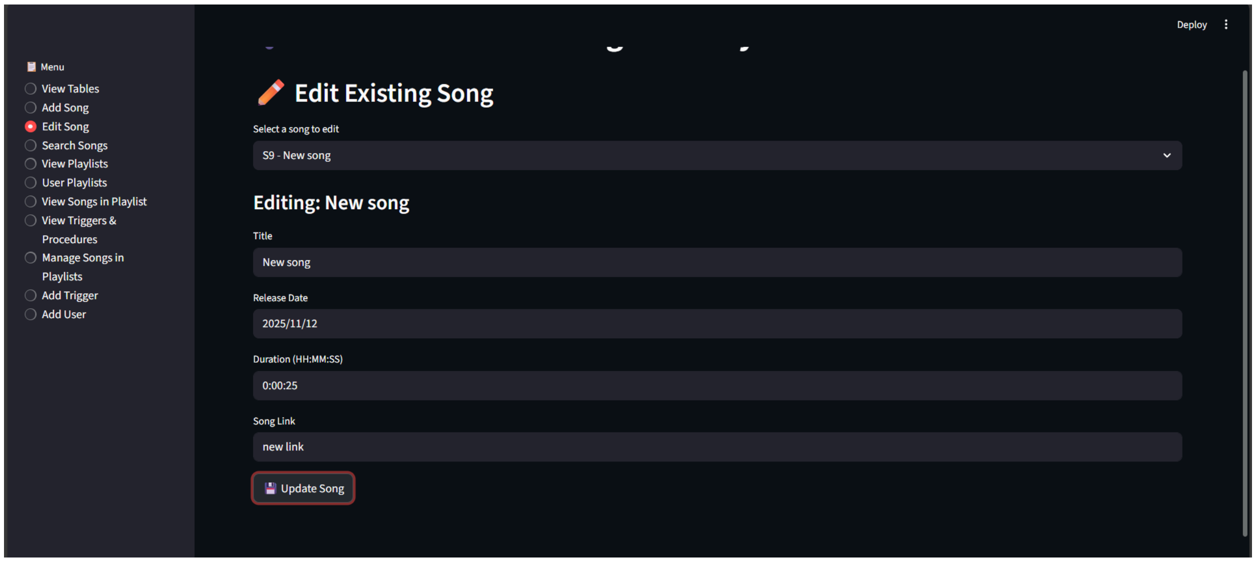
Title: New song

Release Date: 2025/11/12

Duration (HH:MM:SS): 00:00:25

Song Link: youtube.com/soemthing2

Song 'New song' added successfully!



Edit Existing Song

Select a song to edit: S9 - New song

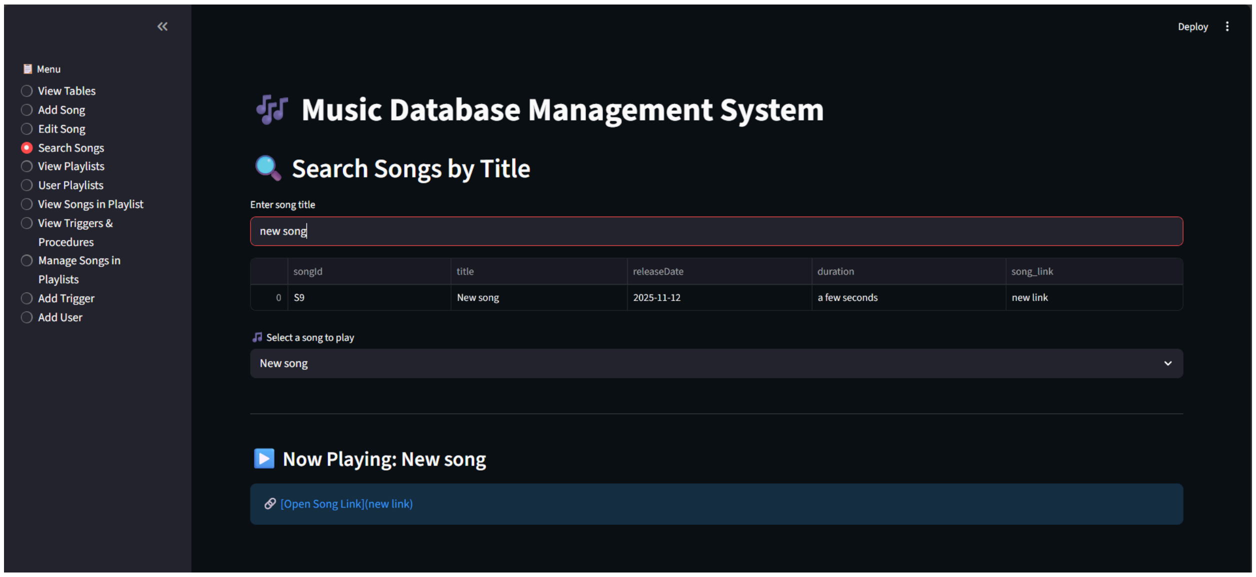
Editing: New song

Title: New song

Release Date: 2025/11/12

Duration (HH:MM:SS): 0:00:25

Song Link: new link



Music Database Management System

Search Songs by Title

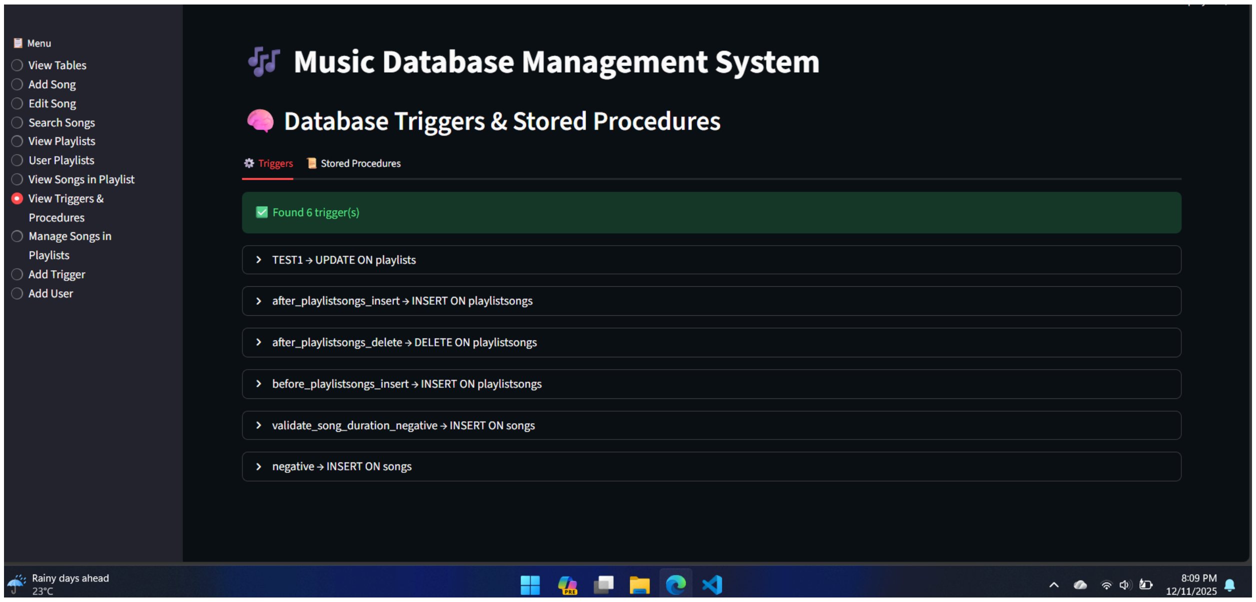
Enter song title: new song

	songId	title	releaseDate	duration	song_link
0	S9	New song	2025-11-12	a few seconds	new link

Select a song to play: New song

Now Playing: New song

[Open Song Link](new link)

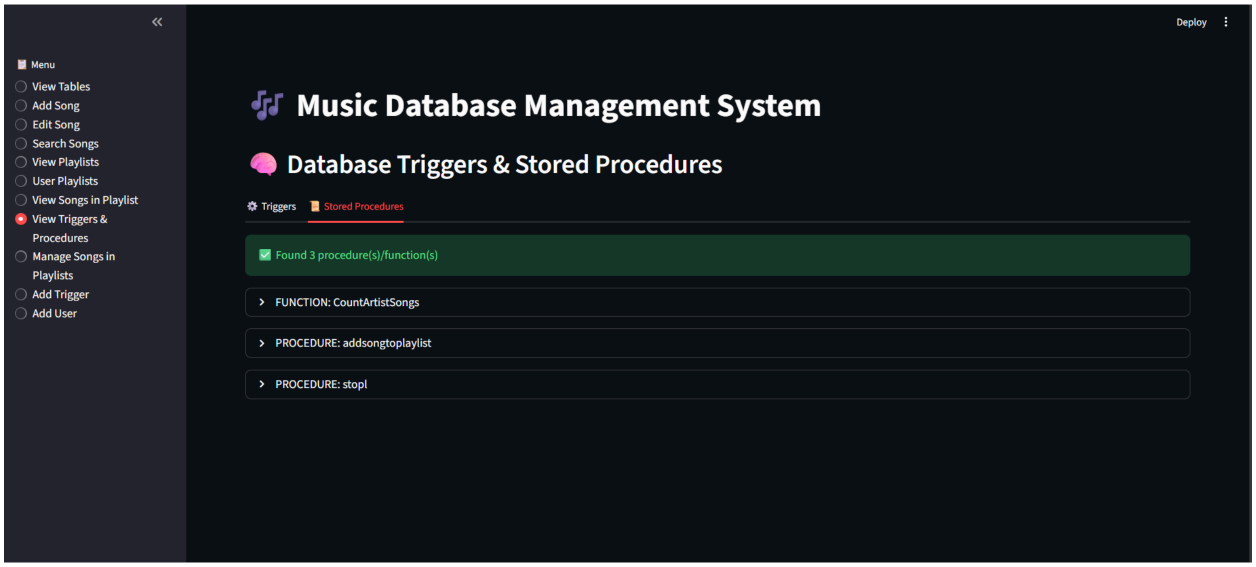


Music Database Management System

Database Triggers & Stored Procedures

Found 6 trigger(s)

- TEST1 → UPDATE ON playlists
- after_playlistsongs_insert → INSERT ON playlistsongs
- after_playlistsongs_delete → DELETE ON playlistsongs
- before_playlistsongs_insert → INSERT ON playlistsongs
- validate_song_duration_negative → INSERT ON songs
- negative → INSERT ON songs



Music Database Management System

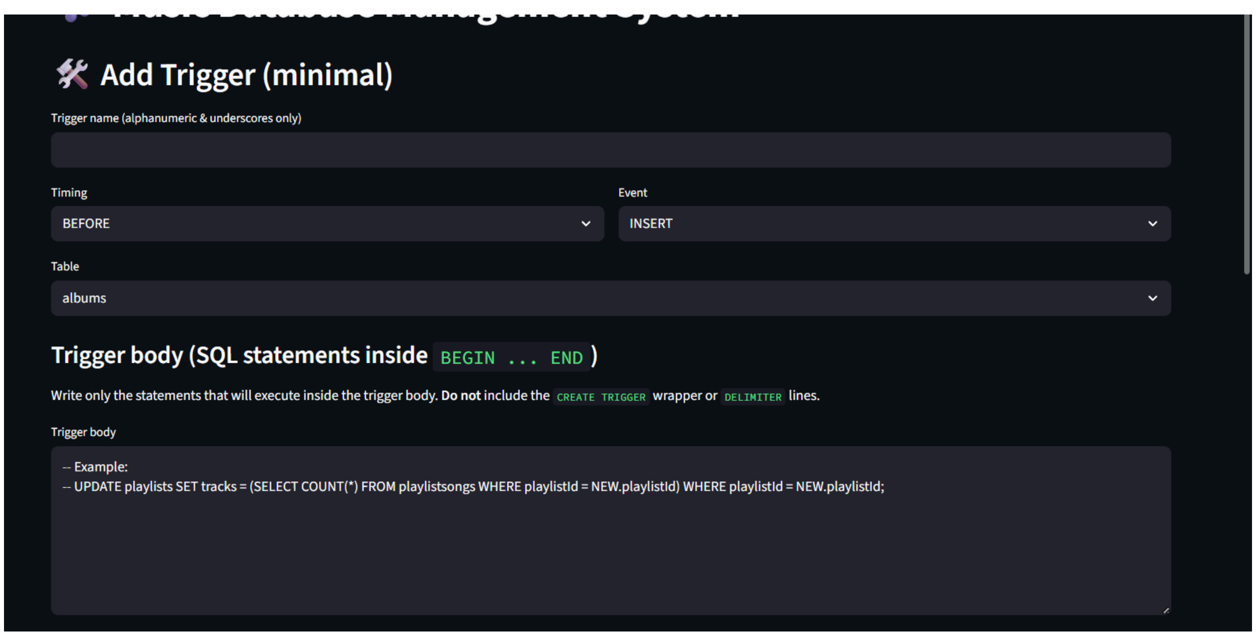
Database Triggers & Stored Procedures

Found 3 procedure(s)/function(s)

FUNCTION: CountArtistSongs

PROCEDURE: addsongtoplaylist

PROCEDURE: stopl



Add Trigger (minimal)

Trigger name (alphanumeric & underscores only)

Timing	Event
BEFORE	INSERT

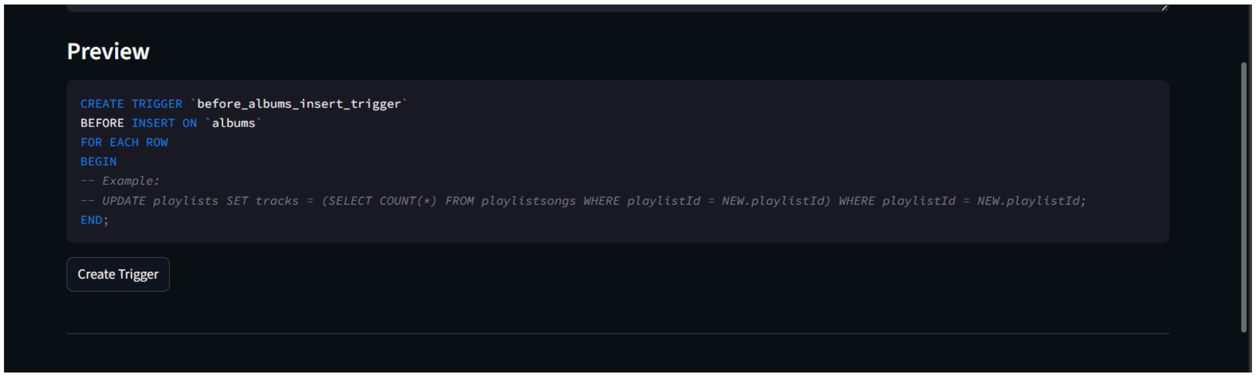
Table: albums

Trigger body (SQL statements inside `BEGIN ... END`)

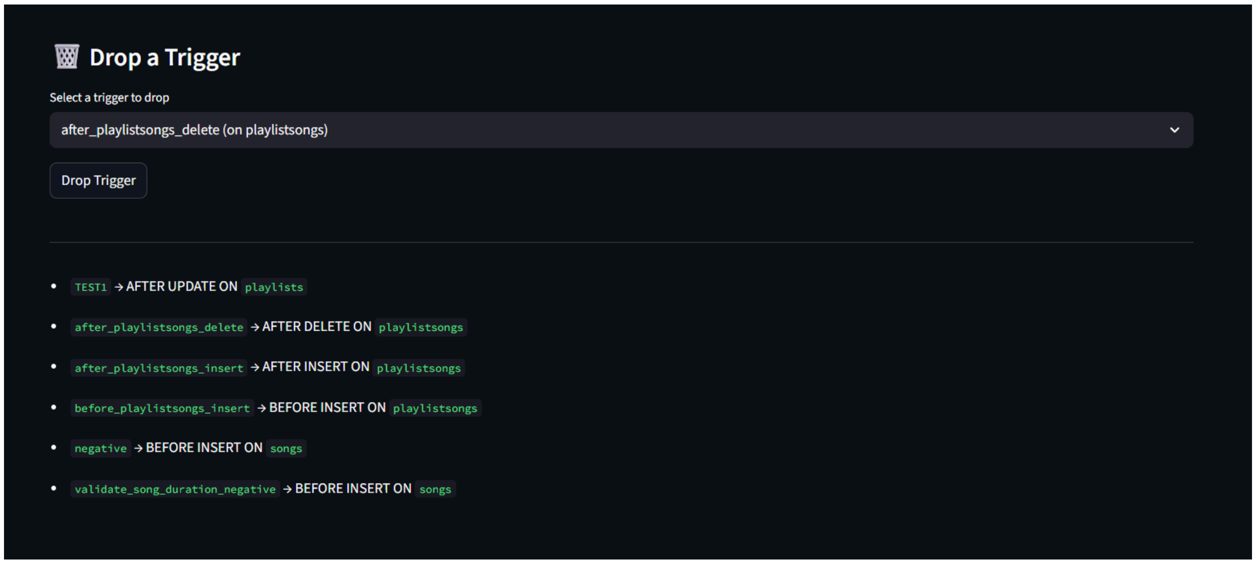
Write only the statements that will execute inside the trigger body. **Do not** include the `CREATE TRIGGER` wrapper or `DELIMITER` lines.

Trigger body

```
-- Example:
-- UPDATE playlists SET tracks = (SELECT COUNT(*) FROM playlistsongs WHERE playlistId = NEW.playlistId) WHERE playlistId = NEW.playlistId;
```


Preview

```
CREATE TRIGGER `before_albums_insert_trigger`
BEFORE INSERT ON `albums`
FOR EACH ROW
BEGIN
-- Example:
-- UPDATE playlists SET tracks = (SELECT COUNT(*) FROM playlistsongs WHERE playlistId = NEW.playlistId) WHERE playlistId = NEW.playlistId;
END;
```


🗑️ Drop a Trigger

Select a trigger to drop

after_playlistsongs_delete (on playlistsongs)

- `TEST1` → AFTER UPDATE ON `playlists`
- `after_playlistsongs_delete` → AFTER DELETE ON `playlistsongs`
- `after_playlistsongs_insert` → AFTER INSERT ON `playlistsongs`
- `before_playlistsongs_insert` → BEFORE INSERT ON `playlistsongs`
- `negative` → BEFORE INSERT ON `songs`
- `validate_song_duration_negative` → BEFORE INSERT ON `songs`

*****