**CSA0593**

**DATABASE MANAGEMENT SYSTEM**

**K. SWETHA**

**192311404**

**ASSIGNMENT - 4**

**"Design a database for a collaborative music creation platform with users, tracks, collaborations, and reviews."  
  
-Model tables for users, music tracks, collaborations between users, and track reviews.**

**-Write stored procedures for uploading tracks, initiating collaborations, and submitting reviews.**

**-Implement triggers to update track popularity and collaboration statistics.**

**-Write SQL queries to analyse user engagement, most collaborative users, and highest-rated tracks.**

**Entities and Relationships**

1. **Users**: Represents platform users who can create, collaborate, and review tracks.
2. **Tracks**: Represents music tracks created by users, either individually or collaboratively.
3. **Collaborations**: Represents the collaboration between users on a particular track.
4. **Reviews**: Represents reviews provided by users for tracks they’ve listened to.

**Entity Relationship Diagram (ERD) Overview**

* **Users** can create multiple **Tracks** and collaborate on **Tracks** with other **Users** through **Collaborations**.
* **Users** can leave **Reviews** for **Tracks** they have listened to.

**Tables and Columns**

**1. Users Table**

This table stores information about the platform users.

CREATE TABLE Users (

UserID INT PRIMARY KEY AUTO\_INCREMENT,

Username VARCHAR(255) NOT NULL,

Email VARCHAR(255) UNIQUE NOT NULL,

PasswordHash VARCHAR(255) NOT NULL,

FirstName VARCHAR(100),

LastName VARCHAR(100),

ProfilePictureURL VARCHAR(255),

Bio TEXT,

DateJoined DATETIME DEFAULT CURRENT\_TIMESTAMP

);

#### ****2. Tracks Table****

This table stores details of individual tracks created by users.

CREATE TABLE Tracks (

TrackID INT PRIMARY KEY AUTO\_INCREMENT,

Title VARCHAR(255) NOT NULL,

Genre VARCHAR(100),

Duration INT, FileURL VARCHAR(255),

CreatedBy INT,

ReleaseDate DATETIME DEFAULT CURRENT\_TIMESTAMP,

Status VARCHAR(50) DEFAULT 'Draft'

FOREIGN KEY (CreatedBy) REFERENCES Users(UserID)

);

#### ****3. Collaborations Table****

This table stores data on collaborations between users on a particular track.

CREATE TABLE Collaborations (

CollaborationID INT PRIMARY KEY AUTO\_INCREMENT,

TrackID INT,

UserID INT,

Role VARCHAR(100), DateJoined DATETIME DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (TrackID) REFERENCES Tracks(TrackID),

FOREIGN KEY (UserID) REFERENCES Users(UserID)

);

#### ****4. Reviews Table****

This table stores user reviews for tracks.

CREATE TABLE Reviews (

ReviewID INT PRIMARY KEY AUTO\_INCREMENT,

TrackID INT,

UserID INT,

Rating INT CHECK (Rating BETWEEN 1 AND 5 Comment TEXT,

DateReviewed DATETIME DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (TrackID) REFERENCES Tracks(TrackID),

FOREIGN KEY (UserID) REFERENCES Users(UserID)

);

### ****Relationships between Tables****

* **Users → Tracks**: A user can create multiple tracks.
* **Users → Collaborations → Tracks**: A user can collaborate with others on a track (many-to-many relationship between users and tracks).
* **Users → Reviews → Tracks**: A user can leave multiple reviews for different tracks, and each review is associated with a specific track.
* **Tracks → Reviews**: Each track can have multiple reviews from different users.

**-Write stored procedures for uploading tracks, initiating collaborations, and submitting reviews:**

### ****1. Stored Procedure for Uploading a Track****

This stored procedure allows a user to upload a new track to the platform, including details like title, genre, duration, file URL, and track status.

#### ****Procedure: UploadTrack****

DELIMITER $$

CREATE PROCEDURE UploadTrack(

IN p\_Title VARCHAR(255),

IN p\_Genre VARCHAR(100),

IN p\_Duration INT, -- Duration in seconds

IN p\_FileURL VARCHAR(255),

IN p\_CreatedBy INT -- UserID of the track creator

)

BEGIN

INSERT INTO Tracks (Title, Genre, Duration, FileURL, CreatedBy, Status)

VALUES (p\_Title, p\_Genre, p\_Duration, p\_FileURL, p\_CreatedBy, 'Draft');

END $$

DELIMITER ;

#### ****Explanation:****

* This procedure takes the track details as input parameters (title, genre, duration, file URL, and the user ID of the creator).
* It inserts a new track into the Tracks table with the status set to Draft by default (the track can later be published or changed to other statuses).

#### ****Usage Example:****

CALL UploadTrack('My New Song', 'Pop', 210, 'url\_to\_file.mp3', 1);

### ****2. Stored Procedure for Initiating a Collaboration****

This procedure allows a user to invite another user to collaborate on a specific track, including specifying their role (e.g., Producer, Vocalist).

#### ****Procedure: InitiateCollaboration****

DELIMITER $$

CREATE PROCEDURE InitiateCollaboration(

IN p\_TrackID INT,

IN p\_UserID INT,

IN p\_Role VARCHAR(100)

)

BEGIN

INSERT INTO Collaborations (TrackID, UserID, Role)

VALUES (p\_TrackID, p\_UserID, p\_Role);

END $$

DELIMITER ;

#### ****Explanation:****

* This procedure takes the TrackID, UserID (collaborator), and Role (e.g., Producer, Vocalist) as input.
* It inserts a record into the Collaborations table, linking the user to the track with a specified role.

#### ****Usage Example:****

CALL InitiateCollaboration(1, 2, 'Producer');

### ****3. Stored Procedure for Submitting a Review****

This stored procedure allows a user to submit a review for a specific track, including a rating (1-5) and an optional comment.

#### ****Procedure: SubmitReview****

DELIMITER $$

CREATE PROCEDURE SubmitReview(

IN p\_TrackID INT,

IN p\_UserID INT,

IN p\_Rating INT,

IN p\_Comment TEXT

)

BEGIN

INSERT INTO Reviews (TrackID, UserID, Rating, Comment)

VALUES (p\_TrackID, p\_UserID, p\_Rating, p\_Comment);

END $$

DELIMITER ;

#### ****Explanation:****

* This procedure takes the TrackID, UserID (reviewer), Rating (1-5), and Comment as inputs.
* It inserts the review into the Reviews table for the given track.

#### ****Usage Example:****

CALL SubmitReview(1, 2, 5, 'Amazing track, I love the production!');

**-Implement triggers to update track popularity and collaboration statistics:**

To implement triggers that update **track popularity** and **collaboration statistics** on the collaborative music creation platform, we can define triggers that automatically update certain fields in the Tracks table when specific actions occur. For example, when a review is submitted or a new collaboration is added, the system can automatically update the **popularity** of the track or the **collaboration count** for the track.

### 1. ****Trigger to Update Track Popularity Based on Reviews****

Each time a new review is submitted, we can update the **popularity score** of the track. The popularity could be based on the average rating of the track.

#### ****Trigger: UpdateTrackPopularity****

This trigger will update the Popularity field in the Tracks table whenever a new review is added.

DELIMITER $$

CREATE TRIGGER UpdateTrackPopularity

AFTER INSERT ON Reviews

FOR EACH ROW

BEGIN

DECLARE avg\_rating DECIMAL(3,2);

SELECT AVG(Rating) INTO avg\_rating

FROM Reviews

WHERE TrackID = NEW.TrackID;

UPDATE Tracks

SET Popularity = avg\_rating

WHERE TrackID = NEW.TrackID;

END $$

DELIMITER ;

#### ****Explanation:****

* The trigger is fired **after** a new review is inserted into the Reviews table.
* It calculates the **average rating** (avg\_rating) for the track using all reviews for that track.
* The Popularity field of the track is updated with this average rating, which reflects the track’s overall popularity.

#### ****Assumptions:****

* There is a field Popularity in the Tracks table to store the average rating.
* You can choose any method for calculating popularity, such as average ratings, the number of reviews, or a weighted system.

### 2. ****Trigger to Update Collaboration Count****

Each time a new user collaborates on a track, we can update the **collaboration count** for that track. This helps track how many collaborators a specific track has.

#### ****Trigger: UpdateCollaborationCount****

This trigger will update the CollaborationCount field in the Tracks table whenever a new collaboration is added.

DELIMITER $$

CREATE TRIGGER UpdateCollaborationCount

AFTER INSERT ON Collaborations

FOR EACH ROW

BEGIN

UPDATE Tracks

SET CollaborationCount = (

SELECT COUNT(\*)

FROM Collaborations

WHERE TrackID = NEW.TrackID

)

WHERE TrackID = NEW.TrackID;

END $$

DELIMITER ;

#### ****Explanation:****

* This trigger is fired **after** a new record is inserted into the Collaborations table.
* It counts the number of collaborators for the track (COUNT(\*)), and updates the CollaborationCount field in the Tracks table with this value.

#### ****Assumptions:****

* There is a field CollaborationCount in the Tracks table to store the number of collaborators.
* This field will store the current number of users collaborating on the track.

### ****Assumptions for Database Design****

1. **Tracks Table**: The Tracks table should have two additional fields to store track statistics:
   * Popularity (to store the average rating of the track)
   * CollaborationCount (to store the number of collaborators on the track)

Example schema for the Tracks table:

CREATE TABLE Tracks (

TrackID INT PRIMARY KEY AUTO\_INCREMENT,

Title VARCHAR(255) NOT NULL,

Genre VARCHAR(100),

Duration INT, -- Duration in seconds

FileURL VARCHAR(255),

CreatedBy INT,

ReleaseDate DATETIME DEFAULT CURRENT\_TIMESTAMP,

Status VARCHAR(50) DEFAULT 'Draft',

Popularity DECIMAL(3,2) DEFAULT 0.00, -- New column for track popularity

CollaborationCount INT DEFAULT 0, -- New column for collaboration count

FOREIGN KEY (CreatedBy) REFERENCES Users(UserID)

);

1. **Reviews Table**: The Reviews table should store the Rating (1-5) for each review, which is used to calculate the **track popularity**.
2. **Collaborations Table**: The Collaborations table should capture each user’s collaboration on a track, helping to calculate the **collaboration count**.

**-Write SQL queries to analyse user engagement, most collaborative users, and highest-rated tracks**:

### ****1. Analyze User Engagement:****

To analyze user engagement, we can look at metrics like:

* The number of tracks a user has created.
* The number of collaborations a user has participated in.
* The number of reviews a user has left.

#### ****Query: User Engagement****

SELECT

u.UserID,

u.Username,

COUNT(DISTINCT t.TrackID) AS TracksCreated, COUNT(DISTINCT c.TrackID) AS Collaborations, COUNT(DISTINCT r.ReviewID) AS ReviewsGiven (COUNT(DISTINCT t.TrackID) + COUNT(DISTINCT c.TrackID) + COUNT(DISTINCT r.ReviewID)) AS TotalEngagement

FROM

Users u

LEFT JOIN

Tracks t ON u.UserID = t.CreatedBy

LEFT JOIN

Collaborations c ON u.UserID = c.UserID

LEFT JOIN

Reviews r ON u.UserID = r.UserID

GROUP BY

u.UserID, u.Username

ORDER BY

TotalEngagement DESC;

#### ****Explanation:****

* The query calculates engagement based on the number of **tracks created**, **collaborations** a user is involved in, and **reviews** they have submitted.
* It uses **LEFT JOIN** to ensure that even users who haven't created tracks or collaborated on any can still be counted in the results.
* The result is ordered by the **TotalEngagement** score, which sums the user's activity across these metrics.

### ****2. Most Collaborative Users:****

To find the most collaborative users, we focus on the number of collaborations each user has participated in.

#### ****Query: Most Collaborative Users****

SELECT

u.UserID,

u.Username,

COUNT(c.CollaborationID) AS CollaborationsCount

FROM

Users u

JOIN

Collaborations c ON u.UserID = c.UserID

GROUP BY

u.UserID, u.Username

ORDER BY

CollaborationsCount DESC;

### ****3. Highest-Rated Tracks:****

To find the highest-rated tracks, we need to compute the **average rating** for each track and sort them by the highest average.

#### ****Query: Highest-Rated Tracks****

SELECT

t.TrackID,

t.Title,

AVG(r.Rating) AS AverageRating

FROM

Tracks t

JOIN

Reviews r ON t.TrackID = r.TrackID

GROUP BY

t.TrackID, t.Title

ORDER BY

AverageRating DESC

LIMIT 10;

#### ****Explanation:****

* The query calculates the **average rating** for each track by joining the Tracks and Reviews tables.
* It then groups the results by TrackID and Title and orders them by AverageRating in descending order.
* The LIMIT 10 ensures that only the top 10 highest-rated tracks are returned.

### ****4. Bonus: Tracks with the Most Reviews (Popular Tracks)****

To identify tracks that are popular based on the number of reviews they have received, you can use the following query:

#### ****Query: Tracks with the Most Reviews****

SELECT

t.TrackID,

t.Title,

COUNT(r.ReviewID) AS ReviewCount

FROM

Tracks t

JOIN

Reviews r ON t.TrackID = r.TrackID

GROUP BY

t.TrackID, t.Title

ORDER BY

ReviewCount DESC

LIMIT 10;

#### ****Explanation:****

* This query counts the number of **reviews** each track has received by joining the Tracks and Reviews tables.
* The result is ordered by ReviewCount in descending order to find the tracks that have the most reviews.

**CONCLUSION:**

The designed database for a collaborative music creation platform effectively facilitates user engagement, collaboration, and track review processes. Through the implementation of various SQL queries and stored procedures, it allows for seamless uploading of tracks, initiating collaborations, and submitting reviews. Triggers ensure automatic updates to track popularity and collaboration statistics, enhancing data consistency. The queries for analyzing user engagement, identifying the most collaborative users, and highlighting the highest-rated tracks provide valuable insights into platform activity, helping identify key contributors and popular content. Overall, this system supports dynamic and efficient interactions among users, fostering creativity and collaboration while maintaining robust data tracking.