USE OF TRIGGERS, FUNCTIONS, STORED PROCEDURES AND VIEWS:

STORED PROCEDURES

In our project, there are 11 stored procedures.

1. When a new user registers, this procedure adds the users details in the users table.

```
DELIMITER $$
CREATE PROCEDURE AddNewUser(
    IN p_user_name VARCHAR(255),
    IN p_email VARCHAR(255),
    IN p_user_password VARCHAR(512),
    IN p_date_of_birth DATE,
    OUT p_user_id INT
)
BEGIN
    INSERT INTO users (User_Name, Email, User_Password, Date_Of_Birth)
    VALUES (p_user_name, p_email, p_user_password, p_date_of_birth);
    SELECT LAST_INSERT_ID() INTO p_user_id;
END $$
DELIMITER ;
```

2. This stored procedure searches for song titles based on a provided search term.

```
DELIMITER //

CREATE PROCEDURE SearchSongs(IN search_term VARCHAR(100))
BEGIN
    SELECT * FROM song
    WHERE song_title LIKE CONCAT('%', search_term, '%');
END //
DELIMITER ;
```

3. This procedure will retrieve information of a specific artist given their Artist_ID.

```
DELIMITER $$
CREATE PROCEDURE GetArtistInformation(
IN artistID INT,
OUT artistName VARCHAR(255),
OUT artistBirthdate DATE,
OUT artistGender VARCHAR(10),
OUT artistHeight DECIMAL(3,2),
OUT artistNationality VARCHAR(255)
)
BEGIN
SELECT Full_Name,Birthdate,Gender,Height,Nationality
INTO artistName,artistBirthdate,artistGender,artistHeight,artistNationality
FROM Artist
WHERE Artist_ID = artistID;
END $$
DELIMITER ;
```

4. This procedure gives a list of top rated songs based on user ratings.

```
DELIMITER $$
CREATE PROCEDURE GetTopRatedSongs(IN topCount INT)
BEGIN
SELECT s.Song_ID,s.Song_Title,s.Time_Period,s.Musical_Key,s.Album_ID,s.Artist_ID,s.Genre_ID,
    AVG(us.Rating) AS Average_Rating
FROM Song s
JOIN User_Song_Rating us ON s.Song_ID = us.Song_ID
GROUP BY s.Song_ID, s.Song_Title, s.Time_Period, s.Musical_Key, s.Album_ID, s.Artist_ID, s.Genre_ID
ORDER BY Average_Rating DESC
LIMIT topCount;
END $$
DELIMITER ;
```

5. This procedure updates information about a specific artist.

```
DELIMITER $$
CREATE PROCEDURE UpdateArtistInformation(
IN artistID INT,
IN newFullName VARCHAR(255),
IN newBirthdate DATE,
IN newGender CHAR(1),
IN newHeight DECIMAL(5,2),
IN newNationality VARCHAR(255))
BEGIN
UPDATE Artist
SET Full_Name = newFullName, Birthdate = newBirthdate, Gender = newGender, Height = newHeight,
    Nationality = newNationality
WHERE Artist_ID = artistID;
END $$
DELIMITER ;
```

6. This procedure will retrieve billboard chart data for a specific song.

```
DELIMITER $$
CREATE PROCEDURE GetBillboardChartData(
    IN songID INT)
BEGIN
    SELECT Chart_ID, Daily_Rank, Daily_Movement, Weekly_Movement
    FROM Billboard_Chart_Data
    WHERE Song_ID = songID;
END $$
DELIMITER ;
```

7. This procedure calculates the average duration of songs for each genre in the database and store the results in a temporary table called AvgDurationsTable.

```
DELIMITER $$
CREATE PROCEDURE GetAverageSongDurationForEachGenre()
BEGIN DECLARE done INT DEFAULT FALSE;
    DECLARE genreID INT;
    DECLARE genreName VARCHAR(255);
    DECLARE avgSongDuration DECIMAL(10,2);
    DECLARE cur CURSOR FOR
        SELECT g.Genre_ID, g.Genre_Name, AVG(s.Time_Period) AS Avg_Song_Duration
        FROM Genre g
        JOIN Song s ON g.Genre_ID = s.Genre_ID
        GROUP BY g.Genre_ID, g.Genre_Name;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
    CREATE TEMPORARY TABLE IF NOT EXISTS AvgDurationsTable ( Genre_ID INT, Genre_Name VARCHAR(255),
        Avg_Song_Duration DECIMAL(10,2));
    OPEN cur;
    FETCH cur INTO genreID, genreName, avgSongDuration;
    WHILE NOT done DO
        INSERT INTO AvgDurationsTable VALUES (genreID, genreName, avgSongDuration);
        FETCH cur INTO genreID, genreName, avgSongDuration;
    END WHILE;
    CLOSE cur;
    SELECT * FROM AvgDurationsTable;
END $$
DELIMITER ;
```

8. This procedure allows users to add songs to their Favorites.

```
DELIMITER $$
CREATE PROCEDURE AddToFavorites(IN p_user_id INT,IN p_song_id INT)
BEGIN
    IF NOT EXISTS (
        SELECT 1
        FROM UserFavorites
        WHERE User_ID = p_user_id AND Song_ID = p_song_id
    ) THEN
        INSERT INTO UserFavorites (User_ID, Song_ID)
        VALUES (p_user_id, p_song_id);
    END IF;
END $$
DELIMITER ;
```

9. This procedure allows users to add songs to their playlist.

```
DELIMITER $$
CREATE PROCEDURE AddToPlaylist(IN p_playlist_id INT, IN p_song_id INT)
BEGIN
    IF NOT EXISTS (
        SELECT 1
        FROM PlaylistSongs
        WHERE Playlist_ID = p_playlist_id AND Song_ID = p_song_id
    ) THEN
        INSERT INTO PlaylistSongs (Playlist_ID, Song_ID)
        VALUES (p_playlist_id, p_song_id);
    END IF;
END $$
DELIMITER ;
```

10. This procedure allows users to delete songs from their playlist.

```
DELIMITER $$
CREATE PROCEDURE DeleteFromPlaylist(
    IN p_playlist_id INT,
    IN p_song_id INT
)
BEGIN
    DELETE FROM playlist
    WHERE Playlist_ID = p_playlist_id AND Song_ID = p_song_id;
END $$
DELIMITER ;
```

11. This procedure allows users to delete songs from their favourites.

```
DELIMITER $$
CREATE PROCEDURE DeleteFromFavorites(
    IN p_user_id INT,
    IN p_song_id INT
)
BEGIN
    DELETE FROM UserFavorites
    WHERE User_ID = p_user_id AND Song_ID = p_song_id;
END $$
DELIMITER ;
```

FUNCTIONS

In our project, there are 3 functions.

1. This function called GetTotalAlbumsByArtist accepts an Artist_ID and returns the total number of albums released by that artist.

```
DELIMITER $$
CREATE FUNCTION GetTotalAlbumsByArtist(
    artistID INT)
RETURNS INT
DETERMINISTIC
BEGIN
    DECLARE totalAlbums INT;
    SELECT COUNT(*) INTO totalAlbums
    FROM Album
    WHERE Artist_ID = artistID;
    RETURN totalAlbums;
END $$
DELIMITER ;
```

2. This function named GetGenreForSong takes a Song_ID and returns the genre of the corresponding song.

```
DELIMITER $$
CREATE FUNCTION GetGenreForSong(
    songID INT)
RETURNS VARCHAR(255)
DETERMINISTIC
BEGIN
    DECLARE genreName VARCHAR(255);
    SELECT g.Genre_Name INTO genreName
    FROM Song s
    JOIN Genre g ON s.Genre_ID = g.Genre_ID
    WHERE s.Song_ID = songID;
    RETURN genreName;
END $$
DELIMITER ;
```

3. This function called GetMostRecentAlbumReleaseDate takes an Artist_ID and returns the most recent release date of any album by that artist.

```
DELIMITER $$
CREATE FUNCTION GetMostRecentAlbumReleaseDate(
    artistID INT)
RETURNS DATE
DETERMINISTIC
BEGIN
    DECLARE mostRecentDate DATE;
    SELECT MAX(a.Release_Date) INTO mostRecentDate
    FROM Album a
    WHERE a.Artist_ID = artistID;
    RETURN mostRecentDate;
END $$
DELIMITER ;
```

TRIGGERS

We have one trigger in our project:

It prevents the deletion of a song if it has received awards.

```
CREATE TRIGGER PreventAlbumDeletion
BEFORE DELETE ON Song
FOR EACH ROW
BEGIN
    DECLARE awardCount INT;
    SELECT COUNT(*) INTO awardCount
    FROM Awards
    WHERE Song_ID = OLD.Song_ID;
    IF awardCount > 0 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Cannot delete album with associated awards';
    END IF;
END $$
DELIMITER ;
```

VIEWS

We have 3 views in our project.

1. This view (TopRatedSongs) joins the Song, User_Song_Rating, and Artist tables. It calculates the average user rating for each song and includes the song ID, title, artist name, and average rating in the view.

```sql
CREATE VIEW TopRatedSongs AS
SELECT
    s.Song_ID,
    s.Song_Title,
    a.Full_Name AS Artist_Name,
    AVG(usr.Rating) AS Average_Rating
FROM
    Song s
JOIN
    User_Song_Rating usr ON s.Song_ID = usr.Song_ID
JOIN
    Artist a ON s.Artist_ID = a.Artist_ID
GROUP BY
    s.Song_ID, s.Song_Title, a.Full_Name;
```

2. This view named LabelAlbumStatistics that includes information about labels, such as label ID, label name, label country, and the total number of albums released by each label, excluding labels with no albums.

```sql
CREATE VIEW LabelAlbumStatistics AS
SELECT
    l.Label_ID,
    l.Label_Name,
    l.Country AS Label_Country,
    COUNT(DISTINCT a.Album_ID) AS Total_Albums
FROM
    Label l
LEFT JOIN
    Album a ON l.Label_ID = a.Label_ID
WHERE
    a.Album_ID IS NOT NULL
GROUP BY
    l.Label_ID, l.Label_Name, l.Country;
```

3. This view includes statistics for each genre, including the number of songs, average song duration, and the most common musical key.

```sql
CREATE VIEW GenreStatistics AS
SELECT g.Genre_ID, g.Genre_Name, COUNT(s.Song_ID) AS Number_of_Songs,
    AVG(s.Time_Period) AS Average_Song_Duration,
    SUBSTRING_INDEX(GROUP_CONCAT(s.Musical_Key ORDER BY keyCounts.KeyCount DESC), ',', 1)
    AS Most_Common_Musical_Key
FROM Genre g
JOIN Song s ON g.Genre_ID = s.Genre_ID
JOIN (SELECT Song_ID, Musical_Key, COUNT(Musical_Key) AS KeyCount
    FROM Song
    GROUP BY Song_ID, Musical_Key) AS keyCounts ON s.Song_ID = keyCounts.Song_ID
GROUP BY g.Genre_ID, g.Genre_Name;
```

The number_of_songs column denotes the number of songs belonging to that genre while the average_song_duration column denotes the average song duration in seconds. The last column denotes the most_common_musical_key used in the songs of that particular genre.

Genre_ID	Genre_Name	Number_of_songs	Average_song_duration	Most_common_musical_key
111111	Art Punk	1	167303	6
111116	Crust Punk (thanks Haug)	1	391888	1
111119	Folk Punk	1	172797	7
111120	Goth / Gothic Rock	2	278641.5	5
111121	Grunge	1	212953	2
111122	Hardcore Punk	1	165582	1
111127	New Wave	1	189901	4
111129	Punk	1	244684	8
111134	Blues Rock	1	191959	6
111136	British Blues	1	319369	10
111142	Contemporary R&B	1	178426	9
111146	Detroit Blues	1	260111	8