Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Mini-Project Report

Academic Year 2024-2025

Course: Database Technologies Laboratory   Course Code: DJS23SCMD301

Class: S.Y.B.Tech.   Semester: III   Division: S

Department: Artificial Intelligence (AI) and Data Science   Batch: A1

Music Album Management System

Prepared by

Sr. No.	Roll No.	Name	SAP ID
1	S012	Dhwaj Jain	60018230070
2	S014	Eric Kurissery	60018230071
3	S015	Hadi Gala	60018230085
4	S026	Kaustav Dedhia	60018230009

Table of Contents

1 Problem Statement

In the music industry, managing a large catalog of albums, artists, tracks, and associated data can be complex and challenging. Music labels and independent artists often struggle to maintain an organized system that tracks all relevant details, including album release dates, track listings, genre classifications, and artist information. Without a unified system, data management becomes fragmented, leading to inconsistent metadata, delays in updating catalog information, and missed opportunities for marketing and distribution. This Music Album Management System project aims to provide a comprehensive solution for organizing, managing, and tracking music catalog data efficiently, ensuring all album-related information is centralized and easily accessible.

2 Project Idea

The Music Album Management System project aims to offer an integrated solution for music labels, artists, and distributors to manage their album catalog efficiently. This system consolidates essential functionalities into a single, user-friendly platform, allowing users to handle various aspects of music catalog data, such as artist details, album information, track listings, release dates, genres, and distribution channels. By organizing these modules into a cohesive system, the project helps maintain accurate and up-to-date records, enabling streamlined catalog management and effective data tracking.

The project includes a graphical user interface (GUI) that allows users to perform key tasks, such as adding, viewing, updating, or deleting records across different modules. Each table, such as “Singers,” “Albums,” “Tracks,” and “Genres,” represents a specific aspect of the music management ecosystem, with the system facilitating smooth data operations while ensuring data integrity and enforcing relationships between tables. For example, each album is linked to an artist, contains multiple tracks, and is classified by genre and release date, creating a comprehensive, interconnected dataset.

By implementing this system, users can reduce errors, enhance data accessibility, and make informed decisions regarding catalog distribution, marketing, and licensing opportunities based on real-time data. The system provides a scalable foundation that could be expanded with additional features, such as automated royalty calculations, performance analytics, or integration with digital streaming platforms, to meet evolving industry needs.

3 Tech Stack

- **Python:** Programming language to connect with MySQL database and create GUI Application.
- **Tkinter:** Python library for creating the graphic user interface (GUI), providing an interactive front-end to users.
- **MySQL:** SQL Database to store, access, update and manage Inventory data.
- **mysql-connector-python:** Python library used to connect MySQL Database with Python for database communication.
- **Pandas:** Python data manipulation library used to provide data frames and represent the data in structured form in GUI application.

4 Code

Python Code:

```
import mysql.connector

from tkinter import *

from tkinter import messagebox, ttk


# Database connection setup (replace with your actual MySQL details)

def create_connection():

    try:

        return mysql.connector.connect(

            host="localhost",

            user="root",

            password="pass@123",

            database="music_album_management_system"

        )

    except mysql.connector.Error as e:
        messagebox.showerror("Connection Error", f"Error connecting to database: {e}")
        return None


# Main application class
class MusicAlbumApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Music Album Management System")
        self.root.geometry("800x600")


        # Label and buttons for each table
        Label(self.root, text="Select a Table to Manage", font=("Arial", 18)).pack(pady=10)


        Button(self.root, text="Manage Songs", command=self.manage_songs, width=20).pack(pady=5)
        Button(self.root, text="Manage Labels", command=self.manage_labels, width=20).pack(pady=5)
        Button(self.root, text="Manage Movies", command=self.manage_movies, width=20).pack(pady=5)
        Button(self.root, text="Manage Albums", command=self.manage_albums, width=20).pack(pady=5)
        Button(self.root, text="Manage Lyrics", command=self.manage_lyrics, width=20).pack(pady=5)
        Button(self.root, text="Manage Tracks", command=self.manage_tracks, width=20).pack(pady=5)
        Button(self.root, text="Manage Singers", command=self.manage_singers, width=20).pack(pady=5)
        Button(self.root, text="Manage Genres", command=self.manage_genres, width=20).pack(pady=5)
        Button(self.root, text="Manage Composers", command=self.manage_composers,
width=20).pack(pady=5)
        Button(self.root, text="Manage Reviews", command=self.manage_reviews,
width=20).pack(pady=5)
        Button(self.root, text="Manage Purchases", command=self.manage_purchases,
width=20).pack(pady=5)
```

```
    # Functions to open management windows for each table

    def manage_songs(self):

        self.manage_table("Song", ["S_id", "S_name", "S_releasedate", "S_language", "S_duration",
"L_id"])


    def manage_labels(self):

        self.manage_table("Label", ["L_id", "L_legacy", "L_count", "L_name"])


    def manage_movies(self):

        self.manage_table("Movies", ["M_id", "M_name", "M_duration", "M_location", "L_id"])


    def manage_albums(self):

        self.manage_table("Albums", ["A_id", "A_name", "A_releasedate", "A_NoOfSongs", "S_id"])


    def manage_lyrics(self):

        self.manage_table("Lyrics", ["Ly_rhyme", "Ly_language", "Ly_pov", "S_id"])


    def manage_tracks(self):

        self.manage_table("Tracks", ["T_id", "T_count", "S_id"])


    def manage_singers(self):

        self.manage_table("Singer", ["Si_name", "Si_age", "Si_gender", "Si_rank", "Si_id"])


    def manage_genres(self):

        self.manage_table("Genre", ["G_id", "G_type", "S_id", "Si_id"])


    def manage_composers(self):
        self.manage_table("Composer", ["C_name", "C_age", "C_gender", "C_id"])


    def manage_reviews(self):
        self.manage_table("Reviews", ["R_rating", "R_date", "R_id", "S_id", "G_id", "M_id", "A_id",
"P_id", "T_id"])


    def manage_purchases(self):
        self.manage_table("Purchase", ["P_amount", "P_duration", "P_mode", "P_id", "L_id"])


    # General function to manage a table
    def manage_table(self, table_name, columns):
        # Create a new window
        table_window = Toplevel(self.root)
        table_window.title(f"Manage {table_name}")


        # Treeview to display records
        tree = ttk.Treeview(table_window, columns=columns, show='headings')
        for col in columns:
            tree.heading(col, text=col)
            tree.column(col, width=100)
        tree.pack(fill=BOTH, expand=1)


        # Database operations
        def load_records():
            conn = create_connection()
            if conn:
                cursor = conn.cursor()
                cursor.execute(f"SELECT * FROM {table_name}")
                records = cursor.fetchall()
                for row in tree.get_children():
                    tree.delete(row)
                for record in records:
                    tree.insert('', 'end', values=record)
                conn.close()


        def add_record():
            values = [entry.get() for entry in entries]
            conn = create_connection()
            if conn:
                cursor = conn.cursor()
                cursor.execute(f"INSERT INTO {table_name} ({', '.join(columns)}) VALUES ({', '.join(['%s']
* len(columns))})", values)
                conn.commit()
                conn.close()
                load_records()


        def delete_record():
            selected = tree.selection()
            if selected:
                record = tree.item(selected[0])['values']
                record_id = record[0]
                conn = create_connection()
                if conn:
                    cursor = conn.cursor()
                cursor.execute(f"DELETE FROM {table_name} WHERE {columns[0]} = %s", (record_id,))
                conn.commit()
                conn.close()
                load_records()

        def update_record():
            selected = tree.selection()
            if selected:
                record = tree.item(selected[0])['values']
                values = [entry.get() for entry in entries]
                conn = create_connection()
                if conn:
                    cursor = conn.cursor()
                    update_stmt = f"UPDATE {table_name} SET " + ", ".join([f"{col} = %s" for col in
columns[1:]]) + f" WHERE {columns[0]} = %s"
                    cursor.execute(update_stmt, values[1:] + [record[0]])
                    conn.commit()
                    conn.close()
                    load_records()

        # Entry fields and buttons
        frame = Frame(table_window)
        frame.pack(fill=X)
        entries = []
        for col in columns:
            Label(frame, text=col).pack(side=LEFT, padx=5)
            entry = Entry(frame)
            entry.pack(side=LEFT, padx=5)
            entries.append(entry)

        Button(frame, text="Add", command=add_record).pack(side=LEFT, padx=5)
        Button(frame, text="Delete", command=delete_record).pack(side=LEFT, padx=5)
        Button(frame, text="Update", command=update_record).pack(side=LEFT, padx=5)

        # Load records initially
        load_records()

# Run the application
root = Tk()
app = MusicAlbumApp(root)
root.mainloop()
```

MySQL Code:

```
create database music_album_management_system;
use music_album_management_system;
create table Song
(
S_id int primary key NOT NULL,
S_name varchar(30) NOT NULL,
S_releasedate varchar(30) NOT NULL,
S_language varchar(30),
S_duration int NOT NULL
);
desc Song;
create table Label
(
L_id int primary key NOT NULL,
L_legacy int,
L_count int NOT NULL,
L_name varchar(30) NOT NULL
);
desc Label;
create table Movies
(
M_id int primary key NOT NULL,
M_name varchar(30) NOT NULL,
M_duration int,
M_location varchar(30)
);
desc Movies;
create table Albums
(
A_id int primary key NOT NULL,
A_name varchar(30) NOT NULL,
A_releasedate varchar(30) NOT NULL,

A_NoOfSongs int NOT NULL

);

desc Albums;

create table Lyrics

(

Ly_rhyme varchar(30),

Ly_language varchar(30) primary key NOT NULL,

Ly_pov varchar(30)

);

desc Lyrics;

create table Tracks

(

T_id int primary key NOT NULL,

T_count int NOT NULL

);

desc Tracks;

create table Singer

(

Si_name varchar(30) NOT NULL,

Si_age int NOT NULL,

Si_gender varchar(30) NOT NULL,

Si_rank int NOT NULL,

Si_id int primary key NOT NULL
```

```
);

desc Singer;

create table Genre

(

G_id int NOT NULL,

G_type varchar(30) NOT NULL primary key

);

desc Genre;

create table Composer

(

C_name varchar(30) NOT NULL,

C_age int NOT NULL,

C_gender varchar(30) NOT NULL,

C_id int NOT NULL primary key

);

desc Composer;

create table Reviews

(

R_rating int NOT NULL,

R_date varchar(30) NOT NULL,

R_id int NOT NULL primary key

);

desc Reviews;

create table Purchase
```

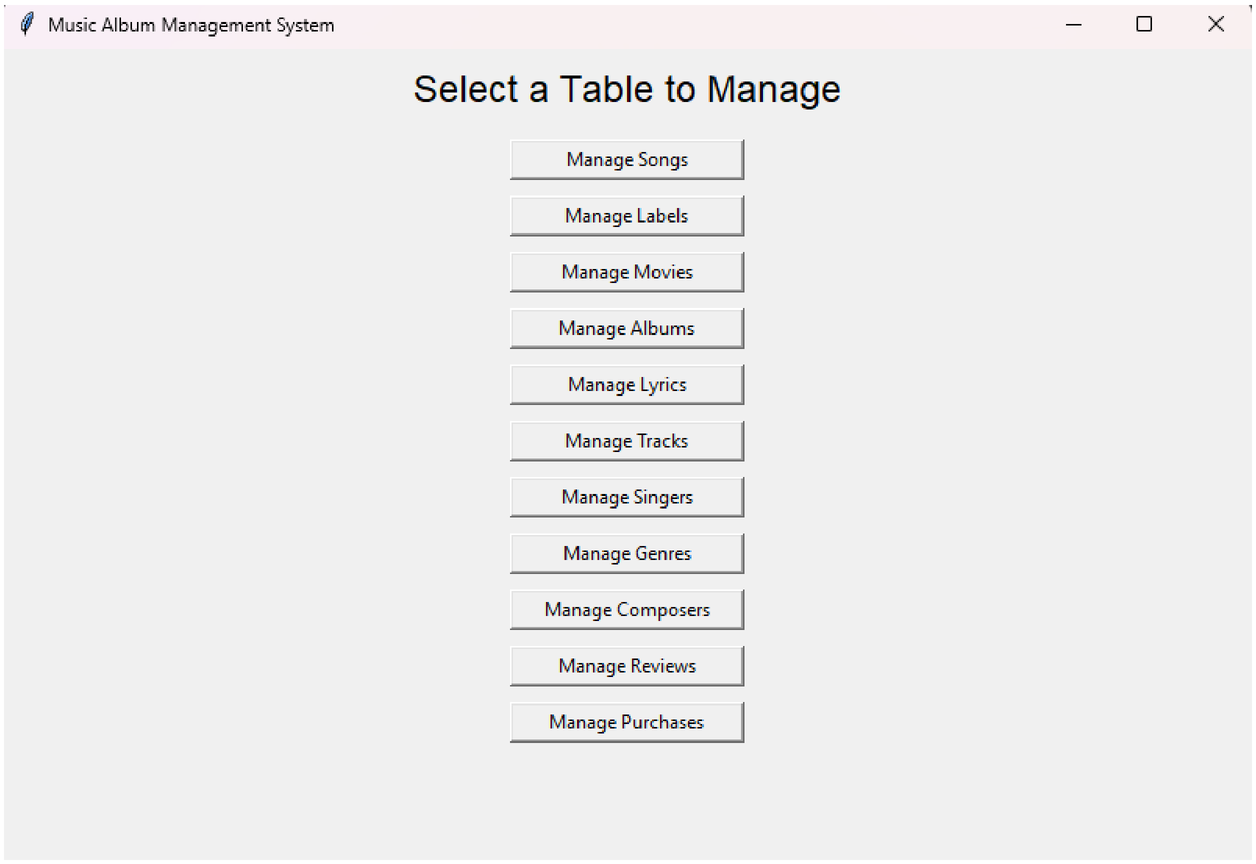
```
(

P_amount int(30) NOT NULL,

P_duration int NOT NULL check(P_duration<=12),

P_mode varchar(30) NOT NULL,

P_id int NOT NULL primary key

);

desc Purchase;

alter table Purchase

add L_id int;

alter table Purchase

add constraint fk_Purchase_Label foreign key(L_id) references Label(L_id);

alter table Movies

add L_id int;

alter table Movies

add constraint fk_Movies_Label foreign key(L_id) references Label(L_id);

alter table Song

add L_id int;

alter table Song

add constraint fk_Song_Label foreign key(L_id) references Label(L_id);

alter table Lyrics

add S_id int;

alter table Lyrics

add constraint fk_Lyrics_Song foreign key(S_id) references Song(S_id);

alter table Albums
```

```
add S_id int;

alter table Albums

add constraint fk_Albums_Song foreign key(S_id) references Song(S_id);

alter table Tracks

add S_id int;

alter table Tracks

add constraint fk_Tracks_Song foreign key(S_id) references Song(S_id);

alter table Genre

add S_id int;

alter table Genre

add constraint fk_Genre_Song foreign key(S_id) references Song(S_id);

alter table Genre

add Si_id int;

alter table Genre

add constraint fk_Genre_Singer foreign key(Si_id) references Singer(Si_id);

alter table Reviews

add S_id int;

alter table Reviews

add constraint fk_Reviews_Song foreign key(S_id) references Song(S_id);

alter table Reviews

add G_id int;

alter table Reviews

add constraint fk_Reviews_Genre foreign key(G_id) references Genre(G_id);

alter table Reviews
```

```
add M_id int;

alter table Reviews

add constraint fk_Reviews_Movies foreign key(M_id) references Movies(M_id);

alter table Reviews

add A_id int;

alter table Reviews

add constraint fk_Reviews_Albums foreign key(A_id) references Albums(A_id);

alter table Reviews

add P_id int;

alter table Reviews

add constraint fk_Reviews_Purchase foreign key(P_id) references Purchase(P_id);

alter table Reviews

add T_id int;

alter table Reviews

add constraint fk_Reviews_Tracks foreign key(T_id) references Tracks(T_id);

-- Insert correct number of values into Song table

insert into Song (S_id, S_name, S_releasedate, S_language, S_duration)

values (1, 'Popular', '20jan', 'English', 4);

insert into Song (S_id, S_name, S_releasedate, S_language, S_duration)

values (2, 'Vele', '21feb', 'Hindi', 3);

insert into Song (S_id, S_name, S_releasedate, S_language, S_duration)

values (3, 'TrueStories', '3march', 'Punjabi', 2);

select * from Song;

-- Deleting record with S_id=2
```

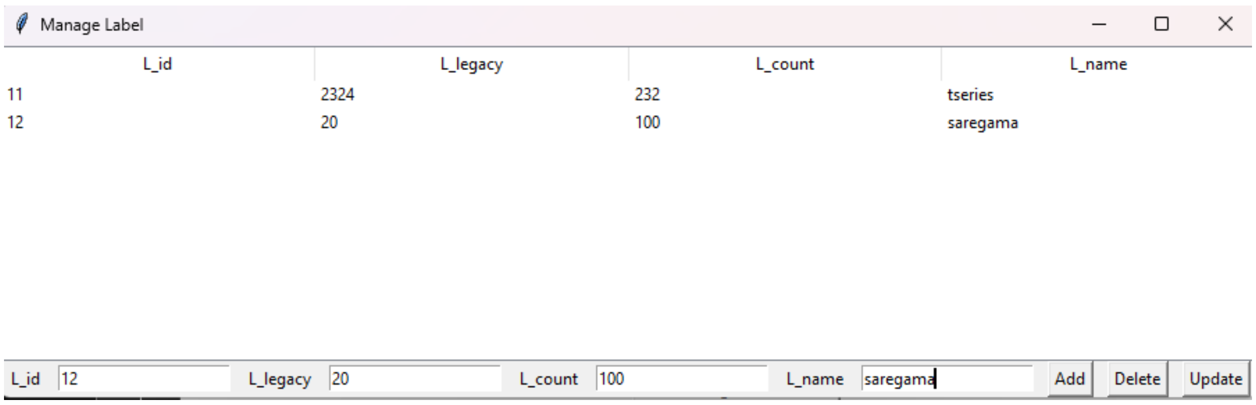
```
delete from Song where S_id=2;

select * from Song;

-- Update Song duration

update Song set S_duration = 5 where S_id=1;

select * from Song;

-- Update Song language

update Song set S_language = 'Mix' where S_id=1;

select * from Song;

-- Insert into Movies

insert into Movies values (401, 'Avatar', 201, 'USA');

insert into Movies values (103, 'Stree', 120, 'India');

select * from Movies;

-- Delete all movies

delete from Movies;

select * from Movies;
```

5 Output



Music Album Management System

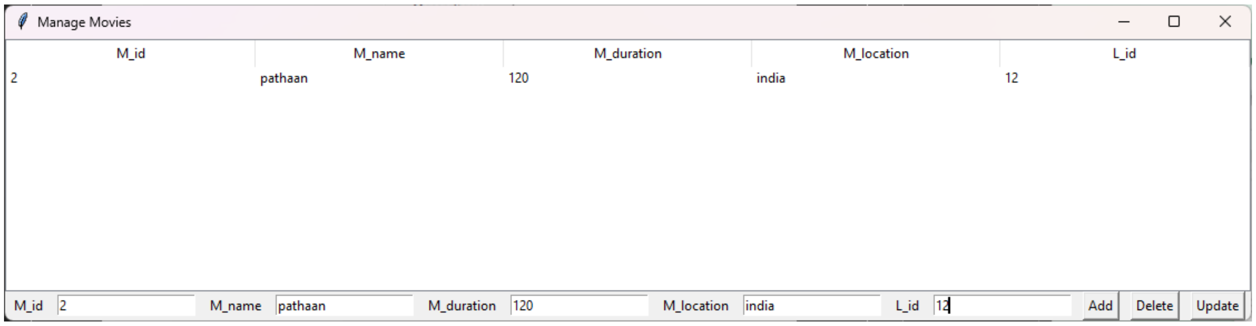
Select a Table to Manage



Manage Label

L_id	L_legacy	L_count	L_name
11	2324	232	tseries
12	20	100	saregama

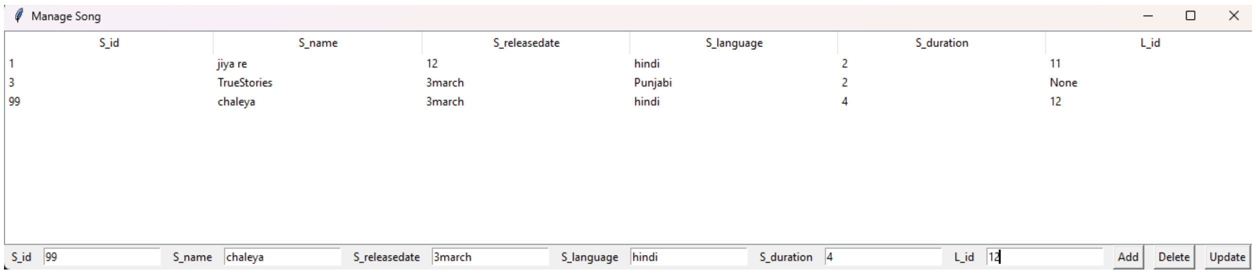
L_id 12 L_legacy 20 L_count 100 L_name saregama



Manage Movies

M_id	M_name	M_duration	M_location	L_id
2	pathaan	120	india	12

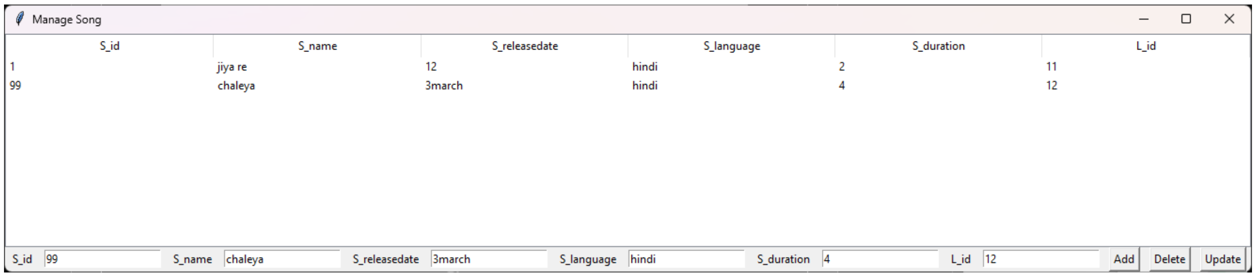
M_id 2 M_name pathaan M_duration 120 M_location india L_id 12



Manage Song

S_id	S_name	S_releasedate	S_language	S_duration	L_id
1	jiya re	12	hindi	2	11
3	TrueStories	3march	Punjabi	2	None
99	chaleya	3march	hindi	4	12

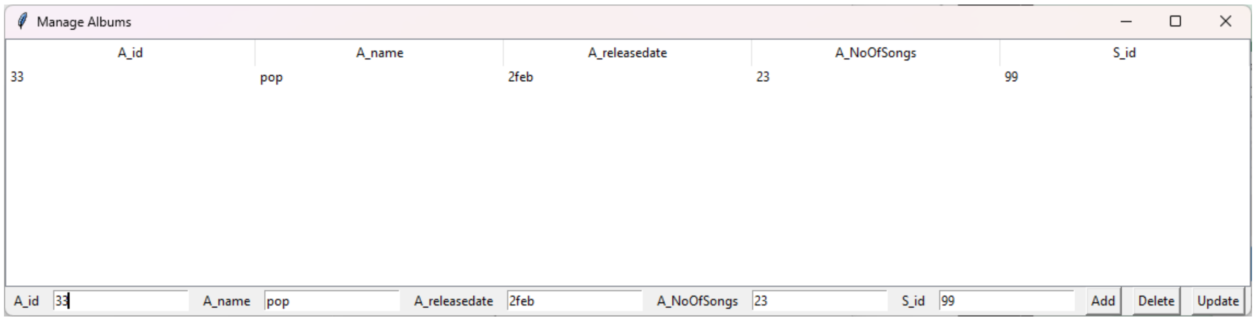
S_id 99 S_name chaleya S_releasedate 3march S_language hindi S_duration 4 L_id 12



Manage Song

S_id	S_name	S_releasedate	S_language	S_duration	L_id
1	jiya re	12	hindi	2	11
99	chaleya	3march	hindi	4	12

S_id 99 S_name chaleya S_releasedate 3march S_language hindi S_duration 4 L_id 12



Manage Albums

A_id	A_name	A_releasedate	A_NoOfSongs	S_id
33	pop	2feb	23	99

A_id 33 A_name pop A_releasedate 2feb A_NoOfSongs 23 S_id 99