Gaming Zone SQL + Python Application

# Tasks Overview

1. Add a new game to the Games table with user input (game name, type, charge).

2. Register a new member with membership type and assign initial hours from Memberships table.

3. Update member hours: when a member plays a game, subtract hours and log gameplay.

4. Delete a member who hasn't played any game (no entry in GamePlays).

5. Write a Python function to display all games with charges above ₹100/hr.

6. Create a Python program to count how many games of each type (Sports, Racing, etc.) exist.

7. Show a list of all members with remaining hours less than 10.

8. List members who have played more than 2 different games.

9. For each membership type, calculate total hours remaining across all members.

10. Find and print total income earned from game plays (based on hours\_played × charge\_per\_hour).

11. Create a Python function to find and print the most active member (max hours\_spent).

12. Display the top 3 most played games based on hours played.

13. Create a report showing total hours played per member per game.

14. Write a Python program to:  
 - Take a member's name and game name as input.  
 - Log gameplay (update GamePlays, deduct hours).  
 - Prevent logging if hours left < required.

15. List all members who have used more than 75% of their allowed hours.

16. Show a detailed report of each member with:  
 - Name, membership type, total games played, total hours played, hours left.

17. Identify members who have never played any game (use LEFT JOIN).

18. Create a Python menu system with options to:  
 - Add Game  
 - Register Member  
 - Log Gameplay  
 - View Reports (1–17)

19. Modify the Python script to gracefully handle MySQL connection failure and display a user-friendly message.

20. Build a Python module (gaming\_utils.py) to separate:  
 - DB connection function  
 - Game operations  
 - Member operations  
 - Report functions

# SQL Schema & Sample Data

CREATE TABLE Games (  
 game\_id INT AUTO\_INCREMENT PRIMARY KEY,  
 game\_name VARCHAR(100),  
 game\_type VARCHAR(50),  
 charge\_per\_hour DECIMAL(5,2)  
);

CREATE TABLE Memberships (  
 membership\_type ENUM('Yearly', 'Monthly', 'Daily') PRIMARY KEY,  
 default\_hours INT  
);

CREATE TABLE Members (  
 member\_id INT AUTO\_INCREMENT PRIMARY KEY,  
 name VARCHAR(100),  
 membership\_type ENUM('Yearly', 'Monthly', 'Daily'),  
 total\_hours INT,  
 hours\_left INT  
);

CREATE TABLE GamePlay (  
 play\_id INT AUTO\_INCREMENT PRIMARY KEY,  
 member\_id INT,  
 game\_id INT,  
 hours\_played INT,  
 FOREIGN KEY (member\_id) REFERENCES Members(member\_id),  
 FOREIGN KEY (game\_id) REFERENCES Games(game\_id)  
);

# Python Code Implementation

## Database Connection

def connect\_db():  
 try:  
 return mysql.connector.connect(  
 host="localhost",  
 user="root",  
 password="Anas@2004",  
 database="GamingZone"  
 )  
 except mysql.connector.Error as err:  
 print("Database connection failed:", err)  
 return None

## Add Game

def add\_game(name, game\_type, charge):  
 conn = connect\_db()  
 cursor = conn.cursor()  
 cursor.execute("ALTER TABLE Games ADD IF NOT EXISTS game\_type VARCHAR(50)")  
 cursor.execute("INSERT INTO Games (game\_name, game\_type, charge\_per\_hour) VALUES (%s, %s, %s)",  
 (name, game\_type, charge))  
 conn.commit()  
 conn.close()

## Register Member

def register\_member(name, membership\_type):  
 conn = connect\_db()  
 cursor = conn.cursor()  
 cursor.execute("SELECT default\_hours FROM Memberships WHERE membership\_type = %s", (membership\_type,))  
 default\_hours = cursor.fetchone()  
 if default\_hours:  
 cursor.execute("INSERT INTO Members (name, membership\_type, total\_hours, hours\_left) VALUES (%s, %s, %s, %s)",  
 (name, membership\_type, default\_hours[0], default\_hours[0]))  
 conn.commit()  
 conn.close()

## Log Gameplay

def log\_gameplay(member\_id, game\_id, hours\_played):  
 conn = connect\_db()  
 cursor = conn.cursor()  
 cursor.execute("SELECT hours\_left FROM Members WHERE member\_id = %s", (member\_id,))  
 hours\_left = cursor.fetchone()[0]  
 if hours\_left >= hours\_played:  
 cursor.execute("INSERT INTO GamePlay (member\_id, game\_id, hours\_played) VALUES (%s, %s, %s)",  
 (member\_id, game\_id, hours\_played))  
 cursor.execute("UPDATE Members SET hours\_left = hours\_left - %s WHERE member\_id = %s",  
 (hours\_played, member\_id))  
 conn.commit()  
 else:  
 print("Not enough hours left.")  
 conn.close()

## Delete Inactive Members

def delete\_inactive\_members():  
 conn = connect\_db()  
 cursor = conn.cursor()  
 cursor.execute("DELETE FROM Members WHERE member\_id NOT IN (SELECT DISTINCT member\_id FROM GamePlay)")  
 conn.commit()  
 conn.close()

## Games Above ₹100

def games\_above\_100():  
 conn = connect\_db()  
 cursor = conn.cursor()  
 cursor.execute("SELECT game\_name, charge\_per\_hour FROM Games WHERE charge\_per\_hour > 100")  
 print(cursor.fetchall())  
 conn.close()

## Game Type Count

def game\_type\_count():  
 conn = connect\_db()  
 cursor = conn.cursor()  
 cursor.execute("SELECT game\_type, COUNT(\*) FROM Games GROUP BY game\_type")  
 print(cursor.fetchall())  
 conn.close()

## Low Hour Members

def low\_hour\_members():  
 conn = connect\_db()  
 cursor = conn.cursor()  
 cursor.execute("SELECT name FROM Members WHERE hours\_left < 10")  
 print(cursor.fetchall())  
 conn.close()

## Frequent Players

def frequent\_players():  
 conn = connect\_db()  
 cursor = conn.cursor()  
 cursor.execute("SELECT member\_id, COUNT(DISTINCT game\_id) FROM GamePlay GROUP BY member\_id HAVING COUNT(DISTINCT game\_id) > 2")  
 print(cursor.fetchall())  
 conn.close()

## Total Hours by Membership

def total\_hours\_by\_membership():  
 conn = connect\_db()  
 cursor = conn.cursor()  
 cursor.execute("SELECT membership\_type, SUM(hours\_left) FROM Members GROUP BY membership\_type")  
 print(cursor.fetchall())  
 conn.close()

## Total Income

def total\_income():  
 conn = connect\_db()  
 cursor = conn.cursor()  
 cursor.execute("SELECT SUM(gp.hours\_played \* g.charge\_per\_hour) FROM GamePlay gp JOIN Games g ON gp.game\_id = g.game\_id")  
 print("Total income: ₹", cursor.fetchone()[0])  
 conn.close()

## Most Active Member

def most\_active\_member():  
 conn = connect\_db()  
 cursor = conn.cursor()  
 cursor.execute("SELECT m.name, SUM(gp.hours\_played) FROM GamePlay gp JOIN Members m ON gp.member\_id = m.member\_id GROUP BY gp.member\_id ORDER BY SUM(gp.hours\_played) DESC LIMIT 1")  
 print(cursor.fetchone())  
 conn.close()

## Top 3 Games

def top\_3\_games():  
 conn = connect\_db()  
 cursor = conn.cursor()  
 cursor.execute("SELECT g.game\_name, SUM(gp.hours\_played) FROM GamePlay gp JOIN Games g ON gp.game\_id = g.game\_id GROUP BY gp.game\_id ORDER BY SUM(gp.hours\_played) DESC LIMIT 3")  
 print(cursor.fetchall())  
 conn.close()

## Member Game Hours

def member\_game\_hours():  
 conn = connect\_db()  
 cursor = conn.cursor()  
 cursor.execute("SELECT m.name, g.game\_name, SUM(gp.hours\_played) FROM GamePlay gp JOIN Members m ON gp.member\_id = m.member\_id JOIN Games g ON gp.game\_id = g.game\_id GROUP BY m.name, g.game\_name")  
 print(cursor.fetchall())  
 conn.close()

## Smart Log Play

def smart\_log\_play(member\_name, game\_name, hours):  
 conn = connect\_db()  
 cursor = conn.cursor()  
 cursor.execute("SELECT member\_id, hours\_left FROM Members WHERE name = %s", (member\_name,))  
 member = cursor.fetchone()  
 if not member:  
 print("Member not found.")  
 return  
 member\_id, hrs\_left = member  
 if hrs\_left < hours:  
 print("Not enough hours.")  
 return  
 cursor.execute("SELECT game\_id FROM Games WHERE game\_name = %s", (game\_name,))  
 game = cursor.fetchone()  
 if not game:  
 print("Game not found.")  
 return  
 game\_id = game[0]  
 cursor.execute("INSERT INTO GamePlay (member\_id, game\_id, hours\_played) VALUES (%s,%s,%s)", (member\_id, game\_id, hours))  
 cursor.execute("UPDATE Members SET hours\_left = hours\_left - %s WHERE member\_id = %s", (hours, member\_id))  
 conn.commit()  
 conn.close()

## Heavy Users (>75% hours used)

def heavy\_users():  
 conn = connect\_db()  
 cursor = conn.cursor()  
 cursor.execute("SELECT name FROM Members WHERE hours\_left < total\_hours \* 0.25")  
 print(cursor.fetchall())  
 conn.close()

## Member Usage Report

def member\_usage\_report():  
 conn = connect\_db()  
 cursor = conn.cursor()  
 cursor.execute("SELECT m.name, m.membership\_type, COUNT(DISTINCT gp.game\_id), SUM(gp.hours\_played), m.hours\_left FROM Members m LEFT JOIN GamePlay gp ON m.member\_id = gp.member\_id GROUP BY m.member\_id")  
 print(cursor.fetchall())  
 conn.close()

## Never Played Members

def never\_played\_members():  
 conn = connect\_db()  
 cursor = conn.cursor()  
 cursor.execute("SELECT m.name FROM Members m LEFT JOIN GamePlay gp ON m.member\_id = gp.member\_id WHERE gp.member\_id IS NULL")  
 print(cursor.fetchall())  
 conn.close()

## Menu System

def menu():  
 while True:  
 print("\n1. Add Game\n2. Register Member\n3. Log Gameplay\n4. View Reports\n5. Exit")  
 choice = input("Choice: ")  
 if choice == '1':  
 add\_game(input("Game: "), input("Type: "), float(input("Charge: ")))  
 elif choice == '2':  
 register\_member(input("Name: "), input("Type (Yearly/Monthly/Daily): "))  
 elif choice == '3':  
 smart\_log\_play(input("Member Name: "), input("Game Name: "), int(input("Hours: ")))  
 elif choice == '4':  
 top\_3\_games()  
 member\_usage\_report()  
 elif choice == '5':  
 break  
 else:  
 print("Invalid.")