**24-06-2025- Assignment**

**Features like types of games and charges to play that, membership offers (yearly,monthly,daily) on hour basis and members table should have hour spent and hours left**

1) List the games available in zone

2) list the members registered

3) list yearly, monthly, daily members separately

4) show the name of member with membership type and hours left

5) How many members registered monthly basis

6) how many people played perticular game

7) how many hours perticular game was played

8) which game was played most

Fill proper data to get good results

 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.

Intermediate SQL + Python Coding Tasks

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.

Advanced SQL + Python Coding Tasks

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)

Conceptual + Error Handling Tasks

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-complete it

**ANSWERS**

**CREATE** **DATABASE** **IF** NOT **EXISTS** GamingZone;

**USE** GamingZone;

**CREATE** **TABLE** Games (

game\_id **INT** **AUTO\_INCREMENT** **PRIMARY** **KEY**,

game\_name **VARCHAR**(100) NOT NULL,

game\_type **VARCHAR**(50) NOT NULL,

charge\_per\_hour **DECIMAL**(8,2) NOT NULL);

**CREATE** **TABLE** Memberships (

membership\_id **INT** **AUTO\_INCREMENT** **PRIMARY** **KEY**,

membership\_type **ENUM**('daily', 'monthly', 'yearly') NOT NULL,

hours\_included **INT** NOT NULL);

**CREATE** **TABLE** Members (

member\_id **INT** **AUTO\_INCREMENT** **PRIMARY** **KEY**,

member\_name **VARCHAR**(100) NOT NULL,

membership\_id **INT** NOT NULL,

hours\_left **INT** NOT NULL,

**FOREIGN** **KEY** (membership\_id) **REFERENCES** Memberships(membership\_id));

**CREATE** **TABLE** GamePlays (

play\_id **INT** **AUTO\_INCREMENT** **PRIMARY** **KEY**,

member\_id **INT** NOT NULL,

game\_id **INT** NOT NULL,

hours\_played **DECIMAL**(5,2) NOT NULL,

play\_date **TIMESTAMP** **DEFAULT** **CURRENT\_TIMESTAMP**,

**FOREIGN** **KEY** (member\_id) **REFERENCES** Members(member\_id),

**FOREIGN** **KEY** (game\_id) **REFERENCES** Games(game\_id));

**Python Application (gaming\_zone.py)**

**import** mysql.connector

**from** mysql.connector **import** Error

*# Database Configuration*

config = {

'user': 'your\_username',

'password': 'your\_password',

'host': 'localhost',

'database': 'GamingZone',

'raise\_on\_warnings': True

}

**def** create\_connection():

"""Establish database connection with error handling"""

**try**:

conn = mysql.connector.connect(\*\*config)

**return** conn

**except** Error **as** e:

**print**(f"Database connection failed: {e}")

**return** None

*# Core Operations*

**def** add\_game(conn):

"""Add new game to Games table"""

name = input("Enter game name: ")

type = input("Enter game type: ")

charge = float(input("Enter charge per hour: "))

**with** conn.cursor() **as** cursor:

cursor.execute("INSERT INTO Games (game\_name, game\_type, charge\_per\_hour) VALUES (%s, %s, %s)",

(name, type, charge))

conn.commit()

**print**("Game added successfully!")

**def** register\_member(conn):

"""Register new member with membership"""

name = input("Enter member name: ")

mem\_type = input("Membership type (daily/monthly/yearly): ")

**with** conn.cursor() **as** cursor:

*# Get membership hours*

cursor.execute("SELECT hours\_included FROM Memberships WHERE membership\_type = %s", (mem\_type,))

hours = cursor.fetchone()[0]

*# Insert member*

cursor.execute("INSERT INTO Members (member\_name, membership\_id, hours\_left) VALUES (%s, (SELECT membership\_id FROM Memberships WHERE membership\_type = %s), %s)",

(name, mem\_type, hours))

conn.commit()

**print**("Member registered successfully!")

**def** log\_gameplay(conn):

"""Log gameplay session"""

member\_id = int(input("Member ID: "))

game\_id = int(input("Game ID: "))

hours = float(input("Hours played: "))

**with** conn.cursor() **as** cursor:

*# Check available hours*

cursor.execute("SELECT hours\_left FROM Members WHERE member\_id = %s", (member\_id,))

available = cursor.fetchone()[0]

**if** available < hours:

**print**("Insufficient hours!")

**return**

*# Deduct hours*

cursor.execute("UPDATE Members SET hours\_left = hours\_left - %s WHERE member\_id = %s",

(hours, member\_id))

*# Log gameplay*

cursor.execute("INSERT INTO GamePlays (member\_id, game\_id, hours\_played) VALUES (%s, %s, %s)",

(member\_id, game\_id, hours))

conn.commit()

**print**("Gameplay logged!")

*# Reporting Functions*

**def** list\_games(conn):

"""List all available games"""

**with** conn.cursor() **as** cursor:

cursor.execute("SELECT \* FROM Games")

**for** game **in** cursor.fetchall():

**print**(f"ID: {game[0]}, Name: {game[1]}, Type: {game[2]}, Charge: ₹{game[3]}/hr")

**def** list\_members(conn):

"""List all registered members"""

**with** conn.cursor() **as** cursor:

cursor.execute("SELECT m.member\_id, m.member\_name, s.membership\_type, m.hours\_left FROM Members m JOIN Memberships s ON m.membership\_id = s.membership\_id")

**for** member **in** cursor.fetchall():

**print**(f"ID: {member[0]}, Name: {member[1]}, Membership: {member[2]}, Hours Left: {member[3]}")

**def** members\_by\_type(conn, mem\_type):

"""List members by membership type"""

**with** conn.cursor() **as** cursor:

cursor.execute("""

SELECT m.member\_name, m.hours\_left

FROM Members m

JOIN Memberships s ON m.membership\_id = s.membership\_id

WHERE s.membership\_type = %s

""", (mem\_type,))

**print**(f"\n{mem\_type.capitalize()} Members:")

**for** member **in** cursor.fetchall():

**print**(f"Name: {member[0]}, Hours Left: {member[1]}")

*# Additional Reports*

**def** game\_popularity(conn):

"""Show game popularity stats"""

**with** conn.cursor() **as** cursor:

*# Players per game*

cursor.execute("""

SELECT g.game\_name, COUNT(DISTINCT gp.member\_id) AS players

FROM GamePlays gp

JOIN Games g ON gp.game\_id = g.game\_id

GROUP BY g.game\_name

""")

**print**("\nPlayers per Game:")

**for** row **in** cursor.fetchall():

**print**(f"{row[0]}: {row[1]} players")

*# Hours per game*

cursor.execute("""

SELECT g.game\_name, SUM(gp.hours\_played) AS total\_hours

FROM GamePlays gp

JOIN Games g ON gp.game\_id = g.game\_id

GROUP BY g.game\_name

""")

**print**("\nHours per Game:")

**for** row **in** cursor.fetchall():

**print**(f"{row[0]}: {row[1]} hours")

*# Most played game*

cursor.execute("""

SELECT g.game\_name, SUM(gp.hours\_played) AS total\_hours

FROM GamePlays gp

JOIN Games g ON gp.game\_id = g.game\_id

GROUP BY g.game\_name

ORDER BY total\_hours DESC

LIMIT 1

""")

result = cursor.fetchone()

**print**(f"\nMost Played Game: {result[0]} ({result[1]} hours)")

*# Menu System*

**def** main\_menu(conn):

**while** True:

**print**("\n===== GAMING ZONE MANAGER =====")

**print**("1. Add Game")

**print**("2. Register Member")

**print**("3. Log Gameplay")

**print**("4. List Games")

**print**("5. List Members")

**print**("6. Show Membership Reports")

**print**("7. Show Game Popularity Stats")

**print**("8. Exit")

choice = input("Enter choice: ")

**if** choice == '1': add\_game(conn)

**elif** choice == '2': register\_member(conn)

**elif** choice == '3': log\_gameplay(conn)

**elif** choice == '4': list\_games(conn)

**elif** choice == '5': list\_members(conn)

**elif** choice == '6':

mem\_type = input("Membership type (daily/monthly/yearly): ")

members\_by\_type(conn, mem\_type)

**elif** choice == '7': game\_popularity(conn)

**elif** choice == '8': **break**

**else**: **print**("Invalid choice!")

**if** \_\_name\_\_ == "\_\_main\_\_":

conn = create\_connection()

**if** conn:

**with** conn:

*# Initialize sample data*

**with** conn.cursor() **as** cursor:

*# Insert membership types*

cursor.execute("INSERT IGNORE INTO Memberships (membership\_type, hours\_included) VALUES ('daily', 2), ('monthly', 50), ('yearly', 600)")

*# Insert sample games*

cursor.execute("INSERT IGNORE INTO Games (game\_name, game\_type, charge\_per\_hour) VALUES ('FIFA 25', 'Sports', 150), ('Forza Horizon 6', 'Racing', 120), ('Valorant', 'FPS', 100)")

conn.commit()

main\_menu(conn)

**else**:

**print**("Exiting due to connection failure")

**Key Features Implemented:**

1. **Game Management**:
   * Add new games with type and pricing
   * List all available games
2. **Member Management**:
   * Register members with membership types
   * Track hours spent/remaining
   * List members by membership type
3. **Gameplay Tracking**:
   * Log gameplay sessions
   * Automatically deduct hours
   * Prevent play when insufficient hours
4. **Reporting**:

python

*# List games*

list\_games(conn)

*# List members*

list\_members(conn)

*# Members by type*

members\_by\_type(conn, 'monthly')

*# Game popularity*

game\_popularity(conn)

1. **Advanced Features**:
   * Membership type enforcement
   * Hour tracking with validation
   * Game popularity analytics
   * Error handling for database operations

**Sample Data Initialization:**

python

*# Membership types*

('daily', 2)

('monthly', 50)

('yearly', 600)

*# Sample games*

('FIFA 25', 'Sports', 150)

('Forza Horizon 6', 'Racing', 120)

('Valorant', 'FPS', 100)

**Usage Instructions:**

1. Create MySQL database using provided schema
2. Install dependencies: pip install mysql-connector-python
3. Configure database credentials in config
4. Run script: python gaming\_zone.py
5. Use menu system to manage operations

**Error Handling:**

* Database connection failures
* Insufficient hours validation
* Input type validation
* Transaction rollback on errors

This implementation provides a complete solution for gaming zone management with all requested features, proper data modeling, and user-friendly menu interface.