```
import mysql.connector
from mysql.connector import Error
import random
from flask import Flask, render_template, request, redirect, url_for, flash, make_response
import csv
import io

app = Flask(__name__)
app.secret_key = 'very_imp'


DB_CONFIG = {
    'host': 'localhost',
    'database': 'chess_tournament',
    'user': 'root',
    'password': 'root'
}

def get_db_connection():
    try:
        conn = mysql.connector.connect(**DB_CONFIG)
        return conn
    except Error as e:
        print(f"Error connecting to MySQL: {e}")
        return None

def calculate_points(tournament_id):
    conn = get_db_connection()
    if conn:
        cursor = conn.cursor(dictionary=True)
        query = """
        SELECT p.id, p.name, p.rating,
        COALESCE(SUM(CASE
            WHEN m.player1_id = p.id THEN m.result
            WHEN m.player2_id = p.id THEN 1 - m.result
            END), 0) AS points
        FROM players p
        LEFT JOIN matches m ON (m.player1_id = p.id OR m.player2_id = p.id) AND m.result
IS NOT NULL
        WHERE p.tournament_id = %s
        GROUP BY p.id
        ORDER BY points DESC, p.rating DESC
        """
        cursor.execute(query, (tournament_id,))
        points = cursor.fetchall()
        cursor.close()
        conn.close()
        return points
    return []

def all_results_entered(tournament_id, round_num):
    conn = get_db_connection()
    if conn:
        cursor = conn.cursor(dictionary=True)
        cursor.execute("""
            SELECT COUNT(*) AS pending
            FROM matches
            WHERE tournament_id = %s AND round_num = %s AND result IS NULL AND player2_id
IS NOT NULL
        """, (tournament_id, round_num))
        pending = cursor.fetchone()['pending']
        cursor.close()
        conn.close()
        return pending == 0
    return False

def generate_pairings(tournament_id, round_num):
    conn = get_db_connection()
    if not conn:
        return False

    cursor = conn.cursor(dictionary=True)

    cursor.execute("SELECT COUNT(*) AS count FROM matches WHERE tournament_id = %s AND
round_num = %s", (tournament_id, round_num))
    if cursor.fetchone()['count'] > 0:
        conn.close()
        return False

    players = calculate_points(tournament_id)
    if round_num == 1:
        players.sort(key=lambda x: x['rating'], reverse=True)

    sorted_players = players

    n = len(sorted_players)
    if n < 2:
        conn.close()
        return False

    bye_player = None
    if n % 2 == 1:
        bye_player = sorted_players.pop()
        n -= 1
        cursor.execute("""
            INSERT INTO matches (tournament_id, round_num, player1_id, player2_id, result)
            VALUES (%s, %s, %s, NULL, 1.0)
        """, (tournament_id, round_num, bye_player['id']))

    if round_num == 1:
        num_groups = 2
    else:
        num_groups = 2 ** (round_num - 1)

    if num_groups > n:
        num_groups = n

    group_size = n // num_groups
    remainder = n % num_groups

    groups = []
    start = 0
    for i in range(num_groups):
        size = group_size + 1 if i < remainder else group_size
        groups.append(sorted_players[start:start + size])
        start += size

    for i in range(0, num_groups, 2):
        if i + 1 >= num_groups:
            break
        group_a = groups[i]
        group_b = groups[i + 1]


        random.shuffle(group_b)


        for a, b in zip(group_a, group_b):
            p1_id = min(a['id'], b['id'])
            p2_id = max(a['id'], b['id'])
            cursor.execute("""
                INSERT INTO matches (tournament_id, round_num, player1_id, player2_id,
result)
                VALUES (%s, %s, %s, %s, NULL)
            """, (tournament_id, round_num, p1_id, p2_id))

    conn.commit()
    cursor.close()
    conn.close()
    return True

@app.route('/')
def index():
    conn = get_db_connection()
    if conn:
        cursor = conn.cursor(dictionary=True)
        cursor.execute("SELECT * FROM tournaments")
        tournaments = cursor.fetchall()
        cursor.close()
        conn.close()
        return render_template('index.html', tournaments=tournaments)
    flash('Database connection error')
    return render_template('index.html', tournaments=[])

@app.route('/create_tournament', methods=['GET', 'POST'])
def create_tournament():
    if request.method == 'POST':
        name = request.form.get('name', '').strip()
        num_rounds_str = request.form.get('num_rounds', '')

        if not name:
            flash('Tournament name is required!')
            return render_template('create_tournament.html')

        try:
            num_rounds = int(num_rounds_str)
            if num_rounds <= 0:
                raise ValueError
        except ValueError:
            flash('Number of rounds must be a positive integer!')
            return render_template('create_tournament.html')

        conn = get_db_connection()
        if conn:
            try:
                cursor = conn.cursor()
                cursor.execute("INSERT INTO tournaments (name, num_rounds) VALUES (%s,
%s)", (name, num_rounds))
                conn.commit()
                tournament_id = cursor.lastrowid
                cursor.close()
                conn.close()
                flash('Tournament created successfully!')
                return redirect(url_for('tournament', id=tournament_id))
            except Error as e:
                flash(f'Error creating tournament: {e}')
        else:
            flash('Database connection error')
    return render_template('create_tournament.html')

@app.route('/tournament/<int:id>', methods=['GET', 'POST'])
def tournament(id):
    conn = get_db_connection()
    if not conn:
        flash('Database connection error')
        return redirect(url_for('index'))

    cursor = conn.cursor(dictionary=True)
    cursor.execute("SELECT * FROM tournaments WHERE id = %s", (id,))
    tour = cursor.fetchone()
    if not tour:
        flash('Tournament not found')
        conn.close()
        return redirect(url_for('index'))

    if request.method == 'POST':
        if 'add_player' in request.form:
            name = request.form.get('name', '').strip()
            rating_str = request.form.get('rating', '')
            if not name:
                flash('Player name is required!')
            else:
                try:
                    rating = int(rating_str)
                    if rating < 0:
                        raise ValueError
                    cursor.execute("INSERT INTO players (tournament_id, name, rating)
VALUES (%s, %s, %s)", (id, name, rating))
                    conn.commit()
                    flash('Player added!')
                except ValueError:
                    flash('Rating must be a non-negative integer!')
                except Error as e:
                    flash(f'Error adding player: {e}')
        elif 'csv_file' in request.files:
            file = request.files['csv_file']
            if file and file.filename.endswith('.csv'):
                try:
                    csv_data = io.StringIO(file.stream.read().decode('utf-8'))
                    reader = csv.reader(csv_data)
                    header = next(reader, None)  # Skip header if present
                    added = 0
                    for row in reader:
                        if len(row) >= 2:
                            name = row[0].strip()
                            rating_str = row[1].strip()
                            if not name:
                                continue
                            try:
                                rating = int(rating_str)
                                if rating < 0:
                                    raise ValueError
                                cursor.execute("INSERT INTO players (tournament_id, name, 
rating) VALUES (%s, %s, %s)", (id, name, rating))
                                added += 1
                            except ValueError:
                                pass
                    if added > 0:
                        conn.commit()
                        flash(f'{added} players imported from CSV!')
                    else:
                        flash('No valid players found in CSV!')
                except Exception as e:
                    flash(f'Error importing CSV: {e}')
            else:
                flash('Invalid file or no file selected!')
    cursor.execute("SELECT * FROM players WHERE tournament_id = %s", (id,))
    players = cursor.fetchall()

    standings = calculate_points(id)

    matches = {}
    for r in range(1, tour['current_round'] + 1):
        cursor.execute("""
            SELECT m.*, p1.name AS p1_name, p2.name AS p2_name
            FROM matches m
            LEFT JOIN players p1 ON m.player1_id = p1.id
            LEFT JOIN players p2 ON m.player2_id = p2.id
            WHERE m.tournament_id = %s AND m.round_num = %s
        """, (id, r))
        matches[r] = cursor.fetchall()

    cursor.close()
    conn.close()
    return render_template('tournament.html', tour=tour, players=players, 
standings=standings, matches=matches)

@app.route('/delete_tournament/<int:id>', methods=['POST'])
def delete_tournament(id):
    conn = get_db_connection()
    if conn:
        cursor = conn.cursor()
        cursor.execute("DELETE FROM tournaments WHERE id = %s", (id,))
        conn.commit()
        cursor.close()
        conn.close()
        flash('Tournament deleted successfully!')
    else:
        flash('Database connection error')
    return redirect(url_for('index'))

@app.route('/generate_pairings/<int:id>')
def generate_pairings_route(id):
    conn = get_db_connection()
    if conn:
        cursor = conn.cursor(dictionary=True)
        cursor.execute("SELECT current_round, num_rounds FROM tournaments WHERE id = %s", 
(id,))
        tour = cursor.fetchone()
        next_round = tour['current_round'] + 1
        if next_round > tour['num_rounds']:
            flash('All rounds completed!')
        else:
            if tour['current_round'] > 0 and not all_results_entered(id, 
tour['current_round']):
                flash('Cannot generate next round pairings until all results for the 
current round are entered!')
            elif generate_pairings(id, next_round):
                cursor.execute("UPDATE tournaments SET current_round = %s WHERE id = %s", 
(next_round, id))
                conn.commit()
                flash(f'Pairings generated for round {next_round}!')
            else:
                flash('Pairings already generated, not enough players, or error.')
        cursor.close()
        conn.close()
    else:
        flash('Database connection error')
    return redirect(url_for('tournament', id=id))

@app.route('/input_results/<int:id>/<int:round_num>', methods=['GET', 'POST'])
def input_results(id, round_num):
    conn = get_db_connection()
    if not conn:
        flash('Database connection error')
        return redirect(url_for('tournament', id=id))

    cursor = conn.cursor(dictionary=True)
    cursor.execute("""
        SELECT m.*, p1.name AS p1_name, p2.name AS p2_name
        FROM matches m
        LEFT JOIN players p1 ON m.player1_id = p1.id
        LEFT JOIN players p2 ON m.player2_id = p2.id
        WHERE m.tournament_id = %s AND m.round_num = %s
    """, (id, round_num))
    matches = cursor.fetchall()

    if request.method == 'POST':
        updated = False
        for match in matches:
            if match['player2_id'] is None:
                continue  # Bye, already set
            result_key = f"result_{match['id']}"
            if result_key in request.form:
                result_str = request.form[result_key]
                if result_str == '':
                    continue
                try:
                    result = float(result_str)
                    if result not in [0.0, 0.5, 1.0]:
                        raise ValueError
                    cursor.execute("UPDATE matches SET result = %s WHERE id = %s", 
(result, match['id']))
                    updated = True
                except ValueError:
                    flash(f'Invalid result for match {match["id"]}: must be 0, 0.5, or 
1!')
        if updated:
            conn.commit()
            flash('Results updated!')
        cursor.close()
        conn.close()
        return redirect(url_for('tournament', id=id))

    cursor.close()
    conn.close()
    return render_template('input_results.html', matches=matches, tour_id=id, 
round_num=round_num)

@app.route('/export_standings/<int:id>')
def export_standings(id):
    standings = calculate_points(id)
    output = io.StringIO()
    writer = csv.writer(output)
    writer.writerow(['Name', 'Rating', 'Points'])
    for player in standings:
        writer.writerow([player['name'], player['rating'], player['points']])
    response = make_response(output.getvalue())
    response.headers["Content-Disposition"] = f"attachment; filename=standings_{id}.csv"
    response.headers["Content-type"] = "text/csv"
    return response

if __name__ == '__main__':
    app.run(debug=True)
```