QUANTUM DICE ROLLER
Database Project Documentation

By

JAMES M. CANIABERAL

ARJONEL M. MENDOZA, MIT
Lecturer

PROJECT OVERVIEW

The Quantum Dice Roller is a Flask based web app that simulates the Quantum randomness scaled up into an interactive way and a user-friendly experience. The system offers user authentication, roll tracking and real-time statistical analysis to show how one, or how many small factors can accumulate and start affecting or possibly change the following outcomes. With results comparable to commercial quantum simulations, this project, built using Python, Flask and SQlite combines the computational randomness with database management while offering insights on having a scalable design and the science behind proper random number generation.

ENTITY-RELATIONSHIP DIAGRAM (ERD)

The ERD outlines the core entities in the Quantum Dice Roller System, including Users, Dice Rolls, and Roll Statistics. Each entity is defined by attributes essential for tracking user activity, managing dice roll results, and storing roll statistics for individual users.

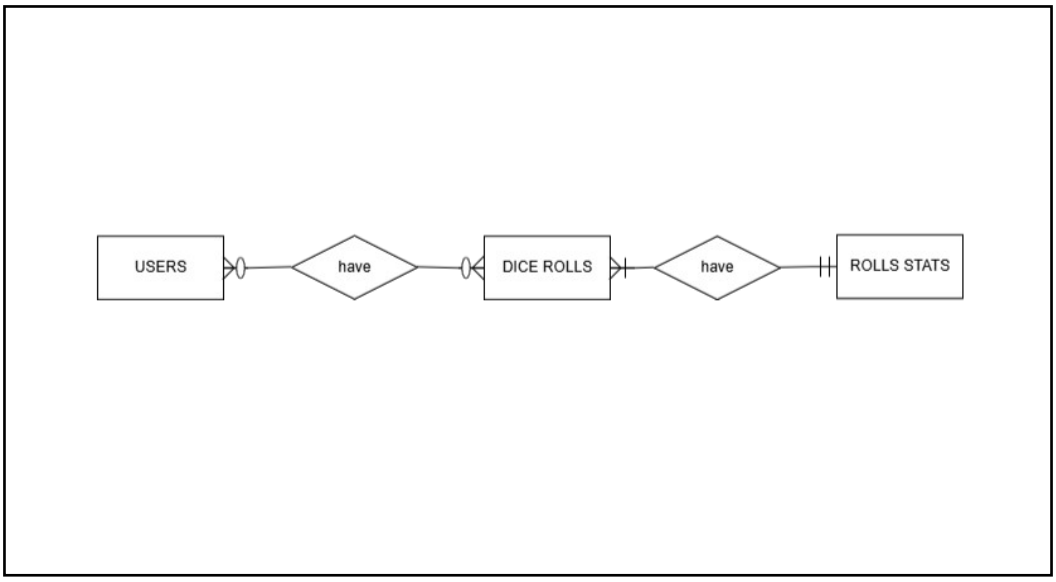

Figure 1. Entity Relationship Diagram

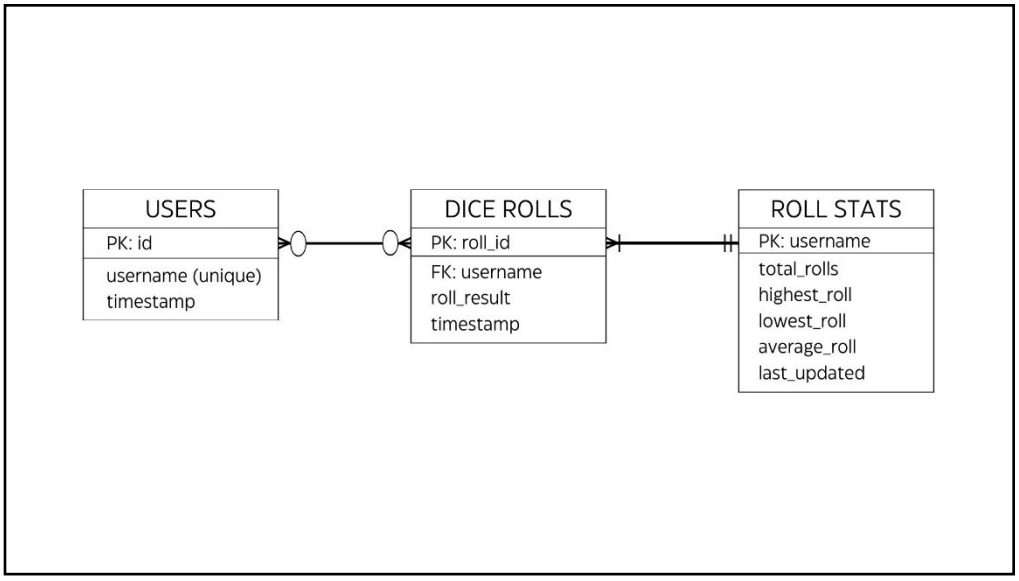

Figure 2. Entity Relationship Diagram

Entities and their Relationships

1. Users and Dice Rolls

- **Type:** One-to-Many (1:N)
- **Scientific Basis:** Each dice roll is an independent event associated with a unique user. This aligns with the principle of independent random events in probability theory, where prior rolls do not influence future outcomes.

2. Users and Roll Statistics

- **Type:** One-to-One (1:1)
- **Scientific Basis:** Statistical summaries provide a snapshot of a user's historical dice rolls, reflecting broader concepts in data aggregation and analysis, such as mean and extrema calculations.

SQL SCRIPTS

```
CREATE TABLE IF NOT EXISTS users (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    username TEXT UNIQUE NOT NULL,
    password TEXT NOT NULL
);
```

```
CREATE TABLE IF NOT EXISTS dice_rolls (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    username TEXT NOT NULL,
    roll_result INTEGER NOT NULL,
    timestamp DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (username) REFERENCES users (username)
);

CREATE TABLE IF NOT EXISTS roll_statistics (
    username TEXT PRIMARY KEY,
    total_rolls INTEGER DEFAULT 0,
    highest_roll INTEGER DEFAULT NULL,
    lowest_roll INTEGER DEFAULT NULL,
    average_roll REAL DEFAULT NULL,
    last_updated DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (username) REFERENCES users (username) ON DELETE CASCADE
);
```

Here are the QUERIES used in the accessing of the database:

Insert New User:

- INSERT INTO users (username, password) VALUES (?, ?)

Check if Username Exists:

- SELECT * FROM users WHERE username = ?

Insert New Dice Roll:

- INSERT INTO dice_rolls (username, roll_result) VALUES (?, ?)

Select All Rolls for User:

- SELECT roll_result FROM dice_rolls WHERE username = ?;

Insert or Update Roll Statistics:

- INSERT INTO roll_statistics (

  username, total_rolls, highest_roll, lowest_roll, average_roll, last_updated

) VALUES (?, ?, ?, ?, ?, ?)

  ON CONFLICT(username) DO UPDATE SET

  total_rolls = EXCLUDED.total_rolls,

  highest_roll = EXCLUDED.highest_roll,

  lowest_roll = EXCLUDED.lowest_roll,

  average_roll = EXCLUDED.average_roll,

  last_updated = CURRENT_TIMESTAMP;

Select Roll Statistics for User:

- SELECT total_rolls, highest_roll, lowest_roll, average_roll, last_updated FROM roll_statistics WHERE username = ?;

Select Roll History (All Rolls with Timestamps):

- SELECT roll_result, timestamp

  FROM dice_rolls

  WHERE username = ?

  ORDER BY timestamp DESC;

  Select Roll History (All Rolls with Timestamps):

-- Complex Query: Users with Average Roll >= 4 and More than 5 Rolls:

- SELECT username, AVG(roll_result) AS average_roll, COUNT(*) AS total_rolls

  FROM dice_rolls

  WHERE roll_result >= 4

  GROUP BY username

  HAVING COUNT(*) > 5;

Delete a Specific Roll:

- DELETE FROM dice_rolls WHERE id = ? AND username = ?;

SAMPLE DATA

Registered Users Data:

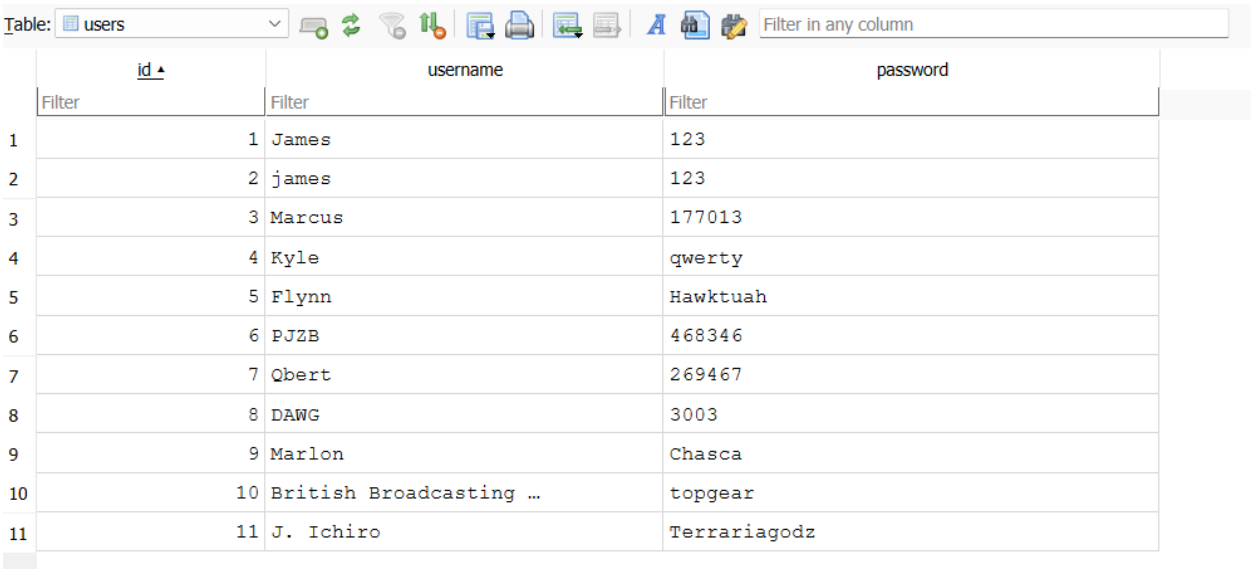
Table: users

Filter in any column

	id ▲	username	password
	Filter	Filter	Filter
1	1	James	123
2	2	james	123
3	3	Marcus	177013
4	4	Kyle	qwerty
5	5	Flynn	Hawktuah
6	6	PJZB	468346
7	7	Qbert	269467
8	8	DAWG	3003
9	9	Marlon	Chasca
10	10	British Broadcasting …	topgear
11	11	J. Ichiro	Terrariagodz

Roll Activity Data:

Table: dice_rolls

Filter in any column

	id	username	ll_resu	timestamp
	Filter	Filter	Filter	Filter
1	1	Marcus	6	2024-12-11 08:23:39
2	2	Marcus	3	2024-12-11 08:23:40
3	3	Marcus	5	2024-12-11 08:23:40
4	4	Marcus	6	2024-12-11 08:23:40
5	5	Marcus	6	2024-12-11 08:23:40
6	6	Marcus	1	2024-12-11 08:23:41
7	7	Marcus	6	2024-12-11 08:23:41
8	8	Marcus	5	2024-12-11 08:23:41
9	9	James	4	2024-12-11 08:23:57
10	10	James	6	2024-12-11 08:23:58
11	11	James	3	2024-12-11 08:23:58
12	12	James	6	2024-12-11 08:23:58
13	13	James	6	2024-12-11 08:23:58
14	14	James	3	2024-12-11 08:23:58
15	15	James	2	2024-12-11 08:23:58
16	16	James	1	2024-12-11 08:23:58
17	17	DAWG	1	2024-12-11 08:24:21
18	18	DAWG	4	2024-12-11 08:24:21
19	19	DAWG	3	2024-12-11 08:24:22
20	20	Marlon	4	2024-12-11 08:24:35
21	21	Marlon	4	2024-12-11 08:24:35
22	22	Marlon	5	2024-12-11 08:24:35
23	23	Marlon	3	2024-12-11 08:24:36
24	24	Marlon	2	2024-12-11 08:24:36
25	25	Marlon	4	2024-12-11 08:24:36
26	26	Marlon	5	2024-12-11 08:24:36
27	27	Marlon	6	2024-12-11 08:24:37
28	28	Marlon	4	2024-12-11 08:24:37
29	29	Marlon	2	2024-12-11 08:24:37

Roll Statistics Data:

Table: roll_statistics

Filter in any column

	username	total_rolls	highest_roll	lowest_roll	average_roll	last_updated
	Filter	Filter	Filter	Filter	Filter	Filter
1	Marcus	8	6	1	4.75	2024-12-11 08:23:41
2	James	8	6	1	3.875	2024-12-11 08:23:58
3	DAWG	3	4	1	2.66666666666667	2024-12-11 08:24:22
4	Marlon	11	6	1	3.63636363636364	2024-12-11 08:24:37

Functions used in accessing the Database:

Register/Login.py

```python
@app.route('/', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']

        conn = get_db_connection()
        cursor = conn.cursor()
        cursor.execute("SELECT * FROM users WHERE username = ?", (username,))
        existing_user = cursor.fetchone()

        if existing_user:
            flash("Username already exists. Please login instead.")
            return redirect(url_for('login'))

        cursor.execute("INSERT INTO users (username, password) VALUES (?, ?)", (username, password))
        conn.commit()
        cursor.close()
        conn.close()
        flash("Registration successful. You can now log in.")
        return redirect(url_for('login'))

    return render_template('register.html')

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']

        conn = get_db_connection()
        cursor = conn.cursor()
        cursor.execute("SELECT * FROM users WHERE username = ? AND password = ?", (username, password))
        user = cursor.fetchone()
        cursor.close()
        conn.close()

        if user:
            return redirect(url_for('roll', username=username))
        else:
            flash("Invalid username or password. Please try again.")
            return redirect(url_for('login'))

    return render_template('login.html')
```

Roll/History.py

```python
@app.route('/roll/<username>', methods=['GET', 'POST'])
def roll(username):
    roll_result = None
    if request.method == 'POST':
        try:
            roll_result = roll_die(1, 6)


            conn = get_db_connection()
            cursor = conn.cursor()
            cursor.execute(
                "INSERT INTO dice_rolls (username, roll_result) VALUES (?, ?)",
                (username, roll_result)
            )


            cursor.execute('''
                SELECT COUNT(*) AS total_rolls, MAX(roll_result) AS highest_roll,
                       MIN(roll_result) AS lowest_roll, AVG(roll_result) AS average_roll
                FROM dice_rolls
                WHERE username = ?
            ''', (username,))
            stats = cursor.fetchone()


            cursor.execute('''
                INSERT INTO roll_statistics (username, total_rolls, highest_roll, lowest_roll, average_roll, last_updated)
                VALUES (?, ?, ?, ?, ?, ?)
                ON CONFLICT(username) DO UPDATE SET
                    total_rolls = ?,
                    highest_roll = ?,
                    lowest_roll = ?,
                    average_roll = ?,
                    last_updated = CURRENT_TIMESTAMP
            ''', (username, stats['total_rolls'], stats['highest_roll'], stats['lowest_roll'], stats['average_roll'], datetime.now(),
                  stats['total_rolls'], stats['highest_roll'], stats['lowest_roll'], stats['average_roll']))

            conn.commit()
            cursor.close()
            conn.close()

        except Exception as e:
            print(f"Error: {e}")
            flash('Error generating dice roll, please try again!', 'error')

    return render_template('roll.html', roll_result=roll_result, username=username)

@app.route('/history/<username>')
def history(username):
    conn = get_db_connection()
    cursor = conn.cursor()
    cursor.execute("SELECT id, roll_result, timestamp FROM dice_rolls WHERE username = ? ORDER BY timestamp DESC LIMIT 20", (username,))
    rolls = cursor.fetchall()
    cursor.close()
    conn.close()
    return render_template('history.html', username=username, rolls=rolls)
```

Statstics.py

```python
@app.route('/statistics/<username>')
def statistics(username):
    conn = get_db_connection()
    cursor = conn.cursor()
    cursor.execute("SELECT total_rolls, highest_roll, lowest_roll, average_roll, last_updated FROM roll_statistics WHERE username = ?", (username,))
    stats = cursor.fetchone()
    cursor.close()
    conn.close()

    if stats:
        return render_template(
            'statistics.html',
            username=username,
            total_rolls=stats['total_rolls'],
            highest_roll=stats['highest_roll'],
            lowest_roll=stats['lowest_roll'],
            average_roll=round(stats['average_roll'], 2) if stats['average_roll'] else None,
            last_updated=stats['last_updated']
        )
    else:
        flash("No statistics available. Start rolling to generate stats!", "info")
        return redirect(url_for('roll', username=username))
```

Delete roll, Logout.py

```python
@app.route('/delete_roll/<username>/<int:roll_id>', methods=['POST'])
def delete_roll(username, roll_id):
    conn = get_db_connection()
    cursor = conn.cursor()
    cursor.execute("DELETE FROM dice_rolls WHERE id = ? AND username = ?", (roll_id, username))
    if cursor.rowcount == 0:
        flash("No such roll found or you are not authorized to delete it.", "error")
    else:
        flash(f"Roll with ID {roll_id} deleted successfully.")
    conn.commit()
    cursor.close()
    conn.close()
    return redirect(url_for('history', username=username))


@app.route('/logout')
def logout():
    flash("Logged out successfully.")
    return redirect(url_for('register'))
```