Patuakhali Science and Technology University
Faculty of Computer Science and Engineering

CCE 224 :: Database System Sessional

Sessional Project Report

Project Title : SQL Judge
Submission Date : Sat 14, June 2025

Submitted to,

Prof. Dr. Md Samsuzzaman
Professor,
Department of Computer and Communication Engineering,
Patuakhali Science and Technology University.

Submitted by,

Md. Sharafat Karim
ID : 2102024,
Reg: 10151

Contents

1. Introduction 3
2. Objective 3
3. Technology 3
  3.1. Database Characteristics 3
  3.2. Schema Diagram 4
  3.3. E-R Diagram 4
    3.3.1. Without attributes 4
    3.3.2. With all attributes 5
4. Database Implementation 5
  4.1. DDL 5
    4.1.1. Database Creation 5
    4.1.2. Table Creation 5
      4.1.2.1. Users Table 5
      4.1.2.2. Feedback Table 6
      4.1.2.3. Blogs and Comments 6
      4.1.2.4. Newsletters Table 7
      4.1.2.5. Contests, Problems 7
      4.1.2.6. Submissions and User Scores 8
    4.1.3. Triggers 8
    4.1.4. Views 10
  4.2. SQL Queries 10
    4.2.1. Authentication 10
    4.2.2. User Profile 10
    4.2.3. Blog 11
    4.2.4. Comment & React 11
    4.2.5. Leaderboard 12
5. Conclusion 13
6. References 13
  6.1. Documentations 13

SQL Judge

1. Introduction

An SQL learning platform that allows users to learn and practice SQL queries. It it provides a set of features including user registration, problem submission, and a leaderboard. And last but not least, it has built in Blog and chatsheet.

2. Objective

To develop a web-based system that provides intelligent recommendations for computer hardware components based on user selections, ensuring compatibility, performance optimization (via bottleneck analysis), and benchmarking. Additionally, the system will include an e-commerce portal for customers and an admin dashboard for managing products and data.

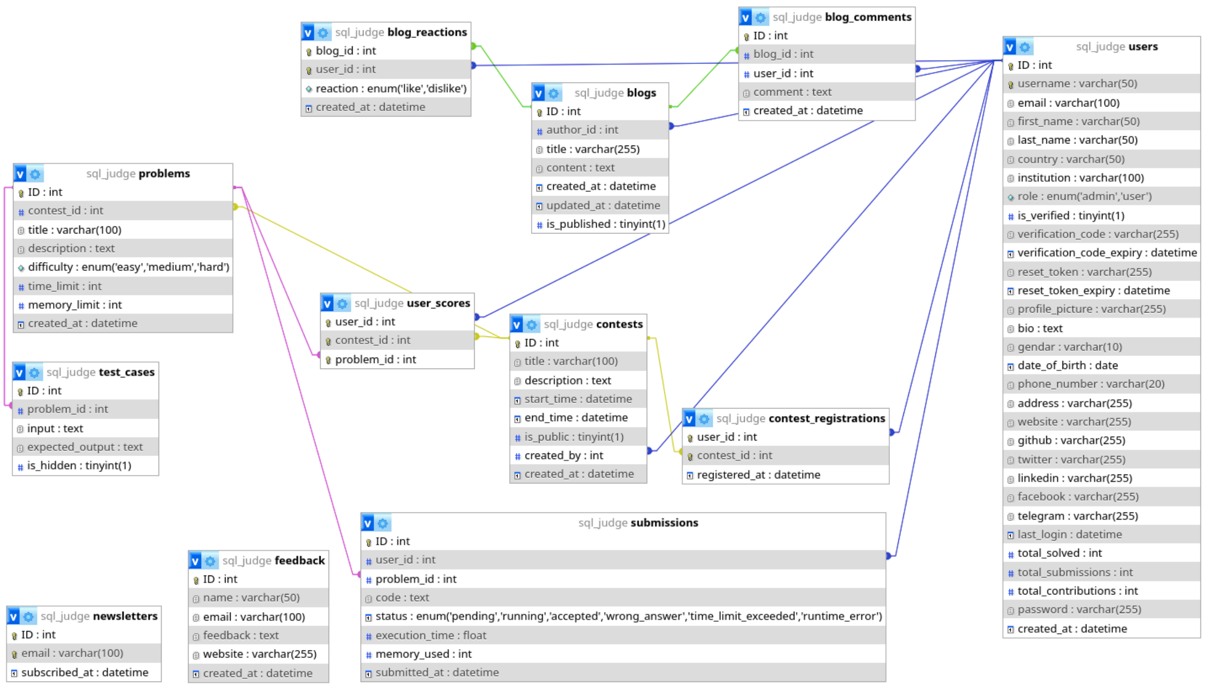
3. Technology

Layer	**Technology**
Frontend	HTML, CSS & JavaScript
Backend	PHP
Database	MySQL
Authentication	Session storage
Hosting	Localhost, infinityfree
Version control	Git
CI/ CD	GitHub

3.1. Database Characteristics

- **CRUD Operations** = Create, Read, Update, Delete
- **Data Integrity** is Enforced through foreign keys and constraints
- **Normalization**: Applied to reduce redundancy
- **Auth Security**: Implemented through user authentication and authorization. Mainly session storage is used for user authentication.
- **Php PDO driver** is used for database interactions, so that it can also connect to other databases like PostgreSQL, SQLite, etc.
- **Parameterized arguments** were used to prevent SQL injection attacks.
- **Database triggers** were used to automatically update total_contribution, total_submission and total_solved per user.

3.2. Schema Diagram

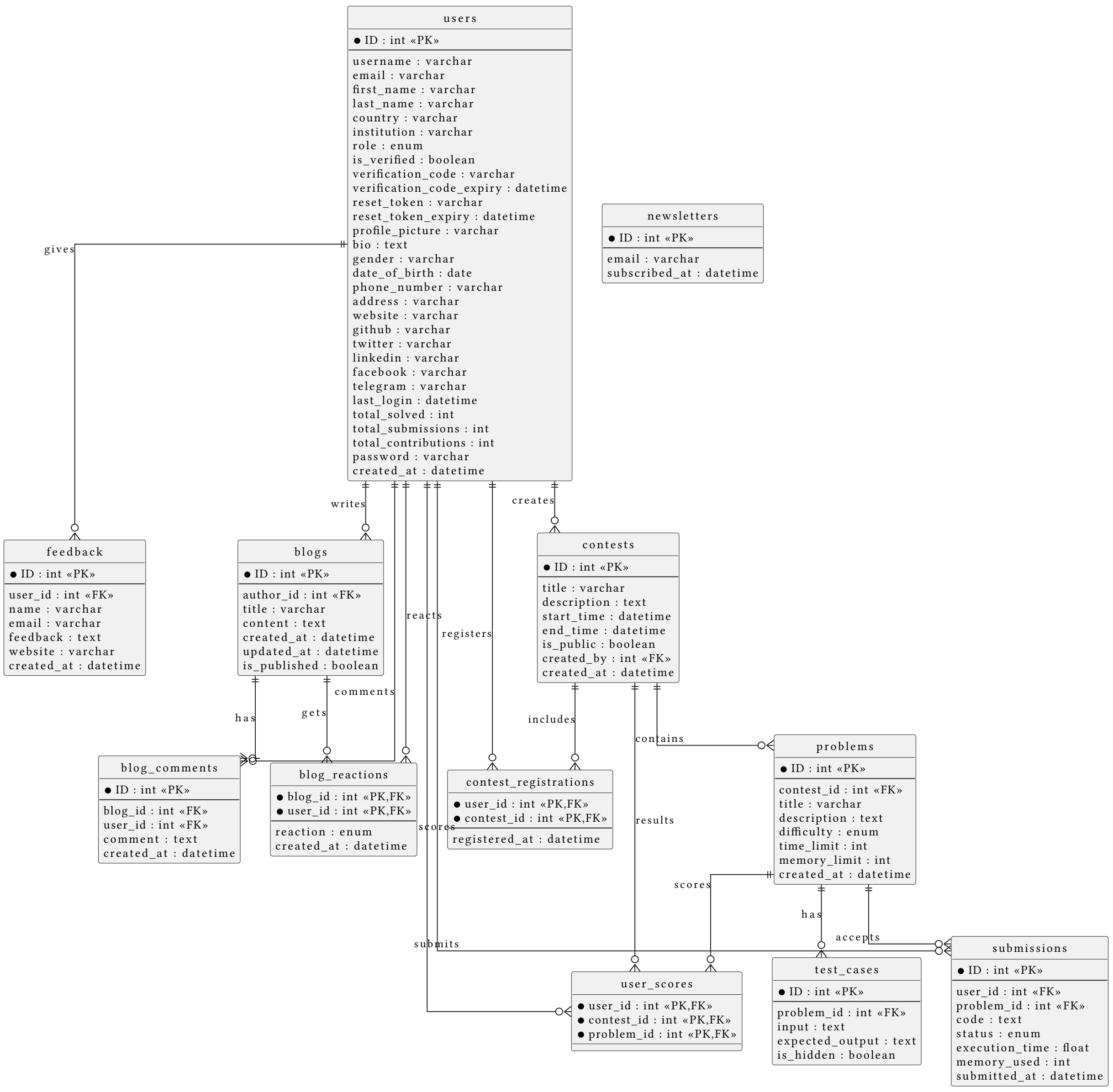


3.3. E-R Diagram

3.3.1. Without attributes

I will do it tomorrow, I guess 😅

3.3.2. With all attributes



4. Database Implementation

4.1. DDL

Data definition language statements,

4.1.1. Database Creation

```sql
CREATE DATABASE IF NOT EXISTS sql_judge;
USE sql_judge;
```

4.1.2. Table Creation

4.1.2.1. Users Table

```sql
DROP TABLE IF EXISTS users;
CREATE TABLE users (
    ID INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    username VARCHAR(50) NOT NULL UNIQUE,
    email VARCHAR(100) NOT NULL,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    country VARCHAR(50),
    institution VARCHAR(100),
    role ENUM('admin', 'user') NOT NULL DEFAULT 'user',
    is_verified BOOLEAN DEFAULT FALSE,
    verification_code VARCHAR(255),
    verification_code_expiry DATETIME,
    reset_token VARCHAR(255),
    reset_token_expiry DATETIME,
    profile_picture VARCHAR(255),
    bio TEXT,
    gender VARCHAR(10),
    date_of_birth DATE,
    phone_number VARCHAR(20),
    address VARCHAR(255),
    website VARCHAR(255),
    github VARCHAR(255),
    twitter VARCHAR(255),
    linkedin VARCHAR(255),
    facebook VARCHAR(255),
    telegram VARCHAR(255),
    last_login DATETIME,
    total_solved INT DEFAULT 0,
    total_submissions INT DEFAULT 0,
    total_contributions INT DEFAULT 0,
    password VARCHAR(255) NOT NULL,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP
);
```

4.1.2.2. Feedback Table

```sql
DROP TABLE IF EXISTS feedback;
CREATE TABLE feedback (
    ID INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    user_id INT,
    name VARCHAR(50) NOT NULL,
    email VARCHAR(100),
    feedback TEXT NOT NULL,
    website VARCHAR(255),
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES users(ID)
);
```

4.1.2.3. Blogs and Comments

```sql
DROP TABLE IF EXISTS blogs;
CREATE TABLE blogs (
    ID INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    author_id INT NOT NULL,
    title VARCHAR(255) NOT NULL,
    content TEXT NOT NULL, -- HTML content
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    updated_at DATETIME ON UPDATE CURRENT_TIMESTAMP,
    is_published BOOLEAN DEFAULT FALSE,
    FOREIGN KEY (author_id) REFERENCES users(ID)
);
```

```sql
DROP TABLE IF EXISTS blog_comments;
CREATE TABLE blog_comments (
    ID INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    blog_id INT NOT NULL,
    user_id INT NOT NULL,
    comment TEXT NOT NULL,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (blog_id) REFERENCES blogs(ID),
    FOREIGN KEY (user_id) REFERENCES users(ID)
);

DROP TABLE IF EXISTS blog_reactions;
CREATE TABLE blog_reactions (
    blog_id INT,
    user_id INT,
    reaction ENUM('like', 'dislike') NOT NULL,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (blog_id, user_id),
    FOREIGN KEY (blog_id) REFERENCES blogs(ID),
    FOREIGN KEY (user_id) REFERENCES users(ID)
);
```

4.1.2.4. Newsletters Table

```sql
DROP TABLE IF EXISTS newsletters;
CREATE TABLE newsletters (
    ID INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    email VARCHAR(100) NOT NULL UNIQUE,
    subscribed_at DATETIME DEFAULT CURRENT_TIMESTAMP
);
```

4.1.2.5. Contests, Problems

```sql
DROP TABLE IF EXISTS contests;
CREATE TABLE contests (
    ID INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    title VARCHAR(100) NOT NULL,
    description TEXT,
    start_time DATETIME NOT NULL,
    end_time DATETIME NOT NULL,
    is_public BOOLEAN DEFAULT TRUE,
    created_by INT NOT NULL,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (created_by) REFERENCES users(ID)
);

DROP TABLE IF EXISTS contest_registrations;
CREATE TABLE contest_registrations (
    user_id INT NOT NULL,
    contest_id INT NOT NULL,
    registered_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (user_id, contest_id),
    FOREIGN KEY (user_id) REFERENCES users(ID),
    FOREIGN KEY (contest_id) REFERENCES contests(ID)
);

DROP TABLE IF EXISTS problems;
CREATE TABLE problems (
    ID INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    contest_id INT,
    title VARCHAR(100) NOT NULL,
    description TEXT NOT NULL,
    difficulty ENUM('easy', 'medium', 'hard') DEFAULT 'medium',
    time_limit INT DEFAULT 2, -- in seconds
    memory_limit INT DEFAULT 256, -- in MB
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (contest_id) REFERENCES contests(ID)
);

DROP TABLE IF EXISTS test_cases;
CREATE TABLE test_cases (
    ID INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    problem_id INT NOT NULL,
    input TEXT,
    expected_output TEXT,
    is_hidden BOOLEAN DEFAULT FALSE,
    FOREIGN KEY (problem_id) REFERENCES problems(ID)
);
```

4.1.2.6. Submissions and User Scores

```
DROP TABLE IF EXISTS submissions;
CREATE TABLE submissions (
    ID INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    user_id INT NOT NULL,
    problem_id INT NOT NULL,
    code TEXT NOT NULL,
    status ENUM('pending', 'running', 'accepted', 'wrong_answer',
'time_limit_exceeded', 'runtime_error') DEFAULT 'pending',
    execution_time FLOAT,
    memory_used INT,
    submitted_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES users(ID),
    FOREIGN KEY (problem_id) REFERENCES problems(ID)
);

DROP TABLE IF EXISTS user_scores;
CREATE TABLE user_scores (
    user_id INT NOT NULL,
    contest_id INT NOT NULL,
    problem_id INT NOT NULL,
    PRIMARY KEY (user_id, contest_id, problem_id),
    FOREIGN KEY (user_id) REFERENCES users(ID),
    FOREIGN KEY (contest_id) REFERENCES contests(ID),
    FOREIGN KEY (problem_id) REFERENCES problems(ID)
);
```

4.1.3. Triggers

```
-- A trigger to increment total_contributions for the author when a new blog is
published
DELIMITER $$
CREATE TRIGGER increment_contributions_after_insert
AFTER INSERT ON blogs
FOR EACH ROW
BEGIN
```

```sql
    IF NEW.is_published = TRUE THEN
        UPDATE users
        SET total_contributions = total_contributions + 5
        WHERE ID = NEW.author_id;
    END IF;
END$$
DELIMITER ;

-- Trigger to decrement total_contributions when a blog is updated from published to
draft
DELIMITER $$
CREATE TRIGGER decrement_contributions_after_update_to_draft
AFTER UPDATE ON blogs
FOR EACH ROW
BEGIN
    IF OLD.is_published = TRUE AND NEW.is_published = FALSE THEN
        UPDATE users
        SET total_contributions = total_contributions - 5
        WHERE ID = NEW.author_id;
    END IF;
END$$
DELIMITER ;

-- Trigger to increment total_contributions when a blog is updated from draft to
published
DELIMITER $$
CREATE TRIGGER increment_contributions_after_update_to_publish
AFTER UPDATE ON blogs
FOR EACH ROW
BEGIN
    IF OLD.is_published = FALSE AND NEW.is_published = TRUE THEN
        UPDATE users
        SET total_contributions = total_contributions + 5
        WHERE ID = NEW.author_id;
    END IF;
END$$
DELIMITER ;

-- Trigger to increment total_contributions by 1 when a new comment is added
DELIMITER $$
CREATE TRIGGER increment_contributions_after_comment
AFTER INSERT ON blog_comments
FOR EACH ROW
BEGIN
    UPDATE users
    SET total_contributions = total_contributions + 1
    WHERE ID = NEW.user_id;
END$$
DELIMITER ;

-- Trigger to increment total_submissions by 1 when a new submission is added
DELIMITER $$
CREATE TRIGGER increment_total_submissions_after_insert
AFTER INSERT ON submissions
FOR EACH ROW
BEGIN
```

```
    UPDATE users
    SET total_submissions = total_submissions + 1
    WHERE ID = NEW.user_id;
END$$
DELIMITER ;

-- Trigger to increment total_solved by 1 when a new user_scores entry is added
DELIMITER $$
CREATE TRIGGER increment_total_solved_after_user_score_insert
AFTER INSERT ON user_scores
FOR EACH ROW
BEGIN
    UPDATE users
    SET total_solved = total_solved + 1
    WHERE ID = NEW.user_id;
END$$
DELIMITER ;
```

4.1.4. Views

```
-- View to get the top 5 users based on total_solved
CREATE VIEW top_rated_5 as
SELECT username, first_name, last_name, total_solved
FROM users
ORDER BY total_solved DESC LIMIT 5;

-- View to get the top 5 users based on total_contributions
CREATE VIEW top_contributors_5 as
SELECT username, first_name, last_name, total_contributions
FROM users
ORDER BY total_contributions DESC LIMIT 5;
```

4.2. SQL Queries

4.2.1. Authentication

1. **User Registration**

   ```
   INSERT INTO users (username, email, first_name, last_name, password, website, bio)
   VALUES (:username, :email, :first_name, :last_name, :password, :website, :bio)
   ```

2. **User Login**

   ```
   SELECT id, username, password FROM users WHERE username = :username"
   ```

3. **Check if user already exists**

   ```
   SELECT id FROM users WHERE username = :username"
   ```

4.2.2. User Profile

1. **Get user profile**

   ```
   SELECT * FROM users WHERE username = :username

     # ID, username, email, first_name, last_name, country, institution, role,
   is_verified, verification_code, verification_code_expiry, reset_token,
   reset_token_expiry, profile_picture, bio, gendar, date_of_birth, phone_number,
   address, website, github, twitter, linkedin, facebook, telegram, last_login,
   total_solved, total_submissions, total_contributions, password, created_at
     1, sharafat, sharafat@duck.com, Sharafat, Karim, Bangladesh, PSTU, user,
   0, , , , , , There&#039;s no end to EXPLORATION!, Male, 2002-11-08, 01953546089,
   ```

```
Patuakhali, https://sharafat.pages.dev, https://github.com/SharafatKarim, , , ,
https://t.me/SharafatKarim, , 0, 1, 32, $2y$12$CceqDu/
Ww9T44k2SdgT5DuzeeyR2ZanlSD8rvZlA/MXcGd3iC2Gbe, 2025-04-08 06:44:18
```

2. **Update user profile**

```
UPDATE users SET
          first_name = :first_name,
          last_name = :last_name,
          email = :email,
          country = :country,
          address = :address,
          institution = :institution,
          bio = :bio,
          gender = :gender,
          date_of_birth = :date_of_birth,
          phone_number = :phone_number,
          website = :website,
          github = :github,
          twitter = :twitter,
          linkedin = :linkedin,
          facebook = :facebook,
          telegram = :telegram
        WHERE username = :username"
```

4.2.3. Blog

1. **Get all blogs**

```
  SELECT blogs.ID, blogs.title, blogs.content, blogs.created_at, users.username,
blogs.is_published
  FROM blogs
  JOIN users ON blogs.author_id = users.ID
  WHERE blogs.is_published = 1 OR blogs.author_id = 1
  ORDER BY blogs.created_at DESC

  # ID, title, content, created_at, username, is_published
'9', 'Testing publishing feature of blogs!', 'This should be published
publicly...', '2025-05-26 04:31:06', 'sharafat', '1'
```

2. **Insert a new data (blog)**

```
INSERT INTO blogs (author_id, title, content, is_published) VALUES
(:author_id, :title, :content, :is_published)
```

3. **Update a blog**

```
UPDATE blogs SET
          title = :title,
          content = :content,
          is_published = :is_published
        WHERE ID = :blog_id"
```

4. **Delete a blog**

```
DELETE FROM blogs WHERE ID = :blog_id
```

4.2.4. Comment & React

1. **Insert a new comment**

```
INSERT INTO blog_comments (blog_id, user_id, comment) VALUES
(:blog_id, :user_id, :comment)
```

2. **Get all comments for a blog**

```
  SELECT blog_comments.comment, blog_comments.created_at, users.username
  FROM blog_comments
  JOIN users
  ON blog_comments.user_id = 1
  WHERE blog_comments.blog_id = 1
  ORDER BY blog_comments.created_at DESC

  # comment, created_at, username
'interesting!', '2025-05-24 03:37:50', 'a'
```

3. **Fetch reactions**

```
SELECT reaction, COUNT(*) as count
FROM blog_reactions
WHERE blog_id = 1 GROUP BY reaction
```

4.2.5. Leaderboard

1. **Get top 5 users based on total_solved**

```
  SELECT *
  FROM top_rated_5

# username, first_name, last_name, total_solved
'sharafat', 'Sharafat', 'Karim', '1'
'a', 'A', 'B', '0'
'b', 'b', 'b', '0'
```

2. **Get 50 user's rank based on total_contribution**

```
SELECT first_name, last_name, username, total_contributions
                  FROM users
                  ORDER BY total_contributions DESC
                  LIMIT 50"

# first_name, last_name, username, total_solved
'Sharafat', 'Karim', 'sharafat', '1'
'A', 'B', 'a', '2'
'b', 'b', 'b', '0'
```

3. **Get 50 user's rank based on total_submission**

```
SELECT first_name, last_name, username, total_submissions
                  FROM users
                  ORDER BY total_submissions DESC
                  LIMIT 50

# first_name, last_name, username, total_submissions
'Sharafat', 'Karim', 'sharafat', '1'
'A', 'B', 'a', '2'
'b', 'b', 'b', '0'
```

4. **Get 50 user's rank based on total_contribution**

```
  SELECT first_name, last_name, username, total_contributions
                  FROM users
                  ORDER BY total_contributions DESC
                  LIMIT 50
```

```
# first_name, last_name, username, total_contributions
'Sharafat', 'Karim', 'sharafat', '32'
'A', 'B', 'a', '1'
'b', 'b', 'b', '0'
```

5. Conclusion

Finally we can conclude that, SQL Judge platform will help mentors and teachers to help spreading the knowledge of SQL and database management. It will also help students to learn and practice SQL queries in a fun and interactive way. The platform is designed to be user-friendly and easy to navigate, making it accessible to users of all skill levels.

6. References

6.1. Documentations

- https://www.w3schools.com/html/ [**W3Schools HTML**]
- https://www.w3schools.com/css/ [**W3Schools CSS**]
- https://www.w3schools.com/js/ [**W3Schools JavaScript**]
- https://www.w3schools.com/sql/ [**W3Schools SQL**]
- https://www.php.net/manual/en/ [**PHP Manual**]
- https://www.w3schools.com/php/default.asp [**W3Schools PHP**]

THE END