Table of Contents

Introduction

This user manual is intended for developers, faculty members, and students working with the OMM Questionnaire Study Bank web application. The purpose of this manual is to provide clear, structured instructions for setting up, understanding, and maintaining the OMM application, which helps students prepare for the osteopathic manipulative medicine (OMM) board exams.

The application is a full-stack web platform built using Flutter for the frontend and Flask for the backend. It features a robust database with tagging, tracking, and analytics to support effective learning. The system also implements key security practices, such as protection against SQL injection and AWS deployment security.

This document follows a waterfall-style structure and includes instructions for:

- Running the application locally
- Understanding key backend and frontend components
- Accessing and interacting with the database
- Testing and expanding the system

Use this guide to onboard quickly, maintain the system, or expand features securely and efficiently.

Application Purpose

The application is focused on creating the "OMM Questionnaire Study Bank", a means to help students studying for the osteopathic manipulative medicine board exam. OMM is a scalable web application that uses Flutter to make studying easier and more organized. Students can utilize tools like text highlighting, strikethrough, and test tracking progress for enhanced studying processes. The web app aims to improve access and enhance the study process, making it more efficient for both students and faculty.

The project includes several key deliverables: an initial design, a working prototype, testing, and a final, scalable app. The app will need to handle high traffic and implement security measures like AWS security and protection against SQL injections. Collaborating with project stakeholders, we will address questions about functionality, assets, device compatibility, features, and UI integration. The main goal is to develop the "OMM Questionnaire Study Bank," which supports Rowan University students in preparing for the osteopathic manipulative medicine board exam.

The goal of the project is to revise an existing Flask OMM application, adding features like text highlighting and strikethrough, among others. Additionally, we'll be focusing on providing security solutions for the AWS production environment, ensuring that the application is both scalable and secure. The aim is to create an efficient and complete study platform for students. For initial steps, meetings with the team, including some key members, will help clarify further details and alignment.

The purpose of the project is to enhance the existing OMM study bank application by adding features like better viewing stats, saving tests, and improved highlighting with strikeout using mouse clicks. It's also crucial to ensure database security to prevent SQL injections, with proper documentation of database changes. This application should help students study more efficiently for their osteopathic manipulative medicine (OMM) board exams, while maintaining a secure and scalable performance for the application.

Here’s a breakdown of our main objectives:

1. **Improve the Study Process:**
    - Replace static Word documents (or disparate Google Forms) with a centralized, interactive study bank that allows students to easily access and track their progress.
2. **Enhance User Experience:**
    - Revise the existing Flutter application to include new features such as text highlighting, strikethrough, improved viewing of statistics, and a more attractive interface for reviewing past tests.
    - Ensure that the user interface is intuitive and meets the needs of both students and faculty.
3. **Ensure Scalability and Security:**
    - Design the application using proper principles so that it can handle high traffic and be scalable as user numbers grow.
    - Implement robust security measures (especially within the AWS environment and the database) to protect against vulnerabilities like SQL injections.
4. **Deliver a Complete, Well-Documented Product:**
    - Deliverables include an initial design, a working prototype, testing results, and the final application.
    - Document all changes—minor database tweaks and major architectural shifts—with clear records (including ER diagrams when necessary).
5. **Collaborate Effectively with Stakeholders:**
    - Work closely with Dr. Lenny Powell, Sean McKay, and other project sponsors to ensure that the final product meets the actual needs of its users.
    - Clarify any questions regarding assets, feature prioritization, device support, and the specifics of the new functionalities (e.g., how highlighting and strikeouts should work).

System Requirements

- **Operating System**: Windows, macOS, or Linux
- **Python**: Version 3.8 or later
- **Flask**: Version 2.0+ (Python web framework)
- **MySQL Server**: Version 8.0 or later
- **pip**: Python package manager
- **Virtualenv**: For managing isolated Python environments
- **Node.js / npm** *(optional)*: If modifying or extending frontend scripts
- **Modern Web Browser**: Chrome, Firefox, Safari (for accessing the local web interface)

File Tree

```
├── DatabaseFunctions
│   ├── __pycache__
│   │   ├── create_question.cpython-311.pyc
│   │   ├── create_question.cpython-38.pyc
│   │   ├── delete_question.cpython-38.pyc
│   │   ├── get_answer.cpython-311.pyc
│   │   ├── get_answer.cpython-38.pyc
│   │   ├── get_attempts.cpython-311.pyc
│   │   ├── get_attempts.cpython-38.pyc
│   │   ├── get_question.cpython-311.pyc
│   │   ├── get_question.cpython-38.pyc
│   │   ├── get_test.cpython-311.pyc
│   │   ├── get_test.cpython-38.pyc
│   │   ├── insert_answer.cpython-311.pyc
│   │   ├── insert_answer.cpython-38.pyc
│   │   ├── is_question_active.cpython-311.pyc
│   │   ├── is_question_active.cpython-38.pyc
│   │   ├── make_attempt.cpython-311.pyc
│   │   ├── make_attempt.cpython-38.pyc
│   │   ├── make_test.cpython-311.pyc
│   │   ├── make_test.cpython-38.pyc
│   │   ├── prepare_string.cpython-311.pyc
│   │   ├── prepare_string.cpython-38.pyc
│   │   ├── update_attempt.cpython-311.pyc
│   │   └── update_attempt.cpython-38.pyc
│   ├── create_question.py
│   ├── get_answer.py
│   ├── get_attempts.py
│   ├── get_question.py
│   ├── get_test.py
│   ├── insert_answer.py
│   ├── is_question_active.py
│   ├── make_attempt.py
│   ├── make_test.py
│   ├── prepare_string.py
│   └── update_attempt.py
├── Fixes.txt
├── Objects
│   ├── Answer.py
│   ├── Question.py
│   ├── __pycache__
│   │   ├── Answer.cpython-311.pyc
│   │   ├── Answer.cpython-38.pyc
│   │   ├── Question.cpython-311.pyc
│   │   ├── Question.cpython-38.pyc
│   │   ├── testSet.cpython-311.pyc
│   │   └── testSet.cpython-38.pyc
│   ├── massAdd.py
│   └── testSet.py
├── __pycache__
│   ├── addQuestion.cpython-311.pyc
│   ├── config.cpython-311.pyc
│   ├── createTest.cpython-311.pyc
│   ├── database_connection.cpython-311.pyc
│   ├── editQuestion.cpython-311.pyc
│   ├── login.cpython-311.pyc
│   ├── searchQuestion.cpython-311.pyc
│   ├── signup.cpython-311.pyc
│   ├── stats.cpython-311.pyc
│   └── submit_data.cpython-311.pyc
├── addQuestion.py
├── app.py
├── config.py
├── createTest.py
├── database_connection.py
├── editQuestion.py
├── login.py
├── searchQuestion.py
├── signup.py
├── static
│   ├── Rowan_OMM1.jpg
│   ├── Rowan_OMM2.jpg
│   ├── Rowan_OMM3.jpg
│   ├── Rowan_OMM4.jpg
│   ├── background.jpg
│   ├── css
│   │   └── style.css
│   ├── rowan-university.png
│   ├── rowan-university2.jpg
│   └── script.js
├── stats.py
├── submit_data.py
├── templates
│   ├── 404.html
│   ├── addQuestion.html
│   ├── createTest.html
│   ├── dashboard.html
│   ├── editQuestion.html
│   ├── home.html
│   ├── index.html
│   ├── navbar.html
│   ├── searchQuestion.html
│   ├── signup.html
│   ├── success_page.html
│   ├── test.html
│   ├── testResult.html
│   ├── testTemp.html
│   ├── test_database.html
│   ├── viewAttempt.html
│   ├── viewStats.html
│   └── viewTests.html
└── testSet.py
```

Back End

- **config (sub-directory)**
    - **config.py**
        - Purpose:
            - Stores essential configuration settings for the application
            - Defines the database username, password, host, and name
            - Specifies the upload folder where question related images are stored, this ensures that media files are properly managed within the application
- **database (sub-directory)**

- **json processes (sub-directory in "database")**
    - **accounts (sub-directory)**
        - test files
    - **questions (sub-directory)**
        - test files
    - **createJsonFile.py**
        - Purpose:
    - **editJsonFile.py**
        - Purpose:
    - **test.py**
        - Purpose:
    - **README.txt**
        - Purpose:
- **schemas (sub-directory in "database")**
    - **fill_tag.sql**
        - Purpose:
            - This SQL script is used to populate the tag table in the omm database with predefined tags. The tags are categorized into 3 types, category, body region, and treatment techniques
    - **omm_create_statements_2.0.sql**
        - Purpose:
            - This SQL script defines the database schema for the OMM study tool. Sets up the tables and relationships to manage users, tests, questions, answers, attempts, and tagging system.

- **omm_create_view_statements.sql**
    - Purpose:
        - This SQL script creates database views to analyze user performance on tagged questions, count question categories, and retrieve test details for attempts. These views help track student progress, question, distribution, and test composition.
- **test_connection.py**
    - Purpose:
        - This python script is used to test the connection to a MYSQL database using credentials stored in a separate configuration file(config.py)
        - Imports mysql.connector to interact with the MYSQL database, and config from config.py to securely retrieve database credentials.
- **connection.py**
    - Purpose:
        - This function, makeConnection(), establishes a connection to the MySQL Database using credentials stored in the config class.
        - Retrieves the database username, password, host, and database name from the config and uses them to connect via mysql.connector.connect()
        - It returns a connection object that allows interaction with the database.
- **create_question.py**
    - Purpose: The program is a Flask-based backend script that handles the creation of questions for the OMM assessment system. It inserts questions into a database, associates them with tags, stores possible answers, marks correct answers, and manages image uploads for questions and explanations.
- **get_answer.py**
    - Purpose: This function retrieves the selected answer for a given question in a specific test attempt from the database. It uses an SQL query to find the answer_id based on the test_id, attempt_num, and question_id.
- **get_attempts.py**

    - Purpose: Retrieves all test attempts made by a specific user from a database and returns them as a list of lists.
  - **get_question.py**
    - Purpose: Retrieves a question from the database using a given question_ID, constructs a Question object with its associated answers and tags, and links any related images if they exist.
  - **get_test.py**
    - Purpose: Retrieves all questions associated with a specific test from the database, constructs Question objects with their associated answers, and adds them to a TestSet object, which is then returned.
  - **insert_answer.py**
    - Purpose: updates a test attempt if no answer was recorded yet
  - **is_question_active.py**
    - Purpose: checks if a question 1. Exists and 2. has been answered yet
  - **make_attempt.py**
    - Purpose: creates a new test attempt and assigns questions to it
  - **make_test.py**
    - Purpose: 1. Filters questions by tags, creates a SQL query, and executes the query to return question IDs in order to create a test
  - **prepare_string.py**
    - Purpose: Takes string and replaces quotes, double quotes, and backslashes w/ proper backslash placements

    - **update_attempt.py**
        - Purpose: 1. Opens a database cursor 2. Converts score to int 3. Updates attempt table where an update id attempt will match an update id 4. Runs the update command 5. Closes the connection
- **models (sub-directory)**
    - **answer.py**
        - Purpose: Manages each answer to the questions by setting the text, if the answer is correct, and the ID of the answer.
    - **mass_add.py**
        - Purpose: Loads questions using a spreadsheet (ew... blegh! >:[ )
    - **question.py**
        - Purpose: Manages the information regarding the questions, whether it's the text of the questions, answers for the question, or images for the question.
    - **test_set.py**
        - Purpose: Loads an arraylist of questions for the current test.
- **routes (sub-directory)**
    - **add_question.py**
        - Purpose:
            - Handles adding questions to the database in a Flask web application.
            - Processes form submissions from a webpage, extracts question details and associated tags
            - Validates user inputs, and interacts with the database to store questions
            - Manages error handling for tag mismatches and input length constraints
            - Uploads images related to the question and redirects users to a success page upon successful submission
    - **app.py**
        - Purpose:
            - This Flash application serves as the backend for a web-based test management system

        - Handles user authentication, question management, test creation, and test-taking functionality
        - Connects to a database to retrieve and store questions, answers, and test results
        - Users can sign up, login, create and edit questions, search for questions, take tests, and view their test history and statistics
        - Provides various routes for rendering different pages, processing form submissions, and managing user sessions
        - Includes functionality for viewing past test attempts, submitting test data, and error handling
- **admin_route.py**
    - Purpose:
        - Creates an admin user that supersedes (in permissions) and supervises all other users. Creates functionality to toggle user roles.
- **edit_question.py**
    - Purpose:
        - Allows users to edit questions in the database. It handles updating question text, explanations, tags, and associated images.
        - When editing, it deactivates the old question, creates a new one, and transfers old images if necessary.
        - It includes error handling for invalid input and ensures images are properly saved.
- **login.py**
    - Purpose:
        - Handles user login by validating email and password against the database.
        - Encrypts the password using SHA-256 and checks for matching record in the users table if a match is found, it stores user details in the session and redirects to the home page. If the login is invalid, it flashes an error message.
- **search_question.py**
    - Purpose:
        - Handles searching for questions based on their ID or tag.
        - Allows users to search for a question by its ID and redirects to an edit page, or search for questions by a specific tag, displaying results on the search page.

        - Queries the database for questions with the selected tag and returns the results as a list of dictionaries containing the question ID, text, and tag
  - **signup.py**
    - Purpose:
      - Handles user sign up functionality, allowing users to create an account as either a student or faculty member.
      - Checks if the account already exists and validates the input fields ensuring required information like email, name, and password are provided.
      - If the input is valid, the password is encrypted, and the user is added to the database. The user is then redirected to the index page.
      - Includes a helper function getID to retrieve the user's ID based on their email.
  - **submit_data.py**
    - Purpose:
      - Handles the submission of exam data, including the state of each question.
      - Extracts the questionStates from the incoming JSON file and saves it to the session.
      - For the last question in the list, it retrieves the score and time, storing them in the session if available.
      - Inserts each answer into the database by calling insertAnswer for each question's state.
      - Updates the exam attempt with the final score, completion status, and time taken by calling updateAttempt then returns a success message in JSON format.
- **scripts (sub-directory)**

  - **createTest.py**
    - Purpose:
      - The function creates a test by collecting user input, such as the number of questions, selected categories, and test options
      - It connects to the database, retrieves the user's ID, generates a test entry
        and records the attempt. The user is then redirected to the test page
  - **stats.py**
    - Purpose:

- Retrieves and calculates statistics for a specific student based on their attempted questions for different tags.
- Queries the database for the total number of questions, the total attempted questions, and the total correct attempts for each tag.
- Calculates the percentage of answered and correct attempts for each tag, the results are then stored in a dictionary with the tag as the key and the total, answered percentage, and correct percentages as values.
- Closes the database connection and returns the result dictionary.

Front End

- **Static (Directory)**
    - **CSS (sub-directory in "static")**
        - **style.css**
            - Purpose:
        - **various jpg files (they need a directory)**
            - Purpose: photos throughout website
    - **js (sub-directory in "static")**
        - **script.js**
            - Purpose: Allow users (faculty & admin) to view tests of students.
- **templates (sub-directory)**
    - **404.html**
        - Purpose: Simple 404 error pop-up.
    - **add_question.html**
        - Purpose: HTML template for adding questions to quiz then database.
    - **create_test.html**
        - Purpose: Create a test for users.
    - **dashboard.html**
        - Purpose: Display main home page/dashboard for users.
    - **edit_question.html**
        - Purpose: Allows users (faculty & admin) to edit question(s).
    - **home.html**
        - Purpose: Main user home page.
    - **index.html**
        - Purpose: Website index; with login page/validation.
    - **navbar.html**
        - Purpose: Navigation bar for site.
    - **search_question.html**
        - Purpose: Extension of 'home.html' that allows users to search for a question based on a unique ID.
    - **sign_up.html**
        - Purpose: User sign up page.

    - **success_page.html**
        - Purpose: Success page for question adding procedure.
    - **test_database.html**
        - Purpose: Run tests on database (dev mode only).
    - **test.html**
        - Purpose: Actual exam page.
    - **test_result.html**
        - Purpose: Page to present users with their test results.
    - **test_temp.html**
        - Purpose: Temporary test mock-up (dev mode only).
    - **view_attempt.html**
        - Purpose: Allows users to view test attempts.
    - **view_stats.html**
        - Purpose: Allow users (faculty & admin) to view stats of exams.
    - **view_tests.html**
        - Purpose: Allow users (faculty & admin) to view tests of students.
    - **admin_dashboard**
        - Purpose: Provide a visual of users for Admin to modify accounts and toggle them from student to faculty.
- **tests (sub-directory)**
    - **test_app.py**
        - Purpose:
    - **test_set.py**
        - Purpose:
- **text-files (sub-directory)**
    - **fixes.txt**
        - Purpose:
    - **team-6-changes.txt**
        - Purpose:

Running OMM Locally

Prerequisites for creating the database

- brew install mysql
- brew services start mysql
- sudo apt mysql_secure_installation // for secure installation of mysql
- pip install mysql-connector-python // install the connector
- mysql -u root -p // test if the mysql is running
- CREATE DATABASE database_name;
- Change the config.py file to this:

```
class config():
   database_host = 'localhost'  # or your AWS server IP
   database_username = 'root'
   database_password = 'your_password'
   database = ‘database_name'
   upload_folder = 'static/question_images'
```

- Make sure the variables in the database_connection.py file are in line with the config.py file
- Create a test_connection.py file and copy and paste this into there:

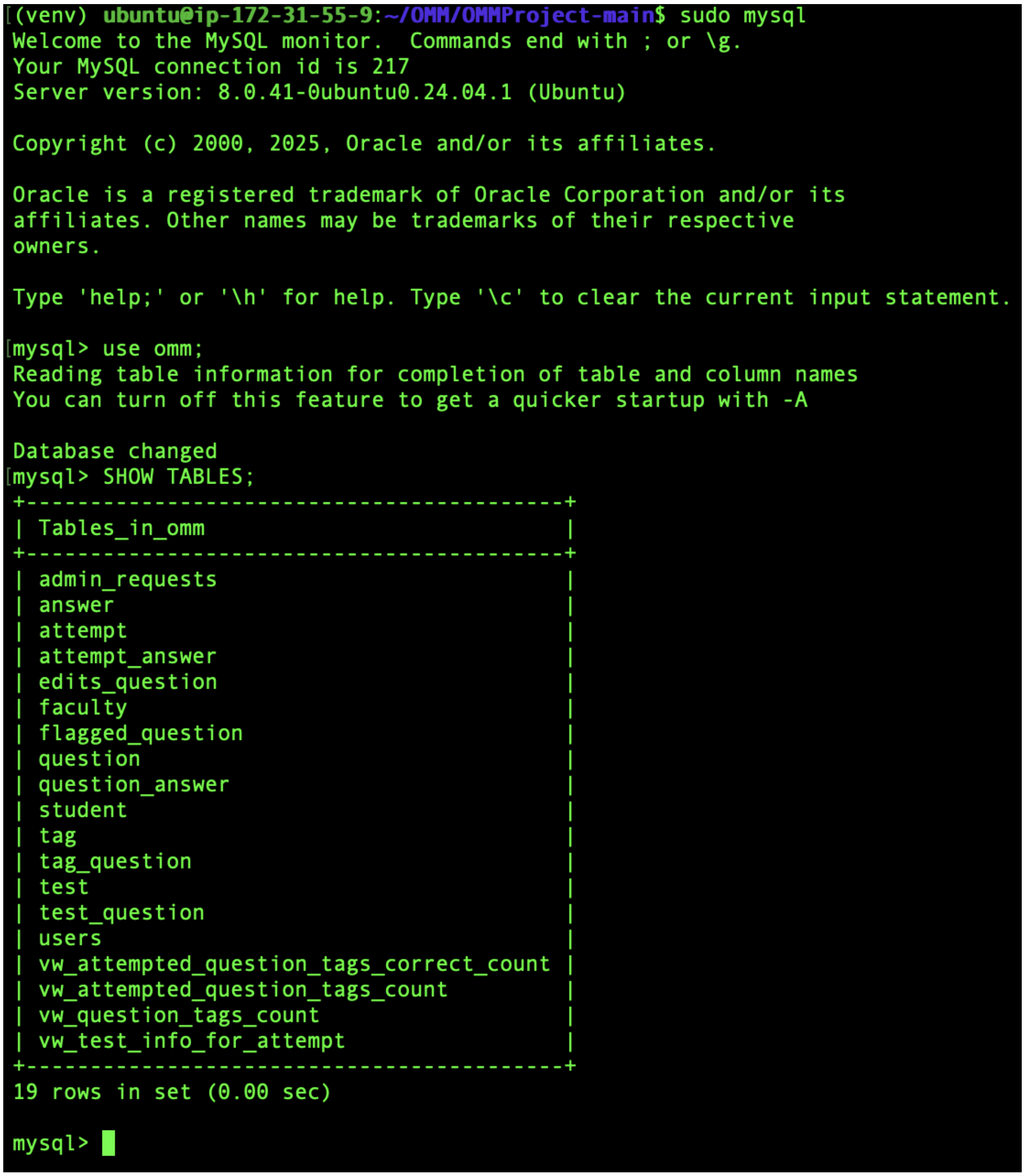
```
import mysql.connector
from config import config  # Import the config class from config.py
def test_database_connection():
   try:
       # Establish the connection using the details from config class
       connection = mysql.connector.connect(
           user=config.database_username,  # Reference class attributes correctly
           password=config.database_password,
           host=config.database_host,
           database=config.database
       )
```

```
        # Check if the connection is successful
        if connection.is_connected():
            print("Connection successful!")
        else:
            print("Failed to connect to the database.")
    except mysql.connector.Error as err:
        print(f"Error: {err}")

    finally:
        # Close the connection
        if 'connection' in locals() and connection.is_connected():
            connection.close()
            print("Connection closed.")
# Call the function to test the connection
test_database_connection()
```

- To run the application run the command in the directory with the file app.py
  - python3 app.py

Accessing the Database



```
[(venv) ubuntu@ip-172-31-55-9:~/OMM/OMMProject-main$ sudo mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 217
Server version: 8.0.41-0ubuntu0.24.04.1 (Ubuntu)

Copyright (c) 2000, 2025, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

[mysql> use omm;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
[mysql> SHOW TABLES;
+-------------------------------------------+
| Tables_in_omm                             |
+-------------------------------------------+
| admin_requests                            |
| answer                                    |
| attempt                                   |
| attempt_answer                            |
| edits_question                            |
| faculty                                   |
| flagged_question                          |
| question                                  |
| question_answer                           |
| student                                   |
| tag                                       |
| tag_question                              |
| test                                      |
| test_question                             |
| users                                     |
| vw_attempted_question_tags_correct_count |
| vw_attempted_question_tags_count          |
| vw_question_tags_count                    |
| vw_test_info_for_attempt                  |
+-------------------------------------------+
19 rows in set (0.00 sec)

mysql> 
```

Usage Guide

Signing Up

- Go to http://localhost:5000/signup
- Enter your name, email, and password
- Choose your role:
  - Student – for taking and reviewing tests
  - Faculty – for adding questions and viewing student performance
- Click Sign Up to complete registration

Using a Student Account

Taking a Test

- Log in to your student account
- Go to the Create Test page
- Select:
  - Number of questions
  - Tags (e.g., HVLA, Cervical, Counterstrain)
- Click Generate Test
- Use the highlight and strikeout tools to mark answer choices
- Click Submit to finish the test

Viewing Results

- Navigate to the View Tests page
- Click any test to see:
  - Score
  - Time taken
  - Missed questions
  - Performance by tag

Using a Faculty Account

Managing Questions

- Log in to your faculty account
- Go to Add Question to:
  - Enter question text
  - Tag the question by category, body region, and technique
  - Upload related images (if needed)
- Use Edit Question to update or deactivate existing questions