CSC3170 Final Project Report
CUHKSZ Student Club Database and Management System

I Overall Project Description

Introduction This project is a database system designed for managing data, records and activities of stud ent clubs at university (fixed as the Chinese University of Hong Kong (Shenzhen) in the back-end). Whe n I was the president of Statistics Alliance, we found that many important materials of past activities can help provide assistance for further issues. Then we propose to build up our own database system and d emonstrate our experience on the Internet. As an origin basic version for the proposal, the project is imp lemented in multiple languages, including HTML (for web content), CSS (for web layout), Javascript (em bedded for web action), Python (for back-end framework), and SQL (for access to data in MySQL datab ase). The virtual environment of the project consists of only MySQL and Python with packages PyMySQ L, Flask and crytography, which is very clear and portable to transfer to other environment. The project consists of 3 roles (admin, club manager, student) and 7 types of different tables with different schemas, where user information are stored in three tables for security to access. Instead of comments, the project depends on very clear naming methods and logging system for debugging. The project consists of 6 pyt hon scripts where four of them are the core parts: dbcursor.py provides an interface to interact with data base in SQL, dbbuilder.py provide an easy start-up back-end program to start the database system, dbman ager consists of large amounts of database operations for student club system, webrender.py interacts with the webpages for rendering under Flask framework. (The left two stores default database setting such as the university name selected before the project.) In the process, all the data are stored in the database.
Environment Setup The project is implemented in multiple language, including HTML, CSS, JavaScript for front-end and Python, SQL for back-end. Setup with Anaconda (conda=4.11.0), the environment consis ts of MySQL=9.0.1, python=3.10 (python packages: PyMySQL=1.1.1, Flask=3.0.3, crytography). The proje ct is tested and performed well on MacOS with localhost=127.0.0.1:5000.

II Requirement Analysis

Background When I was the president of Statistics Alliance (a student club in CUHKSZ), we found that many important materials (including data and texts, and also pictures, recordings, slides) about the past activities can help provide great assistance for further club activities. Then we propose to build up our o wn database system and demonstrate our experience on the Internet. This project is the origin version of this proposal. Futhermore, we will try to support much more functions and data format and make it mor e practical.
Project Objectives: Concretely, the project not only support student clubs to demonstrate information abo ut clubs and events and compute statistics on students who participate in, but also provide a platform for students to join clubs and events and get or review related information about clubs and events. The proj ect maintains information about users (students), clubs, and events.

III Database and SQL Design

Roles Design & Role Previledge: There are three types of roles (**student, club manager, administrator**) in the system which is implemented as priority property (user, club-admin, admin). In the basic design of database system, supposing m administrator and n student clubs, there are (n+m) roles with (n+m) dat abases where each administrator has its database to store its own data. Generally, we use only one admi nistrator (m=1) to build a database and management system for student clubs in a university. Different cl ub manager cannot visit the database of the other clubs. And all the club managers can visit student_clu b_database (created by admin) while admin can visit all the database, including those databases created b y club managers.

User Role Functions: Users can check(display) information (user profile, list of clubs, list of events, list of events hosted by the club), update user profile (including password), register/login account, join a clu b, join an event, review the list of clubs/events that the user participate in.

Club Manager Role Functions: Club managers can check(display) information (club profile, list of clubs, list of events, list of events hosted by the club), update club profile (including password), display event information, update event information, create an event, check the statistics on students in club and relate d events, and search for students' personal information (including contact) via .

Admin Role Functions: Admins can create databases/users and visit all the tables under any database. S pecially, only admins can help create club manager account in the database system.

Schema Design: There are 7 tables (userlist, userinfo, usercontact, clublist, eventlist, club_student, event_s tudent) in the student_club_database created by admin to record global information. There are 2 tables (e ventlist, studentlist) in the database of each club manager to record the events and students of the club. PRIMARY KEYs are in red and FOREIGN KEYs are in blue.

userlist: Basic account information for users (students), club-admins (club managers) and admin(s).

userid	username	passwd	priority	regdate

userinfo: students' personal information that can choose to be public.

userid	gender	affilication	stuid	grade	school	major	interest	username

usercontact: user information about students' contact (more secured and not public).

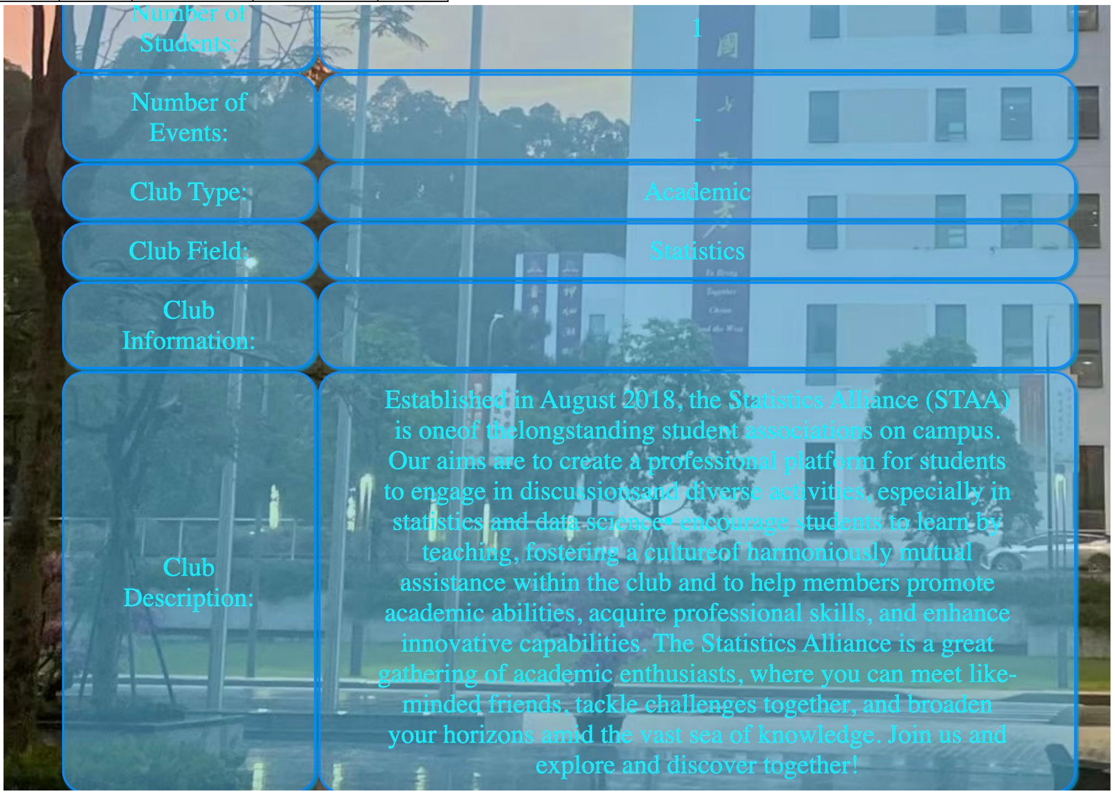
userid	email	phone	wechat	address	postal	username

clublist: information related to club.

clubid	clubname	affilication	school	num_students	num_events	annual_budget	last_year_expense

num_students/num_events: number of students/events in club. club_info/club_desc: used for web design.

type	field	club_info	club_desc	club



Number of Students:	1
Number of Events:	-
Club Type:	Academic
Club Field:	Statistics
Club Information:	
Club Description:	Established in August 2018, the Statistics Alliance (STAA) is oneof thelongstanding student associations on campus. Our aims are to create a professional platform for students to engage in discussionsand diverse activities, especially in statistics and data science• encourage students to learn by teaching, fostering a cultureof harmoniously mutual assistance within the club and to help members promote academic abilities, acquire professional skills, and enhance innovative capabilities. The Statistics Alliance is a great gathering of academic enthusiasts, where you can meet like-minded friends, tackle challenges together, and broaden your horizons amid the vast sea of knowledge. Join us and explore and discover together!

Club Information

eventlist: information related to event. student amount (num_students) cannot exceed quotas (lim_students)

eventid	event	club	club_name	num_students	lim_students	event_date	location	join_ddl

join_ddl: the deadline for student registration. event_info, event_desc: used for web design.

completed	out_campus	transportation	type	budget	event_info	event_desc

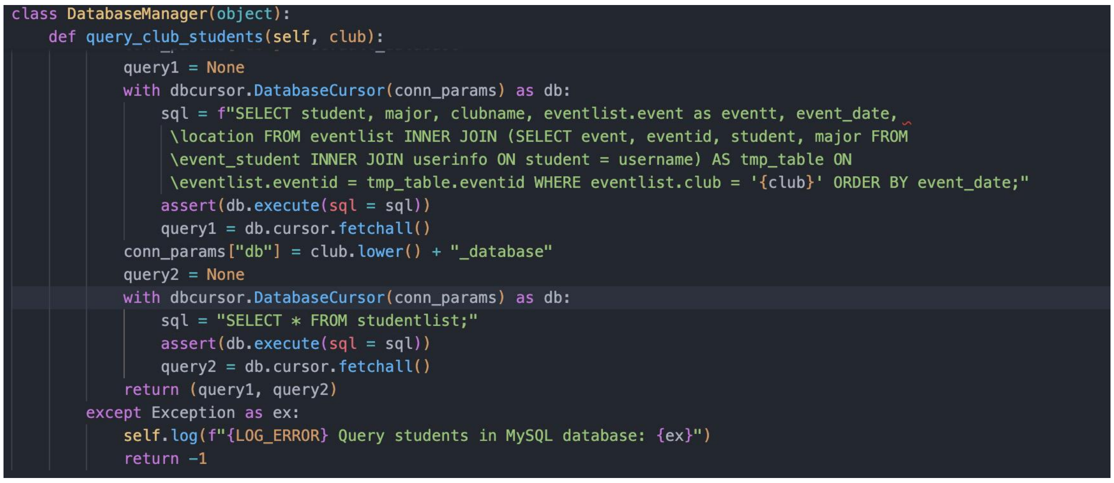
club_student: information about students who joined the club

cspairid	club	clubid	clubname	studentid	student

event_student: information about students who registered the event

espairid	event	eventid	studentid	student

Schema for studentlist table (not mentioned) under each database of club manager is actually the combination of userlist, userinfo and usercontact to form the complete information record of students in the club.


```
class DatabaseManager(object):
    def query_club_students(self, club):
            query1 = None
            with dbcursor.DatabaseCursor(conn_params) as db:
                sql = f"SELECT student, major, clubname, eventlist.event as eventt, event_date,
                 \location FROM eventlist INNER JOIN (SELECT event, eventid, student, major FROM
                 \event_student INNER JOIN userinfo ON student = username) AS tmp_table ON
                 \eventlist.eventid = tmp_table.eventid WHERE eventlist.club = '{club}' ORDER BY event_date;"
                assert(db.execute(sql = sql))
                query1 = db.cursor.fetchall()
            conn_params["db"] = club.lower() + "_database"
            query2 = None
            with dbcursor.DatabaseCursor(conn_params) as db:
                sql = "SELECT * FROM studentlist;"
                assert(db.execute(sql = sql))
                query2 = db.cursor.fetchall()
            return (query1, query2)
        except Exception as ex:
            self.log(f"{LOG_ERROR} Query students in MySQL database: {ex}")
            return -1
```

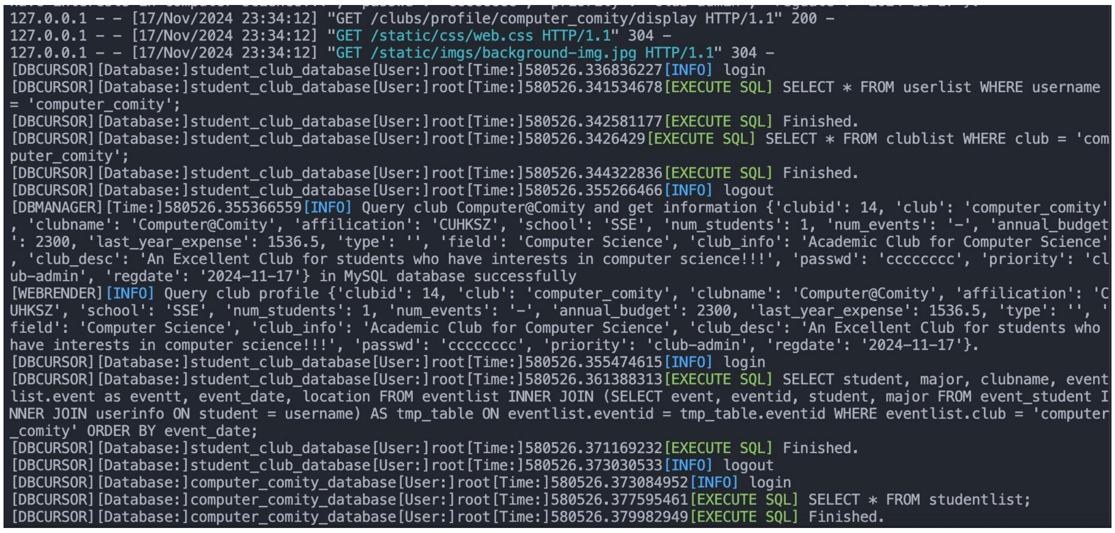
SQL Design: The database system truly refers to some very complex SQL commands, such as the figure above which show a very complex query with inner joinned tables and multiple conditions to sieze out all the information about (student, club, event) triple where student participates the event and the event is hosted by the club. (Here, club is fixed for every query and each query can provide club manager students' information such as distribution on different major background and favorite by students with which gender / grade / school using similar query.)

IV Front-end and Back-end Design and Implementation

Back-end Design: The project builds the back-end based on PyMySQL interfaces and Flask framework. The back-end design is implemented in object-oriented programming paradigm. The class DatabaseCursor() in dbcursor.py provides a MySQL cursor with given setting (username, password and database), which can directly be invoked to execute SQL commands with sufficient logging for debugging. Based on DatabaseCursor(), the class DatabaseManager() in dbmanager.py is a database manager provides multiple methods to complete different database problems given by the front-end, including basic operations to create/drop/update users/databases/tables and insert data. DatabaseManager() also provides higher-level operations for student club management such as user/club manager/admin creation and profile management, students' registration for clubs/events, and statistics on students in clubs/events for club managers.

Front-end Design: The web-based project builds the front-end with HTML, CSS, and Javascript to have iteraction with the Flask framework in the back-end. The script webrender.py is not implemented in object-oriented programming paradigm. Instead, the scripts provides a large amount of functions to render different webpages, query data from the database via DatabaseCursor() and DatabaseManager() classes and transfer data to the front-end on web-page. I also write web.css style file to stylize the website created by the database system and use Javascript functions embedded to add search function and iterate the data which are in table format transferred from queries to MySQL database in the back-end.

Logging System Design: the project has a very nice logging system for debugging, which can clearly display how webpage render, database manager and database cursor (actually sql executor) work at each monment.



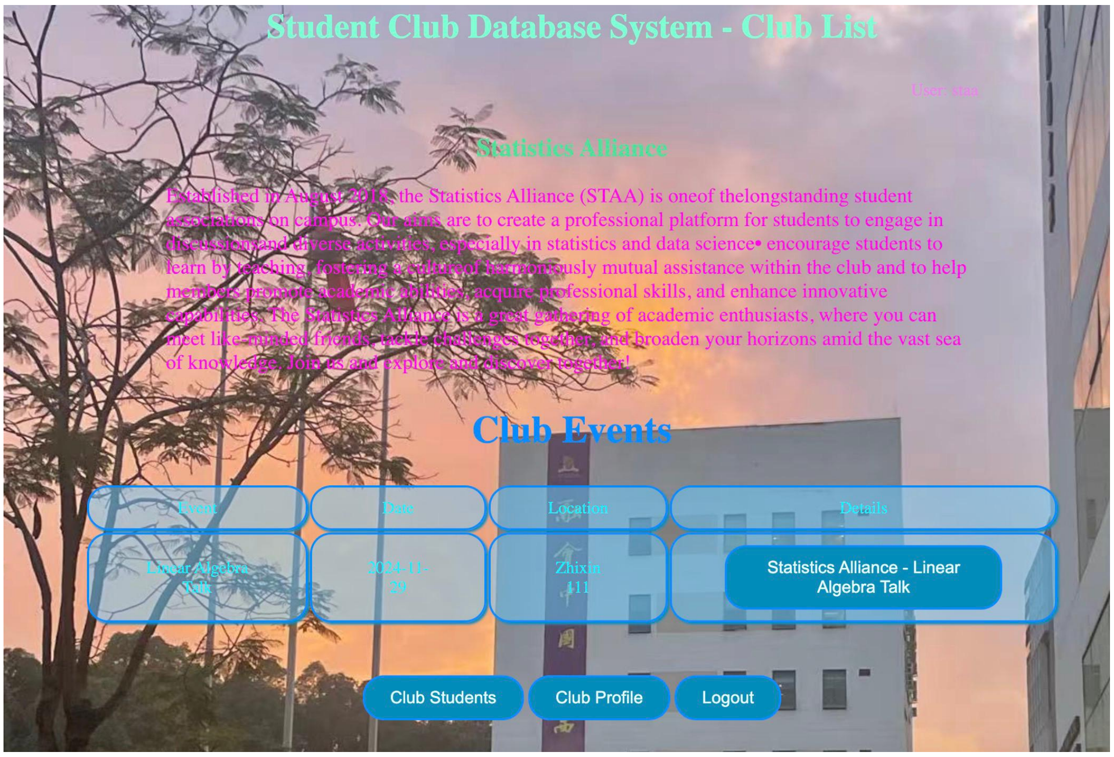
```
127.0.0.1 - - [17/Nov/2024 23:34:12] "GET /clubs/profile/computer_comity/display HTTP/1.1" 200 -
127.0.0.1 - - [17/Nov/2024 23:34:12] "GET /static/css/web.css HTTP/1.1" 304 -
127.0.0.1 - - [17/Nov/2024 23:34:12] "GET /static/imgs/background-img.jpg HTTP/1.1" 304 -
[DBCURSOR][Database:]student_club_database[User:]root[Time:]580526.336836227[INFO] login
[DBCURSOR][Database:]student_club_database[User:]root[Time:]580526.341534678[EXECUTE SQL] SELECT * FROM userlist WHERE username
= 'computer_comity';
[DBCURSOR][Database:]student_club_database[User:]root[Time:]580526.342581177[EXECUTE SQL] Finished.
[DBCURSOR][Database:]student_club_database[User:]root[Time:]580526.3426429[EXECUTE SQL] SELECT * FROM clublist WHERE club = 'com
puter_comity';
[DBCURSOR][Database:]student_club_database[User:]root[Time:]580526.344322836[EXECUTE SQL] Finished.
[DBCURSOR][Database:]student_club_database[User:]root[Time:]580526.355266466[INFO] logout
[DBMANAGER][Time:]580526.355366559[INFO] Query club Computer@Comity and get information {'clubid': 14, 'club': 'computer_comity'
, 'clubname': 'Computer@Comity', 'affilication': 'CUHKSZ', 'school': 'SSE', 'num_students': 1, 'num_events': '-', 'annual_budget
': 2300, 'last_year_expense': 1536.5, 'type': '', 'field': 'Computer Science', 'club_info': 'Academic Club for Computer Science'
, 'club_desc': 'An Excellent Club for students who have interests in computer science!!!', 'passwd': 'cccccccc', 'priority': 'cl
ub-admin', 'regdate': '2024-11-17'} in MySQL database successfully
[WEBRENDER][INFO] Query club profile {'clubid': 14, 'club': 'computer_comity', 'clubname': 'Computer@Comity', 'affilication': 'C
UHKSZ', 'school': 'SSE', 'num_students': 1, 'num_events': '-', 'annual_budget': 2300, 'last_year_expense': 1536.5, 'type': '', '
field': 'Computer Science', 'club_info': 'Academic Club for Computer Science', 'club_desc': 'An Excellent Club for students who
have interests in computer science!!!', 'passwd': 'cccccccc', 'priority': 'club-admin', 'regdate': '2024-11-17'}.
[DBCURSOR][Database:]student_club_database[User:]root[Time:]580526.355474615[INFO] login
[DBCURSOR][Database:]student_club_database[User:]root[Time:]580526.361388313[EXECUTE SQL] SELECT student, major, clubname, event
list.event as eventt, event_date, location FROM eventlist INNER JOIN (SELECT event, eventid, student, major FROM event_student I
NNER JOIN userinfo ON student = username) AS tmp_table ON eventlist.eventid = tmp_table.eventid WHERE eventlist.club = 'computer
_comity' ORDER BY event_date;
[DBCURSOR][Database:]student_club_database[User:]root[Time:]580526.371169232[EXECUTE SQL] Finished.
[DBCURSOR][Database:]student_club_database[User:]root[Time:]580526.373030533[INFO] logout
[DBCURSOR][Database:]computer_comity_database[User:]root[Time:]580526.373084952[INFO] login
[DBCURSOR][Database:]computer_comity_database[User:]root[Time:]580526.377595461[EXECUTE SQL] SELECT * FROM studentlist;
[DBCURSOR][Database:]computer_comity_database[User:]root[Time:]580526.379982949[EXECUTE SQL] Finished.
```

Requirement-Function Form: Here is a form of to show the list of requirements and how to implement it with functions.

Requirements	Functions to Implement	Function Introduction
User (student/ club manager/ admin) Login	[Front-end] index(), login(), login_enter(), index.html, login.html	At “localhost/”, user click to login and index() redirect the page to “localhost/login” rendered by login(). User input username and password, click enter, and login_enter() ask back-end to check.
	[Back-end] query_user_priority()	query_user_priority() asks MySQL for user information based on given username and password, then returns user id and priority.
	[Front-end] login_enter()	login_enter() records the user id and check priority. If administrator, redirect to “localhost/admin”. If club manager, redirect to “localhost/clubs/profile/<club>”. If student, redirect to “localhost/profile/<user>”.
User Logout	[Front-end] logout(), index(), index.html	In almost each webpage, we have logout button. Once clicked, the front-end() will redirect the logout() and back to “localhost/”.
	[Back-end] dbcursor.__exit__()	Back-end will commit updates automatically.
User (student) Registration	[Front-end] index(), register(), register_note(), register_enter(), index.html, register.html	At “localhost/”, user click to register and index() redirect the page to “localhost/register” rendered by register(). User create an account with username and password, click enter, and register_enter() ask back-end to check.
	[Back-end] query_user_priority(), insert_userlist()	query_user_priority() get no priority if username and password pair not exist. Then create an account with insert_userlist() to update userlist, userinfo, usercontact tables.
	[Front-end] profile(), profile_note(), profile.html	Redirect to user’s index page which showing system’s and user’s basic information.
Display User Profile	[Front-end] profile_display(), profile.html, profile-display.html	Ask for user information from back-end by calling query_user(). Then, use GET method to transfer user information on “localhost/profile/<user>” page.
	[Back-end] query_user()	Fetch all the user data from userlist, userinfo, usercontact tables.
Update User Profile	[Front-end] profile_update(), profile_complete(), profile-update.html	Ask the user information from back-end, and then render the webpage.
	[Back-end] query_user()update_user()	Fetch all the user information based on the common userid.
Create Admin	[Back-end] dbbuilder.py	A python script specially designed for create admin account. Build a DatabaseManager() object and create admin user, database (student_club_database) and multiple major tables.
Create Club (adm	[Front-end] admin_create_	At “localhost/admin/createclub”, admin input the usernam

in)	club(), admin/create-club.html	e, club name and password to create a new account for club manager.
	[Back-end] create_club(), query_user_priority()	Check the user’s priority. If no priority, create an accout for new club manager with priority. Create database and tables for club.
	[Front-end] club_creation_complete()	Get information from website, and redirect to admin’s index page after club creation.
Display Club Profile (club manager)	[Front-end] club_profile(), club_profile_note(), club_profile_display(), club-admin/profile.html, club-admin/profile-display.html	Ask for club information from back-end by calling query_club(). Then, use GET method to transfer user information on “localhost/clubs/profile/<club>” page.
	[Back-end] query_club(), /clubs/	Fetch all the club data from clublist table with the club manager username (i.e. club variable).
Update Club Profile	[Front-end] club_profile_update(), club_profile_complete(), club-admin/profile-update.html	Ask the club information from back-end, and then render the webpage.
	[Back-end] query_club() update_club()	Fetch all the information of the club whose club manager account’s username is club.
Create Event (club manager)	[Front-end] club_create_event(), event_creation_complete(), club-admin/create-event.html	Get input from “localhost/clubs/<club>/createevent” which describe the new event with event name, quotas, date, location ... Then get club information and ask back-end to create records to store the event data and query for the new created query id to render the event webpage.
	[Back-end] query_club(), create_event(), query_event_id()	First fetch club information for web render, then create an event with its profile data and join it into eventlist tables and fetch the new generated eventid for web render to rendering the next page i.e. the new event detail page.
Display Event Information	[Front-end] event_profile(), event_profile_display(), event/profile.html, event/profile-display.html	Ask for event information from back-end by calling query_event(). Then, use GET method to transfer user information on “localhost/clubs/<club>/events/profile/eventid=<eventid>/display” page.
	[Back-end] query_club(), query_event()	Fetch all the event related data from eventlist table with eventid variable.
Update Event Information	[Front-end] event_profile_update(), event_profile_complete(), event/profile-update.html	Ask the event information from back-end, and then render the webpage “localhost/clubs/<club>/events/profile/eventid=<eventid>/update”.
	[Back-end] query_club(), query_event(), update_event()	Fetch all the information of the event with given eventid bySQL.
List Events of Club	[Front-end] club_profile_events(), club-admin/events.html	Query club related information to render the webpage. Ask for data about events hosted by given club and transfer the data to the webpage.
	[Back-end] query_club(), query_events()	Fetch club related information first. Then, fetch the all information about the events whose host club is the given club.
List Clubs	[Front-end] clubs(), clubs.html	Render the webpage “localhost/clubs” showing a list of clubs with basic information.
	[Back-end] query_clubs(),	Fetch the selected culumns from the clublist table.
Join Event	[Front-end] event_join()	First, ask for user information and query for all the events that user joined. If user joined the event before, fail to join again. Otherwise, ask the back-end to join user into the selected event.
	[Back-end] query_user(), query_user_events(), query_event(), join_event()	Fetch user information first. Then, fetch all events that user joined from event_student table. If user can join the event, create related record and update tables.
Join Club	[Front-end] club_join()	First ask for user information and check user validity. Then ask for club information to check whether user joined before. If user hasn’t joined in the club, join the user in the club.
	[Back-end] query_user_byname(), query_user(), query_user_clubs(), query_club(), join_club()	First fetch the user’s account information with username andget userid. Then fetch all the user information from userlist, userinfo, and usercontact tables with query_user() referenced by userid. Fetch all the clubs that user has joined from club_student table. If user hasn’t join the club, fetch the club information and add the user to the club.
List Clubs of User (student)	[Front-end] profile_clubs(), profile-clubs.html	Ask for all the information about clubs that user has joined.
	[Back-end] query_user_clubs()	Fetch all the clubs that user has joined from club_student table.
List Events of Us	[Front-end] profile_events	Ask for all the information about events that user has jo

er (student)	(), profile-events.html	ined.
	[Back-end] query_user_ev ents()	Fetch all the events that user has joined from event_stud ent table.
List Events	[Front-end] events(), even ts.html	Render the webpage “localhost/events” showing a list of events with basic information.
	[Back-end] query_events()	Fetch the selected culumns from the eventlist table.
Get Club Informa tion	[Front-end] club(), club.ht ml	To render “localhost/clubs/<club>”, ask for all informatio n about the given club
	[Back-end] query_club()	Fetch the record of the given club from clublist table
Get Event Inform ation	[Front-end] event(), event. html	To render “localhost/clubs/<club>/events/eventid=<eventid >”, ask for all information about the given event
	[Back-end] query_event()	Fetch the record of the given event from eventlist table
List Information of Students Joi ning in Events of Club	[Front-end] club_profile_s tudents()	Directly ask the backend.
	[Back-end] query_club(), query_club_students()	Join tables and fetch the result with INNER JOIN SQL commands and complex conditions to sieze out the resul t.
Search Informatio n about Students in Club	[Front-end] club_profile_s tudents()	Ask for students’ information who join in the club. Writ e prompt-response function in javascript to answer each search of corresponding student.
	[Back-end] query_club(), Query_club_students()	Fetch all information of students who join in the club fr om studentlist table of club manager’s database.


Student Club Database System - Club List

User: staa

Statistics Alliance

Established in August 2018, the Statistics Alliance (STAA) is oneof thelongstanding student associations on campus. Our aims are to create a professional platform for students to engage in discussionsand diverse activities, especially in statistics and data science• encourage students to learn by teaching, fostering a cultureof harmoniously mutual assistance within the club and to help members promote academic abilities, acquire professional skills, and enhance innovative capabilities. The Statistics Alliance is a great gathering of academic enthusiasts, where you can meet like-minded friends, tackle challenges together, and broaden your horizons amid the vast sea of knowledge. Join us and explore and discover together!

Club Events

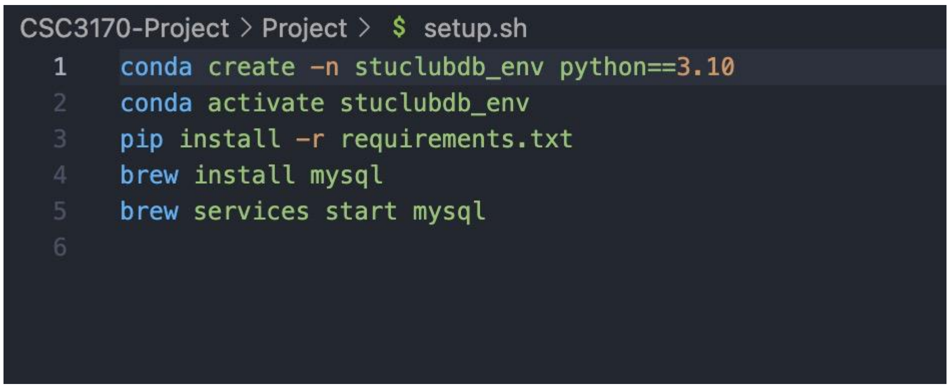
Event	Date	Location	Details
Linear Algebra Talk	2024-11-29	Zhixin 111	Statistics Alliance - Linear Algebra Talk

Function to Display the Events of a Club

V How to Use the Sytem

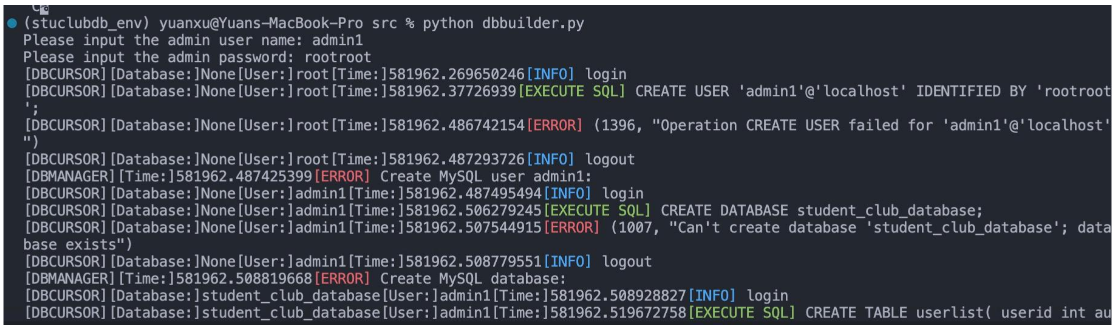
How to Test the Database System:

◆ Step 1: First, environment setup (work on MacOS, unsure for other OS.) Assume that you have inst alled Anaconda to help you control different virtual environments for projects. You can open setup.s h and check the commands to build environment for this project with following commands.



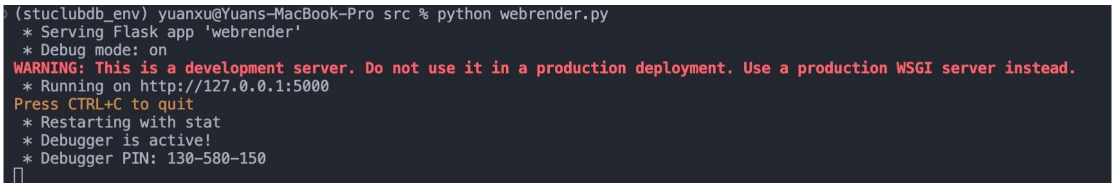
```
CSC3170-Project > Project > $ setup.sh
conda create -n stuclubdb_env python==3.10
conda activate stuclubdb_env
pip install -r requirements.txt
brew install mysql
brew services start mysql
```

◆ Step 2: Second, run python dbbuilder.py to build the database system and create an admin account.



```
● (stuclubdb_env) yuanxu@Yuans-MacBook-Pro src % python dbbuilder.py
  Please input the admin user name: admin1
  Please input the admin password: rootroot
  [DBCURSOR][Database:]None[User:]root[Time:]581962.269650246[INFO] login
  [DBCURSOR][Database:]None[User:]root[Time:]581962.37726939[EXECUTE SQL] CREATE USER 'admin1'@'localhost' IDENTIFIED BY 'rootroot
  ';
  [DBCURSOR][Database:]None[User:]root[Time:]581962.486742154[ERROR] (1396, "Operation CREATE USER failed for 'admin1'@'localhost'
  ")
  [DBCURSOR][Database:]None[User:]root[Time:]581962.487293726[INFO] logout
  [DBMANAGER][Time:]581962.487425399[ERROR] Create MySQL user admin1:
  [DBCURSOR][Database:]None[User:]admin1[Time:]581962.487495494[INFO] login
  [DBCURSOR][Database:]None[User:]admin1[Time:]581962.506279245[EXECUTE SQL] CREATE DATABASE student_club_database;
  [DBCURSOR][Database:]None[User:]admin1[Time:]581962.507544915[ERROR] (1007, "Can't create database 'student_club_database'; data
  base exists")
  [DBCURSOR][Database:]None[User:]admin1[Time:]581962.508779551[INFO] logout
  [DBMANAGER][Time:]581962.508819668[ERROR] Create MySQL database:
  [DBCURSOR][Database:]student_club_database[User:]admin1[Time:]581962.508928827[INFO] login
  [DBCURSOR][Database:]student_club_database[User:]admin1[Time:]581962.519672758[EXECUTE SQL] CREATE TABLE userlist( userid int au
```

◆ Step 3: Third, run python webrender.py and get access to localhost/ (generally, 127.0.0.1:5000) to st art your test on website.



```
 (stuclubdb_env) yuanxu@Yuans-MacBook-Pro src % python webrender.py
  * Serving Flask app 'webrender'
  * Debug mode: on
 WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
  * Running on http://127.0.0.1:5000
 Press CTRL+C to quit
  * Restarting with stat
  * Debugger is active!
  * Debugger PIN: 130-580-150
```