**Part I - Project Information**

**Introduction**

Our database system project “Database Academy” was inspired by the UVA Internal People Search website, as it is an internal database for high school students, staff, and activities. In general, we modelled the high school based on two big attributes; the people and the organizations in it. In general, the people in a high school can be broken into two big categories; teachers and students. In this system, teachers have two roles; all teachers teach a class, and some teachers are sponsors for a club. Each student takes [several] classes, and there are students who are part of one or many clubs (there are also students part of no clubs). The clubs and classes are both types of organizations, and are similar in their overarching design; each has an associated teacher (either as the teacher of the class or sponsor of the club) and multiple students. It is assumed that all the tables in the database at a given time represent data for only the current semester; i.e., the system won’t track a student’s entire list of classes, but rather the list of classes a student is taking this semester. It is assumed that all data in this system will be handled by some system administrator, who will have full responsibility for properly updating the data at the beginning of the semester (once all the students are registered for classes) and will be responsible for inputting the students’ grades into the system at the end of the semester in order for them to be incorporated into the students’ GPAs (which are reflective of their entire time at the school, and not just the current semester).

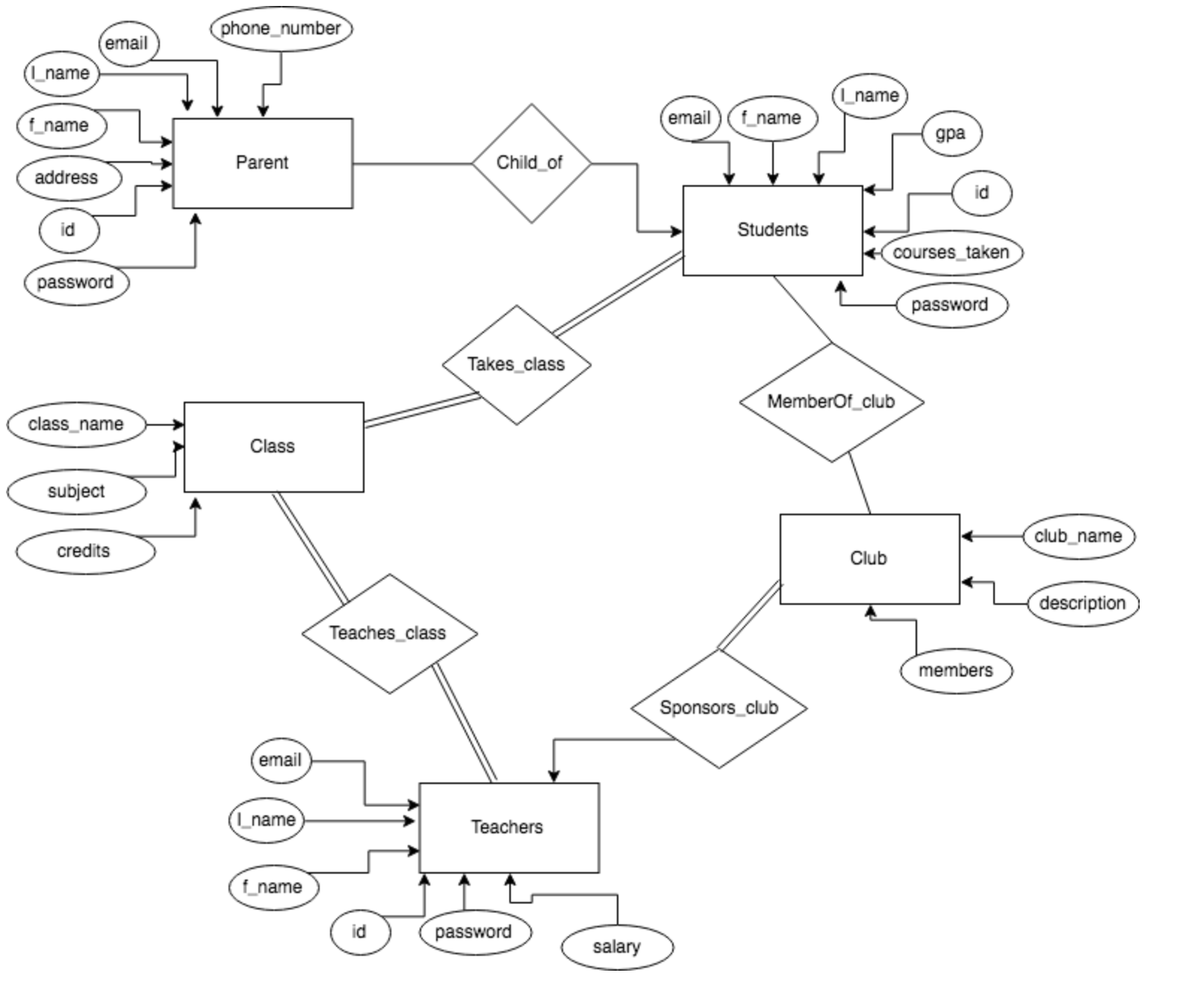
**Requirements**

* The system should be secured with a login page; only designated users should be allowed access to the database.
* If the user is a student at the high school, a teacher, or a parent of a student at the school, they should be able to log in using their email and a password (we assume people logging in already know a username/password combination for a student/teacher/parent they are trying to log in as).
* Any valid user should be able to log in and go to the home screen of the website, which features a menu bar.
* The menu bar should be populated with items relevant to the permission of the user
* All users (students, parents, and teachers) should be able to search (as a baseline permission). In terms of queries, this is represented by these groups having SELECT permission for the tables.
* Parents should not be able to modify or add any data in any table
* Only teachers should be permitted full permissions for the data in tables, including for modifying them (with select queries)
* Teachers should be able to update the GPA for their class for a given student
* Students should be allowed to sign up for clubs
* Data should be downloadable in JSON format
* Students should not be able to download data from the database
* Users should be able to log out.
* The user should be able to see if they are logged in or not.
* GPA should be hidden from all users, although it should be maintained in the database

**Part II - The Design Process**

We began programming our application with the decision to create a series of html pages that would allow you to access the database through search bars. We would link the html page to a specific php file that would actually carry out the correct query on the database and return the result of the query back to the html page. However, as we began developing, we changed our application to better suit the functionalities that we wanted to implement. First, we made our search pages more dynamic by using ajax so the search results could be immediately presented to the screen. We also realized that we needed to handle session data using php, so we changed our flat html files into php files to keep the session data uniform across different screens. In order to make the database more secure, we utilized php to hide the update, delete, and download functionalities from every user that was not a teacher and any user that was not logged in.

**ER Diagram**



**Schemas**

*Entities:*

Parent (email, phone\_number, f\_name, l\_name, address, id, password)

Students (email, f\_name, l\_name, gpa, courses\_taken, id, password)

Class (class\_name, subject, credits)

Club (club\_name, description, members)

Teachers (email, f\_name, l\_name, salary, id, password)

*Relationships*

Child\_of (email (Students) , email (Parent))

Takes\_class (email (Students), class\_name)

MemberOf\_club (email (Students), club\_name)

Teaches\_class (class\_name, email (Teacher))

Sponsors\_club (email (Teacher), club\_name)

**Third Normal Form Proof for Database**

Child\_of

Column has two attributes, [parent\_email] and [student\_email]. Therefore, it is already in BCNF, and by extension, in 3NF form.

Class

*Functional dependencies*: (class\_name -> subject), (class\_name -> credits)

*Canonical Cover:* (class\_name -> subject, credits)

Based on the canonical cover, the database is in the correct 3NF form (class\_name, subject, credits)

Club

*Functional dependencies*: (club\_name -> description), (club\_name -> members)

*Canonical Cover:* (club\_name -> description, members)

Based on the canonical cover, the database is in the correct 3NF form (club\_name, description, members)

MemberOf\_club

Column has two attributes, [student\_email] and [club\_name]. Therefore, it is already in BCNF, and by extension, in 3NF form.

Parent

*Functional dependencies*: (email -> f\_name), (email -> l\_name), (email -> phone\_number), (email -> address), (email -> password), (phone\_number -> f\_name), (phone\_number -> l\_name), (phone\_number -> email), (phone\_number -> address), (phone\_number -> password)

*Canonical Cover:* (email, phone\_number -> f\_name, l\_name,, address, password)

Based on the canonical cover, the database is in the correct 3NF form (email, phone\_number, f\_name, l\_name,, address, password

Sponsors\_club

Column has two attributes, [teacher\_email] and [club\_name]. Therefore, it is already in BCNF, and by extension, in 3NF form.

Students

*Functional dependencies*: (email -> f\_name), (email -> l\_name), (email -> gpa), (email -> classes\_taken), (email -> password), (email -> id)

*Canonical Cover:* (email -> f\_name, l\_name, gpa, classes\_taken, password, id)

Based on the canonical cover, the database is in the correct 3NF form (email, f\_name, l\_name, gpa, classes\_taken, password, id)

Takes Class

Column has two attributes, [student\_email] and [class\_name]. Therefore, it is already in BCNF, and by extension, in 3NF form.

Teachers

*Functional dependencies*: (email -> f\_name), (email -> l\_name), (email -> salary), (email -> password), (email -> id)

*Canonical Cover:* (email -> f\_name, l\_name, salary, password, id)

Based on the canonical cover, the database is in the correct 3NF form (email, f\_name, l\_name, salary, password, id)

Teaches\_class

Column has two attributes, [teacher\_email] and [class\_name]. Therefore, it is already in BCNF, and by extension, in 3NF form.

**Part III - Evaluation of Product**

We first tested the login functionality of our program. First, we tried logging into the website with the correct username/password combinations of various students, teachers, and professors to make sure that all user types had access to the website. Then, we tried logging in with correct usernames but wrong passwords, correct passwords but wrong usernames, and other incorrect username/password combinations to make sure that only the correct users had access to the system.

Next, we tested the search functionality for our system. For each user type (student/teacher/parent) we navigated to the home page and tried different searches. We matched the results of the search to the database to see if the list of people on the results page was actually the people that should have shown up on that search. We also tried searches in which we knew no data from the database would match to and made sure that the correct empty result page was shown. We repeated this process for the courses search page and the clubs search page.

Next, we tested the privacy/security aspects of our system. First, we logged in as a student and made sure we could only view our own GPAs and no other students. We also made sure we couldn’t see the salaries of any of the professors, or edit/update/delete any information from the website. Next, we logged in as a parent and made sure we could only see the GPA of our kids and none of the teachers salaries. We also made sure there was no way for us to update/delete/edit any information. Lastly, we logged in as a teacher and made sure we could not view the salaries of any teacher profile besides our own. We also made sure we could edit/update/add information for any person, club, and course (except the salary info for other professors).

Lastly, we made sure we could easily navigate from Home (People Search Page) 🡪 Course Search Page 🡪 Club Search Page. Then, we tried logging out as a user to make sure it was possible and actually logged us off the website (didn’t allow us to view anything anymore).

**Sample Data and Queries**

*1. People search queries first and last name from parents, teachers, and students*

|  |  |
| --- | --- |
| Query | “ar” |
| SQL | (SELECT f\_name, l\_name FROM Parent WHERE f\_name LIKE ? OR l\_name like ?) UNION (SELECT f\_name, l\_name FROM Teachers WHERE f\_name LIKE ? OR l\_name LIKE ?) UNION (SELECT f\_name, l\_name FROM Students WHERE f\_name LIKE ? OR l\_name LIKE ?) |
| Result |  |

*2. Add student*

|  |  |
| --- | --- |
| Query | First Name: “test”,  Last Name: “test”,  Email: [test@test.com](mailto:test@test.com),  GPA: 4.0,  Password: “test” |
| SQL | INSERT INTO Students (f\_name, l\_name, email, gpa, password) VALUES ('$\_POST[f\_name]', '$\_POST[l\_name]', '$\_POST[email]', '$\_POST[gpa]', '$\_POST[password]') |
| Result |  |

*3. Searching the “clubs” page*

|  |  |
| --- | --- |
| Query |  |
| SQL | SELECT a.club\_name, a.description, b.f\_name, b.l\_name FROM (SELECT Club.club\_name, Club.description, Sponsors\_club.teacher\_email from Club JOIN Sponsors\_club ON Club.club\_name = Sponsors\_club.club\_name) as a JOIN Teachers as b on a.teacher\_email = b.email WHERE a.club\_name like ? |
| Result |  |