

Project #2

Xiaoran Ni

May 10, 2018

1 Introduction

The goal of the project is to design a new social networking site called Jobster that focuses on graduating students looking for jobs.

Specifically, the site should allow students to sign up and post resumes and other information. Companies can also sign up and post information about themselves and job announcements. Students can decide to follow certain companies (which means they will get all job announcements by those companies), can apply for jobs, and can friend other students and forward tips and job announcements to their friends. Companies can search for suitable applicants, and students can search for desirable jobs.

All the data about students, companies, jobs, connections, and notifications is stored in a relational database designed in the first part of the project. In this second part, a web-based interface is designed to present the content to users in a nice form.

In this document, after a brief overview of the database design, a detailed description of the design of the interface will be covered. Each functionalities of the website and implementations will be described.

2 Database Design

2.1 Description

The overall design is illustrated in Figure 1, which is the ER diagram of it. Detailed information and logic of this design are as follows.

2.1.1 Basic entities

The three basic entities are STUDENT, COMPANY, JOB. Considering this is a job searching website:

- For STUDENT, personal information (name) and educational information (university, major, GPA) are attributes, resume is stored as a large file as MEDIUMBLOB type. Here an assumption is made that only one resume is stored in the website for each student. Because different students may have

the same name, and index on student can be anticipated to be used quite often in further queries, a unique id is generated for each student as he signs up on the website first time. Also, for login validation, a unique login name and a password must be resigned by a student and thus saved. In addition, keywords specified by students are stored.

- For COMPANY, a unique id is generated for each company as it signs up for the same reason. Each company have a unique company name and a password must be set. Other attributes are the company name, the location of the company and the industry the company is in.
- For JOB, a unique id is generated for each job. Other attributes are the company id, the job location, the job title, the salary of the job, the background desired in this job and a detailed description of it.

2.1.2 Relational entities

Here relational entities mean entities involve a student and a company, or two different students, or a job and a student of a company. Notification entities are a part of relational entities, but to make things clearer, they are treated as a independent category here.

- FOLLOW stores the information of the relationships between students and companies, a student can follow many companies or none, a company can be followed by many students or no one. A constraint here is that FOLLOW relationship between a student and a company only happens once, hence FOLLOW is a many-to-many relationship. Id of the student, id of the corresponding company and the time when the student follows this specific company are stored.
- To provide the function that students can be friend to each other, FRIEND is created. Two ids of the students involved and the time they become friends are stored. And this is a mutual relationship, in actual storage, considering a student with id_1 and a student with id_2 , when they become friends at time t , both tuple (id_1, id_2, t) and (id_2, id_1, t) are stored instead of storing one of them, though this method is not space efficient, querying about friends of a given student can be computed more efficiently.
- Students can message each other, MESSAGE entity stores the ids of the student who send the message and the one receive it. A student may message another student several times, such that the time of the message is also stored, and an assumption that only one message is allowed at a time is made.
- One of the most important relational entity for this website is the APPLY entity, which stores the id of the student applies, the job id of the job he applies to, the time he applies and the result of this application. The result value can be 'Under view', 'Interviewing', 'Failure' or 'Success'.

2.1.3 Notification entities

The notification is designed in a way that after a user views a notification, this notification is no longer marked to him with a sign in the front end, but it is still

stored in the database. Hence, a field status with a value of 'view' or 'unview' is designed as an attribute of each of them.

- Students can be friends to each other, for two students to become student, one of them have to send a request to the other, when the other student give a confirmation on this request, they become friends, such that a new tuple with their ids and the time will be inserted to the FRIEND table. This process is stored to the table NOTIFY_FRIEND_REQUEST. For the first part, given student with id id_1 sending a friend request to student with id id_2 at time t_1 , a tuple $(id_1, id_2, t_1, null, 'unview', 'view')$ is stored in NOTIFY_FRIEND_REQUEST, the filed for answer is null because student id_2 hasn't replied to this request yet; the answer_status is 'view' hence this request won't be shown on the home page of student id_1 . Meanwhile, a notification pops on the main page of the student id_2 . After the student id_2 clicked some button to see the detail of this notification, $(id_1, id_2, t_1, null, 'unview', 'view')$ is changed to $(id_1, id_2, t_1, null, 'view', 'view')$, and it will no longer show up on the main page of student id_2 . Student id_2 can choose to apply to such request or just disregard. If he choose to be friend of student id_1 , a button with 'accept' is pressed by him at time t_2 , in the mean time, the tuple $(id_1, id_2, t_1, null, 'view', 'view')$ in NOTIFY_FRIEND_REQUEST is changed to $(id_1, id_2, t_1, 'yes', 'view', 'unview')$, additionally, a tuple (id_1, id_2, t_2) is inserted to the relational entity FRIEND. The showing up process of such notification to student id_1 is similar.

- NOTIFY_APPLY is the entity to notify a company that a student has just applied to a job post by them. i.e. A company with id cid posted a job with id jid , when a student with id sid decides to apply to this job at time t , a tuple $(sid, jid, t, 'Underview')$ is inserted to table APPLY, and a tuple $(sid, jid, cid, unview)$ is inserted to NOTIFY_APPLY. The usage of 'unview' and 'view' of this table is similar to as described in NOTIFY_FRIEND_REQUEST and NOTIFY_FRIEND_ANSWER.

- NOTIFY_JOB is the entity to notify a student that a job is announced by a company they are currently following. i.e. When a company with id cid announces a job with id jid at time t , this job is inserted into JOB table first, and if a student with id sid is following this company, a tuple $(jid, sid, t, unview)$ is inserted into NOTIFY_JOB, the usage of 'unview' is similar as above.

2.2 Entity and key relationship

Under the design described in section 2, the schema of the database of job applying website for now is:

STUDENT(sid, sname, login_name, password, university, major, GPA, resume, keywords)
COMPANY(cid, cname, location, industry)
JOB(jid, cid, location, title, salary, background, description)
FOLLOW(sid, cid, time)
FRIEND(sid1, sid2, time)
MESSAGE(from_sid, to_sid, time, content)

FOLLOW.cid is a foreign key referencing COMPANY.cid
 FRIEND.sid1 is a foreign key referencing STUDENT.sid
 FRIEND.sid2 is a foreign key referencing STUDENT.sid
 MESSAGE.from_sid is a foreign key referencing STUDENT.sid
 MESSAGE.to_sid is a foreign key referencing STUDENT.sid
 APPLY.sid is a foreign key referencing STUDENT.sid
 APPLY.jid is a foreign key referencing JOB.jid
 NOTIFY_FRIEND_REQUEST.from_sid is a foreign key referencing STUDENT.sid
 NOTIFY_FRIEND_REQUEST.to_sid is a foreign key referencing STUDENT.sid
 NOTIFY_APPLY.(sid, jid) is a foreign key referencing APPLY.(sid, jid)
 NOTIFY_APPLY.(jid, cid) is a foreign key referencing JOB.(jid, cid)
 NOTIFY_JOB.jid is a foreign key referencing JOB.jid
 NOTIFY_JOB.sid is a foreign key referencing STUDENT.sid

2.3 Procedures

From the practical requirements, several procedures are stored in database to ease the query in php. Based on the functionalities of these procedures, they can be classified into friendship related procedures, follow related procedures and job related ones.

2.3.1 Friendship related

- add_friend(sid1 int, sid2 int): based on the design, tuples (sid1, sid2, NOW()) and (sid2, sid1, NOW()) are both inserted into FRIEND;
 - delete_friend(s1 int, s2 int): tuples in FRIEND with (s1, s2, time) and (s2, s1, time) are both deleted if exists;
 - request_friend(from_id int, to_id int): (from_id, to_id, NOW(), 'unview', 'view') is inserted into NOTIFY_FRIEND_REQUEST with 'answer' set to null;
 - answer_friend(from_id int, to_id int, time, 'yes'/'no'): update the 'answer' of the tuple in NOTIFY_FRIEND_REQUEST to 'yes'/'no', and if 'yes', call add_friend(from_id, to_id).

2.3.2 Follow related

- follow_company(sid int, cid int): insert tuple (sid, cid, NOW()) into FOLLOW table;
- unfollow_company(s int, c int): delete the tuple (s, c, time) in FOLLOW if exists.

2.3.3 Job related

- post_job(cid int, ...): insert a tuple into job, use LAST_INSERT_ID() function of MySQL to get the jid of this newly inserted job, then use this jid to insert tuples into NOTIFY_JOB such that all the students following this companies get notified with this job;

- `apply_job(s int, j int, c int)`: insert (s, j, NOW(), 'Under view') into APPLY, insert (s, j, NOW(), c, 'unview') into NOTIFY_APPLY, such that the company posted this job gets notified automatically.

3 Interface Design

The interface of the JOBSTER website are divided into two independent parts: the system for student users and for company users. For student system, we have a index page, a page for company information, a page for jobs information, a page to connect with people, a page to send messages and a page to handle notifications. For company system, we have a index page, a page to process posted jobs, a page to post jobs and a page to handle notification. Sign up and Log in interfaces for these two sets are similar. The programming environment for server here is PHP 5.6.35 and HTML.

3.1 Sign up/ Login

First on the start page of the website 'start.html', a Jobster header and two links 'student' and 'company' are shown as in Figure 2, user has to select which part of the system they want to use. Because the sign up or login process are similar, only student part will be described as an example.



Figure 2: User Type Selection

Suppose we click on student, we are redirected to 'login_student.php', where an interface with two input groups and a submit button 'login_student' are shown. One is for username, the other is for password. All these elements are combined in a form with method 'post', such that we want to handle the trigger of the submit button in the 'server.php' file, in which we create a session for the whole user experience until the user logs out, \$username, \$name, array \$errors are stored for the whole session.

After 'login_student' button is pressed, `isset($_POST['login_student'])` is true in 'server.php', then whether username and password are blank are checked, if any of them are blank, errors "Username is required." or "Password is required." are pushed into the \$errors array. Next, query are run to check whether this username and password combination is in the STUDENT table, if not, "wrong username/password" is pushed into \$errors. Error shows up as in Figure 3 .The form action is 'login_student.php' itself, such that if any errors are pushed into \$errors, a 'errors.php' file is included to show all the errors. If the

combination is in STUDENT, we set `$_SESSION['username']` and use header to redirect user to 'index_student.php';

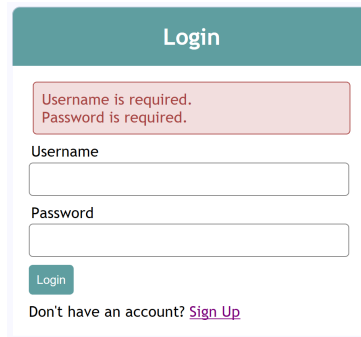


Figure 3: Log In error

If we don't have an account yet, there is a link 'sign up' on 'login_student.php', which redirects user to 'signup_student.php'. In this page, six input groups and a submit button 'signup_student' are shown. Six inputs are username, password, name, university, major and GPA, which corresponds to our attributes design for STUDENT, here resume is not included, which we can upload later. We also use method 'post', after button is clicked, `isset($_POST['signup_student'])` is true in 'server.php', whether username, password and name are blank are check, here we the uniqueness of username is also checked using query. If there are no errors, we use `md5` function to encrypt password, and then insert this new tuple into STUDENT. At last, we set `$_SESSION['username']` and use header to redirect user to 'index_student.php' same as above;

3.2 Page header and footer

All the following pages of both the student system and the company system described further use the same header and footer.

3.2.1 Page header

Here header is the information will be shown on the browser tabs, in which meta data of the page are stored. In file 'head.php', we store the author of web page, the css stylesheet file link and jquery api source. The page title of each page will be shown on the browser tab, as in Figure 4.



Figure 4: Page header

3.2.2 Page footer

In 'footer.php', a copyright of the page are shown, and a javascript function is declared, in which `hover()`, `fadeIn()` and `fadeOut()` function are used to show the drop-down box in 3.3.1 and 3.4.1 when cursor hovers on the certain place of the menu.

3.3 Student system

For student system, we will describe interface based on designed functions: Resume post, find and follow companied, find and apply for jobs, find people and build friendships, send messages to friends and handle notifications. All the pages use the same page structure, header, menu, navigations, content and footer, page header and footer have already been discussed, we will introduce the rest of these first.

3.3.1 Page menu

'menu_student.php' is for student page menu, which consists of three parts: a JOBSTER logo for the website, a link to the help of the website and a drop-down box for setting personal information and logout.

Link is redirect to 'detail_student.php?detail_student=\$_SESSION['student_id']' for setting personal information using 'get' method, where we can view the user's personal information, which will be discussed in find people in detail. There is button 'Edit' in 'detail_student.php?' redirects the page to 'edit_student.php', six input groups and a 'Update' button are shown, there is also an input to select a resume to upload, which will be described in 3.3.2. The process is similar to sign up for student except that `isset($_POST['edit_student'])` is monitored in 'server.php' and a update query will be executed instead of an insertion.

For logout, `_GET['logout']` is monitored in 'server.php', if this gets true, `$_SESSION['username']` gets unset and session is destroyed, page is redirected to 'start.html'.

3.3.2 Resume upload

In 'detail_student.php', a user can find a link to his own uploaded resume right to 'Resume' header, which redirects to 'view_resume.php', method 'get' is used to store student id. 'view_resume.php' is opened in a new page, if the student haven't uploaded a resume, only a sentence 'No resume uploaded.' is shown, otherwise resume is shown in pdf format. Only pdf format is supported.

In edit_student.php, user can upload a file from local to the database using button 'Upload', `isset($_POST['upload_resume'])` is monitored in 'server.php', if set, an UPDATE query is run on STUDENT to upload the resume.

3.3.3 Page navigation

Navigation file 'nav_student.php' for student system consists of six html `` tags inside a ``: Home redirects to 'index_student.php', Company redirects to 'company_student.php', Jobs redirects to 'job_student.php', People redirects to 'people_student.php', Message redirects to 'message.php' and Notifications redirects to 'notify_student.php'. Menu and navigation are shown together in Figure 5.

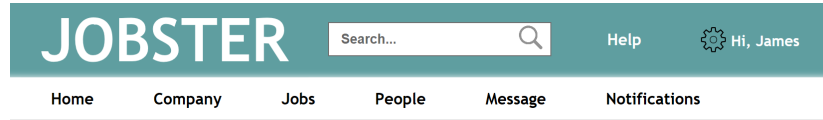


Figure 5: Menu and Navigation

3.3.4 Home page

After a student is redirected to 'index_student.php', the sid for this student user is stored in the session as `$_SESSION['student_id']`.

In 'index_student.php', a query is made on COMPANY, FOLLOW and JOB, jobs posted by all companies following by the student currently are shown in a table in time descendant order. Table attributes are Company, Job Title, Location, Post time and More. More is a link redirects to 'detail_job.php' to show the details of the job, which will be described later in 3.3.6.

3.3.5 Find and follow companies

In 'company_student.php' file, on the top is a search box, then companies are shown in two groups, Following and Other companies. Both are shown in table with table header Name, Location, Industry, Action and More.

In the search box, Location and Industry are two inputs, a 'Search' button can be used to redirects page to 'search_company.php' with 'post' method. In 'search_company.php', query using LIKE operator is made on COMPANY to get information of all the companies meeting searching requirements. If a requirement are blank, we select all the companies meeting the other requirements. If both blank, we select all companies.

In Following group, first a query is made on COMPANY and FOLLOW to get all the companies followed by the user, then cname, location, industry are shown here. Under Action header, user can use button 'unfollow' to un-follow a company which also stored the cid of this company using 'post' method, `$_POST['unfollow']` is monitored in 'server.php', if it is set, procedure 'unfollow_company' will be called in database. A row of this table is shown in Figure 6.

In other companies group, first a query is made to get all the companies not followed by the user, same information are shown as in Following group. Under

Name	Location	Industry	Action	More
Facebook	Seattle	IT	Unfollow	More

Figure 6: Company table

Action, user can use button 'follow' to follow a company, again 'post' method is used, $\$_POST['follow']$ is monitored in 'server.php', if it is set, procedure 'follow_company' will be called in database.

In both groups, under More header, there is a link 'detail_company.php' with the 'get' method storing the cid redirects to the details of the company. On 'detail_company.php', the company information is shown in the first table, then all the jobs posted by it are shown in the second table using query on JOB table.

3.3.6 Apply for jobs and Recommend

In 'job_student.php' file, on the top is a search box, then jobs are also shown in two groups, Applied Jobs and Other Jobs. Both are shown in table with table header Company, Job Title, Location, Action and More.

In the search box, Title and Location are two inputs, a 'Search' button redirects to 'search_job.php'. In 'search_job.php', query is made on JOB, COMPANY to get the job information meeting search criterion, the table format are the same as the below groups.

In Applied Jobs group, a query is made on JOB, COMPANY and APPLY tables to get the cid, cname, title, location. Under Action, the result of this application is shown by query on APPLY table.

In Other Jobs group, a query is made to get information also. Under Action, a button 'apply' can be pressed by user to apply for this job, $\$_POST['apply-job']$ is monitored in 'server.php', and a hidden value of cid is stored, if it is set, procedure 'apply_job' will be called. Once a job is applied, it is moved to Applied Jobs group. A row in Other Jobs group is shown in Figure 7.

Company	Job Title	Location	Action	More
Microsoft	Software Engineer		Apply	More

Figure 7: Job table

In both groups, Company attributes is a link to 'detail_company.php' to show the details of the company. Also under More header, there is a link 'detail_job.php' with the 'get' method storing the jid redirects to the details of the job. On 'detail_job.php', all the job information is shown: Post time, Company, Location, Title, Salary, Background and Description.

At the bottom of 'detail_job.php', there is 'Recommend Job' button redirects to 'recommend_job.php', also transferring a hidden value 'job_id' using 'post'

method. In 'recommend_job.php', there are two tables. In the first one, query is made on JOB to show the brief job information. For the second, query is made on STUDENT and FRIEND to show the friends of the user, there is a 'Recommend' button for each friend, $\$_POST['recommend_job']$ is monitored in 'server.php', if it is set, a tuple is inserted into NOTIFY_JOB using the given jid and sid.

3.3.7 Find people and friendship

In 'people.student.php' file, on the top is a search box, then people are shown in two groups too, Friends and Others. Both are shown in table with header Name, University, Major, Action and More.

In the search box, University and Major are two inputs, a 'Search' button redirects to 'search_people.php'. In 'search_people.php', query is made on STUDENT to get the student user information meeting search criterion, the table format are the same as the below groups.

In Friends group, a query is made on STUDENT and FRIEND to get all the information of each friend of the user, a 'cancel_friendship' button can be used to end the friendship between the user and a given friend, we keep monitoring $\$_POST['cancel_friendship_student']$ in 'server.php', if it is set, procedure delete_friend will be called. A row in Friends group is shown in Figure 8.

Name	University	Major	Action	More
Luna			Cancel friendship	More

Figure 8: Friend table

In Others group, a query is made on STUDENT and FRIEND to get all the information of any other user which is not a friend of the user, a 'request_friendship' button can be used to request friendship to a given user, $\$_POST['cancel_friendship_student']$ is monitored in 'server.php', if it is set, procedure request_friend will be called. Such requests will be treated as notifications in the other user's page, will be covered in the Notification section.

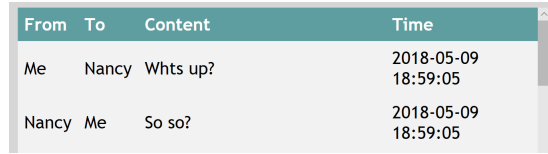
In both groups, under More header, there is a link 'detail_student.php' with 'get' method storing the sid redirect to the details of a user. On 'detail_student.php', all the student information except for the password and username is shown. Plus, as described in 3.3.1, an Edit button is shown to redirects to 'edit_student.php' if sid equals the $\$_SESSION['student_id']$.

3.3.8 Message

In 'message.php', first a query on STUDENT and FRIEND is run to get all the friends of the user, a user can only message his friends. Name, University, Major and Action are the table headers, under Action, a 'message_to' button

can be pressed, which redirects to 'detail_message.php' containing the message history between our user and this friend of his using 'post' method.

In 'detail_message.php', a table containing all the chatting history between these two users and a input box to compose message to this friend are shown. In the first table, a query on MESSAGE and STUDENT is made and show message by time ascending order. In the second box, there is a 'message_content' `<textarea>` and a 'message_student' button, sid of the friend in stored as the hidden input. `$_POST['message_student']` is monitored in 'server.php', if it is set, first whether content is blank is checked, if not blank, then an insertion into MESSAGE is made. Meantime, this tuple is added to the first table. Two messages are shown in Figure 9.



From	To	Content	Time
Me	Nancy	Whats up?	2018-05-09 18:59:05
Nancy	Me	So so?	2018-05-09 18:59:05

Figure 9: Message table

3.3.9 Notification

In 'notify_student.php', there are six tables, three types of notifications.

The first type is Friend Request, table headers are Name, Time and Action, Name stores the name who sent the request to the user, time is the request time. Under Action, a notification for request means this request has not been viewed such that 'receive_status' is 'unview', a query is made on NOTIFY_FRIEND_REQUEST based on this. Two buttons 'Accept' and 'Decline' can be clicked, which correspond to `$_POST['accept_friendship_student']` and `$_POST['decline_friendship_student']` monitored by 'server.php', leading to procedure answer_friend with 'yes' and with 'no' respectively, once the notification is seen by the user, it is moved to history using a update query. In the history table of Friend Request, all the requests received are shown with the same table format.

The second is Request Outcome, the friendship request made by the user to others. Table headers are Name, Time and Outcome. A query is made on NOTIFY_FRIEND_REQUEST where from_sid is `$_SESSION['student_id']` and answer_status is 'unview', the outcome can be 'Agreed' or 'Rejected', once the notification is seen by the user, an update query is made to move it to history table.

The last one is Job Notifications. A query is made on NOTIFY_JOB where status is 'unview', for each job notified, another query is made on APPLY to show the button 'Apply' same as 3.3.6. Once the job is viewed by the user, it is moved to the history table by an update query on NOTIFY_JOB.

3.4 Company system

For company system, designed functions are Job edit and broadcast, Job post and Notifications. Same as for student system, all the pages use the same page structure, header, menu, navigations, content and footer, page header and footer have already been discussed.

3.4.1 Page menu

'menu_company.php' is for company page menu, which consists of three parts: a JOBSTER logo for the website, a link to the help of the website and a drop-down box for setting company information and logout.

Link is redirect to 'setting_company.php' for setting company information, where we can view the company's Company Name, Location and Industry. There is button 'Edit' in 'setting_company.php' redirects the page to 'edit_company.php', three input groups and a 'Update' button are shown, the process is similar to sign up for company except that `isset($_POST['edit_company'])` is monitored in 'server.php' and a update query on COMPANY will be executed instead of an insertion.

For logout, `$_GET['logout']` is monitored in 'server.php', if this gets true, `$_SESSION['companyname']` gets unset and session is destroyed, page is redirected to 'start.html'.

3.4.2 Page navigation

Navigation file 'nav_company.php' for company system consists of four html `` tags inside a ``: Home redirects to 'index_company.php', Jobs redirects to 'job_company.php', Post redirects to 'post_job.php' and Notifications redirects to 'notify_company.php'.

3.4.3 Home page

After a company is redirected to 'index_company.php', the cid for this company user is stored in the session as `$_SESSION['company_id']`. A search box and a table are shown. In the table, a query is made on APPLY, STUDENT and COMPANY to display applications to the jobs posted by the company, with header Apply time, Name, University, Major, Title and Location, data are displayed with time descendant order. Link to the details of the student is attached to Name, which redirects to 'info_student.php' using 'post' method to store the sid. 'info_student.php' is similar to 'detail_student.php', except that if `$_GET['info_student']` is not set or is blank, page is redirected to 'index_company.php'. A link to the resume of the student is also provided, redirecting to 'view_resume.php', same as in 3.3.2.

In the search box, a form using 'post' method redirects to 'search_apply.php', two inputs University, Major and a button Search are shown. These two inputs are treated as keywords to search all the applications this company has received. In 'search_apply.php', a query is made on APPLY, STUDENT and COMPANY

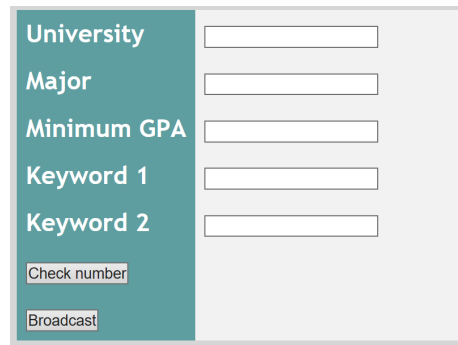
using LIKE operator, then a table with the same attributes as above are shown. Name also redirects to 'info_student.php'.

3.4.4 Job edit and broadcast

In 'job_company.php', a query is made on JOB to select all jobs posted by the company using `$_SESSION['company_id']`, for each job, Job title, Location and More are the headers of the table to show the information. Under the More header, a 'More' button redirects to 'setting_job.php' using 'get' to store 'job_id'.

In 'setting_job.php', there are two tables. The first is the details of the job selected, a query is made on JOB to get time, location, title, salary, background and description of this particular job and is shown in the table, an 'Edit' button redirects to 'edit_job.php', where a form with five input group and an 'Update' button can be used to edit each element of the job, a hidden input is also used to store the job_id, `isset($_POST['edit_job'])` is monitored in 'server.php', if set, whether \$location and \$title are blank get checked and a update query on JOB are made to change the information of the job.

The second table is to broadcast job, shown in Figure 10. It is a form with 'post' method, four input can be entered: University, Major, Minimum GPA, Keyword 1 and Keyword 2, Minimum GPA cannot be blank. Then there are two buttons: one is 'Check number', `isset($_POST['job_checknumber'])` is monitored in 'server.php', if it is set, after \$gpa is checked not to be blank, a select query on STUDENT is made to get the number of students which meet the criterions. This number is shown under the table using array `$count_number`. The second button is 'Broadcast', `isset($_POST['job_broadcast'])` is monitored in 'server.php', if it is set, after \$gpa is checked not to be blank, a insert query on NOTIFY_JOB is made to notify all the students meet this criterion, the state of this broadcast is shown under the table using array `$broadcast_state`.

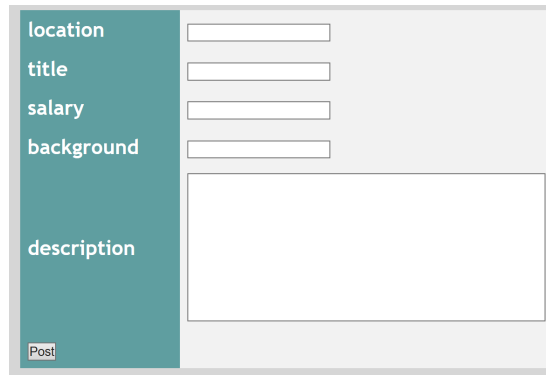


University	<input type="text"/>
Major	<input type="text"/>
Minimum GPA	<input type="text"/>
Keyword 1	<input type="text"/>
Keyword 2	<input type="text"/>
Check number	<input type="button" value="Check number"/>
Broadcast	<input type="button" value="Broadcast"/>

Figure 10: Broadcast table

3.4.5 Job post

In 'post_job.php', a form with method 'post' is the content, there are five inputs each for job location, title, salary, background, description, and a 'Post' button, as shown in Figure 11. `isset($_POST['job-post'])` is monitored in 'server.php', if set, whether \$location and \$title are blank are checked first, if both not blank, procedure post_job is called, and page is redirected to 'job_company.php' to see the posting result.



location	<input type="text"/>
title	<input type="text"/>
salary	<input type="text"/>
background	<input type="text"/>
description	<input type="text"/>
Post	

Figure 11: Job Post Form

3.4.6 Notifications

In 'notify_company.php', there are two tables, both about Applying notifications. First is the real notification, a query on NOTIFY_APPLY where status is 'unview', STUDENT and JOB is made to get the applicant name, university, major, Job title and location information, then an update query is performed on NOTIFY_APPLY to set each notification status as 'view', such that this notification is moved to the second table, which is the applying notifications history table. In both tables, Name attributes redirects to 'info_students.php'.

4 Concurrency Control

There are several parts of the system in need of concurrency control, basically are those modify the data of the database. In mysql InnoDB engine, shared locks and exclusive locks can be set using 'LOCK IN SHARE MODE' and 'FOR UPDATE' respectively. For shared locks, a shared lock on any read rows are set, other sessions can only commit after this transaction commits, but they can read the rows. For exclusive locks, rows and associated index entries are locked, other transactions are blocked from updating those rows, from setting shared locks. It is similar to two-phase lock with same rules set by mysql, but here we only use two basically features.

4.1 Sign Up

In sign up phase, we must check whether the username entered by the user has already be taken before insertion, however, some other user might insert some new username to make the check invalid before our insertion. Hence transaction starts before SELECT, 'FOR UPDATE' is added, thus when inserting, the check is guaranteed to be still valid. The process is added into both student signup and company signup.

4.2 Edit information check

In all edit information parts, there remain checks that a given attribute is unique before insertion, same as sign up, some other users might take this value at the same time, so a transaction is added before check, SELECT is used together with 'FOR UPDATE'.

4.3 Detail information read

Recall that for company, student and job, we all have links to show the details of such a tuple, but the processes are all querying first and then show the results, there is a time space between query and displaying results, during when information may be edited by others. To make sure the results a user sees are the same as in database, a transaction starts before SELECT, and since information is not updated here, only 'LOCK IN SHARE MODE' is needed. Shared lock are added in 'detail_company.php', 'detail_job.php', and 'detail_student.php'.

5 Against SQL injections

To defend possible SQL injections, `mysqli_real_escape_string()` function and prepared statements in MySQLi are used.

5.1 Escape function

Most injections insert ' sign into queries, by using `mysqli_real_escape` function are added before any special marks including ', such that most injections can be defended.

5.2 Prepared statements

To make things even safer, prepared statements are used for most selection queries using input by user. By using `prepare()` function, queries that entail user inputs are prepared and binded with parameters later. The reason prepared statements work is that the database only parse the query once before any input, such that the original statement template is not derived from external input.