**Project Report**

COMP 5531  
Files and Databases  
Professor: Nagi Basha  
TA: Amirali Ashraf  
  
  
Group Name: zic55311  
  
Nicholas LaMothe  
ID: 40114705  
  
Fady Attia  
ID: 40032413

Concordia University  
August 15, 2021

**Reasonable Assumptions**

We assume, based on the lack of security requirements, that there are no stringent security requirements for this application.

As such, important security practices, such as password hashing, have not been implemented. Passwords and security answers are stored in plaintext. For a hosted application with a real user base, this would be considered unacceptable, but for the purpose of this assignment it seems to be acceptable.

Additionally, we have not used prepared statements for data entered by users. Normally, using prepared statements is the standard method for completely ensuring the impossibility of SQL injections performed via user inputs.

However, mysqli\_real\_escape\_string() has been used for each of the $\_POST variables that contains user input. This is generally considered as a good safeguard against SQL injections, but it does not completely remove the risk in the way that prepared statements do.

We have not implemented .htaccess configuration files and Apache directives, which are normally used to control the display of or access to directories within the web server, among other things.

A good deal of validation has been implemented for relevant features, although it is surely incomplete. For the most important features, the user will be immediately notified if the query is successful or if it has failed.

Finally, and importantly, some of the database design does violate 3NF. Our design has at least one relation that is in 2NF. While unnecessary, this helped to simplify certain aspects of querying the database and thus made certain aspects of the back-end code simpler, more readable, and easier to refactor.

**Overview of the Project and Functionality**

We start the application at the **index.html** page within the root directory. Here, we have the simple decision of clicking one of two buttons, **Employer** or **Submit**.

**Employer**

Within the login screen for **Employer**, we can either login if we are of one of two user categories: **admin** or **employer**. Any given correct combination of an admin username and password will successfully redirect, as will any given combination of employer username and password. A correct combination of a <regular> user username and password will *not* successfully redirect.

**Create New Account**

This screen provides the option to create a new employer account. The end user (who is associated with an employer, e.g. a manager, in theory) can enter all of the relevant information and simply create an account, after which the user will be redirected to the login screen.  
  
*The value of the employer\_ID is necessary to create an account. We assume, for the purpose of this project, that an ID is provided to the employer before they create separate accounts for those who are in charge of hiring.*

**Retrieve Your Password**

From the **Employer** login screen, there is a link to retrieve one’s password. As with the login screen, this will only work if the user is either admin or employer. A correct combination of an employer or admin username and security answer will retrieve the password—very insecurely—in plain text on the screen.

**Employer Dashboard**

This is where the most important functionality exists for an employer (or admin):

*Need Help? Contact Us.*

This button, located at the top-right-hand corner of the dashboard, will simply call an alert() function that displays a message telling the user to scroll down where they can read the relevant information.

*User Maintenance*

This first section is only usable by an admin. It allows an administrator to set a user’s status to *active* or *inactive*, and this includes both regular user types (jobseekers) and employers. The ‘Submit’ button must be clicked to perform the action.

*Employer Category*

This will only function for an employer. It uses the $\_SESSION[‘employer’] variable to change the employer’s job\_category to either ‘Employer Prime’ or ‘Employer Gold’ simply by clicking one of the relevant buttons. For each action, an alert() function is called, and then a message is displayed to the end user.  
  
Note that $\_SESSION[] variables are used throughout for authentication.

*Post Job*

Clearly one of the more important pieces of functionality, this is where an employer can post a job. It will only successfully post a job if all of the fields are filled in and the ‘Post Job’ button is clicked.

A primary key (job\_ID) will automatically be created when a job is created. However, the employer must manually enter their employer\_ID. Ideally, the employer should not have to enter this information.

*Current Application Summary*

The functionality of this section goes beyond the requirements, I believe.

If a user is signed in as an administrator (admin), then the panel will automatically—at a glance—display the most recently posted job out of *all* jobs that have been posted (by any employer).

If, however, an employer is signed in, then the panel will display *specifically* the most recently posted job *for that given employer*.

As such, the view for this functionality is properly restrictive.

*Help and Contact*

This section provides a brief overview of the functionality for the employer dashboard view, in addition to providing two administrator e-mail addresses in case the end user has any questions or concerns.

*Update Job*

In this case, having the job\_ID is necessary, as an existing job is being updated. In any case, a user can update any number of attributes of a given job provided that the job\_ID is entered and the ‘Update Job’ button is clicked.

*Update Application*

This feature is to update an application that has already been provided by a user (jobseeker) to an employer. It allows the employer to change the application status (e.g. ‘accepted’, ‘rejected’) and to provide a message for the applicant. All fields must be entered and the ‘Update Application’ button must be clicked.

*Jobs Posted By Employer*

This panel will output all of the jobs that have been posted by the employer currently in session.

*Applications Received By Employer*

This panel will output all of the applications that have been received by the employer currently in session.

*Applications Accepted By Employer*

This panel will output all of the current user’s applications that have been accepted by an employer.

*Return to Home*

Upon redirecting back to the login page, several $\_SESSION variables will be unset.

**Payments**

In the top-right-hand corner of the second employer dashboard panel, there is a button that can be clicked to navigate to the payments page.

Within this page, an employer can add a new payment option, edit a currently existing payment option, or remove a payment option.

Additionally, there is an ‘Account Status’ panel that tells the user whether their account is in good standing.

Importantly, if an account is frozen, then that user will be denied most of the useful functionality of the application, except for a few features that allow for the manipulation/changing of account settings.

*Make a Payment*

From the payments view,, there is a button in the top-right-hand corner of the dashboard. This will navigate to a ‘Make a Payment’ page.

In this page, the employer must enter their account ID, user category, and the total that they wish to pay.

Note that if the employer has insufficient funds in their payments account, then the query will be rejected.

**User**

Within the login screen for **User**, we can either login if we are of one of two user categories: **admin** or **user**. Any given correct combination of an admin username and password will successfully redirect, as will any given combination of user username and password. A correct combination of an employer username and password will *not* successfully redirect.

**Create New Account**

This screen provides the option to create a new account. The user can enter all of the relevant information and simply create an account, after which the user will be redirected to the login screen.

**Retrieve Your Password**

From the **User** login screen, there is a link to retrieve one’s password. As with the login screen, this will only work if the user is either admin or user. A correct combination of a user or admin username and security answer will retrieve the password—very insecurely—in plain text on the screen.

**User Dashboard**

This is where the most important functionality exists for a user (or admin):

*Search All Jobs*

This yellow button—in the top-left-hand corner—upon being clicked, successfully outputs all of the job postings to the ‘Job Data’ panel view.

*Need Help? Contact Us.*

This button, located at the top-right-hand corner of the dashboard, will simply call an alert() function that displays a message telling the user to scroll down where they can read the relevant information.

*Search By Category*

Upon entering an existing job category (e.g. IT, Engineering, Food Preparation) and clicking the ‘Search’ button, jobs that are only of this type will be successfully displayed on the ‘Job Data’ panel view.

*Search By Name*

This is an additional piece of functionality that is not in the requirements.   
  
Upon entering an existing job name (e.g. Web Developer, System Administrator, Barista) and clicking the ‘Search’ button, jobs that are only of this title will be successfully displayed on the ‘Job Data’ panel view

*Job Data*

This view, as previously stated, displays relevant job data.  
  
Note that logic is functional such that the end user can easily switch between the various search options with no errors or bugs.

*Apply for a Job*

This piece of functionality allows the end user to apply for a job posted by any employer. A job\_ID is required, as is the submission text for the application itself. Everything is else is done by the back end. The end user must click the ‘Submit Application’ button.

*Maintain Status*

This feature allows a currently signed-in user to change the status of an application that they have submitted (e.g. ‘active’, ‘inactive’). The end user must provide the ID and the status of the application, and the click the ‘Update Status’ button.  
  
This feature will *correctly* only make such changes for applications that have been submitted by the user who is currently signed in.

*Withdraw Application*

This feature allows a currently signed-in user to withdraw an application that they have submitted. The end user must provide the ID, and then click the ‘Withdraw’ button.  
  
This feature will *correctly* only make such changes for applications that have been submitted by the user who is currently signed in.  
  
Note that this feature does not delete the tuple, but rather nullifies all relevant information.

*Help and Contact*

This section provides a brief overview of the functionality for the user dashboard view, in addition to providing two administrator e-mail addresses in case the end user has any questions or concerns.

*User Category*

This will only function for a user. It uses the $\_SESSION[‘user’] variable to change the user’s job\_category to either ‘User Basic’, ‘User Prime’, or ‘User Gold’ simply by clicking one of the relevant buttons. For each action, an alert() function is called, and then a message is displayed to the end user.

*Update User Profile*

In this feature, a signed-in user can modify any number of attributes in their current profile/account. It uses the $\_SESSION[‘user’] variable and as such will only make modifications for the user currently signed in. The end user must lcick the ‘Update Profile’ button to complete the action.

*Delete User Account*

A longwinded and sardonic message is provied to the end user, asking for the user’s certainty in the decision that they are about to go through with.

In order to “delete” a user account, the user must enter ‘Yes’ into the input box, in addition to clicking the ‘Delete Account’ button. The $\_SESSION[‘user’] variable is once again used so that only the signed-in user’s account can be deleted.

Note that this option does not actually delete the user account tuple, but rather it nullifies all relevant information… except for the username, which must remain for other reasons.

*Applications Sent By User*

This panel will display all of the job applications that have been sent by the user currently in session.

*Applications Accepted*

This panel will display all of thecurrent user’s job applications that were **accepted** by the employer. (In other words, they got the job… or several!)

*Applications Rejected*

This panel will display all of the current user’s job applications that were **rejected** by the employer.

*Return to User Login*

Upon redirecting back to the login page, several $\_SESSION variables will be unset.

**Payments**

In the top-right-hand corner of the second user dashboard panel, there is a button that can be clicked to navigate to the payments page.

Within this page, a user can add a new payment option, edit a currently existing payment option, or remove a payment option.

Additionally, there is an ‘Account Status’ panel that tells the user whether their account is in good standing.

Importantly, if an account is frozen, then that employer will be denied most of the useful functionality of the application, except for a few features that allow for the manipulation/changing of account settings.

*Make a Payment*

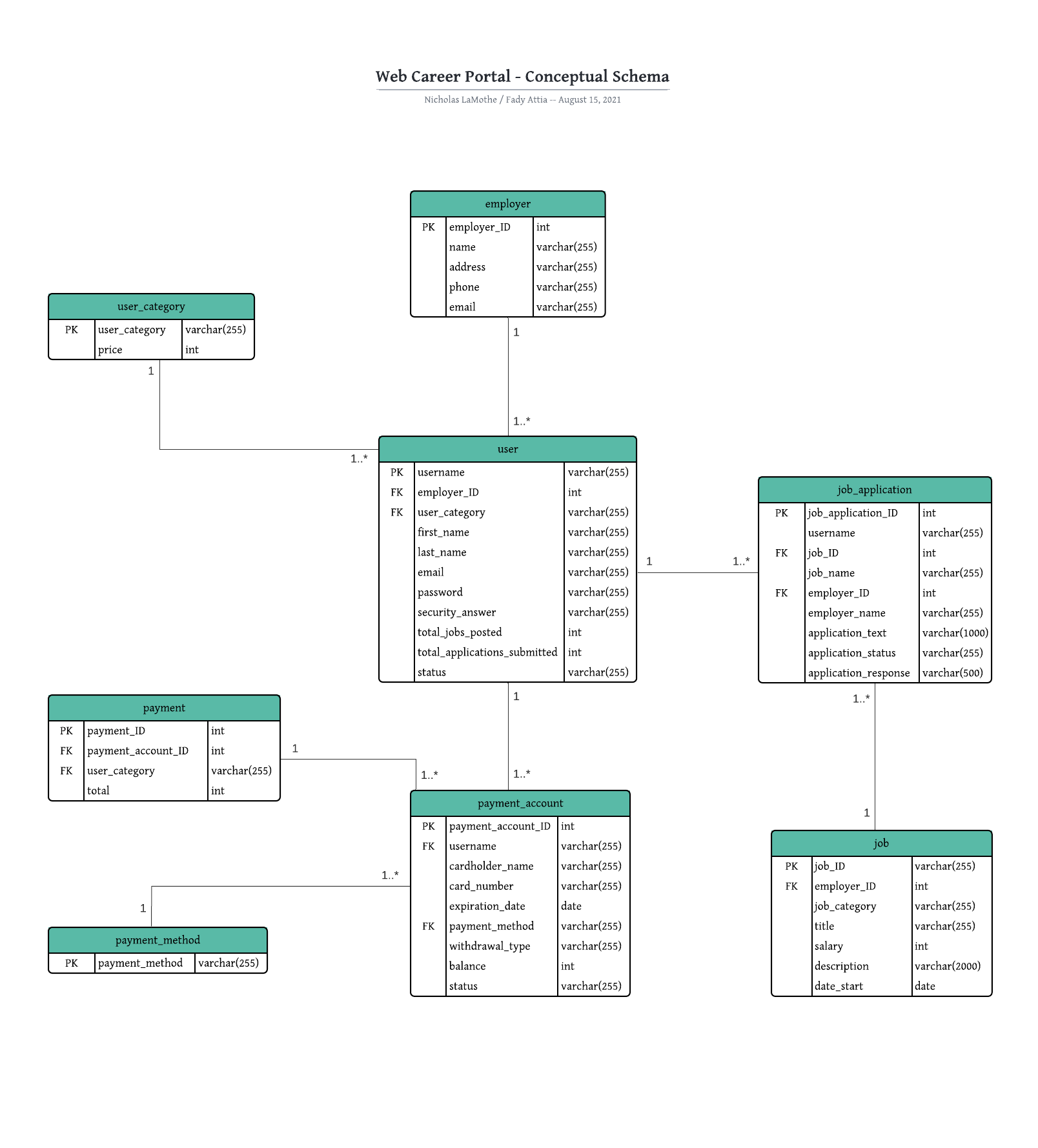
From the payments view, there is a button in the top-right-hand corner of the dashboard. This will navigate to a ‘Make a Payment’ page.

In this page, the user must enter their account ID, user category, and the total that they wish to pay.

Note that if the user has insufficient funds in their payments account, then the query will be rejected.

Diagram

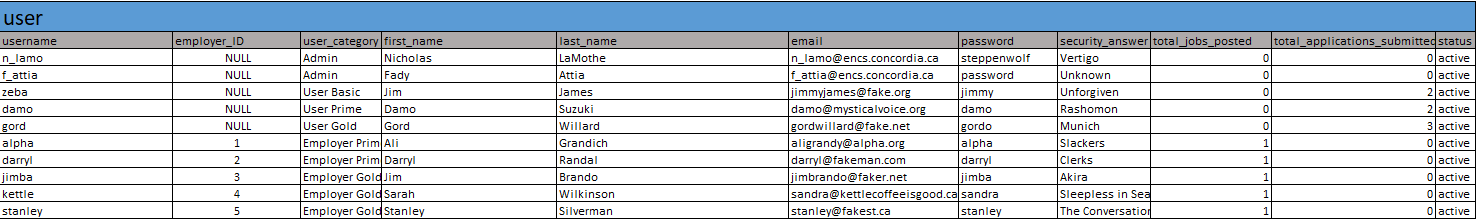
Description automatically generated with medium confidence



**Normalization**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Employer – 3NF | | | | |
| employer\_ID | name | address | phone | email |
| 1 | Alpha Computing | 515 Alpha Way | 555-515-5245 | alphacomputing@alpha.org |
| 2 | Darryl Electronics | 654 Simba Way | 555-515-6216 | darrylelectro@darrylelectro.com |
| 3 | Jimba Microprocessors | 721 Jimba Ave | 555-425-5215 | jimbamicropro@jimba.net |
| 4 | Kettle Coffee | 421 Kettle Boulevard | 555-745-2910 | kettle@kettlecoffeeisgood.ca |
| 5 | Stan\'s Bagels | 425 Bagel Road | 521-542-4919 | stanley@stansbagels.com |
| 6 | Tony Pizzeria | 234 Tony Street | 512-492-1928 | tony@tonypizzanow.org |
| 7 | Denton Photography | 555 Alpha Way | 555-234-8239 | denton@dentonphoto.ca |
| 8 | Fine Refinishing | 7820 Wood Way | 514-849-2938 | refinish@finewood.now |
| 9 | James Richardson Brandy | 514 Avenue Decadent | 512-481-2381 | jrichardson@bestbrandies.org |
| 10 | Smoke Show | 290 Smoker Boulevard | 555-512-1932 | smokeshow@smokeshowsmokeables.org |

Primary Key

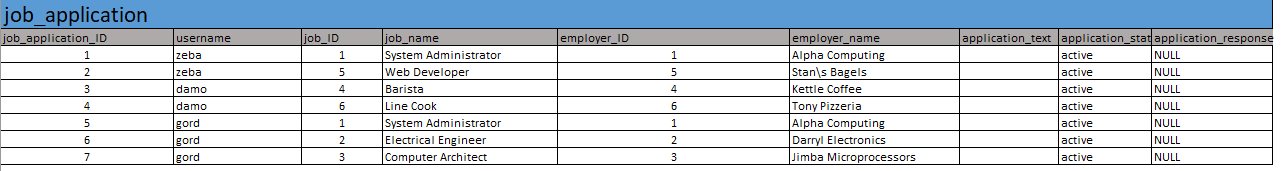
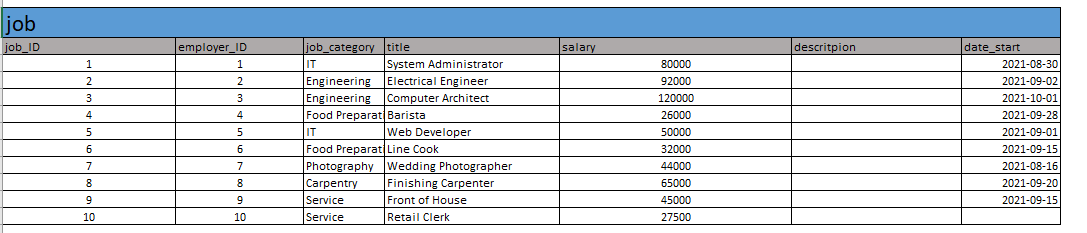


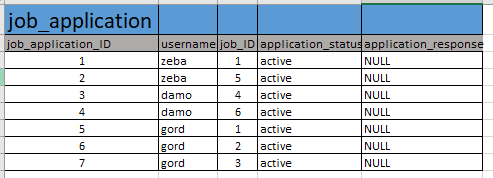
Foreign Key

Primary Key



This relationship is Many to one as 2 unique user can choose the same category eg User Basic





Foreign Key

**Database Schema**

**Relation:`employer`**

`employer\_ID` INT AUTO\_INCREMENT,

`name` VARCHAR(255),

`address` VARCHAR(255),

`phone` VARCHAR(255),

`email` VARCHAR(255),

PRIMARY KEY (employer\_ID)

**Relation:`user\_category`**

`user\_category` VARCHAR(255),

`price` INT,

PRIMARY KEY (user\_category)

**Relation: `user`**

`username` VARCHAR(255),

`employer\_ID` INT NULL,

`user\_category` VARCHAR(255),

`first\_name` VARCHAR(255),

`last\_name` VARCHAR(255),

`email` VARCHAR(255),

`password` VARCHAR(255),

`security\_answer` VARCHAR(255),

`total\_jobs\_posted` INT,

`total\_applications\_submitted` INT,

`status` VARCHAR(255), -- active, inactive

PRIMARY KEY (username),

FOREIGN KEY (employer\_ID) REFERENCES employer (employer\_ID),

FOREIGN KEY (user\_category) REFERENCES user\_category (user\_category)

**Relation: `payment\_method`**

`payment\_method` VARCHAR(255), -- Chequing, Credit

PRIMARY KEY (payment\_method)

**Relation: `payment\_account`**

`payment\_account\_ID` INT AUTO\_INCREMENT,

`username` VARCHAR(255),

`cardholder\_name` VARCHAR(255),

`card\_number` VARCHAR(255),

`expiration\_date` DATE,

`payment\_method` VARCHAR(255), -- Chequing, Credit

`withdrawal\_type` VARCHAR(255), -- Manual, Automatic

`balance` INT,

`status` VARCHAR(255), -- Settled, Frozen

PRIMARY KEY (payment\_account\_ID),

FOREIGN KEY (username) REFERENCES user (username),

FOREIGN KEY (payment\_method) REFERENCES payment\_method   
 (payment\_method)

**Relation: `payment`**

`payment\_ID` INT AUTO\_INCREMENT,

`payment\_account\_ID` INT,

`user\_category` VARCHAR(255),

`total` INT,

PRIMARY KEY (payment\_ID),

FOREIGN KEY (payment\_account\_ID) REFERENCES payment\_account   
(payment\_account\_ID),

FOREIGN KEY (user\_category) REFERENCES user\_category (user\_category)

**Relation: `job`**

`job\_ID` INT AUTO\_INCREMENT,

`employer\_ID` INT,

`job\_category` VARCHAR(255),

`title` VARCHAR(255),

`salary` INT,

`description` VARCHAR(2000), -- much larger text

`date\_start` DATE,

PRIMARY KEY (job\_ID),

FOREIGN KEY (employer\_ID) REFERENCES employer (employer\_ID)

**Relation: `job\_application`**

`job\_application\_ID` INT AUTO\_INCREMENT,

`username` VARCHAR(255),

`job\_ID` INT,

`job\_name` VARCHAR(255),

`employer\_ID` INT,

`employer\_name` VARCHAR(255),

`application\_text` VARCHAR(1000),

`application\_status` VARCHAR(255), -- active, inactive, accepted, rejected

`application\_response` VARCHAR(500),

PRIMARY KEY (job\_application\_ID),

FOREIGN KEY (username) REFERENCES user (username),

FOREIGN KEY (job\_ID) REFERENCES job (job\_ID),

FOREIGN KEY (employer\_ID) REFERENCES employer (employer\_ID)

**Contributions**

Nicholas:

* HTML
* CSS
* PHP
* SQL / SQL Queries
* ER Diagram
* Conceptual Schema
* Database Design
* Documentation
* Project Report

Fady:

* SQL Queries
* Normalization
* Project Report
* Functional Testing
* Minor Features / Touch-ups