

Faculty of Engineering and Computer Science

Expectations of Originality

This form has been created to ensure that all students in the Faculty of Engineering and Computer Science comply with principles of academic integrity prior to submitting coursework to their instructors for evaluation: namely reports, assignments, lab reports and/or software. All students should become familiar with the University's Code of Conduct (Academic) located at http://web2.concordia.ca/Legal_Counsel/policies/english/AC/Code.html

Please read the back of this document carefully before completing the section below. This form must be attached to the front of all coursework submitted to instructors in the Faculty of Engineering and Computer Science.

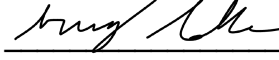
Group Account: gxc353_1


Course Number: COMP 353 **Instructor:** Khaled Jababo


Type of Submission (Please check off responses to both a & b)


- a. ☒ Report ☐ Assignment ☐ Lab Report ☐ Software
- b. ☐ Individual submission ☒ Group Submission (All members of the team must sign below)

Having read both sides of this form, I certify that I/we have conformed to the Faculty's expectations of originality and standards of academic integrity.

Name: Arunraj Adlee ID No: 40059206 Signature:  Date: 2020-08-08
(please print clearly)

Name: Gordon Pham-Nguyen ID No: 40018402 Signature:  Date: 2020-08-08
(please print clearly)

Name: Leo Jr Silao ID No: 40056822 Signature:  Date: 2020-08-08
(please print clearly)

Name: Tiffany Zeng ID No: 40063115 Signature:  Date: 2020-08-08
(please print clearly)

Name: _____ ID No: _____ Signature: _____ Date: _____
(please print clearly)

Name: _____ ID No: _____ Signature: _____ Date: _____
(please print clearly)

Do Not Write in this Space – Reserved for Instructor

EXPECTATIONS OF ORIGINALITY & STANDARDS OF ACADEMIC INTEGRITY

ALL SUBMISSIONS must meet the following requirements:

1. The decision on whether a submission is a group or individual submission is determined by the instructor. Individual submissions are done alone and should not be identical to the submission made by any other student. In the case of group submissions, all individuals in the group must be listed on and must sign this form prior to its submission to the instructor.
2. All individual and group submissions constitute original work by the individual(s) signing this form.
3. Direct quotations make up a very small proportion of the text, i.e., not exceeding 5% of the word count.
4. Material paraphrased from a source (e.g., print sources, multimedia sources, web-based sources, course notes or personal interviews) has been identified by a numerical reference citation.
5. All of the sources consulted and/or included in the report have been listed in the Reference section of the document.
6. All drawings, diagrams, photos, maps or other visual items derived from other sources have been identified by numerical reference citations in the caption.
7. No part of the document has been submitted for any other course.
8. Any exception to these requirements are indicated on an attached page for the instructor's review.

REPORTS and ASSIGNMENTS must also meet the following additional requirements:

1. A report or assignment consists entirely of ideas, observations, information and conclusions composed by the student(s), except for statements contained within quotation marks and attributed to the best of the student's/students' knowledge to their proper source in footnotes or references.
2. An assignment may not use solutions to assignments of other past or present students/instructors of this course or of any other course.
3. The document has not been revised or edited by another student who is not an author.
4. For reports, the guidelines found in Form and Style, by Patrick MacDonagh and Jack Borden (Fourth Edition: May 2000, available at <http://www.encs.concordia.ca/scs/Forms/Form&Style.pdf>) have been used for this submission.

LAB REPORTS must also meet the following requirements:

1. The data in a lab report represents the results of the experimental work by the student(s), derived only from the experiment itself. There are no additions or modifications derived from any outside source.
2. In preparing and completing the attached lab report, the labs of other past or present students of this course or any other course have not been consulted, used, copied, paraphrased or relied upon in any manner whatsoever.

SOFTWARE must also meet the following requirements:

1. The software represents independent work of the student(s).
2. No other past or present student work (in this course or any other course) has been used in writing this software, except as explicitly documented.
3. The software consists entirely of code written by the undersigned, except for the use of functions and libraries in the public domain, all of which have been documented on an attached page.
4. No part of the software has been used in previous submissions except as identified in the documentation.
5. The documentation of the software includes a reference to any component that the student(s) did not write.
6. All of the sources consulted while writing this code are listed in the documentation.

<i>Important: Should you require clarification on any of the above items please contact your instructor.</i>

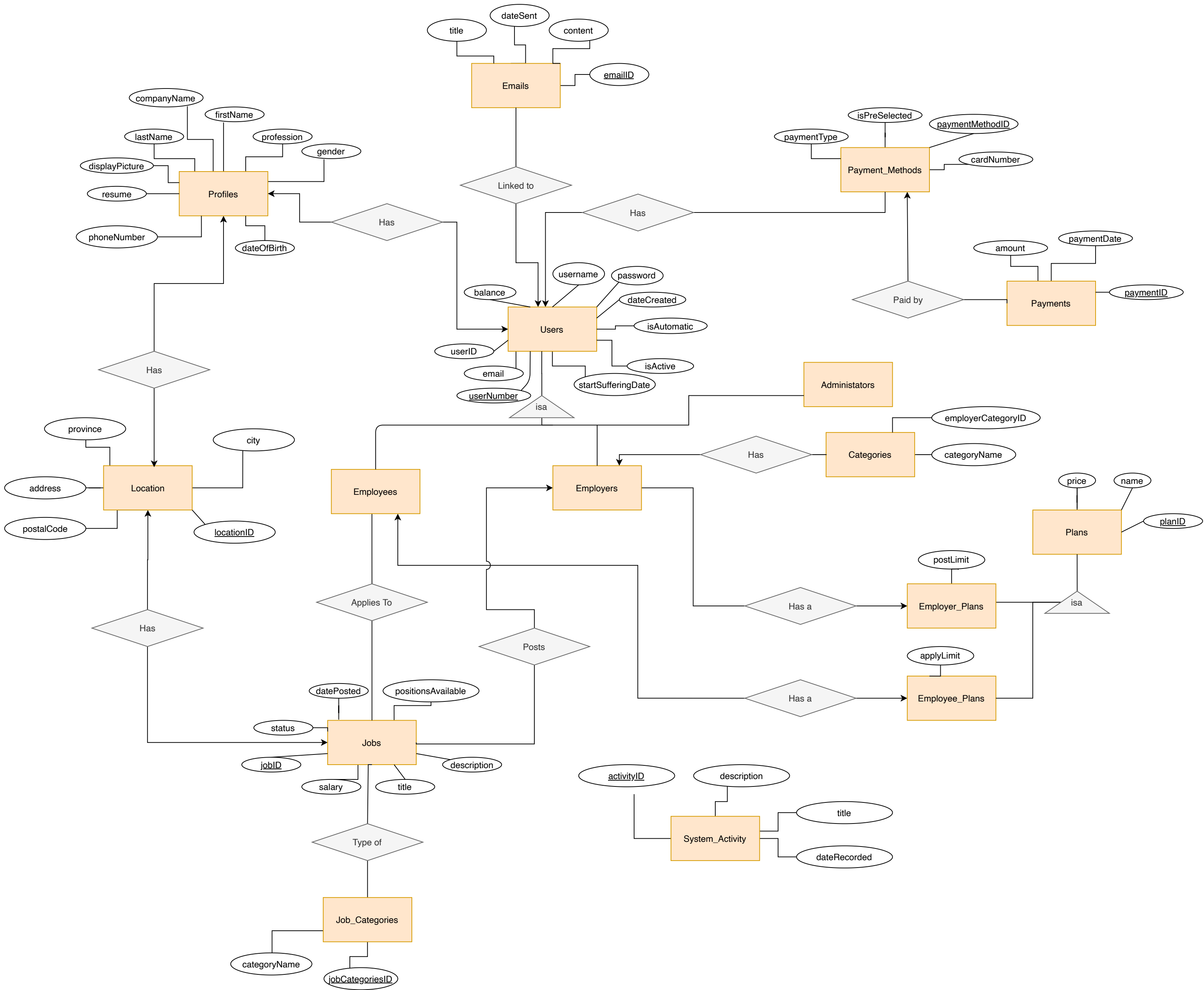
COMP 353: Databases - Summer 2020

Main Project

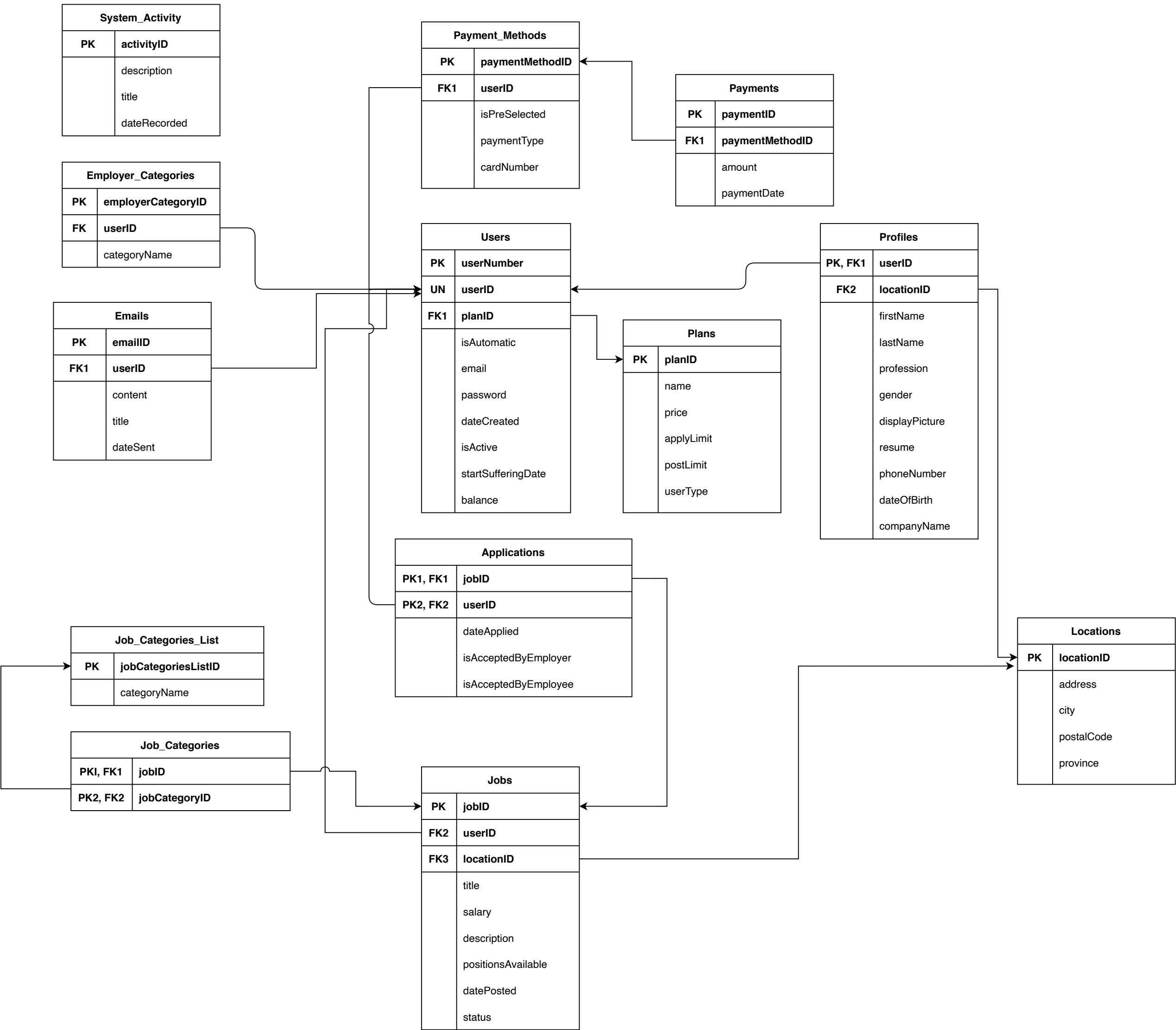
** Important notes from our team

1. We have decided to rename **Categories** to **Plans** to denote the Basic/Prime/Gold pricing as it made more sense to us and creates less confusion with Job Categories.
2. We interpreted **Employer Category** as the field of work that a certain employer works in. e.g., construction, software development, marketing, etc.
3. We decided to treat **Job Categories** as tags (like Instagram # tags), meaning an employer can choose any category that the job posting will belong in by typing any string he/she desires when they are creating a new job posting.
4. We have decided to make **City** dependent on postalCode.
5. We interpreted that **Administrators** do not have payment methods, as their function in the structure is to maintain users and oversee the system as a whole, and therefore do not have any balance associated to their accounts.
6. We have concluded that **Employers** can create a new job posting, but cannot delete nor edit.
7. The following requirements: "[**Users**] **Should be able to maintain new users and update the user table.**" is interpreted as employees can create or delete their accounts and are able to update their account information.
8. The following requirement: "**Employer dashboard should have a contact us section to help user with contact information/helpline.**" is interpreted as displaying the contact information of the Employer whom have created the job posting (it is displayed as a button; onClick will display their contact info) in the Employee's Jobs' postings table.
9. Attached in this report is the E/R Diagram, Relational Database Schema and its Normalization, and the SQL declarations of the relations. See the Implementation Code, Relation Instances, SQL scripts for the queries and transactions and the Sample Data in the attached ZIP file.

E/R Diagram



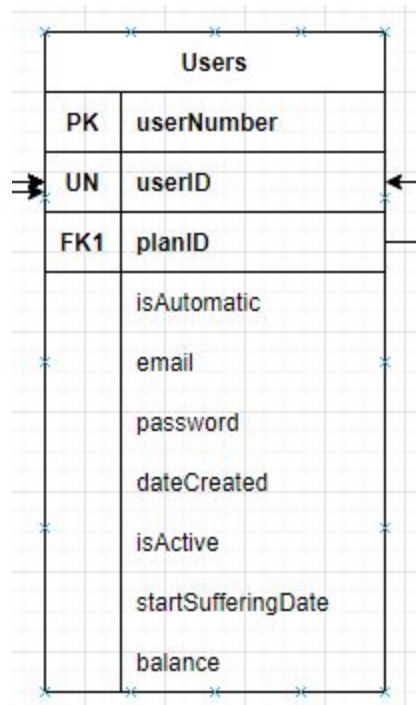
Relational Database Schema



Normalization Process Details

1. Users Table

Schema



Sample Data

<u>userNumber</u>	userID	<u>planID</u> (FK)	email	password	dateCreated	isActive	startSufferingDate	balance	isAutomatic
1	mike	3	mike@..	123	12/12/12	0	12/12/19	-10.00	true
2	doc	4	doc@..	324	12/11/20	1	NULL	33.00	false
3	car	5	car@..	6513	12/23/21	1	NULL	44.00	false

1NF: This table is in 1NF because all the columns hold atomic values.

2NF: This table is in 2NF because it is in 1NF and there are no partial dependencies.

3NF: This table is in 3NF because it is in 2NF and no non-prime is functionally dependent on a field that is not part of the candidate key.

2. Plans Table

Schema

Plans	
PK	planID
	name
	price
	applyLimit
	postLimit
	userType

Sample Data

<u>planID</u>	name	price	applyLimit	postLimit	userType
1	Prime	12.00	0	5	admin
2	Gold	16.00	NULL	0	employee
3	Special	17.00	0	NULL	employer
4	Prime	17.00	0	5	employer

1NF: This table is in 1NF because all the columns hold atomic values.

2NF: This table is in 2NF because it is in 1NF and there are no partial dependencies.

3NF: This table is in 3NF because it is in 2NF and no non-prime is functionally dependent on a field that is not part of the candidate key.

3. Profiles Table

Schema

Profiles	
PK, FK1	userID
FK2	locationID
	firstName
	lastName
	profession
	gender
	displayPicture
	resume
	phoneNumber
	dateOfBirth
	companyName

Sample Data

<u>userID</u>	<u>locationID</u> (FK)	firstName	lastName	profession	gender	displayPicture	resume	phoneNumber	dateOfBirth	companyName
gordon	1	Arunraj	Adlee	Doctor	m	pic.jpg	cv.pdf	514	20/24/88	Montreal Medical
alice	2	Leo	Silao	Engineer	m	pic2.jpg	cv2.pdf	231	12/01/15	Google
tom	1	Jon	Doe	Engineer	f	pic3.jpg	cv3.pdf	123	12/12/96	Amazon
michael	2	Mike	Conway	Lawyer	f	pic4.jpg	cv4.pdf	4455	12/12/20	LawyerGang

1NF: This table is in 1NF because all the columns hold atomic values.

2NF: This table is in 2NF because it is in 1NF and there are no partial dependencies.

3NF: This table is in 3NF because it is in 2NF and there are no transitive dependencies as all columns depend only on the userID key.

4. Locations Table

Schema

Locations	
PK	locationID
	address
	city
	postalCode
	province

Sample Data

locationID	address	city	postalCode	province
1	1095 Dog House	Montreal	H2M 1F8	quebec
2	23123 Park Street	Altoona	35952	Alabama

1NF: This table is in 1NF because all the columns hold atomic values.

2NF: This table is in 2NF because it is in 1NF and there are no partial dependencies.

3NF: This table is in 3NF because it is in 2NF and there are no transitive dependencies as all columns depend only on the locationID key.

We thought about the idea that postalCode could determine province and city, but decided against it as online research showed that it was possible that different countries might share similar zip codes.

5. Jobs Table

Original Schema

Jobs	
PK	jobID
FK2	userID
FK3	locationID
	title
	salary
	description
	companyName
	positionsAvailable
	datePosted
	status

Original Sample Data

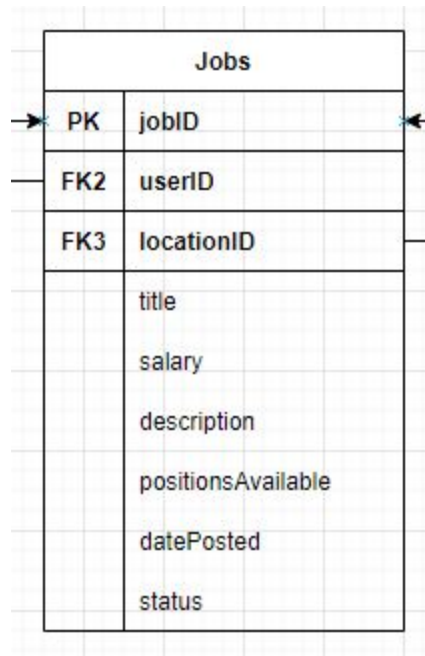
<u>jobID</u>	<u>userID (FK)</u>	<u>locationID (FK)</u>	title	salary	description	company Name	positions Available	datePosted	status
1	bob	1	Software Dev	84000	Angular	Amazon	5	12/20/19	Filled
2	mike	2	Front End Dev	35000	React	Facebook	1	11/3/2020	Open
3	mike	2	Back End Dev	50000	C#	Facebook	1	11/4/2020	Open

1NF: This table is in 1NF because all the columns hold atomic values.

2NF: This table is in 2NF because it is in 1NF and there are no partial dependencies.

3NF: This table is in **NOT** 3NF because we found that the companyName field was reliant on the userID (employer) FK. So we decided to move the companyName field to the Profile entity.

New Schema



New Sample Data

<u>jobID</u>	<u>userID (FK)</u>	<u>locationID (FK)</u>	title	salary	description	positions Available	datePosted	status
1	bob	1	Software Dev	84000	Angular	5	12/20/19	Filled
2	mike	2	Front End Dev	35000	React	1	11/3/2020	Open
3	mike	2	Back End Dev	50000	C#	1	11/3/2020	Open

6. Job_Categories_List Table

Schema

Job_Categories_List	
PK	jobCategoriesListID
	categoryName

Sample Data

<u>jobCategoriesListID</u>	categoryName
1	Javascript
2	React
3	Angular
4	PHP

1NF: This table is in 1NF because all the columns hold atomic values.

2NF: This table is in 2NF because it is in 1NF and there are no partial dependencies.

3NF: This table is in 3NF because it is in 2NF and there are no transitive dependencies as all columns depend only on the unique jobCategoriesListID key.

7. Job_Categories Table

Schema

Job_Categories	
PK1, FK1	jobID
PK2, FK2	jobCategoryID

Sample Data

jobID	jobCategoryID
1	2
2	1
3	3
4	1

1NF: This table is in 1NF because all the columns hold atomic values.

2NF: This table is in 2NF because it is in 1NF and there are no partial dependencies.

3NF: This table is in 3NF because it is in 2NF and there are no transitive dependencies since columns are FKs and PKs in order to connect the job and its categories.

8. Applications Table

Schema

Applications	
PK1, FK1	jobID
PK2, FK2	userID
	dateApplied
	isAcceptedByEmployer
	isAcceptedByEmployee

Sample Data

<u>jobID</u>	<u>userID</u>	dateApplied	isAcceptedByEmployer	isAcceptedByEmployee
1	mike	3/5/2020	False	False
2	jon	2/7/2020	True	True
3	gordon	4/5/2020	True	False
1	chris	4/9/2020	True	True

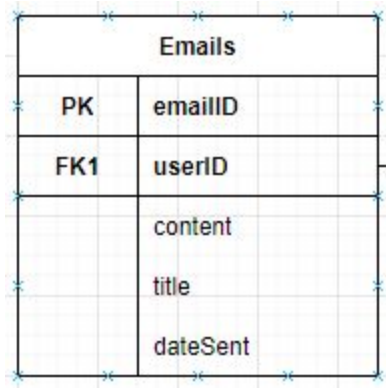
1NF: This table is in 1NF because all the columns hold atomic values.

2NF: This table is in 2NF because it is in 1NF and there are no partial dependencies.

3NF: This table is in 3NF because it is in 2NF and there are no transitive dependencies as all non-prime columns depend on both the userID and jobID both are required to uniquely identify the application.

9. Emails Table

Schema



Sample Data

emailID	userID (FK)	content	title	dateSent
1	bob	Hello World	Forgot Password	3/5/2020
2	alex	Hello World	Forgot Password	2/7/2020
3	gordon	Hello World	Forgot Password	4/5/2020
4	leo	Hello World	Forgot Password	4/9/2020

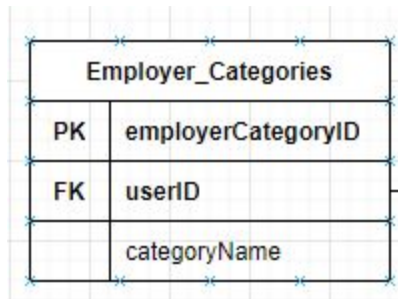
1NF: This table is in 1NF because all the columns hold atomic values.

2NF: This table is in 2NF because it is in 1NF and there are no partial dependencies.

3NF: This table is in 3NF because it is in 2NF and there are no transitive dependencies as all columns depend only on the unique, auto incrementing, emailID key.

10. Employer_Categories Table

Schema



Sample Data

<u>employerCategoryID</u>	<u>userID (FK)</u>	categoryName
1	bob	Senior HR Manager
2	alex	Tech Lead
3	gordon	Junior HR
1	leo	Project Manager

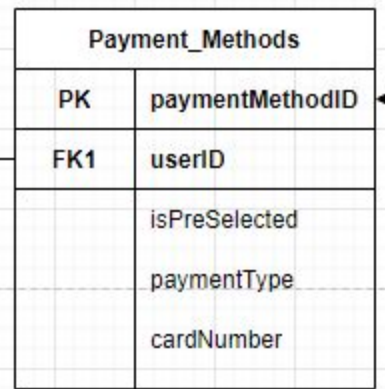
1NF: This table is in 1NF because all the columns hold atomic values.

2NF: This table is in 2NF because it is in 1NF and there are no partial dependencies.

3NF: This table is in 3NF because it is in 2NF and there are no transitive dependencies as all non-prime columns depend on only the employerCategoryID key.

11. Payment_MethodsTable

Schema



Sample Data

<u>paymentMethodID</u>	<u>userID (FK)</u>	isPreSelected	paymentType	cardNumber
1	bob	True	Credit Card	2846*****
2	gordon	False	Checking Account	1561*****
3	tiffany	False	Credit Card	5511*****
1	bob	True	Credit Card	3334*****

1NF: This table is in 1NF because all the columns hold atomic values.

2NF: This table is in 2NF because it is in 1NF and there are no partial dependencies.

3NF: This table is in 3NF because it is in 2NF and there are no transitive dependencies as all non-prime columns depend only on the auto-incrementing paymentMethodID key

12. PaymentsTable

Schema

Payments	
PK	paymentID
FK1	paymentMethodID
	amount
	paymentDate

Sample Data

paymentID	PaymentMethodID(FK)	amount	paymentDate
1	1	10.00	12/12/20
2	1	100.00	11/08/19
3	2	50.00	11/06/19
1	3	20.00	11/08/18

1NF: This table is in 1NF because all the columns hold atomic values.

2NF: This table is in 2NF because it is in 1NF and there are no partial dependencies.

3NF: This table is in 3NF because it is in 2NF and there are no transitive dependencies as all non-prime columns depend only on the auto-incrementing paymentID key

13. System_Activity Table

Schema

System_Activity	
PK	activityID
	description
	title
	dateRecorded

Sample Data

<u>activityID</u>	description	title	dateRecorded
1	Job added a new job	Job added	12/12/20
2	Bob added a new job	Job added	11/08/19
3	Gordon applied to a job	Application added	11/06/19
1	Added new application	Hello world	11/08/18

1NF: This table is in 1NF because all the columns hold atomic values.

2NF: This table is in 2NF because it is in 1NF and there are no partial dependencies.

3NF: This table is in 3NF because it is in 2NF and there are no transitive dependencies as all non-prime columns depend only on the auto-incrementing activityID key

SQL Queries

1. Create/Delete/Edit/Display an Employer.

From the Plans table, we see that the plan with ID 5 has userType 'employer'

	planID	name	price	applyLimit	postLimit	userType
▶	1	Employee Basic	0	0	0	employee
	2	Employee Prime	10	5	0	employee
	3	Employee Gold	20	NULL	0	employee
	4	Employer Prime	50	0	5	employer
	5	Employer Gold	100	0	NULL	employer
	6	Admin	0	0	0	admin
*	NULL	NULL	NULL	NULL	NULL	NULL

CREATE

```
INSERT INTO Users (userID,  
    planID,  
    email,  
    password,  
    isActive,  
    balance,  
    isAutomatic)  
VALUES ('bob',  
    4,  
    'bob@comp353.com',  
    'bob',  
    TRUE,  
    0,  
    TRUE);
```

	userNumber	userID	planID	email	password	dateCreated	isActive	startSufferingDate	balance	isAutomatic
▶	6	bob	5	bob@comp353.com	bob	2020-05-15 00:00:00	1	NULL	0	0

EDIT

```
UPDATE Users  
SET isActive = 0  
WHERE userNumber=6;
```

DISPLAY

```
SELECT * FROM Users  
WHERE userNumber=6;
```

DELETE

```
DELETE FROM Users
WHERE userNumber=6;
```

Note: This delete will cascade and delete rows from all the other tables where the userID 'bob' is a FK.

2. Create/Delete/Edit/Display a category by an Employer.

Create

```
INSERT INTO Employer_Categories (userID, categoryName)
VALUES ('alice', 'Software Engineer'),
      ('alice', 'Tech Lead');
```

EDIT

```
UPDATE Employer_Categories
SET categoryName = 'Project Manager'
WHERE categoryName = 'Tech Lead'
AND userID = 'alice'
```

DISPLAY

```
SELECT * FROM Employer_Categories
WHERE categoryName = 'Project Manager'
AND userID = 'alice'
```

DELETE

```
DELETE FROM Employer_Categories
WHERE categoryName = 'Project Manager'
AND userID = 'alice'
```

3. Post a new job by an employer.

```
INSERT INTO Jobs
(userID, locationID, title, salary, description, positionsAvailable, status)
VALUES ('bob', 2, 'Human Resources', 120000, 'Must have 20 years of experience at any
company', 1, 'active');
```

	jobID	userID	locationID	title	salary	description	positionsAvailable	datePosted	status
▶	2	bob	2	Human Resources	120000	Must have 20 years experience at any company	1	2020-08-02 00:00:00	active

4. Provide a job offer for an employee by an employer.

Applications table before (user has applied to a job):

	jobID	userID	dateApplied	isAcceptedByEmployer	isAcceptedByEmployee
▶	3	alice	2020-08-01 19:31:17	NULL	NULL

Query:

```
UPDATE Applications SET isAcceptedByEmployer = 1 WHERE userID = 'alice' AND jobID = 3;
```

Application after query (employer has accepted the application and offered a job):

	jobID	userID	dateApplied	isAcceptedByEmployer	isAcceptedByEmployee
▶	3	alice	2020-08-01 19:31:17	1	NULL

5. Report of a posted job by an employer (Job title and description, date posted, list of employees applied to the job and status of each application).

```
SELECT j.title, j.description, j.datePosted, IFNULL(a.userID, 'No Applications') AS 'username',  
IFNULL(a.isAcceptedByEmployer, 'N/A') AS 'isAcceptedByEmployer',  
IFNULL(a.isAcceptedByEmployee, 'N/A') AS 'isAcceptedByEmployee'  
FROM Jobs AS j  
LEFT JOIN Applications AS a ON j.jobID = a.jobID  
WHERE j.jobID = [input.jobID]
```

	title	description	datePosted	username	isAcceptedByEmployer	isAcceptedByEmployee
▶	Human Resources	Must have 20 years experience at any company	2020-08-02 00:00:00	tiffany	N/A	N/A
	Human Resources	Must have 20 years experience at any company	2020-08-02 00:00:00	gordon	N/A	N/A
	Human Resources	Must have 20 years experience at any company	2020-08-02 00:00:00	simba	N/A	N/A
	Human Resources	Must have 20 years experience at any company	2020-08-02 00:00:00	arun	N/A	N/A
	Human Resources	Must have 20 years experience at any company	2020-08-02 00:00:00	tyson	1	1

6. Report of posted jobs by an employer during a specific period of time (Job title, date posted, short description of the job up to 50 characters, number of needed employees to the post, number of applied jobs to the post, number of accepted offers).

```
SELECT j.title, SUBSTRING(j.description, 1, 50) AS 'description', j.positionsAvailable,  
j.datePosted, COUNT(a.jobID) AS 'Number of employees applied',  
SUM(IFNULL(a.isAcceptedByEmployee, 0)) AS 'Number of accepted offers'  
FROM Jobs AS j  
LEFT JOIN Applications AS a ON j.jobID = a.jobID  
WHERE j.userID = [input_userID] AND  
j.datePosted BETWEEN 'YYYY-MM-DD' AND 'YYYY-MM-DD'  
GROUP BY j.jobID;
```

title	description	positionsAvailable	datePosted	Number of employees applied	Number of accepted offers
Python Developer	Must know snakes	5	2020-08-01 00:00:00	1	0
Java Developer	Must like coffee	2	2020-08-02 00:00:00	1	0
JavaScript Developer	Must know all frameworks	8	2020-08-03 00:00:00	0	0
Rust Developer	We will ask about metals	3	2020-08-04 00:00:00	0	0
Software Developer	Work on Angular apps	4	2020-08-06 21:40:38	0	0

7. Create/Delete/Edit/Display an Employee.

From the Plans table, we see that the plan with ID 2 has userType 'employee'

	planID	name	price	applyLimit	postLimit	userType
▶	1	Employee Basic	0	0	0	employee
	2	Employee Prime	10	5	0	employee
	3	Employee Gold	20	NULL	0	employee
	4	Employer Prime	50	0	5	employer
	5	Employer Gold	100	0	NULL	employer
	6	Admin	0	0	0	admin
*	NULL	NULL	NULL	NULL	NULL	NULL

CREATE

```
INSERT INTO Users (userID,  
                  planID,  
                  email,  
                  password,  
                  isActive,  
                  balance,  
                  isAutomatic)  
VALUES ('gordon',  
        2,  
        'gordon@comp353.com',  
        'gordon',  
        TRUE,  
        0,  
        TRUE)
```

userNumber	userID	planID	email	password	dateCreated	isActive	startSufferingDate	balance	isAutomatic
1	gordon	2	gordon@comp353.com	gordon	2020-01-14 23:00:00	1	NULL	0	1

EDIT

```
UPDATE Users  
SET isActive = 0  
WHERE userNumber = 1;
```

DISPLAY

```
SELECT * FROM Users  
WHERE userNumber = 1;
```


DELETE

```
DELETE FROM Users  
WHERE userNumber = 1;
```

Note: This delete will cascade and delete rows from all the other tables where the userID 'gordon' is a FK.

8. Search for a job by an employee

```
SELECT title, datePosted, description, companyName, city, salary, positionsAvailable, status  
FROM Jobs  
JOIN Location ON Jobs.locationID = Location.locationID  
JOIN Profiles ON Jobs.userID = Profiles.userID  
JOIN Users ON Users.userID = Jobs.userID  
JOIN Employer_Categories EC ON Jobs.userID = EC.userID  
WHERE LOWER(title) LIKE LOWER('%Developer%')  
OR LOWER(description) LIKE LOWER('%Developer%')  
OR LOWER(Profiles.companyName) LIKE LOWER('%Developer%')  
OR LOWER(Profiles.firstName) LIKE LOWER('%Developer%')  
OR LOWER(Profiles.lastName) LIKE LOWER('%Developer%')  
OR LOWER(Jobs.userID) LIKE LOWER('%Developer%')  
OR LOWER(Location.city) LIKE LOWER('%Developer%');
```

title	datePosted	description	companyName	city	salary	positionsAvailable	status
Software Developer	2020-07-15 00:00:00	Must know C++	BigMusic	Burlington	90000	3	active
Python Developer	2020-08-01 00:00:00	Must know snakes	Logic Industries	North Pender Island	80000	5	active
Java Developer	2020-08-02 00:00:00	Must like coffee	Logic Industries	Cobble Hill	60000	2	active
JavaScript Developer	2020-08-03 00:00:00	Must know all frameworks	Logic Industries	Pickering	70000	8	active
Rust Developer	2020-08-04 00:00:00	We will ask about metals	Logic Industries	Ile Perrot	85000	3	active
Software Developer	2020-08-06 21:40:38	Work on Angular apps	Logic Industries	Montreal	100000	4	active
Software Developer	2020-07-15 00:00:00	Must have 15 years of experience in PHP	Siens	Brossard	85000	3	active

9. Apply for a job by an employee.

Existing jobs in the Jobs table:

	jobID	userID	locationID	title	salary	description	positionsAvailable	datePosted	status
▶	1	leo	1	Software Developer	85000	Must have 15 years of experience in PHP	3	2020-07-15 00:00:00	active
	2	bob	2	Human Resources	120000	Must have 20 years experience at any company	1	2020-08-02 00:00:00	active
	3	sujan	3	IT Help Desk	60000	Required Skills: Java, MySQL	1	2020-08-13 00:00:00	expired
	4	jcole	4	Business Analyst	85000	Requirements: Bachelors in Business	5	2020-07-15 00:00:00	expired
	5	jcole	4	Software Developer	90000	Must know C++	3	2020-07-15 00:00:00	active
	6	ariana	5	Database Administrator	60000	Looking for McGill students only	1	2020-07-06 00:00:00	active
	7	joe	6	Python Developer	80000	Must know snakes	5	2020-08-01 00:00:00	active
	8	joe	7	Java Developer	60000	Must like coffee	2	2020-08-02 00:00:00	active
	9	joe	8	JavaScript Developer	70000	Must know all frameworks	8	2020-08-03 00:00:00	active
	10	joe	9	Rust Developer	85000	We will ask about metals	3	2020-08-04 00:00:00	active
	12	joe	11	Software Developer	100000	Work on Angular apps	4	2020-08-06 21:40:38	active
★	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Query:

```
INSERT INTO Applications (jobID, userID) VALUES (2, 'tyson')
```

New Row in Applications table:

	jobID	userID	dateApplied	isAcceptedByEmployer	isAcceptedByEmployee
▶	2	tyson	2020-08-01 17:51:19	NULL	NULL

10. Accept/Deny a job offer by an employee

Existing applications in the Applications table

jobID	userID	dateApplied	isAcceptedByEmployer	isAcceptedByEmployee
3	leo	2020-08-01 19:18:08	1	NULL
2	tyson	2020-08-01 17:51:19	1	NULL

Deny job offer:

UPDATE Applications

SET isAcceptedByEmployee = 0

WHERE jobID = 2 AND userID = 'tyson';

Accept job offer:

UPDATE Applications

SET isAcceptedByEmployee = 1

WHERE jobID = 3 AND userID = 'leo';

Applications after user 'leo' accepted and user 'tyson' denies

jobID	userID	dateApplied	isAcceptedByEmployer	isAcceptedByEmployee
3	leo	2020-08-01 19:18:08	1	1
2	tyson	2020-08-01 17:51:19	1	0

11. Withdraw from an applied job by an employee.

DELETE FROM Applications

WHERE jobID = 1 and userID = 'leo';

12. Delete a profile by an employee.

DELETE FROM Profiles

WHERE userID = 'gordon';

13. Report of applied jobs by an employee during a specific period of time (Job title, date applied, short description of the job up to 50 characters, status of the application).

```
SELECT j.title, SUBSTRING(j.description, 1, 50) AS 'description', a.dateApplied,
a.isAcceptedByEmployer, a.isAcceptedByEmployee
FROM Applications AS a, Jobs as j
WHERE a.userID = [input.userID] AND j.jobID = a.jobID
AND a.dateApplied BETWEEN 'YYYY-MM-DD' AND 'YYYY-MM-DD'
GROUP BY j.jobID;
```

title	description	dateApplied	isAcceptedByEmployer	isAcceptedByEmployee
Software Developer	Must have 15 years of experience in PHP	2020-08-06 15:54:37	1	NULL
Human Resources	Must have 20 years experience at any company	2020-08-06 15:54:37	NULL	NULL
IT Help Desk	Required Skills: Java, mySQL	2020-08-06 15:54:37	NULL	NULL
Business Analyst	Requirements: Bachelors in Business	2020-08-06 15:54:37	NULL	NULL
Software Developer	Must know C++	2020-08-06 15:54:37	NULL	NULL

14. Add/Delete/Edit a method of payment by a user.

```
INSERT INTO Payment_Methods (
    userID,
    isPreSelected,
    cardNumber,
    paymentType)
VALUES ('tyson',
0,
1111111111,
'credit')
```

paymentMethodID	userID	isPreSelected	paymentType	cardNumber
14	tyson	0	credit	123515
NULL	NULL	NULL	NULL	NULL

DELETE

```
DELETE FROM Payment_Methods
WHERE userID = 'tyson' AND paymentMethodID = 14
```

EDIT

```
UPDATE Payment_Methods
SET isPreSelected = 1
WHERE userID = 'tyson' AND paymentMethodID = 14
```

15. Add/Delete/Edit an automatic payment by a user.

From the Users table, we see that this user uses automatic payments

userID	planID	email	password	dateCreated	isActive	startSufferingDate	balance	isAutomatic
gordon	1	gordon@comp353.com	gordon	2020-01-15 00:00:00	1	NULL	0	1

Since the user wants to use automatic payments, they will be required to add at least one payment method where isPreSelected = 1. This pre-selected payment method will be the one used by the system to charge the user on the first of the month.

CREATE

```
INSERT INTO Payment_Methods (paymentMethodID,
                             userID,
                             isPreSelected,
                             cardNumber,
                             paymentType)
VALUES (1,
        'gordon',
        1,
        1234,
        debit)
```

paymentMethodID	userID	isPreSelected	paymentType	cardNumber
1	gordon	1	debit	1234

DELETE

```
DELETE FROM Payment_Methods
WHERE userID = 'gordon' AND paymentMethodID = 1
```

EDIT

```
UPDATE Payment_Methods
SET isPreSelected = 0
WHERE userID = 'gordon' AND paymentMethodID = 1
```

16. Make a manual payment by a user.

From the Users table, we see that this user does not use automatic payments

	userID	planID	email	password	dateCreated	isActive	startSufferingDate	balance	isAutomatic
▶	arun	3	arun@comp353.com	arun	2020-03-15 00:00:00	1	NULL	10	0

In the Payment_Methods table, user 'arun' has the following payment methods:

	paymentMethodID	userID	isPreSelected	cardNumber	paymentType
▶	2	arun	1	24151431	credit card

Now when making a payment, we use the following query:

```
INSERT INTO Payments (paymentMethodID, amount)
VALUES (2, 30);
```

Content in the payments table:

	paymentID	paymentMethodID	amount	paymentDate
▶	1	2	30	2020-08-03 18:25:48

In the PHP code we then calculate the new balance and perform the following to update the balance.

```
UPDATE Users SET balance = :newBalance WHERE userID = :user"
```

17. Report of all users by the administrator for employers or employees (Name, email, category, status, balance).

```
SELECT p.firstName as 'First Name', p.lastName as 'Last Name', u.email as 'Email', pl.name
as 'Plan Name', pl.userType as 'User Type', u.isActive, u.balance as 'Balance'
FROM Users as u, Profiles as p, Plans as pl
WHERE u.planID = pl.planID AND u.userID = p.userID
AND pl.userType <> 'admin'
```

	First Name	Last Name	Email	Plan Name	User Type	isActive	Balance
▶	gordon	ramsay	gordon@comp353.com	Employee Basic	employee	1	0
	tiffany	zeng	tiffany@comp353.com	Employee Prime	employee	1	10
	tyson	chandler	tyson@comp353.com	Employee Prime	employee	1	-10
	leo	silao	leo@comp353.com	Employer Prime	employer	0	0
	bob	thebuilder	bob@comp353.com	Employer Gold	employer	1	-100

18. Report of all outstanding balance accounts (User name, email, balance, since when the account is suffering).

```
SELECT userID, email, balance, startSufferingDate
FROM Users
WHERE balance < 0;
```

userID	email	balance	startSufferingDate
leo	leo@comp353.com	-50	2020-08-06 22:40:38
khaled	khaled@comp353.com	-50	2019-06-15 00:00:00
sujan	sujan@comp353.com	-50	2020-08-06 22:42:54
jcole	jcole@comp353.com	-50	2020-08-06 22:42:55
tyson	tyson@comp353.com	-10	2020-03-15 00:00:00