

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.


Course Number: _____ **Instructor:** _____

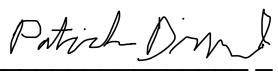
Type of Submission (Please check off responses to both a & b)


- 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: Adnan Ali ID No: 40181614 Signature:  Date: 08/15/2021
(please print clearly)

Name: Daejung Bae ID No: 40013623 Signature:  Date: 08/15/2021
(please print clearly)

Name: Patrick Drummond ID No: 40185198 Signature:  Date: 08/15/2021
(please print clearly)

Name: Seyedsina Mirmaghferaty ID No: 40124936 Signature:  Date: 08/15/2021
(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.

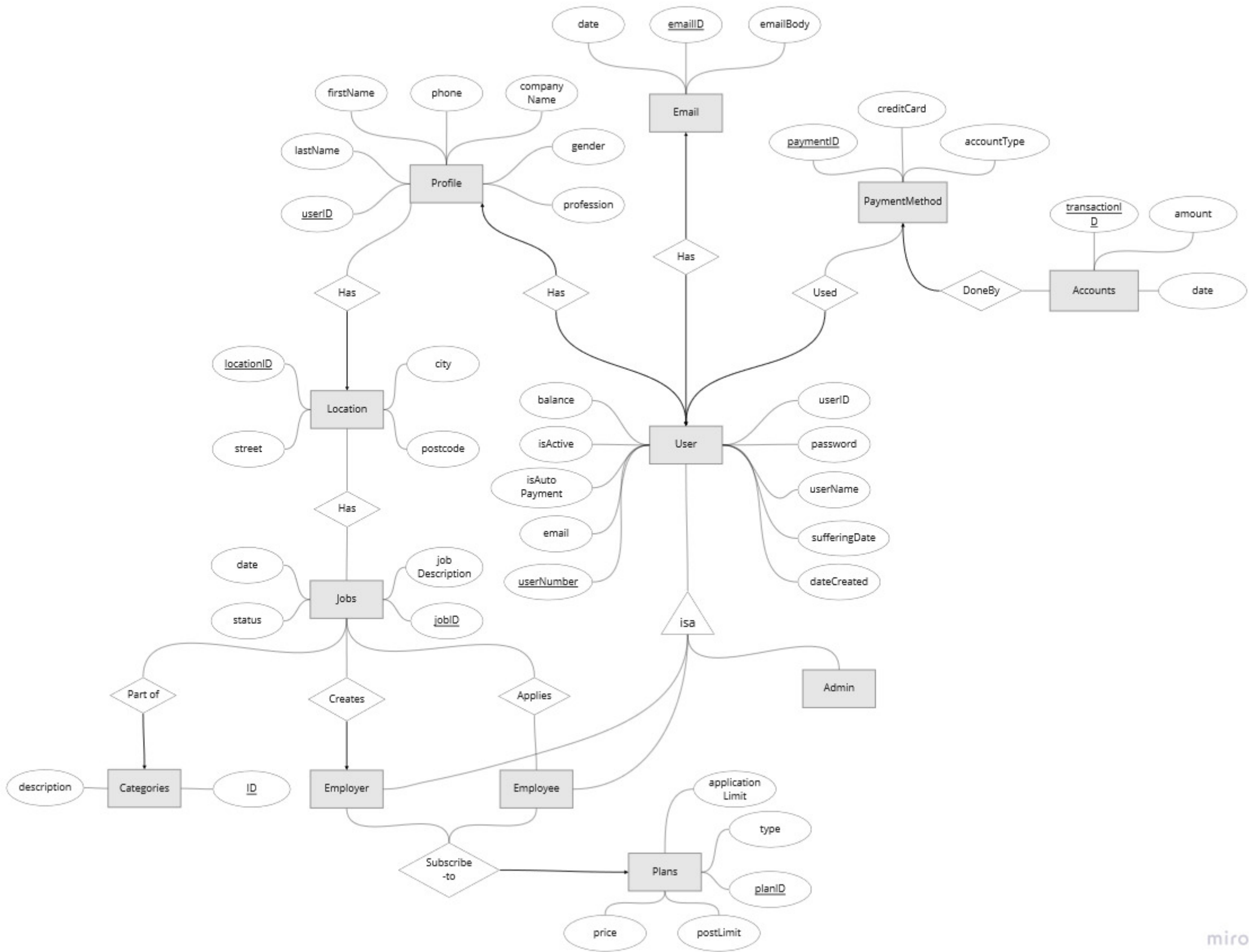
LAB REPORTS must also meet the following requirements:

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.

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



Contributions

Name	Contributions
Adnan Ali	E/R Diagram, Database, Report
Daejung Bae	E/R Diagram, Report
Patrick Drummond	PHP Code, Testing
Seyedsina Mirgmaghferaty	PHP Code, Testing and connecting to DB

Assumptions

Employer_Category table denotes an area of expertise like Entertainment industry, Food industry, Tech industry etc.

Job_Categories table denotes a subdivision of the category where the employer has posted, example. Singer and dancer are a sub-category of the Entertainment industry.

The **Administrator** in the **Subscription_Plans** table is supposed to be an overseer of the system, and even though they do not have any balance attached to them, they can access the database whenever and change their limitations.

Entity-Relationship Diagram

Relational-Database-Schema

- Subscription_Plans (plan_id, name, price, apply_limit, post_limit, user_type)
- Location (location_id, address, city, post_code)
- Users (user_number, user_id, plan_id, email, password, date_created, active_status, suffering_date)
- User_Profiles (user_id, location_id, company_name, name, profession, gender, phone)
- Jobs (job_id, user_id, location_id, job_title, salary, job_description, positions_available, date_posted, job_status)
- Payment_Method (method_id, user_id, is_preselected, payment_type, card_number)
- Transactions (transaction_id, method_id, payment_amount, payment_date)
- Job_Applications (job_id, user_id, date_applied, is_acceptedByEmployer, is_acceptedByEmployee)
- Emails (email_id, user_id, email_body, send_date)
- Employer_Categories (category_id, user_id, name, description)
- Job_Category_List (job_category_id, category_name)
- Job_Category (job_id, job_category_id)

Database-Schema-Normalization

As per requirements, our database was normalized to 3NF.

Keys:

Primary Key (Pk)

Foreign Key (Fk)

Atomic Attribute

Schema:

Subscription_Plans (plan_id, name, price, apply_limit, post_limit, user_type)

- Each column in this tuple holds an atomic value (**1NF**). There are no partial dependencies in this tuple (**2NF**).
- There are no non-prime attributes that are functionally dependent on non-candidate keys making this table **3NF**.

Location (location_id, address, city, post_code)

- Each column in this tuple holds an atomic value (**1NF**). There are no partial dependencies in this tuple (**2NF**).
- There are no non-prime attributes that are functionally dependent on non-candidate keys making this table **3NF**.
- The reason one of the attributes is “address” and not “street address” is because, in an edge case where 2 users are located in the same building, we can include Suite# in there as well.

Users (user_number, user_id, plan_id, email, password, date_created, active_status, suffering_date)

- Each column in this tuple holds an atomic value (**1NF**). There are no partial dependencies in this tuple (**2NF**).
- All attributes are dependent on user_number (which is part of the candidate key), therefore all attributes are functionally dependent on prime members making it **3NF**.

User_Profiles (user_id, location_id, company_name, name, profession, gender, phone)

- Each column in this tuple holds an atomic value (**1NF**). There are no partial dependencies in this tuple (**2NF**).
- All attributes are dependent on user_id (which is **Pk** and **Fk**), therefore all attributes are functionally dependent on prime members making it **3NF**.

Payment_Method (method_id, user_id, is_preselected, payment_type, card_number)

- Each column in this tuple holds an atomic value (**1NF**). There are no partial dependencies in this tuple (**2NF**).
- All attributes are dependent on solely the method_id (which is auto incrementing in this case), which makes it **3NF**.

Transactions (transaction_id, method_id, payment_amount, payment_date)

- Each column in this tuple holds an atomic value (**1NF**). There are no partial dependencies in this tuple (**2NF**).
- All attributes are dependent on solely the transaction_id (which is auto incrementing in this case), which makes it **3NF**.

Job_Applications (job_id, user_id, date_applied, is_acceptedByEmployer, is_acceptedByEmployee)

- Each column in this tuple holds an atomic value (**1NF**). There are no partial dependencies in this tuple (**2NF**).
- Even though job_id and user_id are both **Pk** and **Fk**, there are no other transitive dependencies as all non-prime attributes depend on both **Pk**'s to identify a job application, making it **3NF**.

Emails (email_id, user_id, email_body, send_date)

- Each column in this tuple holds an atomic value (**1NF**). There are no partial dependencies in this tuple (**2NF**).
- All attributes are dependent on solely the email_id (which is auto incrementing in this case), which makes it **3NF**.

Employer_Categories (**category_id**, **user_id**, name, description)

- Each column in this tuple holds an atomic value (**1NF**). There are no partial dependencies in this tuple (**2NF**).
- All attributes are dependent on solely the **category_id** (which is auto incrementing in this case), which makes it **3NF**.

Jobs (**job_id**, **user_id**, **location_id**, job_title, salary, job_description, positions_available, date_posted, job_status)

Job_Category_List (**job_category_id**, category_name)

Job_Category (**job_id**, **job_category_id**)

The three 3 **labelled tables** need to be looked at all together in our current schema.

- Our **Jobs** table does not contain the company_name attribute because we found that if an employer posted more than 1 job, the employer_id (**user_id** in this case) would be the main dependency for the company_name and not the **job_id**. So to make the job table 3NF, we decided to move company_name to the User table.
- This was the same case with the Category mentioned in the requirement section. We initially decided to implement a category_id to each category_name. But each name was also reliant on the job_id being posted.
- So to connect **Job_Category**, we used **job_id** and **category_id** together to connect each job to its category(also used as **Fk** here)
- Finally we gave each **category** a list id and connected it to its name in a 3NF relation.

SQL Queries

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

Create Employer:

```
-- Create Employer
insert into users
(userID, planID, email, passwords, dateCreated, isActive, balance, isAutomatic)
values
("Albert", 4, "Albert@create-user.net", "password", "2021-08-13", FALSE, 0, TRUE);
-- Note: planID (from subscription plans table) determines the user_type = "employer"
```

Result

userNumber	userID	planID	email	passwords	dateCreated	isActive	startSufferingDate	balance	isAutomatic
1	User_1	1	User1@example.net	password	2021-01-15	1		0	1
2	User_2	2	User2@example.net	password	2021-02-15	1		10	0
3	User_3	2	User3@example.net	password	2020-12-15	1		0	1
4	User_4	3	User4@example.net	password	2021-03-15	1		10	0
5	User_5	4	User5@example.net	password	2021-04-15	0		0	1
6	User_6	5	User6@example.net	password	2021-05-15	1		-100	0
7	User_7	6	User_7@example.net	password	2021-06-15	1		0	1
8	User_8	1	User_8@example.net	password	2020-06-15	1		-50	1
9	User_9	4	User_9@example.net	password	2020-06-15	0		50	1
10	User_10	5	User_10@example.net	password	2020-06-15	0		50	1
11	User_11	4	User_11@example.net	password	2020-06-15	0		10	1
12	User_12	4	User_12@example.net	password	2021-01-14	1		0	1
13	tyson	2	tyson@example.net	password	2021-03-15	1		-10	1
14	chris	5	chris@example.net	password	2021-08-07	1		0	1
16	Albert	4	Albert@create-user.net	password	2021-08-13	0		0	1

Edit Employer:

```
-- Edit Employer
update users
set email = "albert@edit-query.com"
where userID = "Albert";
```

Result:

14	chris	5	chris@example.net	password	2021-08-07	1		0	1
18	Albert	4	albert@edit-query.com	password	2021-08-13	0		0	1

Delete Employer:

```
-- Delete Employer
delete from users
where userID = "Albert";
```

Result:

13	tyson	2	tyson@example.net	password	2021-03-15	1		-10	1
14	chris	5	chris@example.net	password	2021-08-07	1		0	1

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

Create a category by an Employer

```
-- Create a category
insert into Employer_Categories
(userID, categoryName)
values
("Albert", "Physicist");
-- Note: "Employer Albert" has be to created from query 1
```

Result:

employerCategoryID	userID	categoryName
1	User_5	Science
2	User_6	Stocks
3	User_9	Design
4	User_10	Music
5	User_11	Music
6	User_12	Software
13	chris	Entertainment
16	Albert	Physicist

Edit a category by Employer

```
-- Edit a category
update Employer_Categories
set categoryName = "Math User"
where userID = "Albert";
```

Result

6	User_12	Software
13	chris	Entertainment
16	Albert	Math User

Delete a category by Employer

```
-- Delete a category
delete from Employer_Categories
where userID = "Albert" and categoryName = "Math User";
```

Result

5	User_11	Music
6	User_12	Software
13	chris	Entertainment

iii. Post a new job by an employer.

```
insert into Jobs
(userID, locationID, title, salary, description, positionsAvailable, status)
values
("User_10", 3, "Main Project Creator", 10000, "Must be able to get 100% on our Project", 1, "Active");
-- Note: "User_10" is an employer already created in the User-Table
```

Result

jobID	userID	locationID	title	salary	description	positionsAvailable	datePosted	status
1	User_5	1	Software Developer	85000	Must have 15 years of experience in PHP	3	2021-07-15	active
2	User_6	2	Human Resources	120000	Must have 20 years experience at any company	1	2021-08-02	active
3	User_9	3	IT Help Desk	60000	Required Skills: Java, mySQL	1	2021-08-13	expired
4	User_10	4	Business Analyst	85000	Requirements: Bachelors in Business	5	2021-07-15	expired
5	User_10	4	Software Developer	90000	Must know C++	3	2021-07-15	active
6	User_11	5	Database Administrator	60000	Looking for McGill students only	1	2021-07-06	active
7	User_12	6	Python Developer	80000	Must know snakes	5	2021-08-01	active
8	User_12	7	Java Developer	60000	Must like coffee	2	2021-08-02	active
9	User_12	8	JavaScript Developer	70000	Must know all frameworks	8	2021-08-03	active
10	User_12	9	Rust Developer	85000	We will ask about metals	3	2021-08-04	active
12	User_12	11	Software Developer	100000	Work on Angular apps	4	2021-08-07	active
13	User_10	3	Main Project Creator	10000	Must be able to get 100% on our Project	1	2021-08-15	active

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

```
update Applications
set isAcceptedByEmployer = 1
where jobID = 5 and userID = "User_2";
-- Note: User_2 is an employer that has previously created a jobID "5"
```

- v. 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 jobs.title, jobs.description, jobs.datePosted as `date posted`,
       Applications.userID as "Applicant name", jobs.status,
       applications.isAcceptedByEmployee, applications.isAcceptedByEmployer
from Jobs, Applications
where jobs.jobID = Applications.jobID and jobs.jobID = 2;
-- Note: 1 (True) means application is accepted by either employee or employer
```

Result

title	description	date posted	Applicant name	status	isAcceptedByEmployee	isAcceptedByEmployer
Human Resources	Must have 20 years experience at any company	2021-08-02	User_2	active		
Human Resources	Must have 20 years experience at any company	2021-08-02	User_3	active		
Human Resources	Must have 20 years experience at any company	2021-08-02	User_4	active		
Human Resources	Must have 20 years experience at any company	2021-08-02	tyson	active	1	1

- vi. 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 jobs.title, jobs.datePosted as `date posted`, substring(jobs.description, 1, 10) as "10-char description", jobs.positionsAvailable,
       count(applications.userID) as "Total Applications", applications.isAcceptedByEmployer
from Jobs
left join applications on jobs.jobID = applications.jobID
where jobs.userID = "User_12" and jobs.datePosted between "2020-08-07" and "2022-08-07"
group by jobs.jobID;
-- Note: Since none of our data is over 50 characters, we decided to shorten the example by replacing 50 with 10.
```

Result

title	date posted	10-char description	positionsAvailable	Total Applications	isAcceptedByEmployer
Python Developer	2021-08-01	Must know	5	0	
Java Developer	2021-08-02	Must like	2	0	
JavaScript Developer	2021-08-03	Must know	8	0	
Rust Developer	2021-08-04	We will as	3	0	
Software Developer	2021-08-07	Work on An	4	0	

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

Create Employee

```
-- Create Employee
insert into users
(userID, planID, email, passwords, dateCreated, isActive, balance, isAutomatic)
values
("Einstein", 3, "Einstein@create-employee.net", "password", "2021-08-13", FALSE, 0, TRUE);
-- Note: planID (from subscription plans table) determines the user_type = "employee", where planID = 1, 2 or 3 are employees
```

Result

14	chris	5	chris@example.net	password	2021-08-07	1		0	1
19	Albert	4	Albert@create-user.net	password	2021-08-13	0		0	1
21	Einstein	3	Einstein@create-employee.net	password	2021-08-13	0		0	1

Edit Employee

```
-- Edit Employee
update users
set email = "Einstein@edit-employee.com"
where userID = "Einstein";
```

Result

14	chris	5	chris@example.net	password	2021-08-07	1		0	1
19	Albert	4	Albert@create-user.net	password	2021-08-13	0		0	1
21	Einstein	3	Einstein@edit-employee.com	password	2021-08-13	0		0	1

Display Employee

```
-- Display Employee
select *
from users
where userID = "Einstein";
```

Result

userNumber	userID	planID	email	passwords	dateCreated	isActive	startSufferingDate	balance	isAutomatic
21	Einstein	3	Einstein@edit-employee.com	password	2021-08-13	0		0	1

Delete Employee

```
-- Delete Employee
delete from users
where userID = "Einstein";
```

Result

14	chris		5	chris@example.net		password		2021-08-07		1		0		1
19	Albert		4	Albert@create-user.net		password		2021-08-13		0		0		1

viii. Search for a job by an employee.

```
select jobs.title, jobs.salary, jobs.description, jobs.positionsAvailable, jobs.datePosted, jobs.status,
       location.city, profiles.userID as "Employer", profiles.companyName as "Hiring Company"
from jobs, location, profiles, Users
where jobs.locationID = location.locationID
      and jobs.userID = profiles.userID
      and jobs.userID = users.userID
      and jobs.status = 'active'
      and lower(jobs.title) like lower('%Developer%');
-- Note: Developer can be replaced by whatever the employee is looking for like profiles.companyName for specific company
```

Result

title	salary	description	positionsAvailable	datePosted	status	city	Employer	Hiring Company
Software Developer	90000	Must know C++	3	2021-07-15	active	Burlington	User_10	Eminem Industry
Python Developer	80000	Must know snakes	5	2021-08-01	active	North Pender Island	User_12	Dr.Dre Industry
Java Developer	60000	Must like coffee	2	2021-08-02	active	Cobble Hill	User_12	Dr.Dre Industry
JavaScript Developer	70000	Must know all frameworks	8	2021-08-03	active	Pickering	User_12	Dr.Dre Industry
Rust Developer	85000	We will ask about metals	3	2021-08-04	active	Ile Perrot	User_12	Dr.Dre Industry
Software Developer	100000	Work on Angular apps	4	2021-08-07	active	Montreal	User_12	Dr.Dre Industry
Software Developer	85000	Must have 15 years of experience in PHP	3	2021-07-15	active	Brossard	User_5	Concordia Diploma

ix. Apply for a job by an employee.

```
insert into applications
(jobID, userID)
values
(5, "Albert");
-- Note: Job description/details will be read by the employee prior to being inserted into the applications table.
```

Result

jobID	userID	dateApplied	isAcceptedByEmployer	isAcceptedByEmployee
1	User_2	2021-08-06	1	
2	User_2	2021-08-06		
3	User_2	2021-08-06		
4	User_2	2021-08-06		
5	User_2	2021-08-06		
2	User_3	2021-08-07		
2	User_4	2021-08-07		
2	tyson	2021-08-07	1	1
5	Albert	2021-08-15		

x. Accept/Deny a job offer by an employee.

```
update applications
set isAcceptedByEmployee = true -- Accept Job Offer
where jobID = 5 and userID = "Albert";
-- Line 2 will be False for Deny job offer
```

Result

2	User_4	2021-08-07		
2	tyson	2021-08-07	1	1
5	Albert	2021-08-15		1

- xi. Withdraw from an applied job by an employee.

```
delete from applications
where userID = "Albert" and jobID = 5;
-- Removing the same application as previous query
```

Result

2	User_4	2021-08-07		
2	tyson	2021-08-07	1	1

- xii. Delete a profile by an employee.

Original User_Profiles table.

userID	locationID	companyName	firstName	lastName	profession	gender	phoneNumber
User_10	4	Eminem Industry	User	10	Singer	Male	123-456-7890
User_11	5	BP Oils	User	11	Engineer	Female	123-456-7890
User_12	9	Dr.Dre Industry	User	12	Dancer	Male	123-456-7890
User_5	1	Concordia Diploma	User	5	Senior Developer	Male	123-456-7890
User_6	2	Grey's Anatomy	User	6	Doctor	Male	123-456-7890
User_9	3	Law & Order	User	9.	Lawyer	Male	123-456-7890

```
-- xii. Delete a profile by an employee.

delete from profiles
where userID = "User_5"
-- Note: User_5 is created from previous query.
```

Result

userID	locationID	companyName	firstName	lastName	profession	gender	phoneNumber
User_10	4	Eminem Industry	User	10	Singer	Male	123-456-7890
User_11	5	BP Oils	User	11	Engineer	Female	123-456-7890
User_12	9	Dr.Dre Industry	User	12	Dancer	Male	123-456-7890
User_6	2	Grey's Anatomy	User	6	Doctor	Male	123-456-7890
User_9	3	Law & Order	User	9.	Lawyer	Male	123-456-7890

xiii. 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 jobs.title, applications.dateApplied, substring(jobs.description, 1, 10) as "10-char description",
       applications.dateApplied, applications.isAcceptedByEmployer, applications.isAcceptedByEmployee
from applications left join Jobs on jobs.jobID = applications.jobID
where jobs.userID = "User_10" and jobs.datePosted between "2020-08-07" and "2022-08-07"
group by jobs.jobID;
-- Note: Since none of our data is over 50 characters, we decided to shorten the example
-- by replacing 50 with 10.
```

Result

title	dateApplied	10-char description	dateApplied	isAcceptedByEmployer	isAcceptedByEmployee
Business Analyst	2021-08-06	Requiremen	2021-08-06		
Software Developer	2021-08-06	Must know	2021-08-06		

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

Add method of Payment

```
-- Add payment
insert into payment_methods
(userID, cardNumber, paymentType, isPreSelected)
values
("Albert", 123456789, "debit", false);
-- Note: Albert is used from previous queries
```

Result

paymentMethodID	userID	isPreSelected	paymentType	cardNumber
1	User_1	1	debit	1234
2	User_2	1	credit	1234
3	User_2	0	credit	1234
4	User_4	1	credit	1234
5	User_5	1	debit	1234
6	User_6	1	credit	1234
7	tyson	1	debit	1234
8	User_8	1	credit	1234
9	User_9	1	debit	1234
10	User_10	1	debit	1234
11	User_11	1	credit	1234
12	User_3	1	credit	1234
13	User_1	0	credit	1234
14	tyson	0	credit	1234
15	User_5	0	credit	1234
16	User_5	0	credit	1234
17	User_2	0	credit	1234
18	User_5	0	credit	1234
19	Albert	0	debit	123456789

Edit method of Payment

```
-- Edit Payment
update payment_methods
set paymentType = "credit"
where userID = "Albert";
```

Result

17	User_2	0	credit	1234
18	User_5	0	credit	1234
19	Albert	0	credit	123456789

Delete method of Payment

```
-- EdDelete Payment
delete from payment_methods
where userID = "Albert";
```

Result

17	User_2	0	credit	1234
18	User_5	0	credit	1234

- xv. Add/Delete/Edit an automatic payment by a user.

Add an automatic payment

```
-- Add automatic payment
insert into payment_methods
(userID, isPreSelected, cardNumber, paymentType)
values
("Albert", 1, 1234, "credit");
-- isPreSelected means a monthly charge
```

Result

paymentMethodID	userID	isPreSelected	paymentType	cardNumber
1	User_1	1	debit	1234
2	User_2	1	credit	1234
3	User_2	0	credit	1234
4	User_4	1	credit	1234
5	User_5	1	debit	1234
6	User_6	1	credit	1234
7	tyson	1	debit	1234
8	User_8	1	credit	1234
9	User_9	1	debit	1234
10	User_10	1	debit	1234
11	User_11	1	credit	1234
12	User_3	1	credit	1234
13	User_1	0	credit	1234
14	tyson	0	credit	1234
15	User_5	0	credit	1234
16	User_5	0	credit	1234
17	User_2	0	credit	1234
18	User_5	0	credit	1234
20	Albert	1	credit	1234

Edit an automatic payment

```
-- Edit automatic payment
update payment_methods
set isPreSelected = false
where userID = "Albert" and cardNumber = 1234;
```

Result

17	User_2	0	credit	1234
18	User_5	0	credit	1234
20	Albert	0	credit	1234

Delete an automatic payment

```
-- Delete automatic payment
delete from payment_methods
where userID = "Albert";
```

Result

17	User_2	0	credit	1234
18	User_5	0	credit	1234

xvi. Make a manual payment by a user.

```
-- xvi. Make a manual payment by a user.  
insert into payments  
(paymentMethodID, amount)  
values  
(2, 30);  
-- Note: The paymentMethodID is used to track down the user  
-- making the payment along with the mode of payment
```

Result:

transactionID	paymentMethodID	amount	paymentDate
1	2	30	2021-08-15

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

```
-- xvii. Report of all users by the administrator for employers or employees (Name, email, category, status, balance).  
select profiles.firstName, profiles.lastName, users.email, plans.name as "category", users.isActive as "Status", users.balance  
from profiles, users, plans  
where users.planID = plans.planID and users.userID = profiles.userID  
       and plans.userType != "admin";  
-- Note: 0 status means inactive
```

Result:

firstName	lastName	email	category	Status	balance
User	10	User_10@example.net	Employer Gold	0	50
User	11	User_11@example.net	Employer Prime	0	10
User	12	User_12@example.net	Employer Prime	1	0
User	6	User6@example.net	Employer Gold	1	-100
User	9.	User_9@example.net	Employer Prime	0	50

- xviii. Report of all outstanding balance accounts (Username, email, balance, since when the account is suffering).

```
-- xviii.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;
```

Result:

```
-- xviii.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;
```