CS2102

Project: Part 2

Marks 18%

1 Objective

The objective of this team project is for you to apply what you have learned in class to design and develop a database application using PostgreSQL. The project is to be done in teams of four students. The project consists of the following four tasks:

1. ~~Design an ER data model for the application. Your ER model should capture as many of the application's requirements as possible.~~
2. ~~Translate your ER data model into a relational database schema. Your relational schema should capture as many of the application's constraints as possible.~~
3. Implement triggers as required by the specification as listed in the Application Triggers.
4. Implement an SQL or PL/pgSQL functions/procedures for each of the functionalities listed in Application Functionalities.

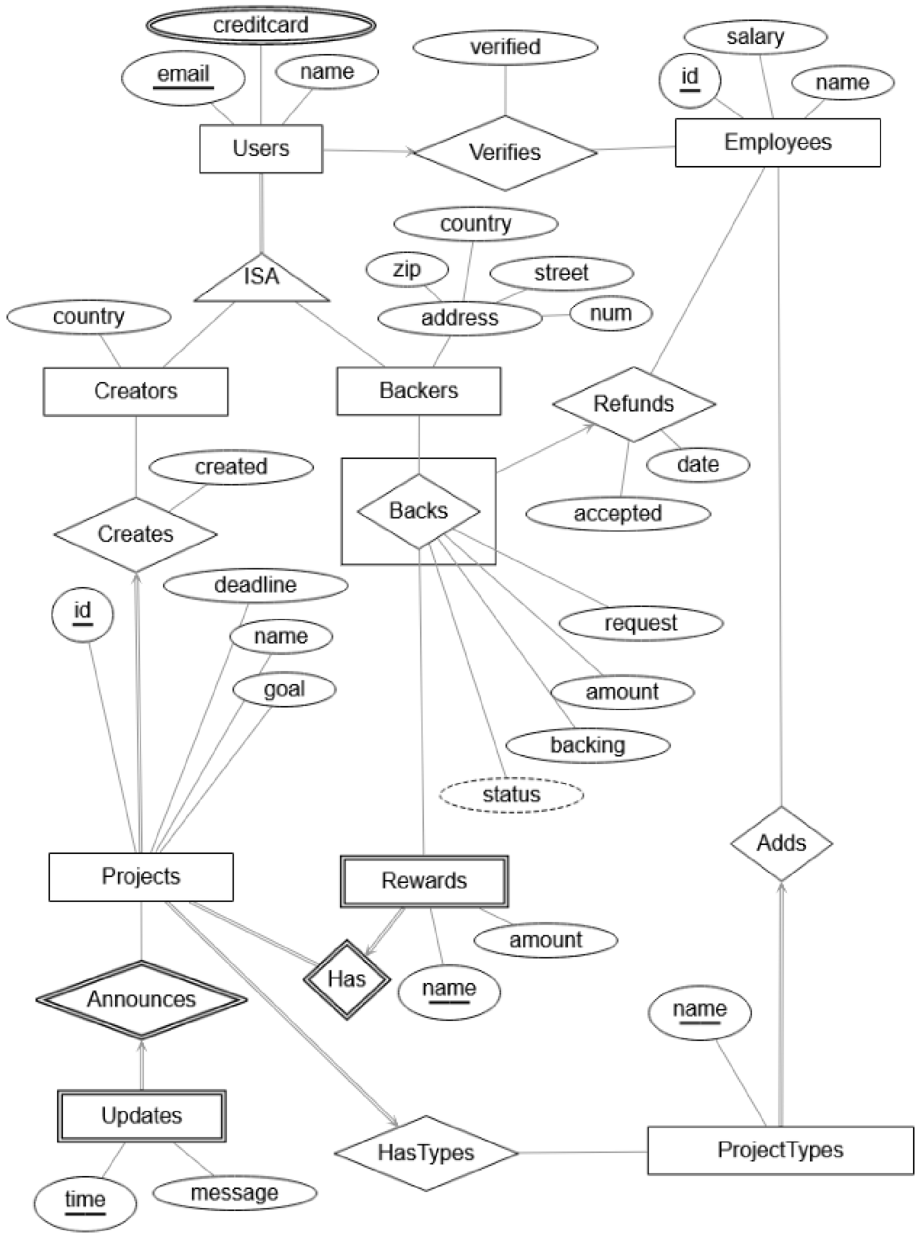
You have done the first two, we are left with task 3 and 4 now. You may want to read the **tl;dr** at the end of this file first.

2 Files

1. `DDL.sql`: The file containing the schema used for part 2 of the project.
2. `Proc.sql`: The file containing the template for the procedures and functions.

3 Recap: Part 1

3.1 ER Diagram



3.2 Schema

```
CREATE TABLE Employees (
  id      INT PRIMARY KEY,
  name    TEXT NOT NULL,
  salary NUMERIC NOT NULL CHECK (salary > 0)
);
```

```
CREATE TABLE Users (
  email   TEXT PRIMARY KEY,
  name    TEXT NOT NULL,
  cc1     TEXT NOT NULL,
  cc2     TEXT CHECK (cc1 <> cc2)
);
```

```
CREATE TABLE Verifies (
  email     TEXT PRIMARY KEY
    REFERENCES Users(email),
  id        INT NOT NULL REFERENCES Employees(id),
  verified DATE NOT NULL
);
```

```
CREATE TABLE Backers (
  email    TEXT PRIMARY KEY
    REFERENCES Users(email) ON UPDATE CASCADE,
  street  TEXT NOT NULL,
  num      TEXT NOT NULL,
  zip      TEXT NOT NULL,
  country TEXT NOT NULL
);
```

```
CREATE TABLE Creators (
  email    TEXT PRIMARY KEY
    REFERENCES Users(email) ON UPDATE CASCADE,
  country TEXT NOT NULL
);
```

```
CREATE TABLE ProjectTypes (
  name  TEXT PRIMARY KEY,
  id    INT NOT NULL REFERENCES Employees(id)
);
```

```sql
CREATE TABLE Projects (
  id       INT PRIMARY KEY,
  email    TEXT NOT NULL
    REFERENCES Creators(email) ON UPDATE CASCADE,
  ptype    TEXT NOT NULL
    REFERENCES ProjectTypes(name) ON UPDATE CASCADE,
  created  DATE NOT NULL, -- alt: TIMESTAMP
  name     TEXT NOT NULL,
  deadline DATE NOT NULL CHECK (deadline >= created),
  goal     NUMERIC NOT NULL CHECK (goal > 0)
);
```

```sql
CREATE TABLE Updates (
  time    TIMESTAMP,
  id      INT REFERENCES Projects(id)
    ON UPDATE CASCADE, -- ON DELETE CASCADE (optional)
  message TEXT NOT NULL,
  PRIMARY KEY (time, id)
);
```

```sql
CREATE TABLE Rewards (
  name    TEXT,
  id      INT REFERENCES Projects(id)
    ON UPDATE CASCADE, -- ON DELETE CASCADE (optional)
  min_amt NUMERIC NOT NULL CHECK (min_amt > 0),
  PRIMARY KEY (name, id)
);
```

```sql
CREATE TABLE Backs (
  email    TEXT REFERENCES Backers(email),
  name     TEXT NOT NULL,
  id       INT,
  backing  DATE NOT NULL, -- backing date
  request  DATE, -- refund request
  amount   NUMERIC NOT NULL CHECK (amount > 0),
  -- status will be derived via queries instead
  PRIMARY KEY (email, id),
  FOREIGN KEY (name, id) REFERENCES Rewards(name, id)
);
```

```sql
CREATE TABLE Refunds (
  email    TEXT,
  pid      INT,
  eid      INT NOT NULL
    REFERENCES Employees(id),
  date     DATE NOT NULL,
  accepted BOOLEAN NOT NULL,
  PRIMARY KEY (email, pid),
  FOREIGN KEY (email, pid)
    REFERENCES Backs(email, id)
);
```

Please study the schema above and note the difference from the ER diagram in the guide. The schema is available in the file `DDL.sql`.

- An additional check on `Users` that `cc2` must be different from `cc1` *(or NULL)*.
- An additional attribute `backing` on `Backs` to record the date the backer backs a project.
- The simplification of data types:
  - `INT` instead of `SERIAL` to avoid complications from running numbers.
    - This means that any `INSERT` will specify the `id` directly.
  - `TEXT` instead of `VARCHAR`.

4 Terminologies

Before we start, we will define a few useful terminology:

- **Successful Projects:** We say that a project is "successful" if the *pledged* amount *(see: money terminology below)* from all backers for all rewards meets or exceeds the funding goal AND the deadline has passed. If the project meets or exceeds the funding BUT the deadline has not passed, the project is not yet considered *successful*.
- **Failed Projects:** We say that a project has "failed" if the *pledged* amount *(see: Money terminology below)* from all backers for all rewards did not meet the funding goal AND the deadline has passed. If the deadline has not passed, the project is not yet considered *failed*.
  - In short, a project that has not passed the deadline is neither successful nor failed.

- **Refund Status:** There are four possible refund status that can be computed *(note, it is a computed attribute in our ER diagram)*:
  - *Not Requested:* A refund status is not requested if there is no request for the refund *(i.e., `Backs.request` is `NULL`)*.
  - *Requested:* A refund status is requested if there is a request for the refund BUT there is no approval and/or rejection yet *(i.e., `Backs.request` is not `NULL` and no entry in `Refunds` table)*.
  - *Accepted:* A refund status is accepted if it is a requested status and has been accepted *(i.e., `Refunds.accepted` is `True`)*.
  - *Rejected:* A refund status is rejected if it is a requested status and has been rejected *(i.e., `Refunds.accepted` is `False`)*.
- **Money:** We have a few terminology for money:
  - *Pledge:* A backer has pledged their money if the backer backs a project regardless of whether the project deadline has passed or not, whether the project is successful or not, and whether a refund request has been requested or not.
  - *Fund:* A backer has funded their money if the backer backs a project and the project is *successful*. Additionally, the backer have *not requested* a refund.
- **Success Metric:** We can rank successful projects by their success metric which is based on:
  - the ratio of the amount of money funded to the project and the funding goal *(i.e., funded / goal)* with larger values indicate a more successful project. The funded money is regardless of whether there are any refund or not.
  - if there are still any ties, we rank by the deadline with the later deadline indicate a more successful project.
  - lastly, if there are still any ties, we rank by the project id with smaller id indicate a more successful project.
- **Popularity Metric:** We can rank projects *(successful or not)* by their current popularity. This metric is based on how fast a project reach their funding goal counted as the number of days since the project is created. The smaller the better.
  - We rank by the popularity metric above.
  - If there are any ties, we rank by the project id in ascending order.
- **Verified Backers:** We say that a backer is a "verified" backer if the user information has been verified by an employee and recorded in the database in the table `Verifies`. Since there is a date component, a user is only verified from the day of verification.

- **Superbacker:** We say that a backer is a superbacker for a given date if the backer is a *verified* backer and the backer has satisfied one or both of the following:
  - The backer has backed at least 5 *successful* projects with the deadline of the project within 30 days of the given date *(in the past)* from at least 3 different project types. This is regardless of whether the backer has requested refund or not.
  - The backer has funded at least $1,500 on *successful* projects with the deadline of the project within 30 days of the given date *(in the past)*. Additionally, the backer must not have any refund requests, accepted refunds, or rejected refunds that are requested within 30 days of the given date.

  To elaborate on *within 30 days*, the date 01-01-2022 is within 30 days of 31-01-2022 but not within 30 days of 01-02-2022.

5 Application Triggers

Our relational schema could not enforce the following constraints listed below. For each constraints below, implement a trigger to enforce them. Unless specified, you only need to consider `INSERT` triggers *(i.e., triggers that are activated by insertions)*. You do NOT need to consider `DELETE` or `UPDATE` *(the only exception is the trigger requirement number 6, which requires an `UPDATE` trigger)*. For each constraint, you are allowed to use multiple triggers to enforce it. Please be careful about the order of triggers to ensure that the constraints are properly enforced.

1. Users must be backers, creators, or both. In other words, there must not be any users that are neither backers nor creators.
   - Consider the procedure `add_user`, how would the procedure affect this trigger?
2. Backers must pledge an amount greater than or equal to the minimum amount for the reward level.
3. Projects must have at least one reward level. In other words, there must not be any projects without any reward level.
   - Consider the procedure `add_project`, how would the procedure affect this trigger?
4. Employees can only approve refund that are requested within 90 days of the project deadline. In other words, for any refund request that are after 90 days of the project deadline, the employee can only reject the refund request. Additionally, refund not requested can neither be approved nor

rejected. If refund request exist, you may assume that the approval/rejection will be done after the request.

5. Backers can only back a project via a reward level before the deadline of the project and after it has been created. Here, we include the day of the deadline as before the deadline *(e.g., if the project deadline is today, then backers can back until the end of today)*.

6. Backers can only request for refund on *successful* projects. You need to create an `UPDATE` trigger here. You may assume that the only change is to set the `Backs.backing` from `NULL` to non-`NULL` values.

6 Application Functionalities

The routines *(i.e., functions/procedures)* application functionalities must be implemented with the same name specified in this document, following the same order of input parameters *(including type)*, returning the exact output parameter type *(if any)*, and using exactly the language specified *(i.e., `sql` or `plpgsql` only)*.

You may change the **name** of the input parameters in your implemented routines, but ***NOT*** the data type of the input parameters. You are encouraged to simply use the template given in `Proc.sql`.

If a call should fail because of invalid parameters or other database constraint violations, the routine should not return any values. You may raise an exception explicitly, or simply allow it to fail silently. It is important to adhere to these instructions since automated testing will be used to evaluate your implementation. Therefore, the input and output formats of the routines have to be fixed. Your code will be tested with different test cases. You are encouraged to create your own test cases for your own testing.

For each routines, you may assume that the input values are all valid. In other words, you do not need to check the validity of the inputs in your implementation.

In addition, if you are to use arrays (`https://www.postgresql.org/docs/14/arrays.html`), you should follow PostgreSQL's default behaviour of starting the index subscript from 1. You may also read up on ways to looping through arrays (`https://www.postgresql.org/docs/14/plpgsql-control-structures.html#PLPGSQL-FOREACH-ARRAY`) or you may refer to Lecture 06 on other array usage. You can also use any functions available in PostgreSQL *(e.g., the `EXTRACT` function for **DATE** or `TIMESTAMP` data type `https://www.postgresql.org/docs/14/functions-datetime.html`)*.

You may add additional procedures and/or functions to help your function. Make sure the name of your routines do not conflict with the given name. Lastly, your procedure/function may call another procedure/function specified in this document ***EXCEPT*** in the case where the language is `sql`. However, to simplify the task, in the case where the language is `sql`, you are allowed to use `CTE`.

In other words, if your routine for task 1 of functions is easier if you call the routine designed for task 2, then you may call the routine designed for task 2 within the routine for task 1. This may require rearrangement of the `CREATE OR REPLACE FUNCTION` statement in the template and you must ensure that your code can be run as if they are copy-pasted onto `psql`. However, do note that we will only grade the final effect. As such, if your routine for task 2 is wrong and is called within task 1, then both will be marked as wrong.

6.1 Procedures

The following routines in this section do not have return values, and should be implemented as PostgreSQL procedures `https://www.postgresql.org/docs/14/sql-createprocedure.html`. The routines must be successful on valid inputs even in the presence of the triggers above.

1. Write a procedure to add a user which may be a backer, a creator, or both.

```
CREATE OR REPLACE PROCEDURE add_user(
  email TEXT, name    TEXT, cc1  TEXT,
  cc2   TEXT, street  TEXT, num  TEXT,
  zip   TEXT, country TEXT, kind TEXT
) AS $$
-- add declaration here
BEGIN
  -- your code here
END;
$$ LANGUAGE plpgsql;
```

   - `email` is the user email.
   - `name` is the user name.
   - `cc1` is the user first credit card number.
   - `cc2` is the user second credit card number and it may be `NULL`.
   - `street` is the user address street name *(only used for backers)*.
   - `num` is the user address house number *(only used for backers)*.
   - `zip` is the user address zip code *(only used for backers)*.
   - `country` is the user address country code *(can be used for both backers and creators)*.
   - `kind` is one of the following:
     - `'BACKER'` if the user is only a backer.
     - `'CREATOR'` if the user is only a creator.
     - `'BOTH'` if the user is both a backer and a creator.

2. Write a procedure to add a project and the corresponding reward levels. If any reward level cannot be created, this procedure should not create the project. You may assume that the creator and project type already exist.

```sql
CREATE OR REPLACE PROCEDURE add_project(
  id       INT,      email  TEXT,    ptype    TEXT,
  created  DATE,     name   TEXT,    deadline DATE,
  goal     NUMERIC, names  TEXT[],
  amounts  NUMERIC[]
) AS $$
-- add declaration here
BEGIN
  -- your code here
END;
$$ LANGUAGE plpgsql;
```

   - `id` is the project id.
   - `email` is the creator email.
   - `ptype` is the project type.
   - `created` is the date the project is created.
   - `name` is the project name.
   - `deadline` is the project deadline.
   - `goal` is the funding goal.
   - `DATATYPE[]` refers to an array of `DATATYPE`.
     - For each index `idx`, the element at index `idx` corresponds to the same reward level.
     - `names` is an array of reward names.
     - `amounts` is an array of minimum amount to back the project through the reward.

3. Write a procedure to help an employee with id `eid` automatically reject all refund request where the date of the request is more than 90 days from the deadline of the project. Set the rejection date as the given date `today`. You may assume for this procedure that all the data in the database exists in the past *(i.e., no need to check date)*.

```sql
CREATE OR REPLACE PROCEDURE auto_reject(
  eid INT, today DATE
) AS $$
-- add declaration here
BEGIN
  -- your code here
END;
$$ LANGUAGE plpgsql;
```

- `eid` is the employee id.
- `today` is today's date *(i.e., reject today)*

6.2 Functions

The following routines in this section have return values, and should be implemented as PostgreSQL function `https://www.postgresql.org/docs/14/sql-createfunction.html`. The routines must be successful on valid inputs even in the presence of the triggers above.

1. Write a function to find the email and name of all *superbackers* for a given month and year.

```
CREATE OR REPLACE FUNCTION find_superbackers(
  today DATE
) RETURNS TABLE(email TEXT, name TEXT) AS $$
-- add declaration here
BEGIN
  -- your code here
END;
$$ LANGUAGE plpgsql;
```

   - `today` is today's date *(i.e., 30 days before today)*.
   - The result should be ordered in ascending order of `email`.
   - The outputs are:
     - `email`: The backer email.
     - `name`: The backer name.

2. Write a function to find project id, the name of the project, the email of the creator, and the amount pledged for the project for the top N most successful project based on the success metric for given date *(i.e., the project deadline must be before the given date)* and for the given project type.

```
CREATE OR REPLACE FUNCTION find_top_success(
  n INT, today DATE, ptype TEXT
) RETURNS TABLE(id INT, name TEXT, email TEXT,
                   amount NUMERIC) AS $$
  SELECT 1, '', '', 0.0; -- replace this
$$ LANGUAGE sql;
```

   - `n` is the top N.
   - `today` is the date we consider today *(i.e., the project deadline must be before today)*.
   - `ptype` is the project type.

- The result should be ordered in descending order of **success metric**.
- The outputs are:
  - `id`: The project id.
  - `name`: The project name.
  - `email`: The creator email.
  - `amount`: The total amount funded *(regardless of whether refund request is created or not)*.
- **WARNING:** The language here is `sql` instead of `plpgsql`.

3. Write a function to find the project id, name of the project, the email of the creator, and the number of days it takes for the project to reach its funding goal for the top N most popular project based on the popularity metric for the given date *(i.e., the project must be created before today)* and for the given project type.

```
CREATE OR REPLACE FUNCTION find_top_popular(
  n INT, today DATE, ptype TEXT
) RETURNS TABLE(id INT, name TEXT, email TEXT,
                days INT) AS $$
    -- your code here
$$ LANGUAGE plpgsql;
```

- `n` is the top N.
- `today` is the date we consider today *(i.e., the projects must be created before today)*.
- `ptype` is the project type.
- The result should be ordered in descending order of **popularity metric**.
- The outputs are:
  - `id`: The project id.
  - `name`: The project name.
  - `email`: The creator email.
  - `days`: The number of days to reach funding
    - If the funding goal is the same as the creation date, this value should be 0.
- **HINT:** You may want to look at `RETURN QUERY` if you have an SQL query solution.

7 Deliverables

Each team is to upload an **ZIP** file named `teamNNN.zip` where `NNN` is the three digit team number according to your project group number. You should add leading zeroes to your team number (*e.g.,* `team005.zip`). The **ZIP** file should contain the following two files:

1. `Report.pdf`: Project report in **PDF** format.
2. `Proc.sql`: The triggers and routines of your implementation *(for simplicity, this is the same name as the template file)*.

Submit your pdf file on Canvas assignment named `ER + Schema` (`https://canvas.nus.edu.sg/courses/24662/assignments/11992`). Only one file is to be submitted per group. If there are multiple submissions for a group, the latest submission will be chosen. If the later submission is late, late penalty will apply.

The project report must be at most **20 pages** with font size of **at least 12 point** and consists of the following:

- Project team number & names of team members (*on the first page*).
- A listing of the Project (Part 2) responsibilities of each team member.
  - Team members who did not contribute a fair share of the project work will not receive the same marks awarded to the team.
- For each trigger:
  - Provide the name of the trigger and the trigger function.
  - Explain the basic idea of the trigger and the trigger function implementation.
- For each routine:
  - Provide the name of other routines used.
  - Explain the basic idea of the routine implementation.
- For each additional routines *(if any)*:
  - Explain the basic idea of the routine implementation.
- A summary of any difficulties encountered and lessons learned from the project.

8 Deadline

The deadline for the submission is 23:59 of Week 13, Saturday, 12 November 2022. Five marks (out of eighteen) will be deducted for submissions up to one day's late. Submissions late for more than one day will receive zero marks and will not be graded.

9 FAQ

1. *"Why are we not allowed to make any changes to the schema provided or the routine names and parameters given?"*
   - This is because we will evaluate each submission using automatic testing. That is, we will prepare some test data following the schema provided to you, and then combine it with your `Proc.sql` and run some queries. We will examine the database state after each query to check whether it is correct; if it is, then you get some marks associated with the query. To facilitate such automatic testing, it is important that every team's implementation is based on exactly the same schema, and uses the exact routine interface that we provided.
2. *"Will the test data be provided?"*
   - Unfortunately no. But you may generate your own test data to check your implementation.

10 tl;dr

Note that these **tl;dr** may be a simplification. You are advised to check with the requirements above for more details.

10.1 Triggers

1. Enforce the constraint `Users === {Creators, Backers}`.
2. Enforce the constraint that (backers' plegde amount) $\geq$ (reward level minimum amount).
3. Enforce the constraint `Projects === Has` *(i.e., every project has at least one reward level).*
4. Enforce the constraint that refund can only be approved for refunds requested within 90 days of deadline. Also enforce the constraint that refund not requested cannot be approved/rejected.
5. Enforce the constraint that backers back before deadline.
6. Enforce the constraint that refund can only be made for successful projects.

10.2 Procedures

1. Write a procedure to add a user which may be a backer, a creator, or both.
2. Write a procedure to add project with all the corresponding reward levels.
3. Write a procedure to help an employee auto-reject all refund request made after 90 days from deadline.

10.3 Function

1. Find all *superbackers* sorted by `email` which satisfies one or both of the following
   - Condition #1: satisfy all
     - Backed $\geq$ 5 successful projects in the last 30 days
     - Backed $\geq$ 3 project types in the last 30 days
   - Condition #2: satisfy all
     - Funded $\geq$ \$1,500 on successful projects in the last 30 days
     - 0 refund request in the last 30 days
2. Find top N projects based on **success metric**. Success metric may be summarised as sorting in the following order:
   - (total money funded) / (project goal) *descending*.
   - (project deadline) *descending*.
   - (project id) *ascending*.
3. Find top N projects based on **popularity metric** Popularity metric may be summarised as sorting in the following order:
   - (# days for funding goal to be reached) *descending*.
   - (project id) *ascending*.