CS2102

Project: Part 1

Marks 10%

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.

2 Application Description

Punchender is a crowd-funding system designed to compete with Kickstarter (`https://www.kickstarter.com/`) and Indiegogo (`https://www.indiegogo.com/`). Its motto is to end the status quo with a punch.

The crowd-funding system allows an aspiring creator to get funding for their project. Projects can be almost anything such as movie production, board game creation, or electronic products. Users of this application are either creator looking to get their projects funded or backers funding parts of the project *(or both)*. Each backer may only fund the project once before the deadline.

Each user is identified by their unique email address. Their names must also recorded in the system. A user may have up to two credit cards but they must have at least one credit card. When users first sign up for an account, they must specify whether they want to be a creator or backer or both.

If the users are backers, their home address must be recorded for the delivery of the final products. An address is composed of the street name, house number, zip code, and country name. On the other hand, if the users are creators, only their country of origin must be recorded for the purpose of taxation since every country have different tax codes.

A project is identified by a unique ID auto-generated by the system so that we can have two projects created by the same creator with the same name. Auto-generated IDs can be created using the `SERIAL` data type. For each project we must record the project names, funding goal, and deadline to reach the funding. There can only be one creator for each project. We must also record the date when the project is created.

Additionally, each project must be created by a creator. In other words, there are no projects that are not created by a creator. However, a project may have many backers funding the project but each backer can only fund the same project exactly once, although they may not fund any project. Obviously, a backer may backs multiple projects at the same time.

Associated with a project is the reward level. For instance, a project called "Metal" may have three reward levels called "Platinum", "Gold" and "Silver". On the other hand, another project called "Olympic" may have three reward levels called "Gold", "Silver" and "Bronze".

Note that the name of the reward level is unique only within the same project. Additionally, each reward level must record the minimum amount of money the backer can fund the project at that reward level. However, the user may decide to back the reward level with a higher amount than the stated value. For example, if the "Gold" reward level of the "Metal" project require the user to fund at least \$100, the user may choose to spend \$120 but not \$99.

A user can only fund the project using the reward level. So a user wants to back the "Metal" project, they must back one of the "Platinum", "Gold" or "Silver" reward level. As such, if a user is already backing the "Gold" reward level for "Metal", the same user cannot back the "Platinum" reward level for "Metal". However, the user may back the "Gold" reward level for "Olympic".

But again, remember that the backers can only fund the same project exactly once. So, in the example above, if the backer already back the "Metal" project on the "Silver" reward level, the same user cannot back the "Metal" project on the "Gold" or "Platinum" level.

To keep the backers up to date, the project creator may provide updates to all the backers of a given project. For each project, the creator can only announce one update at any single time (*here, time includes the date as well*). In other words, no two simultaneous updates. Unfortunately, as some projects may not be satisfactory to the backers even with frequent updates, the backers may choose to request for refund within 90 days of the deadline of the project.

In order to process this refund, Punchender hires a number of employees. Each employee must have a unique ID auto-generated by the system with name and monthly salary recorded. Employees will be needed to approve or reject a refund request by the backer. For each refund request by a backer with a recorded date of request within 90 days of project deadline, the employee either approves the request and record the date of acceptance or rejects the request and record the date of rejection. Note that the user will not be charged if the funding did not meet the goal. In such cases, no refund can be made.

Note that a refund only requires intervention by the employee during approval or rejection only. In particular, when a refund is still pending, there may not be an associated employee. For simplicity, each user may only request for refund exactly once. As such, once a rejection is given, there can be no other request for refund for the given project the user is backing.

The employees are also responsible for verifying users. Although not all users need to be verified, some users may be verified. In the verification process, a user (*regardless of whether they are a backer or a creator*) is verified by at most one employee. We must record the date of verification since a verification that was done too far in the past may have invalid data.

3 Deliverables

Each team is to upload a pdf file named `teamNNN.pdf` 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.pdf`). Submit your pdf file on Canvas assignment named `ER + Schema` (`https://canvas.nus.edu.sg/courses/24662/assignments/7126`). Only one file is to be submitted per group. If there are multiple submissions for a group, the submission with the lower mark will be chosen.

The submitted pdf file must be at most **8 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*).
- ER data model of your application from `https://www.comp.nus.edu.sg/~cs2102/Tools/ERD/`.
  - If your ER model is too large (*i.e., spanning more than a page*), you may want to include a single-page simplified ER model (*i.e., with non-key attributes omitted*) before presenting the detailed ER model.
  - Your ER diagram cannot be too small. The font in the ER diagram must be readable. You may rearrange your ER diagram in `https://www.comp.nus.edu.sg/~cs2102/Tools/ERD/` manually to produce a more vertically oriented ER diagram.
- The relational database schema corresponding to your ER data model in the form of `CREATE TABLE` statements.
- Justification for any non-trivial design decisions made.
  - This includes *but not limited to* the use ISA hierarchies, aggregates, and ternary relationship set.
- List down up to 5 of the application's constraints that are not captured by the proposed ER model.
- List down up to 5 of the application's constraints that are not captured by the proposed relational schema.

In both ER diagram and relational schema, you should:

- capture as many of the application's constraints as possible (*as per specifications*).
- not capture constraints not specified in the application's constraints (*as per specifications*).
  - Your relational schema may capture different sets of constraints compared to your ER diagram.
  - Any constraints that can be captured but did not (*or not specified in specification but erroneously added*) may be penalised *except* for reasonable real-world constraints.

4 Deadline

The deadline for the submission is 6pm of Saturday, 17 September 2022. Two marks (out of ten) 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.

5 Grading

The following grading scheme will be used. The details of the grading scheme is hidden.

5.1 ER Diagram (*5 Marks*)

- **(1 Mark)** Validity of ER Diagram.
  - The ER diagram should not contain invalid constructs.
- **(2 Marks)** Data Requirements
  - ER diagram should capture all the data needed by the application's requirement.
- **(2 Marks)** Constraint Requirements
  - ER diagram should capture as many of the application's requirements as possible.
  - ER diagram should not add constraints not specified in the application's requirements whenever possible.

5.2 Relational Schema (*5 Marks*)

- The relational schema should translate the ER diagram as much as possible when the requirements match with the application's requirements.
- The relational schema should enforce as many of the application's constraints as possible.
  - This includes reasonable real-world constraints.