CS2102

Database Systems

Team 18

Topic B: Crowdfunding

Name and matric

Deric

Wei Zheng

Shi Rong

Chen Zhi Liang A0141010L

# Introduction

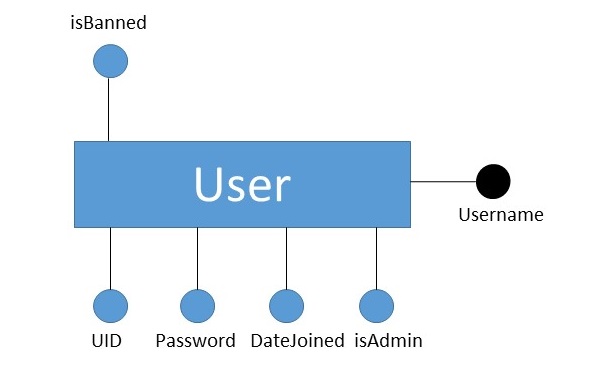
In our crowdfunding website, we have two self-explanatory entities – Users and Projects, with the latter being a weak entity. These two entities are related in relationships called Investment because users can choose to invest in projects of their choice. In the ER diagrams in the following section, we can see the various attributes and keys associated with each entity and relationships.

In this project, we tried as much as possible to enforce constraints, update data and perform meaningful data extraction and feedback on the database itself; this allows us to demonstrate some of the SQL queries and advanced features on the database and not through PHP code on the website.

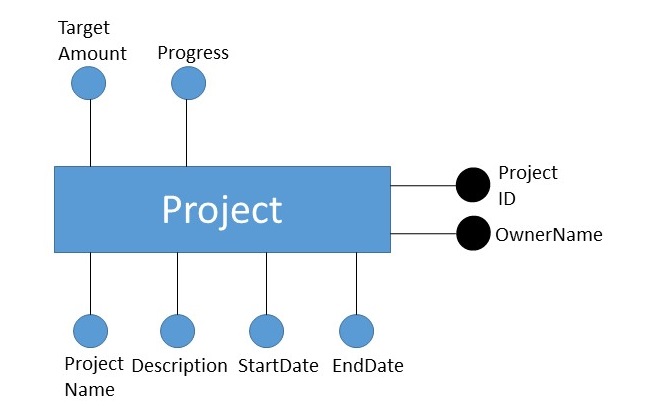
# Project Specifications

Entity Relationship Diagrams

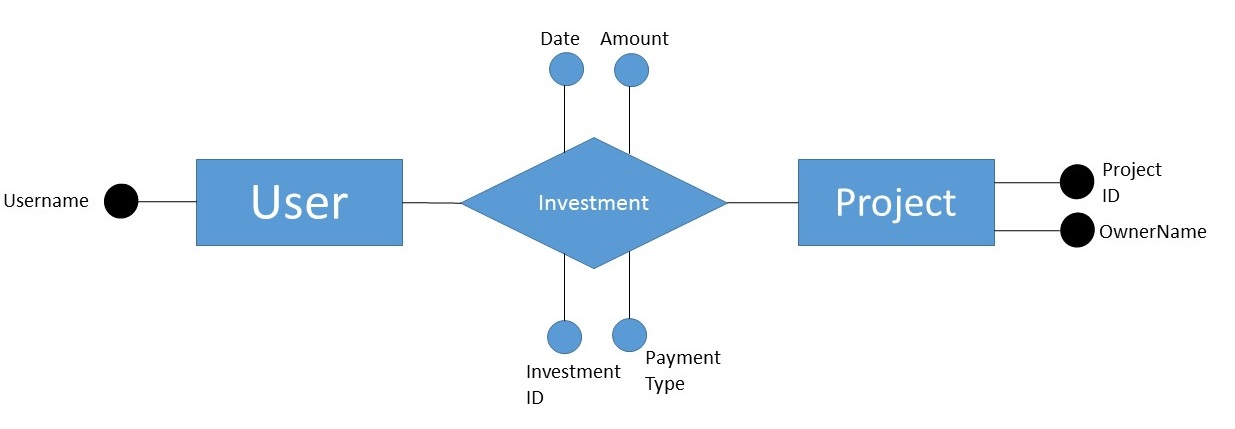
Fig 1. User Entity



*Fig 2. Project Entity*



*Fig 3. Investment Relationship*



**Relational Schema**

**User Entity**

create table users (

UID int,

userName varchar(50),

pssword varchar(50),

dateJoined date,

isAdmin boolean NOT NULL,

isBanned boolean NOT NULL,

billingAddress varchar(100),

PRIMARY KEY (userName)

);

**Project Entity**

create table projectsOwnership (

projectName varchar(100) NOT NULL,

projectDescription varchar(1000),

startDate date NOT NULL,

endDate date NOT NULL,

projectID varchar(50),

ownerName varchar(10)NOT NULL,

targetAmount int NOT NULL CHECK (targetAmount > 0),

progress int NOT NULL CHECK (progress >= 0),

category varchar(20),

projectStatus varchar(10),

PRIMARY KEY (projectID, ownerName),

FOREIGN KEY (ownerName) REFERENCES users(userName)

on update cascade);

**Investment Relationship between User and Project**

create table investments (

amount int NOT NULL CHECK (amount > 0),

dateInvested date NOT NULL,

investmentID varchar(100),

investorName varchar(50) NOT NULL,

investmentType varchar(50) NOT NULL,

projectID varchar(50) NOT NULL,

ownerName varchar(50) NOT NULL,

CONSTRAINT chk\_InvestmentType CHECK (investmentType IN aaaaaaa('eNETS','Paypal', 'Credit Card')),

PRIMARY KEY (investmentID),

FOREIGN KEY (projectID, ownerName) REFERENCES s projectsOwnership(projectID, ownerName)

on delete cascade

on update cascade,

FOREIGN KEY (investorName) REFERENCES users(userName)

on delete cascade

on update cascade);

**Integrity Constraints**

# Data Type constraints:

We need some attributes in the relations to follow certain obvious data type constraints. For example, for crowd funding projects created by users, the target amount to be raised must clearly be more than 0. We use table constraints to achieve this:

targetAmount int NOT NULL CHECK (targetAmount > 0)

When such constraints are broken by users, an exception will be raised in the database which will be converted to relevant feedback on the website.

Other examples of data type constraints during creation of table:

CONSTRAINT chk\_InvestmentType CHECK (investmentType IN ('eNETS', 'Paypal', 'Credit Card'))

progress int NOT NULL CHECK (progress >= 0)

# Foreign Key Constraints:

1. **Project**

We identify crowdfunding projects as weak entity – it cannot exist on its own without a user owner. As such, we use define the owner username and project identification number (ID) as the primary key used to identify projects (Note that ID alone will not allow the identification of projects, much alike the situation of repeated matriculation numbers of students across different universities) Under such circumstances, the owner username naturally becomes a foreign key constraint – the owner of a crowdfunding project must be a registered user under the User schema:

FOREIGN KEY (ownerName) REFERENCES users(userName)

1. **Investment**

Investments, being a relationship, clearly holds the primary keys of User (investors) and Projects. It is clear that these keys become foreign keys under the Investment Schema:

FOREIGN KEY (projectID, ownerName) REFERENCES projectsOwnership(projectID, ownerName)

FOREIGN KEY (investorName) REFERENCES users(userName)

# Primary Key Constraints for Entity:

1. **User**

Because users are required to log into the website with their username, each username will be unique to a user. As such, it becomes straightforward to use the username as the Primary Key under the User schema (See *Fig 1*).

1. **Project**

It would be unwise to use Project description or name as the Primary key (unlike users) because projects can possibly have the same chosen name. As such, we need to impose a project ID for each project as part of the Primary key. However, because we have identified Project as a weak entity which is dependent on User (since a User owns a Project), we additionally include the User’s Primary key as part of the Project Primary key. The resulting Primary key is as follows:

PRIMARY KEY (projectID, ownerName)

**Advanced SQL features**

# Triggers

We use the *Trigger* feature of SQL to automate updates to the database when actions are performed by users. In some situations, *Triggers* can also be used to enforce certain constraints which cannot be implemented when creating the Relational Schema. For example, when a user attempts to change his password into the same password (this cannot be created as a SQL constraint directly), the following *Trigger* is able to reject this action by raising an exception in the database, which is then translated to feedback to the user.

create trigger samePasswordError

before update

on users

for each ROW

when (old.pssword = new.pssword)

EXECUTE PROCEDURE samePassword();

create or replace function samePassword()

returns trigger as $$

BEGIN

raise exception 'New password must be different;

return null;

END;

$$ language plpgsql;

# Aggregation

Because users can invest in various projects and projects can receive investments from different users, it is then natural to use *aggregates* in SQL over such attributes. Various functionalities performed on the website are enabled with the use of aggregate functions. For example, an admin can check the highest investment on a certain date; the query is as follows:

SELECT MAX(investments.Amount) AS max FROM investments WHERE Date = % %ChosenDate%

A user can also choose to view his total investment:

SELECT SUM(amount) AS sum FROM investments WHERE investorName = %name%