

CS2102 Team 18

Crowdfunding



|  |  |
| --- | --- |
| LIM WEI JIE | A0139128A |
| MARCUS NG WEN JIAN | A0139257X |
| SANKARA RAJULU GAUTAM RAJULU | A0168916N |
| SNG YONG JIE DYLAN | A0123456J |

Table of Contents

[Introduction 2](#_Toc527467401)

[Project Specifications 2](#_Toc527467402)

[Entity-relationship Diagram 3](#_Toc527467403)

[Relational Schema 4](#_Toc527467404)

[TABLE member 4](#_Toc527467405)

[TABLE advertised\_project 4](#_Toc527467406)

[TABLE invest 4](#_Toc527467407)

[Integrity Constraints 5](#_Toc527467408)

[Primary Constraints 5](#_Toc527467409)

[Foreign Key Constraints 5](#_Toc527467410)

[Not Null 5](#_Toc527467411)

[Default 6](#_Toc527467412)

[Check 6](#_Toc527467413)

[Advanced SQL Features 7](#_Toc527467414)

[Triggers and Functions 7](#_Toc527467415)

[Aggregation 9](#_Toc527467416)

[Subquery and Sum 9](#_Toc527467417)

[Subquery and Union 9](#_Toc527467418)

[Screenshots 10](#_Toc527467419)

# Introduction

Our crowdfunding website provides the following functionalities:

* Create, Read, Update, Delete (CRUD)

- New members can register for an account

- Members can create projects and put on advertisement

- Members can view all projects in the *Explore* page

- Members can view their own project in the *Profile* page under *My Projects* tab

- Members can view their investments in the *Profile* page under *My Investments* tab

- Members can update details of their own project (Eg. title, amount needed, etc)

- Members can update their personal information

- Members can delete their own project

- Members can delete their investment in a particular project

Terminology

User – Anyone who interact with our website

Member – Anyone who is registered in our database (Entrepreneur/Investor/Admin)

Entrepreneur – Someone who creates and advertised a project

Investor – Someone who contributes to the amount raised of the project

# Project Specifications

|  |  |
| --- | --- |
| Stack | Bitnami |
| Frontend | HTML, CSS, JS |
| Backend | PHP |
| Database | Postgresql / phpPGAdmin |

# Entity-relationship Diagram

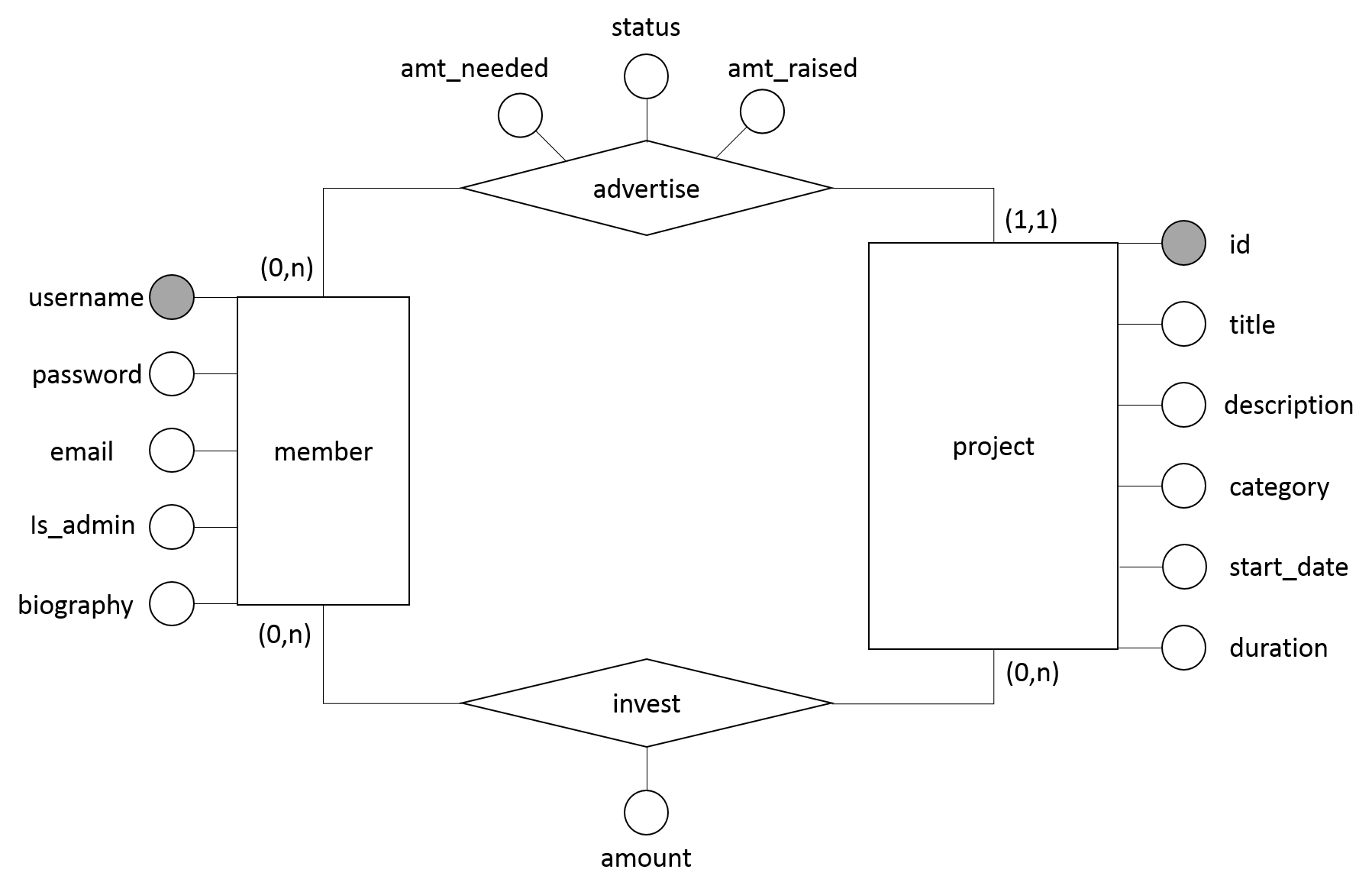


Figure 1. ER Diagram

# Relational Schema

## TABLE member

|  |
| --- |
| CREATE TABLE member (  username VARCHAR(16) PRIMARY KEY,  password VARCHAR(16) NOT NULL,  email VARCHAR(64) NOT NULL DEFAULT '',  biography text NOT NULL DEFAULT '',  is\_admin INT NOT NULL DEFAULT 0 CHECK(is\_admin=0 OR is\_admin=1)  ); |

## TABLE advertised\_project

We have merged the tables ‘advertise’ and ‘project’ and use the primary key of the ‘project’ table. Hence, each project can only be advertised by exactly one entrepreneur.

|  |
| --- |
| CREATE TABLE advertised\_project (  id SERIAL PRIMARY KEY CHECK(id >=0),  entrepreneur VARCHAR(16) REFERENCES member(username) ON DELETE CASCADE,  title VARCHAR(256) NOT NULL,  description VARCHAR(2048) NOT NULL DEFAULT '',  category VARCHAR(256) NOT NULL CHECK(category='Fashion' OR  category='Technology' OR  category='Games' OR  category='Food' OR  category='Music' OR  category='Photography' OR  category='Handicraft' OR  category='Community'),  start\_date DATE NOT NULL DEFAULT CURRENT\_DATE,  duration INT NOT NULL,  amt\_needed NUMERIC(15,2) NOT NULL CHECK(amt\_needed > 0),  amt\_raised NUMERIC(15,2) NOT NULL DEFAULT '0.00'  CHECK(amt\_raised<=amt\_needed AND amt\_raised>=0),  status INT NOT NULL DEFAULT 0 CHECK(status=0 OR status=1)  ); |

## TABLE invest

|  |
| --- |
| CREATE TABLE invest (  investor VARCHAR(16) REFERENCES member(username) ON DELETE CASCADE,  proj\_id SERIAL REFERENCES advertised\_project(id) ON DELETE CASCADE,  amount NUMERIC(15,2) NOT NULL CHECK(amount > 0),  PRIMARY KEY(investor, proj\_id)  ); |

# Integrity Constraints

Integrity constraints provide a mechanism for ensuring that data conforms to our business requirements. The following are some notable examples of the constraints we used in this project. For more examples, please refer to the relational schema.

## Primary Constraints

#### Member

For every user who wants to interact with our website, he/she must first register with a unique username. As the username is unique, it is natural that we choose “username” as the primary key for this table.

|  |
| --- |
| username VARCHAR(16) PRIMARY KEY |

#### Advertised\_Project

Every time an entrepreneur creates a project, it is automatically advertised and published in the *Explore* page. As titles, descriptions and other fields may be the same for different projects, each project is given an id to uniquely identify it. We do not allow a project to be advertised by more than one entrepreneur, hence there cannot be multiple entry of the same project in this table. Therefore, the id of the project has been chosen to be the primary key of this table.

|  |
| --- |
| id SERIAL PRIMARY KEY CHECK(id >=0) |

#### Invest

As it is possible that an investor invests in multiple projects. Hence, we have chose the primary key to be a composite of the username of the investor, as well as the project id, to uniquely identify each investment record.

|  |
| --- |
| PRIMARY KEY(investor, proj\_id) |

## Foreign Key Constraints

#### Advertised Project

A project cannot exist on its own without being advertised by an entrepreneur. As such, the entrepreneur naturally becomes the foreign key constraint. As shown in figure 1, the relationship between entrepreneur and project is (1,1). Hence, making the entrepreneur a foreign key constraint will ensure that there is always a registered entrepreneur. Together with the project id as the primary key, it ensures that the (1,1) participation constraint is met.

|  |
| --- |
| entrepreneur VARCHAR(16) REFERENCES member(username) ON DELETE CASCADE |

## Not Null

This constraint is added to virtually every key to prevent any null values from appearing in the database.

## Default

This constraint provides a default value for a column if no value is specified. For text, it will be set to be an empty string while other fields are set to a reasonable value. Together with the NOT NULL constraint, this will ensure that every record in the tables will not contain null, preventing any error thrown when performing operations on the database. The following are some examples of default values that we specified.

|  |
| --- |
| biography text NOT NULL DEFAULT ''  is\_admin INT NOT NULL DEFAULT 0 CHECK(is\_admin=0 OR is\_admin=1)  start\_date DATE NOT NULL DEFAULT CURRENT\_DATE |

## Check

This constraint limits the value range that can be placed in a column so that they confirm to the business requirements. When such constraints are violated, exceptions will be raised by the database. The following are some examples.

#### All user who registered via the website must be non-admin by default

|  |
| --- |
| is\_admin INT NOT NULL DEFAULT 0 CHECK(is\_admin=0 OR is\_admin=1) |

#### Projects can only be classified under these categories

|  |
| --- |
| category VARCHAR(256) NOT NULL CHECK (category='Fashion' OR  category='Technology' OR  category='Games' OR  category='Food' OR  category='Music' OR  category='Photography' OR  category='Handicraft' OR category='Community') |

#### Investors must invest with a positive sum of money

|  |
| --- |
| amount NUMERIC(15,2) NOT NULL CHECK(amount > 0) |

# Advanced SQL Features

## Triggers and Functions

There are some business requirements that cannot be expressed via integrity constraints in the relational schema. Hence, make use of these triggers to automatically execute the functions when an event occurs in the database server. An event can occur when a user performs some action such that the front-end make a sql query to the database. The following are some notable examples of the triggers and functions we used in this project. For more examples, please refer to the source code.

#### Dynamic updating of value in a column

Whenever there is a change in the amount raised in the TABLE advertised\_project, we have to ensure that the completion status is consistent. This dynamic updating of the value of a column cannot be done within the relational schema. Using triggers and functions, we can automatically change the completion status of a project.

|  |
| --- |
| CREATE OR REPLACE FUNCTION toggle\_status()  RETURNS TRIGGER AS $$  BEGIN  IF OLD.status=0 THEN  UPDATE advertised\_project SET status=1 WHERE id=NEW.id;  ELSE  UPDATE advertised\_project SET status=0 WHERE id=NEW.id;  END IF;  RETURN NEW;  END; $$ LANGUAGE PLPGSQL;  CREATE TRIGGER trigger\_change\_status\_after\_update\_amtRaised  AFTER UPDATE on advertised\_project  FOR EACH ROW  WHEN (NEW.amt\_raised = OLD.amt\_needed AND OLD.status=0 OR NEW.amt\_raised <> OLD.amt\_needed AND OLD.status=1)  EXECUTE PROCEDURE toggle\_status();  CREATE TRIGGER trigger\_change\_status\_after\_update\_amtNeeded  AFTER UPDATE on advertised\_project  FOR EACH ROW  WHEN (NEW.amt\_needed = OLD.amt\_raised AND OLD.status=0 OR NEW.amt\_needed <> OLD.amt\_raised AND OLD.status=1)  EXECUTE PROCEDURE toggle\_status(); |

#### Trigger triggering another trigger

Some events may trigger a trigger, which in turn trigger another trigger. For example, when an investor attempts to invest in a project, his action will trigger the following trigger and functions to check if his investment exceeds the targeted amount for the project. If his investment is accepted (i.e is within range of target amount), it will trigger the above trigger to update the completion status of the project.

|  |
| --- |
| CREATE OR REPLACE FUNCTION update\_amt\_raised\_when\_insert\_invest()  RETURNS TRIGGER AS $$  DECLARE  amtNeeded NUMERIC(15,2);  currentAmtRaised NUMERIC(15,2);  resultingAmtRaised NUMERIC(15,2);  BEGIN  amtNeeded = (SELECT amt\_needed FROM advertised\_project WHERE id = NEW.proj\_id);  currentAmtRaised = (SELECT amt\_raised FROM advertised\_project WHERE id = NEW.proj\_id);  IF currentAmtRaised + NEW.amount <= amtNeeded THEN  UPDATE advertised\_project SET amt\_raised=NEW.amount + amt\_raised WHERE  id=NEW.proj\_id;  RETURN NEW;  ELSE  RAISE 'Your investment has exceeded the amount needed. PLease reduce.';  RETURN NULL;  END IF;  END; $$ LANGUAGE PLPGSQL;  CREATE TRIGGER trigger\_update\_amt\_raised\_when\_insert\_invest  BEFORE INSERT ON invest  FOR EACH ROW  EXECUTE PROCEDURE update\_amt\_raised\_when\_insert\_invest(); |

## Aggregation

Admin is given the rights to gather statistical data about the website. Hence, we use aggregation to allow admin to download a statistical report. The following are some notable examples. For more examples, please refer to generateReport.php in the source code.

### Subquery and Sum

Get the top three investors who invest the most amount of money in total

|  |
| --- |
| WITH rankedtable AS (  SELECT investor, SUM(amount) AS total\_amount,  rank() OVER (ORDER BY SUM(amount) DESC)  FROM invest  GROUP BY investor  )  SELECT investor, total\_amount  FROM rankedtable WHERE rank < 4  ORDER BY rank, investor; |

### Subquery and Union

Get the general statistics count

|  |
| --- |
| SELECT header, count  FROM(  SELECT 'admins' AS header, COUNT(\*), '1' AS idx  FROM member a  WHERE is\_admin = 1  UNION  SELECT 'non-admins', COUNT(\*), '2' AS idx  FROM member m  WHERE is\_admin = 0  UNION  SELECT 'total\_projects', COUNT(\*), '3' AS idx  FROM advertised\_project p  UNION  SELECT 'fully\_funded', COUNT(\*), '4' AS idx  FROM advertised\_project ff  WHERE status = 1  UNION  SELECT 'ongoing', COUNT(\*), '5' AS idx  FROM advertised\_project op  WHERE status = 0  ) AS subq  ORDER BY idx; |

# Screenshots

#### Login

|  |
| --- |
| [Insert screenshot here] |

#### Registration

|  |
| --- |
| [Insert screenshot here] |

#### Explore – Landing page when user first login

|  |
| --- |
| [Insert screenshot here] |

#### Profile

|  |
| --- |
| [Insert screenshot here] |

#### My Projects

|  |
| --- |
| [Insert screenshot here] |

#### My Investments

|  |
| --- |
| [Insert screenshot here] |

#### Admin

|  |
| --- |
| [Insert screenshot here] |