# Lab 3.1: Modelling Many-to-Many Relationships
# IN705 Databases Three

## Introduction

Our Splatter application needs to model “following". Each user will be able to follow and be followed by other users. In general, modelling many-to-many realtionships like this is easy with Rails and ActiveRecord, but this case involves a slightly tricky self join.

Our plan is as follows:

1. Write a migration to create our join table;
2. Add code to our `User` model to describe the relationship;
3. Add methods to our `UserController` to show, add, and list followers;
4. Add routes to the `config/routes.rb` file for our new methods.

## 1 The migration

So far we have generated our mgration files, but in this case we will to write it from scratch. In `db/migrate`, create a new file called `<timestamp>_create_follows.rb`. The timestamp format is YYYYMMDDHHMMSS and should relect the time at which the file is created. It should contain the following:

```
class CreateFollows < ActiveRecord::Migration
  def change
    create_table :follows do |t|
      t.integer :follower_id, index: true
      t.integer :followed_id, index: true
    end
  end
end
```

Since both fields will be used in joins we index them. However, note that even though both fields are intended to be foreign keys, nothing in this code implements foreign key constraints or identifies a primary key. There are a couple of ways to solve this. The most direct method is to call `execute` from our `change` method (right after the `create_table` block and add the SQL to do this. One problem with this approach is that it is not comepletely DBMS-agnostic, but it gets the job done.

There is a view among some Rails developers that constraints should be implemented in the model code rather than the database to avoid spreading our model logic across two locations. While this is not entirely without merit, there probably remains some value in using database constraints. For now, we will leave them off and note that we may wish to include them later.

## 2 Adding followers and followed to our model

We model many-to-many realtionships through join tables with ActiveRecord's `has_and_belongs_to_many` method. In a typical case we need only identify the class at the other end of the relationship, but our case involves a self join, so we need to specify extra options as follows in `app/models/user.rb`.

```
has_and_belongs_to_many :follows,
    class_name: "User",
    join_table: :follows,
    foreign_key: :follower_id,
    association_foreign_key: :followed_id
```

We also need to add the inverse relationship `followed_by` to access the followers of a user. This is done in a very similar way and is left as an exercise.

With this relationship in place, we can add, show, and remove following relationships between users. Given user objects `u1` and `u2`:

```
u1.follows            # returns a list of users followed by u1

u1.follows << u2      # adds u2 to u1's list of followed users

u1.follows.delete(u2) #removes u2 from u1's list
```

## 3 Adding methods to the controller

We need to add four methods to our UserController:

**`show_follows`** uses the `id` value from `params` to return a JSON list (possibly empty) of the users followed by the indicated user. It will be accessed with a URL like `/users/follows/1` using a GET. We have written a similar method before to show a user's splatts.

**`show_followers`** is very similar to `show_follows`, except that it returns a list of followers of a given user.

**`add_follows`** uses the `id` and the `follows_id` values from `params` to add the user with `follow_id` to the list of users followed by the user with `id`. It will be accessed with a URL like `/users/follows` using a POST. The ids are passed in the POST body.

**`delete_follows`** will be accessed with a URL like `/users/follows/1/2` using a DELETE. The method will remove the user with `id` 2 from the list of users followed by the user with `id` 1.

Everything you need to know to write these methods is available in this document or in your `UsersController` class.

## 4 Adding routes

Before we can access our new controller methods with our REST API, we need to add routes to `config/routes.rb` that map HTTP requests controller methods. Since we have not done much with routes, just add the lines below to your routes file, just after the existing routes.

```
get 'users/follows/:id' => 'users#show_follows'
get 'users/followers/:id' => 'users#show_followers'
post 'users/follows' => 'users#add_follows'
delete 'users/follows/:id/:follows_id' => 'users#delete_follows'
```

## 5 Wrapping up

With your new routes in place, start your Rails server and try out the new API features using `curl`. Don't forget to commit your changes to your Git repository.