Friendships Results

This problem in the Friendships exercise shows a special kind of relationship because A can be friends with B; and B can also be friends with A. And also A could be friends with B without B being friends with A.

I looked at the meaning of the Friendships table. It seems each row defines a relationship between a friend seeker and a found friend. In the friendship table, the seeker is “user\_id” and the found friend is “friend\_id”

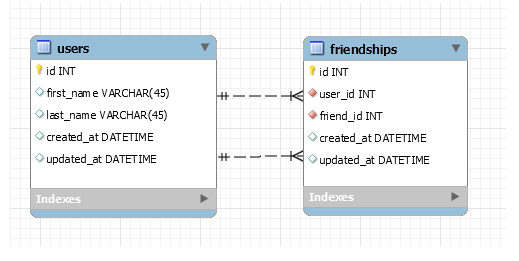
You can see that friend seeker #1 has three friends, one of which has ID=2, but friend seeker #2 also found two friends. For Both columns only have IDs, but we have names. We use the users table to convert, but in this case, it is trickier because we can’t use the same table name twice in the SQL statement

Note also that the table avoids redundancies. If 1 finds 2, then we don’t see an entry for 2 finding one. For human terms, it means that if Kim connects with Lee, then Lee accepts Kim, but can’t find Kim again and have a second friendship

|  |  |  |
| --- | --- | --- |
| # id | user\_id | friend\_id |
| '1' | '1' | '2' |
| '2' | '1' | '3' |
| '3' | '1' | '4' |
| '4' | '2' | '4' |
| '5' | '2' | '5' |
| '6' | '5' | '3' |

Resources:

<https://www.quora.com/MySQL-How-does-a-self-join-work-and-what-does-it-allow-you-to-accomplish>



#In below, we re-use the users table to identify both the friend seeker and the friend for each personal relationship

#However, the SQL can't deal with the same table names being used differently. Solution is to rename one of the instances using the AS keyword

#the first instance users is users and this case is about the person who needs or is seeking a friend

#in the second instance the users table is renamed to user\_as\_found\_friend and in this case users table is about the found friend

select messages.id AS "ID of sender-receiver-relationship" , messages.sender\_id AS "ID of message-sender", messages.receiver AS "ID of receiver"

, users.first\_name AS "Need Friend, First Name", users.last\_name AS "Sender, Last Name"

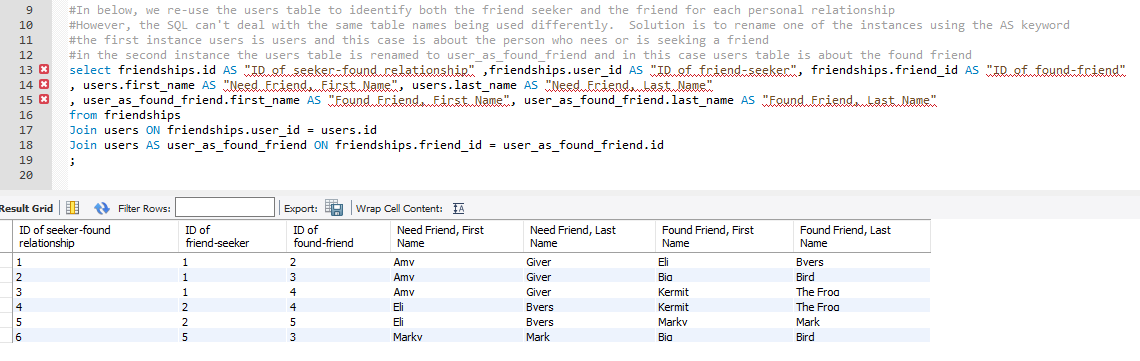
, user\_as\_found\_friend.first\_name AS "Found Friend, First Name", user\_as\_found\_friend.last\_name AS "Found Friend, Last Name"

FROM friendships

Join users ON friendships.user\_id = users.id

Join users AS user\_as\_found\_friend ON friendships.friend\_id = user\_as\_found\_friend.id

;



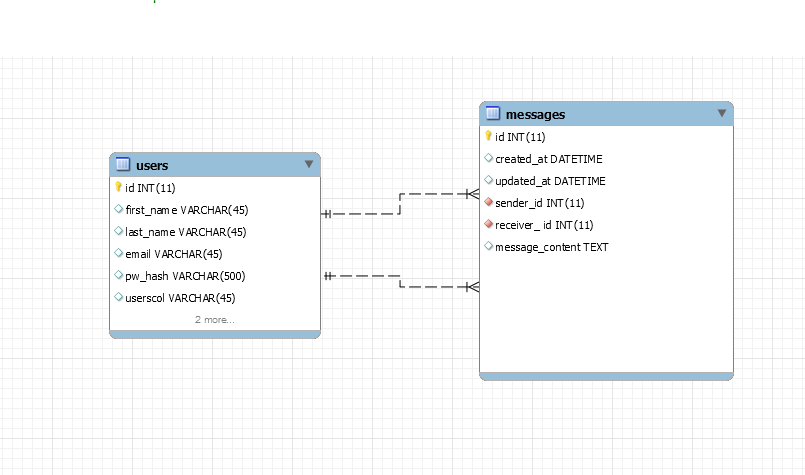
In PW world this works:

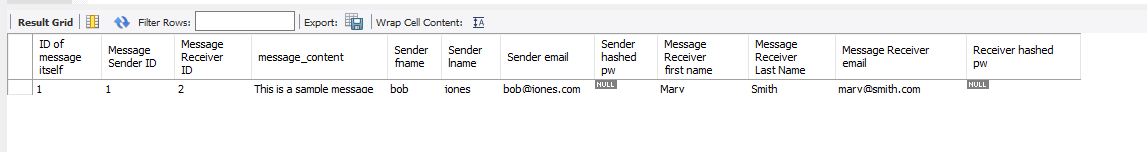
select messages.id as 'ID of message itself', messages.sender\_id AS 'Message Sender ID', receiver\_id AS 'Message Receiver ID', message\_content, users.fname AS 'Sender fname' , users.lname AS 'Sender lname', users.email AS 'Sender email' , users.pw\_hash AS 'Sender hashed pw', user\_as\_message\_receiver.fname AS 'Message Receiver first name', user\_as\_message\_receiver.lname AS 'Message Receiver Last Name', user\_as\_message\_receiver.email AS 'Message Receiver email', user\_as\_message\_receiver.pw\_hash AS 'Receiver hashed pw'

from messages

JOIN users on messages.sender\_id =users.id

JOIN users AS user\_as\_message\_receiver ON messages.receiver\_id = user\_as\_message\_receiver.id





I asked Wes what field name to use if I have such an above double join query should I use the ASed renamed columns and he said “yes”

He also challenged me on why I would need all those columns and when I said I would only need a few, he challenged pw\_hash….with the point being that if I have a user id and need the hashed pw, I don’t need to bother with the messages table, so keep it simple

---------===

Getting SQL statements for the messages prepopulate

**Send side, need**

Tables: users only

Fields: first name, user id

Restrictions: where id != logged id from cookie

Order by users.fname asc

Solution is below where 3 is the proxy for logged in id

select fname, users.id

from users

where users.id != 3

order by fname asc;

Receive side, need

Tables: messages, and users (just sender side)

Fields:

* messages.message.content
* sendersfname (or users.fname AS 'Senderfname')
* messages.created\_at
* messages.id (so it can be deleted)…this goes into the anchor deletion tag…in case clicked

Restrictions: messages.sender\_id != logged id from cookie

Restrictions: messages.receiver\_id = logged id from cookie

Order by sendersfname asc

Solution is below where 4 is the proxy for logged in id

select message\_content, users.fname, messages.created\_at, messages.id from messages JOIN users on messages.sender\_id =users.id where messages.receiver\_id = %(ui)s order by users.fname asc;;

**1. Users who are friends with Kermit**

SELECT friendships.id AS "ID of seeker-found relationship" ,friendships.user\_id AS "ID of friend-seeker", friendships.friend\_id AS "ID of found-friend"

, users.first\_name AS "Need Friend, First Name", users.last\_name AS "Need Friend, Last Name"

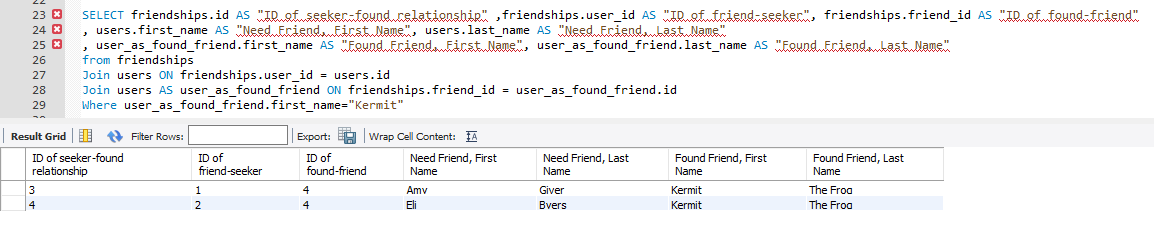
, user\_as\_found\_friend.first\_name AS "Found Friend, First Name", user\_as\_found\_friend.last\_name AS "Found Friend, Last Name"

from friendships

Join users ON friendships.user\_id = users.id

Join users AS user\_as\_found\_friend ON friendships.friend\_id = user\_as\_found\_friend.id

Where user\_as\_found\_friend.first\_name="Kermit"



2. count of all friendships

SELECT count(friendships.id) AS "friendships count"

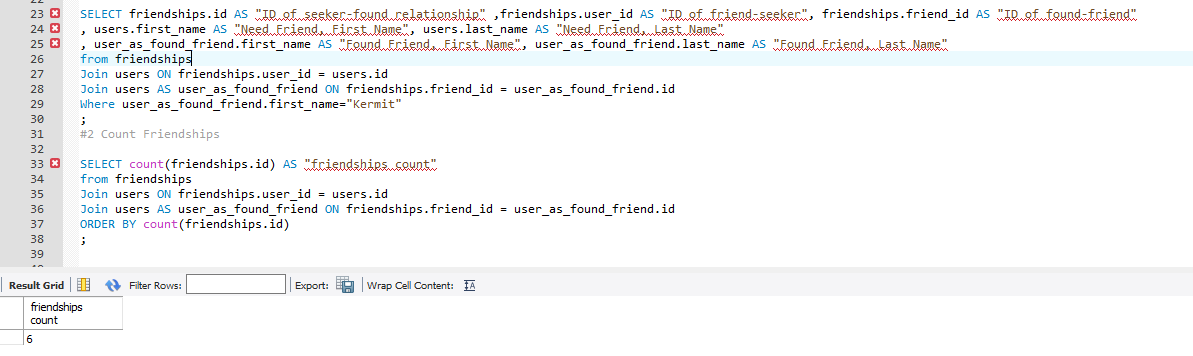
from friendships

Join users ON friendships.user\_id = users.id

Join users AS user\_as\_found\_friend ON friendships.friend\_id = user\_as\_found\_friend.id

ORDER BY count(friendships.id)

;



#3 Who has the most friends  
  
select user\_id,count(user\_id) AS cnt

FROM friendships

GROUP BY user\_id

HAVING cnt=(SELECT COUNT(user\_id) as count

FROM friendships

GROUP BY user\_id

ORDER BY count desc limit 1);

#4 Create a new user and make them friends with Eli Byers, Kermit The Frog, and Marky Mark

insert into users (first\_name, last\_name, created\_at, updated\_at)

values ("Joe", "Johnson", now(), now());

insert into friendships (user\_id, friend\_id, created\_at, updated\_at)

values (6,2, now(), now());

insert into friendships (user\_id, friend\_id, created\_at, updated\_at)

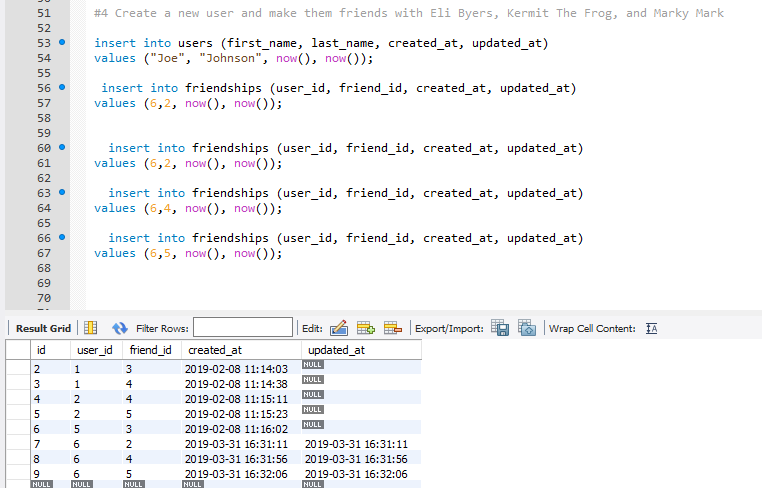
values (6,2, now(), now());

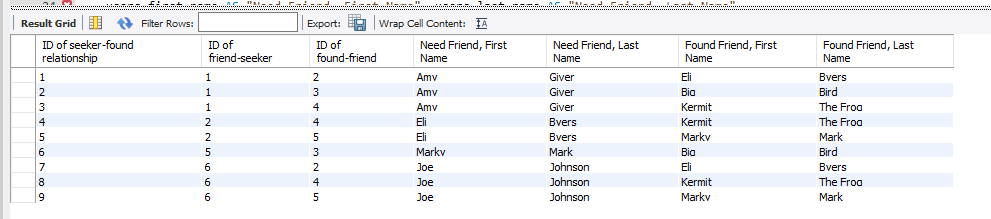
insert into friendships (user\_id, friend\_id, created\_at, updated\_at)

values (6,4, now(), now());

insert into friendships (user\_id, friend\_id, created\_at, updated\_at)

values (6,5, now(), now());





#5Return the friends of Eli in alphabetical order

SELECT friendships.id AS "ID of seeker-found relationship" ,friendships.user\_id AS "ID of friend-seeker", friendships.friend\_id AS "ID of found-friend"

, users.first\_name AS "Need Friend, First Name", users.last\_name AS "Need Friend, Last Name"

, user\_as\_found\_friend.first\_name AS "Found Friend, First Name", user\_as\_found\_friend.last\_name AS "Found Friend, Last Name"

from friendships

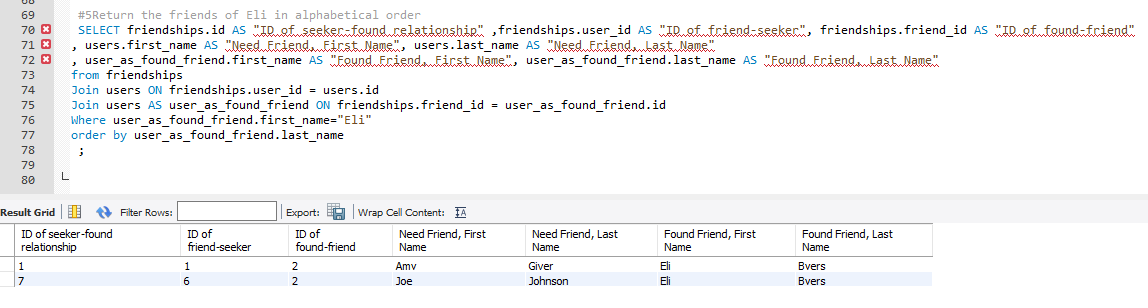
Join users ON friendships.user\_id = users.id

Join users AS user\_as\_found\_friend ON friendships.friend\_id = user\_as\_found\_friend.id

Where user\_as\_found\_friend.first\_name="Eli"

order by user\_as\_found\_friend.last\_name

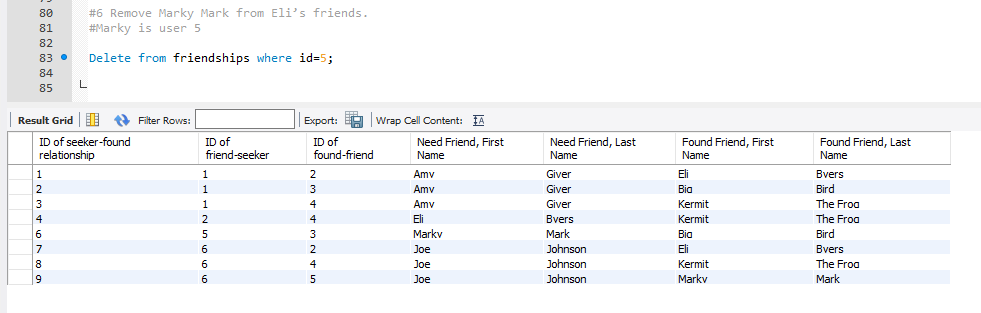
;



#6 Remove Marky Mark from Eli’s friends.

#Marky is user 5

Delete from friendships where id=5;



#7 Return all friendships, displaying just the first and last name of both friends

SELECT users.first\_name AS "Need Friend, First Name", users.last\_name AS "Need Friend, Last Name"

, user\_as\_found\_friend.first\_name AS "Found Friend, First Name", user\_as\_found\_friend.last\_name AS "Found Friend, Last Name"

from friendships

Join users ON friendships.user\_id = users.id

Join users AS user\_as\_found\_friend ON friendships.friend\_id = user\_as\_found\_friend.id

;

