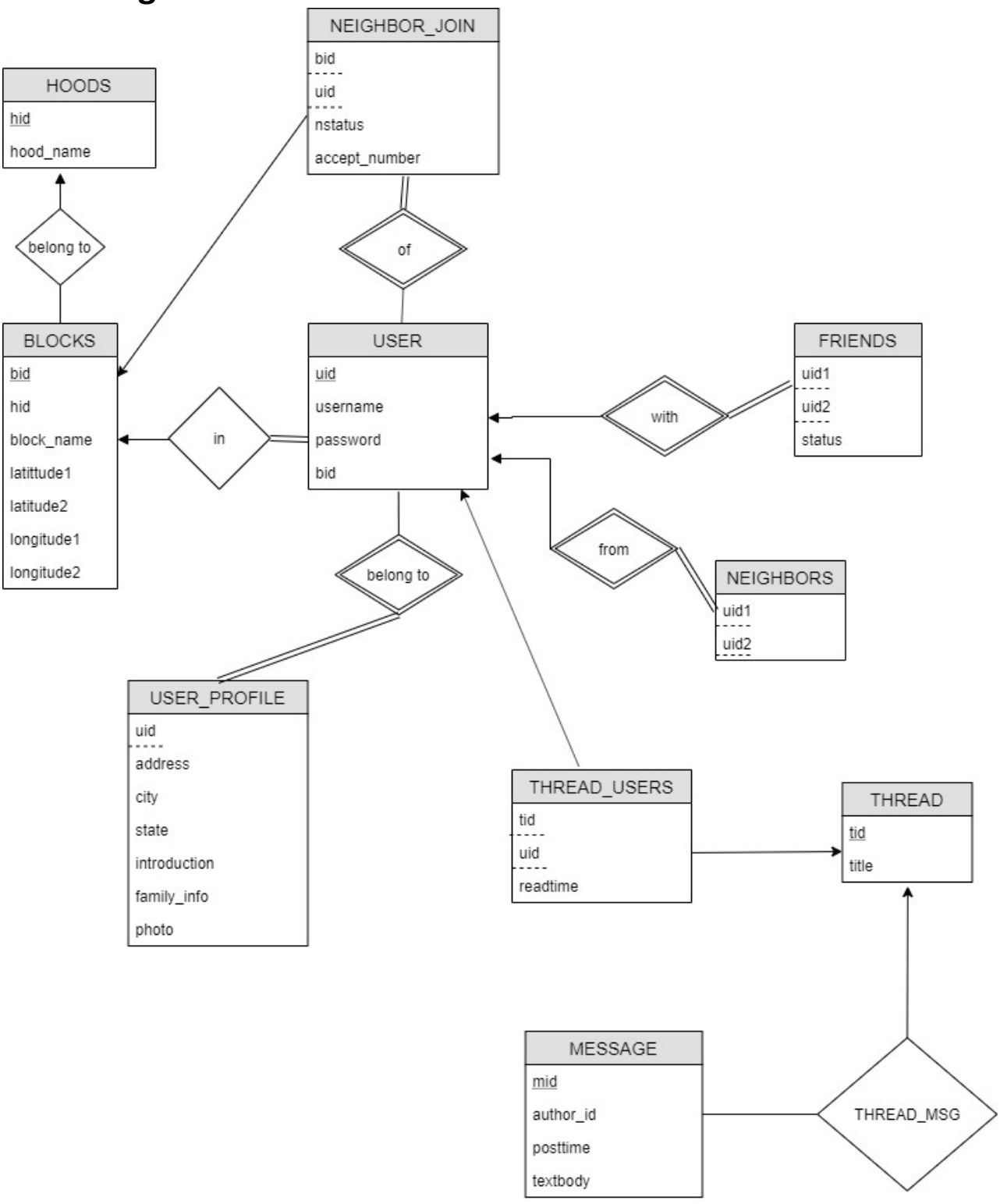
Project
part 1

Jiachen Zhou
Weilu Wang

Introduction and Description

This project is about designing and building a relational backend for a website just like nextdoor.com. We then signed up for an account on that website and got familiar with the basic functions. This made us understand the general concept. Since every member should belong to a block, and several blocks belongs to a hood, so we first started to work on the hood table. Since there is little requirement about hoods, we just add the name of hood. For the block, we also add the name of the block and also the hood it belongs to. By definition, it is defined by two corner points, so we use the latitudes and longitudes of those two points to store the range of each block. For each new user in block, since he or she needs approvals from other members who are already in that block, we build the neighbor join table, which contains user ID, the status of approval (not approved is 0, approved is 1) and also the accepted number, which counts how many approvals has already been received. For the friend table, we use uid1 and uid2 to store the friendship members, and the fstatus to represents the friendship status. If user 1 sends a friend request to user 2, fstatus is 1. If user 2 sends a friend request to user 1, fstatus is 2. If one approves the request from another, fstatus is 3. For the neighbor table we just use user 1 and user 2 to store the relationship of two users who are in the same hood. Furthermore, users can create a thread and send messages. Then we create a thread table. We add the title in the table because for each thread, there is only one title that is created by the first message. Also, since there is a permission for reading a thread, we create a thread users table to store which users can read the messages and the last time they read the messages. To clarify the last read time, it is defined by the time that the user clicks the thread or the time that the user sends a message in the thread. It is updated each time the user performs the above actions. For the message table, we use author ID, posttime and textbody to store the ID of the author, the time he or she posts or replies and the content of the message. Finally, the Thread message table is used to categories each message, it helps us to find which thread the message belongs to. In conclusion, this is a general view of our project and there are more information and details provided in the below sections.

a. ER Diagrams



b. Tables and constraints

USERS (uid, bid, username, pwd, email)

USER_PROFILE (uid, address, city, state, introduction, family_info, photo_url)
Foreign key uid references USERS.uid

HOODS (hid, hood_name)

BLOCKS (bid, hid, block_name, latitude1, latitude2, longitude1, longitude2)
Foreign key hid references HOODS.hid

NEIGHBOR_JOIN (bid, uid, nstatus, accept_number)
Foreign key hid references BLOCKS.bid
Foreign key uid references USERS.uid

FRIENDS (uid1, uid2, fstatus)
Foreign key uid1,uid2 references USERS.uid

NEIGHBORS (uid1, uid2)
Foreign key uid1,uid2 references USERS.uid

THREAD(tid, title)

THREAD_USERS(tid, uid, last_read_time)
Foreign key tid references THREAD.tid
Foreign key uid references USERS.uid

MESSAGE(mid, author_id, posttime, textbody)
Foreign key author_id references THREAD_USERS.uid

THREAD_MSG(tid, mid)
Foreign key mid references MESSAGE.mid

c. Queries

1.
<1>
```
INSERT INTO USERS(username, pwd, email)
SELECT * FROM (SELECT  'ed',  '123', 'ed@gmail.com') as temp
WHERE NOT EXISTS (SELECT username FROM USERS WHERE username ='ed');
```
<2>
```
INSERT NEIGHBOR_JOIN (bid, uid, status , accept_number) VALUES(1, 7, 0, 0)

if one in the same hood guaranteed:
UPDATE NEIGHBOR_JOIN SET accept_number=  accept_number + 1 WHERE bid = 1 AND uid = 7;

check if  he is legal to become members of a block:
```

first we'll check how many members in the block

SELECT count(uid)as count FROM USERS WHERE bid = 1;

then check out if it is legal everytime after guaranteed from frontend,
if count < 3, accept_number == count
if count >= 3, accept_number == 3

UPDATE USERS SET bid = 1
WHERE uid = 7;

UPDATE NEIGHBOR_JOIN SET status = 1
WHERE bid = 1 AND uid = 7;

<3>

INSERT INTO USER_PROFILE(uid, address, city, state, introduction, family_info, photo_url) VALUES(7, '236 Linvingston St', 'Brooklyn', 'New York', 'hello', 'father mother', 'aaa.com') ON DUPLICATE KEY UPDATE
address = '236 Linvingston St',city = 'Brooklyn', city = 'New York', introduction = 'hello', family_info = 'father mother', photo_url = 'aaa.com';

2.

<1>

When a user starts a new thread by posting an initial message,
the database creates a thread first, which is like:

INSERT INTO THREAD(title) VALUES
('title3');
(Here, assume the thread.tid = 3)

After this step, the database should authorize the group who can see this thread depending on which group the user selects:

For example, here the user 1 wants to create a message that only opens to user 2, in this case:

INSERT INTO THREAD_USERS(tid, uid) VALUES
(3, 1),
(3, 2);

Then the database should store the message with its author, the post time, and the text body:

INSERT INTO MESSAGE(author_id, posttime, textbody) VALUES
(1, '2019-11-07 05:25:00', 'I want to eat ice-cream');

Finally the database should add this message into the belonging thread and then update the last read time:

INSERT INTO THREAD_MSG VALUES
(3, 5);

UPDATE THREAD_USERS SET last_read_time = '2019-11-07 05:25:00'
WHERE tid = 3 and uid = 1;

<2>

When a user replies to a message, the database should update the last read time first:

```
UPDATE THREAD_USERS SET last_read_time = '2019-11-07 05:25:00'
WHERE tid = 3 and uid = 2;
```

Then the database should add the message when it replies

```
INSERT INTO MESSAGE(author_id, posttime, textbody) VALUES
(2, '2019-11-08 6:10:00', 'me too');
```

Then the database should update the last read time and add this message into the belonging thread:

```
UPDATE THREAD_USERS SET last_read_time = '2019-11-08 06:10:00'
WHERE tid = 3 and uid = 2;

INSERT INTO THREAD_MSG VALUES
(3, 6);
```

3.

<1>

step1:userA go to check out userB friend status

```
SELECT fstatus FROM FRIENDS WHERE uid1 = “userA_id” AND uid2 = “userB_id”
```

if no matched data, it means they are not friend yet
step: add friend
there are three status type in fstatus:
1: uid1 send request to uid2
2: uid2 send request to uid1
3: one of them answered the request and become friends
If no matched data exist, userA and userB are not friends and never send request before

check out whether userA and userB are in the same hood:

```
step2: SELECT hid FROM USERS natural join BLOCKS WHERE uid = userA_id OR uid = userB_id;
```

step3: userA add userB:

```
INSERT FRIENDS(uid1,uid2,fstatus) values(userA_id, userB_id, 1)
```

step4: userB accept userA’s request:

```
UPDATE FRIENDS SET fstatus = 3 WHERE uid1 = “userA_id” and uid2 = “userB_id”
```

<2>

List all friends of userA:

```
SELECT uid2 FROM FRIENDS WHERE uid1 = “userA_id”
UNION
SELECT uid1 FROM FRIENDS WHERE uid2 = “userA_id”
```

<3>

userA specify userB as his neighbor

INSERT INTO NEIGHBORS(uid1,uid2)VALUES (userA,userB);

<4>

list all neighbors of user A

SELECT uid2 FROM NEIGHBORS WHERE uid1 = userA_id;

4.

<1>

list all threads which user 1 has not read since last time

SELECT uid, tid, last_read_time, posttime
FROM thread natural join thread_users natural join message natural join thread_msg
WHERE uid = 1 and last_read_time < posttime;

<2>

list all messages containing the words "movie" across all feeds that the user can access

SELECT * FROM THREAD natural join THREAD_MSG natural join THREAD_USERS natural join MESSAGE
WHERE uid = 1 and textbody LIKE '%movie%';

<3>

list threads that both user 1 and user 2 has permission to read and reply

SELECT t2.tid
FROM
(SELECT DISTINCT tid
FROM USERS natural join THREAD natural join THREAD_MSG natural join THREAD_USERS natural join MESSAGE
WHERE uid = 1) as t1 join THREAD_USERS as t2
WHERE t1.tid = t2.tid and uid = 2;

d. Tests on sample data

queries from question c:

1.

test<1>: sign up

before ed signup

	uid	bid	username	pwd	email
1	1	1	Lucy	blablabla	lucy@gmail.com
2	2	2	Joy	heiheihei	joy@gmail.com
3	3	2	Nancy	hahaha	nancy@gmail.com
4	4	1	Tom	hohoho	tom@gmail.com
5	5	1	Jack	yiyiyi	jack@gmail.com
6	6	1	Bobby	lololo	bobby@gmail.com

username ed after signup with uid 7

```
INSERT INTO USERS(username, pwd, email)
SELECT * FROM (SELECT  'ed',  '123', 'ed@gmail.com') as temp
WHERE NOT EXISTS (SELECT username FROM USERS WHERE username ='ed');

SELECT * FROM USERS;
```

Output | Result 4-2

7 rows | Tab-se...d (TSV)

	uid	bid	username	pwd	email
1	1	1	Lucy	blablabla	lucy@gmail.com
2	2	2	Joy	heiheihei	joy@gmail.com
3	3	2	Nancy	hahaha	nancy@gmail.com
4	4	1	Tom	hohoho	tom@gmail.com
5	5	1	Jack	yiyiyi	jack@gmail.com
6	6	1	Bobby	lololo	bobby@gmail.com
7	7	<null>	ed	123	ed@gmail.com

test<2>: apply to become members of a block

ed apply to become a member of block1

```
INSERT NEIGHBOR_JOIN (nid, uid, status , accept_number) VALUES(1, 7, 0, 0);

SELECT * FROM NEIGHBOR_JOIN;
```

Output | Result 7

1 row | Tab-se...d (TSV)

	nid	uid	status	accept_number
1	1	7	0	0

when a member in guaranteed ed's application

```
UPDATE NEIGHBOR_JOIN SET accept_number=  accept_number + 1 WHERE nid = 1 AND uid = 7;

SELECT * FROM NEIGHBOR_JOIN;
```

Output | Result 8-2

1 row | Tab-se...d (TSV)

	nid	uid	status	accept_number
1	1	7	0	1

check how many members in this block

Output | cou

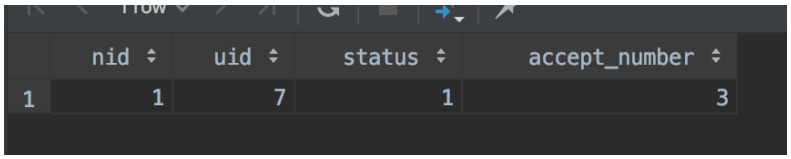
1 row

	count
1	4

if count < 3, accept_number == count to become legal member

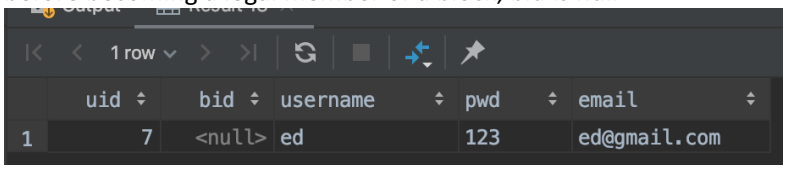
if count >= 3, accept_number == 3 to become legal member

set status to 1



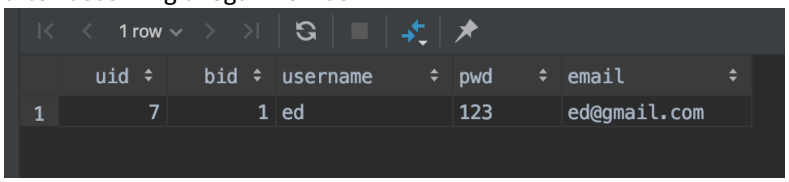
	nid	uid	status	accept_number
1	1	7	1	3

before becoming a legal member of a block, bid is null



	uid	bid	username	pwd	email
1	7	<null>	ed	123	ed@gmail.com

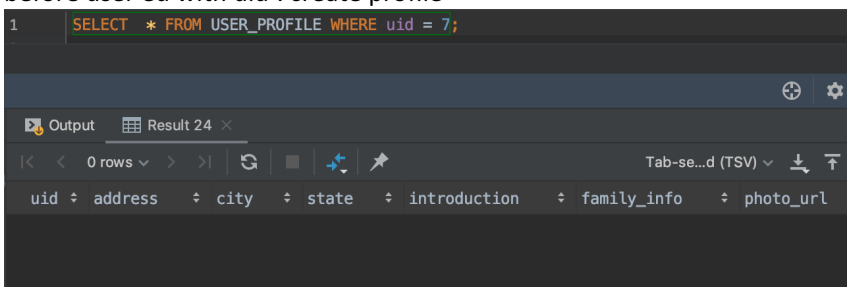
after becoming a legal member



	uid	bid	username	pwd	email
1	7	1	ed	123	ed@gmail.com

test<3>: create or edit their profiles

before user ed with uid 7create profile

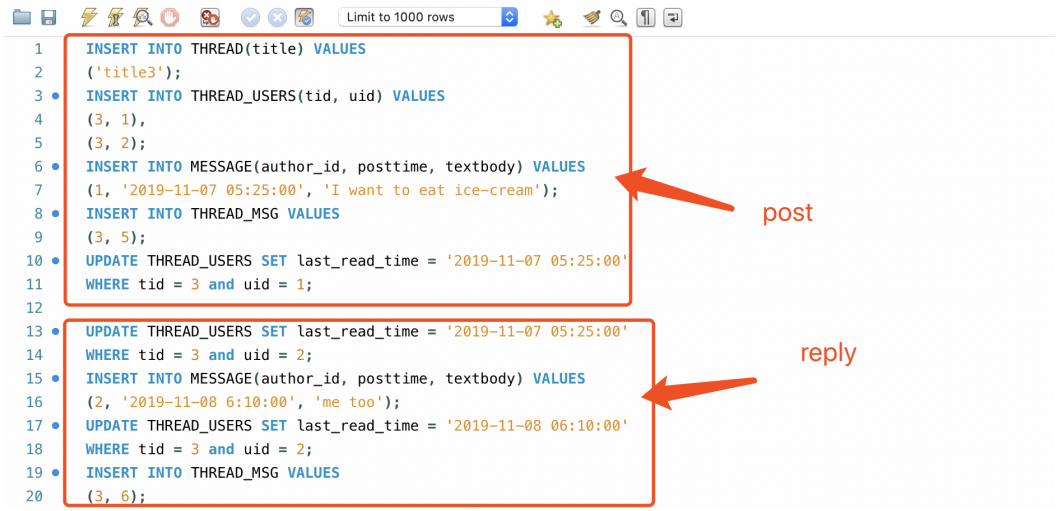


after create his profile

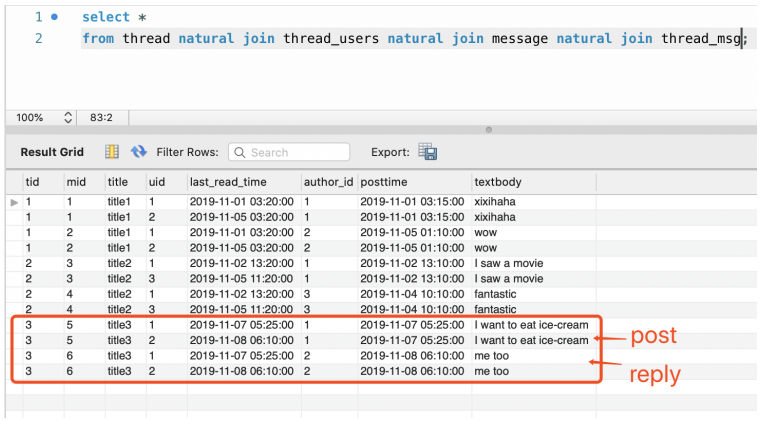
after edit his profile

2.

test<1>: post and reply (the whole explanation is already in (c))



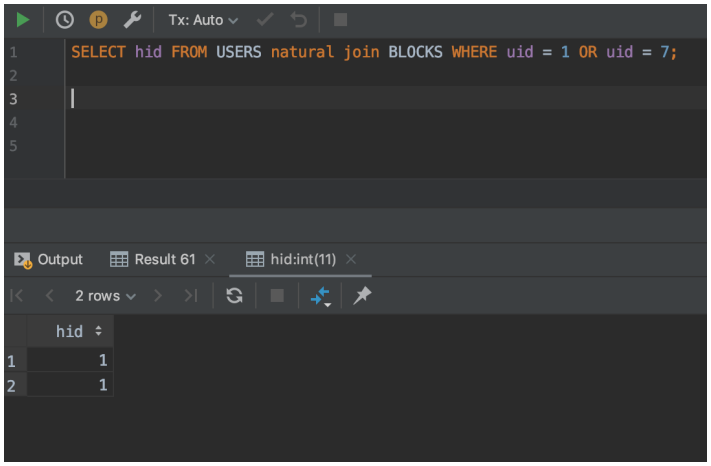
after that:



3.

test<1>add friend

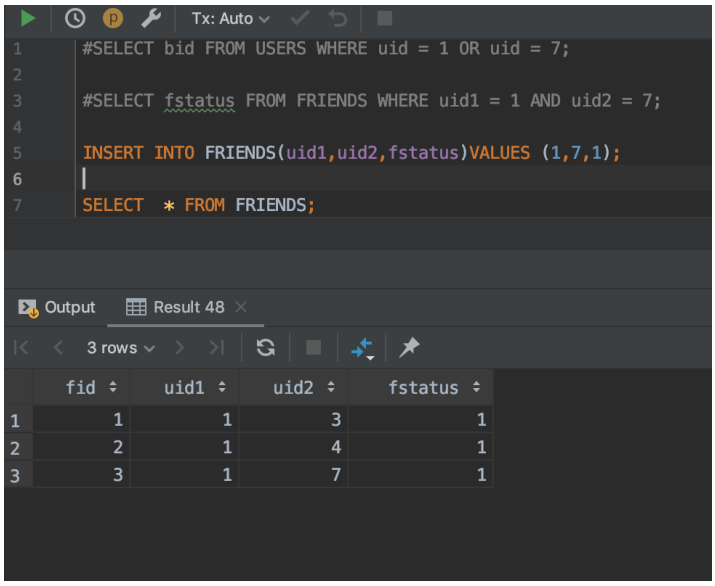
check out if two userA with uid 1 and userB with uid 7 in the same hood



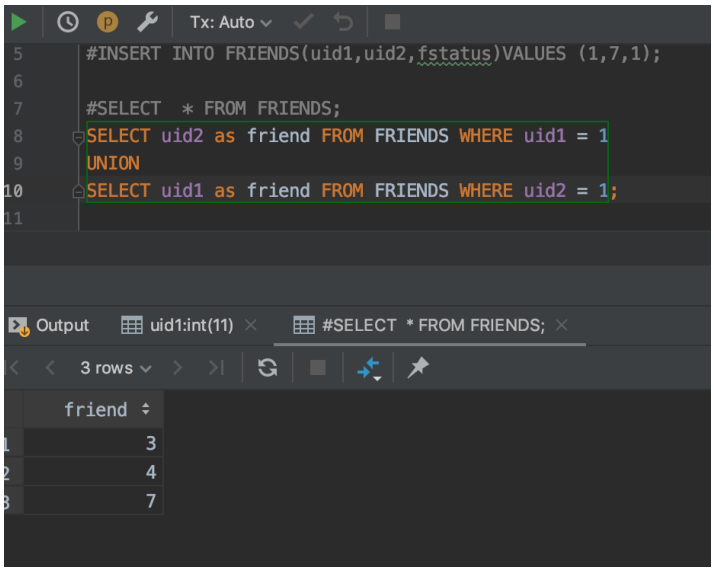
userA send request to userB

fstatus 1 means uid1 send request to uid2

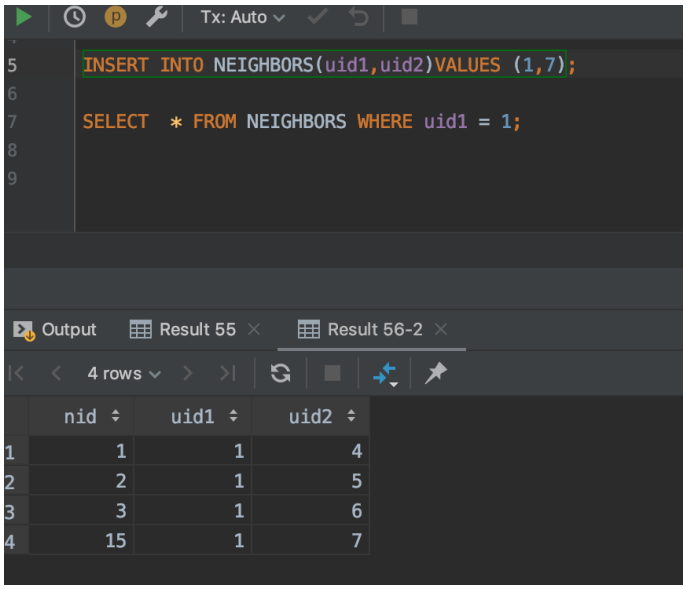
fstatus 2 means uid2 send request to uid1
fstatus 3 means uid1 and uid2 become friends



test<2>:list all their current friends



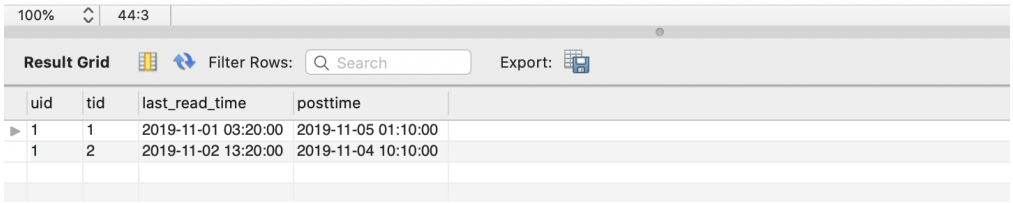
test<3>:add neighbor and list all neighbors of user uid 7



4.

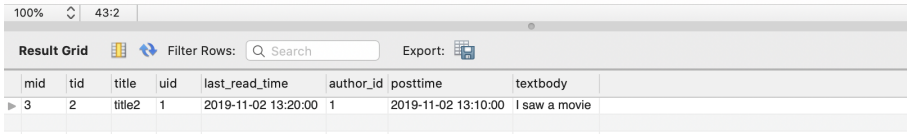
test<1>: list all threads which user 1 has not read since last time

```
select uid, tid, last_read_time, posttime
from thread natural join thread_users natural join message natural join thread_msg
where uid = 1 and last_read_time < posttime
```



test<2>: list all messages containing the words “movie” across all feeds that the user can access.

```
SELECT * FROM THREAD natural join THREAD_MSG natural join THREAD_USERS natural join MESSAGE
WHERE uid  = 1 and textbody LIKE '%movie%'
```



test<3>: list threads that both user 1 and user 2 has permission to read and reply

```
1 •  SELECT t2.tid
2    FROM
3    (SELECT DISTINCT tid
4    FROM USERS natural join THREAD natural join THREAD_MSG natural join THREAD_USERS natural join MESSAGE
5    where uid = 1) as t1 join THREAD_USERS as t2
6    where t1.tid = t2.tid and uid = 2
```

100% 34:6

Result Grid Filter Rows: Search Export:

tid
1
3

e. test data

USERS TABLE

```
1 •  select * from users
```

100% 20:1

Result Grid Filter Rows: Search

uid	bid	username	pwd	email
1	1	Lucy	blablabla	lucy@gmail.com
2	2	Joy	heiheihei	joy@gmail.com
3	2	Nancy	hahaha	nancy@gmail.com
4	1	Tom	hohoho	tom@gmail.com
5	1	Jack	yiyiyi	jack@gmail.com
6	1	Bobby	lololo	bobby@gmail.com
NULL	NULL	NULL	NULL	NULL

USER_PROFILE TABLE

```
1 •  select * from user_profile
```

100% 27:1

Result Grid Filter Rows: Search Edit:

uid	address	city	state	introducti...	family_info	photo_url
1	100 Hoyt St	Brooklyn	New York	111~~~	intro1	www.1.com
2	33 Livingston St	Brooklyn	New York	222~~~	intro2	www.2.com
3	50 Livingston St	Brooklyn	New York	333~~~	intro3	www.3.com
4	11 Schemerhorn St	Brooklyn	New York	444~~~	intro4	www.4.com
5	90 Hoyt St	Brooklyn	New York	555~~~	intro5	www.5.com
6	80 Hoyt St	Brooklyn	New York	666~~~	intro6	www.6.com
NULL	NULL	NULL	NULL	NULL	NULL	NULL

HOODS TABLE

```
1 •  select * from hoods
```

100% 20:1

Result Grid Filter Rows: Search

hid	hood_name
1	Downtown Hoyt
2	Livingston Clinton
3	Boerum Hill
4	Fulton Ashland
5	Tillary Adams
NULL	NULL

BLOCKS TABLE

```
1 •  select * from blocks
```

100% 20:1

Result Grid Filter Rows: Search Edit:

bid	hid	block_name	latitude1	latitude2	longitude1	longitude2
1	1	Hoyt	100.33	112.13	230.12	240.24
2	2	Schemerhorn	90.3	100.33	230.12	240.24
3	2	Livingston	140.32	160.67	230.12	240.24
4	2	Stone	112.32	140.32	230.12	240.24
5	3	Pacfic	80.32	90.3	210.53	230.12
6	3	Ocean	80.32	90.3	210.53	230.12
NULL	NULL	NULL	NULL	NULL	NULL	NULL

NEIGHBOR_JOIN TABLE

```
1 •  select * from neighbor_join
```

100% 28:1

Result Grid Filter Rows: Search

bid	uid	nstatus	accept_number
NULL	NULL	NULL	NULL

NEIGHBORS TABLE

```
1 •  select * from neighbors
```

100% 24:1

Result Grid Filter Rows: Search

uid1	uid2
4	1
5	1
6	1
3	2
2	3
1	4
5	4
6	4
1	5
4	5
6	5
1	6
4	6
5	6
NULL	NULL

FRIENDS TABLE

```
1 •  select * from friends
```

100% 22:1

Result Grid Filter Rows: Search

uid1	uid2	fstatus
1	4	1
1	5	3
1	6	2
4	5	2
4	6	3
5	6	3
NULL	NULL	NULL

THREAD TABLE

```
1 •  select * from thread
```

100% 21:1

Result Grid Filter Rows:

tid	title
1	title1
2	title2
NULL	NULL

THREAD_USERS TABLE

```
1 •  select * from thread_users
```

100% 27:1

Result Grid Filter Rows: Search

tid	uid	last_read_time
1	1	2019-11-01 03:20:00
1	2	2019-11-05 03:20:00
2	1	2019-11-02 13:20:00
2	3	2019-11-05 11:20:00
NULL	NULL	NULL

MESSAGE TABLE

```
1 •  select * from message
```

100% 22:1

Result Grid Filter Rows: Search

mid	author_id	posttime	textbody
1	1	2019-11-01 03:15:00	xixihaha
2	2	2019-11-05 01:10:00	wow
3	1	2019-11-02 13:10:00	I saw a movie
4	3	2019-11-04 10:10:00	fantastic
NULL	NULL	NULL	NULL

THREAD_MSG TABLE

```
1 •  select * from thread_msg
```

100% 22:1

Result Grid Filter Rows: Searc

tid	mid
1	1
1	2
2	3
2	4
NULL	NULL