Vin Liu
Qi Zhang
Professor Torsten Suel
CS-GY 6083
3 December 2019

Final Project Part 1

In this document, we derive a relational schema for a web application that supports location-based user communities. The web application allows a user to register an account to access the services and build up social networks within two levels of locality, hoods, and blocks. A user can add friends and neighbors in his/her social networks and send direct messages with them. A user can also post and reply messages within a community of a block or a hood to make his or her messages visible to all the community members. A user can filter and display incoming messages with various scopes. We will explain how we achieve those core functional requirements as well as many user scenarios in this document.

The first step is the logical design. We started from ER model. we modelled seven entities as well as the relationships associate among them. Then used ER diagram to express the overall logical structure of our database. The ER diagram is attached below:

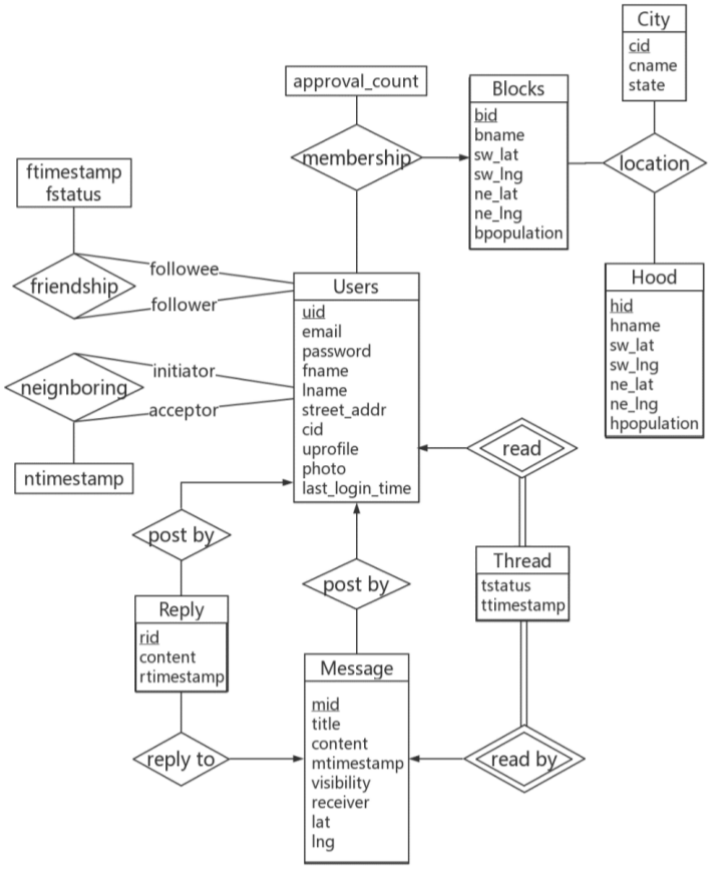


Figure 1 ER diagram

Then we translated the above ER schemas into relational schemas:

- Users (uid, email, pword, fname, lname, street_addr, *cid*, uprofile, photo, last_login_timestamp)
- Hood (hid, hname, sw_lat, sw_lng, ne_lat, ne_lng, hpopulation)
- Blocks (bid, bname, sw_lat, sw_lng, ne_lat, ne_lng, bpopulation)
- Location (bid, hid, cid)
- Membership (uid, bid, approval_count)
- Friendship (follower, followee, ftimestamp, fstatus)
- Neighboring (initiator, acceptor, ntimestamp)
- Message (mid, *author*, title, content, mtimestamp, visibility, *receiver*, lat, lng)
- Reply (rid, *mid*, *author*, content, rtimestamp)
- Thread (*uid*, *mid*, tstatus, ttimestamp)
- City (cid, cname, state)

In the schemas above, primary key is labelled with underline, and foreign key is in *italic*.

Next, we are going to claim the constraints, assumptions and justifications we made for the schemas.

Users	
uid	INT(10), UNSINGED, PRIMARY KEY
email	VARCAHR(45), NOT NULL
pword	VARCAHR(45), NOT NULL
fname	VARCAHR(45), NOT NULL
lname	VARCAHR(45), NOT NULL
street_addr	VARCAHR(45), NOT NULL
cid	INT(10), FOREIGN KEY referencing cid in City
uprofile	VARCAHR(140)
photo	VARCAHR(45)
last_login_timestamp	TIMESTAMP, NOT NULL, DEFAULT CURRENT_TIMESTAMP, ON UPDATE CURRENT_TIMESTAMP

- uid is auto-generated and auto-incremented number, serves as a unique identifier of a user. It is generated upon user registration. The value is not recycled when a user is deleted.
- A user does not know his or her uid. An email directly serves as a username. A user registers and logins via his or her email and password.
- fname and lname store the first name and last name of a user, respectively.
- street_addr stores street address of a user, which be manually entered by the user or by selected on a map upon user registration.
- uprofile stores a profile, like a bio, for a user. It is nullable.
- photo stores an avatar for a user. It is nullable.
- last_login_timestamp is a timestamp auto-generated and updated every time a user logins in.

Hood	
hid	INT(10), UNSINGED, PRIMARY KEY
hname	VARCAHR(45), NOT NULL
sw_lat	DOUBLE, NOT NULL
sw_lng	DOUBLE, NOT NULL
ne_lat	DOUBLE, NOT NULL
ne_lng	DOUBLE, NOT NULL
hpopulation	INT(11) , NOT NULL

- hid is auto-generated and auto-incremented number, serves as a unique identifier of a hood. The value is not recycled when a user is deleted. According to the project description, we can predefine hoods. Users register into a block and a hood predefined in the system. So, we assume that addition or deletion of a hood is not common.
- hname stores the name of a hood.
- sw_lat, sw_lng, ne_lat, ne_lng store the coordinates of a rectangle defining the region of a hood on a map.
- hpopulation stores the population of a hood. It should be a sum of the population in all blocks within the hood. This can be ensured either by defining a trigger in MySQL, or by calculation done in a server application. This will be implemented in project part 2.

Blocks	
bid	INT(10), UNSINGED, PRIMARY KEY
bname	VARCAHR(45), NOT NULL
sw_lat	DOUBLE, NOT NULL
sw_lng	DOUBLE, NOT NULL
ne_lat	DOUBLE, NOT NULL
ne_lng	DOUBLE, NOT NULL
bpopulation	INT(11) , NOT NULL

- The setting of this schema is similar to Hood.
- bpopulation tracks the population of a block. According to the project description, a user's request to join a block needs to be approved by three members in that particular block or by all members if this block has less than three members. We will need to compare approval_count in Membership with bpopulation to achieve that.

Location	
bid	INT(10), PRIMARY KEY, FOREIGN KEY referencing bid in Blocks
hid	INT(10), PRIMARY KEY, FOREIGN KEY referencing hid in Hood
cid	INT(10), PRIMARY KEY, FOREIGN KEY referencing cid in City

- We use the composite primary key here to handle the problem of repetition of hood names or block names in different cities. For example, names like "High Street", "Main Street", and "Midtown" often appear in many cities.

Membership	
uid	INT(10), PRIMARY KEY, FOREIGN KEY referencing uid in Users
bid	INT(10), PRIMARY KEY, FOREIGN KEY referencing bid in Blocks
approval_count	INT(11), NOT NULL, DEFAULT 0

- The uid and the bid here represent a potential membership, which mean a user is a member of a block. This membership is uniquely identified by the composite primary key.
- As we mentioned in Blocks, we will compare approval_count with bpopulation to determine if this potential membership is pending or passed. Arguably, we could have a status attribute to represent it. We would figure out if it is necessary in project part 2.

Friendship	
follower	INT(10), PRIMARY KEY, FOREIGN KEY referencing uid in Users
followee	INT(10), PRIMARY KEY, FOREIGN KEY referencing uid in Users
ftimestamp	TIMESTAMP, NOT NULL, DEFAULT CURRENT_TIMESTAMP, PRIMARY KEY
fstatus	VALUE IN (‘pending’, ‘rejected’, ‘accepted’), DEFAULT ‘pending’

- follower and followee here are foreign keys referencing to uid in Users. A follower adds a followee to create a potential friendship at the ftimestamp, and a followee can accept the request to confirm the friendship or not.
- Because a friendship request can be made multiple times, we add ftimestamp to identify them. For example, when A adds B, B rejects it, then A adds B again.
- fstatus is set to ‘pending’ when a friendship request is created. A followee will receive a notification and be able to accept or reject the request. Upon acceptance, the follower will receive a notification. Also, when a fstatus is ‘accepted’, the follower and the followee will have access messages with visibility of ‘friend’ from each other.

Neighboring	
initiator	INT(10), PRIMARY KEY, FOREIGN KEY referencing uid in Users
acceptor	INT(10), PRIMARY KEY, FOREIGN KEY referencing uid in Users
ntimestamp	TIMESTAMP, NOT NULL, DEFAULT CURRENT_TIMESTAMP, PRIMARY KEY

- The setting of this schema is similar to Friendship.
- Adding of a neighbor is uniliteral according to the project description. An initiator can add an acceptor to his/her neighbor directly without permission. So, we do not need to mark its status like ‘pending’ or ‘accepted’.

Message	
mid	INT(10), UNSINGED, PRIMARY KEY
author	INT(10), FOREIGN KEY referencing uid in Users
title	VARCAHR(45), NOT NULL
content	VARCAHR(280), NOT NULL
mtimestamp	TIMESTAMP, NOT NULL, DEFAULT CURRENT_TIMESTAMP
visibility	VARCAHR(45), NOT NULL, VALUE IN (‘direct’, ‘friend’, ‘neighbor’, ‘block’, ‘hood’)
receiver	INT(10), FOREIGN KEY referencing uid in Users, NULLABLE, DEFAULT NULL
lat	DOUBLE, NULLABLE
lng	DOUBLE, NULLABLE

- mid is an auto-generated and auto-incremented number, serves as a unique identifier of a message. It is generated upon posting a message. The value is not recycled when a reply is deleted.
- mtimestamp here is for displaying when the message is posted.
- A message has visibility of ‘direct’, ‘friend’, ‘neighbor’, ‘block’, and ‘hood’. When a user chooses to post a direct message, a text field for entering a message receiver is shown. This is done by the front end. Then the receiver is recorded in receiver attribute. However, when visibility is set to be ‘friend’, ‘neighbor’, ‘block’, or ‘hood’, the receiver text field is hidden, and receiver attribute is set to be NULL.
- lat and lng are coordinates where the message is pinned on a map. Users can choose to not pin the message.

Reply	
rid	INT(10), UNSINGED, PRIMARY KEY
mid	INT(10), FOREIGN KEY referencing mid in Message
author	INT(10), FOREIGN KEY referencing uid in Users
content	VARCAHR(280), NOT NULL
rtimestamp	TIMESTAMP, NOT NULL, DEFAULT CURRENT_TIMESTAMP

- rid is an auto-generated and auto-incremented number, serves as a unique identifier of a reply. It is generated upon posting a reply. The value is not recycled when a reply is deleted.
- mid represents that this reply is replying to the message with the particular message.
- rtimestamp here is for displaying when the reply is posted.
- According to the project description, a replay has the same visibility as the original message, so we do not need visibility attribute here.
- According to the project description, we can decide the reply depth on our own judgment. So, we decided the reply depth to be 1, which means all replies are replying to the original message. They are organized by chronological order with the aid of rtimestamp.

Thread	
uid	INT(10), FOREIGN KEY referencing uid in Users, PRIMARY KEY
mid	INT(10), FOREIGN KEY referencing mid in Message, PRIMARY KEY
tstatus	VARCAHR(45), NOT NULL, VALUE IN ('read', 'unread'), DEFAULT 'unread'
ttimestamp	TIMESTAMP, NOT NULL, DEFAULT CURRENT_TIMESTAMP

- A thread here is defined as a piece of information consisting of a message and corresponding replies. In essence, we decide to pre-compute what information a user should be able to see when he or she accesses the timeline news feed. Such pieces of information are identified by uid and mid. Twitter uses pre-computed information for its timeline too. But the information is stored in memory by the aid of Redis. Definitely, Twitter's algorithm will be much more sophisticated. We would like to try a simplified approach here by storing pre-computed timeline in Thread schema.
- By pre-computing the timeline, we mean that when a message is posted, we immediately determine who can read this message. For example, when A post a direct message to B, with A's uid=1, B's uid=2, and mid=3, we add (2, 3, 'unread', CURRENT_TIMESTAMP) to Thread schema. So, when B loads his or her timeline news feed, message with mid=3 will be displayed. Similarly, when A post a message to his or her friends, entries with A's friends' uid will be added. Anyone who is not his or her friend will not even know the message is posted by querying Thread table, and hence the thread is not going to be in his or her timeline. In this way, we pre-compute the timeline for each user. Automatically computing uids and inserting entries into Thread table are done by a server application. We will write it in Python during project part 2.
- tstatus determines whether a user has read the message. It helps when we need to display the unread message only.
- Each time a reply is posted for a message, we automatically set tstatus back to 'unread'. This is done by the same server application. So, the audience will be about to notice the reply.
- ttimestamp can help when we need to display messages or replies after the last login.

City	
cid	INT(10), UNSINGED, PRIMARY KEY
cname	VARCAHR(45), NOT NULL
state	VARCAHR(45), NOT NULL

- cid is an auto-generated and auto-incremented number, serves as a unique identifier of a city. The value is not recycled when a reply is deleted. According to the project description, blocks and hoods are predefined for simplicity. So, we assume cities and states are also predefined. We will retrieve cities and states information from open-source data on the internet. Users select them upon user registration.
- By using cid as a primary key, we can distinguish the cities with the same name but in different states. For example, the name 'Portland' appears in both Oregon and Maine.

We create our schema by using MySQL. The tasks in (c) are done using the following SQL queries:

(1) User Y lives in city X and becomes a member of block Z

Sign up:
INSERT INTO Users(email, pword, fname, lname, street_addr, cid) VALUE
('zl1477@nyu.edu', 'pword', 'Vin', 'Liu', '110 1st St.', X);

Become a member of a block:
INSERT INTO Membership(uid, bid) VALUE (X,Y);

Create or edit profile:
UPDATE Users SET uprofile = 'I love SQL!'
WHERE uid = Y;

(2) User X posts a message U at coordinate (Y, Z). User V replies it.

Post an initial message:
INSERT INTO Message(author, title, content, mtimestamp, visibility, receiver, lat, lng)
VALUE (X, 'Posting test case', 'Hi there! I am testing the c2 by posting a new message',
CURRENT_TIMESTAMP, 'hood', NULL, Y, Z);

Reply a message:
INSERT INTO Reply(mid, author, content, rtimestamp) VALUE
(U, V, 'Hi there! I my testing the c2 by replying a new message',
CURRENT_TIMESTAMP);

(3) User X adds Y as a friend. User X adds Y as a neighbor. User Y accepts the friendship.

Add friend:
INSERT INTO Friendship VALUE
(X, Y, CURRENT_TIMESTAMP, 'pending');

Add neighbor:
INSERT INTO Neighboring VALUE
(X, Y, CURRENT_TIMESTAMP);

Accept friendship:
UPDATE Friendship SET fstatus = 'accepted'
WHERE followee = Y AND follower = X;

List all friends:

```
(SELECT follower FROM Friendship
WHERE followee = X)
UNION
(SELECT followee FROM Friendship
WHERE follower = X);
```

List all neighbors:

```
SELECT accepter FROM Neighboring
WHERE initiator = X;
```

(4) User X

List all threads that have new message since last access:

```
SET @llt = (SELECT last_login_timestamp FROM Users AS u
WHERE u.uid = X);

SELECT mid FROM Thread
        WHERE ttimestamp > @llt AND uid = X;

SELECT * FROM Message
WHERE mid IN (SELECT mid FROM Thread
        WHERE ttimestamp > @llt AND uid = X);

SELECT * FROM Reply
WHERE mid IN (SELECT mid FROM Message
WHERE mid IN (SELECT mid FROM Thread
        WHERE ttimestamp > @llt AND uid = X));
```

List all unread thread in friend feed:

```
SELECT * FROM Message AS w,
(SELECT mid FROM Message
WHERE author IN
        (SELECT follower FROM Friendship
                WHERE followee = X AND fstatus = 'accepted'
        UNION
        SELECT followee FROM Friendship
                WHERE follower = X AND fstatus = 'accepted')
AND visibility = 'friend') AS m
WHERE w.mid IN
(SELECT mid FROM Thread
WHERE uid = X AND tstatus = 'unread') AND m.mid = w.mid;
```

List all "bicycle accident" threads:

```
SELECT * FROM Thread
WHERE uid = X AND mid IN
(SELECT mid FROM Message
WHERE title LIKE '%bicycle accident%' OR content LIKE '%bicycle accident%');
```

We list some sample data below:

Users:

uid	email	pword	fname	lname	street_addr	cid	uprofile	photo	last_login_timesta...
1	ewaters@me.com	pword1	Shaneka	Franck	110 Pikachu Rd	1	NULL	NULL	NULL
2	okroeger@yahoo.com	pword1	Kathry	Grimsley	607 Shady Court	1	NULL	NULL	NULL
3	lstaf@comcast.net	pword1	Rochell	Brigance	7477 Pearl St	1	NULL	NULL	NULL
4	auronen@live.com	pword1	Cami	Silk	56 Marvon St	1	NULL	NULL	NULL
5	grdschl@icloud.com	pword1	Ryan	Dilks	9700 Armstrong St	3	NULL	NULL	NULL
6	inico@sbcglobal.net	pword1	Helen	Uresti	3 Wentworth Dr	3	NULL	NULL	NULL
7	harpes@outlook.com	pword1	Kylee	Deskins	33 Hill St	3	NULL	NULL	NULL
8	mrdvt@gmail.com	pword1	Cristie	Bonnell	37 Holly Road	1	NULL	NULL	NULL
9	dodong@yahoo.com	pword1	Alden	Mee	7569 Grant Ave	3	NULL	NULL	NULL
10	killmenow@optonline.net	pword1	Todd	Carl	68 Oakwood Drive	3	NULL	NULL	NULL
11	vin_lz@outlook.com	pword1	Vin	Liu	110 1st St	3	NULL	NULL	NULL
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

City:

cid	cname	cstate
1	New York	New York
2	White Plains	New York
3	Jersey City	New Jersey
4	Yonkers	New York
5	Hoboken	New Jersey
6	Harrison	New Jersey
7	Weehawken	New Jersey
8	West New York	New Jersey
9	Newark	New Jersey
10	Long Beach	New York
NULL	NULL	NULL

Hood:

hid	hname	sw_lat	sw_lng	ne_lat	ne_lng	hpopulati...
3	Financial District	0	0	0	0	300
4	Two Bridges	0	0	0	0	100
5	SoHo	0	0	0	0	400
6	Bowery	0	0	0	0	2
7	Brooklyn Heights	0	0	0	0	1000
8	Grove Street	0	0	0	0	300
9	Newport	0	0	0	0	1
10	Downtown Newark	0	0	0	0	500
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Blocks:

bid	bname	sw_lat	sw_lng	ne_lat	ne_lng	bpopulati...
3	45 Christopher Street	0	0	0	0	25
4	49 Delancy Street	0	0	0	0	2
5	Newport Center	0	0	0	0	3
6	Cadman Plaza	0	0	0	0	60
7	Prince Street	0	0	0	0	110
8	Provost Square	0	0	0	0	70
9	First Street	0	0	0	0	80
10	Newark Avenue	0	0	0	0	100
11	Warren Street	0	0	0	0	10
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Location:

bid	hid	cid
3	1	1
7	5	1
4	6	1
6	7	1
8	8	3
9	8	3
10	8	3
11	8	3
5	9	3
NULL	NULL	NULL

Membership:

uid	bid	approval_count
1	4	1
2	3	3
3	2	3
4	1	2
5	5	0
6	4	0
7	1	0
8	7	3
9	7	3
10	8	3
11	8	3
NULL	NULL	NULL

Friendship:

follower	followee	ftimestamp	fstatus
3	8	2017-09-21 20:24:00	accepted
11	1	2018-03-07 20:24:00	accepted
11	2	2018-06-07 20:24:00	accepted
11	3	2018-03-09 20:24:00	accepted
11	4	2018-08-07 20:24:00	accepted
11	5	2018-12-07 20:24:00	accepted
11	6	2018-03-01 20:24:00	accepted
11	7	2018-04-08 20:24:00	accepted
11	8	2018-10-10 20:24:00	rejected
11	10	2018-11-03 20:24:00	pending
NULL	NULL	NULL	NULL

Neighboring:

initiator	acceptor	ntimestamp
11	1	2018-03-07 20:24:00
11	2	2018-06-07 20:24:00
11	3	2018-03-09 20:24:00
11	4	2018-08-07 20:24:00
11	5	2018-04-08 20:24:00
11	6	2018-10-10 20:24:00
11	10	2018-11-03 20:24:00
6	11	2018-03-01 20:24:00
5	11	2018-12-07 20:24:00
3	8	2017-09-21 20:24:00
2	3	2019-07-07 20:24:00
NULL	NULL	NULL

Message:

mid	author	title	content	mtimestamp	visibility	receiver	lat	lng
3	10	Brain problem	My right brain has noting left.	2018-03-02 20:00:00	block	NULL	1.2	1.2
4	10	Brain problem	My left brain has nothing right	2018-03-03 20:00:00	block	NULL	1.1	1.1
5	11	Pokemon go fans	Anyone found a pikachu near the new Xmas tree?	2019-11-29 16:00:01	friend	NULL	1.1	1.1
6	11	Hitchhicker for Woodbury shopping	Hi Cami! One spare spot to Woodbury on Friday…	2019-11-20 01:40:49	direct	5	1.1	1.1
7	1	FFVII Remake is coming	Yo, bro! Final Fantasy VII Remake is coming ne…	2019-08-20 01:40:33	direct	11	NULL	NULL
8	5	Street Food Festival	Chilli Daddy has the best noodle soup for you!	2018-01-02 20:00:00	block	NULL	1.1	1.1
9	5	AD: Free cake	Free cake give away at the Plaza	2018-01-02 20:00:01	block	NULL	1.1	1.1
10	5	Street Food Festival Again	Chilli Daddy has the best noodle soup for you!	2018-01-02 21:00:00	friend	NULL	1.1	1.1
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Reply:

rid	mid	author	content	rtimestamp
3	1	10	Sorry to hear that.	2018-03-03 21:00:00
4	7	11	Gotta buy a PS4 for gaming!	2019-08-20 10:41:33
5	6	6	I wish I could go but I'm working on the databas…	2019-11-21 00:00:49
6	5	5	I got it!	2019-11-29 16:00:05
7	5	3	I got it!	2019-11-29 16:00:07
8	3	2	I didn't get it.	2018-03-02 20:20:04
NULL	NULL	NULL	NULL	NULL

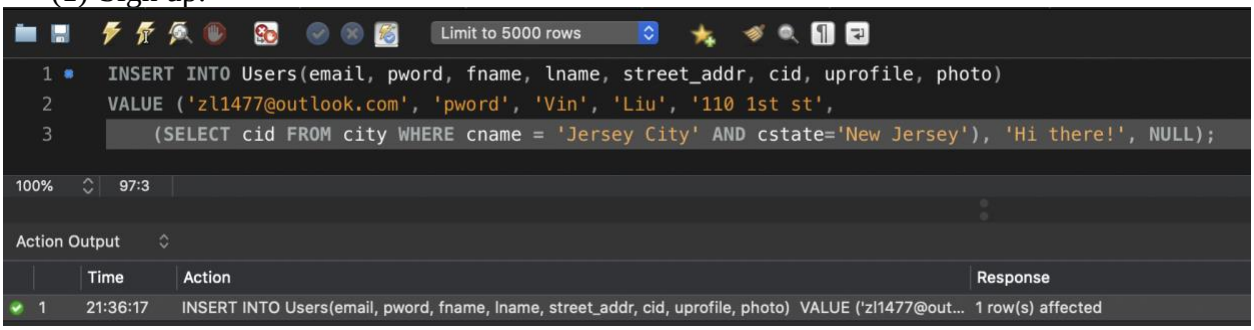
Thread:

uid	mid	tstatus	ttimestamp
3	1	unread	2019-12-04 02:32:36
3	2	unread	2019-12-04 02:32:36
3	3	unread	2019-12-04 02:32:36
3	4	unread	2019-12-04 02:32:36
3	5	unread	2019-12-04 02:32:36
3	6	unread	2019-12-04 02:32:36
3	7	unread	2019-12-04 02:32:36
3	8	unread	2019-12-04 02:32:36
3	9	unread	2019-12-04 02:32:36
5	1	unread	2019-12-04 02:32:36
5	2	unread	2019-12-04 02:32:36
5	3	unread	2019-12-04 02:32:36
5	4	unread	2019-12-04 02:32:36
5	5	unread	2019-12-04 02:32:36
5	6	unread	2019-12-04 02:32:36

Thread 1

Then we test the queries in part (c):

(1) Sign up:



Become a member of a block:

```
1  INSERT INTO Membership(uid, bid) VALUE (1,1);
```

100% 46:1

Action Output

	Time	Action	Response
1	21:37:27	INSERT INTO Membership(uid, bid) VALUE (1,1)	1 row(s) affected

Create or edit profile:

```
1  UPDATE Users SET uprofile = 'I love SQL!'
2  WHERE uid = 2;
```

100% 15:2

Action Output

	Time	Action	Response
1	21:37:49	UPDATE Users SET uprofile = 'I love SQL!' WHERE uid = 2	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0

(2) Post an initial message:

```
1  INSERT INTO Message(author, title, content, mtimestamp, visibility, receiver, lat, lng) VALUE
2  (11, 'Posting test case', 'Hi there! I am testing the c2 by posting a new message', CURRENT_TIMESTAMP, 'hood', NULL, 1.1, 1.1);
```

100% 128:2

Action Output

	Time	Action	Response
1	21:38:43	INSERT INTO Message(author, title, content, mtimestamp, visibility, receiver, lat, lng) VALUE (11, 'Posti...	1 row(s) affected

Reply a message:

```
1  INSERT INTO Reply(mid, author, content, rtimestamp) VALUE
2  (10, 10, 'Hi there!I my testing the c2 by replying a new message', CURRENT_TIMESTAMP);
```

100% 87:2

Action Output

	Time	Action	Response
1	21:39:13	INSERT INTO Reply(mid, author, content, rtimestamp) VALUE (10, 10, 'Hi there!I my testing the c2 by re...	1 row(s) affected

(3) Add friend:

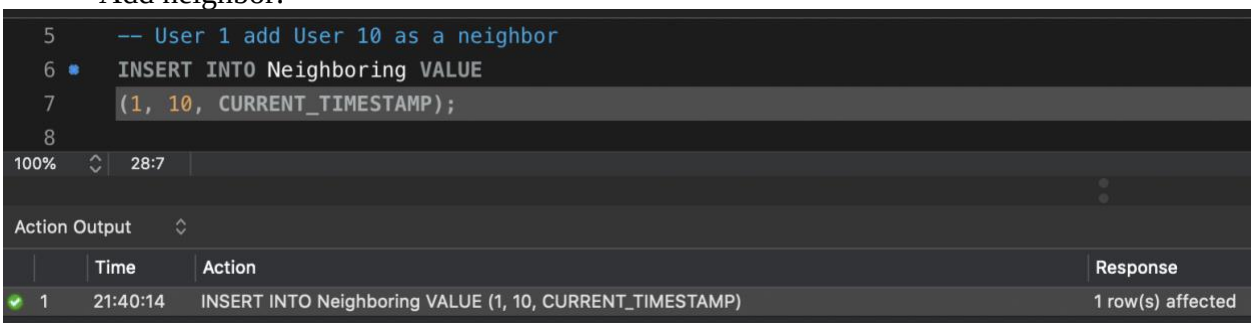
```
1  -- User 1 add User 10 as a friend
2  INSERT INTO Friendship VALUE
3  (1, 10, CURRENT_TIMESTAMP, 'pending');
4
```

100% 39:3

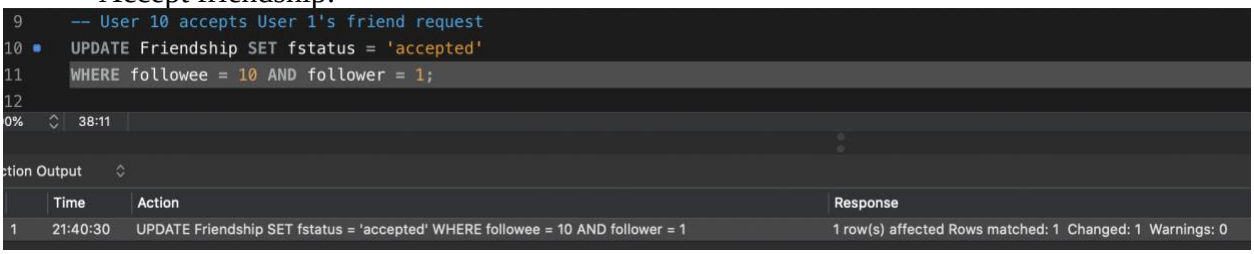
Action Output

	Time	Action	Response
1	21:39:55	INSERT INTO Friendship VALUE (1, 10, CURRENT_TIMESTAMP, 'pending')	1 row(s) affected

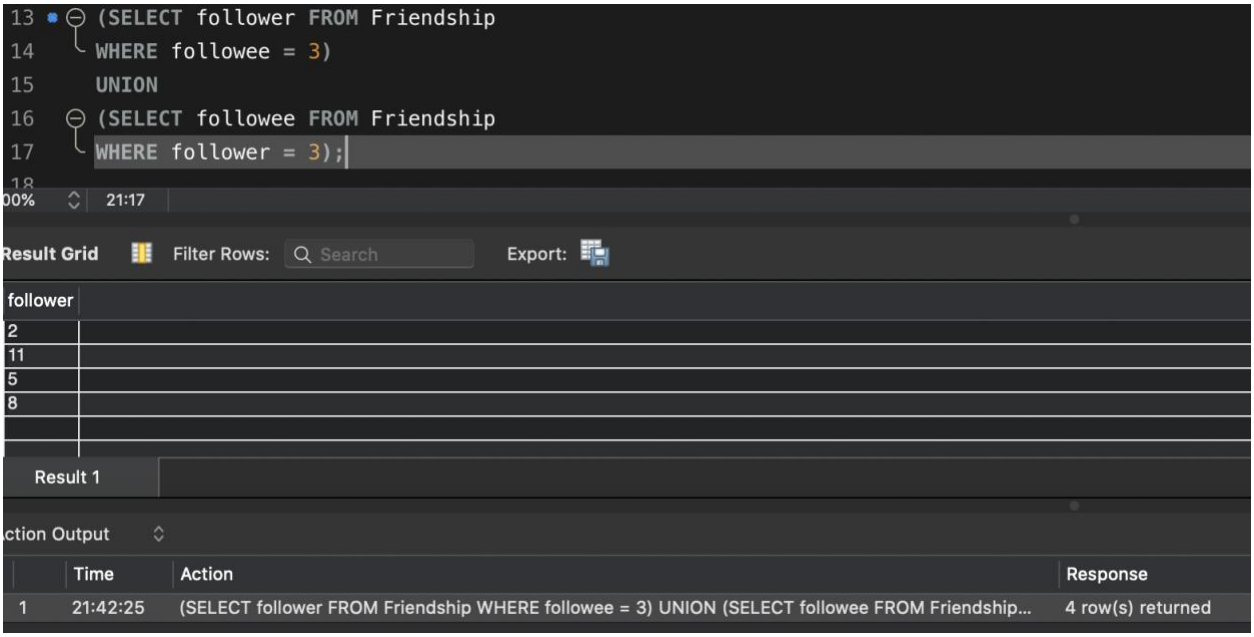
Add neighbor:



Accept friendship:



List all friends:



List all neighbor:

```
19 •  SELECT acceptor FROM Neighboring
20    WHERE initiator = 3;
```

100% 21:20

Result Grid | Filter Rows: Search | Export:

acceptor
8

Neighboring 5

Action Output

	Time	Action	Response
1	21:43:43	SELECT acceptor FROM Neighboring WHERE initiator = 3 LIMIT 0, 5000	1 row(s) returned

(4) List all threads that have new message since last access:

```
2  •  SET @query_user = 11;
3  •  SET @llt = (SELECT last_login_timestamp FROM Users AS u
4     WHERE u.uid = @query_user);
5
6  •  SELECT mid FROM Thread
7         WHERE ttimestamp > @llt AND uid = @query_user;
8
9  •  SELECT * FROM Message
10    WHERE mid IN (SELECT mid FROM Thread
11        WHERE ttimestamp > @llt AND uid = @query_user);
12
```

100% 49:11

Result Grid | Filter Rows: Search | Edit: | Export/Import:

mid	author	title	content	mtimestamp	visibility	receiver	lat	lng
1	11	Bicycle accident	There is a bicycle accident on the third street. A...	2018-03-02 20:00:00	hood	NULL	1.1	1.1
2	10	New hot dog store	The new dog store is so hot, yay!	2018-03-03 10:00:00	hood	NULL	1.1	1.1
3	10	Brain problem	My right brain has noting left.	2018-03-02 20:00:00	block	NULL	1.2	1.2
4	10	Brain problem	My left brain has nothing right	2018-03-03 20:00:00	block	NULL	1.1	1.1
5	11	Pokemon go fans	Anyone found a pikachu near the new Xmas tree?	2019-11-29 16:00:01	friend	NULL	1.1	1.1
6	11	Hitchhicker for Woodbury shopping	Hi Cami! One spare spot to Woodbury on Friday...	2019-11-20 01:40:49	direct	5	1.1	1.1
7	1	FFVII Remake is coming	Yo, bro! Final Fantasy VII Remake is coming ne...	2019-08-20 01:40:33	direct	11	NULL	NULL
8	5	Street Food Festival	Chilli Daddy has the best noodle soup for you!	2018-01-02 20:00:00	block	NULL	1.1	1.1
9	5	AD: Free cake	Free cake give away at the Plaza	2018-01-02 20:00:01	block	NULL	1.1	1.1
10	5	Street Food Festival Again	Chilli Daddy has the best noodle soup for you!	2018-01-02 21:00:00	friend	NULL	1.1	1.1
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Message 7

Action Output

	Time	Action	Response
1	21:44:40	SET @query_user = 11	0 row(s) affected
2	21:44:43	SET @llt = (SELECT last_login_timestamp FROM Users AS u WHERE u.uid = @query_user)	0 row(s) affected
3	21:44:47	SELECT mid FROM Thread WHERE ttimestamp > @llt AND uid = @query_user LIMIT 0, 5000	10 row(s) returned
4	21:44:49	SELECT * FROM Message WHERE mid IN (SELECT mid FROM Thread WHERE ttimestamp > @llt AND...	10 row(s) returned

and replies:

```
13  SELECT * FROM Reply
14  WHERE mid IN (SELECT mid FROM Message
15  WHERE mid IN (SELECT mid FROM Thread
16      WHERE ttimestamp > @llt AND uid = @query_user));
17
18  -- List all unread messages in friend feed
19  SET @query_user = 11;
```

100% 50:16

Result Grid | Filter Rows: Search | Edit: | Export/Import:

rid	mid	author	content	rtimestamp
1	1	9	I didn't see it.	2018-03-03 20:00:00
2	1	8	I didn't see it, either.	2018-03-04 20:00:00
3	1	10	Sorry to hear that.	2018-03-03 21:00:00
8	3	2	I didn't get it.	2018-03-02 20:20:04
6	5	5	I got it!	2019-11-29 16:00:05
7	5	3	I got it!	2019-11-29 16:00:07
5	6	6	I wish I could go but I'm working on the databas…	2019-11-21 00:00:49
4	7	11	Gotta buy a PS4 for gaming!	2019-08-20 10:41:33
9	10	10	Hi there!I my testing the c2 by replying a new m…	2019-12-04 02:39:13
NULL	NULL	NULL	NULL	NULL

Reply 8

Action Output

	Time	Action	Response
1	21:44:40	SET @query_user = 11	0 row(s) affected
2	21:44:43	SET @llt = (SELECT last_login_timestamp FROM Users AS u WHERE u.uid = @query_user)	0 row(s) affected
3	21:44:47	SELECT mid FROM Thread WHERE ttimestamp > @llt AND uid = @query_user LIMIT 0, 5000	10 row(s) returned
4	21:44:49	SELECT * FROM Message WHERE mid IN (SELECT mid FROM Thread WHERE ttimestamp > @llt AND…	10 row(s) returned
5	21:45:29	SELECT * FROM Reply WHERE mid IN (SELECT mid FROM Message WHERE mid IN (SELECT mid FROM…	9 row(s) returned

List all unread thread in friend feed:

```
-- List all unread messages in friend feed
SET @query_user = 11;

SELECT * FROM Message AS w,
(SELECT mid FROM Message
WHERE author IN
    (SELECT follower FROM Friendship
        WHERE followee = @query_user AND fstatus = 'accepted'
    UNION
    SELECT followee FROM Friendship
        WHERE follower = @query_user AND fstatus = 'accepted')
AND visibility = 'friend') AS m
WHERE w.mid IN
(SELECT mid FROM Thread
WHERE uid = @query_user AND tstatus = 'unread') AND m.mid = w.mid;
```

100% 67:32

Result Grid Filter Rows: Search Export:

mid	author	title	content	mtimestamp	visibility	receiver	lat	lng	mid
10	5	Street Food Festival Again	Chilli Daddy has the best noodle soup for you!	2018-01-02 21:00:00	friend	NULL	1.1	1.1	10

Result 9

Action Output

	Time	Action	Response
1	21:46:01	SELECT * FROM Message AS w, (SELECT mid FROM Message WHERE author IN (SELECT follower FR...	1 row(s) returned

List all “bicycle accident” threads: (we omit the actual message content, just showing thread)

```
-- List all "bicycle accident" message the user can access
SELECT * FROM Thread
WHERE uid = @query_user AND mid IN
(SELECT mid FROM Message
WHERE title LIKE '%bicycle accident%' OR content LIKE '%bicycle accident%');
```

100% 77:38

Result Grid Filter Rows: Search Edit: Export/Import:

uid	mid	tstatus	ttimestamp
11	1	unread	2019-12-04 02:32:36
NULL	NULL	NULL	NULL

Thread 10

Action Output

	Time	Action	Response
1	21:46:32	SELECT * FROM Thread WHERE uid = @query_user AND mid IN (SELECT mid FROM Message WHERE t...	1 row(s) returned