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

Project Report

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.

1. Logical Design

ER model

The first step is the logical design. We started with the ER model. we modeled six entities as well as the relationships associate among them. Then used the ER diagram to express the overall logical structure of our database. In part 2 of the project, we decided to add another relation, Approval, to describe the action when a user who is an existing block member approves a pending member of that block. With the aid of Approval schema, we can ensure one user can only approve a pending member once. The ER diagram is attached below:

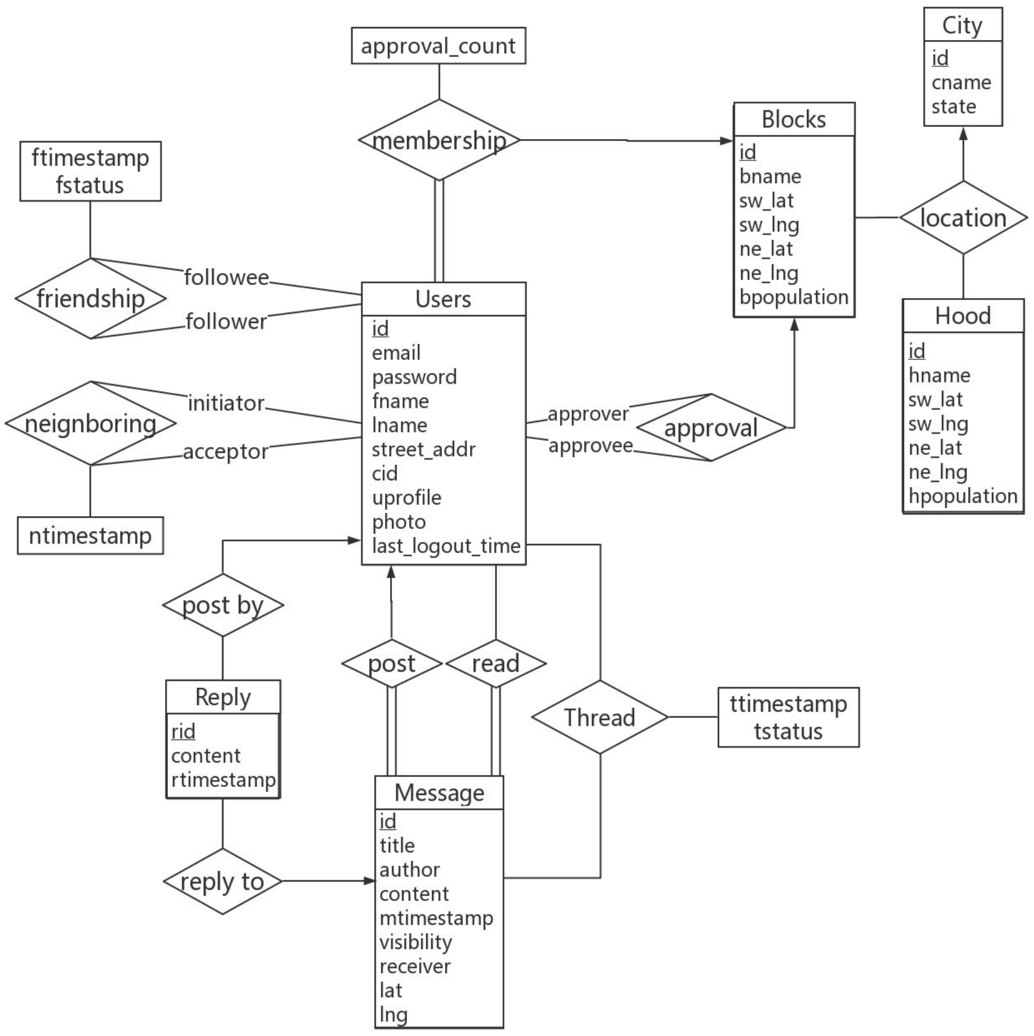


Figure 1 ER diagram

Relational Schemas

Then we translated the above ER schemas into relational schemas:

- Users (id, email, pword, fname, lname, street_addr, *cid*, uprofile, photo, last_logout_timestamp)
- Hood (id, hname, sw_lat, sw_lng, ne_lat, ne_lng, hpopulation)
- Blocks (id, 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 (id, *author*, title, content, mtimestamp, visibility, *receiver*, lat, lng)
- Reply (rid, *mid*, *author*, content, rtimestamp)
- Thread (*uid*, *mid*, tstatus, ttimestamp)
- City (id, cname, cstate)
- Approval (*approver*, *approvee*, *bid*)

In the schemas above, the primary key is labeled with an underline, and the foreign key is in *italics*.

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

Users	
id	INTEGER, UNIQUE, PRIMARY KEY
email	VARCAHR(45), NOT NULL, UNIQUE
pword	VARCAHR(45), NOT NULL
fname	VARCAHR(45), NOT NULL
lname	VARCAHR(45), NOT NULL
street_addr	VARCAHR(45), NOT NULL
cid	INTEGER, FOREIGN KEY referencing cid in City
uprofile	TEXT
photo	TEXT
last_logout_timestamp	TIMESTAMP, NOT NULL, DEFAULT CURRENT_TIMESTAMP, ON UPDATE CURRENT_TIMESTAMP

Table 1 User

- id is an 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 id. 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_logout_timestamp is a timestamp auto-generated and updated every time a user logouts.

Hood	
id	INTEGER, UNIQUE, PRIMARY KEY
hname	VARCAHR(45), NOT NULL
sw_lat	FLOAT, NOT NULL
sw_lng	FLOAT, NOT NULL
ne_lat	FLOAT, NOT NULL
ne_lng	FLOAT, NOT NULL
hpopulation	INTEGER, NOT NULL

Table 2 Hood

- id is an auto-generated and auto-incremented number that 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 the 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 will be implemented in our backend application.

Blocks	
id	INTEGER, UNIQUE, PRIMARY KEY
bname	VARCAHR(45), NOT NULL
sw_lat	FLOAT, NOT NULL
sw_lng	FLOAT, NOT NULL
ne_lat	FLOAT, NOT NULL
ne_lng	FLOAT, NOT NULL
bpopulation	INTEGER, NOT NULL

Table 3 Blocks

- 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	INTEGER PRIMARY KEY, FOREIGN KEY referencing id in Blocks
hid	INTEGER, PRIMARY KEY, FOREIGN KEY referencing id in Hood
cid	INTEGER, PRIMARY KEY, FOREIGN KEY referencing id in City

Table 4 Location

- 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	INTEGER, PRIMARY KEY, FOREIGN KEY referencing id in Users
bid	INTEGER, PRIMARY KEY, FOREIGN KEY referencing id in Blocks
approval_count	INTEGER, NOT NULL, DEFAULT 0

Table 5 Membership

- The uid and the bid here represent a potential membership, which means 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. We decided once the user is approved to join a block, his/her approval_count is set to -1.

Friendship	
follower	INTEGER, PRIMARY KEY, FOREIGN KEY referencing id in Users
followee	INTEGER, PRIMARY KEY, FOREIGN KEY referencing id in Users
ftimestamp	TIMESTAMP, NOT NULL, DEFAULT CURRENT_TIMESTAMP
fstatus	VALUE IN ('pending', 'rejected', 'accepted'), DEFAULT 'pending'

Table 6 Friendship

- follower and followee here are foreign keys referencing to id 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	INTEGER, PRIMARY KEY, FOREIGN KEY referencing id in Users
acceptor	INTEGER, PRIMARY KEY, FOREIGN KEY referencing id in Users
ntimestamp	TIMESTAMP, NOT NULL, DEFAULT CURRENT_TIMESTAMP

Table 7 Neighboring

- The setting of this schema is similar to Friendship.
- Adding of a neighbor is unilateral 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	
id	INTEGER, UNIQUE, PRIMARY KEY
author	INTEGER, FOREIGN KEY referencing uid in Users
title	TEXT, NOT NULL
content	TEXT, NOT NULL
mtimestamp	TIMESTAMP, NOT NULL, DEFAULT CURRENT_TIMESTAMP
visibility	VARCAHR(45), NOT NULL, VALUE IN (‘direct’, ‘friend’, ‘neighbor’, ‘block’, ‘hood’)
receiver	INTEGER, FOREIGN KEY referencing id in Users, NULLABLE, DEFAULT NULL
lat	FLOAT, NULLABLE
lng	FLOAT, NULLABLE

Table 8 Message

- id 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	
id	INTEGER, UNIQUE, PRIMARY KEY
mid	INTEGER, FOREIGN KEY referencing id in Message
author	INTEGER, FOREIGN KEY referencing id in Users
content	TEXT, NOT NULL
rtimestamp	TIMESTAMP, NOT NULL, DEFAULT CURRENT_TIMESTAMP

Table 9 Reply

- 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 a 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	INTEGER, FOREIGN KEY referencing id in Users, PRIMARY KEY
mid	INTEGER, FOREIGN KEY referencing id in Message, PRIMARY KEY
tstatus	VARCAHR(45), NOT NULL, VALUE IN (‘read’, ‘unread’), DEFAULT ‘unread’
ttimestamp	TIMESTAMP, NOT NULL, DEFAULT CURRENT_TIMESTAMP

Table 10 Thread

- 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, a message with mid=3 will be displayed. Similarly, when A post a message to his or her friends, entries with A’s friends’ id will be added. Anyone who is not his or her friend will not even know the message is posted by the 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 the Thread table are done by a server application.
- 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	
id	INTEGER, UNIQUE, PRIMARY KEY
cname	VARCAHR(45), NOT NULL
state	VARCAHR(45), NOT NULL

Table 11 City

- id 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 id 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.

Approval	
approver	INTEGER, NOT NULL, PRIMARY KEY, FOREIGN KEY referencing id in Users
approvee	INTEGER, NOT NULL, PRIMARY KEY, FOREIGN KEY referencing id in Users
bid	INTEGER, NOT NULL, PRIMARY KEY, FOREIGN KEY referencing id in Blocks

Table 12 Approval

- approver and approvee here are foreign keys referencing to id in Users. An approver approves an approvee's pending membership to join his/her block.
- bid is a primary key and a foreign key referencing to id in Blocks. It specifies which blocks approval is for.

Function Implementation

We create our schema by using SQLite. The required functions 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 id = 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 to a message:

```
INSERT INTO Reply(mid, author, content, rtimestamp) VALUE
(U, V, 'Hi there! I am 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_logout_timestamp FROM Users AS u
WHERE u.id = X);

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

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

SELECT * FROM Reply
WHERE mid IN (SELECT id 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 id 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.id IN
(SELECT mid FROM Thread
WHERE uid = X AND tstatus = 'unread') AND m.mid = w.id;
```

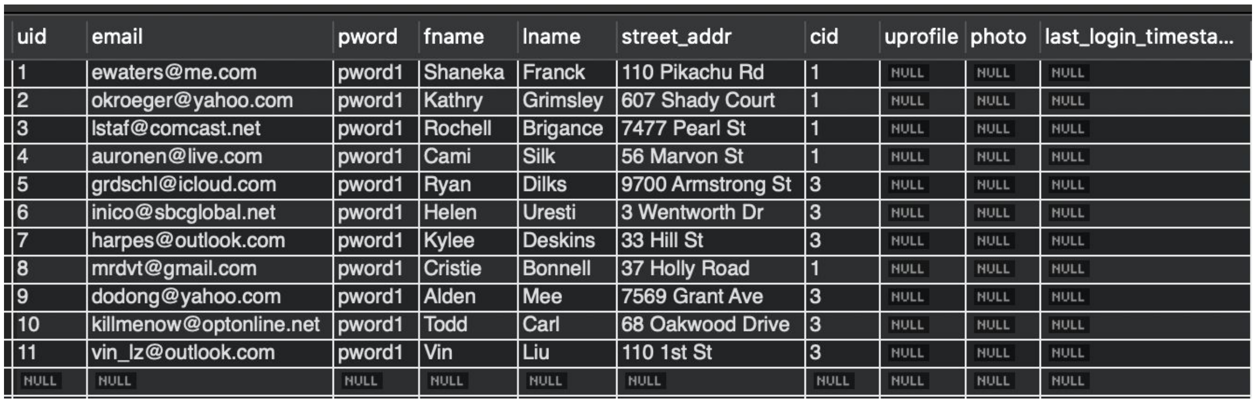
List all "bicycle accident" threads:

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

Sample Data

We list some sample data below:

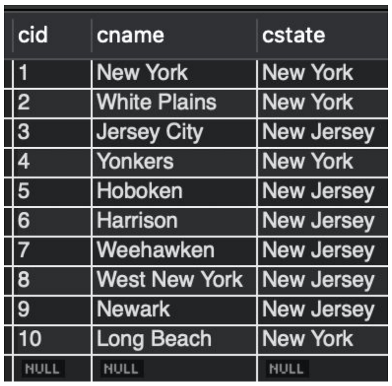
Users:



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

Figure 2 sample data for User

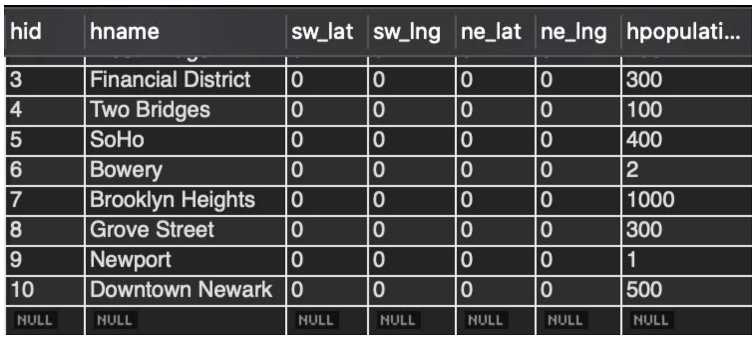
City:



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

Figure 3 sample data for City

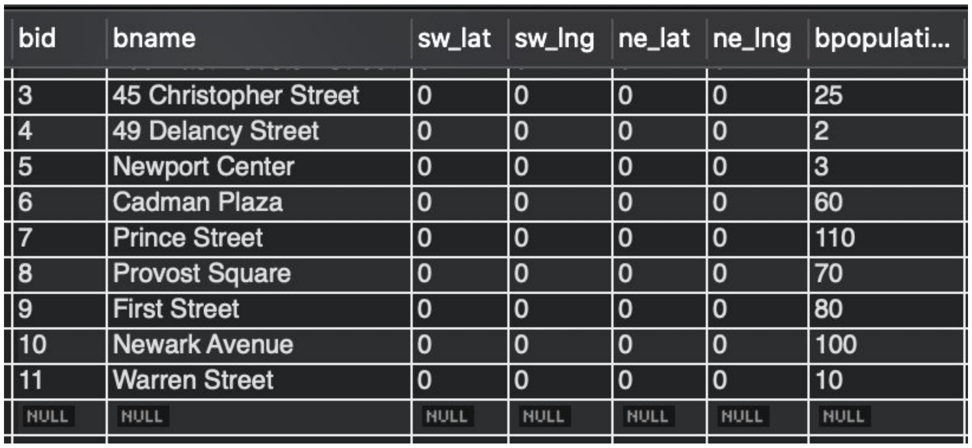
Hood:



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

Figure 4 sample data for Hood

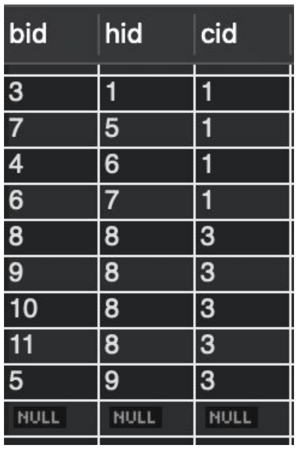
Blocks:



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

Figure 5 sample data for Blocks

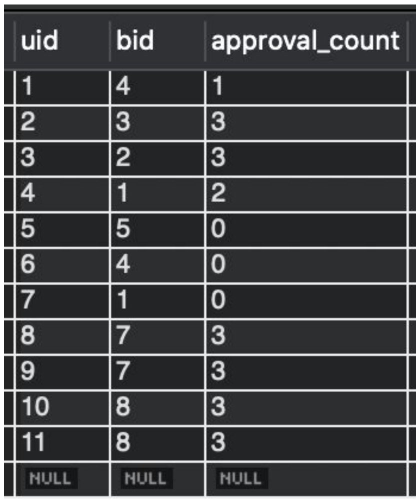
Location:



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

Figure 6 sample data for Location

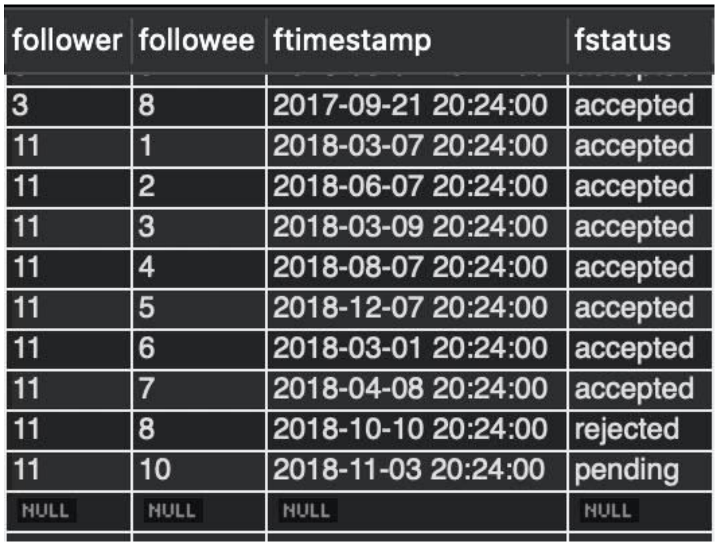
Membership:



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

Figure 7 sample data for Membership

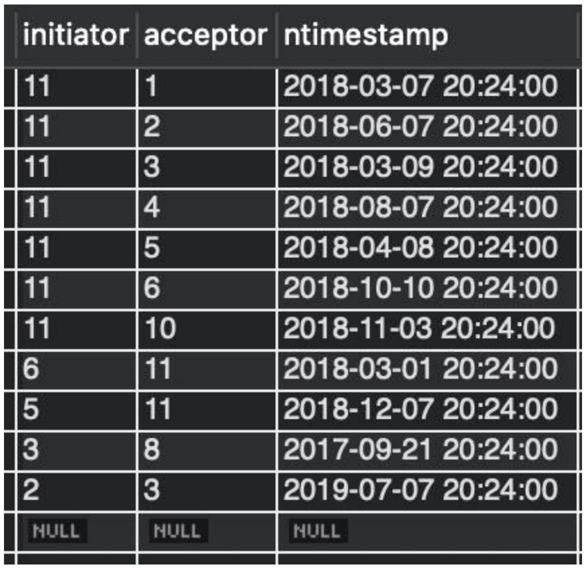
Friendship:



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

Figure 8 sample data for Friendship

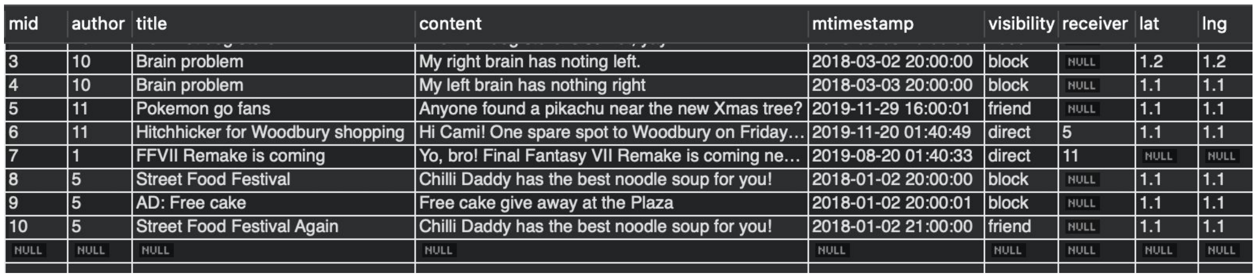
Neighboring:



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

Figure 9 sample data for Neighboring

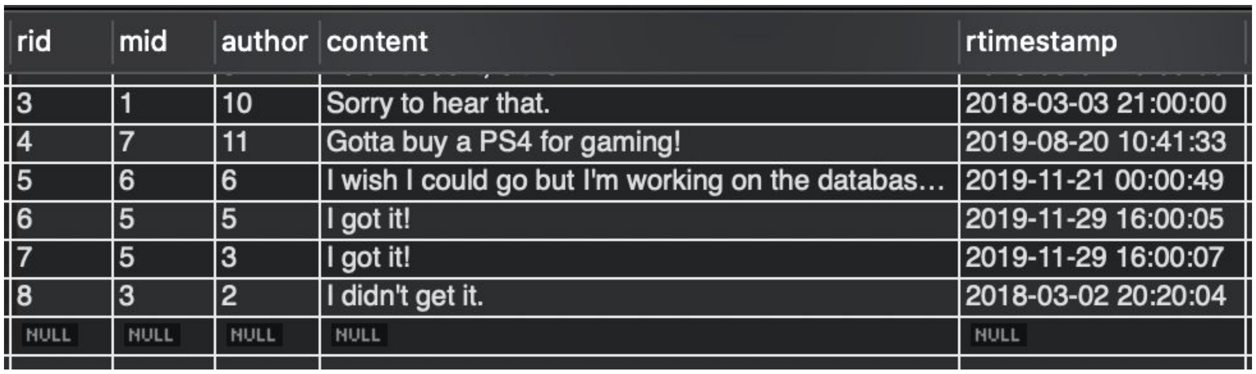
Message:



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

Figure 10 sample data for Message

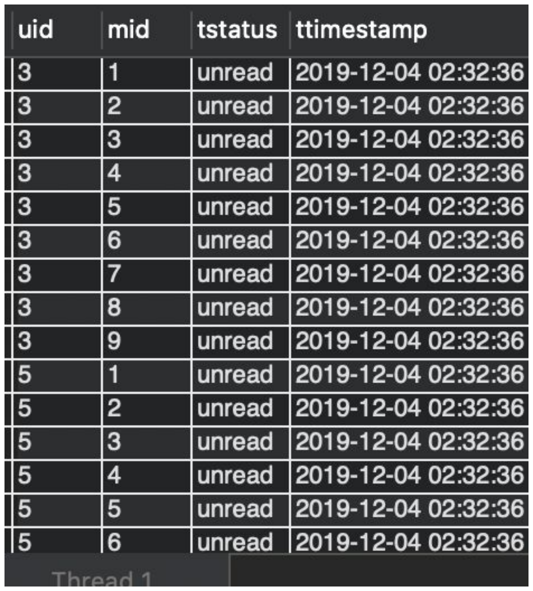
Reply:



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

Figure 11 sample data for Reply

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

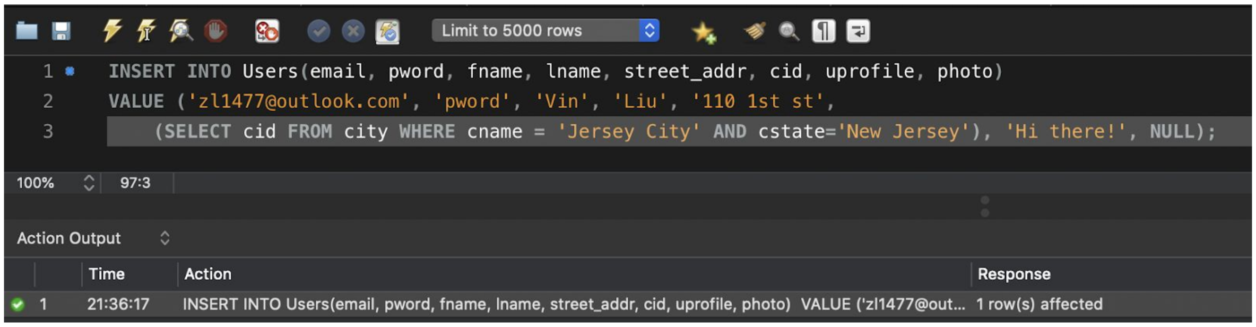
Figure 12 sample data for Thread

Function Test

In this part, we test the required function asked in project 1

(1) Account

Sign up:



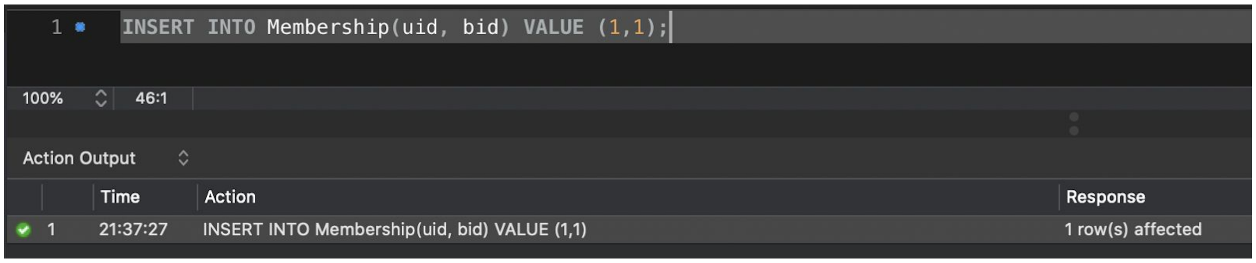
```
INSERT INTO Users(email, pword, fname, lname, street_addr, cid, uprofile, photo)
VALUE ('zl1477@outlook.com', 'pword', 'Vin', 'Liu', '110 1st st',
    (SELECT cid FROM city WHERE cname = 'Jersey City' AND cstate='New Jersey'), 'Hi there!', NULL);
```

```
100%    97:3

Action Output

     Time      Action                                                                                           Response
✓ 1  21:36:17  INSERT INTO Users(email, pword, fname, lname, street_addr, cid, uprofile, photo)  VALUE ('zl1477@out...  1 row(s) affected
```

Become a member of a block:



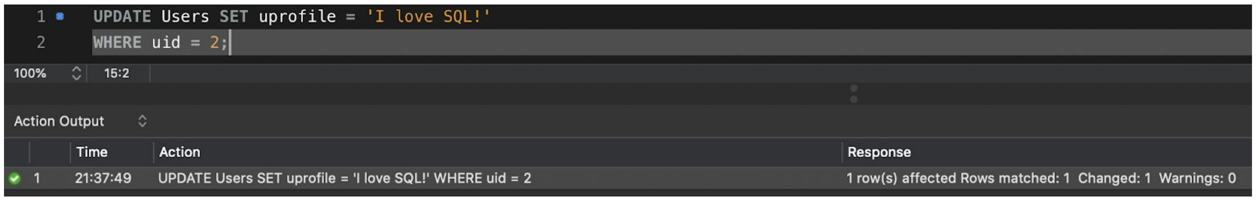
```
INSERT INTO Membership(uid, bid) VALUE (1,1);
```

```
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:



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

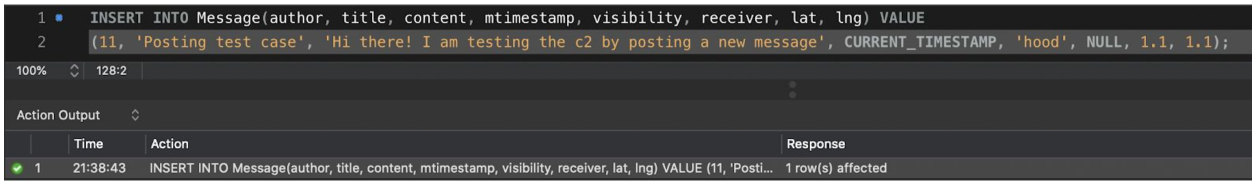
```
100%    15:2

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) Message

Post an initial message:



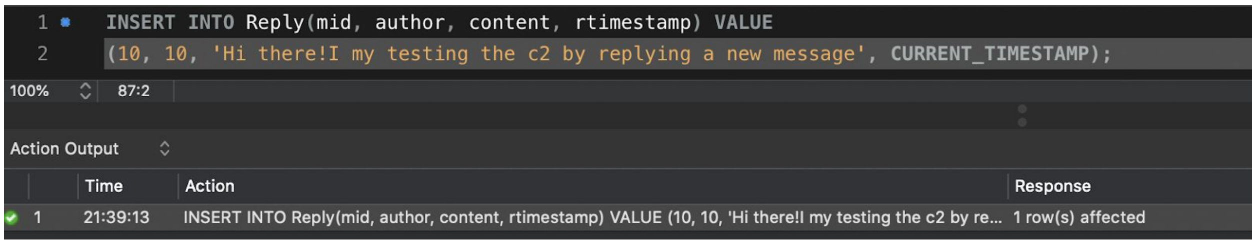
```
INSERT INTO Message(author, title, content, mtimestamp, visibility, receiver, lat, lng) VALUE
(11, 'Posting test case', 'Hi there! I am testing the c2 by posting a new message', CURRENT_TIMESTAMP, 'hood', NULL, 1.1, 1.1);
```

```
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:



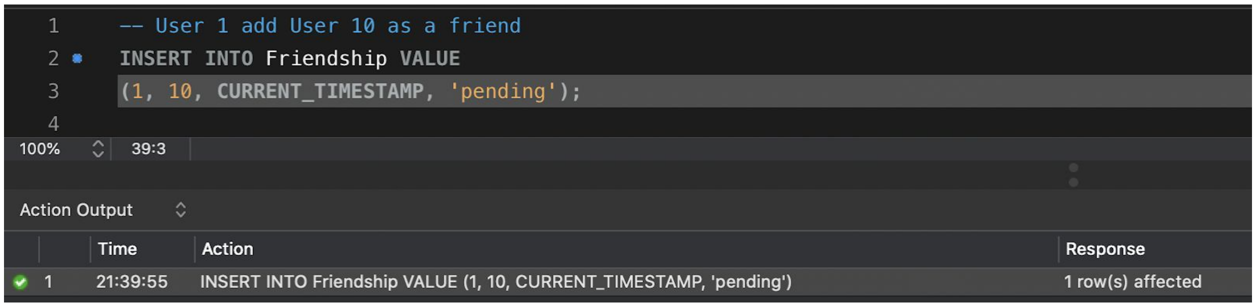
```
INSERT INTO Reply(mid, author, content, rtimestamp) VALUE
(10, 10, 'Hi there!I my testing the c2 by replying a new message', CURRENT_TIMESTAMP);
```

```
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:



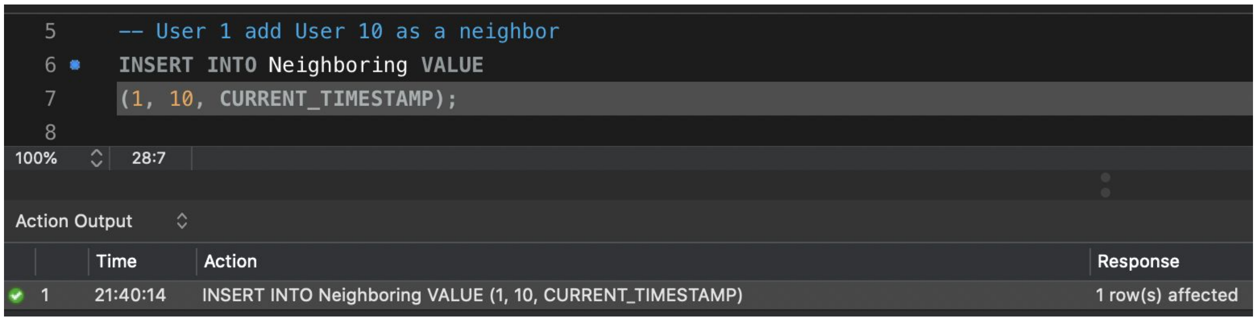
```
-- User 1 add User 10 as a friend
INSERT INTO Friendship VALUE
(1, 10, CURRENT_TIMESTAMP, 'pending');
```

```
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
```

(4) Add neighbor:



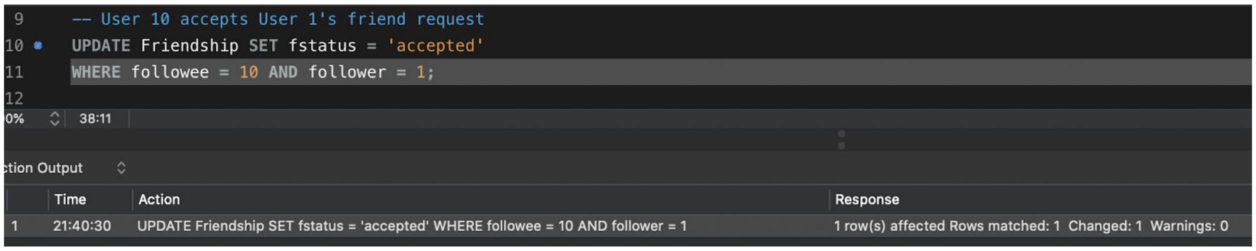
```
-- User 1 add User 10 as a neighbor
INSERT INTO Neighboring VALUE
(1, 10, CURRENT_TIMESTAMP);
```

```
100%    28:7

Action Output

     Time       Action                                                    Response
✓ 1  21:40:14   INSERT INTO Neighboring VALUE (1, 10, CURRENT_TIMESTAMP)   1 row(s) affected
```

(5) Accept friendship:



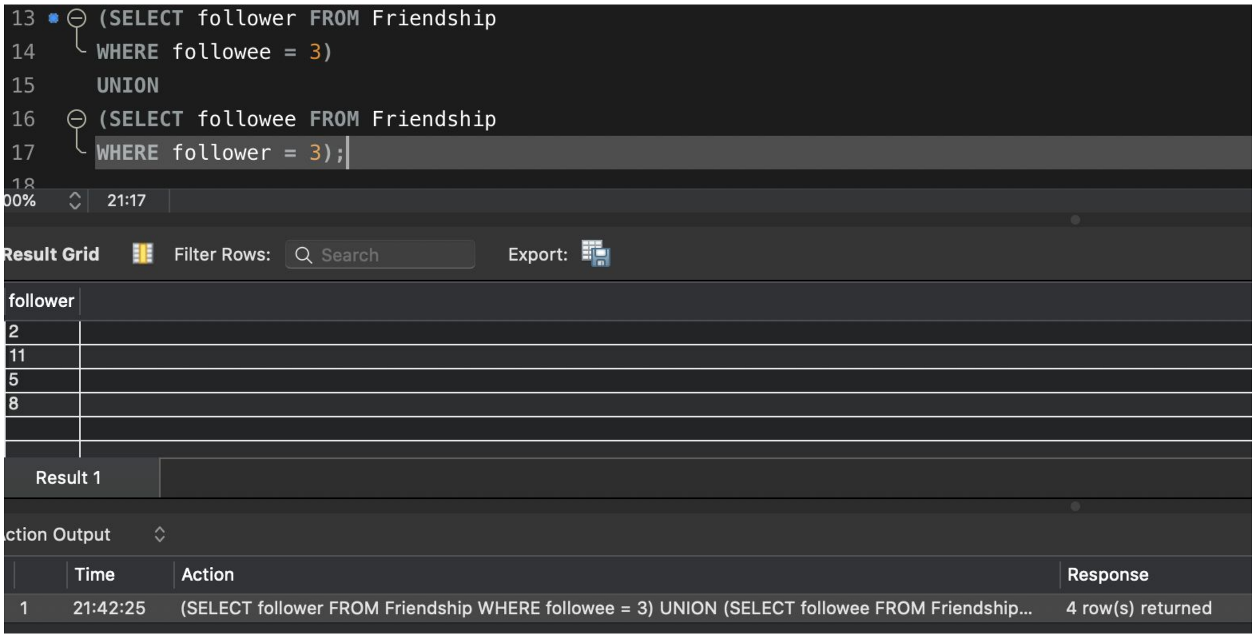
```
-- User 10 accepts User 1's friend request
UPDATE Friendship SET fstatus = 'accepted'
WHERE followee = 10 AND follower = 1;
```

```
0%    38:11

tion Output

    Time       Action                                                                      Response
1   21:40:30   UPDATE Friendship SET fstatus = 'accepted' WHERE followee = 10 AND follower = 1   1 row(s) affected Rows matched: 1  Changed: 1  Warnings: 0
```

(6) List all friends:



```sql
(SELECT follower FROM Friendship
WHERE followee = 3)
UNION
(SELECT followee FROM Friendship
WHERE follower = 3);
```

```
00%    21:17
```

Result Grid   Filter Rows: Search   Export:

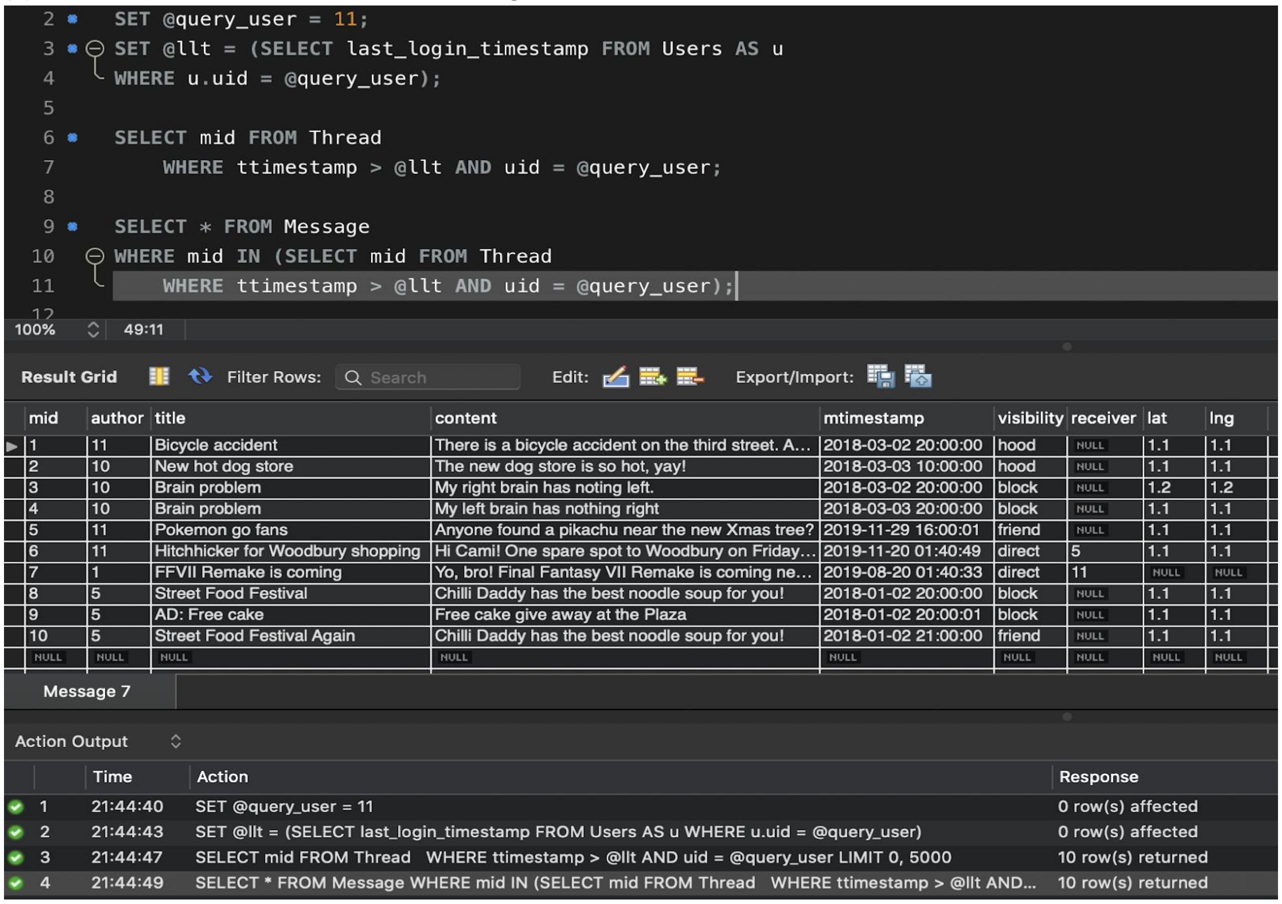
follower
2
11
5
8

Result 1

ction Output

```
    Time      Action                                                                                     Response
1   21:42:25  (SELECT follower FROM Friendship WHERE followee = 3) UNION (SELECT followee FROM Friendship...  4 row(s) returned
```

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



```sql
SET @query_user = 11;
SET @llt = (SELECT last_login_timestamp FROM Users AS u
WHERE u.uid = @query_user);

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

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

```
100%    49:11
```

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

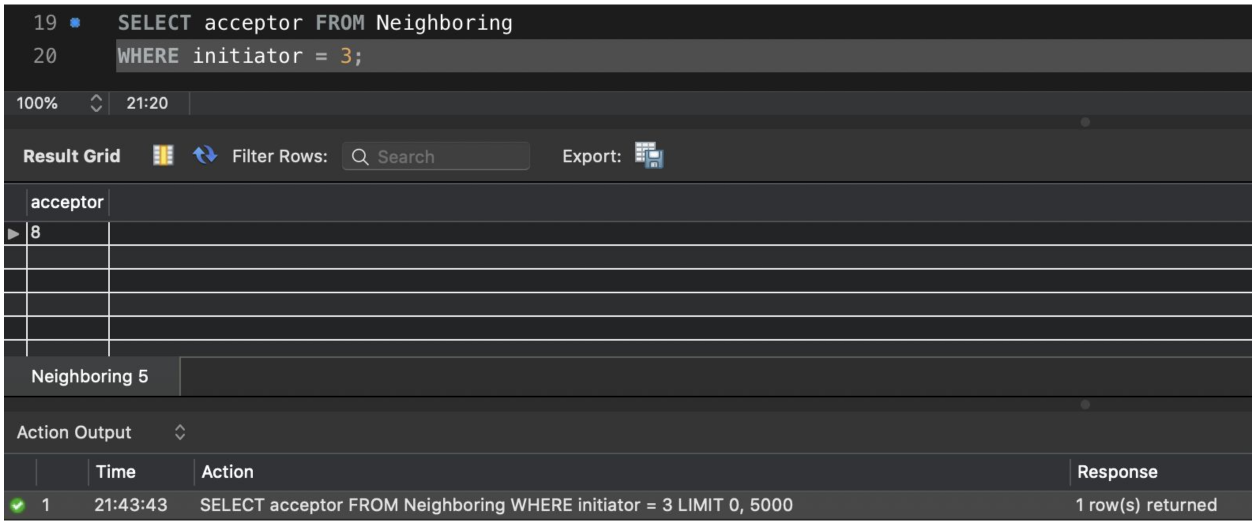
mid	author	title	content	mtimestamp	visibility	receiver	lat	lng
1	11	Bicycle accident	There is a bicycle accident on the third street. A...	2018-03-02 20:00:00	hood	NULL	1.1	1.1
2	10	New hot dog store	The new dog store is so hot, yay!	2018-03-03 10:00:00	hood	NULL	1.1	1.1
3	10	Brain problem	My right 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
```

(8) List all neighbor:



```sql
SELECT acceptor FROM Neighboring
WHERE initiator = 3;
```

100% 21:20

Result Grid | Filter Rows: Search | Export:

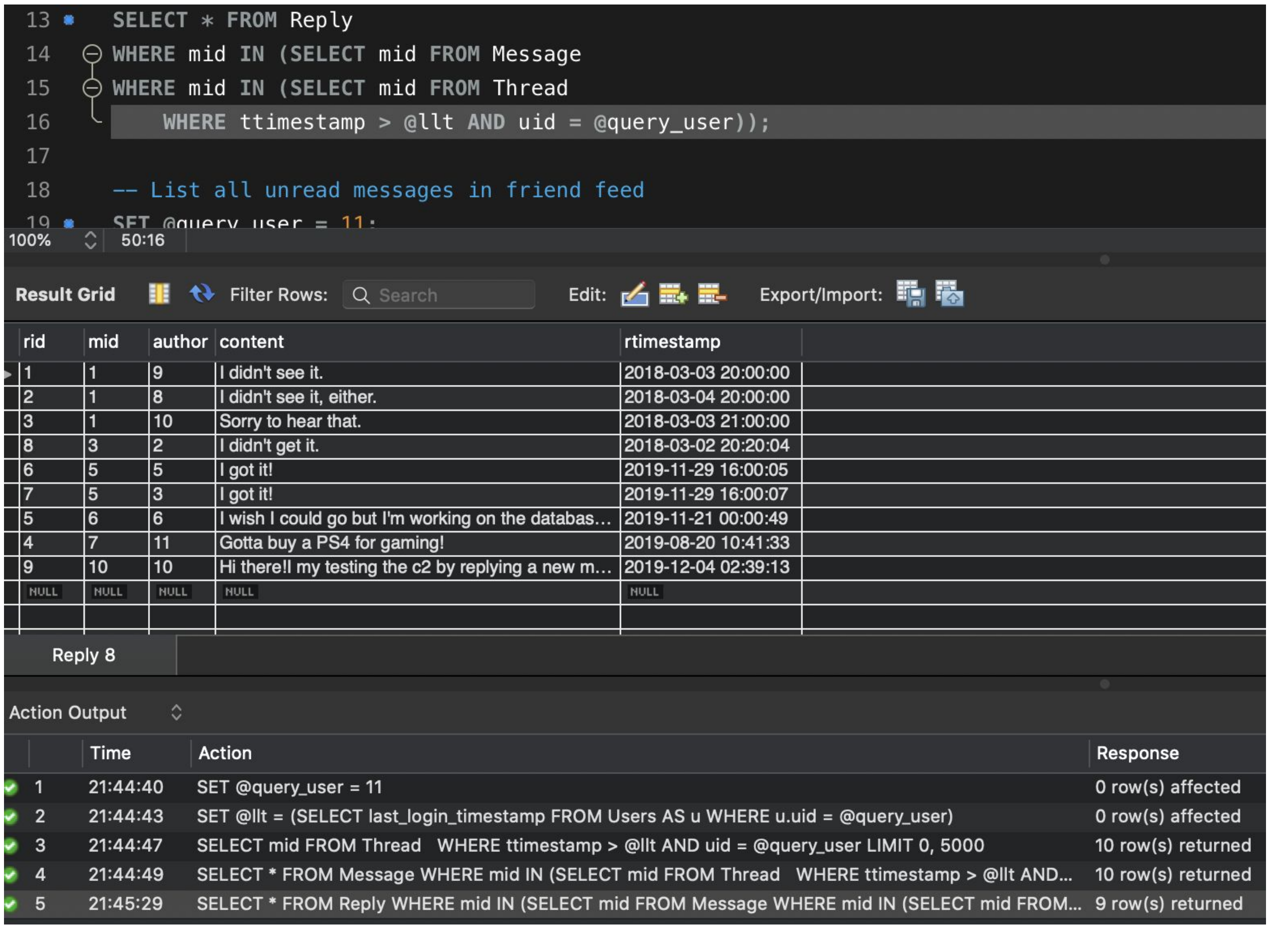
acceptor
8

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

(9) replies:



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

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

100% 50:16

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

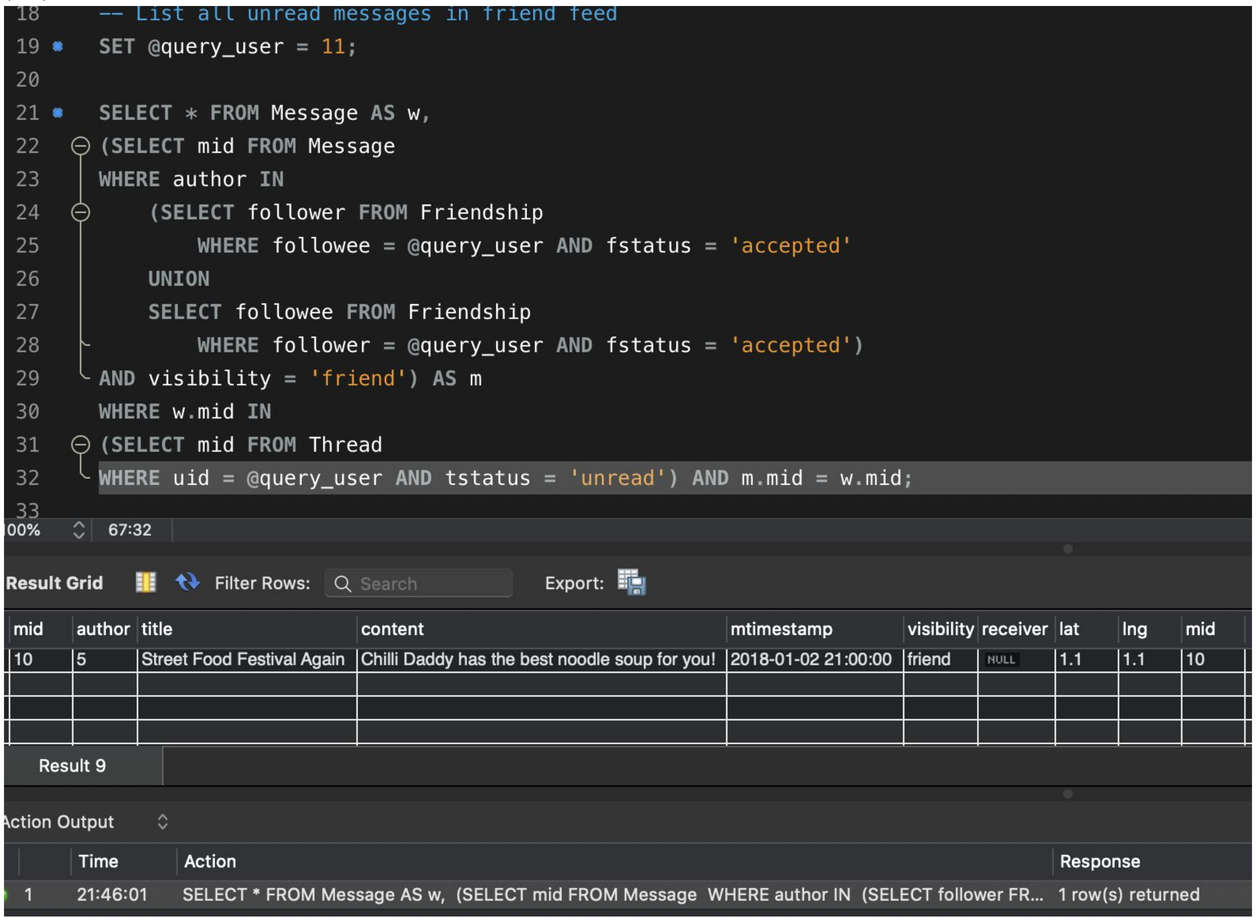
rid	mid	author	content	rtimestamp
1	1	9	I didn't see it.	2018-03-03 20:00:00
2	1	8	I didn't see it, either.	2018-03-04 20:00:00
3	1	10	Sorry to hear that.	2018-03-03 21:00:00
8	3	2	I didn't get it.	2018-03-02 20:20:04
6	5	5	I got it!	2019-11-29 16:00:05
7	5	3	I got it!	2019-11-29 16:00:07
5	6	6	I wish 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!! 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

(10) 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;
```

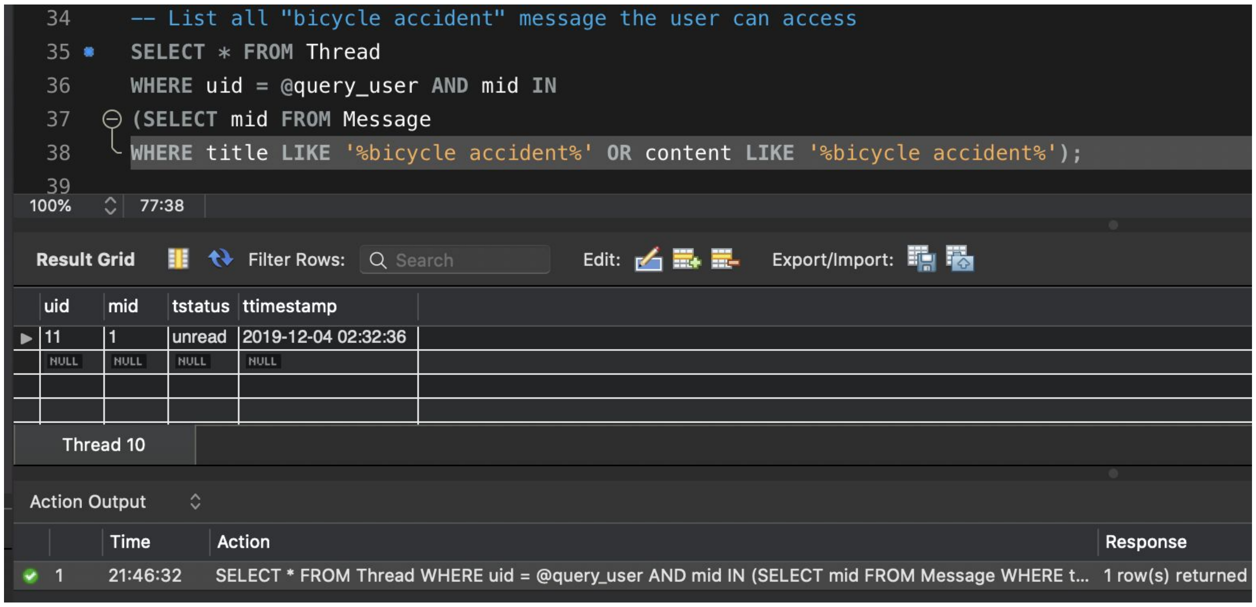
mid	author	title	content	mtimestamp	visibility	receiver	lat	lng	mid
10	5	Street Food Festival Again	Chilli Daddy has the best noodle soup for you!	2018-01-02 21:00:00	friend	NULL	1.1	1.1	10

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

(11) 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%');
```

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

Thread 10

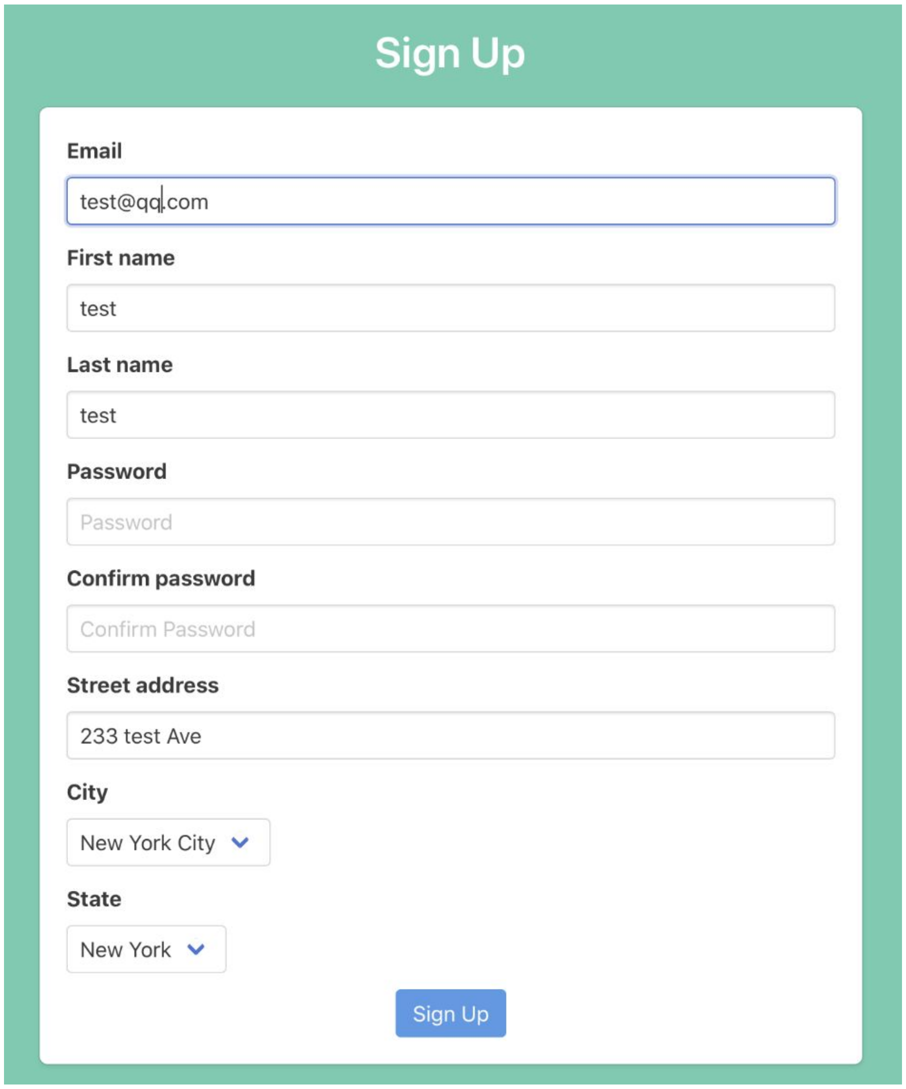
Action Output

	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

2. Web Application

In this part, we design a web-based user interface for the database designed in the first project. We use Python for the web application design, with the help of Flask and SQLAchemy to communicate with the database. The following are examples of how to use this application. (For the installation of the environment, please see README.md in the project folder)

- Creating a New User



Sign Up

Email: test@qq.com

First name: test

Last name: test

Password: Password

Confirm password: Confirm Password

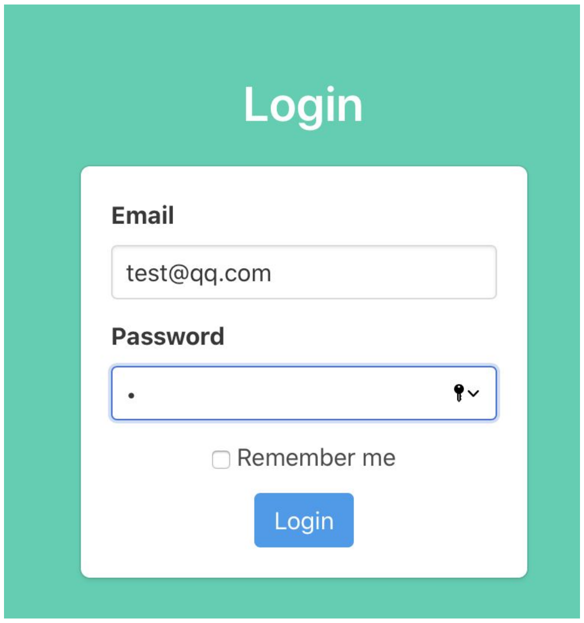
Street address: 233 test Ave

City: New York City

State: New York

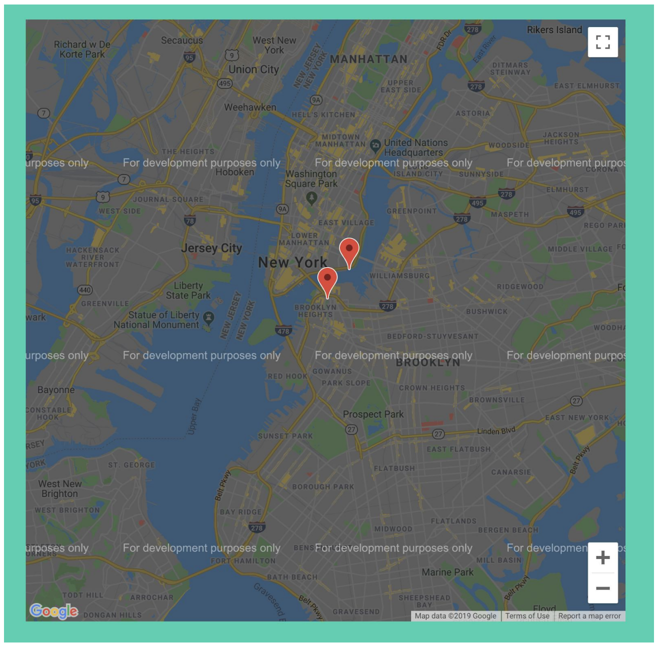
Here we are signing up a test user.

- Login Page



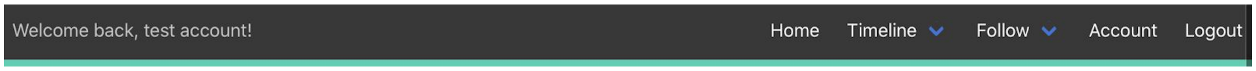
After login, a session will be created for the user to grant access until the user logouts.

- Embedded Map



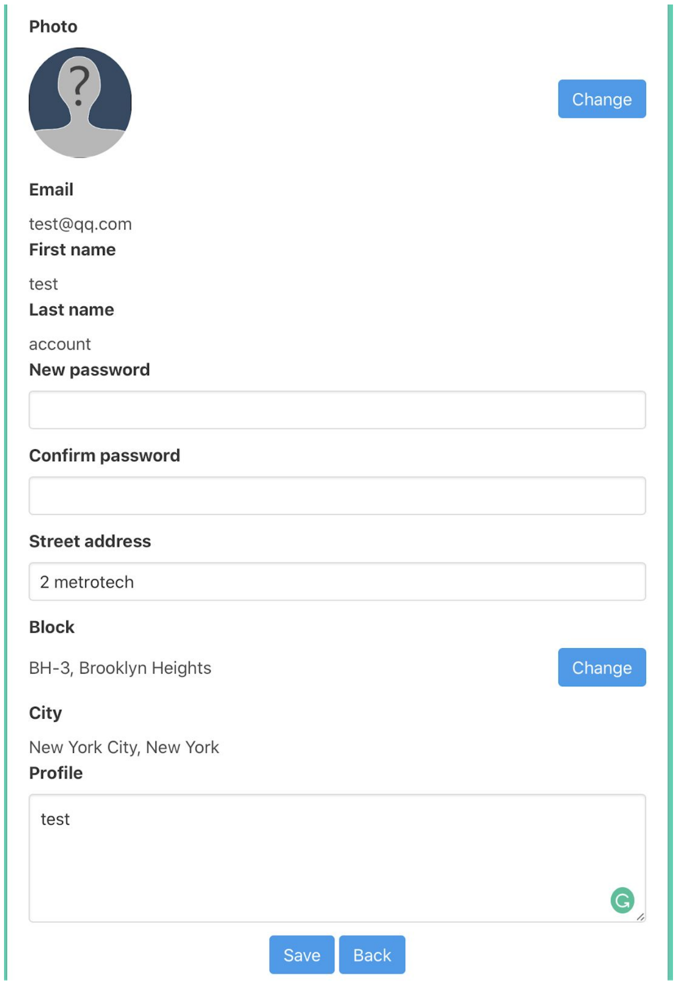
Notifications involving geological data are pinned on the map.

- Navigation Banner



A navigation banner on the top of the page provides a menu for the user to explore the application.

- Changing Account Info



Photo

Change

Email

test@qq.com

First name

test

Last name

account

New password

Confirm password

Street address

2 metrotech

Block

BH-3, Brooklyn Heights

Change

City

New York City, New York

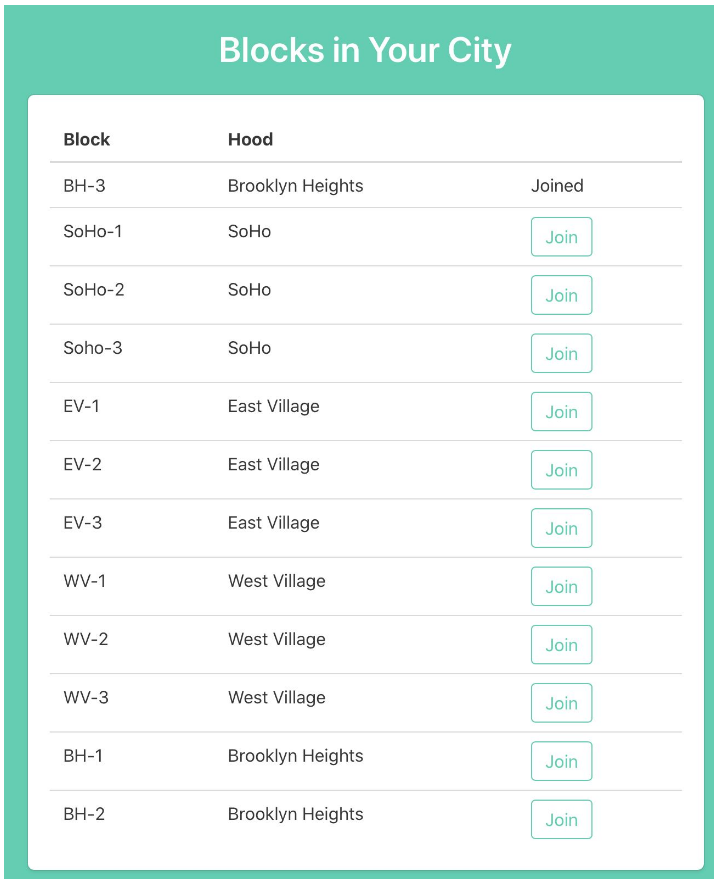
Profile

test

Save Back

Users can change his/her account information on the Account page.

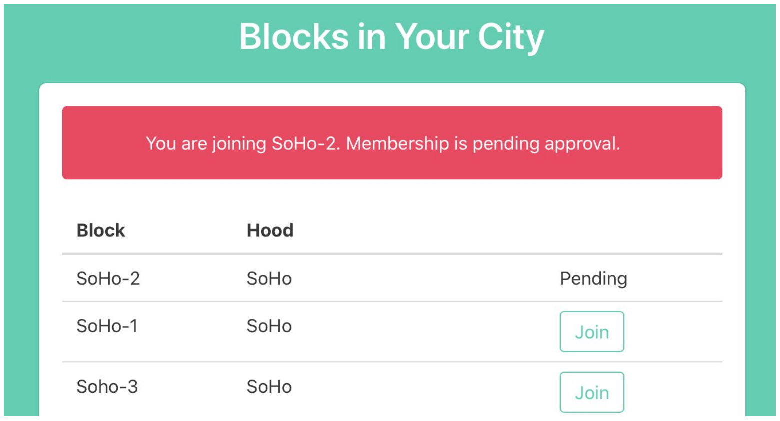
- Change to another Block or Join a Block



Blocks in Your City

Block	Hood	
BH-3	Brooklyn Heights	Joined
SoHo-1	SoHo	Join
SoHo-2	SoHo	Join
Soho-3	SoHo	Join
EV-1	East Village	Join
EV-2	East Village	Join
EV-3	East Village	Join
WV-1	West Village	Join
WV-2	West Village	Join
WV-3	West Village	Join
BH-1	Brooklyn Heights	Join
BH-2	Brooklyn Heights	Join

Click the " change block" button on the Account page lead the user to explore a block list in his/her city. The user can apply to join a block here.



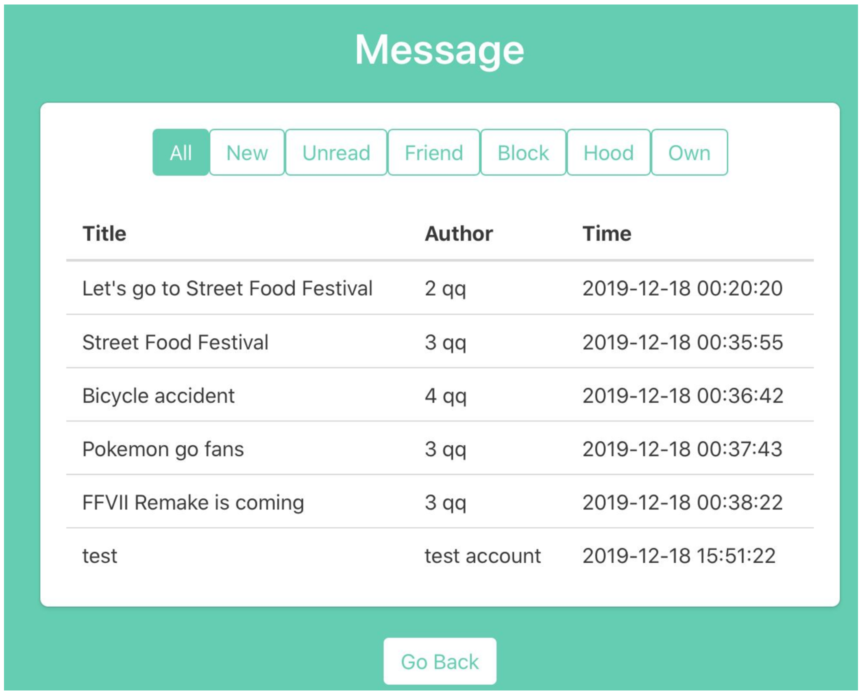
Blocks in Your City

You are joining SoHo-2. Membership is pending approval.

Block	Hood	
SoHo-2	SoHo	Pending
SoHo-1	SoHo	Join
Soho-3	SoHo	Join

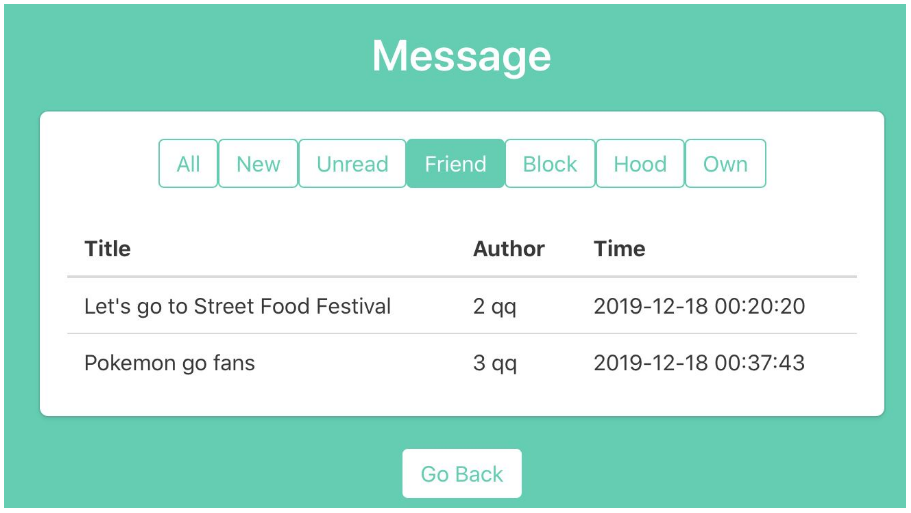
Once a joining application is sent, the membership is pending under approval by existing members in that block. If there the user is the first member in this block, the application will be accepted at once.

- Exploring the Timeline Messages

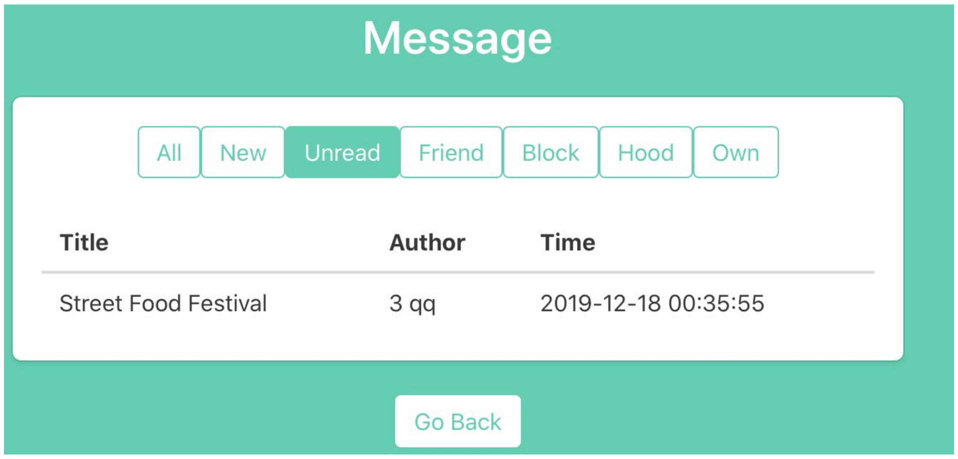


Title	Author	Time
Let's go to Street Food Festival	2 qq	2019-12-18 00:20:20
Street Food Festival	3 qq	2019-12-18 00:35:55
Bicycle accident	4 qq	2019-12-18 00:36:42
Pokemon go fans	3 qq	2019-12-18 00:37:43
FFVII Remake is coming	3 qq	2019-12-18 00:38:22
test	test account	2019-12-18 15:51:22

Click Timeline -> View on the navigation to see messages in the user's timeline.



Title	Author	Time
Let's go to Street Food Festival	2 qq	2019-12-18 00:20:20
Pokemon go fans	3 qq	2019-12-18 00:37:43


Message

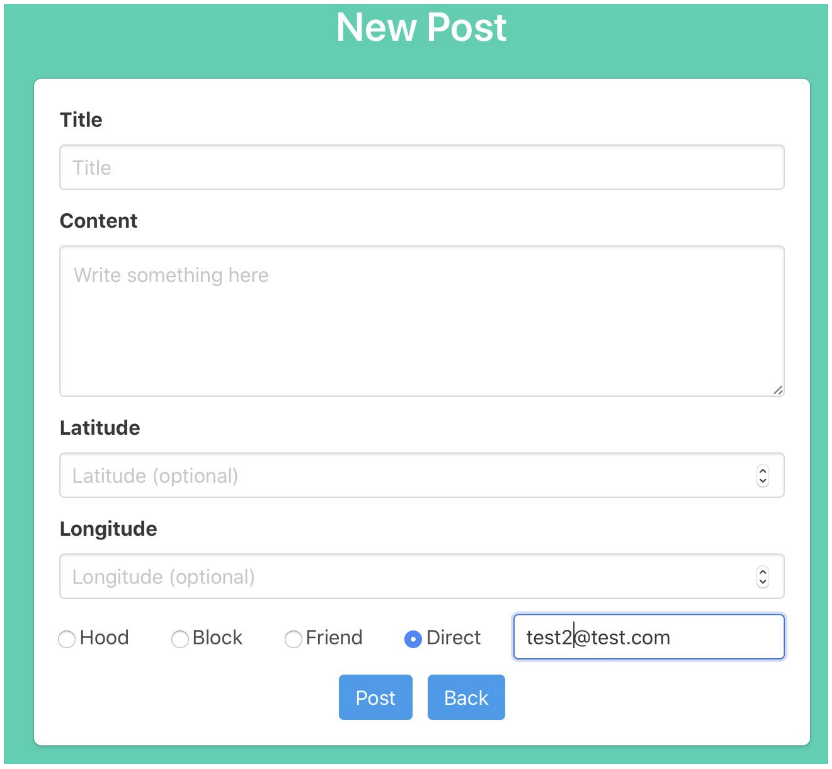
All | New | Unread | Friend | Block | Hood | Own

Title	Author	Time
Street Food Festival	3 qq	2019-12-18 00:35:55

Go Back

Messages can be categorized into different scopes. Click the tabs to filter the messages. `New` displays all messages posted after the time when the user last logout. `Unread` displays all messages have not been read by the user. `Friend` displays all messages from the user's friends. `Block` and `Hood` display messages within the geographical scope. The messages posted by the user will be displayed under the `Own` tab.

- Create a Message


New Post

Title: Title

Content: Write something here

Latitude: Latitude (optional)

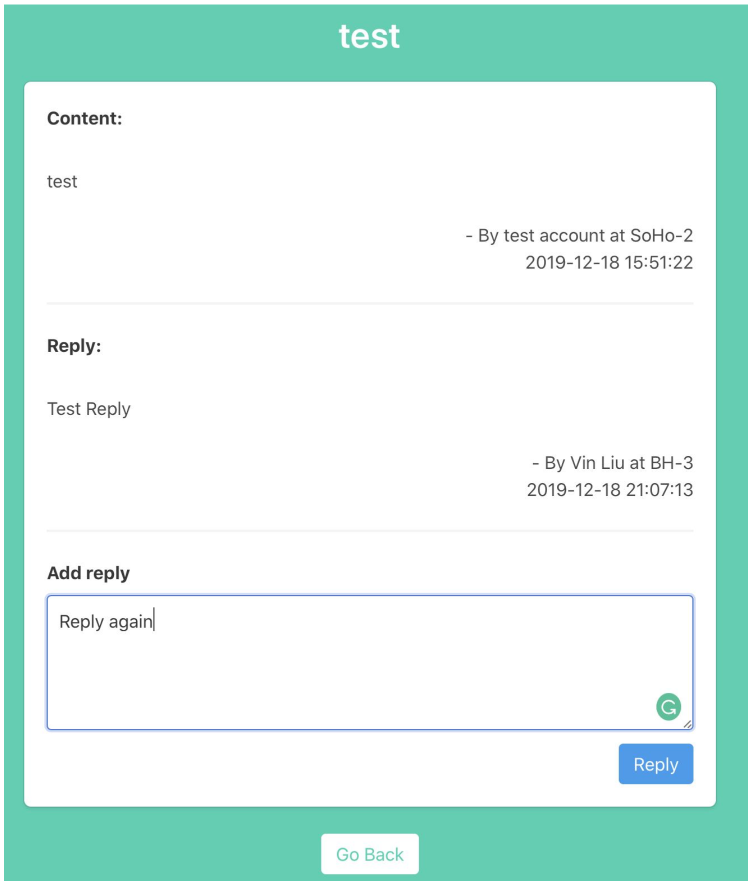
Longitude: Longitude (optional)

Hood [✓ Direct] Block Friend — test2@test.com

Post | Back

Click Timeline -> Create to compose a new post. The user can choose who can see the post. If the user wants to send a direct message, an email of the recipient user is needed.

- View a Message Detail and Reply a Message



test

Content:

test

- By test account at SoHo-2
2019-12-18 15:51:22

Reply:

Test Reply

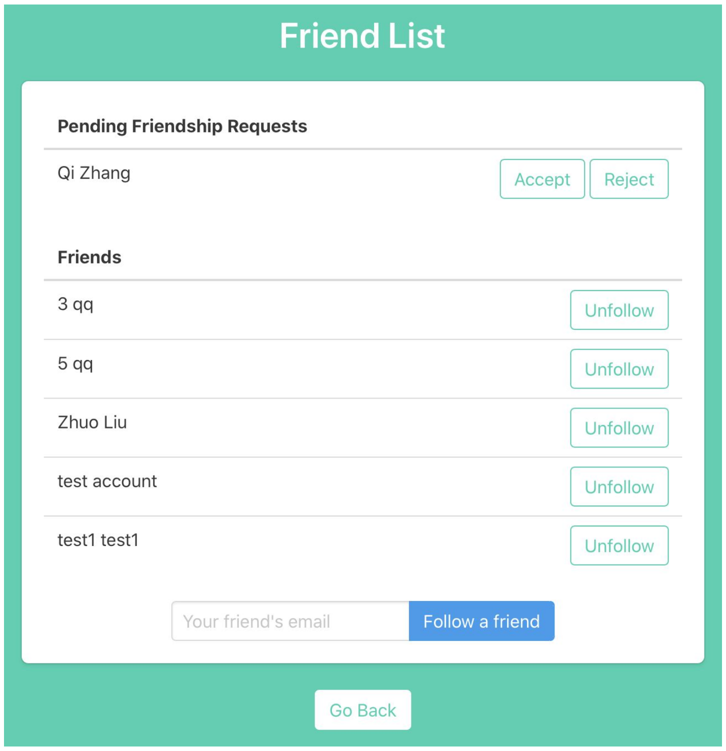
- By Vin Liu at BH-3
2019-12-18 21:07:13

Add reply

Reply again

Click the title of any message on the timeline to see the content of the message, along with all existing replies. The user can also reply below the message.

- Friend List



Friend List

Pending Friendship Requests

Qi Zhang

Friends

3 qq

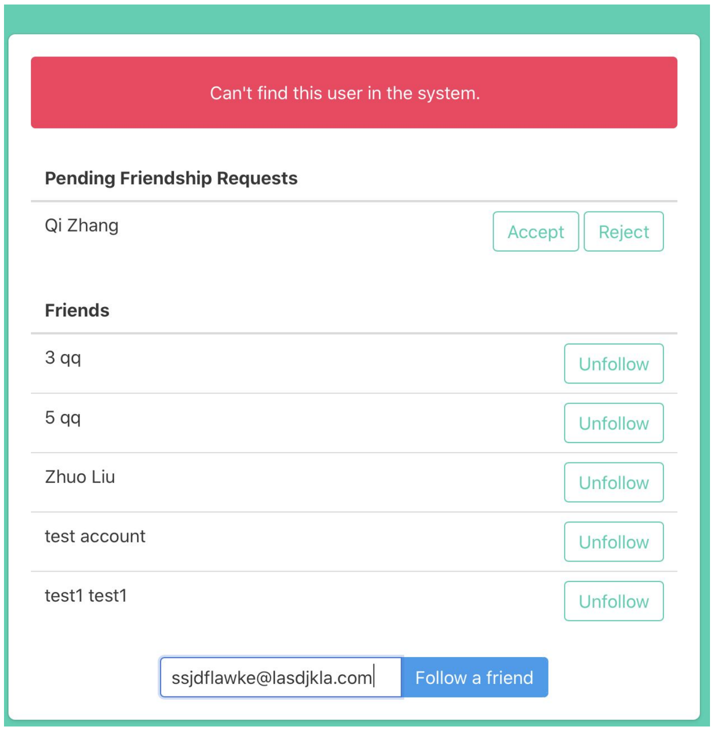
5 qq

Zhuo Liu

test account

test1 test1

Click Follow -> Friend to see a friend list. Pending friendship request is listed on the top in the friend list page. The user can also add a new friend via email. Here we assume that if one user wants to send a friendship request to others, they have to know each other and their email addresses. A friendship request will be sent to the followee. Also, adding an unregistered user is not allowed.



Can't find this user in the system.

Pending Friendship Requests

Qi Zhang

Friends

3 qq

5 qq

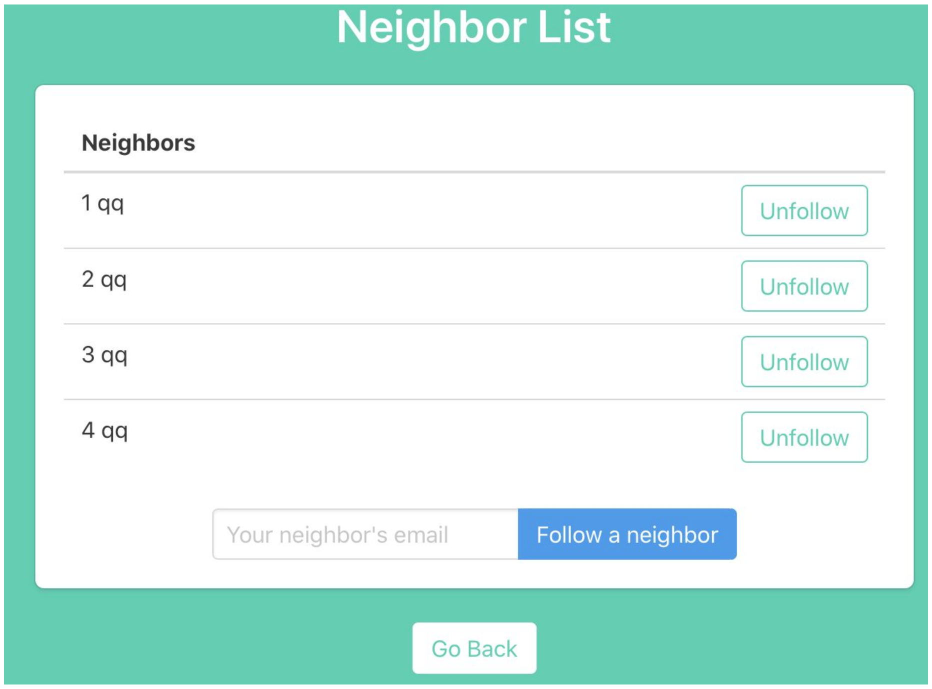
Zhuo Liu

test account

test1 test1

ssjdflawke@lasdjkla.com

- Neighbor List



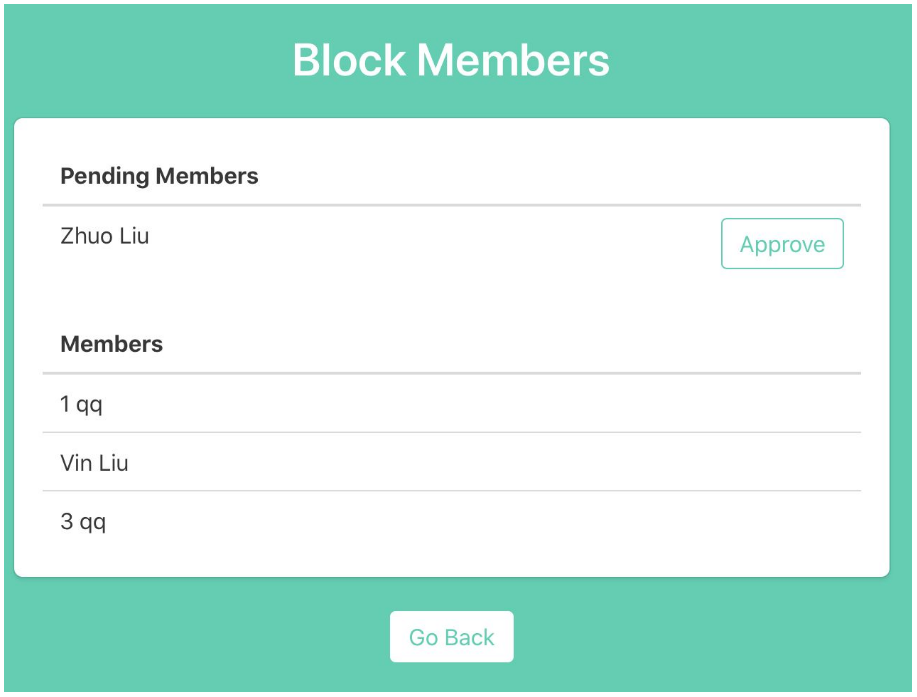
Neighbor List

Neighbors

1 qq	Unfollow
2 qq	Unfollow
3 qq	Unfollow
4 qq	Unfollow

Click Follow -> Neighbor to view a neighbor list. Functions are similar to Friend List.

- View or Approve Block Members



Block Members

Pending Members

Zhuo Liu	Approve

Members

1 qq

Vin Liu

3 qq

Click Follow -> Block Member to view a list of members in the user's block. Pending membership is listed at the top part. The user can approve the pending membership.

3. Addressing Security Concerns

- The SQL injection is prohibited by using flask-sqlalchemy module to handle all queries. We are not writing raw SQL statements to communicate with the database. Any special characters are automatically escaped and quoted by the SQL engine object. Also, the SQL statement is also automatically prepared by SQLalchemy.
- XSS is handled by flask-login module. (https://flask.palletsprojects.com/en/1.0.x/security/)
- All passwords are hashed before storing them into the database.
- The application is divided into Unauth & Auth parts. Any action in the Auth part can only be performed with a valid user session.