Principle of Database

CS6083 Spring 2013

Jingo Project

2

Ching-Che Chang 0435353
Jiankai Dang 0482923

Ching-Che Chang 0435353
Jiankai Dang 0482923

Table of Contents

Ching-Che Chang 0435353
Jiankai Dang 0482923

I. Introduction

Jingo web application is a mobile-like application that makes people share information with others easily. Using Jingo, people are able to publish, receive, and search useful notes on Google Maps directly, an intuitive and convenient way that users do not need to go through web pages back and forth. If you have not found an app to help you out, try Jingo without hesitation.

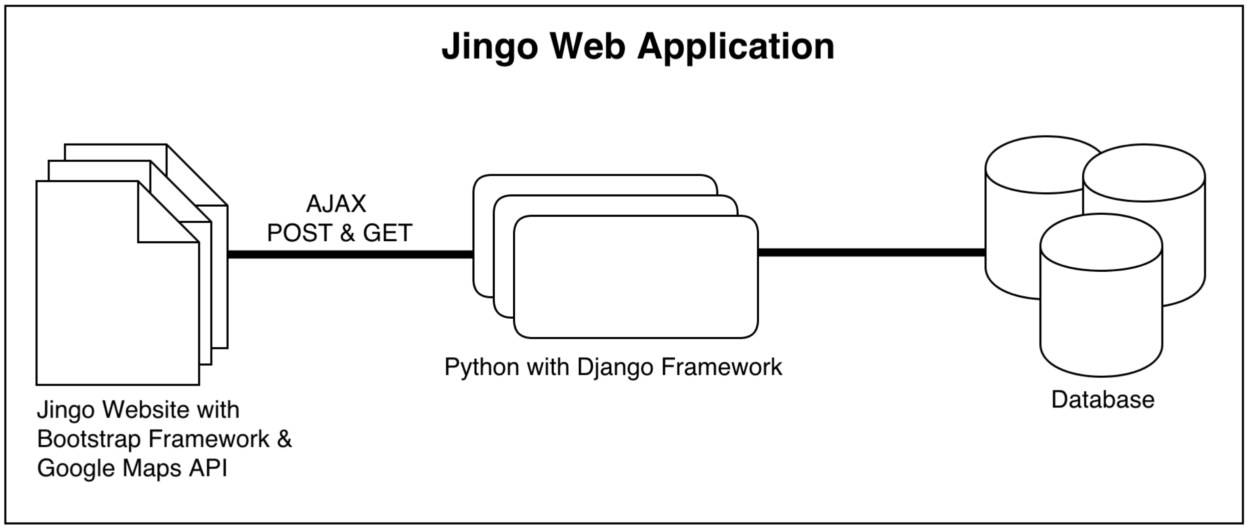
This document provide you with all the details about Jingo project, including the overall view of the Jingo application, database design and testing, the techniques used in back-end and front-end design, and finally system demo showing you how to use Jingo to make your life better than ever.

II. System Design

a. System Architecture

Jingo web application is designed with Django framework based on MVC architecture, including model design that handling data storage between database and the application, view design that is in charge of user interface, and control design that makes model and view perfectly cowork together.

In the front-end design, several techniques such as HTML5, CSS3, JavaScript, Bootstrap and Google Maps API will contribute to Jingo web application. Based on Google Maps API, Jingo application will co-operate with Google Maps to provide subscribers with a new user experience. For the back-end design, Python will play a lead role to handle requests from front end and deal with data access to MySQL.



b. Assumptions on System Design

For basic requirements:

- We use our own database system, MySQL, on our laptops.
- There will be a login page, a page where a user can sign up for the first time (by supplying an email address and choosing a user name), and a page where users can create or update their profiles. Email address must be unique. User name does not need to be unique.
- Users can publish information (small messages or notes). There is a length limit of 140 characters, similar to an SMS or tweet.
- These notes are linked to certain times, a schedule when the note can be seen, on a particular date, or "every day between 2pm and 3pm", or "on 2/15/2013 from 4pm to 5pm".
- Notes could be received by friends or everybody. Notes could also be private.
- Other users will also be able to attach comments to the note, if the user allows it, such as "Thanks, great restaurant!".
- Users are able to ask other users to become friends, and answer friend requests without checking their emails. There are 3 states of friend requests: pending, accepted, and denied.
- Users can have different filters (sets of rules) on what they want to see based on their current state (such as "at work", "lunch break", "just chilling", or whatever they choose as the description), the current time, and their current location. Users will have their default public state, once they sign up. Based on the default public state, users can receive any note according to their current time, and their current location. They can also create their customized states and change one of the customized states into their current state.

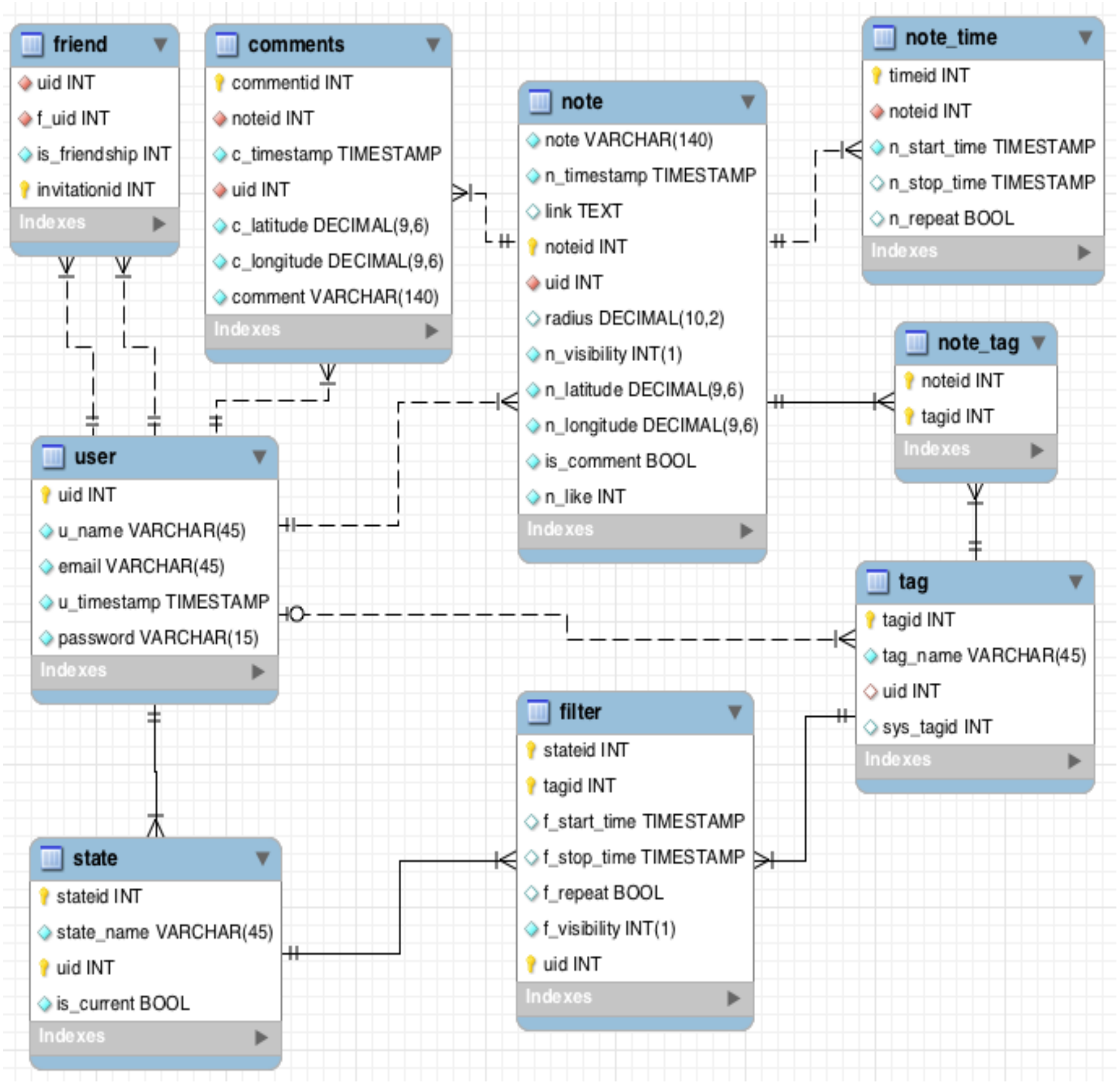
For extra credit:

- We will provide a smart way to model areas (in SoHo) through Google API, allowing us not to develop complicated schemas to handle data storage and reducing the time of development.
- Certain common (system default) tags, such as #me, #food, #tourism, or #shopping, are predefined and others can be invented as the user wants to. User invented tags will inherit from a specific system default tag, in order to establish a relation between them.
- We will add a "like" button so that other users could rate the usefulness of a note. The number of like for a note is used as a strategy to filter and sort, especially when users would like to search notes without entering keywords.
- We will always store timestamps and locations when a note is posted by a user.
- There will not be a separate DBMS account for each user, but the web interface and application itself will log into the database. So, the system we implement can see all the content, but at the application level each logged-in user is identified through the use of cookies in the second part of the project.
- We will not use database permissions or views to implement content access restrictions.

We will design a novel vision for users based on Google Map, making Jingo nice and easy to use.

C. Database Design

i ER Model



ii Relational Schema of Database

user(uid, u_name, email, u_timestamp, password)

note(noteid, note, n_timestamp, link, uid, radius, n_visibility, n_latitude, n_longitude, is_comment, n_like)
uid references uid in user

tag(tagid, tag_name, uid, sys_tagid)
uid references uid in user

friend(invitationid, uid, f_uid, is_friendship)
uid references uid in user
f_uid references uid in user

note_time(timeid, noteid, n_start_time, n_stop_time, n_repeat)
noteid references noteid in note

state(stateid, uid, state_name, is_current)
uid references uid in user

filter(stateid, tagid, uid, f_start_time, f_stop_time, f_repeat, f_visibility)
(stateid, uid) references (stateid, uid) in state
tagid references tagid in tag

note_tag(noteid, tagid)
noteid references noteid in note
tagid references tagid in tag

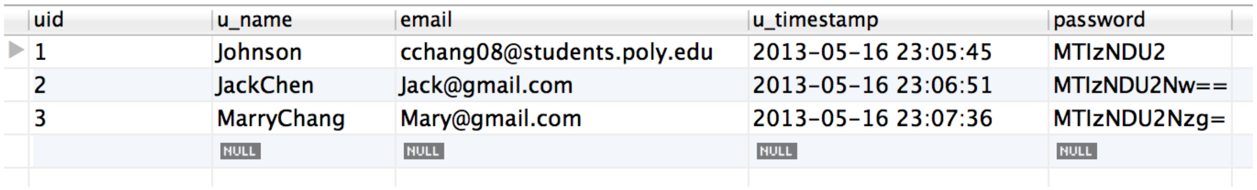
comments(commentid, noteid, c_timestamp, uid, c_latitude, c_longitude, comment)
noteid references noteid in note

Ching-Che Chang 0435353
Jiankai Dang 0482923

d. Testing on Database

i Signing Up and Posting

- **sign up**

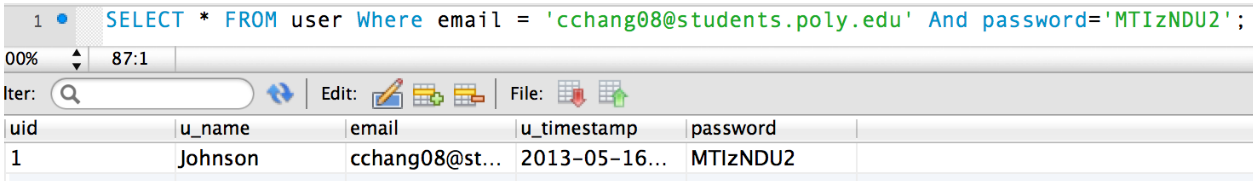


uid	u_name	email	u_timestamp	password
1	Johnson	cchang08@students.poly.edu	2013-05-16 23:05:45	MTIzNDU2
2	JackChen	Jack@gmail.com	2013-05-16 23:06:51	MTIzNDU2Nw==
3	MarryChang	Mary@gmail.com	2013-05-16 23:07:36	MTIzNDU2Nzg=
	NULL	NULL	NULL	NULL

The structure of the data:

- uid
  The unique id of the user.
- u_name
  The name of the user. u_name does not need to be unique.
- email
  The email address of the user.
- password
  The password of the user.
- u_timestamp
  The timestamp when the user sign up.

- **login**



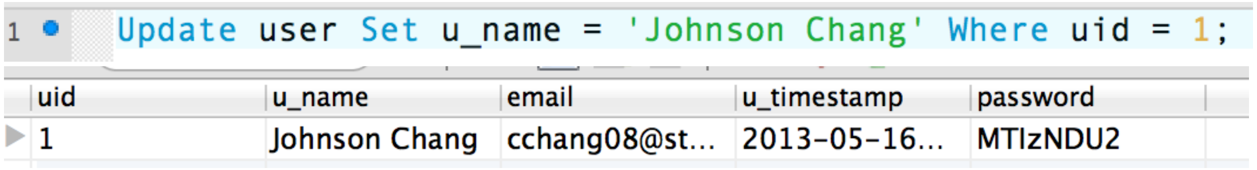
```
SELECT * FROM user Where email = 'cchang08@students.poly.edu' And password='MTIzNDU2';
```

00% 87:1

lter: Edit: File:

uid	u_name	email	u_timestamp	password
1	Johnson	cchang08@st...	2013-05-16...	MTIzNDU2

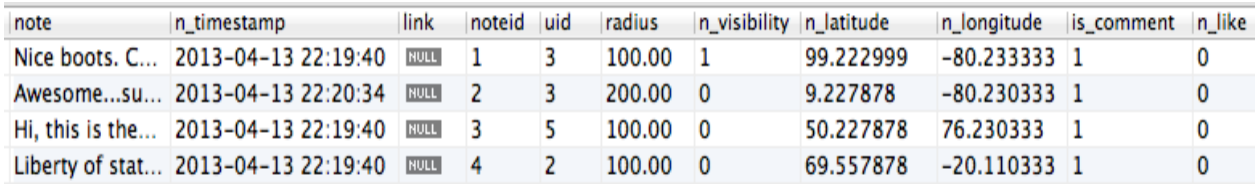
- **edit profile**



```
Update user Set u_name = 'Johnson Chang' Where uid = 1;
```

uid	u_name	email	u_timestamp	password
1	Johnson Chang	cchang08@st...	2013-05-16...	MTIzNDU2

- **post notes**

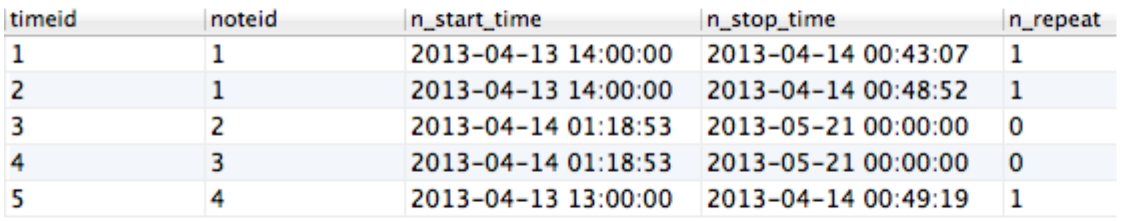


note	n_timestamp	link	noteid	uid	radius	n_visibility	n_latitude	n_longitude	is_comment	n_like
Nice boots. C...	2013-04-13 22:19:40	NULL	1	3	100.00	1	99.222999	-80.233333	1	0
Awesome...su...	2013-04-13 22:20:34	NULL	2	3	200.00	0	9.227878	-80.230333	1	0
Hi, this is the...	2013-04-13 22:19:40	NULL	3	5	100.00	0	50.227878	76.230333	1	0
Liberty of stat...	2013-04-13 22:19:40	NULL	4	2	100.00	0	69.557878	-20.110333	1	0

The structure of the data:

- noteid
  The unique id of the note.
- note
  The small message published by the user.

- n_latitude
  The latitude of the user current location.
- n_longitude
  The longitude of the user current location.
- radius
  The radius of interest around the location, where the note can be seen, in units of yard.
- n_visibility
  The visibility of this note.
  - 0: public
  - 1: friend
  - 2: private
- link
  Hyperlink information with this note. link could be null.
- n_like
  The number of likes this note has received.
- is_comment
  Whether the user allows other users to attach comments to the note.
- n_timestamp
  The timestamp when the user publish this note.

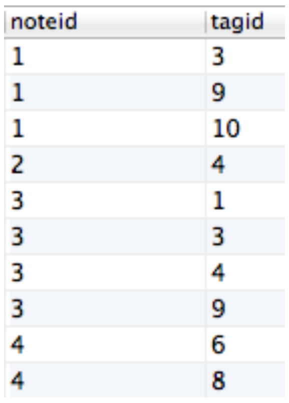


timeid	noteid	n_start_time	n_stop_time	n_repeat
1	1	2013-04-13 14:00:00	2013-04-14 00:43:07	1
2	1	2013-04-13 14:00:00	2013-04-14 00:48:52	1
3	2	2013-04-14 01:18:53	2013-05-21 00:00:00	0
4	3	2013-04-14 01:18:53	2013-05-21 00:00:00	0
5	4	2013-04-13 13:00:00	2013-04-14 00:49:19	1

The structure of the data:

- timeid
  The unique id of the schedule.
- noteid
  The id of the note, to which this schedule belongs.
- n_start_time
  The start time of the schedule.
- n_stop_time
  The stop time of the schedule.
- n_repeat
  The repeat type of the schedule.
  - 0: repeat
  - 1: no repeat

Ching-Che Chang 0435353
Jiankai Dang 0482923

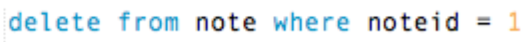


noteid	tagid
1	3
1	9
1	10
2	4
3	1
3	3
3	4
3	9
4	6
4	8

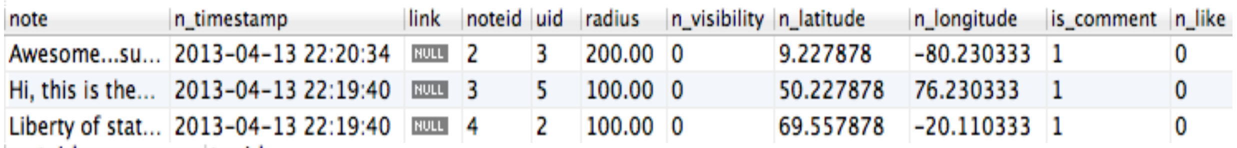
The structure of the data:

- noteid
  The id of the note, to which this tag belongs.
- tagid
  The unique id of the tag.

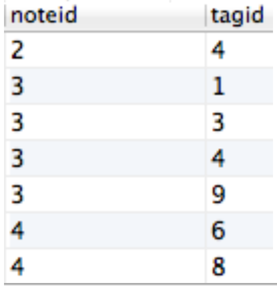
- **delete notes**



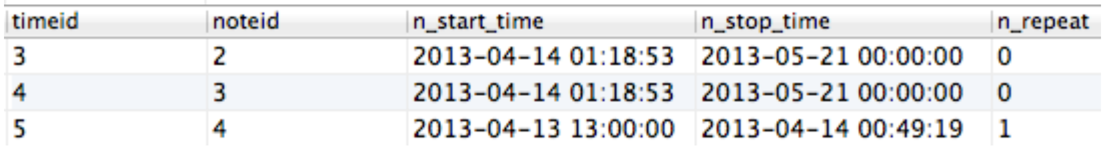
```
delete from note where noteid = 1
```



note	n_timestamp	link	noteid	uid	radius	n_visibility	n_latitude	n_longitude	is_comment	n_like
Awesome...su...	2013-04-13 22:20:34	NULL	2	3	200.00	0	9.227878	-80.230333	1	0
Hi, this is the...	2013-04-13 22:19:40	NULL	3	5	100.00	0	50.227878	76.230333	1	0
Liberty of stat...	2013-04-13 22:19:40	NULL	4	2	100.00	0	69.557878	-20.110333	1	0



noteid	tagid
2	4
3	1
3	3
3	4
3	9
4	6
4	8

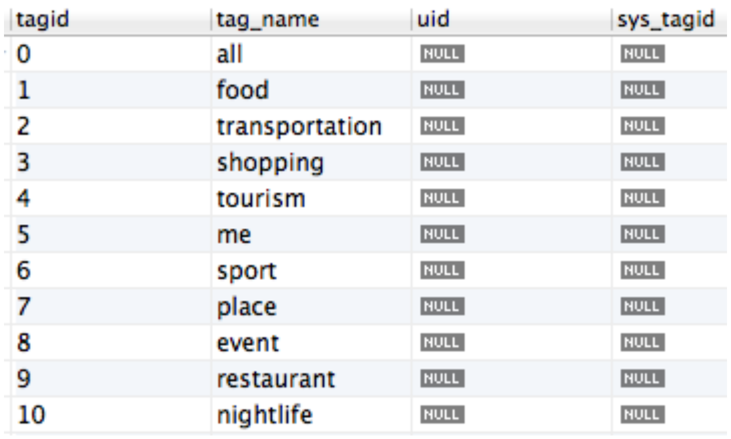


timeid	noteid	n_start_time	n_stop_time	n_repeat
3	2	2013-04-14 01:18:53	2013-05-21 00:00:00	0
4	3	2013-04-14 01:18:53	2013-05-21 00:00:00	0
5	4	2013-04-13 13:00:00	2013-04-14 00:49:19	1

Ching-Che Chang 0435353
Jiankai Dang 0482923

ii Filters

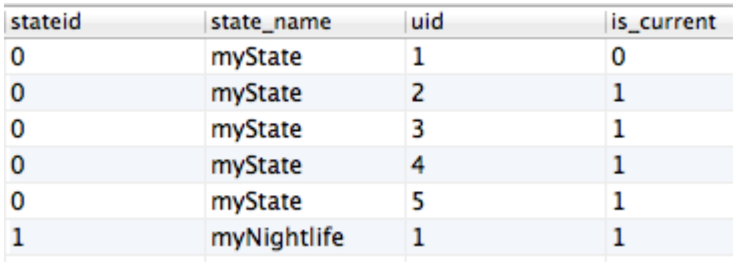
- **system default tags**



tagid	tag_name	uid	sys_tagid
0	all	NULL	NULL
1	food	NULL	NULL
2	transportation	NULL	NULL
3	shopping	NULL	NULL
4	tourism	NULL	NULL
5	me	NULL	NULL
6	sport	NULL	NULL
7	place	NULL	NULL
8	event	NULL	NULL
9	restaurant	NULL	NULL
10	nightlife	NULL	NULL

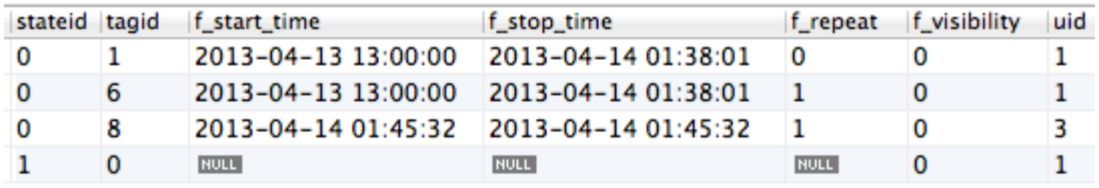
The structure of the data:
  - tagid
    The unique id of this tag.
  - tag_name
    The name of this tag.
  - uid
    The id of the user who created this tag.
  - sys_tagid
    The id of the system default tag, from which this tag inherits.

- **create filters**



stateid	state_name	uid	is_current
0	myState	1	0
0	myState	2	1
0	myState	3	1
0	myState	4	1
0	myState	5	1
1	myNightlife	1	1

The structure of the data:
  - stateid
    The unique id of the user state.
  - state_name
    The name of the user state.
  - uid
    The id of the user who created this state.
  - is_current
    Whether this state is the current user state.
      - 0: not current state
      - 1: current state



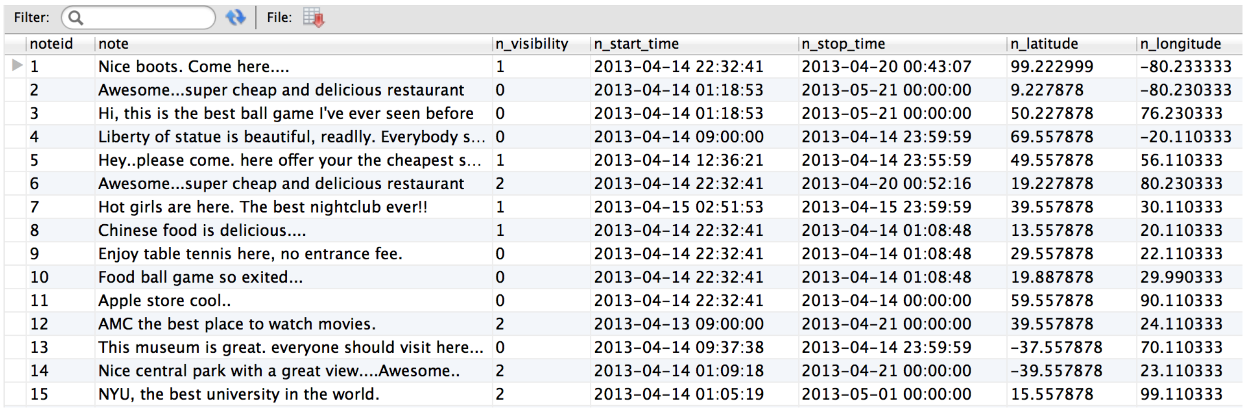
stateid	tagid	f_start_time	f_stop_time	f_repeat	f_visibility	uid
0	1	2013-04-13 13:00:00	2013-04-14 01:38:01	0	0	1
0	6	2013-04-13 13:00:00	2013-04-14 01:38:01	1	0	1
0	8	2013-04-14 01:45:32	2013-04-14 01:45:32	1	0	3
1	0	NULL	NULL	NULL	0	1

The structure of the data:

- stateid
  The id of the state, to which this filter belongs.
- tagid
  The id of the tag, to which this filter is tagged.
- uid
  The id of the user who created this filter.
- f_start_time
  The start time of the schedule for this filter.
- f_stop_time
  The stop time of the schedule for this filter.
- f_repeat
  The repeat type of the schedule for this filter.
  - 0: repeat
  - 1: no repeat
- f_visibility
  What kind of notes this filter will see.
  - 0: public
  - 1: friend
  - 2: private

iii Finding Notes

- find a note



Filter: File:

noteid	note	n_visibility	n_start_time	n_stop_time	n_latitude	n_longitude
1	Nice boots. Come here....	1	2013-04-14 22:32:41	2013-04-20 00:43:07	99.222999	-80.233333
2	Awesome...super cheap and delicious restaurant	0	2013-04-14 01:18:53	2013-05-21 00:00:00	9.227878	-80.230333
3	Hi, this is the best ball game I've ever seen before	0	2013-04-14 01:18:53	2013-05-21 00:00:00	50.227878	76.230333
4	Liberty of statue is beautiful, readlly. Everybody s...	0	2013-04-14 09:00:00	2013-04-14 23:59:59	69.557878	-20.110333
5	Hey..please come. here offer your the cheapest s...	1	2013-04-14 12:36:21	2013-04-14 23:55:59	49.557878	56.110333
6	Awesome...super cheap and delicious restaurant	2	2013-04-14 22:32:41	2013-04-20 00:52:16	19.227878	80.230333
7	Hot girls are here. The best nightclub ever!!	1	2013-04-15 02:51:53	2013-04-15 23:59:59	39.557878	30.110333
8	Chinese food is delicious....	1	2013-04-14 22:32:41	2013-04-14 01:08:48	13.557878	20.110333
9	Enjoy table tennis here, no entrance fee.	0	2013-04-14 22:32:41	2013-04-14 01:08:48	29.557878	22.110333
10	Food ball game so exited...	0	2013-04-14 22:32:41	2013-04-14 01:08:48	19.887878	29.990333
11	Apple store cool..	0	2013-04-14 22:32:41	2013-04-14 00:00:00	59.557878	90.110333
12	AMC the best place to watch movies.	2	2013-04-13 09:00:00	2013-04-21 00:00:00	39.557878	24.110333
13	This museum is great. everyone should visit here...	0	2013-04-14 09:37:38	2013-04-14 23:59:59	-37.557878	70.110333
14	Nice central park with a great view....Awesome..	2	2013-04-14 01:09:18	2013-04-21 00:00:00	-39.557878	23.110333
15	NYU, the best university in the world.	2	2013-04-14 01:05:19	2013-05-01 00:00:00	15.557878	99.110333

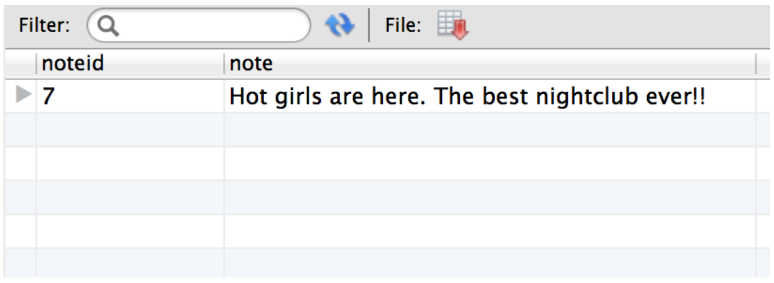
The structure of the data:

- noteid
  The unique id of the note.
- note
  The small message published by the user.
- n_visibility
  The visibility of this note.
  - 0: public
  - 1: friend
  - 2: private
- n_start_time
  The start time of the note allows other users to read
- n_stop_time
  The stop time of the note allows other users to read
- n_latitude
  The latitude of the user current location.
- n_longitude
  The longitude of the user current location.

Assume that the user with the uid '**2**' get into the area (**39.557878, 30.110333**) in the current time between **2013/04/15 03:00:00** and **2013/04/15 23:59:59**. From the assumption, the result should be the note with noteid '**7**'. Now, we put all the conditions into SQL statement as below and then execute it.

```sql
Select k.noteid, k.note
From (
/*
notes only have system_tags
*/
(Select a.uid, a.noteid, a.note, a.n_visibility, c.tagid, d.sys_tagid,
b.n_start_time, b.n_stop_time, a.n_latitude, a.n_longitude
From note as a, note_time as b, note_tag as c, tag as d
Where
a.noteid = b.noteid And a.noteid = c.noteid And
c.tagid = d.tagid And d.sys_tagid is Null
Group By a.noteid, c.tagid)
Union
/*
notes only have user-defined tags or have both sys and user-defined
*/
(Select a.uid, a.noteid, a.note, a.n_visibility, d.sys_tagid, d.sys_tagid,
b.n_start_time, b.n_stop_time, a.n_latitude, a.n_longitude
From note as a, note_time as b, note_tag as c, tag as d
Where
a.noteid = b.noteid And a.noteid = c.noteid And
c.tagid = d.tagid And d.sys_tagid is Not Null
Group By a.noteid, c.tagid)) as k,
friend as b
Where k.uid = b.uid And b.f_uid = 2 And
k.n_latitude = '39.557878' And k.n_longitude = '30.110333' And
k.tagid in (
/*
a filter of a specific user with sys_tagid
*/
Select 0
Union
(Select b.tagid
From state as a, filter as b, tag as c
Where a.stateid = b.stateid And b.tagid = c.tagid And b.uid = a.uid And
a.is_current = 1 And a.uid = 2)
) And
k.noteid in (
Select distinct a.noteid
From note as a, note_time as b
Where a.noteid = b.noteid And
(Now() Between b.n_start_time And b.n_stop_time)
Order By a.noteid
)
```

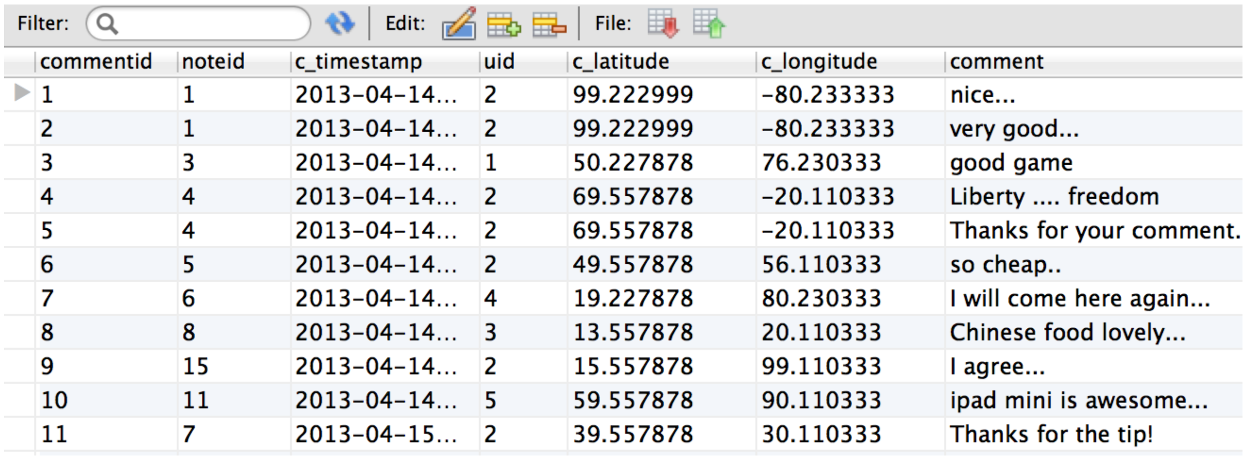
Here is the result after the SQL statement above was executed.



noteid	note
7	Hot girls are here. The best nightclub ever!!

- **add a comment**

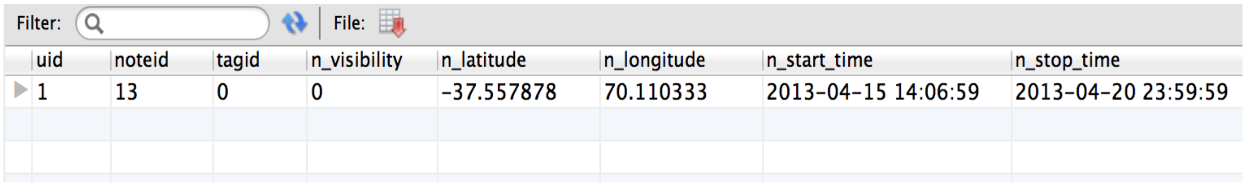
```
INSERT INTO comments(
`commentid`, `noteid`, `c_timestamp`, `uid`,
`c_latitude`, `c_longitude`, `comment`)
VALUES
(11, 7, Now(), 2, '39.557878', '30.110333','Thanks for the tip!')
```



commentid	noteid	c_timestamp	uid	c_latitude	c_longitude	comment
1	1	2013-04-14...	2	99.222999	-80.233333	nice...
2	1	2013-04-14...	2	99.222999	-80.233333	very good...
3	3	2013-04-14...	1	50.227878	76.230333	good game
4	4	2013-04-14...	2	69.557878	-20.110333	Liberty .... freedom
5	4	2013-04-14...	2	69.557878	-20.110333	Thanks for your comment.
6	5	2013-04-14...	2	49.557878	56.110333	so cheap..
7	6	2013-04-14...	4	19.227878	80.230333	I will come here again...
8	8	2013-04-14...	3	13.557878	20.110333	Chinese food lovely...
9	15	2013-04-14...	2	15.557878	99.110333	I agree...
10	11	2013-04-14...	5	59.557878	90.110333	ipad mini is awesome...
11	7	2013-04-15...	2	39.557878	30.110333	Thanks for the tip!

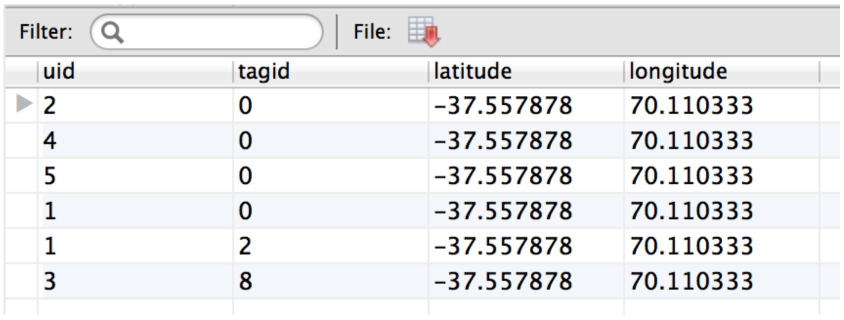
iv Reverse Finding Notes

Suppose we have a note as below and we try to output a list of all readers who can see this note under their own filters. In addition, this note has a value of visibility zero, which means the note is a public note that anyone can see if the location is matched.



uid	noteid	tagid	n_visibility	n_latitude	n_longitude	n_start_time	n_stop_time
1	13	0	0	-37.557878	70.110333	2013-04-15 14:06:59	2013-04-20 23:59:59

Here are the users who are in the area and some of them would like to see public notes with tagid '0' while the others would like specific notes with tagid **'2'** or **'8'**. In the above assumption, we are supposed to get uids **'1'**, **'2'**, **'4'**, and **'5'**.

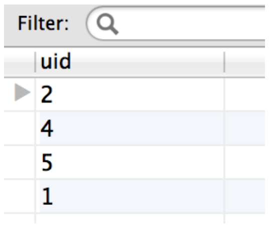


uid	tagid	latitude	longitude
2	0	-37.557878	70.110333
4	0	-37.557878	70.110333
5	0	-37.557878	70.110333
1	0	-37.557878	70.110333
1	2	-37.557878	70.110333
3	8	-37.557878	70.110333

Now, we start to filter all users for the note using the following rules. (n_latitude, n_longitude) = (**-37.557878, 70.110333**) is viewed as users' current location. (**n_start_time, n_stop_time**) is a time span that users can read the notes, so the current time that users get in the area should be between this time span of the notes. Finally, we still need to check whether or not users' personal filters are matched with the setting of visibility of the note.

```
Select Distinct m.uid
From (Select j.uid, j.noteid, j.tagid, n_latitude, n_longitude, n_start_time, n_stop_time
    From
        (Select uid, noteid, tagid, n_visibility, n_latitude, n_longitude, n_start_time, n_stop_time
        From
        ((Select
        a.uid, a.noteid, a.note, a.n_visibility, c.tagid, d.sys_tagid, b.n_start_time,
        b.n_stop_time, a.n_latitude, a.n_longitude
        From note as a, note_time as b, note_tag as c, tag as d
        Where a.noteid = b.noteid And a.noteid = c.noteid
            And c.tagid = d.tagid
            And d.sys_tagid is Null
        Group By a.noteid , c.tagid)
        Union (
        Select
            a.uid, a.noteid, a.note, a.n_visibility, d.sys_tagid, d.sys_tagid,
            b.n_start_time, b.n_stop_time, a.n_latitude, a.n_longitude
        From note as a, note_time as b, note_tag as c, tag as d
        Where a.noteid = b.noteid And a.noteid = c.noteid
            And c.tagid = d.tagid
            And d.sys_tagid is Not Null
        Group By a.noteid , c.tagid)) as t
        Where noteid = 13) as j, friend as k
    Where j.uid = k.uid) as l,
    ((Select uid, 0 as tagid, - 37.557878 as latitude, 70.110333 as longitude
    From
        state
    Where stateid = 0 And is_current = 1 And uid Not In (
            Select a.uid
            From filter as a, state as b, tag as c
            Where
                a.stateid = b.stateid And a.tagid = c.tagid
                And b.uid = a.uid And b.is_current = 1
                And c.sys_tagid is Null))
    Union (
    Select a.uid, a.tagid, - 37.557878 as latitude, 70.110333 as longitude
    From filter as a, state as b, tag as c
    Where a.stateid = b.stateid And a.tagid = c.tagid And b.uid = a.uid
        And b.is_current = 1 And c.sys_tagid is Null)) as m
Where l.tagid = m.tagid And latitude = '-37.557878' And longitude = '70.110333'
        And (Now() Between n_start_time And n_stop_time)
```

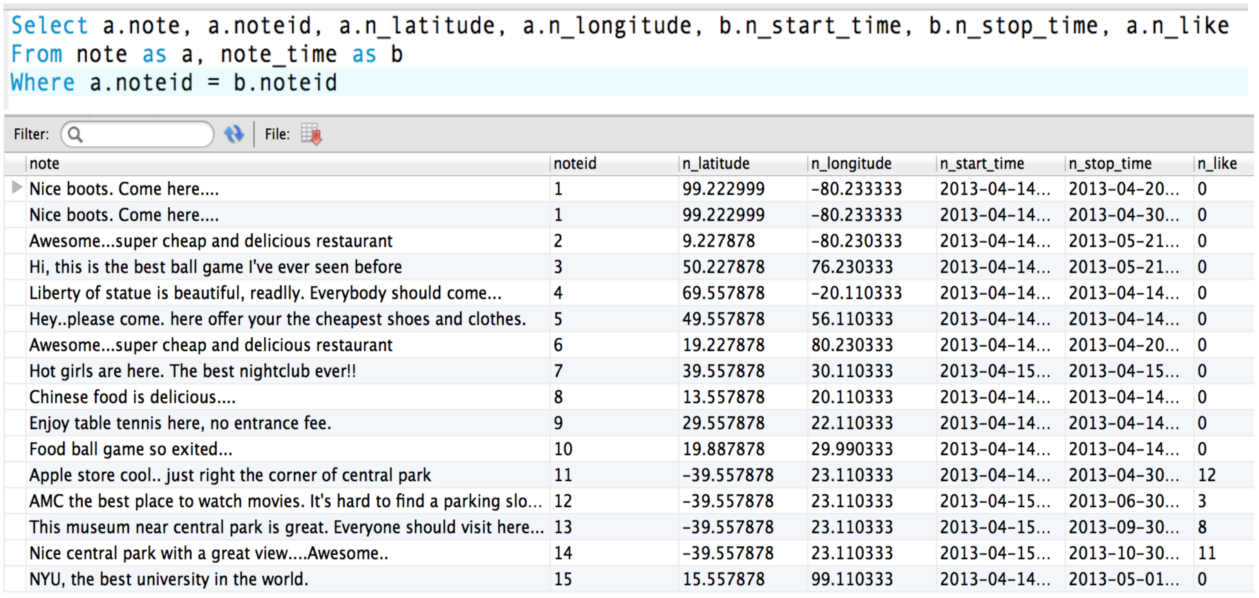
After the above SQL statement executed, we can get the result as below. The result is the same as our previous expectation.



```
Filter:
uid
2
4
5
1
```

V Searching Notes

Suppose that the user only defined a general filter to receive public notes from posters within a specific area and the user would like to read the notes containing the keyword '**park**'. First, here are all the public notes for receivers.

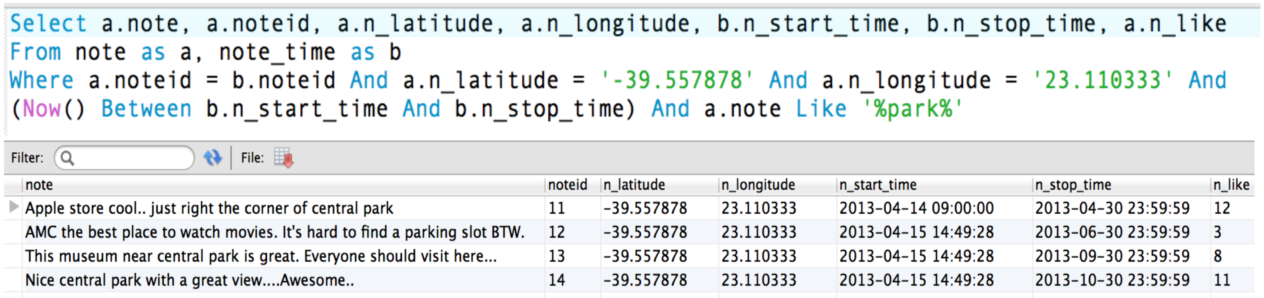


```sql
Select a.note, a.noteid, a.n_latitude, a.n_longitude, b.n_start_time, b.n_stop_time, a.n_like
From note as a, note_time as b
Where a.noteid = b.noteid
```

Filter: File:

note	noteid	n_latitude	n_longitude	n_start_time	n_stop_time	n_like
Nice boots. Come here....	1	99.222999	-80.233333	2013-04-14...	2013-04-20...	0
Nice boots. Come here....	1	99.222999	-80.233333	2013-04-14...	2013-04-30...	0
Awesome...super cheap and delicious restaurant	2	9.227878	-80.230333	2013-04-14...	2013-05-21...	0
Hi, this is the best ball game I've ever seen before	3	50.227878	76.230333	2013-04-14...	2013-05-21...	0
Liberty of statue is beautiful, readlly. Everybody should come...	4	69.557878	-20.110333	2013-04-14...	2013-04-14...	0
Hey..please come. here offer your the cheapest shoes and clothes.	5	49.557878	56.110333	2013-04-14...	2013-04-14...	0
Awesome...super cheap and delicious restaurant	6	19.227878	80.230333	2013-04-14...	2013-04-20...	0
Hot girls are here. The best nightclub ever!!	7	39.557878	30.110333	2013-04-15...	2013-04-15...	0
Chinese food is delicious....	8	13.557878	20.110333	2013-04-14...	2013-04-14...	0
Enjoy table tennis here, no entrance fee.	9	29.557878	22.110333	2013-04-14...	2013-04-14...	0
Food ball game so exited...	10	19.887878	29.990333	2013-04-14...	2013-04-14...	0
Apple store cool.. just right the corner of central park	11	-39.557878	23.110333	2013-04-14...	2013-04-30...	12
AMC the best place to watch movies. It's hard to find a parking slo...	12	-39.557878	23.110333	2013-04-15...	2013-06-30...	3
This museum near central park is great. Everyone should visit here...	13	-39.557878	23.110333	2013-04-15...	2013-09-30...	8
Nice central park with a great view....Awesome..	14	-39.557878	23.110333	2013-04-15...	2013-10-30...	11
NYU, the best university in the world.	15	15.557878	99.110333	2013-04-14...	2013-05-01...	0

Now, we start to filter all notes for receivers using the following rules. (n_latitude, n_longitude) = (**-39.557878, 23.110333**) is viewed as receivers' current location. (**n_start_time, n_stop_time**) is a time span that receivers can read the notes, so the current time that receivers get in the area should be between this time span of the notes. Finally, we need to consider the keyword '**park**'.



```sql
Select a.note, a.noteid, a.n_latitude, a.n_longitude, b.n_start_time, b.n_stop_time, a.n_like
From note as a, note_time as b
Where a.noteid = b.noteid And a.n_latitude = '-39.557878' And a.n_longitude = '23.110333' And
(Now() Between b.n_start_time And b.n_stop_time) And a.note Like '%park%'
```

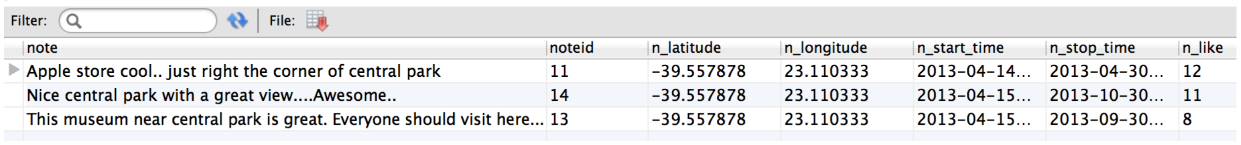
Filter: File:

note	noteid	n_latitude	n_longitude	n_start_time	n_stop_time	n_like
Apple store cool.. just right the corner of central park	11	-39.557878	23.110333	2013-04-14 09:00:00	2013-04-30 23:59:59	12
AMC the best place to watch movies. It's hard to find a parking slot BTW.	12	-39.557878	23.110333	2013-04-15 14:49:28	2013-06-30 23:59:59	3
This museum near central park is great. Everyone should visit here...	13	-39.557878	23.110333	2013-04-15 14:49:28	2013-09-30 23:59:59	8
Nice central park with a great view....Awesome..	14	-39.557878	23.110333	2013-04-15 14:49:28	2013-10-30 23:59:59	11

For extra credit, if the user did not supply keywords, we will use ***the number of like*** on the notes from past readers to rank all the notes from most to least important. Then, we will use a threshold, such as 5 in this case, to filter out all the notes again.

Ching-Che Chang 0435353
Jiankai Dang 0482923

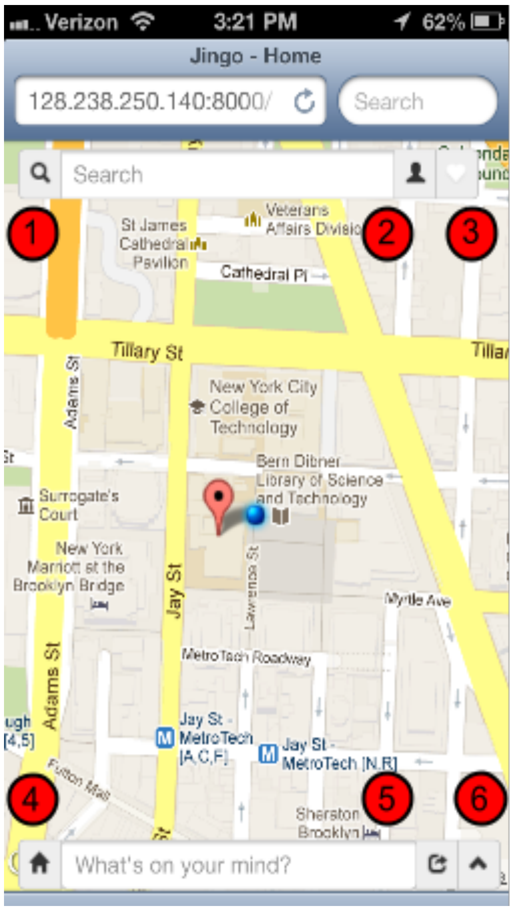
```
Select a.note, a.noteid, a.n_latitude, a.n_longitude, b.n_start_time, b.n_stop_time, a.n_like
From note as a, note_time as b
Where a.noteid = b.noteid And a.n_latitude = '-39.557878' And a.n_longitude = '23.110333' And
(Now() Between b.n_start_time And b.n_stop_time) And a.n_like >= 5
Order By a.n_like DESC
```


Filter:
File:

note	noteid	n_latitude	n_longitude	n_start_time	n_stop_time	n_like
Apple store cool.. just right the corner of central park	11	-39.557878	23.110333	2013-04-14...	2013-04-30...	12
Nice central park with a great view....Awesome..	14	-39.557878	23.110333	2013-04-15...	2013-10-30...	11
This museum near central park is great. Everyone should visit here...	13	-39.557878	23.110333	2013-04-15...	2013-09-30...	8

III. System Demo

a. Legend of Jingo



1. Search button: used for searching notes by keywords.

2. Profile button: used for setting user-defined filter of notes users are interested in.

3. Friend button: used for managing the circle of your friendship.

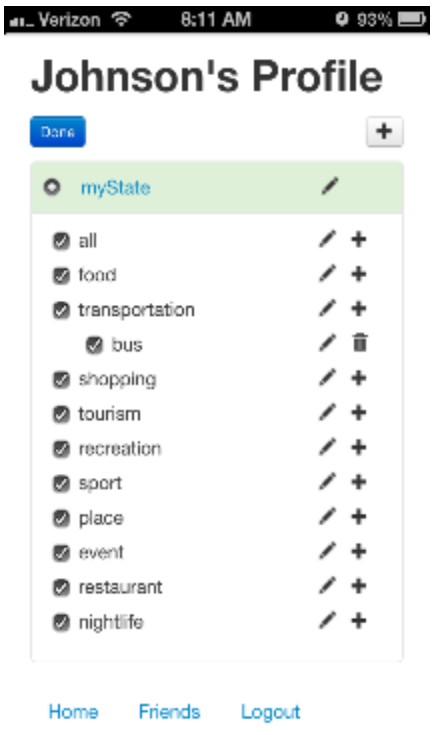
4. Home button: used for returning to your actual location.

5. Quick posting button: used for publishing and sharing notes.

6. Advanced posting button: used for posting notes with the setting features of schedule, visibilities, and tags.

b. Feature Introduction

i Profile Setting

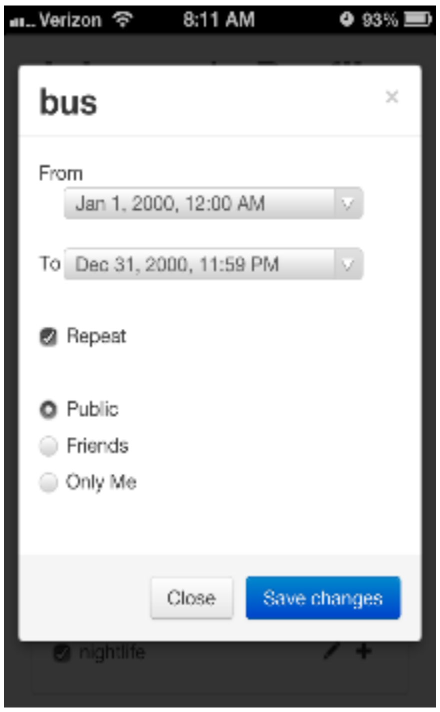


Profile Setting

Jingo has very powerful ways for users to set up their own filters by setting users' profile. Profile setting is very important because it can help users decide what kinds of notes they are interested in receiving and reading. When users sign up as a member of Jingo app, their profile will have an default state named "my state" including eleven default filters as shown in the figure of right hand side. If users want to specify their own filters, they need to press the **Edit button** to change current mode to editing mode. the **Plus button** provide users with the ability to create their own tags in a convenient way as shown in the figure of right hand side.

Extra Credits Here

Creating custom tags is provided when users edit their profile, which means users can set up their own tags in advance for the purpose of posting notes. Another way to do this is also available for users when they are posting notes.



From Jan 1, 2000, 12:00 AM

To Dec 31, 2000, 11:59 PM

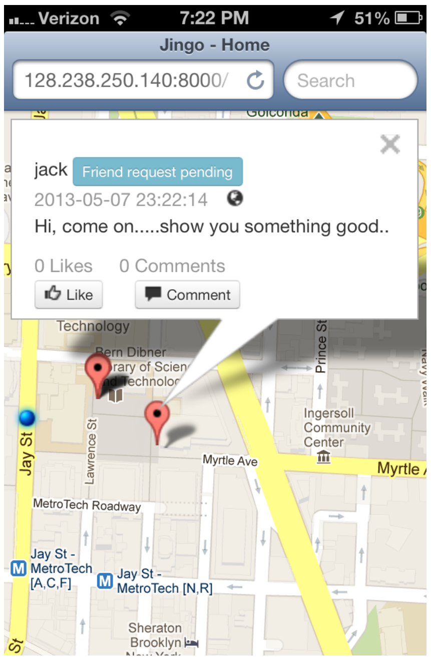
✓ Repeat

[✓ Public] Friends Only Me

Filter Setting

Here is how users can edit a filter in detail. Jingo give users a simple way to specify when they want to or do not want to receive notes and who can read their notes. Setting schedule for a filter is easy for users because they only need to tell Jingo a period of time when user want to receive notes. The rest of things are just left to Jingo to finish for you. Also, users are able to set up a level of visibility for their notes. There are three options they can choose to use: "Public" is everyone can read, "Friends" is only your friends can read, and "Only me" is only you can read.

ii Read Notes

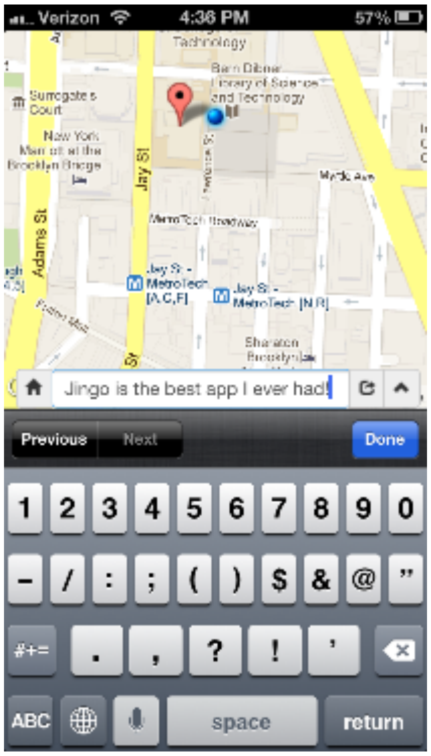


Read Notes
Jingo provides users with an intuitive way to read notes on Google Maps. Once users get into Jingo home page, Jingo will mark red spots within user's current location automatically. In the meantime, what users need to do is tapping one of red spots they are interested in so that Jingo will pop out an info window showing the content of the note. Of course, users can press like button and leave a comment to posters at this time. Also, users are able to follow this poster, which means users can make friends with the poster by pressing the follow button. After that, the info window will show "Friend request pending" meaning the user need to wait for the reply from the poster.

Extra Credits Here
Like button makes users vote which note is most popular or useful for others. In addition, the number of like will be used for filtering notes for users automatically.

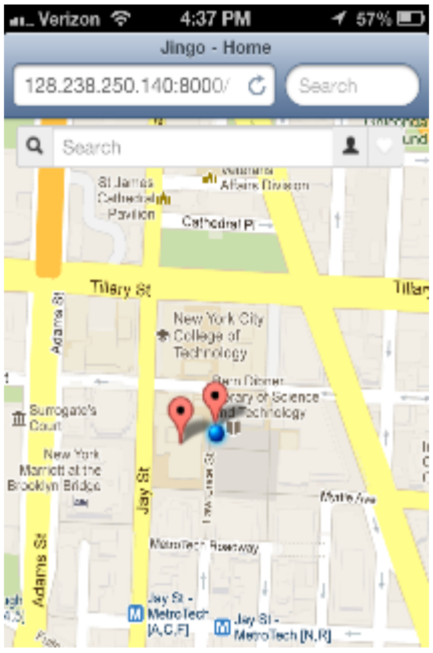
iii Post Notes

Quick mode for posting notes



Quick mode for posting notes
Jingo provides users with an quick way to post notes on Google Maps. There's a textarea with placeholder "What's on your mind?". Users can quickly input note in this textarea, and click the publish button to post the note.

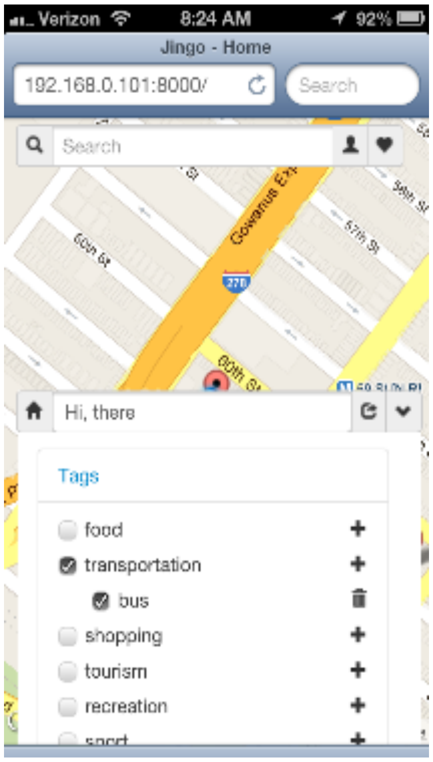
Ching-Che Chang 0435353
Jiankai Dang 0482923



The result of posting notes

Jingo creates new marker for the newly posted note instantly after user post it. There's a DROP animation effect to show the marker, which makes the newly posted note and the marker noticeable to user and let user aware of the geolocation of that note.

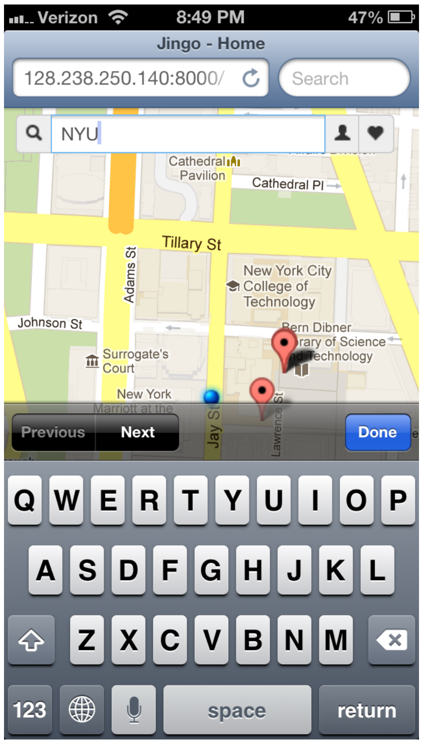
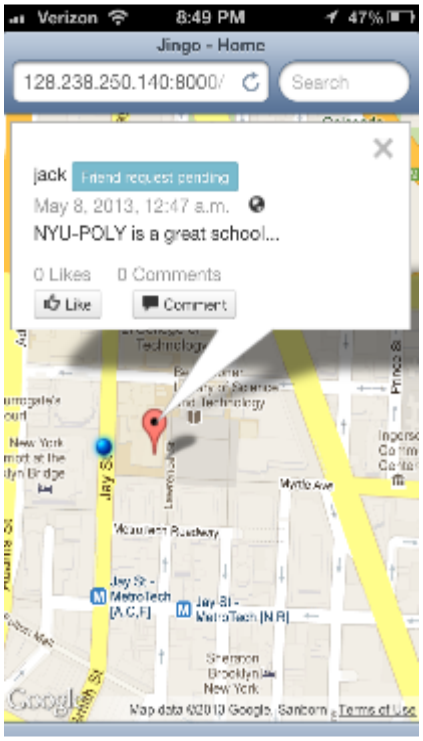
Advanced mode for posting notes



Advanced mode for posting note

Jingo provides users with an advanced way to post notes on Google Maps. After click the upward arrow, the advanced mode will be shown in the panel. Users could input "Link", "Radius", "Tags", "Schedule", "Repeat", "Allow comments", and "Privacy" fields in this panel. For the "Radius" field, the default radius is 200 yards. For the "Tags", field, users could expand and collapse the tags panel. Inside "Tags" panel, users could check any tag that is related to the note and create customized tags, which will be added into users' profile. For the "Schedule" field, users could set a time period in which other users could receive this note. For the "Repeat" field, users could set the time schedule to be repeated by checking the checkbox of "Repeat". For the "Allow comment" field, users could check the checkbox to allow other users to comment on this note. Users could also set the "Privacy" field to "Public", "Friends", or "Only Me".

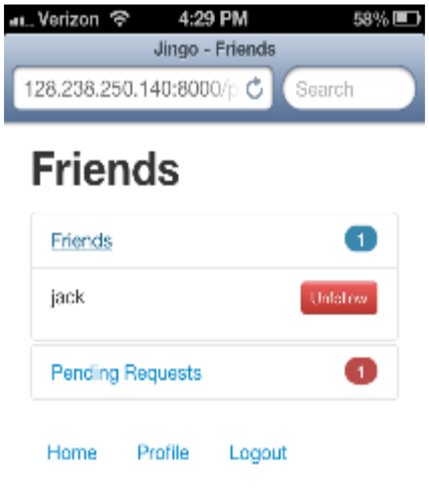
iv Search Notes

	**Search notes** Jingo provides users with an easy way to search notes on Google Maps. Users could input keywords inside the "Search" box. After click the "Search" button, Jingo will search all the notes available and return the result notes, which include the keywords in the notes content or match the tag names of the notes.
	**Search result** Jingo will create new markers for the result notes. Users could tap a specific marker to read the detail information of the note. Jingo will perform a BOUNCE animation effect for the marker which was clicked by users. In the info window, users could get all the detailed information about the note.

Verizon 8:49 PM 47%
Jingo - Home
128.238.250.140:8000/
NYU

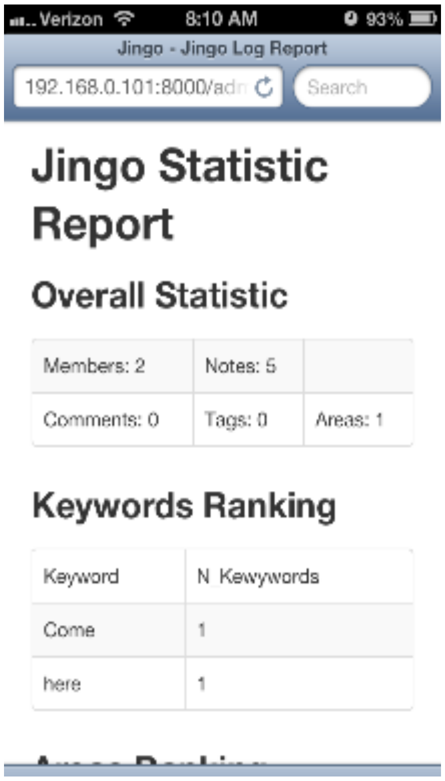
Verizon 8:49 PM 47%
Jingo - Home
128.238.250.140:8000/
jack Friend request pending
May 8, 2013, 12:47 a.m.
NYU-POLY is a great school...
0 Likes 0 Comments

V Friendship



Friends
Jingo provides a brand new page for users to manage friendship. There are two types of lists in this page: “Friends” and “Pending Requests”. In “Friend” list, users could “Unfollow” a friend. In “Pending Requests” list, users could “Accept” or “Deny” a friend request.

C. Statistic Report



Jingo Statistic Report

Overall Statistic

Members: 2	Notes: 5	
Comments: 0	Tags: 0	Areas: 1

Keywords Ranking

Keyword	N_Kewywords
Come	1
here	1

Jingo Statistic Report
Jingo do statistic at any time when users are doing something with Jingo app. In the admin page of Jingo app, several informative statistic will show up as follows:
Overall Statistic:
Tell you how many members sign up for Jingo, how many notes are posted on Jingo, how many comments users reply to posters, etc.
Keywords Ranking:
Help you find out which keyword is most frequently used for searching notes.
Area Ranking:
Show you which area is most popular among the members of Jingo and how many notes are here.
Notes Ranking:
Disclose which note is the most popular to be read by members of Jingo.
Poster Ranking:
Find out who is the poster with the highest number of posting notes.
Tags Ranking:
Present which tag is most frequently used by users.

Ching-Che Chang 0435353
Jiankai Dang 0482923

IV. Conclusion

The main idea in Jingo is that users can publish information (small messages or notes), and then link these notes to certain locations and certain times. Other users can then receive these notes based on their own location, the current time, and based on what type of messages they want to receive.

This document describes the basic idea behind the system. In this first part of the course project, we design the relational database that stores all the information about users, friendships between users, notes published by users, and filters that users have for what kind of notes they want to receive at different times and in different situations (i.e., states and locations).

In the second project, we have created a web-based user interface for the Jingo database designed in the first project. In particular, users are able to register, create a profile, log in, post notes, and comment on notes they have received. Users are able to define filters for receiving notes and to send and answer friend requests.

Users are able to perform all operations via a standard web browser. This is implemented by writing a program that is called by a web server, connects to our database, then calls appropriate stored procedures and queries that we have defined in the database, and finally returns the results as a web page. We use frameworks such as Python and Django to connect to our backend database.