Zixian Lai

CS6083

November 15, 2018

# Project 1

due November 27

## Introduction

The paper introduces the database design for the new mobile app oingo. The main idea in oingo is that users can publish information in the form of short notes, and then link these notes to certain locations and certain times. Other users can then receive these notes based ob their own location, the current time, and based on what type of messages they want to receive. The database stores various data sets to support this idea such as the location of each user, the current time for each user, a schedule when the note can be seen, etc. The paper explain the database design in detail with some test cases.

# ER diagram

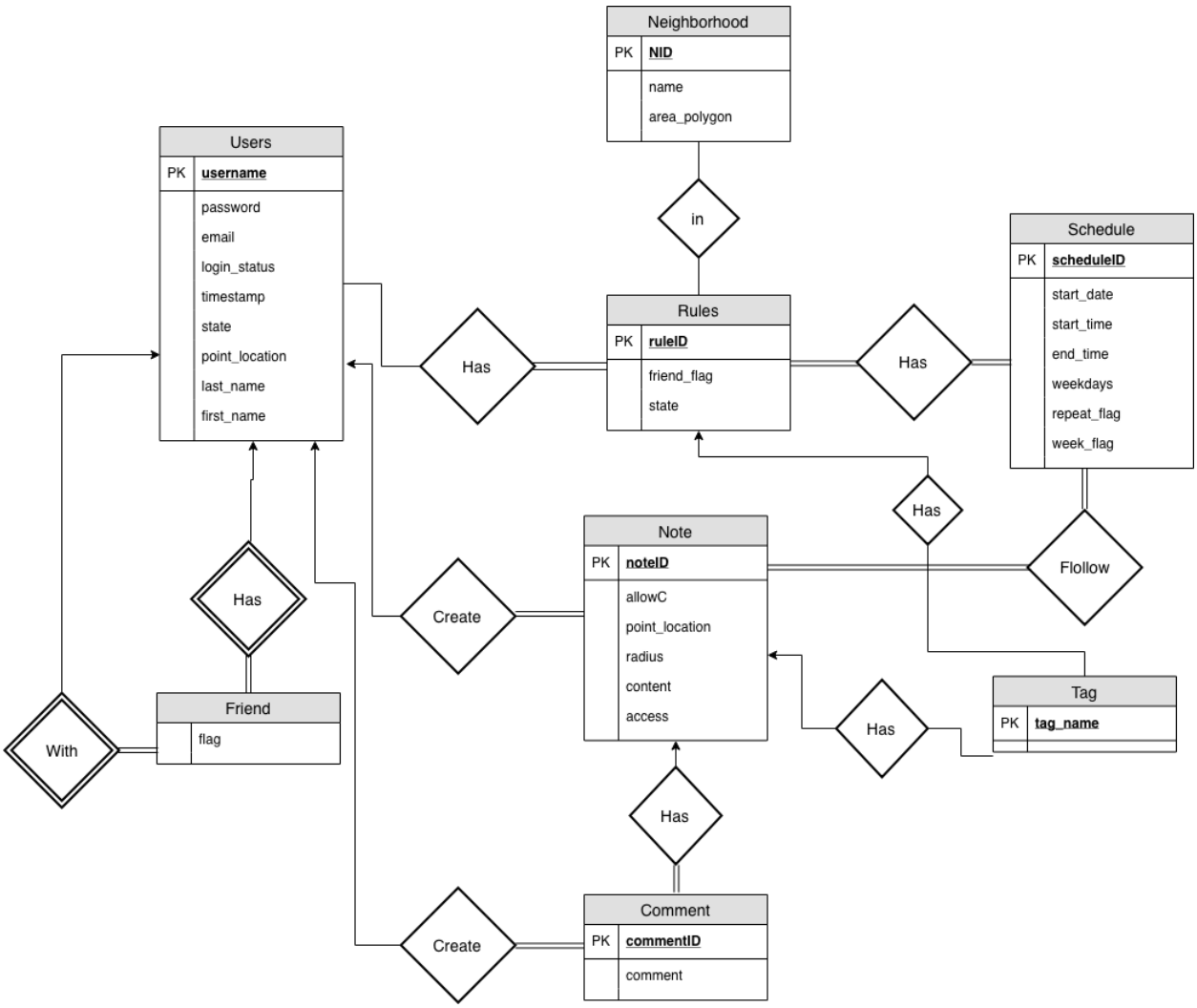



Figure 1

## Relation Schema

**Users(**username, last_name, first_name, password, email, login_status, timestamp, point_location, state)

**Rules**(ruleID, username, tag_name, scheduleID, NID, friend_flag, state)

**Schedule**(scheduleID, start__time, end_time, start_date, repeat_flag, weekdays, week_flag)

**Friend**(user, friend, flag)

**Note**(username, noteID, scheduleID, point_location, radius, content, allowC, access)

**Has_tag**(noteID, tag_name)

**Tag**(tag_name)

**Comments**(noteID, commentID, username, comment)

**Neighborhood**(NID, name, area_polygon)

## Constraints

In the Users relation, username is the primary key.

In the Rules relation, ruleID is the primary key. username is a foreign key referencing Users; _name is a foreign key referencing Tag; scheduleID is a foreign key referencing Schedule; NID is a foreign key referencing Neighborhood.

In the Schedule relation, scheduleID is the primary key.

In the Friend relation, user is a foreign key referencing Users; friend is a foreign key referencing Users.

In the Note relation, noteID is the primary key. Username is a foreign key referencing Users; scheduleID is a foreign key referencing Schedule.

In the Has_tag relation, noteID is a foreign key referencing Note; tag_name is a foreign key referencing Tag.

In the Tag relation, tag_name is the primary key.

In the Comments relation, commentID is the primary key; noteID is a foreign key referencing Note; username is a foreign key referencing Users.

In the Neighborhood relation, NID is the primary key.

## Description

In the Users relation, login_status is a 1/0 value which indicates the login_status of the user. The timestamp and point_location stores the location of the user at that time(should be updated every 5 mins or whenever the user performs any action.

In the Rules relation, friend_flag is a 1/2/3 value which indicates that if the user wants to receive notes from everyone or just friends or just himself/herself. The NID point to the Neighborhood relation to indicate that users will receive notes when at that neighborhood. The scheduleID point to the Schedule relation to indicate at what time the user wants to receive notes. The tag_name points Tag relation to indicate notes users want to receive with that tag. The state indicates at what states users want to receive notes.

In the Schedule relation, the start_data and start_time indicate the starting time and the date of the schedule; the end_time indicates the ending time of the schedule; the time_range indicates the duration of the schedule; repeat_flag indicates if the schedule repeats daily(1), monthly(2), yearly(3) or not at all(0); the week_flag is a 1/0 value which indicates if the schedule repeats weekly or not and the weekdays stores weekday(1..7) that will repeat every week.

In the friend relation, flag is a 1/0 value which indicates if the user and friend are friends.

In the note relation, the point_location and the radius indicate the area that the note can be seen; allowC is a 1/0 value indicates if the note allows comments. The username indicates the user who created the note; the scheduleID indicates the schedule that the note can be seen. Access is a 1/2/3 value which indicates that if the user wants to show the note to everyone or just friends or just himself/herself.

The Has_tag relation associates one note with one or more tags.

The Tag relation stores the predefined tag name.

The Comments relation associates one note with one or more comments.

The Neighborhood relation stores predefined neighborhood name and their area as polygon.

# Queries

(1)

```
insert into Users(username, last_name, first_name, pass_word, email) values
    ('username2', 'Hubbard', 'Milton', 'password2', 'MiltonHubbard@email.com');
```

(2)

```
# creating testing note
insert into Schedules(start_time, end_time, start_date, repeat_flag, weekdays, week_flag) values
    ('17:00:00', '19:00:00', '2018-11-27', 0, 0, false);
insert into Note(username, scheduleID, point_location, radius, content, allowC, access) values
    ('username1', (select last_insert_id()), point(40.755756, -73.977648), 1000,
    'note show on 2018-11-27 5pm-7pm; does not repeat; in Midtown with radius 1000; allow comment; everyone can see(1)',
     true, 1);
insert into Has_tag(noteID, tag_name) values
    ((select last_insert_id()), 'shopping'),
    ((select last_insert_id()), 'friends'),
    ((select last_insert_id()), 'me'),
    ((select last_insert_id()), 'food');
```

(3)

```
# (3) show friend list
select T.friend from(select case when username = 'username2' then friend
                                 when friend = 'username2' then username
                                 end as `friend`
                     from Friend
                     where flag = true) T
where T.friend is not NULL;
```

(4)

```
# question 4
set @test_user = 'username1';
set @user_location = (select point_location from Users where username = @test_user);
set @user_time = (select time(time_stamp) from Users where username = @test_user);
set @user_date = (select date(time_stamp) from Users where username = @test_user);
set @user_state = (select state from Users where username = @test_user);
set @user_timestamp = (select time_stamp from Users where username = @test_user);

drop temporary table if exists filtered_tags;

create temporary table filtered_tags    #tempory table for user which contains all the tags and people user want to see base on filter and state
select tag_name, friend_flag from Rules R left join Neighborhood N on R.NID = N.NID
where username = @test_user and (st_contains(area_polygon, @user_location) or (R.NID is null)) and
check_s(R.scheduleID, @user_timestamp) and (R.state = @user_state or R.state is NULL)
;

select N.noteID, content from Note N left join Has_tag H on N.noteID = H.noteID
where (st_distance_sphere(@user_location, point_location) < radius or point_location is NULL or radius is NULL) and # checking the user's location with notes
       check_s(scheduleID, @user_timestamp) and check_access(access, N.username, @test_user)
              and (exists (       # checking with the user's filter's tag
                          select 1 from filtered_tags FT
                          where (FT.tag_name is null and check_access(FT.friend_flag, N.username, @test_user)) or
                          (FT.tag_name = H.tag_name and check_access(FT.friend_flag, N.username, @test_user))
              ));


# function for checking schedule and access
create function check_s(sID INT, c_time datetime)
returns boolean deterministic
return (select (exists
       (select 1 from Schedules
       where ScheduleID = sID and
       ((time(c_time) between start_time and end_time) or start_time is null or end_time is null) and
       (repeat_flag = 1 or (repeat_flag = 2 and day(date(c_time)) = day(start_date))
              or (repeat_flag = 3 and day(date(c_time)) = day(start_date) and month(date(c_time)) = month(start_date))
              or (repeat_flag = 0 and date(c_time) = start_date)
              or (week_flag = true and weekdays = dayofweek(date(c_time))))
       )));

drop function if exists check_access;
create function check_access(access INT, user1 varchar(50), user_me varchar(50))
returns boolean deterministic
return ((access = 1 or (access = 2 and (
       exists (select 1 from Friend F
       where (F.username = user1 and F.friend = user_me and F.flag = true) or (F.username = user_me and F.friend = user1 and F.flag = true) )
                                          ))
              or (access = 3 and user1 = user_me)));
```

(5)

```
drop view if exists user_location;
create view user_location as select username, point_location, state from Users;
select * from user_location;
set @testNote = 2;
set @testNote_point = (select point_location from Note where noteID = @testNote);
set @testNote_radius = (select radius from Note where noteID = @testNote);
set @testNote_owner = (select username from Note where noteID = @testNote);
set @testNote_access = (select access from Note where noteID = @testNote);
set @testNote_schedule = (select scheduleID from Note where noteID = @testNote);
set @curr_time = '2018-11-28 04:00:00';


select R.username from Rules R left join Neighborhood N on R.NID = N.NID left join user_location U on U.username = R.username
where (st_contains(area_polygon, point_location) or (R.NID is null)) and
check_s(R.`scheduleID`, @curr_time) and (R.state = U.state or R.state is NULL) and check_s(@testNote_schedule, @curr_time) and
               (st_distance_sphere(point_location, @testNote_point) < @testNote_radius or @testNote_point is NULL or @testNote_radius is NULL) and
               check_access(R.friend_flag, @testNote_owner, R.username) and check_access(@testNote_access, @testNote_owner, R.username) and
               (R.tag_name is NULL or exists(select 1 from Has_tag H where H.`noteID` = @testNote and H.tag_name = R.tag_name))
;
# function for checking schedule and access
drop function if exists check_s
create function check_s(sID INT, c_time datetime)
returns boolean deterministic
return (select (exists
       (select 1 from Schedules
       where ScheduleID = sID and
       ((time(c_time) between start_time and end_time) or start_time is null or end_time is null) and
       (repeat_flag = 1 or (repeat_flag = 2 and day(date(c_time)) = day(start_date))
              or (repeat_flag = 3 and day(date(c_time)) = day(start_date) and month(date(c_time)) = month(start_date))
              or (repeat_flag = 0 and date(c_time) = start_date)
              or (week_flag = true and weekdays = dayofweek(date(c_time))))
       )));

drop function if exists check_access;
create function check_access(access INT, user1 varchar(50), user_me varchar(50))
returns boolean deterministic
return ((access = 1 or (access = 2 and (
       exists (select 1 from Friend F
       where (F.username = user1 and F.friend = user_me and F.flag = true) or (F.username = user_me and F.friend = user1 and F.flag = true) )
                                          ))
              or (access = 3 and user1 = user_me)));
```

(6)

```
#question 6
set @test_user = 'username3';
set @user_location = (select point_location from Users where username = @test_user);
set @user_time = (select time(time_stamp) from Users where username = @test_user);
set @user_date = (select date(time_stamp) from Users where username = @test_user);
set @user_state = (select state from Users where username = @test_user);
set @user_timestamp = (select time_stamp from Users where username = @test_user);

drop temporary table if exists filtered_tags;

create temporary table filtered_tags    #tempory table for user which contains all the tags and people user want to see base on filter and state
select tag_name, friend_flag from Rules R left join Neighborhood N on R.NID = N.NID
where username = @test_user and (st_contains(area_polygon, @user_location) or (R.NID is null)) and
check_s(R.scheduleID, @user_timestamp) and (R.state = @user_state or R.state is NULL)
;

create temporary table all_notes
select N.noteID, content from Note N left join Has_tag H on N.noteID = H.noteID
where (st_distance_sphere(@user_location, point_location) < radius or point_location is NULL or radius is NULL) and # checking the user's location with notes
        check_s(scheduleID, @user_timestamp) and check_access(access, N.username, @test_user)
                and (exists (       # checking with the user's filter's tag
                            select 1 from filtered_tags FT
                            where (FT.tag_name is null and check_access(FT.friend_flag, N.username, @test_user)) or
                            (FT.tag_name = H.tag_name and check_access(FT.friend_flag, N.username, @test_user))
                ));
select * from all_notes;

set @keywords = '23 year';
set @pattern = (select replace(@keywords, ' ', '.+'));
select * from all_notes where content REGEXP @pattern;
```

## Sample data

• In order to test the date base, two predefined Neighborhoods were inserted into the **Neighborhood table**. The area_polygon is a geometric polygon with latitudes and longitudes.

| NID | N_name | area_polygon |
|---|---|---|
| 2 | Midtown | POLYGON((40.768175 -73.98248700000001,40.762319 -73.968191,40.7… |
| 3 | Turtle Bay | POLYGON((40.758338 -73.97099,40.75419 -73.96108,40.747862 -73.966… |

• And four predefined tags were inserted into the **Tag table**.

| tag_name |
|---|
| food |
| friends |
| nature |
| shopping |

• To test query (1), three users were inserted into the **Users table.** username1 and username2's last recorded locations are in Midtown and username3's last recorded location is in Turtle Bay. point_location is a geometric point with latitude and longitude.

| username | last_name | first_name | pass_word | email | login_status | time_stamp | point_location | state |
|---|---|---|---|---|---|---|---|---|
| username1 | Harris | Rocky | password1 | RockyHarris@email.com | 1 | 2018-11-28 03:03:12 | POINT(40.759683 -73.97807899999999) | working |
| username2 | Hubbard | Milton | password2 | MiltonHubbard@email.com | 0 | 2018-11-28 05:03:12 | POINT(40.759258 -73.973899) | hungry |
| username3 | Jonny | Viola | password3 | ViolaJonny@email.com | 0 | 2018-11-28 18:00:00 | POINT(40.750937 -73.96815700000001) | idle |

• In order to test the note and filters, five different schedules were inserted into the **Schedules table**. repeat_flag tells if the schedule repeats. repeat_flag = 1 means that the schedule repeats every day. repeat_flag = 2 means that the schedule repeats every month. repeat_flag = 3 means that the schedule repeats every year. repeat_flag = 0 means that the schedule does not repeat at all. week_flag tells if the schedule is set for week_days. In the Schedules table, Schedule 1 represents a schedule that starts at 00:00:00 and ends at 23:59:59 and repeats every day; Schedule 2 represents a schedule that starts at 00:00:00 and ends at 23:00:00 and repeats every month at day 28; Schedule 3 represents a schedule that starts at 00:00:00 and ends at 20:00:00 and it repeats every year at 11/28; Schedule 4 represents a schedule that starts at 00:00:00 and ends at 18:00:00 and it does not repeat at all; Schedule 5

represents a schedule that starts at 00:00:00 and ends at 06:00:00 and it repeats every Wednesday (Sunday is the first day of the week).

| scheduleID | start_time | end_time | start_date | repeat_flag | weekdays | week_flag |
|---|---|---|---|---|---|---|
| 1 | 00:00:00 | 23:59:59 | 2018-11-28 | 1 | 0 | 0 |
| 2 | 00:00:00 | 23:00:00 | 2018-12-28 | 2 | 0 | 0 |
| 3 | 00:00:00 | 20:00:00 | 2019-11-28 | 3 | 0 | 0 |
| 4 | 00:00:00 | 18:00:00 | 2018-11-28 | 0 | 0 | 0 |
| 5 | 00:00:00 | 06:00:00 | NULL | NULL | 4 | 1 |

- In order to test note, 5 notes were inserted into the **Note table.**

| noteID | username | scheduleID | point_location | radius | content | allowC | access |
|---|---|---|---|---|---|---|---|
| 2 | username2 | 1 | POINT(40.755756 -73.977648) | 1000 | note show on 2018-11-27 5pm-7pm; does not repeat; in Midtown with radius 1000; allow comment; everyone can see(1) | 1 | 1 |
| 3 | username1 | 2 | POINT(40.761518 -73.972707) | 4000 | note show 0-23; repeat every month; in midtown with radius 4000, does not allow comment; only friend can see(2) | 0 | 2 |
| 4 | username3 | 3 | POINT(40.75094 -73.968153) | 2000 | note show 0-20; repeat every year; in turtle bay with radius 2000; allows comment; only owner can see(3) | 1 | 3 |
| 5 | username1 | 4 | POINT(40.750936 -73.96815599999999) | 10 | note show 0-18; does not repeat; in turtle bay with radius 10; allow comment; everyone can see(1) | 1 | 1 |
| 6 | username2 | 5 | POINT(40.761518 -73.972707) | 4000 | note show 0-6; repeat every wendsday; in midtown with radius 4000; allow comment; only friend can see(2) | 1 | 2 |

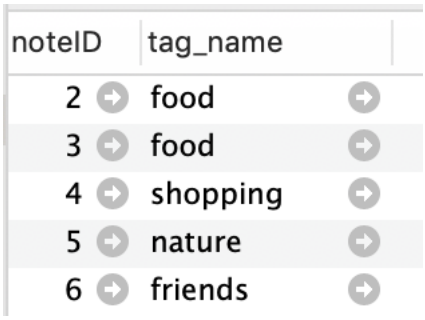
- Three friendship relations were inserted into the **Friend table**. Username1 and username2 are friends; username1 and username3 are friends; username2 and username3 are not friends.

| username | friend | flag |
|---|---|---|
| username1 | username2 | 1 |
| username1 | username3 | 1 |
| username2 | username3 | 0 |

- Three filters were inserted into the **Rules table**. username1 has one rule which follows schedule with scheduleID = 5, the rule filters notes from friends(friend_flag = 2) and with tag = friends in midtown(NID = 2). username2 has one rule which follows schedule with scheduleID = 1, the rule filters notes from everyone(friend_flag = 1) and with tag = food in midtown(NID = 2). username3 has one partial rule which follows schedule with scheduleID = 1, the rule receives note from every without specify the tag of the note, the state of the user, and the neighborhood the user is in.

| ruleID | username | tag_name | scheduleID | NID | friend_flag | state |
|---|---|---|---|---|---|---|
| 8 | username2 | food | 1 | 2 | 1 | hungry |
| 9 | username1 | friends | 5 | 2 | 2 | NULL |
| 10 | username3 | NULL | 1 | NULL | 1 | NULL |

- Five relation were inserted into the **Has_tag table**, one for each note.



| noteID | tag_name |
|---|---|
| 2 | food |
| 3 | food |
| 4 | shopping |
| 5 | nature |
| 6 | friends |

## Testing queries

- **Testing query (1) for inserting new testing user without setting the location, timestamp, login_status, and state.**

```
insert into Users(username, last_name, first_name, pass_word, email) values
    ('test_user', 'Test', 'User', 'test_pass', 'TestUser@email.com');
```

Result of Users table

| username | last_name | first_name | pass_word | email | login_status | time_stamp | point_location | state |
|---|---|---|---|---|---|---|---|---|
| test_user | Test | User | test_pass | TestUser@email.com | NULL | NULL | NULL | NULL |
| username1 | Harris | Rocky | password1 | RockyHarris@email.com | 1 | 2018-11-28 03:03:12 | POINT(40.759683 -73.97807899999999) | working |
| username2 | Hubbard | Milton | password2 | MiltonHubbard@email.com | 0 | 2018-11-28 05:03:12 | POINT(40.759258 -73.973899) | hungry |
| username3 | Jonny | Viola | password3 | ViolaJonny@email.com | 0 | 2018-11-28 18:00:00 | POINT(40.750937 -73.96815700000001) | idle |

- **Testing query (2) for inserting new testing note with tag.**

```
insert into Note(username, scheduleID, point_location, radius, content, allowC, access) values
    ('test_user', 1, point(40.755756, -73.977648), 1000,
    'note this it the testing note',
     true, 1);
insert into Has_tag(noteID, tag_name) values
    ((select last_insert_id()), 'shopping');
```

Result of Note table and Has_tag table

| noteID | username | scheduleID | point_location | radius | content | allowC | access |
|---|---|---|---|---|---|---|---|
| 2 | username2 | 1 | POINT(40.755756 -73.977648) | 1000 | note show on 2018-11-27 5pm-7pm; does not repeat; in Midtown with radius 1000; allow comment; everyone can see(1) | 1 | 1 |
| 3 | username1 | 2 | POINT(40.761518 -73.972707) | 4000 | note show 0-23; repeat every month; in midtown with radius 4000, does not allow comment; only friend can see(2) | 0 | 2 |
| 4 | username3 | 3 | POINT(40.75094 -73.968153) | 2000 | note show 0-20; repeat every year; in turtle bay with radius 2000; allows comment; only owner can see(3) | 1 | 3 |
| 5 | username1 | 4 | POINT(40.750936 -73.96815599999999) | 10 | note show 0-18; does not repeat; in turtle bay with radius 10; allow comment; everyone can see(1) | 1 | 1 |
| 6 | username2 | 5 | POINT(40.761518 -73.972707) | 4000 | note show 0-6; repeat every wendsday; in midtown with radius 4000; allow comment; only friend can see(2) | 1 | 2 |
| 10 | test_user | 1 | POINT(40.755756 -73.977648) | 1000 | note this it the testing note | 1 | 1 |

| noteID | tag_name |
|---|---|
| 2 | food |
| 3 | food |
| 4 | shopping |
| 5 | nature |
| 6 | friends |
| 10 | shopping |

- **Testing query (3) for finding all friends of a user.**

Friend table

| username | friend | flag |
|---|---|---|
| username1 | username2 | 1 |
| username1 | username3 | 1 |
| username2 | username3 | 0 |

Finding all friends for user username1

```
select T.friend from(select case when username = 'username1' then friend
                                 when friend = 'username1' then username
                                 end as `friend`
                     from Friend
                     where flag = true) T
where T.friend is not NULL;
```

Result:

| friend |
|---|
| username2 |
| username3 |

- **Testing query (4) for outputting all the notes a user should see base on current time, current location, current state.**

Details of Schedules table, Rules table, User table, Friends table, Note table, Neighborhood table, and Has_tag table used for this test can be found in the sample data section.

Result of running query (4) on username1:

| noteID | content |
|---|---|
| 6 | note show 0-6; repeat every wendsday; in midtown with radius 4000; allow comment; only friend can see(2) |

| noteID | Can see | Reason |
|---|---|---|
| 2 | No | Tag is different. |
| 3 | No | Tag is different. |
| 4 | No | Note 4 can only be seen by the owner. |

| noteID | Can see | Reason |
|---|---|---|
| 5 | No | Tag is different. |
| 6 | Yes | username1's current time, current location and current state satisfy all the conditions. And he is a friend of username2 who owns Note 6. And Note6's tag satisfy the rule's tag. |

Result of running query (4) on username2:

| noteID | content |
|---|---|
| 2 | note show on 2018-11-27 5pm-7pm; does not repeat; in Midtown with radius 1000; allow comment; everyone can see(1) |
| 3 | note show 0-23; repeat every month; in midtown with radius 4000, does not allow comment; only friend can see(2) |

| noteID | Can see | Reason |
|---|---|---|
| 2 | Yes | username1's current time, current location and current state satisfy all the conditions. And he is a friend of username2 who owns Note 6. And Note6's tag satisfy the rule's tag. |
| 3 | Yes | username1's current time, current location and current state satisfy all the conditions. And he is a friend of username2 who owns Note 6. And Note6's tag satisfy the rule's tag. |
| 4 | No | Tag is different and Note 4 can only be seen by the owner. |
| 5 | No | Tag is different. |
| 6 | No | Tag is different. |

Result of running query (4) on username3:

| noteID | content |
|---|---|
| 3 | note show 0-23; repeat every month; in midtown with radius 4000, does not allow comment; only friend can see(2) |
| 4 | note show 0-20; repeat every year; in turtle bay with radius 2000; allows comment; only owner can see(3) |
| 5 | note show 0-18; does not repeat; in turtle bay with radius 10; allow comment; everyone can see(1) |

| noteID | Can see | Reason |
|---|---|---|
| 2 | No | Username3's current location is not in the effective range of Note 2. |
| 3 | Yes | The username3's rule only has a schedule, it will receive all notes which satisfy the time constrain. And note 5 satisfies the time constrain. |
| 4 | Yes | The username3's rule only has a schedule, it will receive all notes which satisfy the time constrain. And note 5 satisfies the time constrain. |
| 5 | Yes | The username3's rule only has a schedule, it will receive all notes which satisfy the time constrain. And note 5 satisfies the time constrain. |
| 6 | No | Note 6 can only be seen by owner's(username2) friend. And username2 and username3 are not friends. |

- **Testing query (5) for outputting all users who can see the given note.**

Result for running query (5) on note 2

| username |
| --- |
| username2 |

Result for running query (5) on note 3

| username |
| --- |
| username2 |
| username3 |

Result for running query (5) on note 4

| username |
| --- |
| username3 |

Result for running query (5) on note 5

| username |
| --- |
| username3 |

Result for running query (5) on note 6

```
drop view if exists user_location;
create view user_location as select username, point_location, state from Users;
select * from user_location;
set @testNote = 6;
set @testNote_point = (select point_location from Note where noteID = @testNote);
set @testNote_radius = (select radius from Note where noteID = @testNote);
set @testNote_owner = (select username from Note where noteID = @testNote);
set @testNote_access = (select access from Note where noteID = @testNote);
set @testNote_schedule = (select scheduleID from Note where noteID = @testNote);
set @curr_time = '2018-11-28 04:00:00';


select R.username from Rules R left join Neighborhood N on R.NID = N.NID left join user_location U on U.username = R.username
where (st_contains(area_polygon, point_location) or (R.NID is null)) and
check_s(R.`scheduleID`, @curr_time) and (R.state = U.state or R.state is NULL) and check_s(@testNote_schedule, @curr_time) and
                (st_distance_sphere(point_location, @testNote_point) < @testNote_radius or @testNote_point is NULL or @testNote_radius is NULL) and
                check_access(R.friend_flag, @testNote_owner, R.username) and check_access(@testNote_access, @testNote_owner, R.username) and
                (R.tag_name is NULL or exists(select 1 from Has_tag H where H.`noteID` = @testNote and H.tag_name = R.tag_name))
;
```

Base on the results from running query (4), query (5) returns the correct users for each note.

- **Testing query (6) for filtering notes by key words.**

All the notes username3 can see

| noteID | content |
|---|---|
| 3 | note show 0-23; repeat every month; in midtown with radius 4000, does not allow comment; only friend can see(2) |
| 4 | note show 0-20; repeat every year; in turtle bay with radius 2000; allows comment; only owner can see(3) |
| 5 | note show 0-18; does not repeat; in turtle bay with radius 10; allow comment; everyone can see(1) |

Result after running query (6) with key words 'turtle comment'

| noteID | content |
|---|---|
| 4 | note show 0-20; repeat every year; in turtle bay with radius 2000; allows comment; only owner can see(3) |
| 5 | note show 0-18; does not repeat; in turtle bay with radius 10; allow comment; everyone can see(1) |

Result after running query (6) with key words 'allows (1)'

| noteID | content |
|---|---|
| 4 | note show 0-20; repeat every year; in turtle bay with radius 2000; allows comment; only owner can see(3) |

Result after running query (6) with key words 'year month'

| noteID | content |
|---|---|

Result after running query (6) with key words 'month'

| noteID | content |
|---|---|
| 3 | note show 0-23; repeat every month; in midtown with radius 4000, does not allow comment; only friend can see(2) |

Result after running query (6) with key words 'Midtown comment'

| noteID | content |
|---|---|
| 3 | note show 0-23; repeat every month; in midtown with radius 4000, does not allow comment; only friend can see(2) |

Result after running query(6) with key words 'NOTE'

| noteID | content |
|---|---|
| 3 | note show 0-23; repeat every month; in midtown with radius 4000, does not allow comment; only friend can see(2) |
| 4 | note show 0-20; repeat every year; in turtle bay with radius 2000; allows comment; only owner can see(3) |
| 5 | note show 0-18; does not repeat; in turtle bay with radius 10; allow comment; everyone can see(1) |