THE DOORBELL CO.

Principles of Database Systems (CS-GY 6083)

Project (Group 24)

Gaurav Agrawal (ga1380)

Ronak Fofaliya (rf1999)

Table of Contents

Project Part -1 Introduction

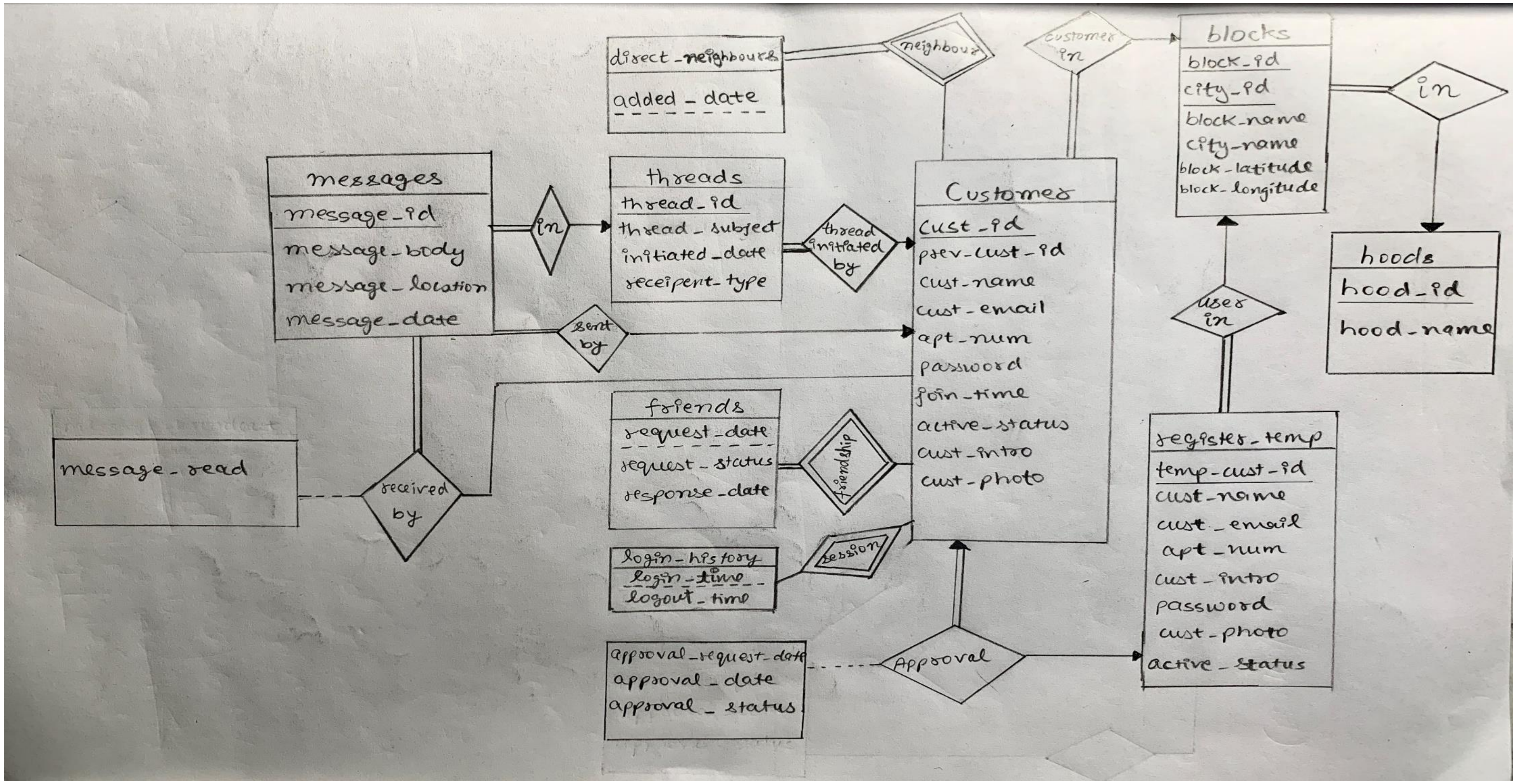
This project deals with building a social network for neighbourboods where people get to know people who live nearby. This website would allow people to communicate and share things with people in their neighbourhood. In particular, people would be able to sign up for the service and specify where they live; they can then send and receive messages to other users living close by, and participate in discussions with those users. The first part of this project deals with designing & developing the relation database and also envision and layout a plan for the second part, which would be designing the web interface.

We would be using MySQL/MariaDB database for the relational schema and HTML, CSS, BootStrap, PHP and Javascript for designing the web interface.

Key Features of the website

- Users can register for the service by applying to join a block on the signup page. While signing up, the users can introduce themselves, their family and optionally upload their photographs. The joining request would go to all the members of the block in which user has applied along with user's intro. This request have to be approved by atleast 3 existing members of the same block (or all the members if there are less than 3).
- While registering, user can either select the block by clibking on the map pointer or by selecting value in the block field dropdown.
- Once the user is approved, he/she would access to the content under his/her block, hood based on factors like his/friends, neighbours, threads etc. The user can also perform various activities.
- A user can edit his/her profile after being approved.
- A user can ask another user from the same hood to become his/her friend. The friend request has to be approved by the other user or vice-versa.
- A user can specify his/her direct neighbours. Only people living very close by can be added as direct neighbours. We are considering a block as to be the scope for very close by.
- A user can initiate a thread by choosing a subject, the content and the recipients. There are many options for choosing a recipient. The recipient can either be a friend, a direct neighbour, all the friends, the complete block or the complete hood.
- A user can reply to message which are directed towards him/her. It can be a direct message, or a message where is the user is part of the recipient scope.
- In case the user decides to leave the block and wants to reapply the for a new a new block, the user can use the **address update feature**.

E-R Diagram


direct_neighbours
added_date
neighbour
customer in
blocks
block_id
city_id
block_name
city_name
block_latitude
block_longitude
in
hoods
hood_id
hood_name
messages
message_id
message_body
message_location
message_date
in
threads
thread_id
thread_subject
initiated_date
receipent_type
thread initiated by
Customer
cust_id
prev_cust_id
cust_name
cust_email
apt_num
password
join_time
active_status
cust_intro
cust_photo
user in
sent by
message_read
received by
friends
request_date
request_status
response_date
friendship
session
login_history
login_time
logout_time
register_temp
temp_cust_id
cust_name
cust_email
apt_num
cust_intro
password
cust_photo
active_status
approval_request_date
approval_date
approval_status
Approval

Relational Schema

Hoods (hood_id, hood_name)

Blocks (block_id, city_id, hood_id, block_name, city_name, block_latitude, block_longitude)

hood_id REFERENCES hood_id in HOODS

REGISTER_TEMP (temp_cust_id, cust_name, cust_email, city_id, block_id, apt_num, cust_intro, cust_photo, password, active_status)

block_id,city_id REFERENCES block_id,city_id in BLOCKS

Constraint: UNIQUE(cust_email,active_status)

CUSTOMER (cust_id, prev_cust_id, cust_name, cust_email, city_id, block_id, apt_num, cust_intro, cust_photo, password, join_time, active_status)

block_id,city_id REFERENCES block_id,city_id in BLOCKS

Constraint: UNIQUE(cust_email,active_status)

REQUEST_APPROVAL (temp_cust_id, approver_id, approval_request_date, approval_date, approval_status)

temp_cust_id REFERENCES temp_cust_id in REGISTER_TEMP

approver_id REFERENCES cust_id in CUSTOMER

FRIENDS (sent_by_cust, received_by_cust, request_date, request_status, response_date)

sent_by_cust REFERENCES cust_id in CUSTOMER

received_by_cust REFERENCES cust_id in CUSTOMER

DIRECT_NEIGHBOURS (user_id, neighbour_id, added_date)

user_id REFERENCES cust_id in CUSTOMER

neighbour_id REFERENCES cust_id in CUSTOMER

THREADS (thread_id, thread_subject, initiated_by, initiated_date, recipient_type, recipient_user_id)

initiated_by REFERENCES cust_id in CUSTOMER

recipient_user_id REFERENCES cust_id in CUSTOMER

MESSAGES (thread_id, message_id, message_author_id, message_body, message_location, message_date)

message_author_id REFERENCES cust_id in CUSTOMER

thread_id REFERENCES thread_id in THREADS

MESSAGE_BROADCAST (message_id, message_receiver_id, message_read)

message_receiver_id REFERENCES cust_id in CUSTOMER

message_id REFERENCES message_id in MESSAGES

LOGIN_HISTORY (cust_id, login_time, logout_time)

Cust_id REFERENCES cust_id in CUSTOMER

Assumptions / Justfications:

- Cities are divided into predefined **hoods** and **blocks**: a block would always be defined under a particular hood.
- Users would selects the city and block from a predefined list while signing up.
- When the user signs up, a entry would go into **REGISTER_TEMP** table with all the user details and status as 'active'. After the insert into the **REGISTER_TEMP,** a trigger(defined below) would be invoked which would send the approval requests to the pool of users from the same block, i.e insertion into **REQUEST_APPROVAL**. Also, if there are no existing users in the block for which the approval is being requested, the new user would be auto approved and would be inserted into **Customer** table.
- After each approval, one more trigger would be invoked to check if the necessary number of approvals are complete and if yes, the customer would go to the **CUSTOMER** table and would be active.
- If the user updates the address, the user's access to the website would be put on hold temporarily and the user would have to wait for the approval from at least 3 members from the new block he/she is applying to. After the approval, the user would have access to all the old messages he/she was part of from the previous block but would not be able to access any new messages from the old block. However, the user would now start receiving the messages from the new block. The user would be given a new customer ID in this case and the his previous customer ID would be stored in 'previous_cust_id' column in customer table in the new tuple with new cust_id. The record with old customer ID would be marked as inactive in the customer table and the register_temp table. Hence unique(cust_email, active_status) constraint is put on both the tables.
- Having previous_customer_id stored, we would be able to give the user, the access to the messages from the previous block.
- In the friends table, request_status would be either 'pending', 'approved' or 'rejected'. For status other than 'rejected', the pair 'sent_by_cust' and 'received_by_cust' should be unique considering both the orders. If the request_status is 'rejected', then the friend request can be sent again by either one so the pair would appear again in the table.
- Two users of same block can make each other their neighbour. These are not dependent on each other and can remove and make the same neighbour again, hence all three columns are the primary key.
- Whenever a thread is started, a unique thread ID would be generated. This thread contents would be considered as a first message of the thread. Hence entries would go into **THREADS**, **MESSAGES** and **MESSAGE_BROADCAST** table.
- A thread can be initiated to various type of recipients hence **THREADS** table has column 'recipient_type' which can have values 'friend', 'direct_neighbour', 'allfriends', 'block' or 'hood'. If the reciepient type field has 'friend' or 'direct_neighbour', then the 'recipient_user_id' field in the 'threads' table would have the cust_id of the friend/direct neighbour . Or else, 'recipient_user_id' would be null. This defines the reciepients of the thread.
- Once the thread is defined, the recipents cannot be modified for that thread.
- Whenever a message is sent, a unique message ID is geneated and data goes to **MESSAGES** table which contains the message details. The trigger defined below on after insert on messages table would put entires into **MESSAGE_BROADCAST.** This tables tracks the read status of each message per user to whom the message was sent.
- Every time the users logs in and logs out, **LOGIN_HISTORY** table would be populated with the approriate timestamps and customer ID. This would help us in getting all information on

sessions. For e.g. Listing all threads in a user's block feed that have new messages since the last time the user accessed the system.

Write the following SQL queries and execute them in your database system. Show the queries and the results:

Joining: Write a few (3-4) queries that users need to sign up, to apply to become members of a block, and to create or edit their profiles.

a) **When user enters the details and signs up, the data would go into the register_temp table as below. The user would fill all the details on the web page and the below stored procedure would be called on submit.**

```
DELIMITER //

CREATE PROCEDURE tempRegistration
(IN cust_name varchar(255),IN cust_email varchar(255),IN city_id int,in
block_id int, in apt_num int, in cust_intro varchar(4000), in cust_photo blob,
in password varchar(255))

begin

INSERT INTO REGISTER_TEMP (cust_name, cust_email, city_id, block_id, apt_num,
cust_intro,cust_photo, password,active_status)
VALUES (cust_name, cust_email, city_id, block_id, apt_num,
cust_intro,cust_photo, password,'active');

END //
DELIMITER ;
```

b) **After the insert into REGISTER_TEMP table, the below trigger written would be invoked which would send the approval requests to the pool of users in the block. Also, if there are no existing users in the block for which the approval is being requested, the new user would be auto approved and would be inserted into customer table.**

```
drop trigger if exists customer_approval_requests;

DELIMITER //

create trigger customer_approval_requests after insert
on REGISTER_TEMP for each row begin

declare total_existing_users_block integer;
declare approver integer;
declare finished integer default 0;
declare curApprovers
        cursor for
        select cust_id from CUSTOMER where block_id = new.block_id
        and active_status='active';
        -- declare NOT FOUND handler
declare continue HANDLER for not found set finished = 1;
```

```
select count(*) into total_existing_users_block
from CUSTOMER where block_id = new.block_id and active_status='active';

if total_existing_users_block = 0 then
       -- if no existing users then auto approve
       insert  into CUSTOMER
              (prev_cust_id, cust_name, cust_email, city_id, block_id, apt_num,
cust_intro, cust_photo, password, join_time, active_status)
              values
              (null
                , new.cust_name, new.cust_email, new.city_id, new.block_id,
new.apt_num, new.cust_intro, new.cust_photo, new.password, current_timestamp,
'active');

else
       open curApprovers;
       getApprover:loop
              fetch curApprovers into approver;

              if finished = 1 then
                     leave getApprover;
              end if;

              insert  into REQUEST_APPROVAL
                     (temp_cust_id, approver_id, approval_request_date,
approval_date, approval_status
                     )
                     values
                     (new.temp_cust_id, approver, current_timestamp,
null,'pending');

       end loop getApprover;
       close curApprovers;
end if;
end; //

DELIMITER ;
```

c) **On acknowledgement of each approval request (i.e. when any of the user approves the request), the status would be updated to 'approved' from the server in the REQUEST_APPROVAL table and the below trigger would be invoked which would check the total number of approval requests and the number of approval requests acknowledged. If the total number of request are less than 3 and all are acknowledged or if the number od requests are more than 3 and at least 3 are acknowledged, the customer would be inserted into the customer table.**

```
drop trigger if exists requests_approved;

DELIMITER //

create trigger requests_approved after update
on REQUEST_APPROVAL for each row begin

declare total_approval_requests integer;
declare approved_requests integer;
declare is_address_update integer;
declare is_already_approved integer;
declare v_prev_cust_id integer;
```

```sql
select count(*) into total_approval_requests
from REQUEST_APPROVAL
where temp_cust_id = old.temp_cust_id;

select count(*) into approved_requests
from REQUEST_APPROVAL
where temp_cust_id = old.temp_cust_id
and approval_status='approved';

-- To check if the customer is making an address update
select count(*) into is_address_update from customer c where c.cust_email =(
select distinct cust_email from register_temp rt
where rt.temp_cust_id=old.temp_cust_id)
and active_status='deactive';

-- To check if the customer is already approved
select count(*) into is_already_approved from customer c where c.cust_email =(
select distinct cust_email from register_temp rt
where rt.temp_cust_id=old.temp_cust_id)
and active_status='active';

if(is_already_approved = 0) then
	if total_approval_requests <3 then
		if(total_approval_requests=approved_requests) then
			insert  into CUSTOMER
			(prev_cust_id, cust_name, cust_email, city_id, block_id,
apt_num, cust_intro, cust_photo, password, join_time, active_status)
			select null as prev_cust_id, cust_name, cust_email,
city_id, block_id, apt_num, cust_intro,cust_photo, password, current_timestamp
as join_time, 'active' as active_status
			from register_temp
			where temp_cust_id=old.temp_cust_id;

			if is_address_update >0 then
				select cust_id into v_prev_cust_id from customer c
				where c.cust_email =(select distinct cust_email from
register_temp rt
							where
rt.temp_cust_id=old.temp_cust_id)
				and active_status='deactive' order by join_time desc
limit 1;

				update customer set prev_cust_id=v_prev_cust_id
				where cust_email =(select distinct cust_email from
register_temp rt
							where
rt.temp_cust_id=old.temp_cust_id)
				and active_status='active';
			end if;
		end if;
else
		if(approved_requests >= 3) then
			insert  into CUSTOMER
			(prev_cust_id, cust_name, cust_email, city_id, block_id,
apt_num, cust_intro, cust_photo, password, join_time, active_status)
			select null as prev_cust_id, cust_name, cust_email,
city_id, block_id, apt_num, cust_intro,cust_photo, password, current_timestamp
as join_time, 'active' as active_status
			from register_temp
			where temp_cust_id=old.temp_cust_id;
```

```
                    if is_address_update >0 then
                          select cust_id into v_prev_cust_id from customer c
                          where c.cust_email =(select distinct cust_email from
register_temp rt
                                       where rt.temp_cust_id=old.temp_cust_id)
             and active_status='deactive' order by join_time desc limit 1;

                          update customer set prev_cust_id=v_prev_cust_id
                          where cust_email =(select distinct cust_email from
register_temp rt
                                              where
rt.temp_cust_id=old.temp_cust_id)
                          and active_status='active';

                    end if;
             end if;
       end if;
end if;

end; //
DELIMITER ;
```

d) When user updates his/her address, the below procedure gets called. After which the user is disabled and new approvals are triggered through the trigger on register_temp.

```
DELIMITER //

CREATE PROCEDURE updateAddress
(IN v_cust_id int,IN v_city_id int,IN v_block_id int,in v_apt_num int)
begin

DECLARE v_cust_email varchar(255);
DECLARE v_cust_name varchar(255);
DECLARE v_cust_intro varchar(255);
DECLARE v_cust_photo blob;
DECLARE v_password varchar(255);

select cust_name, cust_email, cust_intro,cust_photo, password
into v_cust_name, v_cust_email, v_cust_intro,v_cust_photo, v_password
from customer
where cust_id=v_cust_id
and active_status='active';

update customer
set active_status='deactive'
where cust_id=v_cust_id
and active_status='active';

update register_temp
set active_status='deactive'
where cust_email=v_cust_email
and active_status='active';

INSERT INTO REGISTER_TEMP (cust_name, cust_email, city_id, block_id, apt_num,
cust_intro,cust_photo, password,active_status)
VALUES (v_cust_name, v_cust_email, v_city_id, v_block_id, v_apt_num,
v_cust_intro,v_cust_photo, v_password,'active');
END
```

d) Creation of the customer profile is done when the user signs up. This profile is available to the user for editing, after he is approved. If the user, updates his/her profile, the customer table would be updated. Suppose, he/she updates both intro and photo.

```
update customer set cust_intro='updated intro'
and cust_photo='new photo'
where cust_id=1;
```

Content Posting: Write queries that implement what happens when a user starts a new thread by posting an initial message, and replies to a message.

a) When user starts a thread, the below stored procedure would be called and there would be an insert operation on the 'threads' table and the thread would be given an Thread ID by the system. Also, while creating the thread, the user would select the reciepient type which might be a direct neighbour, a friend, all friends, the whole block or the whole hood.

Based on the reciepient type selected, the 'reciepient_type' field would have the corresponding value. If the reciepient type field has 'friend' or 'direct_neighbour', then the 'recipient_user_id' field in the 'threads' table would have the cust_id of the friend/direct neighbour . Or else, 'recipient_user_id' would be null. This defines the reciepients of the thread.

Also, along with a thread, there would an automatic insert into 'messages' for the above thread ID. A new message ID would be generated by the system and this would be considered the first message of the thread. This table would have the author of the message and message details.

```
DELIMITER //

CREATE PROCEDURE createThread
(IN thread_subject varchar(4000), IN thread_content varchar(4000), IN
initiated_by int, IN recipient_type VARCHAR(30), IN recipient_user_id int,in
gps_coordinates double)

begin
DECLARE v_thread_id INT;
set @currentTime = now();

INSERT INTO THREADS (thread_subject, initiated_by, initiated_date,
recipient_type, recipient_user_id)
VALUES
(thread_subject, initiated_by ,@currentTime, recipient_type,
recipient_user_id);

select max(thread_id) into v_thread_id from THREADS;

INSERT INTO MESSAGES (thread_id, message_author_id, message_body,
message_location, message_date)
VALUES
(v_thread_id, initiated_by, thread_content , gps_coordinates , @currentTime);

END //

DELIMITER ;
```

Also, based on the reciepient selected, for each reciepient, there would be an automatic entry in the ‘message_broadcast’ table where read status of each message would be tracked. This would be handled by the below trigger on the messages table.

```
drop trigger if exists message_broadcasts;

DELIMITER //

create trigger message_broadcasts after insert
on MESSAGES  for each row begin

declare v_recipient_type varchar(255);
declare v_recipient_user_id integer;
declare v_initiated_by integer;

declare finished integer default 0;

declare curallFriends
       cursor for
             select received_by_cust as recipients from FRIENDS
             where sent_by_cust = (select initiated_by from threads where
thread_id = new.Thread_id)
             and request_status='approved'
             union
             select sent_by_cust as recipients from FRIENDS
             where received_by_cust = (select initiated_by from threads where
thread_id = new.Thread_id)
             and request_status='approved';


declare curblock
       cursor for
                    select cust_id as recipients  from customer
             where block_id=(
                    select block_id from customer
                    where cust_id=(select initiated_by from threads where
thread_id = new.Thread_id)
                    and active_status='active')
             and active_status='active'
             and cust_id <> (select initiated_by from threads where thread_id =
new.Thread_id);


declare curhood
       cursor for
             select a.cust_id as recipients from customer a, blocks b
             where a.block_id=b.block_id
             and b.hood_id = (select hood_id from customer, blocks
                    where cust_id=(select initiated_by from threads where
thread_id = new.Thread_id)
                    and customer.block_id=blocks.block_id)
             and a.active_status='active'
             and a.cust_id <> (select initiated_by from threads where thread_id
= new.Thread_id);

       -- declare NOT FOUND handler
declare continue HANDLER for not found set finished = 1;
```

```
select recipient_type,initiated_by into v_recipient_type,v_initiated_by from
threads where thread_id =new.Thread_id;

if(v_initiated_by=new.message_author_id) then
INSERT INTO MESSAGE_BROADCAST (message_id, message_receiver_id, message_read)
      VALUES (new.message_id, v_initiated_by, 'READ');
else
INSERT INTO MESSAGE_BROADCAST (message_id, message_receiver_id, message_read)
      VALUES (new.message_id, v_initiated_by, 'UNREAD');
end if;

if(v_recipient_type= 'friend' or v_recipient_type= 'direct_neighbour') then
      select recipient_user_id into v_recipient_user_id from threads where
thread_id =new.Thread_id;

      if(v_recipient_user_id=new.message_author_id) then
            INSERT INTO MESSAGE_BROADCAST (message_id, message_receiver_id,
message_read)
            VALUES (new.message_id, v_recipient_user_id, 'READ');
      else
            INSERT INTO MESSAGE_BROADCAST (message_id, message_receiver_id,
message_read)
            VALUES (new.message_id, v_recipient_user_id, 'UNREAD');
      end if;

elseif (v_recipient_type='allfriends') then
      open curallFriends;
      allfrnds:loop
            fetch curallFriends into v_recipient_user_id;

            if finished = 1 then
                  leave allfrnds;
            end if;
      if(v_recipient_user_id=new.message_author_id) then
            INSERT INTO MESSAGE_BROADCAST (message_id, message_receiver_id,
message_read)
            VALUES (new.message_id, v_recipient_user_id, 'READ');
      else
            INSERT INTO MESSAGE_BROADCAST (message_id, message_receiver_id,
message_read)
            VALUES (new.message_id, v_recipient_user_id, 'UNREAD');
      end if;

      end loop allfrnds;
      close curallFriends;
```

```
elseif (v_recipient_type='block') then
       open curblock;
       allblock:loop
              fetch curblock into v_recipient_user_id;

              if finished = 1 then
                     leave allblock;
              end if;
       if(v_recipient_user_id=new.message_author_id) then
              INSERT INTO MESSAGE_BROADCAST (message_id, message_receiver_id,
message_read)
              VALUES (new.message_id, v_recipient_user_id, 'READ');
       else
              INSERT INTO MESSAGE_BROADCAST (message_id, message_receiver_id,
message_read)
              VALUES (new.message_id, v_recipient_user_id, 'UNREAD');
       end if;

       end loop allblock;
       close curblock;

elseif (v_recipient_type='hood') then
       open curhood;
       allhood:loop
              fetch curhood into v_recipient_user_id;

              if finished = 1 then
                     leave allhood;
              end if;
       if(v_recipient_user_id=new.message_author_id) then
              INSERT INTO MESSAGE_BROADCAST (message_id, message_receiver_id,
message_read)
              VALUES (new.message_id, v_recipient_user_id, 'READ');
       else
              INSERT INTO MESSAGE_BROADCAST (message_id, message_receiver_id,
message_read)
              VALUES (new.message_id, v_recipient_user_id, 'UNREAD');
       end if;

       end loop allhood;
       close curhood;

end if;
end; //
DELIMITER ;
```

b) When a user replies to a message, the below stored procedure would be called and system will insert an entry to messages table where the thread ID would come from the thread on which the user is replying, the author ID would be of the logged in user, and the message content would be provided by the user. The recipents of the message would be same as the reciepients defined in the thread. The trigger on messages table would insert records into message broadcast table.

```
DELIMITER //

CREATE PROCEDURE createNewMessage
(IN thread_id int,IN message_author_id int,IN message_body varchar(4000),in
gps_coordinates double)

begin

DECLARE v_thread_id INT;

set @currentTime = now();


INSERT INTO MESSAGES (thread_id, message_author_id, message_body,
message_location, message_date)
VALUES
(thread_id, message_author_id, message_body , gps_coordinates , @currentTime);

END //

DELIMITER ;
```

After the insertion in messages table, the entries would go into message broadcast table based on the recipient type which would be handled by ‘`message_broadcasts`’ trigger defined above.

Friendship: Write queries that users can use to add or accept someone as their friend or neighbor, and to list all their current friends and neighbors.

a) For adding a friend, user would send a friend request to other user from his/her own hood.

```
DELIMITER //

CREATE PROCEDURE addFriend
(IN sent_by int, sent_to int)
begin
	INSERT INTO FRIENDS (sent_by_cust, received_by_cust, request_status,
request_date, response_date)
VALUES
(sent_by, sent_to, 'pending', current_timestamp, null);

END //

DELIMITER ;
```

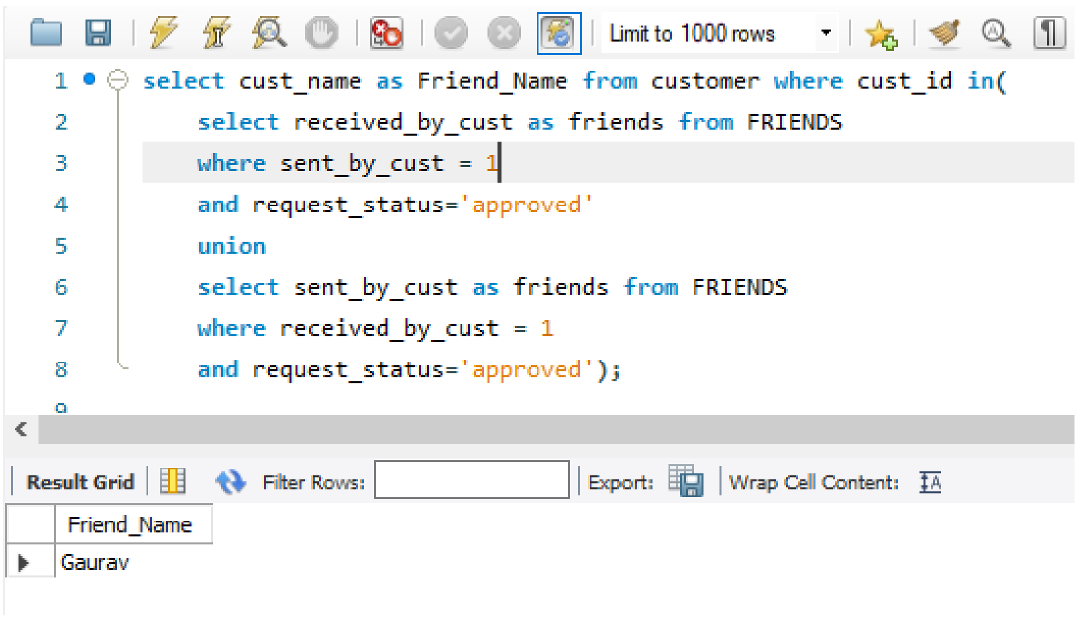
When a user accepts the friend request, request_status column in FRIENDS table is updated with the value ‘approved’

b) For adding a direct neighbour, user can select anyone from their block.

```
DELIMITER //

CREATE PROCEDURE addneighbour
(IN added_by int, added_user int)
begin
      INSERT INTO DIRECT_NEIGHBOURS (user_id, neighbour_id, added_date)
VALUES
(added_by, added_user, current_timestamp);

END //

DELIMITER ;
```

c) Query to list all the current friends for a user.

```
select cust_name as Friend_Name from customer where cust_id in(
      select received_by_cust as friends from FRIENDS
      where sent_by_cust = <loggedin_user_id>
      and request_status='approved'
      union
      select sent_by_cust as friends from FRIENDS
      where received_by_cust = < loggedin_user_id>
      and request_status='approved');
```



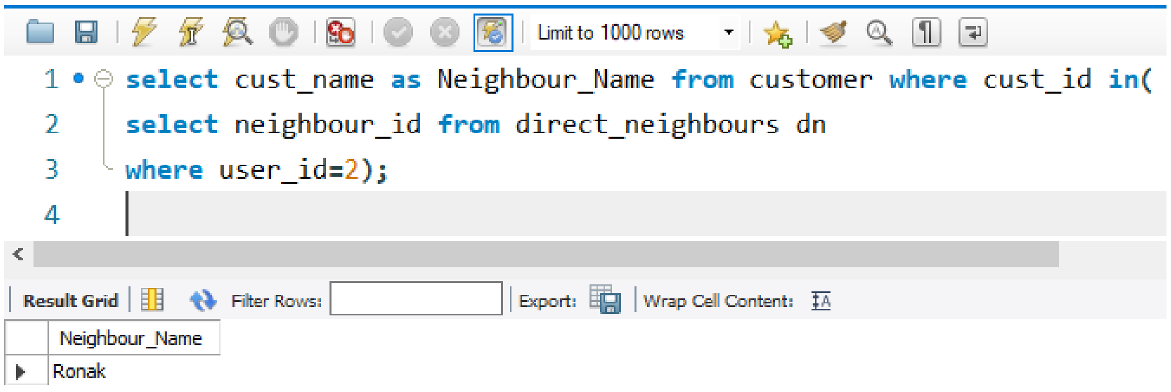
```
select cust_name as Friend_Name from customer where cust_id in(
    select received_by_cust as friends from FRIENDS
    where sent_by_cust = 1
    and request_status='approved'
    union
    select sent_by_cust as friends from FRIENDS
    where received_by_cust = 1
    and request_status='approved');
```

Friend_Name
Gaurav

Note: Taken cust_id as 1 for testing

d) Query to list all the current neighbours for a user.

```
select cust_name as Neighbour_Name from customer where cust_id in(
      select neighbour_id from direct_neighbours dn
      where user_id=<loggedin_user_id>);
```



```
select cust_name as Neighbour_Name from customer where cust_id in(
select neighbour_id from direct_neighbours dn
where user_id=2);
```

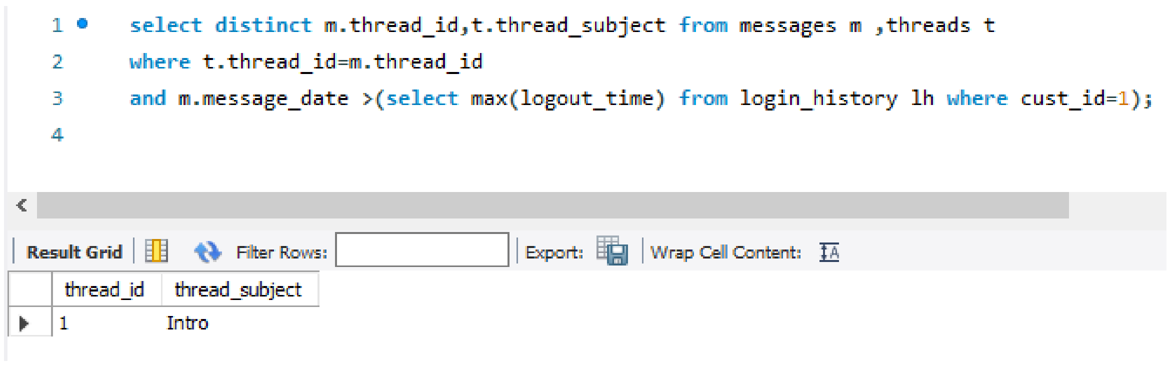
```
Neighbour_Name
Ronak
```

Note: Taken cust_id as 2 for testing

Browse and Search Messages: Write a few (say 3-4) different queries that might be useful when a user accesses content. For example, list all threads in a user's block feed that have new messages since the last time the user accessed the system, or all threads in her friend feed that have unread messages, or all messages containing the words "bicycle accident" across all feeds that the user can access.

a) List all threads in a user's block feed that have new messages since the last time the user accessed the system

```
select distinct m.thread_id,t.thread_subject from messages m ,threads t
where t.thread_id=m.thread_id
and m.message_date >(select max(logout_time) from login_history lh where
cust_id=<loggedin_user_id>);
```



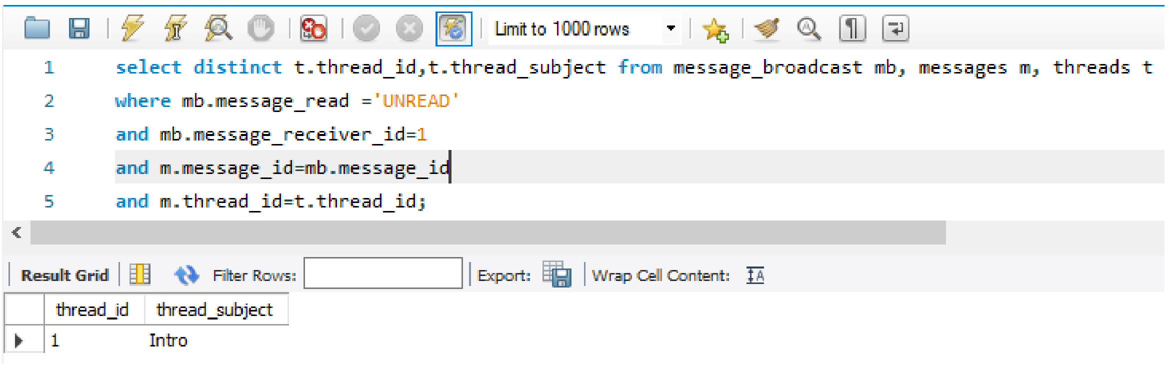
```
select distinct m.thread_id,t.thread_subject from messages m ,threads t
where t.thread_id=m.thread_id
and m.message_date >(select max(logout_time) from login_history lh where cust_id=1);
```

```
thread_id   thread_subject
1           Intro
```

Note: Taken cust_id as 1 for testing

b) All threads in her friend feed that have unread messages

```
select distinct t.thread_id,t.thread_subject from message_broadcast mb,
messages m, threads t
where mb.message_read ='UNREAD'
and mb.message_receiver_id=<loggedin_user_friend_user_id>
and m.message_id=mb.message_id
and m.thread_id=t.thread_id;
```



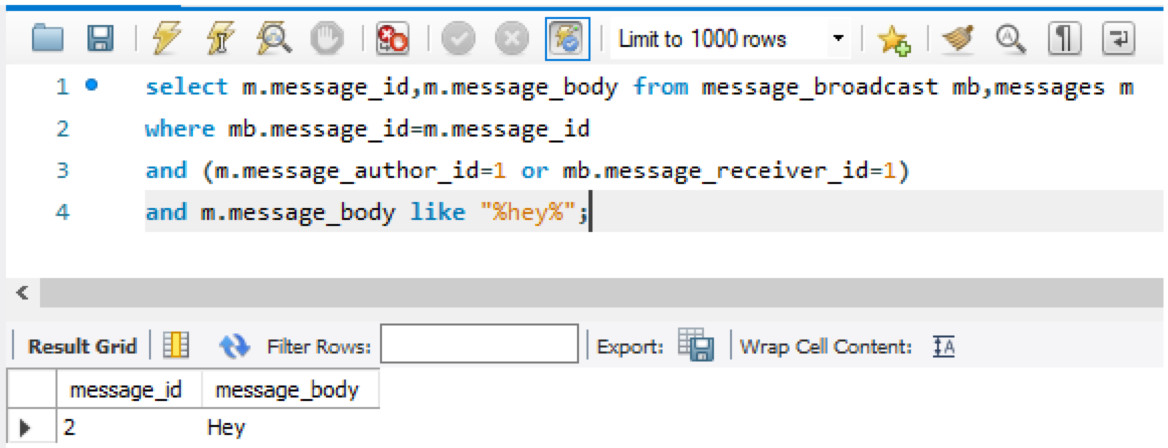
```
select distinct t.thread_id,t.thread_subject from message_broadcast mb, messages m, threads t
where mb.message_read ='UNREAD'
and mb.message_receiver_id=1
and m.message_id=mb.message_id
and m.thread_id=t.thread_id;
```

thread_id	thread_subject
1	Intro

Note: Taken friend_user_id as 1 for testing

b) All messages containing the words "bicycle accident" across all feeds that the user can access.

```
select m.message_id,m.message_body from message_broadcast mb,messages m
where mb.message_id=m.message_id
and (m.message_author_id=<user_id> or mb.message_receiver_id=<user_id>)
and m.message_body like "%bicycle accident%";
```



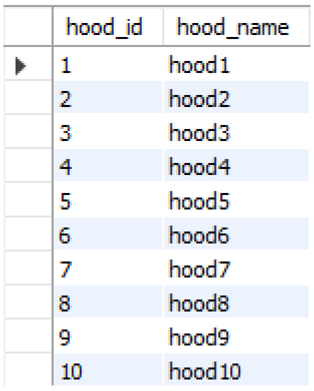
```
select m.message_id,m.message_body from message_broadcast mb,messages m
where mb.message_id=m.message_id
and (m.message_author_id=1 or mb.message_receiver_id=1)
and m.message_body like "%hey%";
```

message_id	message_body
2	Hey

Note: Taken user_id as 1 for testing and '%hey%' instead of 'bicycle accident'

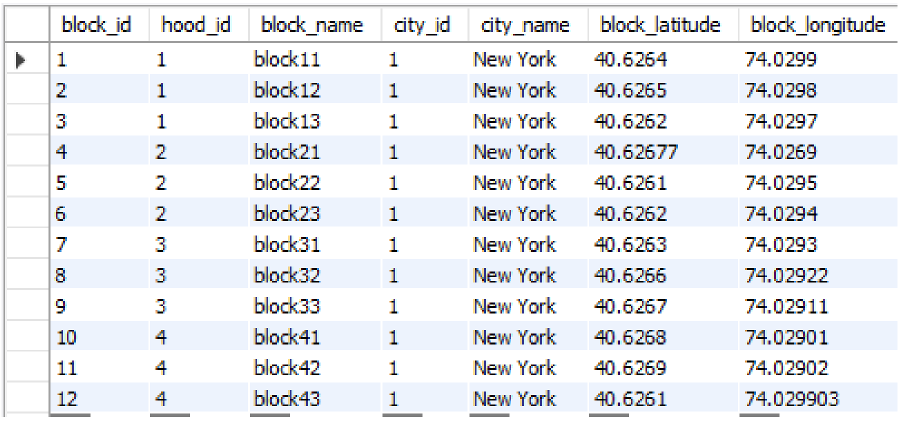
Draw and submit a little picture of your tables that fits on one or two pages and that illustrates your test data.

HOODS



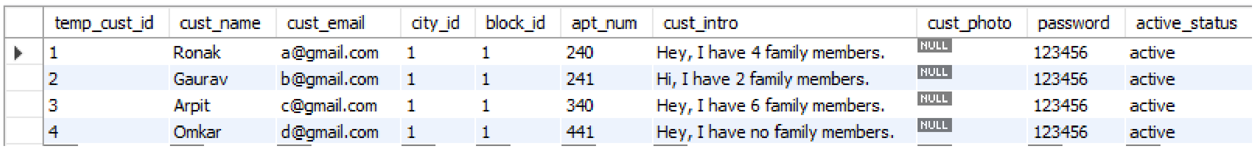
hood_id	hood_name
1	hood1
2	hood2
3	hood3
4	hood4
5	hood5
6	hood6
7	hood7
8	hood8
9	hood9
10	hood10

BLOCKS



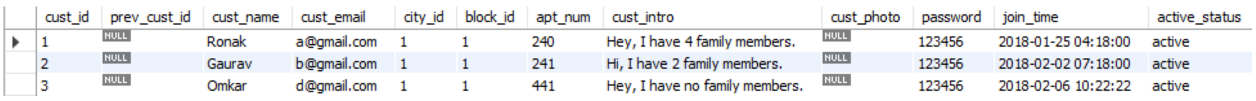
block_id	hood_id	block_name	city_id	city_name	block_latitude	block_longitude
1	1	block11	1	New York	40.6264	74.0299
2	1	block12	1	New York	40.6265	74.0298
3	1	block13	1	New York	40.6262	74.0297
4	2	block21	1	New York	40.62677	74.0269
5	2	block22	1	New York	40.6261	74.0295
6	2	block23	1	New York	40.6262	74.0294
7	3	block31	1	New York	40.6263	74.0293
8	3	block32	1	New York	40.6266	74.02922
9	3	block33	1	New York	40.6267	74.02911
10	4	block41	1	New York	40.6268	74.02901
11	4	block42	1	New York	40.6269	74.02902
12	4	block43	1	New York	40.6261	74.029903

REGISTER_TEMP



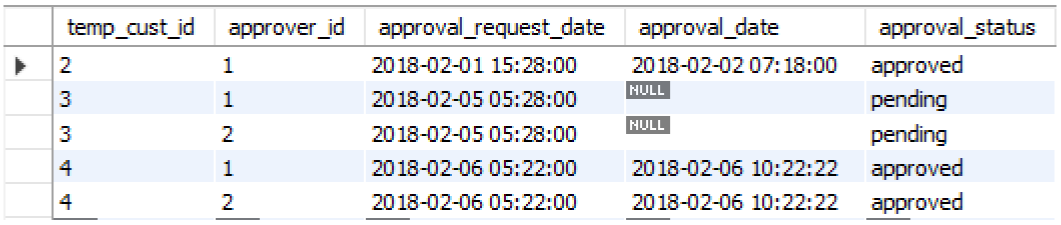
temp_cust_id	cust_name	cust_email	city_id	block_id	apt_num	cust_intro	cust_photo	password	active_status
1	Ronak	a@gmail.com	1	1	240	Hey, I have 4 family members.	NULL	123456	active
2	Gaurav	b@gmail.com	1	1	241	Hi, I have 2 family members.	NULL	123456	active
3	Arpit	c@gmail.com	1	1	340	Hey, I have 6 family members.	NULL	123456	active
4	Omkar	d@gmail.com	1	1	441	Hey, I have no family members.	NULL	123456	active

CUSTOMER



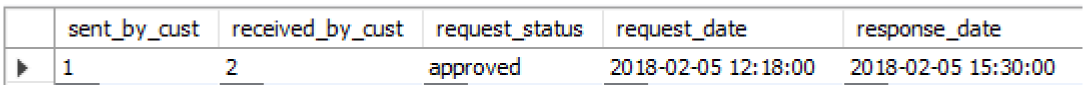
cust_id	prev_cust_id	cust_name	cust_email	city_id	block_id	apt_num	cust_intro	cust_photo	password	join_time	active_status
1	NULL	Ronak	a@gmail.com	1	1	240	Hey, I have 4 family members.	NULL	123456	2018-01-25 04:18:00	active
2	NULL	Gaurav	b@gmail.com	1	1	241	Hi, I have 2 family members.	NULL	123456	2018-02-02 07:18:00	active
3	NULL	Omkar	d@gmail.com	1	1	441	Hey, I have no family members.	NULL	123456	2018-02-06 10:22:22	active

REQUEST_APPROVAL



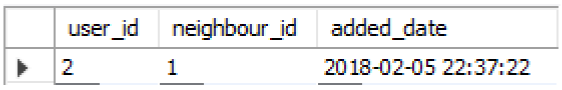
temp_cust_id	approver_id	approval_request_date	approval_date	approval_status
2	1	2018-02-01 15:28:00	2018-02-02 07:18:00	approved
3	1	2018-02-05 05:28:00	NULL	pending
3	2	2018-02-05 05:28:00	NULL	pending
4	1	2018-02-06 05:22:00	2018-02-06 10:22:22	approved
4	2	2018-02-06 05:22:00	2018-02-06 10:22:22	approved

FRIENDS



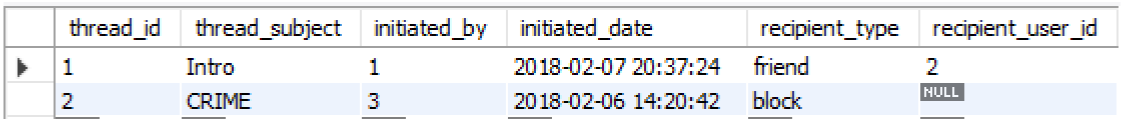
sent_by_cust	received_by_cust	request_status	request_date	response_date
1	2	approved	2018-02-05 12:18:00	2018-02-05 15:30:00

DIRECT_NEIGHBOURS



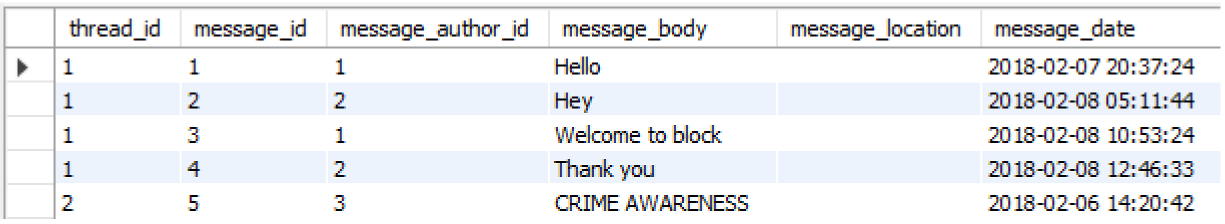
user_id	neighbour_id	added_date
2	1	2018-02-05 22:37:22

THREADS



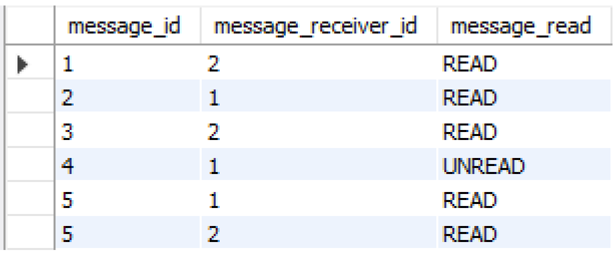
thread_id	thread_subject	initiated_by	initiated_date	recipient_type	recipient_user_id
1	Intro	1	2018-02-07 20:37:24	friend	2
2	CRIME	3	2018-02-06 14:20:42	block	NULL

MESSAGES



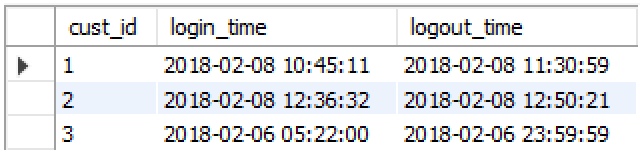
thread_id	message_id	message_author_id	message_body	message_location	message_date
1	1	1	Hello		2018-02-07 20:37:24
1	2	2	Hey		2018-02-08 05:11:44
1	3	1	Welcome to block		2018-02-08 10:53:24
1	4	2	Thank you		2018-02-08 12:46:33
2	5	3	CRIME AWARENESS		2018-02-06 14:20:42

MESSAGE_BROADCAST



message_id	message_receiver_id	message_read
1	2	READ
2	1	READ
3	2	READ
4	1	UNREAD
5	1	READ
5	2	READ

LOGIN_HISTORY



cust_id	login_time	logout_time
1	2018-02-08 10:45:11	2018-02-08 11:30:59
2	2018-02-08 12:36:32	2018-02-08 12:50:21
3	2018-02-06 05:22:00	2018-02-06 23:59:59

Project Part 2- Frontend Design

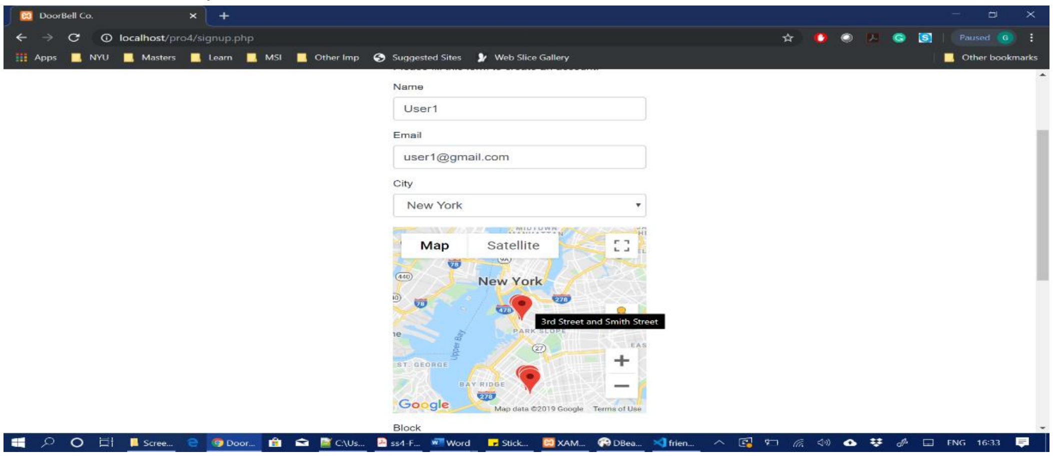
Design and Architechture

- We have designed the frontend using **HTML, CSS** and have used **Bootstrap** extensively.
- PHP is used as the server side technology along with **JavaScript** and **Jquery** for client side validations.
- The database used is **MariaDB** which comes along with Xampp.
- Our website 'DOORBELL.CO' focuses on brilliant user experience along with key features and functionalities.
- We have taken care of all the security issues like Permission Check, XSS/ CSRF, SQL injection and password hasing.
    - XSS/CSRF is implemented by using htmlspecialchars in php, starting a session only when user logs in and destroying the session when users logs out.
    - Permission Check is implemented as the session is being checked on every page so user cannot use browser's back button after he/she logs out. Also, user cannot directly access any PHP page (except singup and login) of the website because these are restriced to only signed in users.
    - We have used mySQL prepared statements along with bind_param to prevent SQL injection for all the POST calls.
    - We are hashing the password before storing it into database so we don't know the actual password looking at the database.
- When users log in, they are landed to a starting page where they can see all the threads ordered by date in descing order and all the messages in the threads, assecible to them based on their block, hood etc.
- Using JavaScript, all the necessary validations are in place like
    - User cannot signup without filling in all the details in the signup form.
    - The email field format has to be appropriate.
    - Same email cannot be registered again.
- Since we are using MariaDB as our database and we have taken care of the security measures in PHP, this website is scalable in real time.

Key Features

We have implemented some really cool features like

- Selecting the block while signing up, using google maps. The blocks are automatically marked on the map, with block name as markers, when the user selects the city. User can then click on any marker and select the block which he/she wants to join. This is implemented using google maps API.   Also, there is a block dropdown which can be used alternatively.



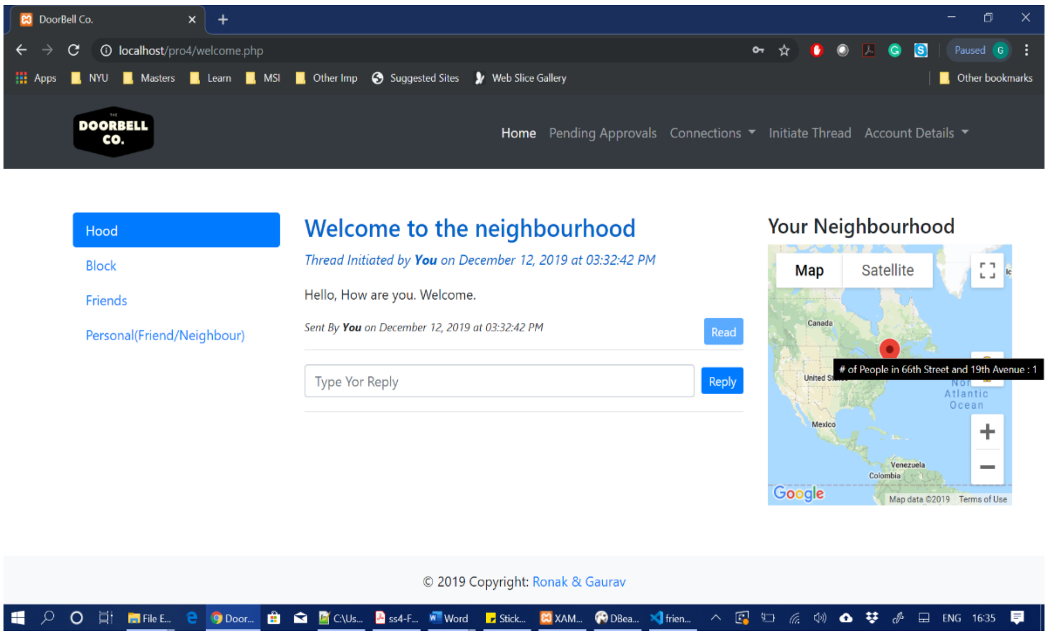
Name
User1

Email
user1@gmail.com

City
New York

Block

- Showing the number of people in the neighbourhood of a user, when the user logs in. It would display the blocks in the neighbourhood of the logged in user with number of people in each block.



Welcome to the neighbourhood

*Thread Initiated by **You** on December 12, 2019 at 03:32:42 PM*

Hello, How are you. Welcome.

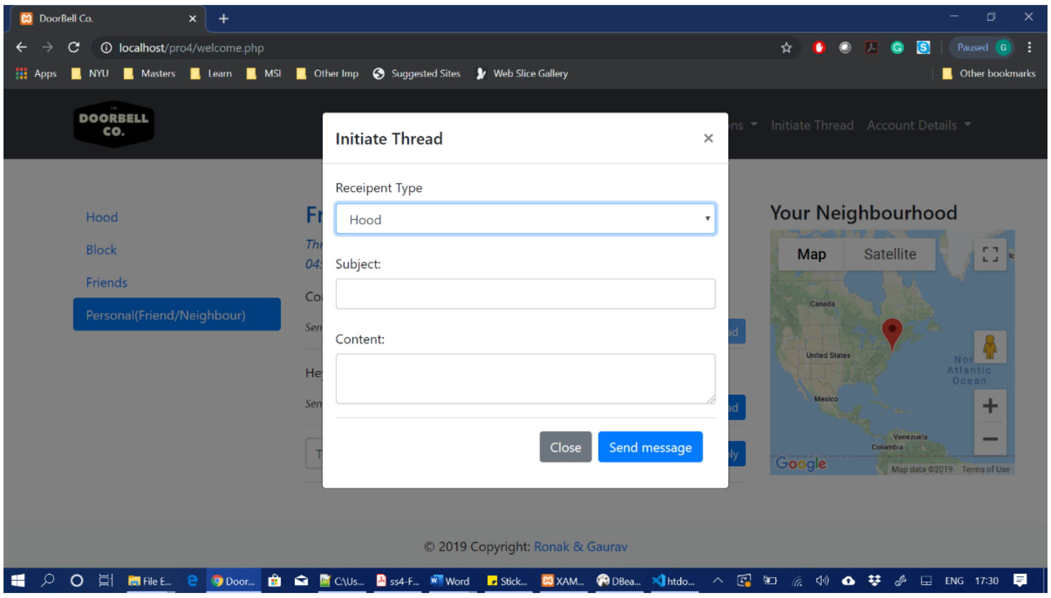
*Sent By **You** on December 12, 2019 at 03:32:42 PM*

Your Neighbourhood

# of People in 66th Street and 19th Avenue : 1

© 2019 Copyright: Ronak & Gaurav

- Initiating threads with various options like initiating to the whole hood, block, all friends or personally to a friend or a neighbour.



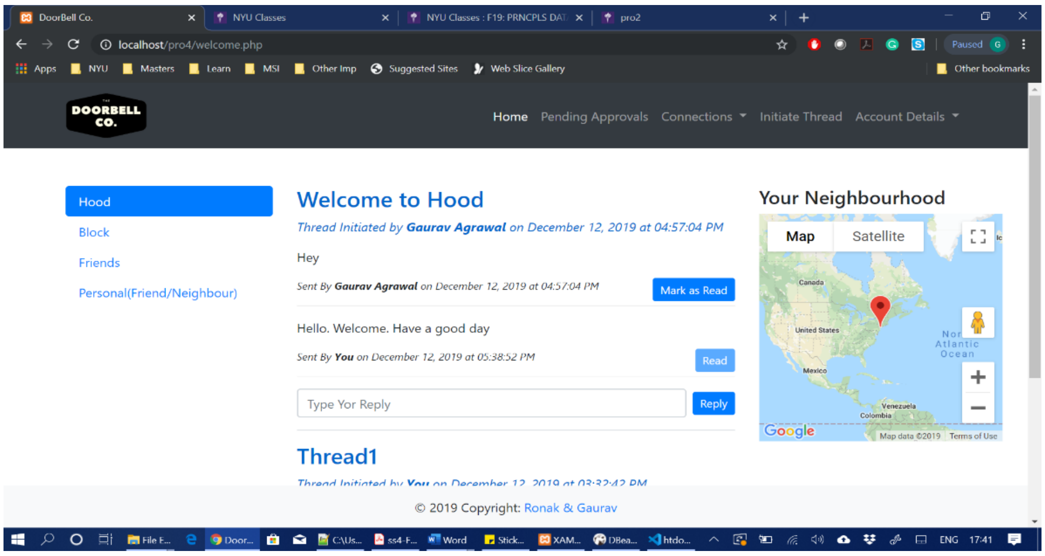
Initiate Thread

Receipent Type: Hood

Subject:

Content:

- Every thread initiated is segregated into tabs based on the reciepient type. Every thread initated and every message sent has the detailed information of who sent the message and when and to whom.
- Users can mark the received messages as Read which then gets stored into database as read.



Welcome to Hood

*Thread Initiated by **Gaurav Agrawal** on December 12, 2019 at 04:57:04 PM*

Hey

*Sent By **Gaurav Agrawal** on December 12, 2019 at 04:57:04 PM*

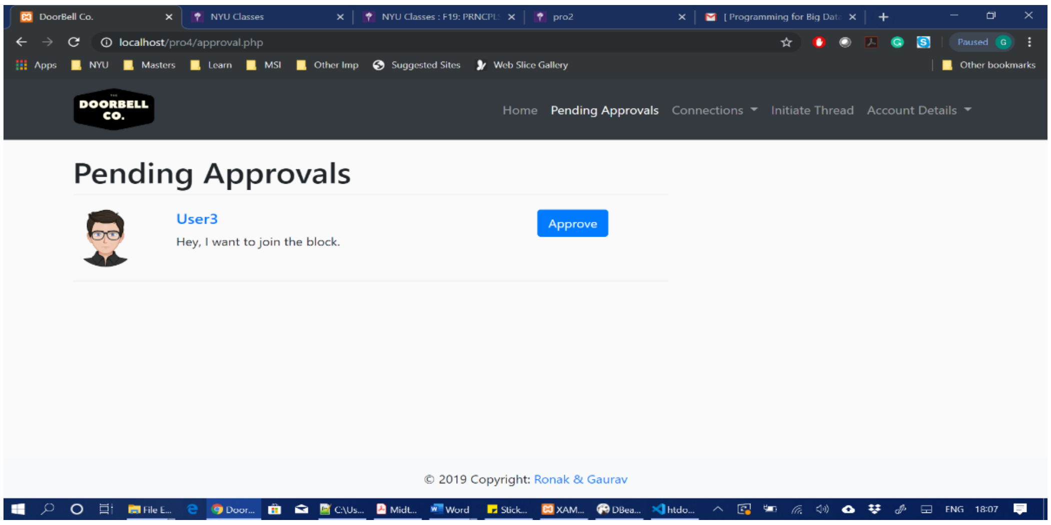
Hello. Welcome. Have a good day

*Sent By **You** on December 12, 2019 at 05:38:52 PM*

Thread1

© 2019 Copyright: Ronak & Gaurav

- The second page is pending approvals where the user would get the request to approve if a new member requests to join the same block. The approver can see the new users name and intro before approving.



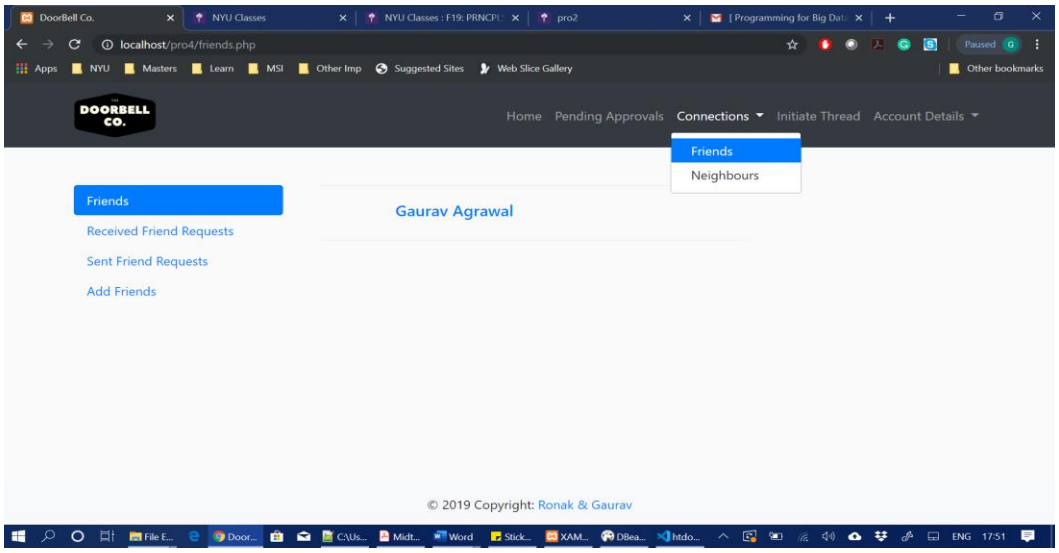
Pending Approvals

User3

Hey, I want to join the block.

© 2019 Copyright: Ronak & Gaurav

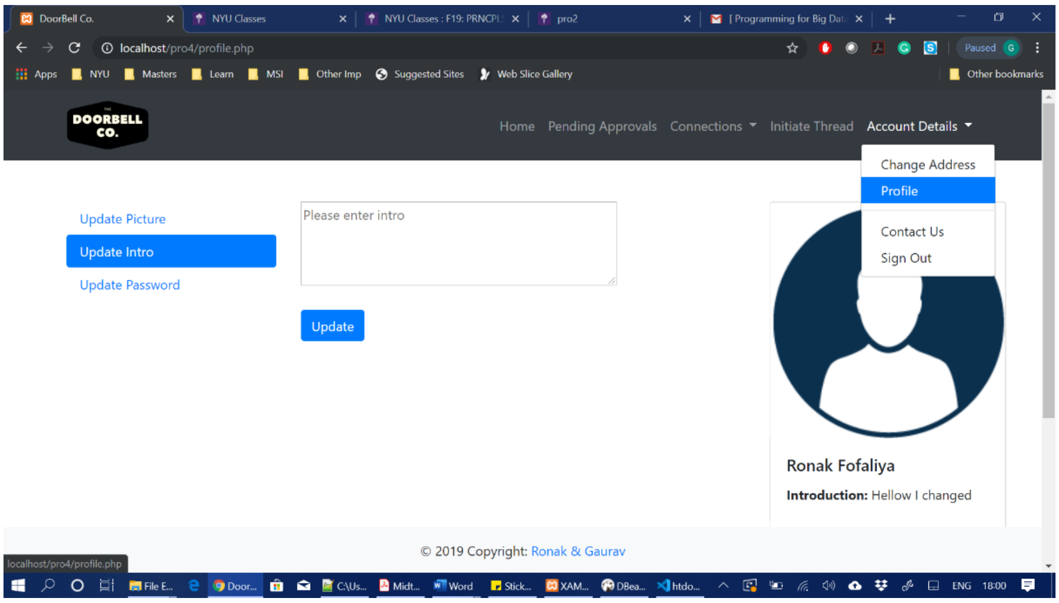
- Next we the connections tab where the user can see his/her friends and neighbours.They can add friends and neighbours as well on friends and neighbours page respectively



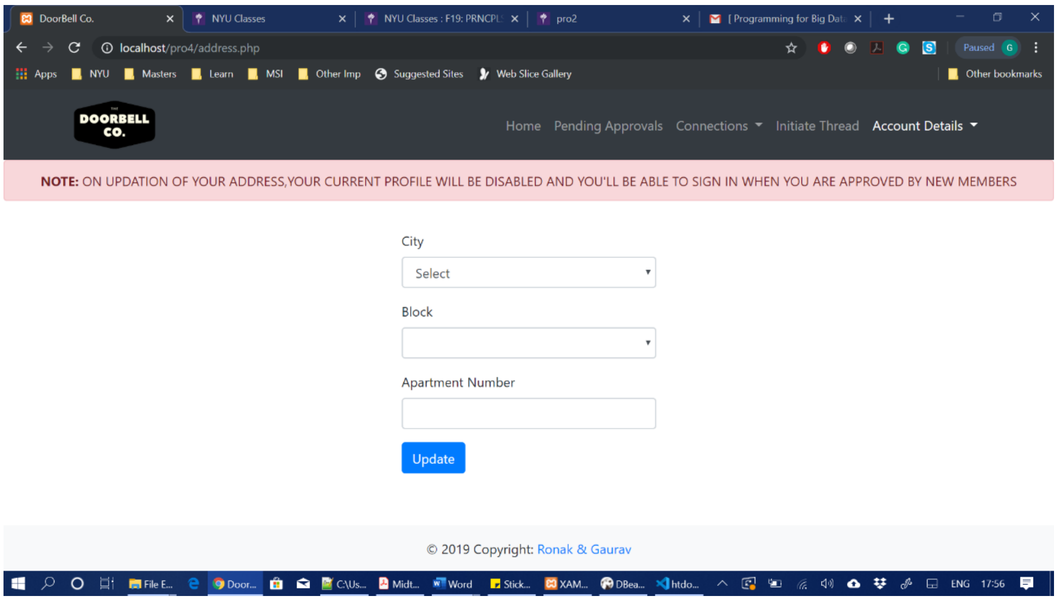
Gaurav Agrawal

© 2019 Copyright: Ronak & Gaurav

- Also, there is profile page where user can see his/her profile information. Also they have an option to change their profile information. This is a real time update. As soon as the user clicks on update, he/she would be able to see the changed information in his/her profile.



- The user can also opt for an address change after which the user would be logged out automatically and would have to go through the approval process again. There is validation here that the user cannot select the block which he/she is already in.


localhost/pro4/address.php
Home
Pending Approvals
Connections
Initiate Thread
Account Details
NOTE: ON UPDATION OF YOUR ADDRESS,YOUR CURRENT PROFILE WILL BE DISABLED AND YOU'LL BE ABLE TO SIGN IN WHEN YOU ARE APPROVED BY NEW MEMBERS
City
Select
Block
Apartment Number
Update
© 2019 Copyright: Ronak & Gaurav

Attached files

Along with this file a zipped folder (**ga1380-rf1999-pdsproject**) is attached which contains the following:

1. The folder (**pro4**) for all the PHP code.
2. A readme on how to run the code.
3. Script to create the schema (**Schema.sql**)
4. Script to Insert predefined blocks and hoods (**Insert.sql**)
5. Script to create needed procedures and triggers (**Script.sql**)