

CSGY-6083

Principles of Database Systems

Project Report

Submitted by:

Rahul (rr3687)
Dinesh Sreekanthan (ds5786)

Submitted to:

Prof. Torsten Suel



Relational Schema

Tables

signin (uemail, upass)

user (uemail, ufname, ulname, ubio, upic, ufamily)

address (uemail, ustreet, uapt, ucity, ucountry, uzip)

neighborhood (neighborhood_id, neighbourhoud_name, nSW_latitude, nSW_longitude, nNE_latitude, nNE_longitude)

block (neighborhood_id, block_id, Block_name, bSW_latitude, bSW_longitude, bNE_latitude, bNE_longitude)

apply (applicant_email, neighbourhood_id, block_id)

verified (verification_id, applicant_email, neighbourhood_id, block_id, verifieremail_1, verifieremail_2, verifieremail_3)

residents (uemail, neighbourhood_id, block_id)

friend_request (uemail_1, uemail_2, status)

friends (uemail_1, uemail_2)

direct_neighbours (uemail_1, uemail_2)

message (message_id, receiver_type, friend_neighbor_email, sender_email, mthread, msubject, mtext, mtimestamp)

inbox (message_id, is_read, sender_email, receiver_email, mthread, msubject, mtext, mtimestamp)

reply (reply_message_id, responder, mthread, mtext, mtimestamp)

logs (uemail, ltimestamp)

Foreign Key Constraints

'uemail' in 'user' references to 'uemail' in 'signin'

'uemail' in 'address' references to 'uemail' in 'user'

'neighborhood_id' in 'block' references to 'neighborhood_id' in 'neighborhood'

'applicant_email' in 'apply' references to 'uemail' in 'address'

'neighbourhood_id' in 'apply' references to 'neighbourhood_id' in 'block'

'block_id' in 'apply' references to 'block_id' in 'block'

'applicant_email' in 'verified' references to 'applicant_email' in 'apply'

'neighbourhood_id' in 'verified' references to 'neighbourhood_id' in 'apply'

'block_id' in 'verified' references to 'block_id' in 'apply'

'verifieremail_1' in 'verified' references to 'uemail' in 'residents'

'verifieremail_2' in 'verified' references to 'uemail' in 'residents'

'verifieremail_3' in 'verified' references to 'uemail' in 'residents'

'uemail' in 'residents' references to 'uemail' in 'address'

'neighbourhood_id' in 'residents' references to neighbourhood_id in 'block'

'block_id' in 'residents' references to 'block_id' in 'block'

'uemail_1' in 'friend_request' references to 'uemail' in 'residents'

'uemail_2' in 'friend_request' references to 'uemail' in 'residents'

'uemail_1' in 'friends' references to 'uemail' in 'residents'

'uemail_2' in 'friends' references to 'uemail' in 'residents'

'uemail_1' in 'direct_neighbors' references to 'uemail' in 'residents'

'uemail_2' in "direct_neighbors" references to 'uemail' in 'residents'

'sender_email' in 'message' references to 'uemail' in 'residents'

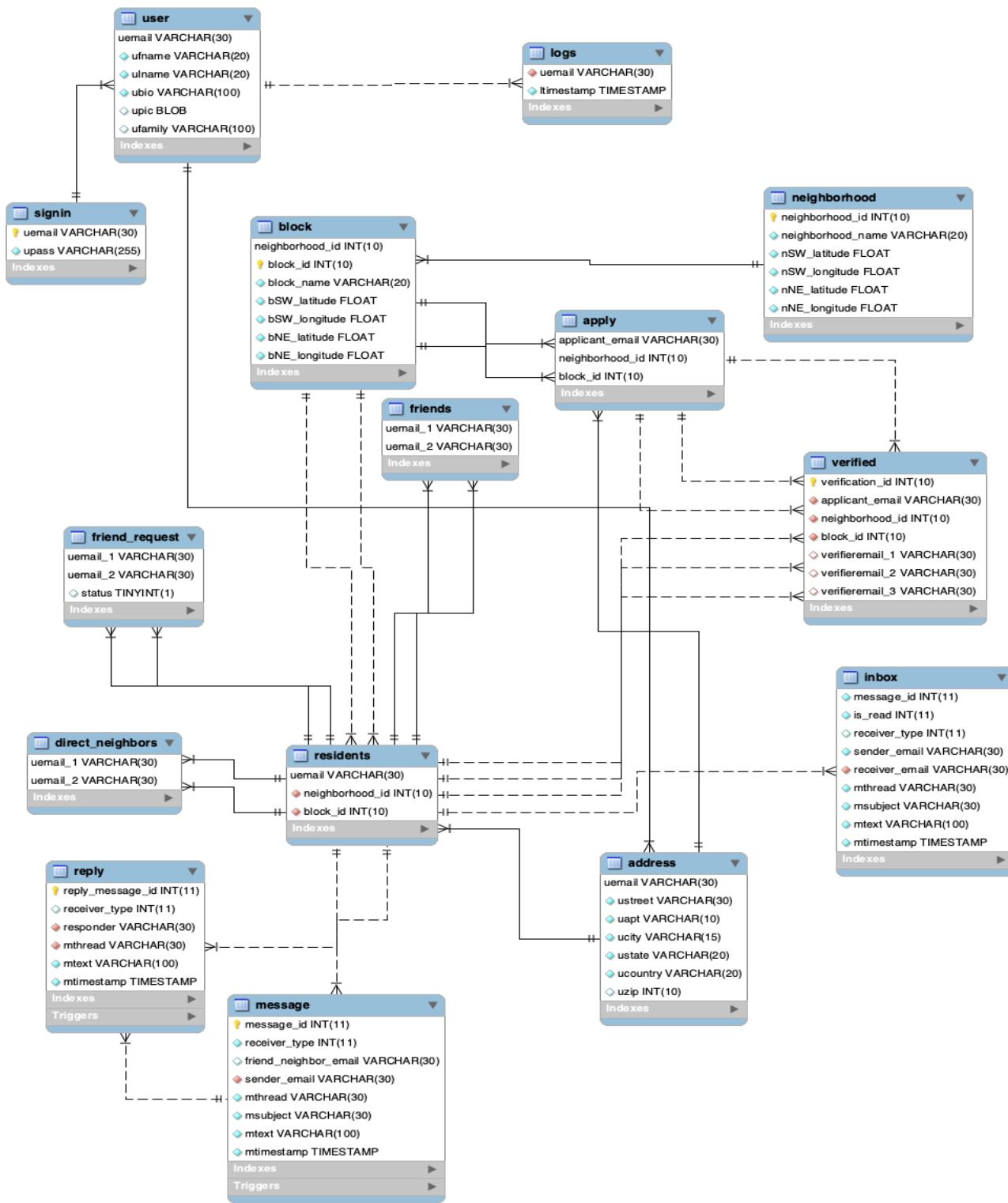
'receiver_email' in 'inbox' references to 'uemail' in 'residents'

'responder' in 'reply' references to 'uemail' in 'residents'

'mthread' in 'reply' references to 'mthread' in 'message'

'uemail' in 'logs' references to 'uemail' in 'user'

ER Diagram



Justification of the Design

Tables

The proposed system allows people to communicate with others in their neighborhood. The system maintains records of users, the locality they belong to (neighborhoods and blocks) and modules on which the users can post, read, and reply to messages. Authentication is provided for this application by allowing only registered users can access the system.

Upon accessing the service, the user needs to sign in by entering their email and password (*uemail* and *upass*) in table **signin**. Upon registering, the database contains information about the user such as the user's email, first name, last name, a short bio, a profile picture, and a list of family members in the table **user**. We use foreign key *uemail* in **user** table to refer to *uemail* in **signin** table because *uemail* is the unique identifier that every user has.

Because the location of the user is so integral to the functionality of the system, we use a separate table **address** to hold the address locations of where the user lives. This stores information such as the user's street, apartment number, city, state, country, and zip code. We use foreign key *uemail* in **address** to refer to *uemail* in **user**. The primary key is *uemail* since it uniquely defines each user's address.

In the database, we store two levels of locality, namely neighborhoods and blocks within neighborhoods. The **neighborhood** table contains information such as a unique neighborhood ID, the neighborhood's name, and the latitude and longitude points. These latitude and longitude values are axis-aligned rectangles which are defined by two corner points (such as latitude of Southwest, longitude of Southwest, latitude of Northeast and longitude of Northeast points). The primary key of this table is *neighborhood_id* which is used to uniquely identify neighborhoods. On the other hand we have a table **block** which contains information such as the neighborhood ID that the block belongs in, the unique block ID, the name of the block, and the latitude and longitude points which are also axis-aligned rectangles which are defined by two corner points. The primary key for this table is (*neighborhood_id*, *block_id*) because a neighborhood may contain several blocks.

Users have the option to apply to join a block using table **apply** which contains information such the user email, the neighborhood, and the block, they wish to apply to. The primary key is *applicant_email*, *neighborhood_id*, *block_id*. The *applicant_email* references **address** table, *neighborhood_id* references **block**, *block_id* references **block** table. They are accepted as a new member to the Blocks table if at least three members approve or all the members in that block approve (if the block has less than 2 members).

We use table **verified** which holds a unique verification ID, the email of the user applying, the neighborhood ID to which the block belongs to, the block ID that the user is applying to, and the email IDs of the three verifiers. A user can only be a member of one block, and is also automatically a member of the neighborhood in which the block is located. As per the problem definition, the names and definitions of blocks and neighborhoods are predefined by the us, so it is not required to add further functionality for the user to update these tables. The *verification_id* is the primary key since it uniquely identify the tuple. The *applicant_email*, *block_id* and *neighborhood_id* reference the table **apply**, and *verifieremail_1*, *verifieremail_2*, and *verifieremail_3* reference **residents** table.

The table **residents** contains the final verified information about which users live in which blocks and neighborhoods in the form of user email and neighborhood ID and the block ID. The primary key is *uemail* since it is used to uniquely identify a user. *uemail* references **address**, *neighborhood_id*, *block_id* references **block**.

Residents can specify two different relationships with other residents; they can be *friends* or *direct neighbors*. Friendship is a relationship between two users and requires both users to accept. Direct neighbors are members who live next door to each other or live in the same apartment/building or very close by. As resident can choose his direct neighbors unilaterally. The two tables **friends** and **direct_neighbors** both contain two sets of user email addresses which refer to the relationship between the two users. The relationship cannot exist without two unique users, so, *uemail_1, uemail_2* is the combined primary key.

The table **friend_request** contains the two email IDs of the users that want to be friends, and an integer value which signifies whether or not the second user has accepted the request. The primary key is *uemail_1, uemail_2* which refers to the **residents** table.

Messaging is a core functionality of this service. A user can send messages to a particular person who is either a friend or a neighbor, or all of the users on their friends list, or to the entire block or the entire neighborhood they are a member of. The **message** table contains the information about each message sent, which includes a unique message ID, the type of receiver (i.e. value = 1 for friends, value = 2 for direct neighbors, value = 3 for block members, value = 4 for neighborhood members), the email of the sender, the name of the thread, the name of the subject, the content of the message, and the timestamp of when the message was sent. *message_id* is the primary key since it is unique to every message stored on the database. The sender's email refers to **residents** table.

Inbox is unique to each user and stores all of the messages received. It includes information such as the unique ID of the message, a status attribute (*is_read*) to show whether or not the message has been read (which is incremented when the user opens the mail for the first time), the email of the receiver, the email of the sender, the timestamp of when the message was sent, the name of the thread, and the name of the subject. The *receiver_email* references **residents** table, *msubject* and *mthread* references **message** table.

The table **reply** is used to store the replies that the recipients of the first message (the first message with a new thread) sent. The table contains a unique reply ID, responder's ID, name of the thread, the content of the message, and the timestamp of when the message was sent. The primary key is *reply_message_id* which uniquely identifies each reply that a user sends. The *responder* refers to **residents** table, *mthread* refers to **Message** table.

The table **Logs** contains the email of the user as well as a log of the recent logins using the procedure *curtime()*. *uemail* in **Logs** references *uemail* in **User**.

Triggers

i) 'send_message'

This trigger will be invoked whenever there is a new entry into the 'message' table. In this trigger, there are 3 conditions on the 'receiver_type' of the 'message' table. If the 'receiver_type' value becomes equal to 1, then it sends message to the friends of the sender by inserts into 'inbox' table having 'receiver_email' equals to the emails of their friends. Right now, the resident has to specify the 'receiver_email' of their friend whom the sender wants to send the message. But when we will create the website, it will have the option to choose among their friends whom they want to send the message to. If the 'receiver_type' value becomes equal to 2, then it sends message to the direct neighbors of the sender by inserts into 'inbox' table having value of the attribute 'receiver_email' equals to the emails of their neighbors. Right now, the resident has to specify the 'receiver_email' of their friend

whom the sender wants to send the message. But when we will create the website, it will have the option to choose among their neighbors whom the sender wants to send the message to.

If the 'receiver_type' value becomes 3, then it sends message to all the block members who live in the same block as sender. A procedure named 'block_members_message' is called which takes values of attributes from the tuple in the 'message' table into its parameters and through cursor inserts into 'inbox' table having value of the attribute 'receiver_email' equals to the emails of residents having 'block_id' is equal to the block_id of the sender.

If the 'receiver_type' value becomes 4, then it sends message to all the hood members who live in the same neighborhood as sender. A procedure named 'hood_members_message' is called which takes values of attributes from the tuple in the 'message' table into its parameters and through cursor inserts into 'inbox' table having value of the attribute 'receiver_email' equals to the emails of residents having 'neighborhood_id' is equal to the neighborhood_id of the sender.

ii) 'reply_to_inbox'

This trigger will be invoked whenever someone replies to a message using the thread. The reply will go into the 'reply' table and this trigger will be invoked which basically sends the reply to the inbox of all the members whom the earlier message was sent. This trigger calls a procedure named as 'reply_to_inbox' which takes the tuples from 'reply' into its parameters. Then, through the SQL query the email list of those people is found using 'mthread' whom the earlier message was sent. Then through the cursor, the tuple in 'reply' is inserted into the 'inbox' having value of the attribute 'receiver_email' equal to the emails whom the earlier message was sent (including the original sender's email too).

Procedures

i) 'block_members_message'

This stored procedure is called through the trigger 'send_message' when a resident wants to send a message to all of the residents having same 'block_id' as that of sender in the 'residents' table by inserting the value of attribute 'receiver_type' = 3. Through the SQL query, the list of emails of all the residents having 'block_id' equals to the sender's 'block_id' is found, and then through a cursor, the message is inserted into the 'inbox' table having 'receiver_email' attribute equal to the list of emails, we just found.

ii) 'hood_members_message'

This stored procedure is called through the trigger 'send_message' when a resident wants to send a message to all of the residents having same 'block_id' as that of sender in the 'residents' table by inserting the value of attribute 'receiver_type' = 4. Through the SQL query, the list of emails of all the residents having 'neighborhood_id' equals to the sender's 'neighborhood_id' is found, and then through a cursor, the message is inserted into the 'inbox' table having 'receiver_email' attribute equal to the list of emails, we just found.

iii) 'reply_to_inbox'

This stored procedure is called through the trigger 'reply_to_message' when a resident wants to reply to a message sent to a group of residents earlier. Through the SQL query, the list of emails of all the residents having same value of the 'mthread' as of the tuple in 'reply' table is found (including the person who the first message with that thread). And then through a cursor, the reply is inserted into the 'inbox' table having 'receiver_email' attribute equal to the list of emails, we just found.

Choice on Functionality

We chose to treat messages like emails. If a resident moves to a new block, they would still have the access to their old messages (the ones they sent as well as those they received). A resident can view his/her inbox which will have all the past messages, even if he/she moves to a new block. After moving to a new block, that resident would get messages from the new block members and new neighborhood members. As we are not making any SQL query to delete the messages from a resident's inbox when that resident moves to a new block, therefore, the resident would always have access to his past messages.

When the user wants to start a new thread, they start by creating a message in the **message** table with the name of the thread. A trigger is used to identify the who are the receivers of this message going to be i.e. 1 being friends, 2 being direct neighbors, 3 being block members, and 4 being neighborhood members. Based on the integer value of the receiver type, the message is inserted into the **Inbox** of the receiver. When any of the receivers want to reply, the message is inserted from the **reply** table to the inbox of all the recipients of the original message. An important assumption we make is that there are no two threads under the same name.

Planned Interface

The first page is going to be the sign in page where a user will have the option to sign up to create a new account and as well as to login into the already created account. If the user is a new user that just signed up, that user will be directed to fill in his information such as first name, last name, bio, pic, family. After that the user will be directed to fill in the address, and then he/she will have to apply to a block in a neighborhood. If the request is accepted by at least 3 members or all the members (if less than 2), the user will have an entry into 'residents' table. After that the user can send friend requests to the residents in same neighborhood. If the request is accepted, the user will have a friend in the 'friends' table. The user will also have the option to add someone as direct neighbor which will add a tuple in the 'direct_neighbor' table. A user will have the option to send message to any or all of their friends, any or all of their direct neighbors, all of residents living in the same block, or all the residents living in the same neighborhood as the user. A user can reply to any message he/she got earlier, or any other resident can reply to the user's message. If the user has already a fully setup account, they will see the new messages, new friend requests since the last time they visited the website. This will be done using the last entry in the 'logs' table.

Queries

(1) Joining:

For joining, there will be simple insert statements that will populate data in 'signin', 'user', 'address' tables. For example:

```
insert into `signin`(`uemail`,`upass`) values
```

```
('alex.gordon97@gmail.com', 'pass@12345');
```

Inserted Sample data:

uemail	upass
alex.gordon97@gmail.com	pass@12345
elizabethwarren.hbs@yahoo.com	iamfor2020@win
iamcooper260@gmail.com	qazwsxedcqaz#90120
miguel.hernandez@gmail.com	ownerrealestate#082
mike2020@bloomberg.org	bloomingguy#768
rahulgarg156989@gmail.com	rahulhere4u@678
rajeshsharma1975@gmail.com	rumplustelskin@123
shrutika19newyork@gmail.com	realnewyorker129
xiang.zhao956@aol.com	xiang34@prchina
xinjiangmao@rocketmail.com	chinain45@newyork
NULL	NULL

```
insert into `user`(`uemail`,`ufname`,`ulname`,`ubio`,`upic`,`ufamily`) values
```

```
('alex.gordon97@gmail.com', 'Alex', 'Gordon', 'Hi there, I am from New York. I work as a software engineer',  
NULL, 'We a family of 5- Alex, Mike, Joey, Melania, Jordan');
```

Inserted Sample data:

uemail	ufname	ulname	ubio	upic	ufamily
alex.gordon97@gmail.com	Alex	Gordon	Hi there, I am from New York. I work as a software engineer	NULL	We a family of 5- Alex, Mike, Joey, Melania, Jordan
elizabethwarren.hbs@yahoo.com	Elizabeth	Warren	Former Harvard Law School Professor, 2020 presidential candidate	NULL	
iamcooper260@gmail.com	Bredly	Cooper	Hollywood Actor	NULL	Everyone knows my family
miguel.hernandez@gmail.com	Miguel	Hernandez	Owner at Realty props	NULL	MY family includes my mom Helena, my wife Juliana, my son Jack, my daughter Je...
mike2020@bloomberg.org	Alex	Gordon	I am the owner of Bloomberg LLC and I am also the 2020 president...	NULL	My family- Diana(wife), Georgina Bloomberg(daughter), Emma Bloomberg(daughter)
rahulgarg156989@gmail.com	Rahul	Garg	Computer Science Student at NYU Tandon	NULL	my mom- Rashmi, my dad- Rajkumar, my sister- Sheena, my grandfather- Chandra
rajeshsharma1975@gmail.com	Rajesh	Sharma	Professor at NYU	NULL	Family: Swati(my wife), my daughter- Helena
shrutika19newyork@gmail.com	Shrutika	Gupta	Receptionist at Marriot	NULL	
xiang.zhao956@aol.com	Xiang	Zhao	Student at NYU Tandon	NULL	Jianjing- My mom, Xi- my dad, shiniyang- my sister
xinjiangmao@rocketmail.com	Xinjiang	Mao	#Basket ball player	NULL	3 family members- my mom Sdhiyong, my dad Zianing, my sister Shizuka
NULL	NULL	NULL	NULL	NULL	NULL

```
insert into `address`(`uemail`,`ustreet`,`uapt`,`ucity`,`ustate`,`ucountry`,`uzip`) values
```

```
('alex.gordon97@gmail.com', '65th Street', '456', 'Brooklyn', 'New York', 'USA', 11201);
```

Inserted Sample data:

uemail	ustreet	uapt	ucity	ustate	ucountry	uzip
alex.gordon97@gmail.com	65th Street	456	Brooklyn	New York	USA	11201
elizabethwarren.hbs@yahoo.com	45th Street	1223	Cambridge	Massachusetts	USA	2138
iamcooper260@gmail.com	1st Street	788	Los Angeles	California	USA	45324
miguel.hernandez@gmail.com	54th Street	309	Brooklyn	New York	USA	11209
mike2020@bloomberg.org	74th Street	2	Manhattan	New York	USA	11001
rahulgarg156989@gmail.com	88th Street	34	Los Angeles	California	USA	45268
rajeshsharma1975@gmail.com	65th Street	457	Brooklyn	New York	USA	11201
shrutika19newyork@gmail.com	8th Street	2121	Queens	New York	USA	11230
xiang.zhao956@aol.com	65th Street	454	Brooklyn	New York	USA	11201
xinjiangmao@rocketmail.com	23th Street	22	Los Angeles	California	USA	45082
NULL	NULL	NULL	NULL	NULL	NULL	NULL

To apply to become a member of a block, there will be simple insert statement that will populate data in the 'apply' table. For example:

```
insert into `apply`(`applicant_email`, `neighborhood_id`, `block_id`) values
('alex.gordon97@gmail.com', 1, 1);
```

Inserted sample data:

applicant_email	neighborhood...	block_id
alex.gordon97@gmail.com	1	1
elizabethwarren.hbs@yahoo.com	1	2
iamcooper260@gmail.com	1	1
mike2020@bloomberg.org	1	1
rajeshsharma1975@gmail.com	2	4
miguel.hernandez@gmail.com	3	6
rahulgarg156989@gmail.com	3	6
shrutika19newyork@gmail.com	3	6
xiang.zhao956@aol.com	3	6
xinjiangmao@rocketmail.com	3	6
NULL	NULL	NULL

To edit the user's profile, there will be a simple update statement. For example:

```
update user
set ubio = 'Hi there, I am from New York. I am now financial consultant',
ufamily = 'Alex, Mike, Joey, Melania, Jordan'
where uemail = 'alex.gordon97@gmail.com';
```

Updated Sample data:

uemail	ufname	ulname	ubio	upic	ufamily
alex.gordon97@gmail.com	Alex	Gordon	Hi there, I am from New York. I am now financial consultant	NULL	Alex, Mike, Joey, Melania, Jordan
elizabethwarren.hbs@yahoo.com	Elizabeth	Warren	Former Harvard Law School Professor, 2020 presidential candidate	NULL	
iamcooper260@gmail.com	Bredly	Cooper	Hollywood Actor	NULL	Everyone knows my family
miguel.hernandez@gmail.com	Miguel	Hernandez	Owner at Realty props	NULL	MY family includes my mom Helena, my wife Juliana, my son Jack, my daughter Je...
mike2020@bloomberg.org	Alex	Gordon	I am the owner of Bloomberg LLC and I am also the 2020 president...	NULL	My family- Diana(wife), Georgina Bloomberg(daughter), Emma Bloomberg(daughter)
rahulgarg156989@gmail.com	Rahul	Garg	Computer Science Student at NYU Tandon	NULL	my mom- Rashmi, my dad- Rajkumar, my sister- Sheena, my grandfather- Chandra
rajeshsharma1975@gmail.com	Rajesh	Sharma	Professor at NYU	NULL	Family: Swati(my wife), my daughter- Helena
shrutika19newyork@gmail.com	Shrutika	Gupta	Receptionist at Marriot	NULL	
xiang.zhao956@aol.com	Xiang	Zhao	Student at NYU Tandon	NULL	Jianjing- My mom, Xi- my dad, shiniyang- my sister
xinjiangmao@rocketmail.com	Xinjiang	Mao	#Basket ball player	NULL	3 family members- my mom Sdhiyong, my dad Zianing, my sister Shizuka
NULL	NULL	NULL	NULL	NULL	NULL

(2) Content Posting:

First of all, here are the populated tables for 'friends', 'direct neighbors', and 'residents'.

The 'friends' table:

uemail_1	uemail_2
alex.gordon97@gmail.com	elizabethwarren.hbs@yahoo.com
mike2020@bloomberg.org	elizabethwarren.hbs@yahoo.com
alex.gordon97@gmail.com	mike2020@bloomberg.org
NULL	NULL

The 'direct_neighbors' table:

uemail_1	uemail_2
rajeshsharma1975@gmail.com	alex.gordon97@gmail.com
alex.gordon97@gmail.com	rajeshsharma1975@gmail.com
alex.gordon97@gmail.com	xiang.zhao956@aol.com
rajeshsharma1975@gmail.com	xiang.zhao956@aol.com
NULL	NULL

The 'residents' table:

uemail	neighborhood...	block_id
alex.gordon97@gmail.com	1	1
elizabethwarren.hbs@yahoo.com	1	2
miguel.hernandez@gmail.com	3	6
mike2020@bloomberg.org	1	1
rajeshsharma1975@gmail.com	2	4
shrutika19newyork@gmail.com	3	6
xiang.zhao956@aol.com	3	6
NULL	NULL	NULL

According to our choice of functionality of the system, when first message is written with a new thread, it will go into the 'message' table and will be sent to the 'inbox' of the receivers that the sender intended to send to.

Testing case for 'receiver_type' = 4

Say, Shrutika wants to send a new message to his neighborhood members, then there will be an insert statement as follows that will populate the message table:

insert into `message`(`receiver_type`, `sender_email`, `mthread`, `msubject`, `mtext`) values

(4, 'shrutika19newyork@gmail.com', 'fashion', 'cheapshow', 'cheap tickets available for the fashion show at cheapshowtickets.com');

message_id	receiver_type	friend_neighbor_em...	sender_email	mthread	msubject	mtext	mtimestamp
2	4	NULL	shrutika19newyork@gmail.com	fashion	cheapshow	cheap tickets available for the fashion show at c...	2019-12-05 18:50:52
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Here the 'receiver_type' = 4 signifies that Shrutika wants to send to his block members. As we have a trigger, that will check the value of 'receiver_type', here = 4, it will call the procedure 'hood_members_message' that

will insert that message into the 'inbox' table having receiver_email equals to emails of the residents having same 'neighborhood_id' as that of the sender.

After running this procedure, the 'inbox' table would be:

message_id	is_read	sender_email	receiver_email	mthread	msubject	mtext	mtimestamp
2	0	shrutika19newyork@gmail.com	miguel.hernandez@gmail.com	fashion	cheapshow	cheap tickets available for the fashion show at c...	2019-12-05 18:50:52
2	0	shrutika19newyork@gmail.com	xiang.zhao956@aol.com	fashion	cheapshow	cheap tickets available for the fashion show at c...	2019-12-05 18:50:52

Now, say, if Miguel replies to Shrutika's message, then Miguel's reply will go into the 'reply' table through a simple insert statement.

Insert into 'reply'('responder', 'mthread', 'mtext') values

('miguel.hernandez@gmail.com', 'fashion', 'Hey everyone, I can get you all stuffs at cheaper price than sale');

reply_message_id	responder	mthread	mtext	mtimestamp
1	miguel.hernandez@gmail.com	fashion	Hey everyone, I can get you all stuffs at cheape...	2019-12-05 18:55:27
NULL	NULL	NULL	NULL	NULL

Now, a trigger on the table 'reply_to_inbox' will run that will call a procedure 'reply_to_inbox'. The procedure then will insert the reply into the 'inbox' table having value of attribute 'receiver_email' equal to those emails who the earlier message was sent and the original sender too.

After running this procedure, the 'inbox' table would be:

message_id	is_read	sender_email	receiver_email	mthread	msubject	mtext	mtimestamp
2	0	shrutika19newyork@gmail.com	miguel.hernandez@gmail.com	fashion	cheapshow	cheap tickets available for the fashion show at c...	2019-12-05 18:50:52
2	0	shrutika19newyork@gmail.com	xiang.zhao956@aol.com	fashion	cheapshow	cheap tickets available for the fashion show at c...	2019-12-05 18:50:52
1	0	miguel.hernandez@gmail.com	xiang.zhao956@aol.com	fashion	cheapshow	Hey everyone, I can get you all stuffs at cheape...	2019-12-05 18:55:27
1	0	miguel.hernandez@gmail.com	shrutika19newyork@gmail.com	fashion	cheapshow	Hey everyone, I can get you all stuffs at cheape...	2019-12-05 18:55:27

Now, if the sender wants to sent the message to some/all of his/her friends or some/all of his/her direct neighbors, then the sender has to specify the email ids of the ones, whom the user wants to send message. When we will design the webpage, it will have the option to choose from his friends/neighbors but as of now, the trigger by checking the 'receiver_type' value equal to 1 or 2, inserts into the inbox of the recipients.

Testing the sample case for 'receiver_type' = 1 i.e. friends

insert into 'message'('receiver_type', 'sender_email', 'friend_neighbor_email', 'mthread', 'msubject', 'mtext') values

(1, 'mike2020@bloomberg.org', 'alex.gordon97@gmail.com', 'regarding sale', 'chair sale', 'Hey, I am selling my Ikea chair, are you interested to buy? It is new.');

The message table after the above insertion statement:

message_id	receiver_type	friend_neighbor_email	sender_email	mthread	msubject	mtext	mtimestamp
2	4	NULL	shrutika19newyork@gmail.com	fashion	cheapshow	cheap tickets available for the fashion show at c...	2019-12-05 18:50:52
3	1	alex.gordon97@gmail.com	mike2020@bloomberg.org	regarding sale	chair sale	Hey, I am selling my Ikea chair, are you interest...	2019-12-05 19:12:06
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

The 'inbox' table after this insertion statement:

message_id	is_read	sender_email	receiver_email	mthread	msubject	mtext	mtimestamp
2	0	shrutika19newyork@gmail.com	miguel.hernandez@gmail.com	fashion	cheapshow	cheap tickets available for the fashion show at c...	2019-12-05 18:50:52
2	0	shrutika19newyork@gmail.com	xiang.zhao956@aol.com	fashion	cheapshow	cheap tickets available for the fashion show at c...	2019-12-05 18:50:52
1	0	miguel.hernandez@gmail.com	xiang.zhao956@aol.com	fashion	cheapshow	Hey everyone, I can get you all stuffs at cheape...	2019-12-05 18:55:27
1	0	miguel.hernandez@gmail.com	shrutika19newyork@gmail.com	fashion	cheapshow	Hey everyone, I can get you all stuffs at cheape...	2019-12-05 18:55:27
3	0	mike2020@bloomberg.org	alex.gordon97@gmail.com	regarding sale	chair sale	Hey, I am selling my Ikea chair, are you interest...	2019-12-05 19:12:06

If the sender wants to send the message to his/her block members, then the value of 'receiver_type' attribute in the trigger would be 3 and then the trigger will the procedure 'block_members_message' which will insert the message into the 'inbox' having value of the attribute 'receiver_email' equals to the emails of the residents having 'block_id' same as that of the sender.

Testing the sample case for 'receiver_type' = 3 i.e. Block Members

insert into `message`(`receiver_type`, `sender_email`, `mthread`, `msubject`, `mtext`) values

(3, 'alex.gordon97@gmail.com', 'salenearby', 'sale at BM', 'Hi everyone, there is very good sale at the nearby banker market, hope it would be beneficial');

The message table after the above insertion statement:

message_id	receiver_type	friend_neighbor_email	sender_email	mthread	msubject	mtext	mtimestamp
2	4	NULL	shrutika19newyork@gmail.com	fashion	cheapshow	cheap tickets available for the fashion show at c...	2019-12-05 18:50:52
3	1	alex.gordon97@gmail.com	mike2020@bloomberg.org	regarding sale	chair sale	Hey, I am selling my Ikea chair, are you interest...	2019-12-05 19:12:06
4	3	NULL	alex.gordon97@gmail.com	salenearby	sale at BM	Hi everyone, there is very good sale at the near...	2019-12-05 19:15:19
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

The 'inbox' table after this insertion statement:

message_id	is_read	sender_email	receiver_email	mthread	msubject	mtext	mtimestamp
2	0	shrutika19newyork@gmail.com	miguel.hernandez@gmail.com	fashion	cheapshow	cheap tickets available for the fashion show at c...	2019-12-05 18:50:52
2	0	shrutika19newyork@gmail.com	xiang.zhao956@aol.com	fashion	cheapshow	cheap tickets available for the fashion show at c...	2019-12-05 18:50:52
1	0	miguel.hernandez@gmail.com	xiang.zhao956@aol.com	fashion	cheapshow	Hey everyone, I can get you all stuffs at cheape...	2019-12-05 18:55:27
1	0	miguel.hernandez@gmail.com	shrutika19newyork@gmail.com	fashion	cheapshow	Hey everyone, I can get you all stuffs at cheape...	2019-12-05 18:55:27
3	0	mike2020@bloomberg.org	alex.gordon97@gmail.com	regarding sale	chair sale	Hey, I am selling my Ikea chair, are you interest...	2019-12-05 19:12:06
4	0	alex.gordon97@gmail.com	mike2020@bloomberg.org	salenearby	sale at BM	Hi everyone, there is very good sale at the near...	2019-12-05 19:15:19

(3) Friendship:

To send someone a friend request, there is simple insert statement that will insert into 'friend_request' table. The status attribute will remain 0 until the friend request receiver i.e. uemail_2 accepts the friend request. When the status becomes equal to 1, then that friends tuple will be inserted into the 'friends' table.

insert into friend_request(uemail_1, uemail_2) values

('alex.gordon97@gmail.com', 'mike2020@bloomberg.org');

Inserted Sample friend requests:

uemail_1	uemail_2	status
alex.gordon97@gmail.com	elizabethwarren.hbs@yahoo.com	0
alex.gordon97@gmail.com	mike2020@bloomberg.org	0
mike2020@bloomberg.org	elizabethwarren.hbs@yahoo.com	0
NULL	NULL	NULL

Right now, the friends table is blank because no request has been accepted yet.

Now, say Elizabeth Warren accepts the friend request of Alex Gordon, then the query that will run

update friend_request

set status = 1 where uemail_2 = 'elizabethwarren.hbs@yahoo.com' and uemail_1 = 'alex.gordon97@gmail.com';

Updated sample friend requests:

uemail_1	uemail_2	status
alex.gordon97@gmail.com	elizabethwarren.hbs@yahoo.com	1
alex.gordon97@gmail.com	mike2020@bloomberg.org	1
mike2020@bloomberg.org	elizabethwarren.hbs@yahoo.com	1
NULL	NULL	NULL

The 'friends' table will be populated only when the uemail_2 i.e. friend request receiver updates the status attribute from 0 to 1

Query for that will be a simple insert statement:

insert into `friends`(`uemail_1`, `uemail_2`) values

('alex.gordon97@gmail.com', 'elizabethwarren.hbs@yahoo.com');

Updated friends table:

uemail_1	uemail_2
alex.gordon97@gmail.com	elizabethwarren.hbs@yahoo.com
mike2020@bloomberg.org	elizabethwarren.hbs@yahoo.com
alex.gordon97@gmail.com	mike2020@bloomberg.org
NULL	NULL

Now, say Alex wants to see his friends list. Then, the following query will run:

select uemail_2 from friends

where uemail_1 = 'alex.gordon97@gmail.com';

Alex' friends list:

uemail_2
elizabethwarren.hbs@yahoo.com
mike2020@bloomberg.org

If a resident wants to see his/her direct neighbors list, he/she must have added the other resident as his/neighbor first by inserting data through insert statement into the 'direct_neighbors' table.

Say, we have already inserted data, and now Alex wants to see his direct neighbors' list. A simple SQL query for that is as follows:

```
select uemail_2 from direct_neighbors
where uemail_1 = 'alex.gordon97@gmail.com';
```

Alex's neighbors list:

uemail_2
rajeshsharma1975@gmail.com
xiang.zhao956@aol.com

```
insert into `friends`(`uemail_1`, `uemail_2`) values
('alex.gordon97@gmail.com', 'mike2020@bloomberg.org'),
('alex.gordon97@gmail.com', 'elizabethwarren.hbs@yahoo.com'),
('mike2020@bloomberg.org', 'elizabethwarren.hbs@yahoo.com');
```

uemail_1	uemail_2
alex.gordon97@gmail.com	elizabethwarren.hbs@yahoo.com
mike2020@bloomberg.org	elizabethwarren.hbs@yahoo.com
alex.gordon97@gmail.com	mike2020@bloomberg.org
NULL	NULL

```
insert into `direct_neighbors`(`uemail_1`, `uemail_2`) values
('alex.gordon97@gmail.com', 'rajeshsharma1975@gmail.com'),
('alex.gordon97@gmail.com', 'xiang.zhao956@aol.com'),
('rajeshsharma1975@gmail.com', 'alex.gordon97@gmail.com'),
('rajeshsharma1975@gmail.com', 'xiang.zhao956@aol.com');
```

uemail_1	uemail_2	
rajeshsharma1975@gmail.com	alex.gordon97@gmail.com	
alex.gordon97@gmail.com	rajeshsharma1975@gmail.com	
alex.gordon97@gmail.com	xiang.zhao956@aol.com	
rajeshsharma1975@gmail.com	xiang.zhao956@aol.com	
NULL	NULL	

(4) Browse and Search Messages:

Say, Shrutika logged into her account and then sent a message to her block members and then she logged out. Before she logged in again, there were 2 replies that she got into her inbox on the message she sent when she was online last time. When she logs again, she sees 2 replies on her message.

Now, data in the 'logs' table when she logged to send the message:

uemail	ltimestamp
miguel.hernandez@gmail.com	2019-12-05 21:58:06
mike2020@bloomberg.org	2019-12-05 21:58:25
alex.gordon97@gmail.com	2019-12-05 21:58:42
shrutika19newyork@gmail.com	2019-12-05 21:58:55

Then she sent a message, the data in the message table:

message_id	receiver_type	friend_neighbor_email	sender_email	mthread	msubject	mtext	mtimestamp
2	4	NULL	shrutika19newyork@gmail.com	fashion	cheapshow	cheap tickets available for the fashion show at c...	2019-12-05 18:50:52
3	1	alex.gordon97@gmail.com	mike2020@bloomberg.org	regarding sale	chair sale	Hey, I am selling my Ikea chair, are you interest...	2019-12-05 19:12:06
4	3	NULL	alex.gordon97@gmail.com	salenearby	sale at BM	Hi everyone, there is very good sale at the near...	2019-12-05 19:15:19
5	3	NULL	alex.gordon97@gmail.com	greatsale	selling table	Hey, I am selling my wooden table, if anyone is i...	2019-12-05 21:43:16
7	3	NULL	shrutika19newyork@gmail.com	comedyshow	comedy at Gotham	Hey, there is a comedy show at Gotham Comed...	2019-12-05 22:02:59
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

The data in the 'inbox' table having last 2 entries shows that the message has gone into the 'inbox' of her all the block members:

message_id	is_read	sender_email	receiver_email	mthread	msubject	mtext	mtimestamp
2	0	shrutika19newyork@gmail.com	miguel.hernandez@gmail.com	fashion	cheapshow	cheap tickets available for the fashion show at c...	2019-12-05 18:50:52
2	0	shrutika19newyork@gmail.com	xiang.zhao956@aol.com	fashion	cheapshow	cheap tickets available for the fashion show at c...	2019-12-05 18:50:52
1	0	miguel.hernandez@gmail.com	xiang.zhao956@aol.com	fashion	cheapshow	Hey everyone, I can get you all stuffs at cheape...	2019-12-05 18:55:27
1	0	miguel.hernandez@gmail.com	shrutika19newyork@gmail.com	fashion	cheapshow	Hey everyone, I can get you all stuffs at cheape...	2019-12-05 18:55:27
3	0	mike2020@bloomberg.org	alex.gordon97@gmail.com	regarding sale	chair sale	Hey, I am selling my Ikea chair, are you interest...	2019-12-05 19:12:06
4	0	alex.gordon97@gmail.com	mike2020@bloomberg.org	salenearby	sale at BM	Hi everyone, there is very good sale at the near...	2019-12-05 19:15:19
2	0	shrutika19newyork@gmail.com	miguel.hernandez@gmail.com	fashion	cheapshow	I checked everywhere, there is no cheaper price	2019-12-05 20:19:08
2	0	shrutika19newyork@gmail.com	xiang.zhao956@aol.com	fashion	cheapshow	I checked everywhere, there is no cheaper price	2019-12-05 20:19:08
2	0	shrutika19newyork@gmail.com	shrutika19newyork@gmail.com	fashion	cheapshow	I checked everywhere, there is no cheaper price	2019-12-05 20:19:08
7	0	shrutika19newyork@gmail.com	miguel.hernandez@gmail.com	comedyshow	comedy at Gotham	Hey, there is a comedy show at Gotham Comed...	2019-12-05 22:02:59
7	0	shrutika19newyork@gmail.com	xiang.zhao956@aol.com	comedyshow	comedy at Gotham	Hey, there is a comedy show at Gotham Comed...	2019-12-05 22:02:59

Then she log out of her account.

Now, Miguel and Xiang, her 2 block members reply on her message:

The data in the reply table (last 2 entries shows the 2 replies):

reply_message_id	responder	mthread	mtext	mtimestamp
1	miguel.hernandez@gmail.com	fashion	Hey everyone, I can get you all stuffs at cheape...	2019-12-05 18:55:27
2	shrutika19newyork@gmail.com	fashion	I checked everywhere, there is no cheaper price	2019-12-05 20:19:08
3	mike2020@bloomberg.org	greatsale	I am interested, please tell the price	2019-12-05 21:44:17
4	miguel.hernandez@gmail.com	comedyshow	I am interested in going. What is the price, you...	2019-12-05 22:03:53
5	xiang.zhao956@aol.com	comedyshow	I might want to know the price first and why are...	2019-12-05 22:04:23
NULL	NULL	NULL	NULL	NULL

Now the data in the 'inbox' table:

message_id	is_read	sender_email	receiver_email	mthread	msubject	mtext	mtimestamp
2	0	shrutika19newyork@gmail.com	miguel.hernandez@gmail.com	fashion	cheapshow	cheap tickets available for the fashion show at c...	2019-12-05 18:50:52
2	0	shrutika19newyork@gmail.com	xiang.zhao956@aol.com	fashion	cheapshow	cheap tickets available for the fashion show at c...	2019-12-05 18:50:52
1	0	miguel.hernandez@gmail.com	xiang.zhao956@aol.com	fashion	cheapshow	Hey everyone, I can get you all stuffs at cheape...	2019-12-05 18:55:27
1	0	miguel.hernandez@gmail.com	shrutika19newyork@gmail.com	fashion	cheapshow	Hey everyone, I can get you all stuffs at cheape...	2019-12-05 18:55:27
3	0	mike2020@bloomberg.org	alex.gordon97@gmail.com	regarding sale	chair sale	Hey, I am selling my Ikea chair, are you interest...	2019-12-05 19:12:06
4	0	alex.gordon97@gmail.com	mike2020@bloomberg.org	salenearby	sale at BM	Hi everyone, there is very good sale at the near...	2019-12-05 19:15:19
2	0	shrutika19newyork@gmail.com	miguel.hernandez@gmail.com	fashion	cheapshow	I checked everywhere, there is no cheaper price	2019-12-05 20:19:08
2	0	shrutika19newyork@gmail.com	xiang.zhao956@aol.com	fashion	cheapshow	I checked everywhere, there is no cheaper price	2019-12-05 20:19:08
2	0	shrutika19newyork@gmail.com	shrutika19newyork@gmail.com	fashion	cheapshow	I checked everywhere, there is no cheaper price	2019-12-05 20:19:08
7	0	shrutika19newyork@gmail.com	miguel.hernandez@gmail.com	comedyshow	comedy at Gotham	Hey, there is a comedy show at Gotham Comed...	2019-12-05 22:02:59
7	0	shrutika19newyork@gmail.com	xiang.zhao956@aol.com	comedyshow	comedy at Gotham	Hey, there is a comedy show at Gotham Comed...	2019-12-05 22:02:59
4	0	miguel.hernandez@gmail.com	xiang.zhao956@aol.com	comedyshow	comedy at Gotham	I am interested in going. What is the price, you...	2019-12-05 22:03:53
4	0	miguel.hernandez@gmail.com	shrutika19newyork@gmail.com	comedyshow	comedy at Gotham	I am interested in going. What is the price, you...	2019-12-05 22:03:53
5	0	xiang.zhao956@aol.com	miguel.hernandez@gmail.com	comedyshow	comedy at Gotham	I might want to know the price first and why are...	2019-12-05 22:04:23
5	0	xiang.zhao956@aol.com	shrutika19newyork@gmail.com	comedyshow	comedy at Gotham	I might want to know the price first and why are...	2019-12-05 22:04:23

As the reply also go the block members whom the earlier message was sent, therefore, there are 4 new entries in the last.

Now, Shrutika login again. Data in the 'logs' table (see last entry):

uemail	ltimestamp
miguel.hernandez@gmail.com	2019-12-05 21:58:06
mike2020@bloomberg.org	2019-12-05 21:58:25
alex.gordon97@gmail.com	2019-12-05 21:58:42
shrutika19newyork@gmail.com	2019-12-05 21:58:55
shrutika19newyork@gmail.com	2019-12-05 22:05:07

sender_email	mthread	msubject	mtext	mtimestamp
miguel.hernandez@gmail.com	comedyshow	comedy at Gotham	I am interested in going. What is the price, you...	2019-12-05 22:03:53
xiang.zhao956@aol.com	comedyshow	comedy at Gotham	I might want to know the price first and why are...	2019-12-05 22:04:23

sender_email	receiver_email	mthread	msubject	mtext	mtimestamp
alex.gordon97@gmail.com	mike2020@bloomberg.org	salenearby	sale at BM	Hi everyone, there is very good sale at the near...	2019-12-05 19:15:19
mike2020@bloomberg.org	alex.gordon97@gmail.com	regarding sale	chair sale	Hey, I am selling my Ikea chair, are you interest...	2019-12-05 19:12:06

Say, Shrutika wants to see all of her messages that she sent or received containing the word say 'cheap'. According to our design, a every message will be in the 'inbox' table that a resident has sent or received in the past. Therefore, in order to see all her messages across all the feeds, an SQL query will be:

`select sender_email, receiver_email, mthread, msubject, mtext, mtimestamp from inbox where sender_email = 'shrutika19newyork@gmail.com' or receiver_email = 'shrutika19newyork@gmail.com';`

sender_email	receiver_email	mthread	msubject	mtext	mtimestamp
shrutika19newyork@gmail.com	miguel.hernandez@gmail.com	fashion	cheapshow	cheap tickets available for the fashion show at c...	2019-12-05 18:50:52
shrutika19newyork@gmail.com	xiang.zhao956@aol.com	fashion	cheapshow	cheap tickets available for the fashion show at c...	2019-12-05 18:50:52
miguel.hernandez@gmail.com	shrutika19newyork@gmail.com	fashion	cheapshow	Hey everyone, I can get you all stuffs at cheape...	2019-12-05 18:55:27
shrutika19newyork@gmail.com	miguel.hernandez@gmail.com	fashion	cheapshow	I checked everywhere, there is no cheaper price	2019-12-05 20:19:08
shrutika19newyork@gmail.com	xiang.zhao956@aol.com	fashion	cheapshow	I checked everywhere, there is no cheaper price	2019-12-05 20:19:08

Project Part 2

Front End Design

Design and Functionality

We designed our fully functional website using HTML and PHP as the server-side scripting language. The database we used is XAMPP. Our website is functional at its core and we have ensured that many edge cases have been tested. Furthermore, we have ensured that we have several security mechanisms in place.

Security by Design Principles

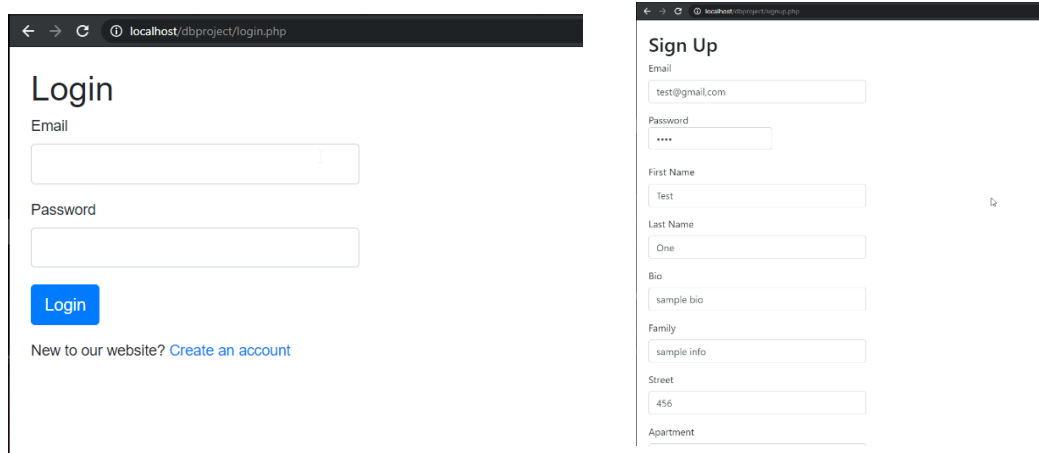
Password hashing is used to prevent an unauthorized user from gaining access to a database and viewing the passwords in plaintext. Instead, the passwords are hashed via a one way transform when a user signs up with their new password. We implemented password hashing using the built in bcrypt hashing algorithm in PHP. When logging the typed password is hashed and compared to the stored hash password.

We prevent SQL injection by sanitizing and filtering through the use of prepared statements and binding parameters, meaning the query and the data are sent to the database separately.

XSS and CSRF are prevented by using htmlspecialchars and making sure to only allow users to login and destroy their session when they log out.

Flow of Control

The user is greeted with a login page where they can log in using their verified credentials or where they can sign up.



The image displays two side-by-side screenshots of a web application interface. The left screenshot shows the 'Login' page, which has a dark header bar with navigation icons and the URL 'localhost/dbproject/login.php'. The main content area is white and features a 'Login' title, an 'Email' input field, a 'Password' input field, and a blue 'Login' button. Below the button is a link that says 'New to our website? [Create an account](#)'. The right screenshot shows the 'Sign Up' page, also with the same dark header. The main content area is white and contains a 'Sign Up' title followed by several input fields: 'Email' (with 'test@gmail.com' entered), 'Password' (with four asterisks), 'First Name' (with 'Test' entered), 'Last Name' (with 'One' entered), 'Bio' (with 'sample bio' entered), 'Family' (with 'sample info' entered), 'Street' (with '456' entered), and 'Apartment'. A mouse cursor is visible over the 'First Name' field.

If the user is a new member, they can sign up by entering all of their profile data. After successfully providing the website with their information, they must apply to a block in the neighborhood that they live in. They must then wait for three block members to verify their status or they will not be allowed to log in. If a new member tries to login, they will be unable to login even if they are partially verified by one or two block members.

The first screenshot shows the 'localhost/dbproject/apply.php' page. It features a text input for 'test@gmail.com' and a row of neighborhood buttons: Verazzano, Bell Church, Golden Market, Chinatown, Hollywood Town, Soho, and Metropolis. A mouse cursor is hovering over 'Chinatown'.

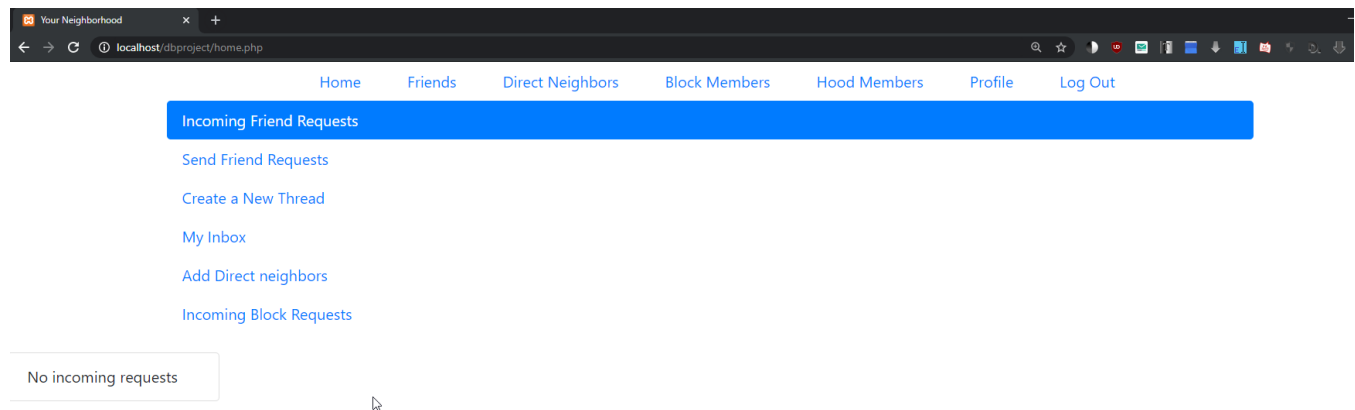
The second screenshot shows the 'localhost/dbproject/blocks.php' page. It displays 'test@gmail.com' and 'Golden Market' with a numeric input field containing the value '6'.

The third screenshot shows the 'localhost/dbproject/confirmation2.php' page. It displays 'test@gmail.com', 'Golden Market', and '63'. Below this, a message states: 'Your block join request has been submitted' followed by a purple link labeled 'Login here'.

Upon being successfully verified, the new user is then added to the record of residents which allows them to sign up and avail of the services. After signing in, the user is greeted by a fully functional and robust home page. The navigation bar provides the user the ability to view their profile page, view their friends, direct neighbors, neighborhood members and bock members. They can also view their own profile pages and update the information.

The screenshot shows a web application interface. At the top is a navigation bar with links: Home, Friends, Direct Neighbors, Block Members, Hood Members, Profile, and Log Out. The main content area displays a user profile for 'Test One' with a placeholder for a 380 x 500 profile picture. To the right of the profile picture, it shows 'Sample Bio', 'test@gmail.com', and 'Family Info'. Below the profile section is an 'Update Profile' form with fields for 'First Name', 'Last Name', and 'Bio', each followed by a 'Submit' button.

On the home page, the user has the option of viewing friend requests that other users have sent them, being able to send friend requests to users, a messages tab where the user can compose and email and send it to a friend, a group of friends, a direct neighbor, a group of direct neighbors, all his block members, or to all of their neighborhood members, a messages tab showing a list of received threads which have the option to reply to them, and a tab that shows incoming block requests from new users.



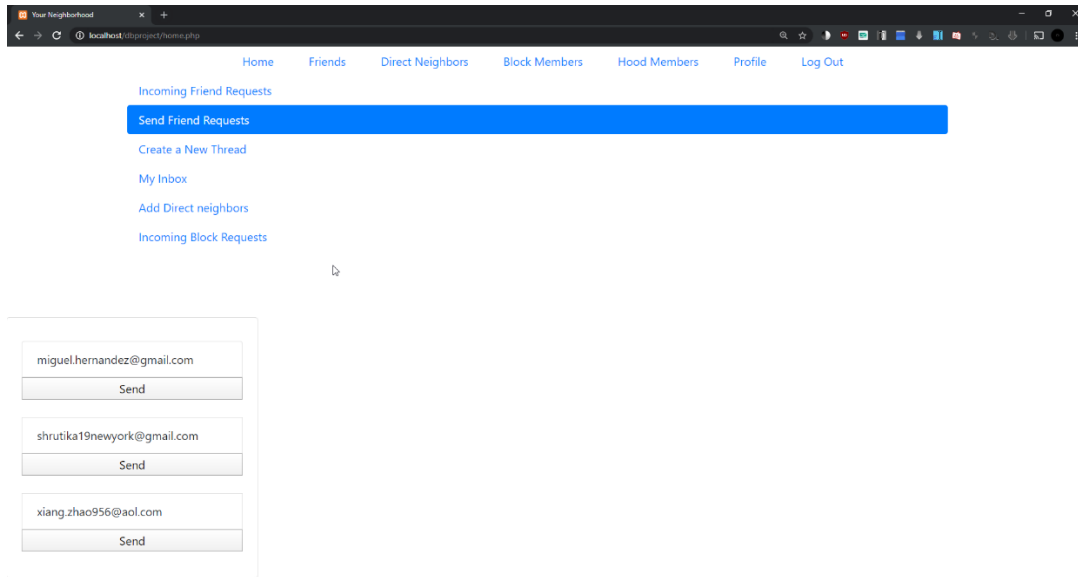
Every time a user logs in, the **logs** table is updated with the session user's email address as well as the current timestamp in order to store a record of recent log ins.

+ Options	
uemail	ltimestamp
test@gmail.com	2019-12-23 16:32:11
shrutika19newyork@gmail.com	2019-12-19 14:21:10

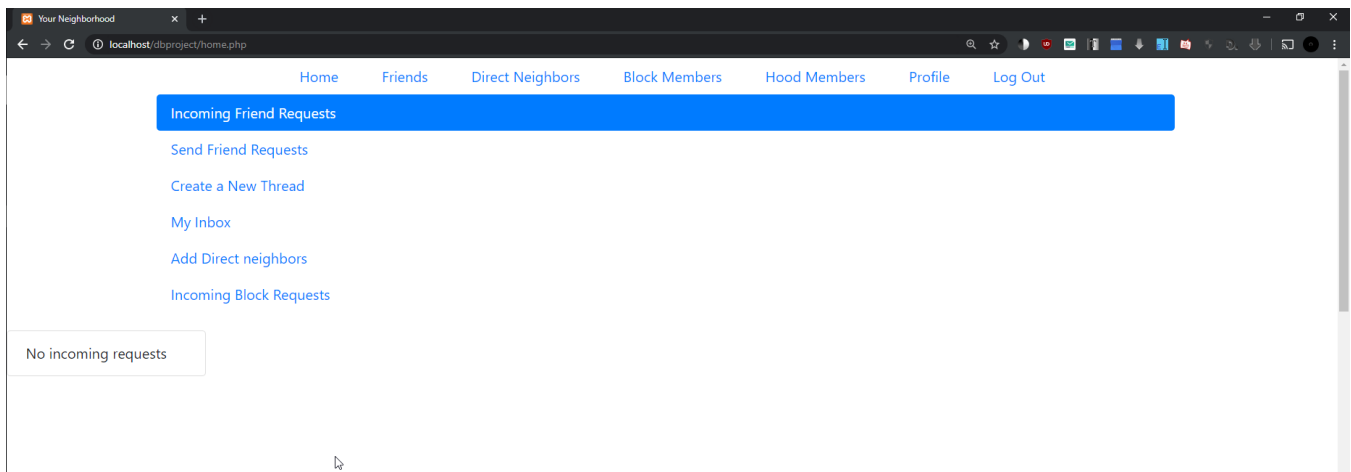
Incoming Block Requests module shows the names of new applicants who live in the same block as the session user. It shows the name as well as a button to verify a user. It is only once three different users verify a new applicant can the new applicant log in. We assume that the blocks must contain at least three different users. We use an SQL query to find the applicant in the **verified** table and check which column to insert the session user's email address if they wish to verify. Through the button *verify* we pass the neighborhood ID and block ID as hidden values to a new form. This new form uses three procedures *startverify_1*, *startverify_2*, and *startverify_3* in order to determine which column of the **verified** table is not null and inserts into it. Once the table contains the emails of three different users, the applicant is then accepted into **residents** table and it is updated. After verifying an applicant, the webpage is redirected back to the homepage after a brief delay.

The Send Friend Requests module is used to view the names of and send a friend request to any of their neighborhood members. If the user has no neighborhood members, then a "No neighborhood members found" message is displayed. On selecting the name of a neighborhood member that the session user wishes to send a

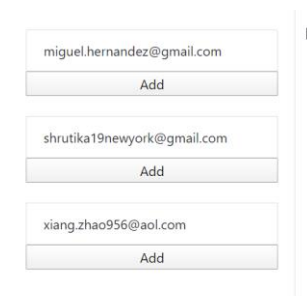
friend request to, the page is directed to *sendreq.php*, where the procedure *frerequest()* is performed. This updates the *friend_request* table with the names of the requestee and the requested. Because friendship status is symmetrical, it is necessary that the requested must accept the friend request before they can be friends. The page is redirected to the home page.



The Incoming Friend Requests module is used to view the names of people who have sent a friend request to the session user and allow the session user to accept or deny a friend request. If the user accepts, the page is sent to a new form *acceptreq.php*. Here the procedure *accept()* is called where the two users are updated into the **friends** table and the friend request status is updated to 1 in the **friend_request** table. If the user denies, the page is sent to a new form *denyreq.php*. Here the procedure *reject()* is called where the two users are deleted from the *friend_requests* table. This allows the users to send each other friend requests at a later time if they wish to. The page is then redirected back to the home page after a short pause.



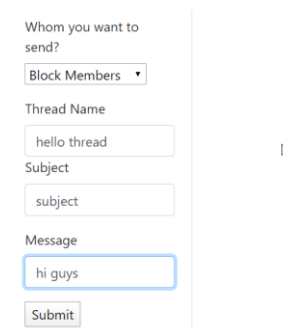
The Add Direct Neighbor module is used to view the names of people who belong to the same neighborhood as the session user and enables the option of adding them as direct neighbors. The key difference is that direct neighborhood is asymmetric, meaning that the neighborhood member does not need to accept or deny a request; instead they are directly added as a direct neighbor. On selecting a neighborhood member from a list, the page is sent to a new form containing *adddn.php*, which further calls the *adddn()* procedure. This procedure adds the two user emails to the **direct_neighbors** table. The page is then redirected back to the home page after a brief pause.



miguel.hernandez@gmail.com	Add
shrutika19newyork@gmail.com	Add
xiang.zhao956@aol.com	Add

The Messages module is used to create a new thread by the session user. They can first select the receiver type of their message via drop down menu, '1' being among either a specific friend or group of friends, '2' being among a specific direct neighbor or a group of direct neighbors, '3' being among all block members, and '4' being among all neighborhood members. The user can then input a thread name, subject, and body of text. Depending on the receiver type, the system behaves differently. The page directs to *newmessage.php*. All of the required variables are sent via POST method.

If the user chooses to send the message to his friends (1), a list of all of the session user's friends are displayed, next to which a button Send is used to send the message. On clicking a button next to a friend's name, the page is sent to a new form pointing to *sendmsgfriend.php*. The essential message values are once again sent via POST to this page, where the procedure *frienddnemail()* is called. This procedure inserts the message into the **message** table with all of the necessary column data. As soon as the data is inserted, a trigger *send_message* is immediately called which is used to update the **inbox** with the data. In order to let the user continue sending the same message to other friends, we once again POST all of the required data back to *newmessage.php* so as to provide a fluid pipeline where the session user doesn't need to restart the messaging process from the beginning and can just send the same message to a different friend.



Whom you want to send?
 Block Members ▾

Thread Name
 hello thread

Subject
 subject

Message
 hi guys

Submit

If the user chooses to send the message to his direct neighbors (2), a list of all of the session user's direct neighbors are displayed, next to which a button Send is used to send the message. On clicking a button next to a direct neighbor's name, the page is sent to a new form pointing to *sendmsgdn.php*. The essential message values are once again sent via POST to this page, where the procedure *frienddnemail()* is called. This procedure inserts the message into the **message** table with all of the necessary column data. As soon as the data is inserted, a trigger *send_message* is immediately called which is used to update the **inbox** table with the data. In order to let the user continue sending the same message to other direct neighbors, we once again POST all of the required data back to *newmessage.php* so as to provide a fluid pipeline where the session user doesn't need to restart the messaging process from the beginning and can just send the same message to a different direct neighbor.

If the user chooses to send the message to all of his block members (3) or neighborhood members (4), then first the system checks a test case if the thread already exists. This is to differentiate the case when a session user creates a thread for his friend/direct neighbor and wishes to create another thread with the same thread name for his block members or neighborhood members. The database is stored in such a way that two friends can send one another and other people a thread with the same name as threads between them and other people, but thread names between block members and neighborhood members must be unique. If there does not exist a thread name already with the same thread name as the proposed message, then the website proceeds without displaying a "Thread Already Exists" page. When the page directs to *newmessage.php*, because the selected receiver type is block members (3) or (4), a procedure *blockemail()* is called which inserts all the data from the message into the **message** table. Similar to earlier, as soon as the data is inserted, a trigger *send_message* is immediately called which is used to update the **inbox** table with the data. The page is then redirected back to home page after a brief delay.

The Inbox module is used to display the inbox of the session user. The inbox is divided into four different feeds corresponding to senders belonging to their friends list, their direct neighbors list, their block member list, or their neighborhood member list. Depending on receiver type, the messages are categorized into different feeds and displayed to the user along with a button that allows the session user to reply to the thread. We pass all of the necessary variables via POST method to *view_thread.php* if the receiver type is either friends or direct neighbors and to *view_thread_bl.php* if the receiver type is either block members or direct neighbors. On these pages, the entire thread messages are displayed including the sender's email, the content of their reply, as well as the timestamp of when the message was sent. Below the thread messages, an empty text box can be used by the session user to compose a message to the thread and submit via a button Reply.

In the first case, the form points to *frienddnreply.php* which is used to call the procedure *frienddnreply()*. This procedure directly inserts the message into the **inbox** table along with all of the necessary information associated with the message. After this, the session variables are sent back to *view_thread.php*, which is used to refresh the page and display all the replies to the thread including the newly sent message when the user clicks on a button to go back to the thread.

Friends Feed	No
Message from any friend	
Direct	
Neighbors	
Feed	
No Message from any friend	
Block Members	
Feed	
comedyshow	
View Thread	
hello thread	
View Thread	
Neighborhood	
Members Feed	
fashion	
View Thread	

In the second case, the form points to *blockhoodreply.php* from *view_thread_bl.php*. This page similarly receives all of the required data from the previous page via POST method and calls the procedure *blockhoodreply()*, which inserts the message data into the **reply** table. Upon inserting data into this table, a trigger *reply_to_inbox* is called which updates the **inbox** table with all of the required message data. After this, the session variables are sent back to *view_thread.php*, which is used to refresh the page and display all the replies to the thread including the newly sent message when the user clicks on a button to go back to the thread.

3
hi guys
test@gmail.com
2019-12-23 16:49:05
Enter reply
Submit

Logging out will take the user back to the home page to sign in again and destroy the current session's credentials.