# Database Project

NYU Tandon - CS6083 - Fall 2019

| | |
|---|---|
| Wenzhou Li | wl2154 |
| Linyi Yan | ly1333 |

# Contents

## Database Project

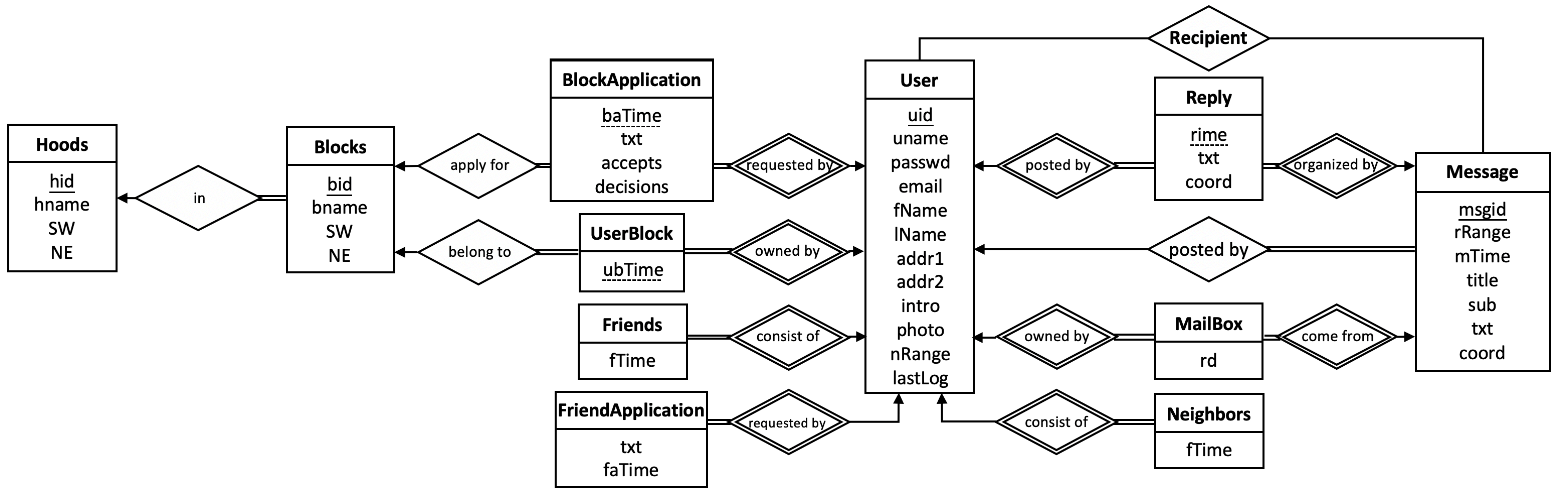
# Part I - Relational Backend Design

## a) Relational Schema & E-R Diagram

*This part describes and justifies how we design the project in detail*

### i. E-R Diagram

*This part is the E-R diagram for the project*



### ii. Relational Schema

*This part shows the final relational schema we designed.*

**Users**(uid, uname, passwd, email, fName, lName, addr1, addr2, intro, photo, nRange, lastLog)

**Hoods**(hid, hname, SW, NE)

**Blocks**(bid, bname, hid, SW, NE)

**UserBlock**(uid, ubTime, bid)

**BlockApplication**(applicant, baTime, bid, txt, accepts, decisions)

**Friends**(uidA, uidB, fTime)

**FriendApplication**(applicant, recipient, txt, faTime)

**Neighbors**(uidA, uidB, nTime)

**Message**(msgid, author, rRange, mtime, title, sub, txt, coord)

**Recipient**(msgid, uid)

**MailBox**(uid, msgid, rd)

**Reply**(msgid, uid, rTime, txt, coord)

## iii. Schema Description

*This part describes relational schema in detail.*

**Users**: Personal information
Primary key: uid
uid: unique id identifies each user
uname, passwd, email, fName, lName: name, password, email, first name, last name of user
addr1, addr2: addr1: street and number, P.O. box, c/o.; addr2: apartment, suite, unit, building, floor, etc.
intro, photo: introduction of user and photo path in server default null when sign up, can be added later
nRange: neighbor range settings, limited to: same building(0) / block(1) / hood(2), default 2
lastLog: last log-out timestamp, update it each time user logs out, default current_timestamp

**Hoods**: ID and names of every hood
Primary key: hid
hid: unique id identifies each hood
hname: name of hood
SW & NE: SoutheEast and NorthEast boundary points of hood

**Blocks**: ID, name and which hood every block belongs to
Primary key: bid
Foreign key: hid refers to hid in Hoods
bid: unique id identifies each block
bname: name of block
hid: hood each block locates in
SW & NE: SoutheEast and NorthEast boundary points of block

**UserBlock**: Members of each block and joining time
Primary key: uid and ubTime
Foreign key: uid refers to uid in Users; bid refers to bid in Blocks
uid: user id of member
ubTime: time when member joins the block
bid: block member joins

**BlockApplication**: Application of joining a block (temporary)
Primary key: applicant and baTime
Foreign key: applicant refers to uid in Users; bid refers to bid in Blocks
applicant: user who apply to join the block
baTime: time when application is submitted
bid: block user apply to join
txt: text applicant can attach with
accepts, decisisons: number of members accepted and decided on the application, default 0
New data can only be inserted after duplicate check

**Friends**: Friends pairs and the time two users become friends
Primary key: uidA and uidB
Foreign key: uidA refers to uid in Users; uidB refers to uid in Users
uidA & uidB: user id who become friends. uidA < uidB
fTime: time users become friend

**FriendApplication**: Application for requests to be friends (temporary)
Primary key: applicant and recipient
Foreign key: applicant refers to uid in Users; recipient refers to uid in Users
applicant: user apply for friendsship
recipient: user receive application
txt: text applicant can attach with
faTime: time when application is submitted
New data can only be inserted after duplicate check

**Neighbors**: User can specify neighbors unilaterally
Primary key: uidA and uidB
Foreign key: uidA refers to uid in Users; uidB refers to uid in Users
uidA & uidB: use A specifies user B as his/her neighbor
nTime: time when user specifies neighbor

**Message**: Messages sent by users
Primary key: uid, msgid
Foreign key: author refers to uid in Users
msgid: unique id identifies each message thread
author: author of initial message in this thread
rRange: recipient range, limited to: particular(0) / friends(1) / neighbors(2) / block(3) / hood(4), def 4
mtime: time when initial message is put
title & sub &txt: title, subject and text. Photo can be embedded in text with path in special format
coord: co-ordinates where author sends the message, default null

**Recipient**: Individuals receive the message when rRange is paricular(0)
Primary key: msgid
Foreign key: msgid refers to msgid in Message; uid refers to uid in Users
msgid: message thread where the recipient is assigned
uid: user id who is assigned to be recipient

**MailBox**: Mail box for each user
Primary key: uid and msgid
Foreign key: msgid refers to msgid in Message; uid refers to uid in Users
uid: owner if mailbox
msgid: message owner can read and reply
rd: bool variant marking read or not, mark False if not read, True if read, default False

**Reply**: Replies to message threads
Primary key: msgid, uid, rTime
Foreign key: msgid refers to msgid in Message; uid refers to uid in Users
msgid: message thread the reply belongs to
uid: user who replies
rTime: time when reply is put
txt: content of reply
coord: co-ordinates where replier sends the message, default null

## iv. Functional Design Description

*This part mainly shows system demand analysis, which is how we design our backend to meet every need.*

**Function 1:** Users should register for the service, specify where they live, post profiles, introduce themselves, upload photos and specify neighbors.

**Design 1**: We designed a Users table to store all kinds of information we need of each user. We use two attributes addr1 and addr2 to represent an address, where addr1 stores street and number, P.O. box, c/o. and addr2 stores apartment, suite, unit, building, floor, etc. Specifically, we design an attribute nRange, which enables users to specify their neighbors (users should have joined the block of current hood). And there are three available choices for nRange: "same building", "same block" and "same hood". We store these choices in integer format in our database: same building (0) / same block (1) / same hood (2). As for addresses, we will preprocess them into a certain format in our background before inserting. We record and update last-log time (lastLog) every time user logs out.

**Relevant Tables**:

Users(uid, uname, passwd, email, fName, lName, addr1, addr2, intro, photo, nRange, lastLog)

**Function 2**: In the website, there are two levels of locality, hoods and blocks. Both cannot be created by users and are modeled as axis-aligned rectangles that can be defined by two corner points.

**Design 2**: We designed Hoods and Blocks table to implement this function. For both hoods and blocks, we record a unique id for each data to make them easier to use (hid and bid). We can then acquire full information about given block or hood. Meanwhile, a block is a part of a neighborhood and each block only belongs to one neighborhood. So, we add hid to Blocks. Since blocks and neighborhoods are defined by two corner points, we use SW (southwest) and NE (northeast) corner points to model them.

**Relavant Tables**:

Hoods(hid, hname, SW, NE)

Blocks(bid, bname, hid, SW, NE)

**Function 3**: Users can apply to join a block, and they are accepted as a new member if at least three existing members or all members if less than three approve. A user can only be in one block and automotically a member of hood in which the block is located.

**Design 3**: We designed a BlockApplication table to temporarily store join-block application. When someone submits an application, we store it in BlockApplication table. User can attach some notes in txt in order to get his/her applications approved with higher probability. Every time a user logs in, we scan this table and check whether there is someone trying to join his/her block. If there is one application record whose bid is just the same as his/her block's bid, we push a notification to let him/her decide whether to allow the applicant to join(Yes) or not(No). We add 1 to accepts when someone presses "Yes", and 1 to decisions no matter Yes or No. Every time either of them increases, we check whether the applicant can join the block. If the number of accepts satisfies the requirement, we put the user into UserBlock table, which means he/she is officially a member of the block now. Otherwise, if the number in decisions is equal to the number of members in such block (can be counted in UserBlock) but the accepts does not meet the requirement, we push a rejection notification to the applicant. Besides, system regards applications existing over two weeks also as rejected. After applicant received either notification, we will delete the row in BlockApplication. Every time an applicant submits a request, we check whether there is already one and not expired in BlockApplication. (can be determined by applicant and bid). If there is, we notify the applicant that he/she has already submitted. Since the relationship between hood and block is established in Blocks table, we can join

UserBlock and Blocks to deduce which hood a user is automatically a member of.
**Relevant Tables**:
Blocks(bid, bname, hid, SW, NE)
UserBlock(uid, ubTime, bid)
BlockApplication(applicant, baTime, bid, txt, accepts, decisions)

**Function 4**: Members can specify two types of relationships (friends or neighbors)
**Design 4**: We've explained the attribute nRange in Users table, which allows users to unilaterally choose his/her neighbors. Thus, we designed table Neighbors for each user (uidA) to store neighbors (uidB) they choose. We also store the time an user specifies neighbor in nTime. For friendship, we designed FriendApplication and Friends tables. They are similar to UserBlock and BlockApplication mentioned in Design 3. One can request friendship and recipient will decide whether to accept. This will be recorded in FriendApplication temporarily, which will be deleted once decision is made. If recipient accepts, a new record will be inserted to Friends. Particularly, we need to point it out that we will check whether applicant and recipient are both in a row from Friends (as uidA and uidB). If it is, we will not initiate the request and immediately notify applicant that they are already friends. Or, if an application is already in FriendApplication (applicant and recipient can identify), we will return to applicant that he/she has already pulled the request, then update faTime. Nothing will change in Friends and FriendApplication under both circumstances.
**Relevant Tables**:
UserBlock(uid, ubTime, bid)
BlockApplication(applicant, baTime, bid, txt, accepts, decisions)
Friends(uidA, uidB, fTime)
Neighbors(uidA, uidB, nTime)
FriendApplication(applicant, recipient, txt, faTime)

**Function 5**: People can post, read and reply messages. A user can send message to a person who is a friend or a neighbor, or all of their friends, or entire block, or entire hood they are a member of. Reply can be read and replied by anyone who received the earlier message. Feeds can be separated.
**Design 5**: We designed Message, Reply and MailBox tables to meet these requirements. We use msgid to identify each thread of messages. We only have the initial message in Message, while all replies are stored in Reply and identified by msgid from Message. And author, mTime, title, sub and rRange are only in Message, because they are redundant to Reply. For rRange, we limit them to predefined choices: particular, friends, neighbor, block and hood. We store these as preset integers: particular (0) / friends(1) / neighbor (2) / block(3) / hood(4)). If a user chooses to send a message to a group (friends, block, etc.), we can query relevant tables (Blocks, Neighbors, UserBlock, Friends, etc.) and deduce who they are specifically, then store the message in MainBox according to uid, msgid and mark rd as False. Or, if a user wants to direct a message to some particular people (by just presenting their names), he shall choose "particular" and we will store all these particular individuals into Recipient table based on msgid. And before we actually act, we will check whether those recipients are his/her friends or neighbors by querying tables: Friends, Neighbors, UserBlock and Blocks. As for replies, we need to check whether current replier is one of the message's recipients, or he/she is the author (join Message, Reply, Recipient). For MailBox, we check and push notification every time users log in. Every time a new message or reply is put, we mark message thread (msgid in MailBox) as unread for all recipients, unless he/she is author or replier. And we can separate messages into neighbor, friend, block and hood

according to rRange (Message, MailBox, UserBlock, Users, Friends). For each thread, we list messages according to mTime or rTime.
**Relevant Tables**:
Users(uid, uname, passwd, email, fName, lName, addr1, addr2, intro, photo, nRange, lastLog)
Blocks(bid, bname, hid, SW, NE)
UserBlock(uid, ubTime, bid)
Friends(uidA, uidB, fTime)
Neighbors(uidA, uidB, nTime)
Message(msgid, author, rRange, mtime, title, sub, txt, coord)
Recipient(msgid, uid)
MailBox(uid, msgid, rd)
Reply(msgid, uid, rTime, txt, coord)

**Fuction 6**: System can show only threads with new messages since the last time visited, or profiles of new members, or threads with new messages unread.
**Design 6**: In Users, we record user's last log-out information (lastLog). We can filter message since lastLog, then query MailBox (msgid) and Message (mTime) based on it. The same thing is with new members since we also store the time when a new member joins (ubTime in UserBlock). As for threads with new message unread, we've described it in Design 5 by using rd in MailBox. Every time a new message or reply comes up, we mark it as unread.
**Relevant Tables**:
Users(uid, uname, passwd, email, firstName, lastName, addr, intro, photo, neighborRange, lastLog)
UserBlock(uid, ubTime, bid)
Message(msgid, author, rRange, mtime, title, sub, txt, coord)
MailBox(uid, msgid, rd)

**Function 7**: Users move to another block
**Design 7**: If users apply to join another block and get approved, we will not delete the previous data in UserBlock but add new rows (we have ubTime as part of primary key). Since we already treat our messages as Email, users can decide whether to hide previous messages. We can do this by comparing mTime in and ubTime. The reason we choose it this way is that we think there may be some important information in the past messages user might need to look up again in the future. However, since the user is no longer in the previous block, he/she would not be in the recipients relevant to it any longer. Thus, he/she shall not receive any new message threads from previous blocks or hoods. As for friends, we think friendship may last forever, so we won't delete friend information. However, users can hide old friends by comparing fTime and ubTime in Friends and UserBlock. In all, we will keep the past information about messages and friends for users and users can choose to hide these information or not. However, neighbors are unilaterally from the start, so after a user moves to another block and join the new group, he/she should select his/her new neighbors (all old ones will be deleted).
**Relevant Tables**:
Users(uid, uname, passwd, email, fName, lName, addr1, addr2, intro, photo, nRange, lastLog)
UserBlock(uid, ubTime, bid)
Friends(uidA, uidB, fTime)
Neighbors(uidA, uidB, nTime)
Message(msgid, author, rRange, mtime, title, sub, txt, coord)

## b) Create Database

*This part is how we build our database named neighborChat*

```
DROP DATABASE IF EXISTS neighborChat;
CREATE DATABASE neighborChat;
USE neighborChat;

DROP TABLE IF EXISTS Users;
CREATE TABLE Users (
    uid INT(11) NOT NULL AUTO_INCREMENT,
    uname VARCHAR(50) UNIQUE NOT NULL,
    passwd VARCHAR(50) NOT NULL,
    fName VARCHAR(50) NOT NULL,
    lName VARCHAR(50) NOT NULL,
    addr1 VARCHAR(50) DEFAULT NULL,
    addr2 VARCHAR(50) DEFAULT NULL,
    intro VARCHAR(50) DEFAULT NULL,
    photo VARCHAR(100) DEFAULT NULL,
    nRange INT(1) DEFAULT 2,
    lastLog TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (uid));

DROP TABLE IF EXISTS Hoods;
CREATE TABLE Hoods (
    hid INT(11) NOT NULL AUTO_INCREMENT,
    hname VARCHAR(50) NOT NULL,
    SW VARCHAR(50) NOT NULL,
    NE VARCHAR(50) NOT NULL,
    PRIMARY KEY (hid));

DROP TABLE IF EXISTS Blocks;
CREATE TABLE Blocks (
    bid INT(11) NOT NULL AUTO_INCREMENT,
    bname VARCHAR(50) NOT NULL,
    hid INT(11) NOT NULL,
    SW VARCHAR(50) NOT NULL,
    NE VARCHAR(50) NOT NULL,
    PRIMARY KEY (bid),
    FOREIGN KEY (hid) REFERENCES Hoods (hid) ON DELETE CASCADE ON UPDATE CASCADE);

DROP TABLE IF EXISTS UserBlock;
CREATE TABLE UserBlock (
    uid INT(11) NOT NULL,
    ubTime TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    bid INT(11) NOT NULL,
    PRIMARY KEY (uid, ubTime),
    FOREIGN KEY (uid) REFERENCES Users (uid) ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (bid) REFERENCES Blocks (bid) ON DELETE CASCADE ON UPDATE CASCADE);

DROP TABLE IF EXISTS BlockApplication;
CREATE TABLE BlockApplication (
    applicant INT(11) NOT NULL,
    baTime TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    bid INT(11) NOT NULL,
    txt VARCHAR(100) DEFAULT NULL,
    accepts INT DEFAULT 0,
    decisions INT DEFAULT 0,
    PRIMARY KEY (applicant, baTime),
    FOREIGN KEY (applicant) REFERENCES Users (uid) ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (bid) REFERENCES Blocks (bid) ON DELETE CASCADE ON UPDATE CASCADE);

DROP TABLE IF EXISTS Friends;
CREATE TABLE Friends (
    uidA INT(11) NOT NULL,
    uidB INT(11) NOT NULL,
    fTime TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (uidA, uidB),
    FOREIGN KEY (uidA) REFERENCES Users (uid) ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (uidB) REFERENCES Users (uid) ON DELETE CASCADE ON UPDATE CASCADE);

DROP TABLE IF EXISTS FriendApplication;
CREATE TABLE FriendApplication (
    applicant INT(11) NOT NULL,
    recipient INT(11) NOT NULL,
    txt VARCHAR (100) DEFAULT NULL,
    faTime TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (applicant, recipient),
    FOREIGN KEY (applicant) REFERENCES Users (uid) ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (recipient) REFERENCES Users (uid) ON DELETE CASCADE ON UPDATE CASCADE);

DROP TABLE IF EXISTS Neighbors;
CREATE TABLE Neighbors (
    uidA INT(11) NOT NULL,
    uidB INT(11) NOT NULL,
    nTime TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (uidA, uidB),
    FOREIGN KEY (uidA) REFERENCES Users (uid) ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (uidB) REFERENCES Users (uid) ON DELETE CASCADE ON UPDATE CASCADE);

DROP TABLE IF EXISTS Message;
CREATE TABLE Message (
    msgid INT(20) NOT NULL AUTO_INCREMENT,
    author INT(11) NOT NULL,
    rRange INT(1) DEFAULT 4,
    mtime TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    title VARCHAR(50) NOT NULL,
    sub VARCHAR(50) DEFAULT NULL,
    txt VARCHAR(1000) NOT NULL,
    coord VARCHAR(50) DEFAULT NULL,
    PRIMARY KEY (msgid),
    FOREIGN KEY (author) REFERENCES Users (uid) ON DELETE CASCADE ON UPDATE CASCADE);

DROP TABLE IF EXISTS Recipient;
CREATE TABLE Recipient (
    msgid INT(20) NOT NULL,
    uid INT(11) NOT NULL,
    PRIMARY KEY (msgid),
    FOREIGN KEY (msgid) REFERENCES Message (msgid) ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (uid) REFERENCES Users (uid) ON DELETE CASCADE ON UPDATE CASCADE);

DROP TABLE IF EXISTS MailBox;
CREATE TABLE MailBox (
    uid INT(11) NOT NULL,
    msgid INT(20) NOT NULL,
    rd BOOL NOT NULL DEFAULT FALSE,
    PRIMARY KEY (uid, msgid),
    FOREIGN KEY (uid) REFERENCES Users (uid) ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (msgid) REFERENCES Message (msgid) ON DELETE CASCADE ON UPDATE CASCADE);

DROP TABLE IF EXISTS Reply;
CREATE TABLE Reply (
    msgid INT(20) NOT NULL,
    uid INT(11) NOT NULL,
    rTime TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    txt VARCHAR(1000) NOT NULL,
    coord VARCHAR(50) DEFAULT NULL,
    PRIMARY KEY (msgid, uid, rTime),
    FOREIGN KEY (msgid) REFERENCES Message (msgid) ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (uid) REFERENCES Users (uid) ON DELETE CASCADE ON UPDATE CASCADE);
```

## c) SQL Queries

*This part are SQL queries for given tasks*

### i. Join

-- Users sign up

INSERT INTO Users VALUES (1, "user01", "12345678", "Justin", "Bieber", "NULL", "NULL", "NULL", "NULL", 0, "2019-11-01 12:00:00");

-- Users apply to become members of a block

INSERT INTO BlockApplication VALUES (6, "2019-01-01 12:00:02", 2, "I am your new neighbor", 0, 0);

-- Users create profiles

UPDATE Users SET addr1 = "343 Gold Street, Brooklyn", addr2 = "Apt 0001", intro = "LOL", photo = "/users/photo/1.png"
WHERE uid = 1;

-- Users edit profiles

UPDATE Users SET addr1 = "343 Gold Street, Brooklyn", addr2 = "Apt 4001", intro = "Hello World!"

### ii. Content Posting

-- User starts a thread by posting an initial message

INSERT INTO Message VALUES (6, 9, 0, "2019-01-01 12:00:04", "Hi", "Life", "I love you", "(40.7657, -73.9761)");

-- User replies to a message

INSERT INTO Reply VALUES (6, 10, "2019-01-01 13:00:04", "I love you too", "");

### iii. Friendship

-- User applies for friendship

INSERT INTO FriendApplication VALUES (2, 3, "I wanna be your friend", "2019-01-01 12:00:03");

-- User accepts friend request

DELETE FROM FriendApplication WHERE applicant = 2 AND recipient = 3;
INSERT INTO Friends VALUES (2, 3, "2019-11-29 12:00:03");

-- User adds new neighbor

INSERT INTO Neighbors VALUES (3, 1, "2019-01-01 12:00:03");

-- All current friends to user whose uid = 2

SELECT * FROM Friends WHERE uidA = 2 OR uidB = 2;

-- All current neighbors to uid = 3

SELECT * FROM Neighbors WHERE uidA = 3;

### iv. Browse and Search Messages

-- List all threads in a user(9)'s block feed that have new messages since the last time the user accessed

SELECT * FROM Message
WHERE rRange = 3 AND mtime >= (SELECT lastLog FROM Users WHERE uid = 9)
AND author in (SELECT uid FROM UserBlock WHERE bid = (SELECT bid FROM UserBlock WHERE uid = 9));

-- List all threads in user(2)'s friend feed that have unread messages

SELECT * FROM Message NATURAL JOIN MailBox WHERE uid = 2 AND rd = FALSE AND rRange = 1;

-- List all messages containing the words “bicycle accident” across all feeds that user(8) can access

SELECT * FROM Mailbox NATURAL JOIN Message
WHERE uid = 8 AND (title LIKE "%bicycle accident%" OR sub LIKE "%bicycle accident%" OR txt LIKE "%bicycle accident%");

## d) Sample Data Test

### i. Insert Sample Data

*This part we insert sample data in order to test our database*

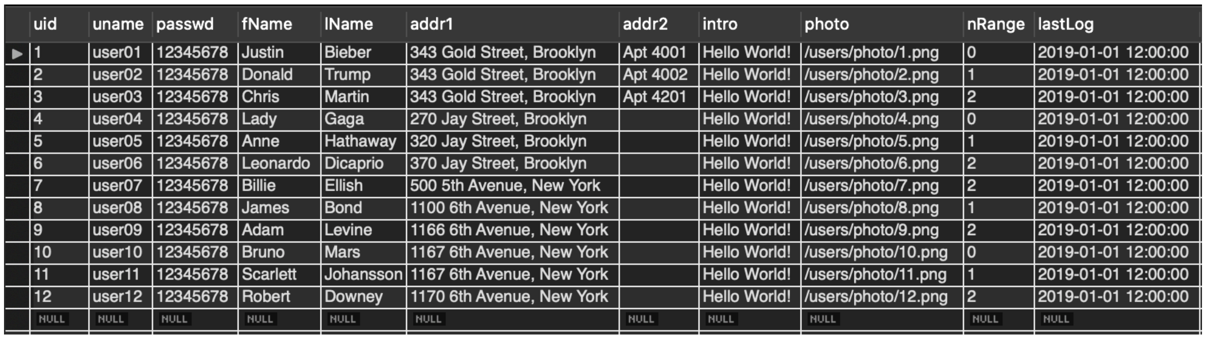
**Users** (uid, uname, passwd, email, fName, lName, addr1, addr2, intro, photo, nRange, lastLog)

nRange: same building(0) / block(1) / hood(2)

-- Live in the same building, different nRange: uid 1 – 3; uid 10 - 11

-- Live in the same block, different nRange: uid 1 – 3; uid 4 – 6; uid 7; uid 8 – 12

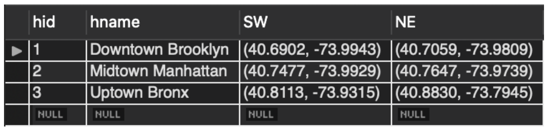
-- Live in the same hood, different nRange: uid 1 – 6; uid 7 – 12



| uid | uname | passwd | fName | lName | addr1 | addr2 | intro | photo | nRange | lastLog |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | user01 | 12345678 | Justin | Bieber | 343 Gold Street, Brooklyn | Apt 4001 | Hello World! | /users/photo/1.png | 0 | 2019-01-01 12:00:00 |
| 2 | user02 | 12345678 | Donald | Trump | 343 Gold Street, Brooklyn | Apt 4002 | Hello World! | /users/photo/2.png | 1 | 2019-01-01 12:00:00 |
| 3 | user03 | 12345678 | Chris | Martin | 343 Gold Street, Brooklyn | Apt 4201 | Hello World! | /users/photo/3.png | 2 | 2019-01-01 12:00:00 |
| 4 | user04 | 12345678 | Lady | Gaga | 270 Jay Street, Brooklyn | | Hello World! | /users/photo/4.png | 0 | 2019-01-01 12:00:00 |
| 5 | user05 | 12345678 | Anne | Hathaway | 320 Jay Street, Brooklyn | | Hello World! | /users/photo/5.png | 1 | 2019-01-01 12:00:00 |
| 6 | user06 | 12345678 | Leonardo | Dicaprio | 370 Jay Street, Brooklyn | | Hello World! | /users/photo/6.png | 2 | 2019-01-01 12:00:00 |
| 7 | user07 | 12345678 | Billie | Ellish | 500 5th Avenue, New York | | Hello World! | /users/photo/7.png | 2 | 2019-01-01 12:00:00 |
| 8 | user08 | 12345678 | James | Bond | 1100 6th Avenue, New York | | Hello World! | /users/photo/8.png | 1 | 2019-01-01 12:00:00 |
| 9 | user09 | 12345678 | Adam | Levine | 1166 6th Avenue, New York | | Hello World! | /users/photo/9.png | 2 | 2019-01-01 12:00:00 |
| 10 | user10 | 12345678 | Bruno | Mars | 1167 6th Avenue, New York | | Hello World! | /users/photo/10.png | 0 | 2019-01-01 12:00:00 |
| 11 | user11 | 12345678 | Scarlett | Johansson | 1167 6th Avenue, New York | | Hello World! | /users/photo/11.png | 1 | 2019-01-01 12:00:00 |
| 12 | user12 | 12345678 | Robert | Downey | 1170 6th Avenue, New York | | Hello World! | /users/photo/12.png | 2 | 2019-01-01 12:00:00 |
| NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL |

**Hoods** (hid, hname, SW, NE)

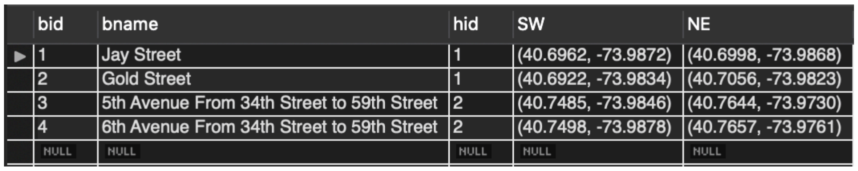
-- Live in different hoods: hid 1 - 3



| hid | hname | SW | NE |
|---|---|---|---|
| 1 | Downtown Brooklyn | (40.6902, -73.9943) | (40.7059, -73.9809) |
| 2 | Midtown Manhattan | (40.7477, -73.9929) | (40.7647, -73.9739) |
| 3 | Uptown Bronx | (40.8113, -73.9315) | (40.8830, -73.7945) |
| NULL | NULL | NULL | NULL |

**Blocks** (bid, bname, hid, SW, NE)

-- Live in different blocks: bid 1 - 4



| bid | bname | hid | SW | NE |
|---|---|---|---|---|
| 1 | Jay Street | 1 | (40.6962, -73.9872) | (40.6998, -73.9868) |
| 2 | Gold Street | 1 | (40.6922, -73.9834) | (40.7056, -73.9823) |
| 3 | 5th Avenue From 34th Street to 59th Street | 2 | (40.7485, -73.9846) | (40.7644, -73.9730) |
| 4 | 6th Avenue From 34th Street to 59th Street | 2 | (40.7498, -73.9878) | (40.7657, -73.9761) |
| NULL | NULL | NULL | NULL | NULL |

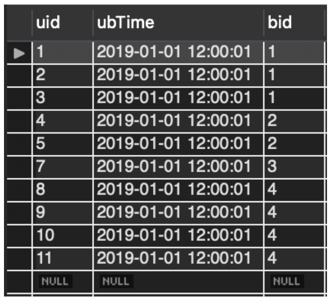
**UserBlock** (uid, ubTime, bid)

-- For block 1, all users(uid 1-3) in this block have joined the block group (3 accepts required)

-- For block 2, two out of three users (uid 4-5) have already joined (all members accepts required)

-- For block 3, a user(uid 7) is the first person to join the block (applicant already automatically joined)

-- For block 4, four out of five users(uid 8-11) have already joined (3 acceptes required)



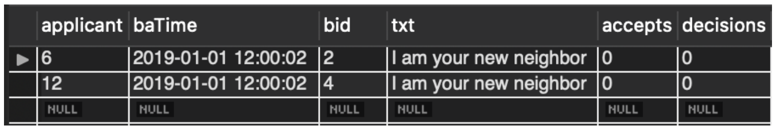
| uid | ubTime | bid |
|---|---|---|
| 1 | 2019-01-01 12:00:01 | 1 |
| 2 | 2019-01-01 12:00:01 | 1 |
| 3 | 2019-01-01 12:00:01 | 1 |
| 4 | 2019-01-01 12:00:01 | 2 |
| 5 | 2019-01-01 12:00:01 | 2 |
| 7 | 2019-01-01 12:00:01 | 3 |
| 8 | 2019-01-01 12:00:01 | 4 |
| 9 | 2019-01-01 12:00:01 | 4 |
| 10 | 2019-01-01 12:00:01 | 4 |
| 11 | 2019-01-01 12:00:01 | 4 |
| NULL | NULL | NULL |

**BlockApplication** (applicant, baTime, bid, txt, accepts, decisions)

-- For block 2, applicant (6) applies to join. Since block members <= 3, all members agreement required

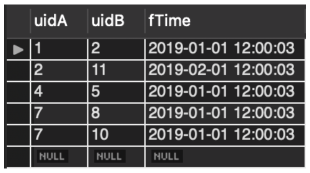
-- For block 3, block member was 0, applicant 7 automatically joined, so there is no record here

-- For block 4, , applicant (12) applies to join. Since block member > 3, only 3 accepts required



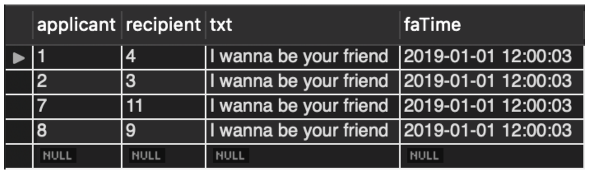
| | applicant | baTime | bid | txt | accepts | decisions |
|---|---|---|---|---|---|---|
| ▶ | 6 | 2019-01-01 12:00:02 | 2 | I am your new neighbor | 0 | 0 |
| | 12 | 2019-01-01 12:00:02 | 4 | I am your new neighbor | 0 | 0 |
| | NULL | NULL | NULL | NULL | NULL | NULL |

**Friends** (uidA, uidB, fTime)
-- Two people already in the same block group: uid 1&2; uid 4&5
-- Two people live in the same hood, but not the same block: uid 7&8, uid 7&10
-- One moves to a new hood maintaining old friends and making new friends: uid 2&11



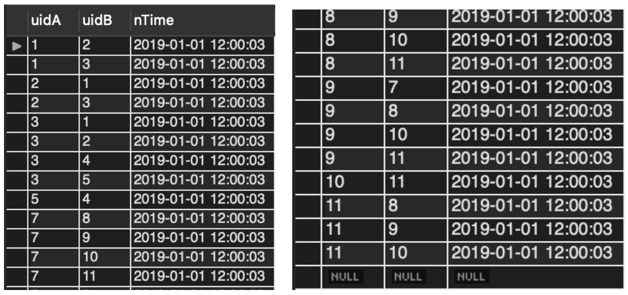
| | uidA | uidB | fTime |
|---|---|---|---|
| ▶ | 1 | 2 | 2019-01-01 12:00:03 |
| | 2 | 11 | 2019-02-01 12:00:03 |
| | 4 | 5 | 2019-01-01 12:00:03 |
| | 7 | 8 | 2019-01-01 12:00:03 |
| | 7 | 10 | 2019-01-01 12:00:03 |
| | NULL | NULL | NULL |

**FriendApplication** (applicant, recipient, txt, faTime)
-- Two people already in the same block group: applicant 2, 8
-- Two people live in the same hood, but not the same block: applicant 1, 7



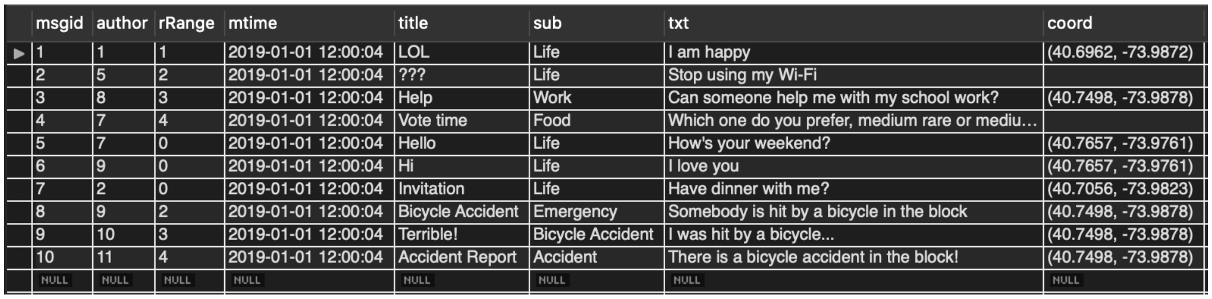
| | applicant | recipient | txt | faTime |
|---|---|---|---|---|
| ▶ | 1 | 4 | I wanna be your friend | 2019-01-01 12:00:03 |
| | 2 | 3 | I wanna be your friend | 2019-01-01 12:00:03 |
| | 7 | 11 | I wanna be your friend | 2019-01-01 12:00:03 |
| | 8 | 9 | I wanna be your friend | 2019-01-01 12:00:03 |
| | NULL | NULL | NULL | NULL |

**Neighbors** (uidA, uidB, nTime)
-- Users select their neighbor range unilaterally(referenced to nRange in Users table)



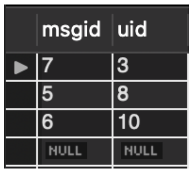
| | uidA | uidB | nTime |
|---|---|---|---|
| ▶ | 1 | 2 | 2019-01-01 12:00:03 |
| | 1 | 3 | 2019-01-01 12:00:03 |
| | 2 | 1 | 2019-01-01 12:00:03 |
| | 2 | 3 | 2019-01-01 12:00:03 |
| | 3 | 1 | 2019-01-01 12:00:03 |
| | 3 | 2 | 2019-01-01 12:00:03 |
| | 3 | 4 | 2019-01-01 12:00:03 |
| | 3 | 5 | 2019-01-01 12:00:03 |
| | 5 | 4 | 2019-01-01 12:00:03 |
| | 7 | 8 | 2019-01-01 12:00:03 |
| | 7 | 9 | 2019-01-01 12:00:03 |
| | 7 | 10 | 2019-01-01 12:00:03 |
| | 7 | 11 | 2019-01-01 12:00:03 |
| | 8 | 9 | 2019-01-01 12:00:03 |
| | 8 | 10 | 2019-01-01 12:00:03 |
| | 8 | 11 | 2019-01-01 12:00:03 |
| | 9 | 7 | 2019-01-01 12:00:03 |
| | 9 | 8 | 2019-01-01 12:00:03 |
| | 9 | 10 | 2019-01-01 12:00:03 |
| | 9 | 11 | 2019-01-01 12:00:03 |
| | 10 | 11 | 2019-01-01 12:00:03 |
| | 11 | 8 | 2019-01-01 12:00:03 |
| | 11 | 9 | 2019-01-01 12:00:03 |
| | 11 | 10 | 2019-01-01 12:00:03 |
| | NULL | NULL | NULL |

**Message** (msgid, author, rRange, mtime, title, sub, txt, coord)
-- Someone sends message to friends: msgid 1
-- Someone sends message to neighbors, and coordinates is null: msgid 2
-- Someone sends message to block: msgid 3
-- Someone sends message to hood, and coordinates is null: msgid 4
-- Someone sends message to particular person from friends: msgid 5
-- Someone sends message to particular person from neighbors: msgid 6- 7
-- Specifically for bicycle accident: msgid 8 - 10
---- Keyword in title: msgid 8 ---- Keyword in subject: msgid 9 ---- Keyword in text: msgid 10



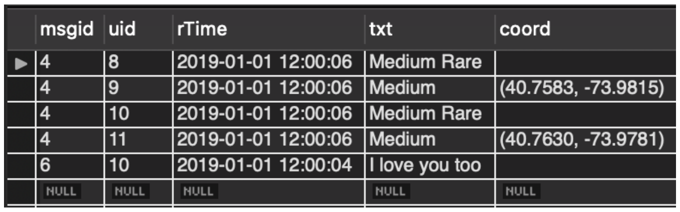
| | msgid | author | rRange | mtime | title | sub | txt | coord |
|---|---|---|---|---|---|---|---|---|
| ▶ | 1 | 1 | 1 | 2019-01-01 12:00:04 | LOL | Life | I am happy | (40.6962, -73.9872) |
| | 2 | 5 | 2 | 2019-01-01 12:00:04 | ??? | Life | Stop using my Wi-Fi | |
| | 3 | 8 | 3 | 2019-01-01 12:00:04 | Help | Work | Can someone help me with my school work? | (40.7498, -73.9878) |
| | 4 | 7 | 4 | 2019-01-01 12:00:04 | Vote time | Food | Which one do you prefer, medium rare or mediu... | |
| | 5 | 7 | 0 | 2019-01-01 12:00:04 | Hello | Life | How's your weekend? | (40.7657, -73.9761) |
| | 6 | 9 | 0 | 2019-01-01 12:00:04 | Hi | Life | I love you | (40.7657, -73.9761) |
| | 7 | 2 | 0 | 2019-01-01 12:00:04 | Invitation | Life | Have dinner with me? | (40.7056, -73.9823) |
| | 8 | 9 | 2 | 2019-01-01 12:00:04 | Bicycle Accident | Emergency | Somebody is hit by a bicycle in the block | (40.7498, -73.9878) |
| | 9 | 10 | 3 | 2019-01-01 12:00:04 | Terrible! | Bicycle Accident | I was hit by a bicycle... | (40.7498, -73.9878) |
| | 10 | 11 | 4 | 2019-01-01 12:00:04 | Accident Report | Accident | There is a bicycle accident in the block! | (40.7498, -73.9878) |
| | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL |

**Recipient** (msgid, uid)
-- Recipient test data based on test data in Message: msgid 5 - 7



| msgid | uid |
|---|---|
| 7 | 3 |
| 5 | 8 |
| 6 | 10 |
| NULL | NULL |

**Reply** (msgid, uid, rTime, txt, coord)
-- Reply to particular message: msgid 6
-- Reply to group message: msgid 4

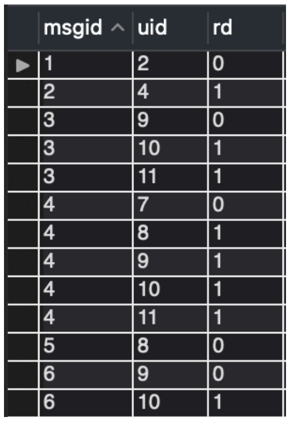


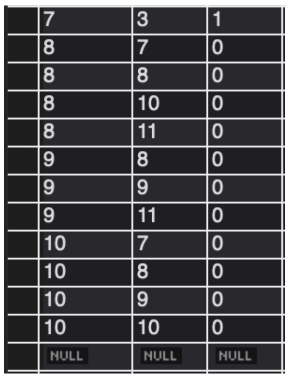
| msgid | uid | rTime | txt | coord |
|---|---|---|---|---|
| 4 | 8 | 2019-01-01 12:00:06 | Medium Rare | |
| 4 | 9 | 2019-01-01 12:00:06 | Medium | (40.7583, -73.9815) |
| 4 | 10 | 2019-01-01 12:00:06 | Medium Rare | |
| 4 | 11 | 2019-01-01 12:00:06 | Medium | (40.7630, -73.9781) |
| 6 | 10 | 2019-01-01 12:00:04 | I love you too | |
| NULL | NULL | NULL | NULL | NULL |

**MailBox** (uid, msgid, rd)
-- Based on three tables above(Message, Recipient, Reply), we can draw the following table, which can be used to deduce MailBox.

| msgid | nRange | author | recipient(s) |
|---|---|---|---|
| 1 | Friend | 1 | 2 |
| 2 | Neighbor | 5 | 4 |
| 3 | Block | 8 | 9, 10, 11 |
| 4 | Hood | 7 | 7, 8, 9, 10, 11 |
| 5 | Particular | 7 | 8 |
| 6 | Particular | 9 | 9, 10 |
| 7 | Particular | 2 | 3 |
| 8 | Neighbor | 9 | 7, 8, 10, 11 |
| 9 | Block | 10 | 8, 9, 11 |
| 10 | Hood | 11 | 7, 8, 9, 10 |

-- General test data based on test data in Message: msgid 1 - 3
-- They need to reply, so they must have read the message: uid 8 - 11, msgid 4
-- After being replied, author of message thread will have his/her mailbox updated: uid 7, msgid 4
-- Specifically for bicycle accident: uid 7 - 11, msgid 8 - 10





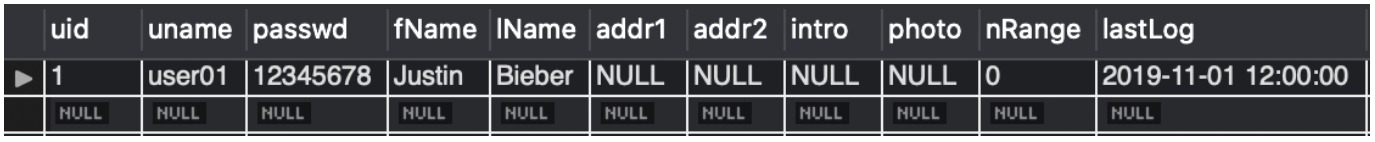
| msgid | uid | rd |
|---|---|---|
| 1 | 2 | 0 |
| 2 | 4 | 1 |
| 3 | 9 | 0 |
| 3 | 10 | 1 |
| 3 | 11 | 1 |
| 4 | 7 | 0 |
| 4 | 8 | 1 |
| 4 | 9 | 1 |
| 4 | 10 | 1 |
| 4 | 11 | 1 |
| 5 | 8 | 0 |
| 6 | 9 | 0 |
| 6 | 10 | 1 |
| 7 | 3 | 1 |
| 8 | 7 | 0 |
| 8 | 8 | 0 |
| 8 | 10 | 0 |
| 8 | 11 | 0 |
| 9 | 8 | 0 |
| 9 | 9 | 0 |
| 9 | 11 | 0 |
| 10 | 7 | 0 |
| 10 | 8 | 0 |
| 10 | 9 | 0 |
| 10 | 10 | 0 |
| NULL | NULL | NULL |

## ii. Test Queries

*This part we test SQL queries written in c) with sample data from d) i*

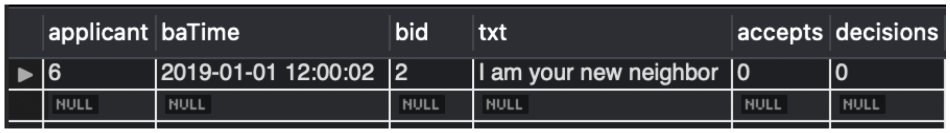
**Join**

-- Users sign up



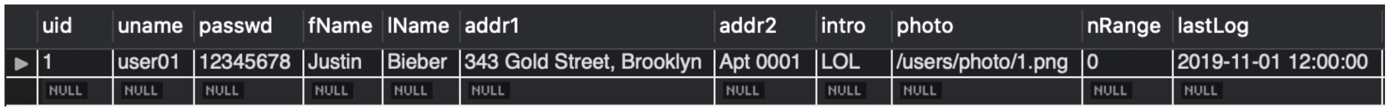
| | uid | uname | passwd | fName | lName | addr1 | addr2 | intro | photo | nRange | lastLog |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ▶ | 1 | user01 | 12345678 | Justin | Bieber | NULL | NULL | NULL | NULL | 0 | 2019-11-01 12:00:00 |
| | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL |

-- Users apply to become members of a block



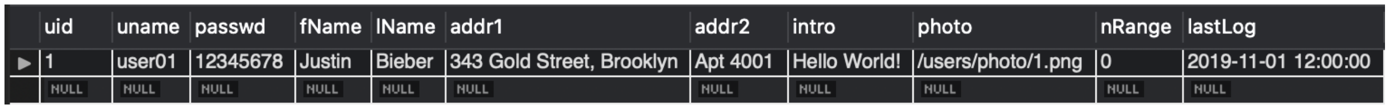
| | applicant | baTime | bid | txt | accepts | decisions |
|---|---|---|---|---|---|---|
| ▶ | 6 | 2019-01-01 12:00:02 | 2 | I am your new neighbor | 0 | 0 |
| | NULL | NULL | NULL | NULL | NULL | NULL |

-- Users create profiles



| | uid | uname | passwd | fName | lName | addr1 | addr2 | intro | photo | nRange | lastLog |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ▶ | 1 | user01 | 12345678 | Justin | Bieber | 343 Gold Street, Brooklyn | Apt 0001 | LOL | /users/photo/1.png | 0 | 2019-11-01 12:00:00 |
| | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL |

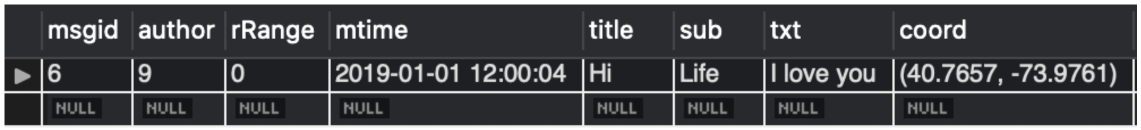
-- Users edit profiles



| | uid | uname | passwd | fName | lName | addr1 | addr2 | intro | photo | nRange | lastLog |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ▶ | 1 | user01 | 12345678 | Justin | Bieber | 343 Gold Street, Brooklyn | Apt 4001 | Hello World! | /users/photo/1.png | 0 | 2019-11-01 12:00:00 |
| | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL |

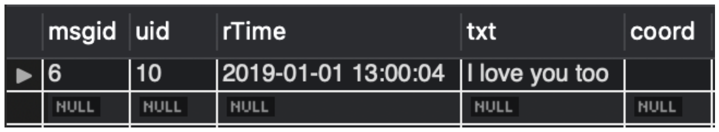
**Content Posting**

-- User starts a thread by posting an initial message



| | msgid | author | rRange | mtime | title | sub | txt | coord |
|---|---|---|---|---|---|---|---|---|
| ▶ | 6 | 9 | 0 | 2019-01-01 12:00:04 | Hi | Life | I love you | (40.7657, -73.9761) |
| | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL |

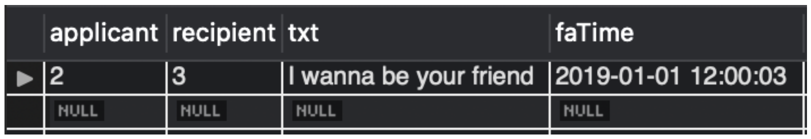
-- User replies to a message



| | msgid | uid | rTime | txt | coord |
|---|---|---|---|---|---|
| ▶ | 6 | 10 | 2019-01-01 13:00:04 | I love you too | |
| | NULL | NULL | NULL | NULL | NULL |

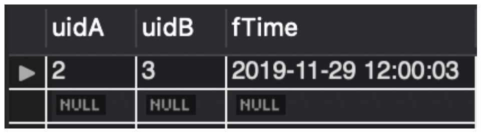
**Friendship**

-- User applies for friendship



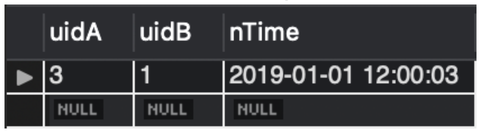
| | applicant | recipient | txt | faTime |
|---|---|---|---|---|
| ▶ | 2 | 3 | I wanna be your friend | 2019-01-01 12:00:03 |
| | NULL | NULL | NULL | NULL |

-- User accepts friend request



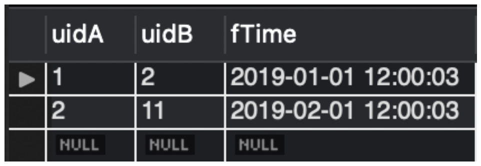
| | uidA | uidB | fTime |
|---|---|---|---|
| ▶ | 2 | 3 | 2019-11-29 12:00:03 |
| | NULL | NULL | NULL |

-- User adds new neighbor



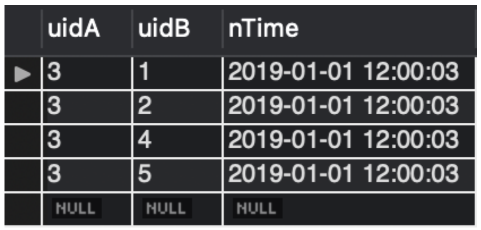
| uidA | uidB | nTime |
|---|---|---|
| 3 | 1 | 2019-01-01 12:00:03 |
| NULL | NULL | NULL |

-- All current friends to user whose uid = 2



| uidA | uidB | fTime |
|---|---|---|
| 1 | 2 | 2019-01-01 12:00:03 |
| 2 | 11 | 2019-02-01 12:00:03 |
| NULL | NULL | NULL |

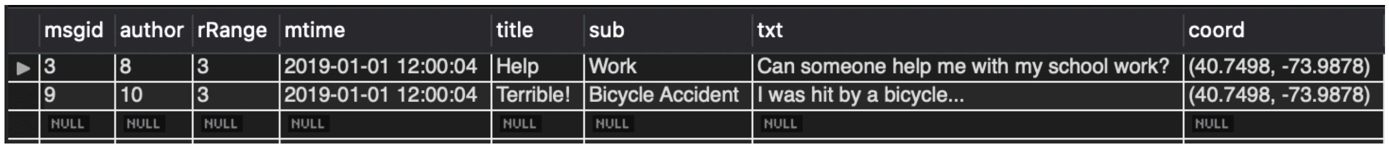
-- All current neighbors to uid = 3



| uidA | uidB | nTime |
|---|---|---|
| 3 | 1 | 2019-01-01 12:00:03 |
| 3 | 2 | 2019-01-01 12:00:03 |
| 3 | 4 | 2019-01-01 12:00:03 |
| 3 | 5 | 2019-01-01 12:00:03 |
| NULL | NULL | NULL |

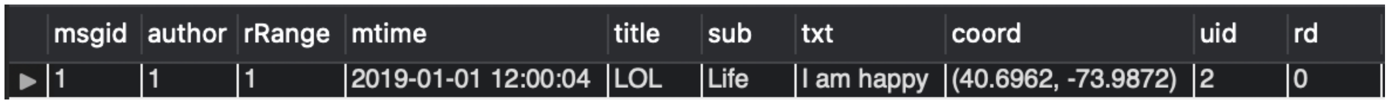
**Browse And Search Messages**

-- List all threads in a user(9)'s block feed that have new messages since the last time the user accessed



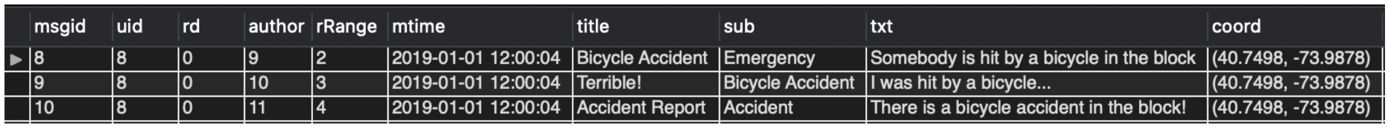
| msgid | author | rRange | mtime | title | sub | txt | coord |
|---|---|---|---|---|---|---|---|
| 3 | 8 | 3 | 2019-01-01 12:00:04 | Help | Work | Can someone help me with my school work? | (40.7498, -73.9878) |
| 9 | 10 | 3 | 2019-01-01 12:00:04 | Terrible! | Bicycle Accident | I was hit by a bicycle... | (40.7498, -73.9878) |
| NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL |

-- List all threads in user(2)'s friend feed that have unread messages



| msgid | author | rRange | mtime | title | sub | txt | coord | uid | rd |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 2019-01-01 12:00:04 | LOL | Life | I am happy | (40.6962, -73.9872) | 2 | 0 |

-- List all messages containing the words "bicycle accident" across all feeds that user(8) can access



| msgid | uid | rd | author | rRange | mtime | title | sub | txt | coord |
|---|---|---|---|---|---|---|---|---|---|
| 8 | 8 | 0 | 9 | 2 | 2019-01-01 12:00:04 | Bicycle Accident | Emergency | Somebody is hit by a bicycle in the block | (40.7498, -73.9878) |
| 9 | 8 | 0 | 10 | 3 | 2019-01-01 12:00:04 | Terrible! | Bicycle Accident | I was hit by a bicycle... | (40.7498, -73.9878) |
| 10 | 8 | 0 | 11 | 4 | 2019-01-01 12:00:04 | Accident Report | Accident | There is a bicycle accident in the block! | (40.7498, -73.9878) |

# Part II - Web based User Interface

To be continued