Alpha Messenger

By Vasanth, Devi Sri Charan, Rishabh, Akhil

Problem Statement:

How might we enable users to send messages to a person/s by saving their contact number over the internet?

Scope of the Project:

Using alpha messenger, users can send messages to other users over the internet. Can search for other users to text and see who is active currently. They can create groups and send messages to more than one user simultaneously.

Stakeholders:

Application Stakeholders:

Users, owner/administrator, developers, technical team, design team, business analysis team

Database Stakeholders:

Administrator, developers, technical team, business analysis team

Entities and their attributes:

All attributes are atomic and single-valued unless stated otherwise. Assuming that there is at least one user, participants, status_views, and group_message_views are weak entities.

1. user

user_id	Int Primary Key Not Null
first_name	varchar(30) not null
last_name	varchar(30)

phone_no	varchar(10) not null unique
password	varchar(30) not null
is_active	varchar(30)
about	varchar(1000)
profile_pic	varchar(1000)
joined_at	datetime

2. groups_

group_id	Int Primary Key Not Null
group_name	varchar(30) not null
admin_id	int not null
created_at	datetime

3. direct_message

msg_id	Int Primary Key Not Null

msg_body	varchar(1000) not null
sender_id	int not null
receiver_id	int not null
timestamp	datetime
is_read	varchar(30)

4. group_message

msg_id	Int Primary Key Not Null
msg_body	varchar(1000) not null
sender_id	int not null
group_id	int not null
timestamp	datetime

5. group_message_views

msg_id	Int Primary Key Not Null
msg_body	varchar(1000)

	not null
sender_id	int not null
viewer_id	int not null
is_seen	varchar(30)
group_id	int not null
timestamp	datetime

6. participants

group_id	Int Not null
user_id	int not null

7. status

status_id	int primary not null
posted_by	int not null
posted_time	datetime

8. status_views

status_id	int

	not null
user_id	int not null
seen_time	datetime

9. contacts

contact_id	int not null
user_id	int not null

10. calls

call_id	int primary not null
call_duration	time
caller_id	int not null
callee_id	int not null
timestamp	datetime

Relationships:

Relationships are partial except stated otherwise

1) sends - between the **user** and **direct_message**

   Type: one to many

Participation: participation of **user** is total

Attributes: user_id, **group_id**

2) makes - between the **user** and **call**

   Type: one to many

   Attributes: user_id, **call_id**

3) has - between the **user** and **contacts**

   Type: one to many

   Attributes: group_id, **msg_id**

4) posts - between the **user** and **status**

   Type: one to one

   Attributes: user_id, **status_id**

5) has - between **status** and **status_views**

   Type: one to many

   Attributes: status_id

6) joins - between **user** and **group_chat**

   Type: many to many

   Attributes: user_id, **group_id**

7) has - between **group_chat** and **participants**

   Type: many to many

   Attributes: group_id, **group_id**

8) send - between **user** and **group_message**

   Type: one to many

   Attributes: msg_id, **group_id**

9) has - between **group_message** and **group_message_views**

   Type: many to one

   Attributes: msg_id

Relation Schemas:

1) **user** (user_id, first_name, last_name, password, phone_no, status, joined_at)

   Candidate Keys: {user_id, phone_no, password}

2) **direct_message** (msg_id, msg_body, sender_id, receiver_id, timestamp, is_read)

   Candidate Keys: { msg_id}

3) **groups**_(group_id, admin_id, group_name, created_at)

   Candidate Keys: {group_id}

4) **group_message** (msg_id, msg_body, sender_id, group_id, timestamp)

   Candidate Keys: {msg_id}

5) **group_message_views** (msg_id, msg_body, sender_id, viewer_id, is_seen, group_id, timestamp)

   Candidate Keys: {(msg_id, group_id)}

6) **participants** (user_id, group_id)

   Candidate Keys: {(user_id, group_id) }

7) **status** (status_id, posted_by, posted_time)

   Candidate Keys: {status_id}

8) **status_views** (status_id, user_id, seen_time)

   Candidate Keys: {(status_id,user_id)}

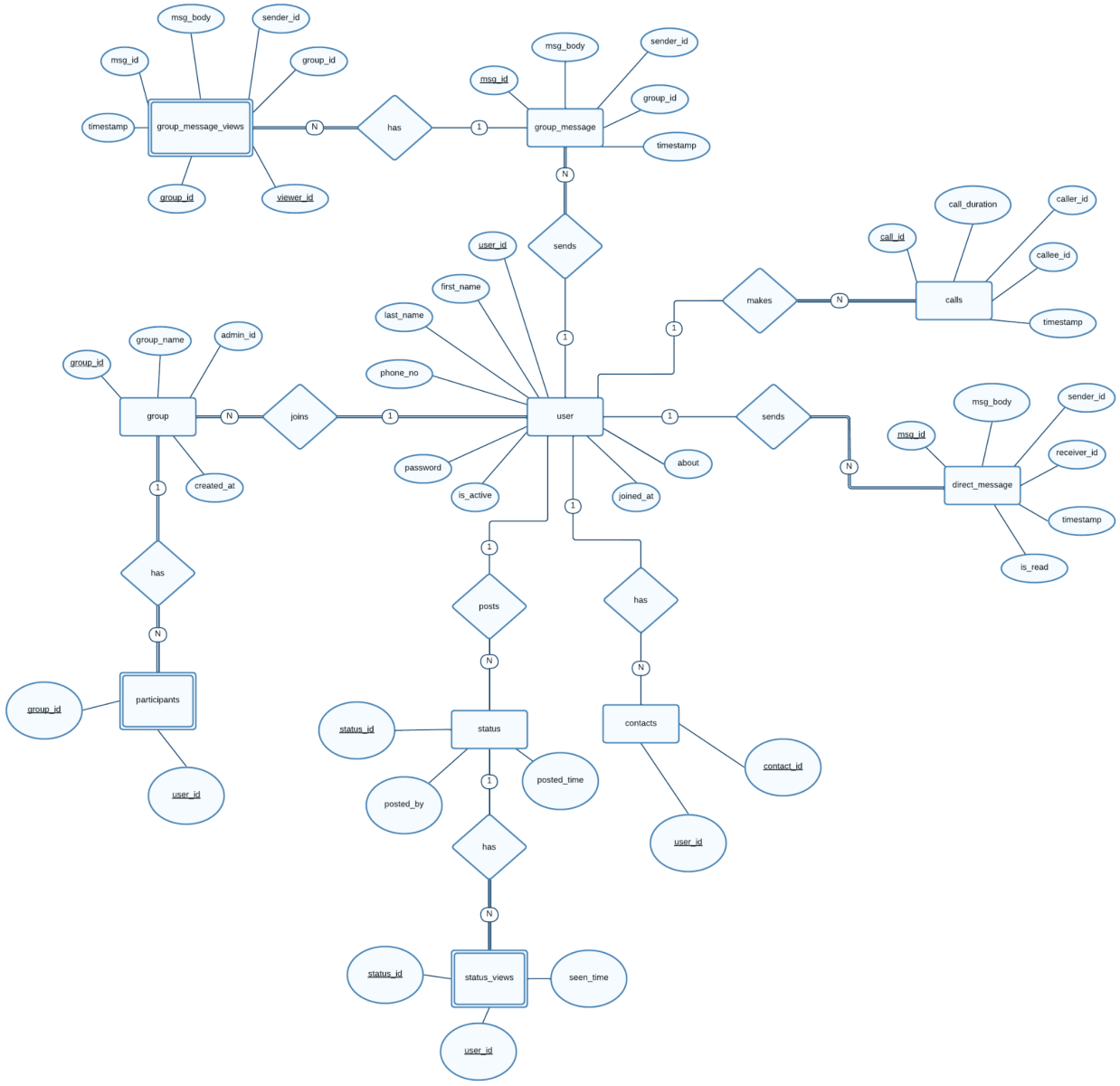
9) **contacts** (user_id, contact_id)

   Candidate Keys: {(user_id, contact_id)}

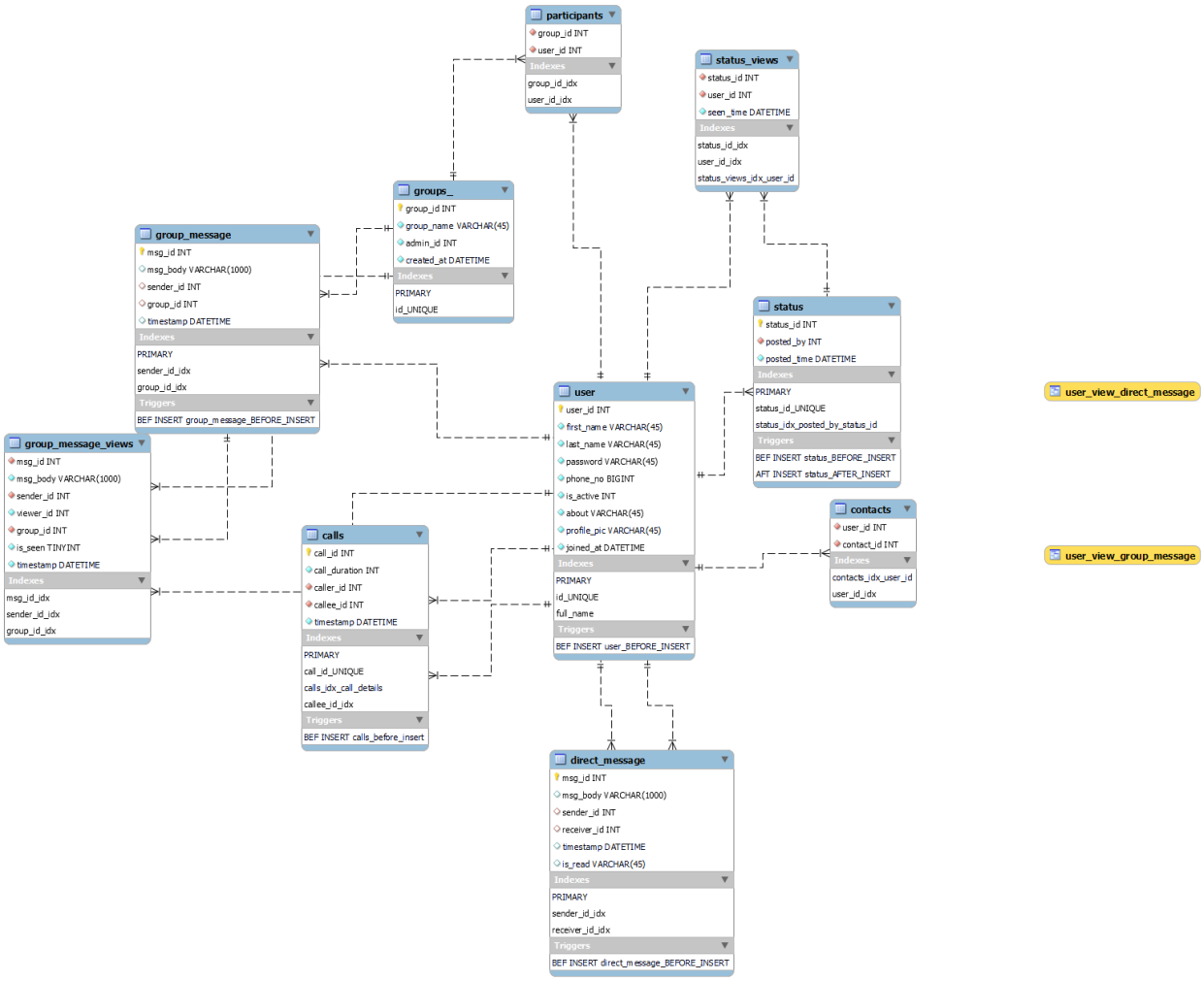
10) **calls** (call_id, call_duration, caller_id, callee_id, timestamp)

Candidate Keys: {call_id}

ER Diagram:



Schema:



Views :

User:

1. **direct_message** (msg_id, msg_body, sender_id, timestamp, is_read)
2. **group_message** (msg_id, msg_body, sender_id, group_id, timestamp)
3. **group_message_views** (msg_id, msg_body, sender_id, viewer_id, is_seen, group_id, timestamp)
4. **groups_** (group_id, admin_id, group_name, created_at)
5. **participants** (user_id, group_id)

6. **status** (status_id, posted_by, posted_time)
7. **status_views** (status_id, user_id, seen_time)
8. **contacts** (user_id, contact_id)
9. **calls** (call_id, call_duration, caller_id, callee_id, timestamp)

Explanation:

The user can view the different groups and contacts they have. They can also view the calls and messages of individual contacts and their statuses.

Database Administrator:

Everything

Views for user:

1. ```
   CREATE
         algorithm = undefined
         definer = `root`@`localhost`
         sql security definer
   VIEW `user_view_direct_message` AS
         SELECT    `USER`.`user_id`             AS `user_id`,
                   `direct_message`.`msg_body`  AS `msg_body`,
                   `direct_message`.`timestamp` AS `timestamp`,
                   `direct_message`.`is_read`   AS `is_read`
         FROM     ( `direct_message`
         JOIN
         `USER`
         ON (((`USER`.`user_id` = `direct_message`.`sender_id`)
                           OR
         (`USER`.`user_id` = `direct_message`.`receiver_id` ) )) )
         GROUP BY `USER`.`user_id`
   ```

2. ```
   CREATE algorithm = undefined definer = `root`@`localhost` sql
   security definer
   VIEW `user_view_group_message` AS
   SELECT    `USER`.`user_id`             AS `user_id`,
             `group_message`.`msg_body`   AS `msg_body`,
             `group_message`.`group_id`   AS `group_id`,
             `group_message`.`timestamp`  AS `timestamp`
   ```

```
FROM     ( `group_message`
JOIN
`USER`
ON ((`USER`.`user_id` = `group_message`.`sender_id`)) )
GROUP BY `USER`.`user_id`
```

Roles:

1. `CREATE role administrator`
2. `CREATE role USER`

Grants :

1. `GRANT SELECT ON user_view_direct_message TO USER;`
2. `GRANT SELECT ON user_view_group_message TO USER;`

Explanation:

The user is granted to view messages and statuses of other users and groups who are a part of their chat.

Indexing:

1. ```
   ALTER TABLE `user`
     ADD INDEX `full_name` (`first_name`, `last_name`);
   ```
2. ```
   ALTER TABLE `calls`
         ADD INDEX `calls_idx_call_details` (`caller_id`,
         `callee_id`, `timestamp`);
   ```
3. ```
   ALTER TABLE `contacts`
     ADD INDEX `contacts_idx_user_id` (`user_id`);
   ```
4. ```
   ALTER TABLE `status`
     ADD INDEX `status_idx_posted_by_status_id` (`posted_by`, `status_id`);
   ```
5. ```
   ALTER TABLE `status_views`
     ADD INDEX `status_views_idx_user_id` (`user_id`);
   ```
6. ```
   ALTER TABLE `calls`
         ADD INDEX `calls_idx_call_duration` (`call_duration`);
   ```

Triggers :

1. we trigger when password is not more than 8 letters

```
CREATE definer=`root`@`localhost` TRIGGER `user_before_insert`
before INSERT
on `USER` FOR each row
BEGIN
DECLARE error_msg varchar(255);
SET error_msg = ('password should greater than 8 letters');
IF Length(new.password) < 8 then
signal sqlstate '45000'
SET message_text = error_msg;
ENDIF;
END
```

2. trigger timestamp when user is sending a direct message

```
CREATE definer = CURRENT_USER TRIGGER
`alpha`.`direct_message_before_insert`
before INSERT
on `direct_message` FOR each row
BEGIN
SET new.timestamp = CURRENT_TIMESTAMP();
END
```

3. trigger timestamp when user is is sending a group message

```
CREATE definer=`root`@`localhost` TRIGGER
`group_message_before_insert`
before INSERT
on `group_message` FOR each row
BEGIN
SET new.timestamp = CURRENT_TIMESTAMP();
END
```

4. trigger timestamp when user is calling

```
CREATE definer=`root`@`localhost` TRIGGER `calls_before_insert`
before INSERT
on `calls` FOR each row
BEGIN
SET new.timestamp = CURRENT_TIMESTAMP();
```

```
END
```

5. trigger timestamp when user is posting a status

```
CREATE  definer=`root`@`localhost`  TRIGGER  `status_before_insert`
before INSERT
on  `status`  FOR each row
BEGIN
SET new.posted_time = CURRENT_TIMESTAMP();
END
```

6. trigger(delete) when status duration has passed 24 hours

```
CREATE  definer=`root`@`localhost`  TRIGGER  `status_after_insert`
after INSERT
on  `status`  FOR each row
BEGIN
DELETE
FROM   status
WHERE  (status.posted_time - currenttimestamp() > 24);
END
```

SQL Queries :

1. Find no.of messages sent in each group before “2022-01-01 00:00:00”

```
SELECT g.group_name,
  Count(gm.msg_id) AS "No.of Messages"
FROM   groups_ g
       JOIN group_message gm
         ON gm.group_id = g.group_id
WHERE  timestamp < '2022-01-01 00:00:00'
GROUP  BY g.group_id;
```

2. Find user details who has viewed the status posted by a particular user

```
SELECT user_id,
       first_name,
       last_name
FROM   USER
WHERE  user_id IN (SELECT user_id
                   FROM   status_views
                   WHERE  status_id IN (SELECT status_id
                                        FROM   status
                                        WHERE  posted_by = 77));
```

3. Find the group name and participants with maximum no.of participants

```
SELECT group_id,
       Count(user_id) AS NoOfParticipants
FROM   participants
GROUP  BY group_id
```

```
HAVING Count(user_id) = (SELECT Max(noofparticipants) AS
highestNoOfParticipants
                         FROM   (SELECT group_id,
                                        Count(user_id) AS NoOfParticipants
                                 FROM   participants
                                 GROUP  BY group_id) AS T);
```

4. Find the user details whose call duration is maximum

```
SELECT first_name,
       last_name,
       phone_no
FROM   USER
WHERE  user_id = (SELECT caller_id
                  FROM   calls
                  WHERE  call_duration = (SELECT Max(call_duration)
                                          FROM   calls))
```

5. Find the user name who has maximum contacts

```
SELECT first_name,
       last_name
FROM   USER
WHERE  user_id IN (SELECT user_id
                   FROM   contacts
                   GROUP  BY user_id
                   HAVING Count(contact_id) = (SELECT
                          Max(noofcontacts) AS highestNoOfContacts
                                               FROM   (SELECT user_id,
                                                              Count(contact_id)
                                                              AS
                                                              NoOfContacts
                                                       FROM   contacts
                                                       GROUP  BY user_id) AS T));
```

6. Find the number of direct messages sent by each user

```
SELECT user_id,
       first_name AS `Sender's first name`,
       last_name  AS `Sender's last name`,
       Count(*)   AS `No. of direct messages send`
FROM   user
       INNER JOIN direct_message
               ON user_id = sender_id
GROUP  BY user_id,
          first_name,
          Last_name;
```

7. Find the callee's phone numbers of which users called to with date?

```
SELECT u1.user_id      AS "user",
       u2.phone_no     AS "called to",
       Date(timestamp) AS "date"
FROM   calls
       JOIN USER u1
         ON caller_id = u1.user_id
```

```
        JOIN USER u2
          ON callee_id = u2.user_id
GROUP  BY Date(timestamp)
ORDER  BY u1.user_id
```

8. Find the group details with participants

```
SELECT p1.group_id                          AS "group id",
       Group_concat(DISTINCT p2.user_id) AS "group participants"
FROM   participants p1
       JOIN participants p2
         ON p1.group_id = p2.group_id
GROUP  BY p1.group_id;
```

9. Find the user ids who has seen the status posted by user 97

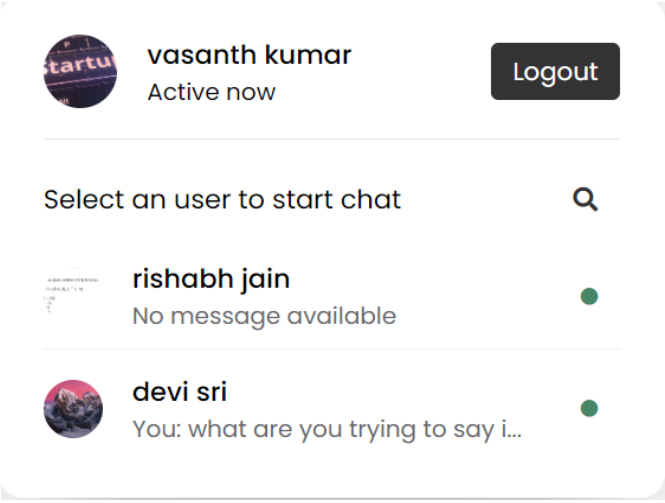
```
SELECT user_id, seen_time
                  FROM   status_views
                  WHERE  status_id IN (SELECT status_id
                                       FROM   status
                                       WHERE  posted_by = 77)
```

10. Find the users who created their account in 2021

```
SELECT Count(user_id)
FROM   USER
WHERE  Year(joined_at) = 2021;
```

Embedded SQL Queries:

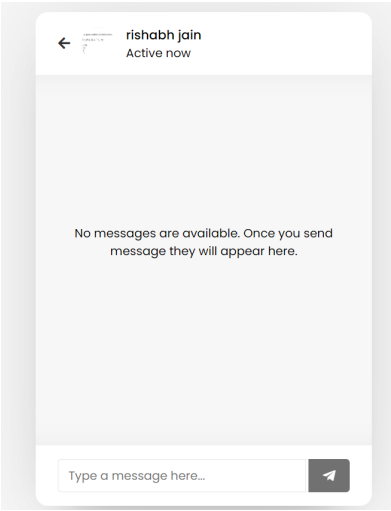
1. When the users log in to the website and send/receive messages so to display what a user see before clicking the user to text to. For fetching the data i.e. messages sent by the user and received by a particular user we will be executing the below code in which first we execute the sql query of messages sent by both users. If the query output has no columns which means no messages are available and if the msg_body increase the lenght of 28 we will further show it as ….
Like “what happened today when you are……”
See the below image



code<>

```
WHILE($row = Mysqli_fetch_assoc($query)){
$sql2 = "SELECT *
FROM messages
WHERE (reciever_id = {$row['unique_id']}OR
sender_id = {$row['unique_id']})
AND (sender_id = {$outgoing_id}
OR reciever_id = {$outgoing_id})
ORDER BY msg_id
DESC limit 1";
$query2 = mysqli_query($conn, $sql2);
$row2 = mysqli_fetch_assoc($query2);
(mysqli_num_rows($query2) > 0) ? $result = $row2['msg_body'] : $result
="no message available";
(strlen($result) > 28) ? $msg_body = SUBSTR($result, 0, 28) . '...' :
$msg_body = $result;
```

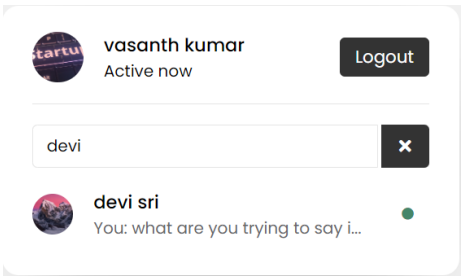
2. IN the below picture when the user is new to the website and two users want to message they open the chat if they are chatting for the first time then it displays that “NO MESSAGES ARE AVAILABLE. ONCE YOU SEND MESSAGE THEY WILL APPEAR HERE” for this we have to fetch the message details

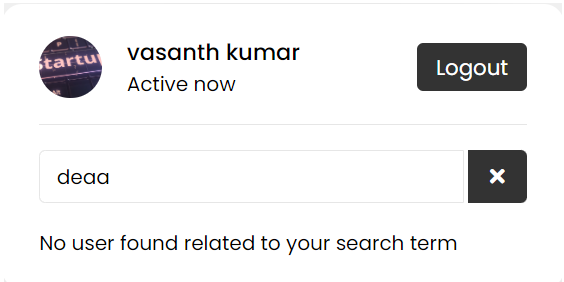


code<>

```
$sql = "SELECT *
FROM messages
LEFT JOIN users
ON users.unique_id = messages.sender_id
WHERE (sender_id = {$outgoing_id} AND reciever_id = {$incoming_id})
OR (sender_id = {$incoming_id} AND reciever_id = {$outgoing_id}) ORDER
BY msg_id";
$query = mysqli_query($conn, $sql);
IF(Mysqli_num_rows($query) > 0)
{WHILE($row = mysqli_fetch_assoc($query)){
      IF($row['sender_id'] === $outgoing_id){
            $output .= ''. $row['msg_body'] .'';
            } } }
ELSE{ $output .= 'No messages are available. Once you send message
they will appear here.
'; }
```

3. For searching for any user if they are available in the website or not





code<>

```
$outgoing_id = $_session['unique_id'];
$searchTerm = mysqli_real_escape_string($conn, $_post['searchTerm']);
$sql = "SELECT *
FROM users
WHERE NOT unique_id = {$outgoing_id}
AND (first_name LIKE '%{$searchTerm}%'
OR last_name LIKE '%{$searchTerm}%') ";
$output = "";
$query = mysqli_query($conn, $sql);
IF(Mysqli_num_rows($query) > 0){
include_once "data.php";
}
ELSE{
$output .= 'No user found related to your search term';
}
echo $output;
```

4. If there are no any users to chat when user is the first person to open website it will show that no users are available to chat

code<>

```
$sql = "SELECT * FROM users WHERE NOT unique_id = {$outgoing_id} ORDER
BY user_id DESC";
$query = mysqli_query($conn, $sql);
$output = "";
IF(Mysqli_num_rows($query) == 0){
$output .= "No users are available to chat";
}
elseif(mysqli_num_rows($query) > 0){
include_once "data.php";
}
echo $output;
```

Query optimization :

1. In many cases, an EXISTS subquery with a correlated condition will perform better than a non correlated IN subquery.

```
SELECT USER.user_id,
 USER.first_name,
 USER.last_name
FROM   USER
WHERE  EXISTS (SELECT 1
               FROM   status_views
```

```
                WHERE  ( EXISTS (SELECT 1
                                 FROM   status
                                 WHERE  ( status.posted_by = 77 )
                                        AND ( status_views.status_id =
                                              status.status_id
                                            )) )
                       AND ( USER.user_id = status_views.user_id ))
```

2. By default, the database sorts all 'GROUP BY col1, col2,... queries as if you specified 'ORDER BY col1, col2,... in the query as well. If a query includes a GROUP BY clause but you want to avoid the overhead of sorting the result, you can suppress sorting by specifying 'ORDER BY NULL'.

```
SELECT USER.first_name,
       USER.last_name
FROM   USER
WHERE  EXISTS (SELECT 1
               FROM   contacts
               WHERE  ( USER.user_id = contacts.user_id )
               GROUP  BY contacts.user_id
               HAVING Count(contacts.contact_id) = (SELECT
                      Max(noofcontacts) AS highestNoOfContacts
                                                    FROM
                      (SELECT
                              contacts.user_id,
                              Count(contacts.contact_id) AS NoOfContacts
                                                                 FROM
contacts
                                                          GROUP  BY
                      contacts.user_id) AS
                      T)
               ORDER  BY NULL)
```

3. We advise against using subqueries as they are not optimized well by the optimizer. Therefore, it's recommended to join a newly created temporary table that holds the data, which also includes the relevant search index.

```
SELECT USER.first_name,
       USER.last_name,
       USER.phone_no
FROM   USER
WHERE  USER.user_id = (SELECT calls.caller_id
                       FROM   calls
                       WHERE  calls.call_duration = (SELECT
                              Max(calls.call_duration)
                                                     FROM   calls))
```

4. In many cases, an EXISTS subquery with a correlated condition will perform better than a non correlated IN subquery.

```
SELECT USER.first_name,
USER.last_name
```

```
FROM   USER
WHERE  EXISTS (SELECT 1
               FROM   contacts
               WHERE  ( USER.user_id = contacts.user_id )
               GROUP  BY contacts.user_id
               HAVING Count(contacts.contact_id) = (SELECT
                      Max(noofcontacts) AS highestNoOfContacts
                                                FROM
                      (SELECT
                              contacts.user_id,
                              Count(contacts.contact_id) AS NoOfContacts
                                                    FROM   contacts
                                                    GROUP  BY
                      contacts.user_id) AS
                      T)
               ORDER  BY NULL)
```