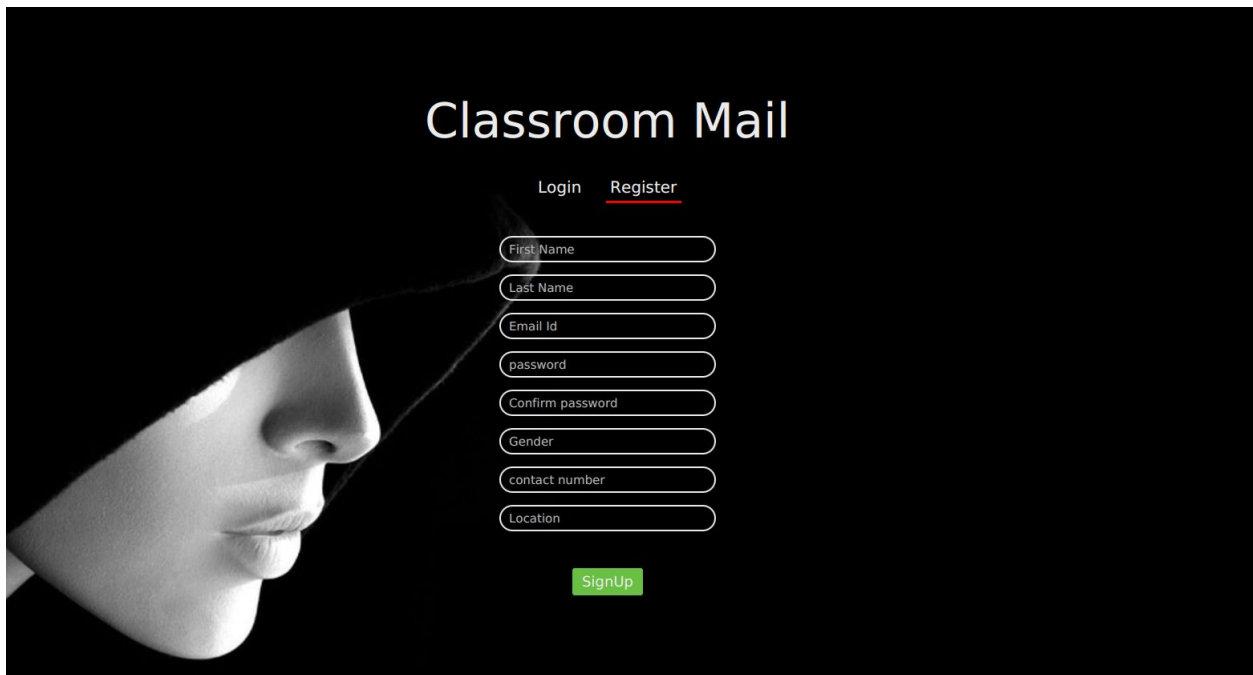


Database Management System

# Course Project

## CSN-361

## Classroom Mail

A screenshot of a web application interface for 'Classroom Mail'. The background is black with a white profile of a person wearing a hijab on the left. The title 'Classroom Mail' is centered at the top in white. Below it are two links: 'Login' and 'Register', with 'Register' underlined in red. A series of white rounded rectangular input fields are stacked vertically, labeled 'First Name', 'Last Name', 'Email Id', 'password', 'Confirm password', 'Gender', 'contact number', and 'Location'. A green 'SignUp' button is positioned below the last input field.

Classroom Mail

Login Register

First Name

Last Name

Email Id

password

Confirm password

Gender

contact number

Location

SignUp

### Team Members:

15114013	Anmol Anand
15114020	Chirag Maheswari
15114046	Mohit Jindal
15114048	Nitish Bansal
15114061	Sahil Grover

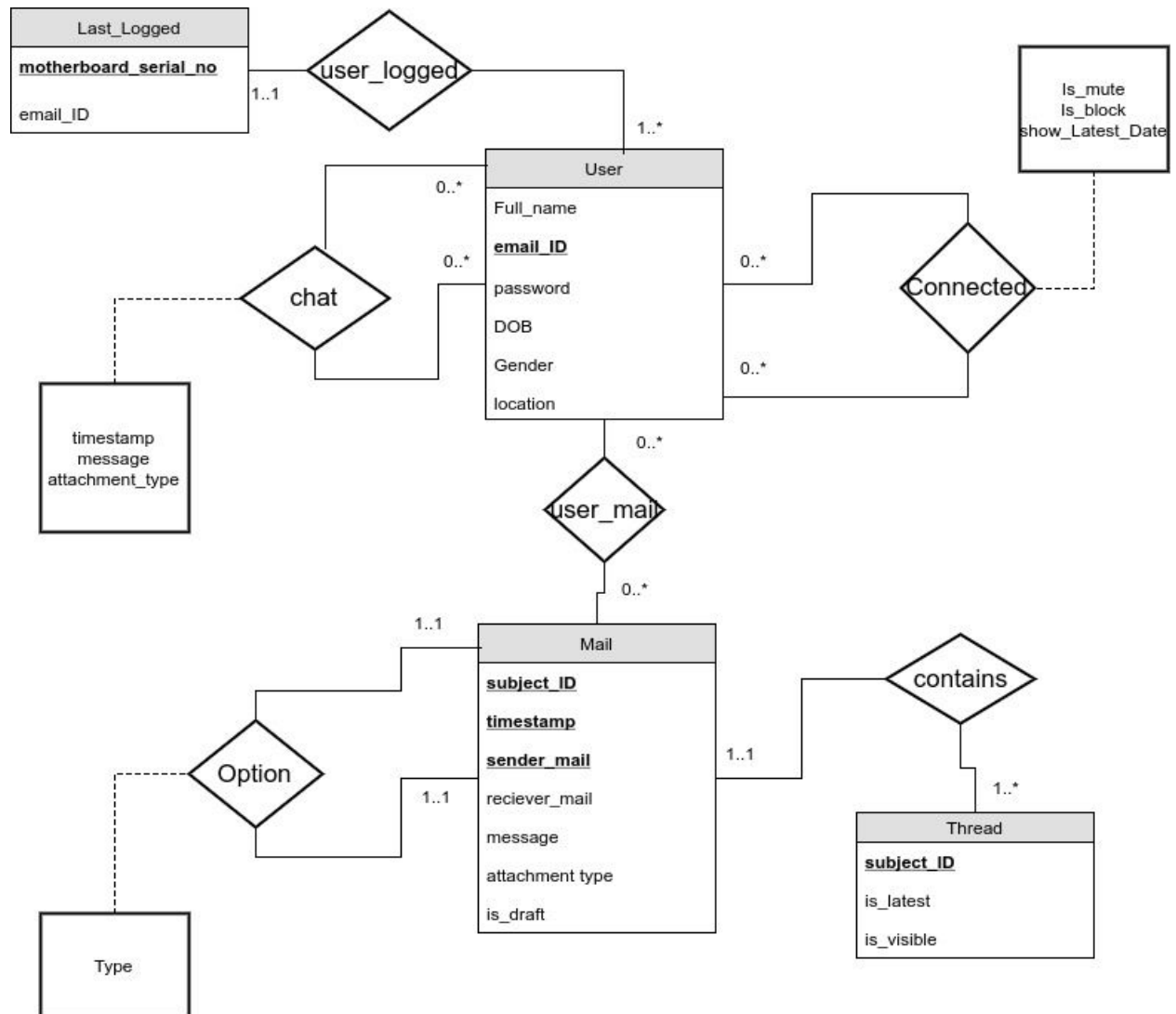
# Introduction

Classroom Mail app contains the functionality of sending mails to other users. The users can send/recieve mails, delete them, mark as important and variety of other features. Not only this it includes an additional feature of chat too. Friends can be added in our contact list and we can just chat with them as hangouts. We have worked on the sql database going deep into writing queries and then changing the corresponding User Interface. Some of queries along with the ER diagram and schema diagram is follows below.

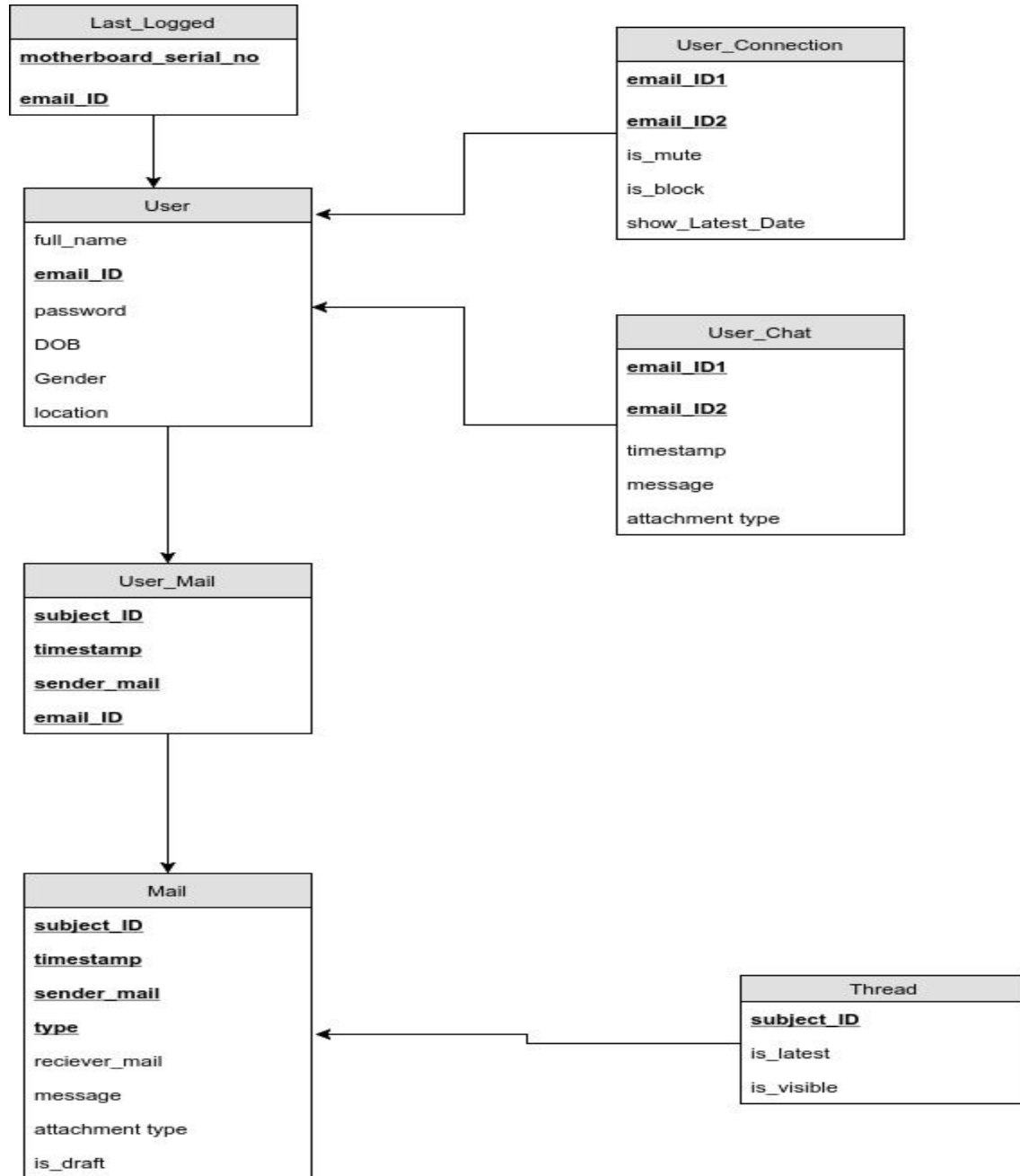
# **ASSUMPTIONS:**

1. A person CANNOT send multiple messages/mails at same point of time.
2. Motherboard serial numbers are unique
3. Reply mails and Forward mails are considered to be in same thread as their parent.
4. Each time a user views a mail, a new tuple is added with new timestamp (even if user had read that mail earlier).

# ER Diagram



# Schema Diagram



# Functional Dependencies & Candidate Keys

## Functional Dependencies

1. **User** (email\_ID, full\_name, password, DOB, Gender, location)
  - email\_ID -> full\_name, password, DOB, Gender, location
2. **Last\_Logged** (email\_ID, motherboard\_serial\_no)
  - No FD
3. **User\_Connection** (email\_ID1, email\_ID2, is\_mute, is\_block, show\_latest\_date)
  - {email\_ID1, email\_ID2} -> is\_mute, is\_block, show\_latest\_date
4. **User\_Chat** (email\_ID1, email\_ID2, timestamp, message, attachment\_type)
  - {email\_ID1, email\_ID2} -> timestamp, message, attachment\_type
5. **User\_mail** (subject\_ID, timestamp, sender\_mail, email\_ID)
  - No FD
6. **Mail** (subject\_ID, timestamp, sender\_mail, type, receiver\_mail, message, attachment\_type, is\_draft)
  - {subject\_ID, timestamp, sender\_mail, type} -> receiver\_mail, message, attachment\_type, is\_draft
7. **Thread** (subject\_ID, is\_latest, is\_visible)
  - subject\_ID -> is\_latest, is\_visible

# Candidate Keys

1. **User:** {email\_ID}
2. **Last\_Logged:** {email\_ID, motherboard\_serial\_no}
3. **User\_Connection:** {email\_ID1, email\_ID2}
4. **User\_Chat:** {email\_ID1, email\_ID2}
5. **User\_mail:** {subject\_ID, timestamp, sender\_mail, email\_ID}
6. **Mail:** {subject\_ID, timestamp, sender\_mail, type}
7. **Thread:** {subject\_ID}

## Minimal Cover

Since there is no FD in any relation, in which LHS is not a candidate key, the above written dependencies are also the minimal cover for the schema.

## 1NF, 2NF, 3NF, BCNF 4NF, 5NF?

**Observation:** In all our tables, we have FDs of type  $X \rightarrow A$ ,  $X \in$  Candidate Key.

### 1. 1NF:

- a. Since all our columns are atomic, the tables are in 1NF.

### 2. 2NF:

- a. Since  $X \in$  Candidate Key, no non-prime attribute can be dependent on a subset of candidate key.

### 3. 3NF:

- a. Since  $X \in$  Candidate Key, there cannot be a functional dependency from non-prime attribute to non-prime attribute

### 4. BCNF:

- a. Since  $X \in$  Candidate Key  $\in$  Super Key, this means all our tables are in **BCNF**.

Since there are no multi-valued dependencies or Join Dependencies, the schema is in **5NF**



# Working of the App and SQL queries

1. When a user starts the ClassroomMail app:

a. A query is made to check whether any user is already logged in from this machine or not.

b. The (motherboard serial number)a.k.a(ID) is passed to the query.

**Select \* from currentuser where id =  
“motherboard\_serial\_no”**

c. If any user already logged in, the main page for that user is opened. Following two queries are made to the database to get the name of user:

**Select DISTINCT(receiver) as mailID from chat where  
sender = ‘emailID’**

**Select DISTINCT(sender) as mailID from chat where  
receiver = ‘emailID’**

d. Otherwise, the login/register page is opened.

2. When a user clicks on **log in**:

a. The email id is validated and a query is made to the database.

b. User emailID and password are passed to the query

**Select \* from userdetail where emailID = emailID and  
password = password**

- c. Now, the profile page for the user is opened.
- 3. When the user clicks on **Register**:
  - a. The user is asked to fill the following information about himself/herself:
    - i. First Name
    - ii. Last Name
    - iii. Password
    - iv. EmailID
    - v. Contact
    - vi. Gender
    - vii. Location
  - b. The user information is validated and queries are made to the database to insert the new user information:  
**Insert into userdetail (FirstName, LastName, password, emailID, Contact, Gender, Location) Values(FirstName, LastName, password, emailID, Contact, Gender, Location);**  
**Insert into userdetail (ID, FirstName, emailID) Values(ID, FirstName, emailID);**
  - c. Now, the profile page for the user is opened.
- 4. Search Mails: This takes a text value input from the user and searches for that value in user mails.
  - a. Following query is made to the database to fetch the result of search query

```

Select * from (select subjectID, messageTimestamp,
message from mails where (receiveMail like
'%keyword%' or senderMail like '%keyword%' or
message like '%keyword%')) and (senderMail =
'mailID' or receiverMail = 'mailID') group by subjectID
desc)
UNION
(select subjectID, draftTimestamp as
messageTimestamp, draftMessage as message from
subjectdetails where mailID = 'mailID' and
(draftMessage like '%keyword%' or draftRecipient like
'%keyword%'))
UNION
(select subjectID, messageTimestamp, message from
mails where mails.subjectID in (select subjectID from
subjectdetails where mailID = 'mailID' and isDraft =
'False' and subjectName like '%keyword%'))
As T group by subjectID asc 'filter';

```

There is option on the right side of the search bar. User select the options in which order to show the mails. This option value is the filter. (Order by timestamp asc or desc)

5. Compose Mail: A panel on the right side is opened where the user is asked to compose the mail. It have the following options:
  - a. **Recipient:** User is supposed to write the emailID of the recipient.
  - b. **Subject:** User is supposed to write the subject of the mail.
  - c. **Message:** User is supposed to write the contents of message in this field.

d. **Important:** If user selects this option, the mail is saved as an important mail in the database.

e. **Send:** Mails information is inserted in the tables: subjectDetails and mails.

i. Following are the queries made:

**Insert into mails (messageTimestamp, subjectID, senderMail, receiverMail, message) VALUES ();**

**Insert into subjectDetails (subjectID, subjectName, important, deleted, latestMessageRead, isDraft, draftMessage, draftRecipient, draftTimestamp) VALUES ();**

f. **SaveAsDraft:** If user clicks on this button, the mail is saved as draft instead of sending to the recipient.

i. The following queries are made to the database:

**UPDATE subjectdetails SET isDraft = 'true'  
WHERE ( subjectId = 'subjectID' AND mailId = 'mailID' );**

**UPDATE subjectdetails SET draftMessage = 'message' WHERE ( subjectId = 'subjectID' AND mailId = 'mailID' );**

**UPDATE subjectdetails SET draftReceipients = 'recipient' WHERE ( subjectId = 'subjectID' AND mailId = 'mailID' );**

**UPDATE subjectdetails SET draftTimestamp = 'timestamp' WHERE ( subjectId = 'subjectID' AND mailId = 'mailID' );**

g. **Forward:** This button is used to forward the latest message by timestamp to the recipient entered by the user.

i. **Select \* from mails where senderMail = mailID and timestamp >= all (select timestamp from mails where senderMail = mailID);**

6. **Add To Contacts:** A prompt is opened where user has to add the emailID of his/her friend. The emailID is first validated the database is searched for this emailID. If this emailID exists, now that emailID is shown in the contacts list of the user and the user can now direct chat with that second user.

a. Query to search the database, a variable user2 emailID is passed:

**Select firstName from userDetails where emailID = 'user2';**

b. Inserting the friends and user details into the chat table:

**Insert into chat (timestamp, sender, receiver, message) VALUES (timestamp, user1, user2, message);**

7. **Chat Box:** When the user clicks on any of his friends from contact list, a chat box is opened on the right side:

a. On sending a message following query is made to the database:

**Select \* from chat where ( sender = 'user1' and receiver = 'user2' ) or ( sender = 'user2' and receiver = 'user1' ) order by filter**

**Insert into chat (timestamp, sender, receiver, message)  
VALUES  
(timestamp, user1, user2, message);**

8. **Trash:** This contains the messages which are marked isDeleted as 'True'.

- a. Following query is made to fetch the deleted mails:  
**Select \* from mails where receiverID like '%mailID%' or senderID like '%mailID%' and delete = 'True' order by filter**

9. **Important:** This contains the messages which are marked as Important as 'True'.

- a. The following query is made to fetch the important mails:  
**Select \* from mails where receiverMail like '%mailID%' or senderMail like '%mailID%' where Important = 'True' order by filter;**

10. **Draft:** This contains the messages which are marked as IsDraft as 'True'.

- a. Following query is made to the database:  
**Select \* from subjectDetails where mailID = 'mailID' and isDraft = 'True' order by filter**

11. **Sent Mail:** Clicking on this option will show the sent messages:

- a. Following query is made to the database:

**Select \* from mails where senderMail like '%mailID%' order by filter**

12. **Inbox:** Clicking on this option will show all the non-deleted mails:

- a. Following query is made to fetch the mails:

**Select \* from mails where receiverMail like '%mailID%' order by filter**

13. **Update the important mail info:** To mark/unmark any mail as important, click the rectangular box ahead of the subject of the mail.

- a. The following update query is made to the database:

**Update subjectDetails SET important = 'true/false' where subjectID = 'subjectID' and mailID = 'mailID'**

14. **Logout:** When the user clicks on this button, the user info from current user logged in is removed.

- a. Following query is made to the database:

**Delete from currentUser where id = 'userID'**

# What we learnt

This project really did boost our skills to write and execute sql queries. We had never worked on such kind of project before. Connecting the tables with information received from the users by executing queries was really fascinating. Overall working on this project was indeed a huge learning curve for each of our member. Also each such team work project definitely uplifts your team work qualities.