CS 432 - DATABASES

ASSIGNMENT - 4

TEAM: OCTACORE

Assignment 4: DEPLOYING THE DBMS

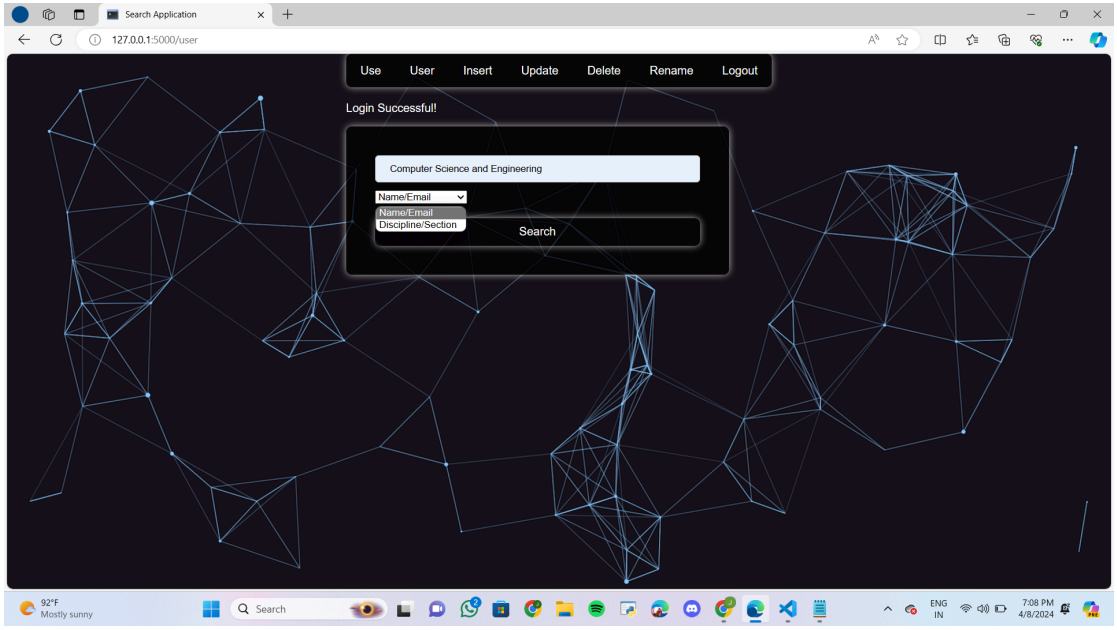
G1	G2
Sujith - 21110100	Jethru - 21110089
Uday - 21110084	Himasagar - 21110158
Keerthi - 21110176	Sudharshan - 21110218
Ajith - 21110045	Rudreshwar - 21110172
Pardheev Sai - 21110097	Vinay - 21110125

3. Tasks

3.1 Responsibility of G1:

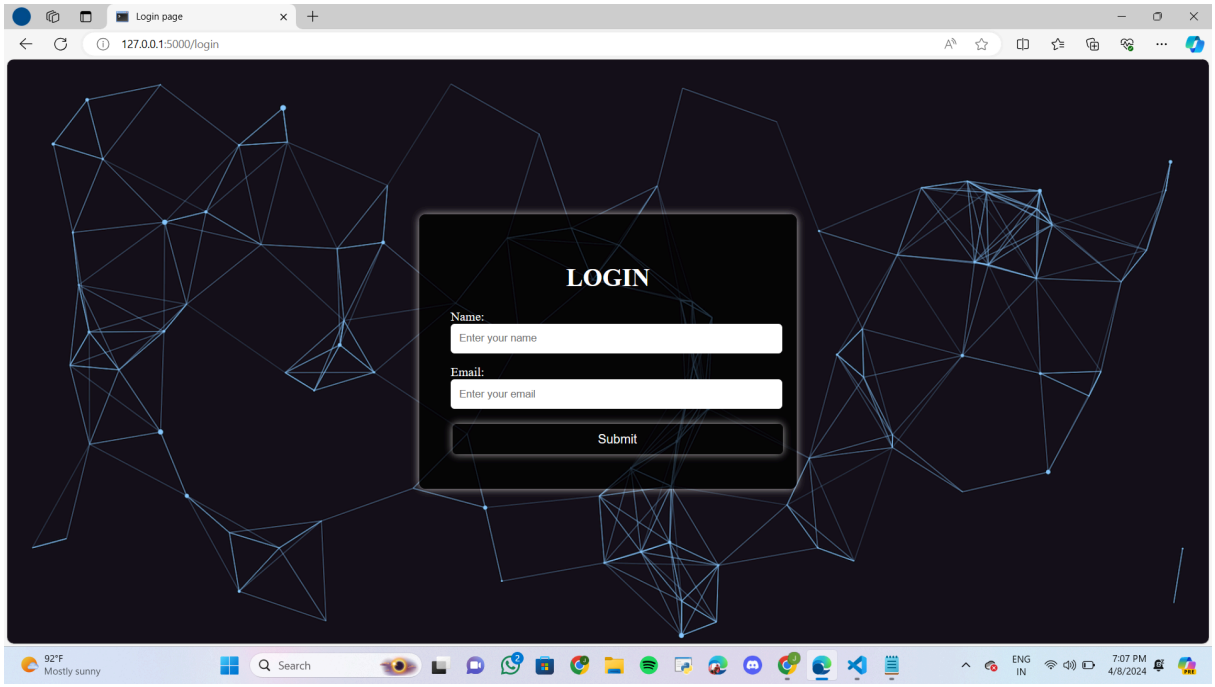
1. **Before the first feedback:**

i) Initially, there was no drop-down menu that listed all of the possible disciplines/sections on campus. Here's how it looked before the alterations were done.

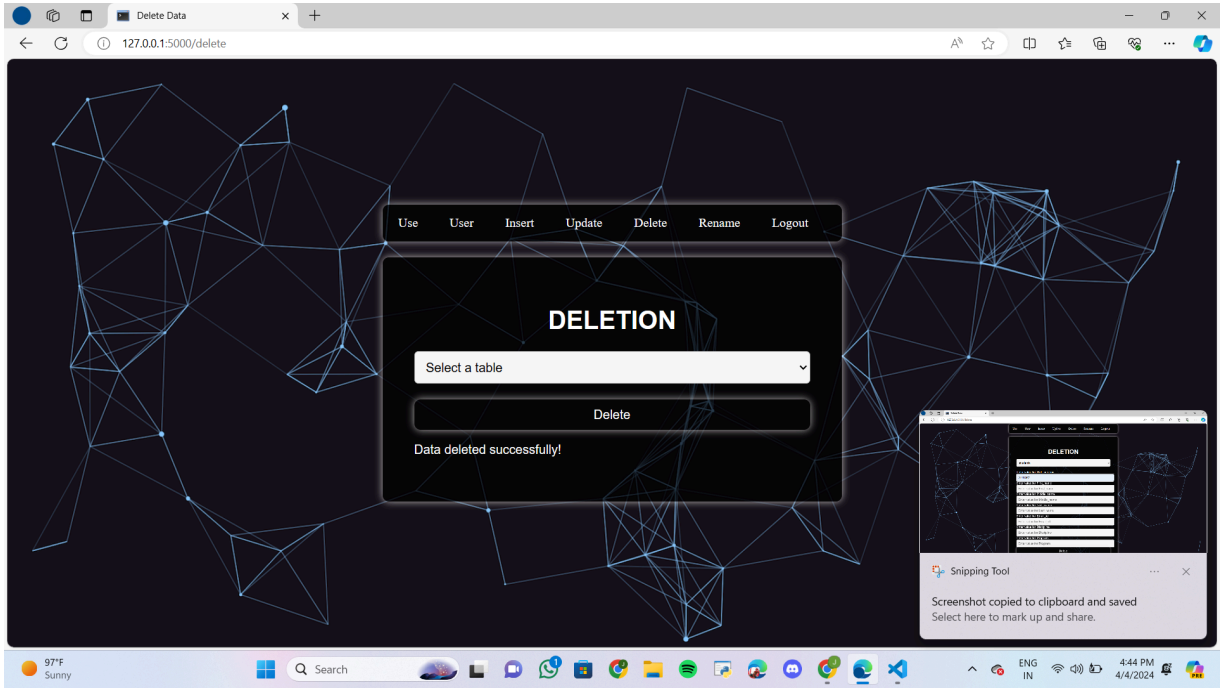


ii) There is no application page or route showing the relationships present in the database to the administrator. This will be useful to the administrator when he/she tries to insert or delete the data in the tables so they can easily sort the way of inserting or deleting values into different tables. First into the parent entity next to the child. We added a new page called Relations to the application, which can be seen below in the changes after the first feedback 2nd point.

iii) The earlier login page does not have a password section for logging in. Only name and email were enough. Now, we added a password field to the login page. So, email and password are required for login and registering, as well as name, email, and password. Image of login page showing before the changes:

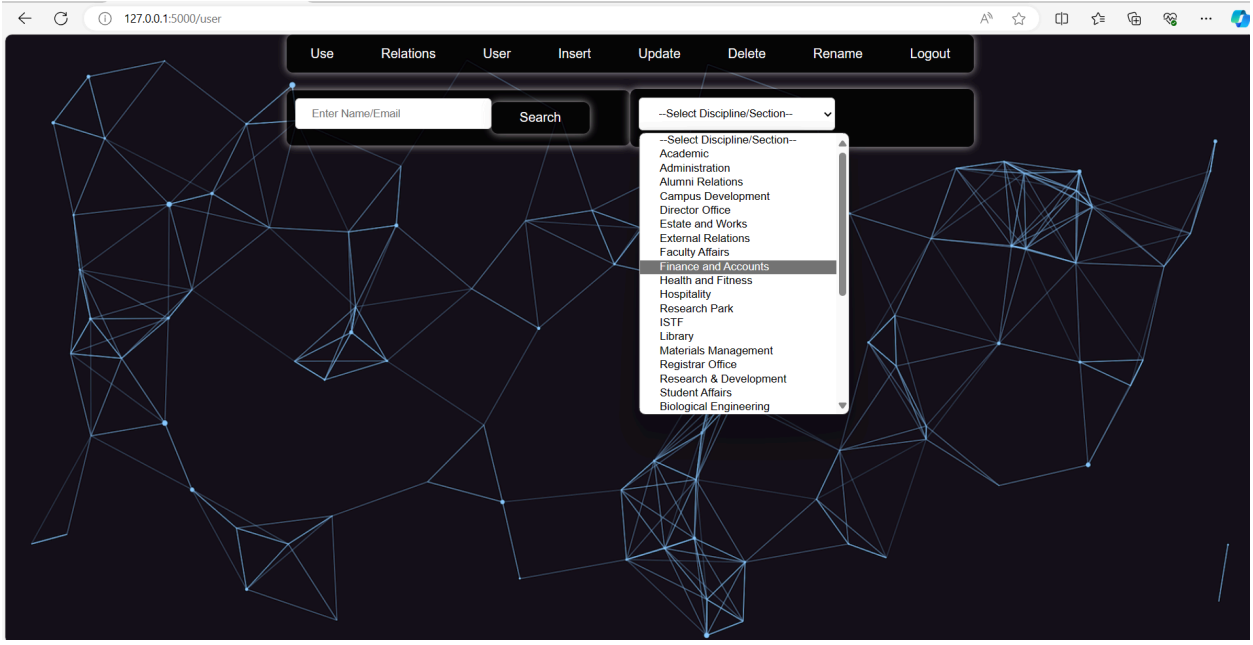


iv) The error messages are shown as paragraph tags within the containers. But we made the pop-ups show the same messages only for CRUD operations, i.e., for insert, update, delete, and rename pages. The login, use, and user pages still have the same paragraph tags. The sentence in white, as shown in the picture below the delete button, depicts this. The same delete operation can be seen as a pop-up in the changes after the second feedback.

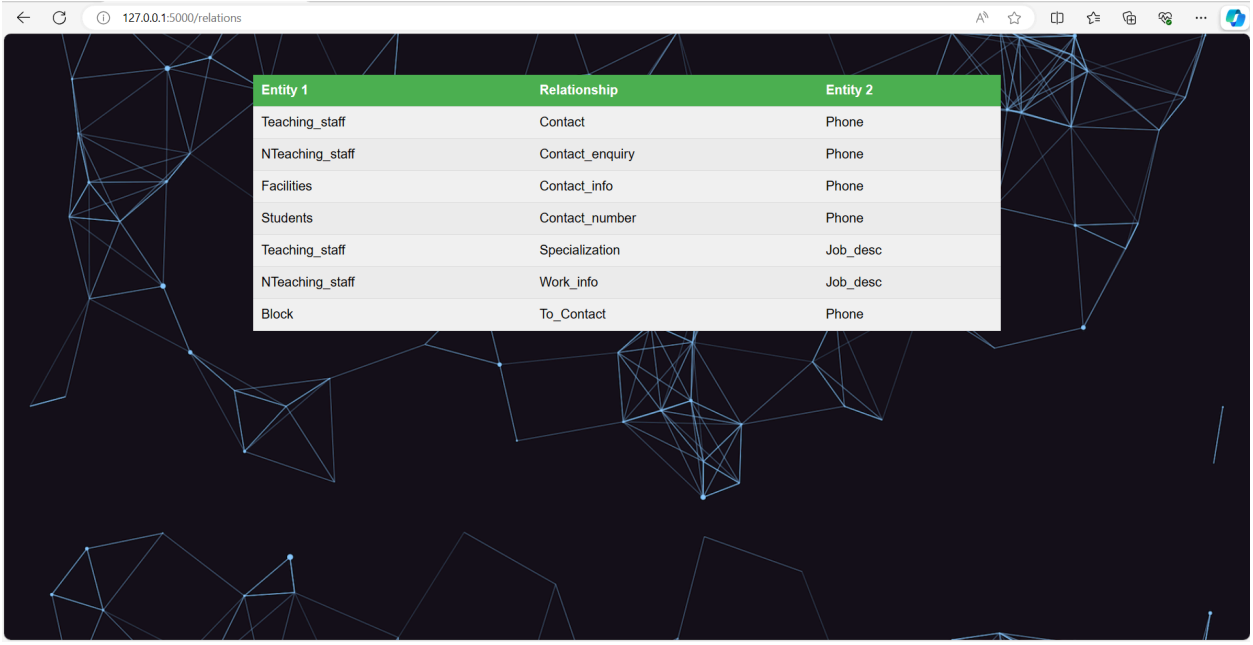


After the first feedback:

i) We have successfully inserted a drop-down for the disciple/section form on the user page. The drop-down includes several disciplines and sections, as we can see below. This is the first feature update in the application.



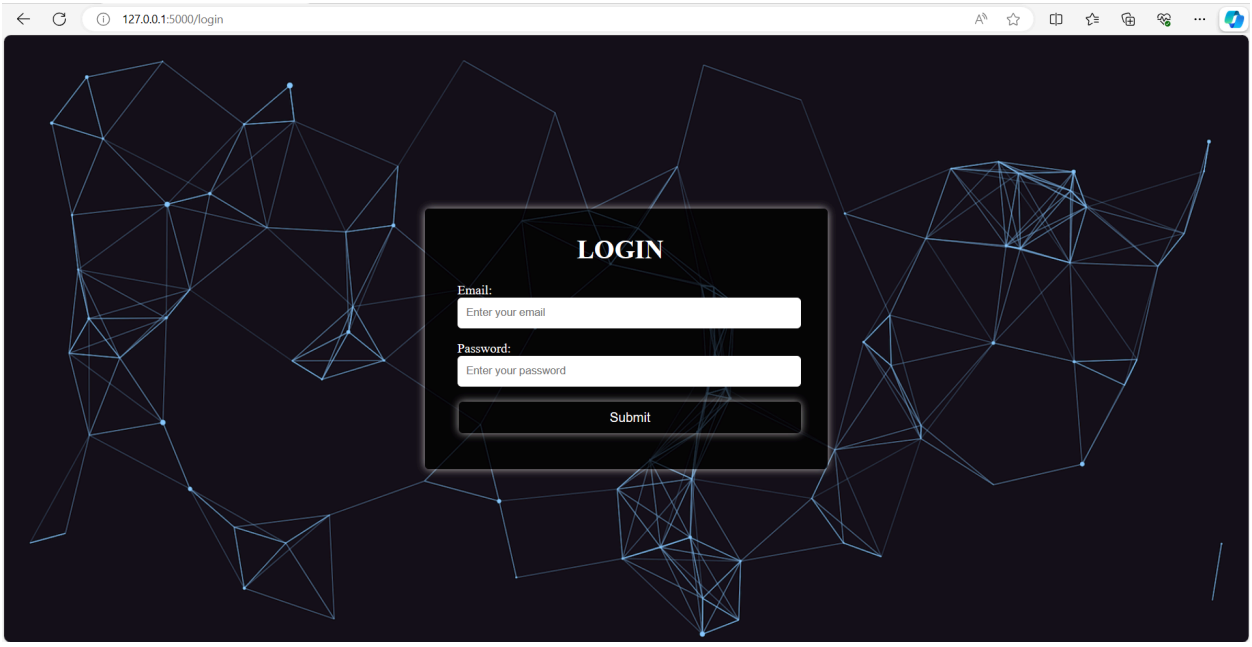
ii) A new page/route, relations, has been added. It contains a table showing all the relationships between various entities in the database. This is the second updated feature of the application.



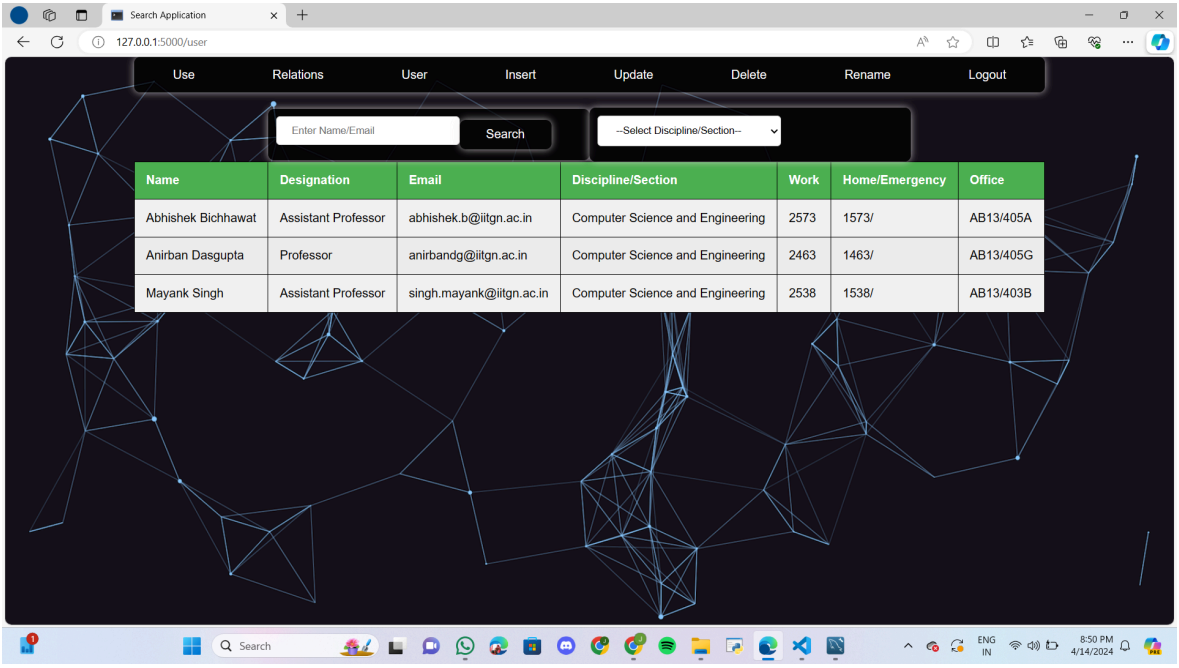
Entity 1	Relationship	Entity 2
Teaching_staff	Contact	Phone
NTeaching_staff	Contact_enquiry	Phone
Facilities	Contact_info	Phone
Students	Contact_number	Phone
Teaching_staff	Specialization	Job_desc
NTeaching_staff	Work_info	Job_desc
Block	To_Contact	Phone

iii)The third update to the application:

Additionally, we have added a password requirement to the login page. A new register page or route is added to the index.html or home page. Both register and login require a password for the particular user to access the data in the application.



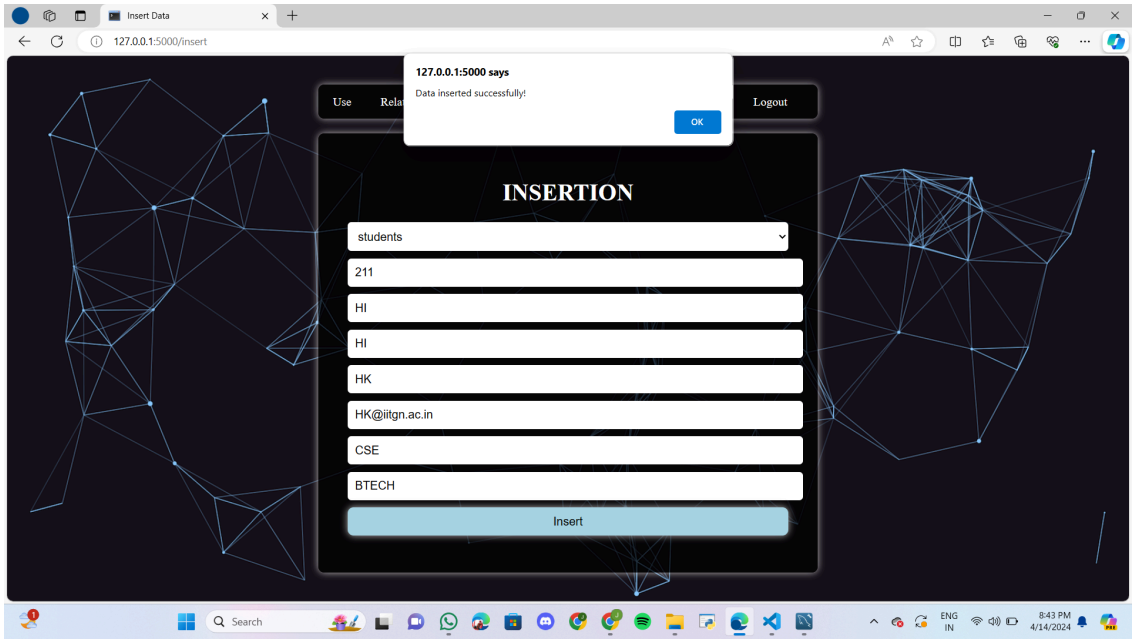
iv) Stylings for the user page are changed: The following picture shows the new user page with changed stylings. The search option by Name/Email is separated from the search by Discipline/Section field. As mentioned in (i), we also made the Discipline/Section field a drop-down showing all the possible disciplines on campus.



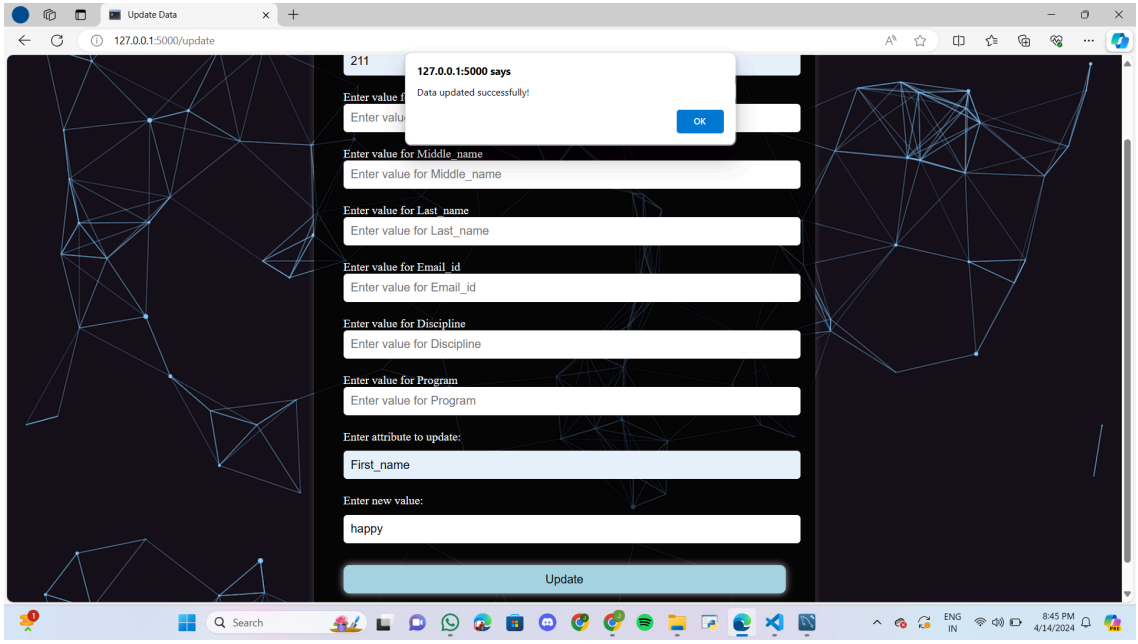
After the second feedback:

In the second feedback, we got input from stakeholders to add alerts/pop-ups when CRUD operations are done to the database tables. We changed the flash messages from the Flask application earlier from paragraph tags to alerts. A dialogue box shows that the query has been successfully executed when we do the insert/update/delete/rename operations.

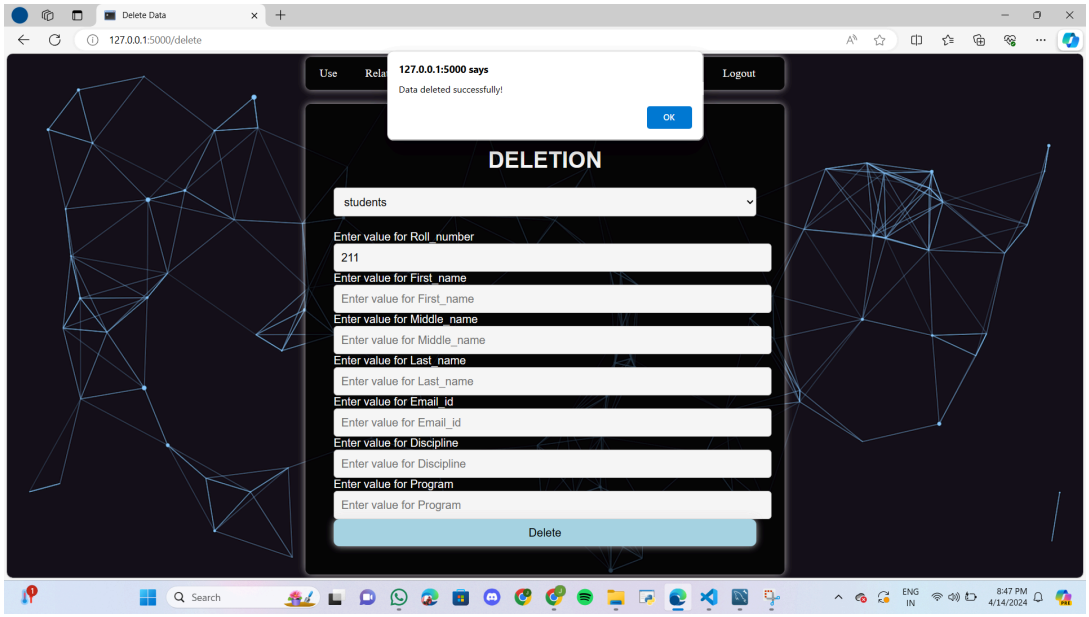
This is when inserting new student information. Roll_number = 211



Updating the First_name of the same student as above



Finally deleting the same student with Roll_number = 211 below:



2. **Screenshots of different Users**

There are only two different types of users within this application.

i) One is the administrator. He is identified by the following:

Name: admin

Email: admin249@iitgn.ac.in

Password: H1

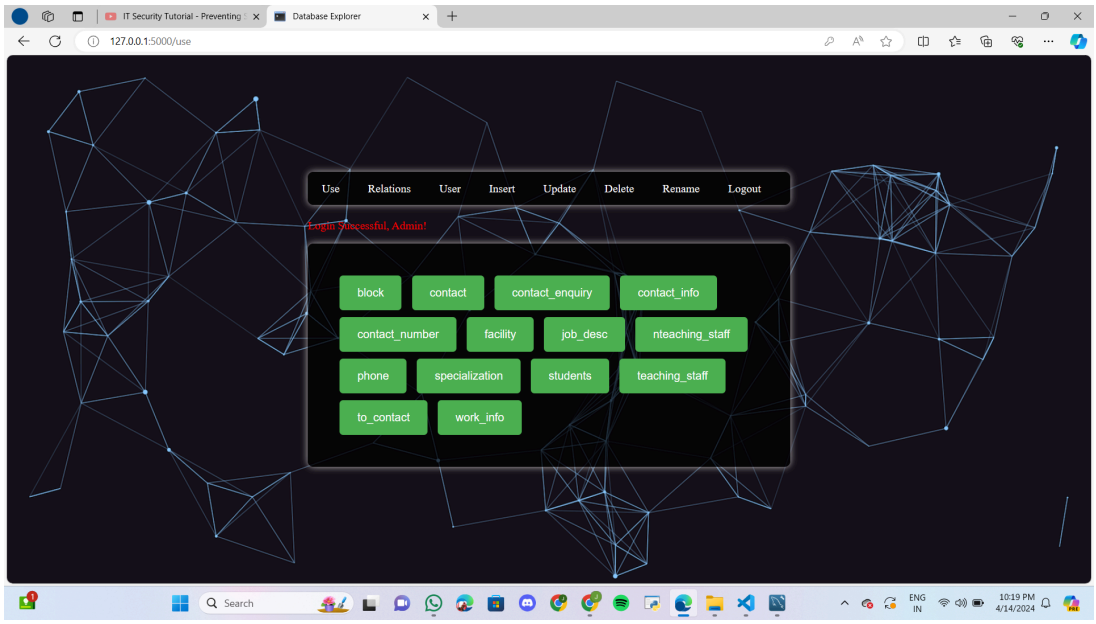
Or

Name: PseudoAdmin

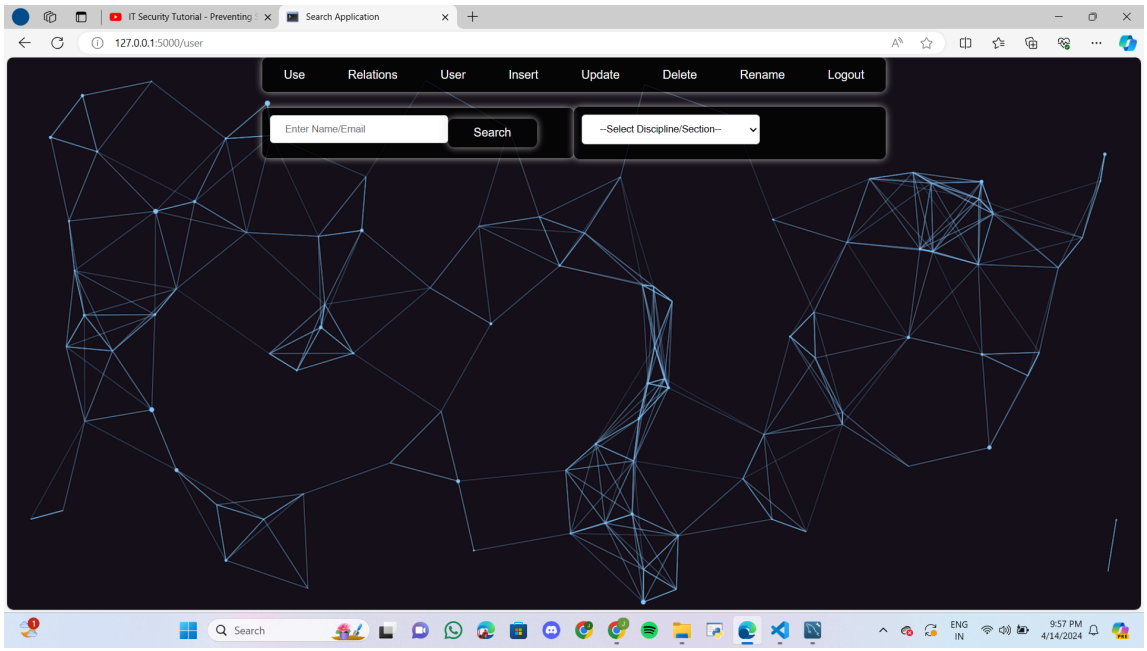
Email: pseudo@iitgn.ac.in

Password: 12345

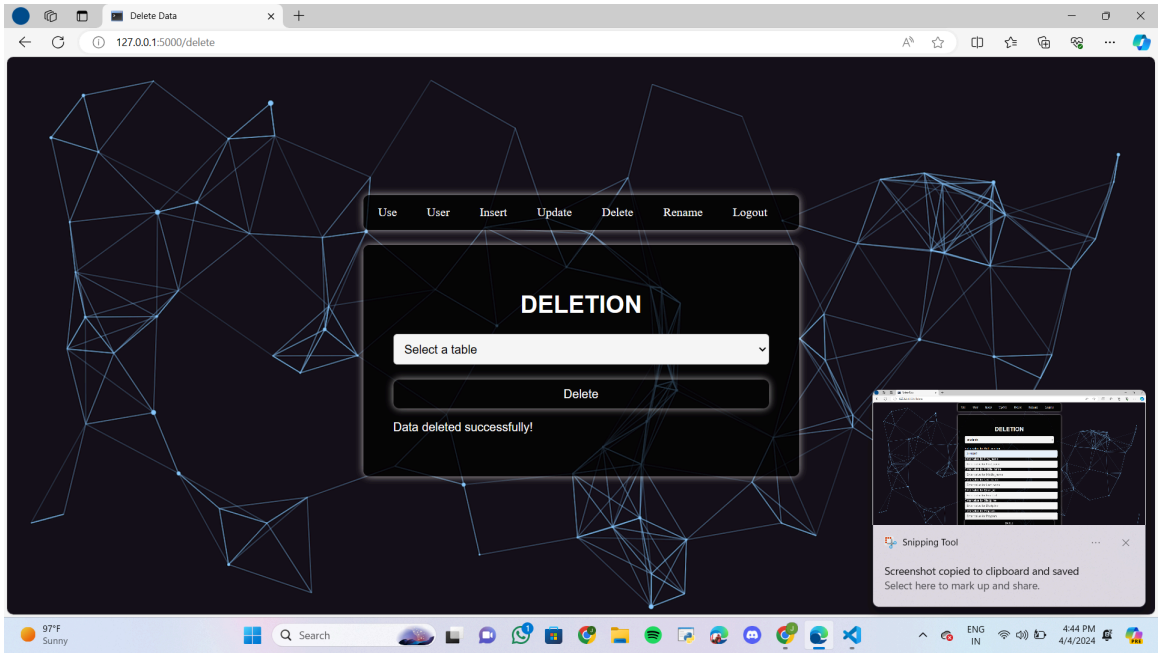
After Logging in as an administrator with these details, we will land on the Use page.



Privileges: An administrator can access all the operations. Namely, Use, Relations, Insert, Update, Delete, Rename, User pages. The User page is as shown below:



When we try to access other pages as administrators, we can see the following pages without issues. The delete page will be as follows:

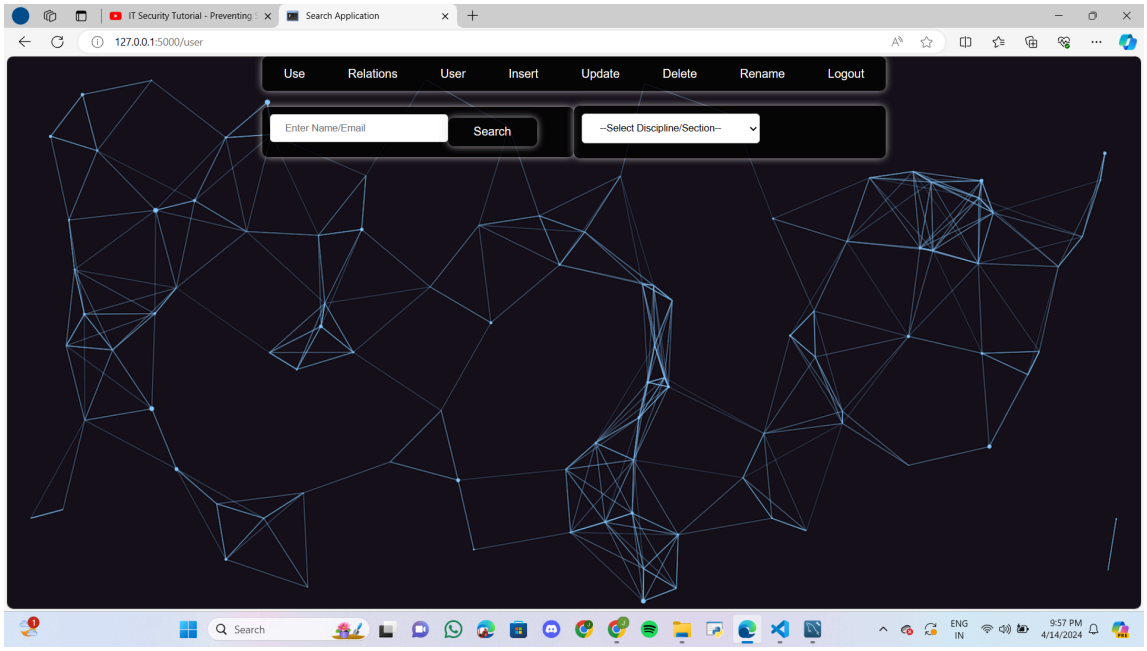


Similarly all the other pages can be accessed as an administrator.

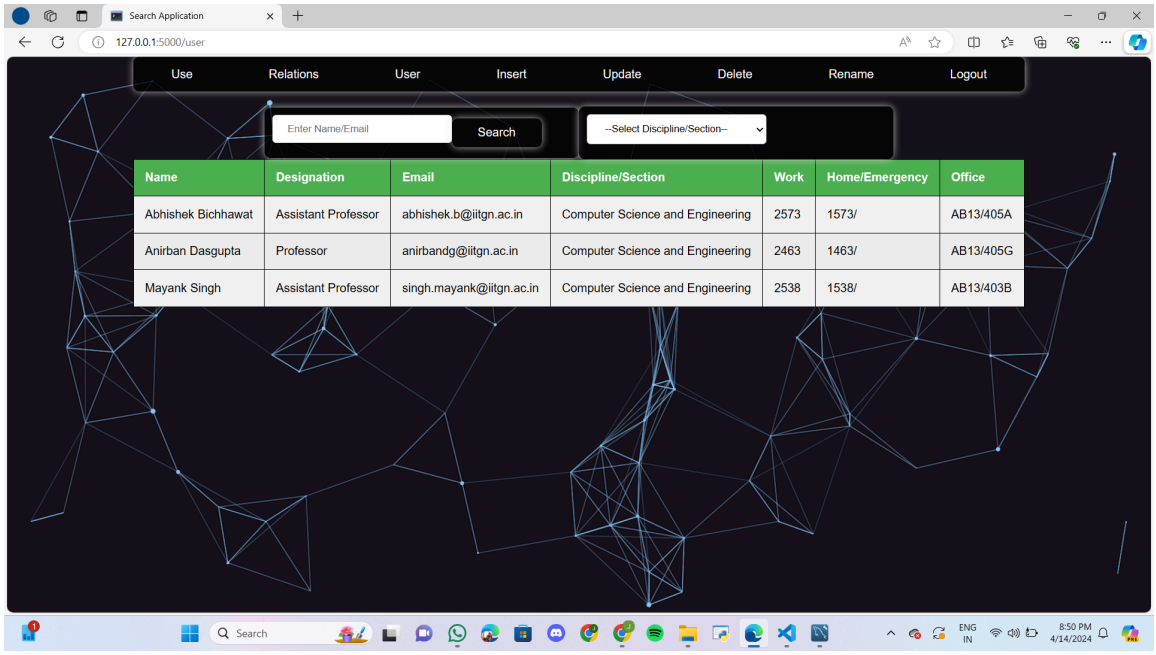
ii) The second type comprises all those who are not the administrators.
The email should contain the @iitgn.ac.in tag. Because only people from the IITGN community can access this application.

Privileges: A normal user can only access the User page. He/she can not access and is not authorized to access the Use, Relations, Insert, Update, Delete, and Rename pages.

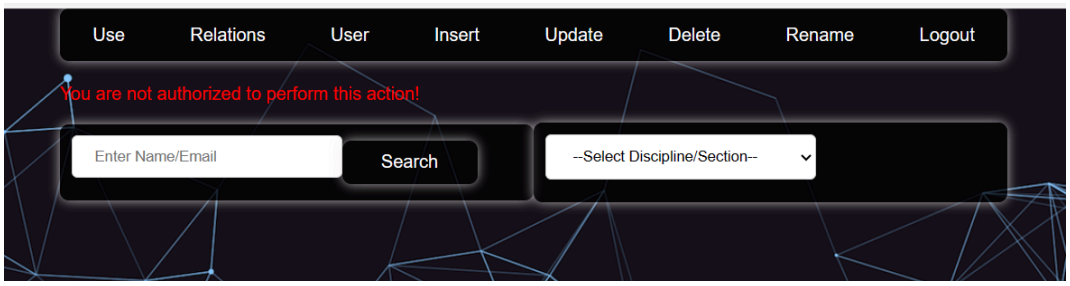
After logging in as a normal person or other than an administrator, we will land on the User page. This is what differentiates a normal user from an admin. While the admin lands on the Use page, normal people are redirected to the user page.
The User page is the only one relevant to normal users to get full information about a person. We can search the records by their Name/email or Discipline/section. User page:



When searches are made:



If we try to access any other page say Use, Relations, Insert, Update, Delete, Rename, as a normal user, we will get a red color paragraph showing the text:



3.2 Responsibility of G2:

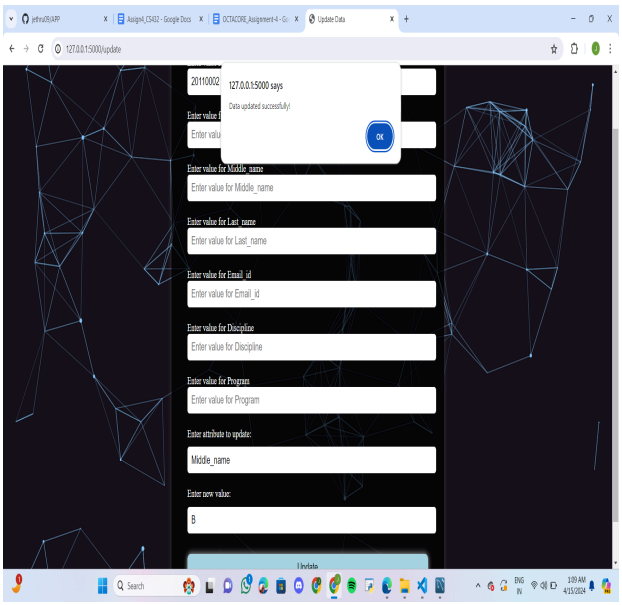
1. **<u>Concurrent multi-user access:</u>**
   For concurrent multi access we created a pseudo admin user functionality with privileges same as admin to the database, Now to include the multi-user access functionality without conflicts during update or rename we implemented it through locking the table for the user while using the feature to update the table. The code is as shown below to include in the backend

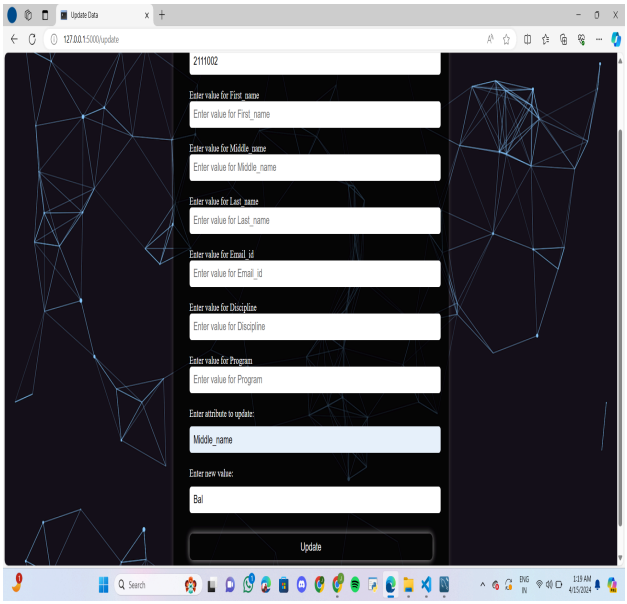
```
cur.execute(f"LOCK TABLES {table_name} WRITE")
cur.execute(f"UNLOCK TABLES")
```

Similarly for Rename operation this is as follows:

cur.execute(f"LOCK TABLES {old_table_name} WRITE")

cur.execute(f"UNLOCK TABLES")





Roll_number	First_name	Middle_name	Last_name	Email_id	Discipline	Program
2	K	NULL	K	K	K	K
20110002	Abhishek	Ba	Mungekar	abhishek.mungekar@iitgn.ac.in	Mechanical Engineering	Btech
20110003	Progyan	Balaji	Das	progyandas@iitgn.ac.in	Computer Science Engineering	Btech

2. **Changes in the database as per the feedback:**

i) The first change we made is to have our own login and registration pages. So we added the password for extra security purposes. These passwords are stored in the hash forms.
The code relevant to the password hashing is as follows: It uses Python's hashlib to do this.

```
26
27      # Hash the password using SHA-256
28      password_hash = hashlib.sha256(password.encode()).hexdigest()
29
```

The hashed passwords are stored in our octacore database itself. We created an extra table for this use. The table name is users. As shown, the email address will be unique. This is done in the database.

```
432
433 ● CREATE TABLE users (
434       id INT AUTO_INCREMENT PRIMARY KEY,
435       name VARCHAR(100),
436       email VARCHAR(100) UNIQUE,
437       password_hash CHAR(64)
438   );
```

For login, password hashes are checked. I.e. entered password's hash and stored hash is checked.

```
59
60      # Check if the password is correct
61      if hashlib.sha256(password.encode()).hexdigest() != password_hash:
62          flash("Incorrect password. Please try again.")
63          return redirect(url_for("login"))
64
```

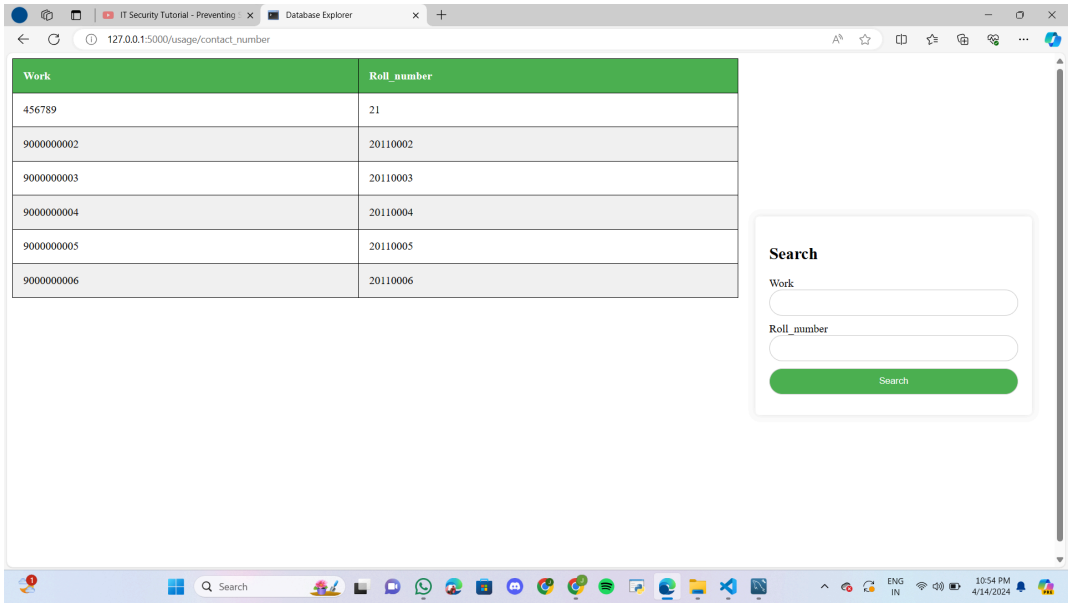
ii) The second change made to the database is the updating of all the relationships between the two parent entities. All tables that contain the foreign key constraints are changed to have

- ON DELETE CASCADE ON UPDATE CASCADE

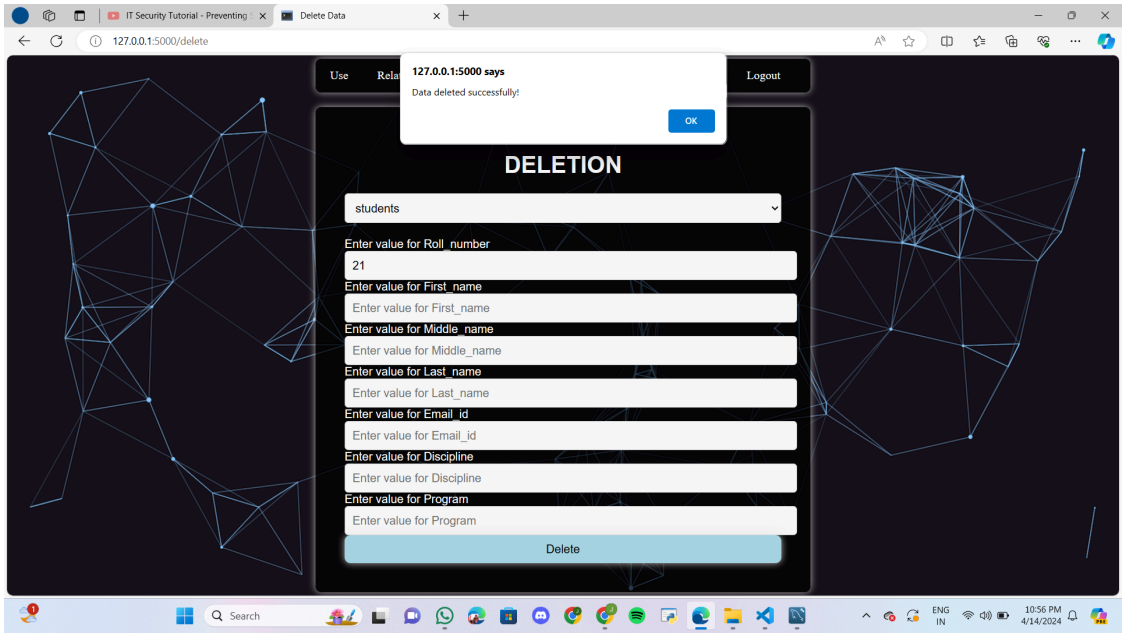
This helps us in many ways. Let's say I am trying to delete a student's record, now I can just use the delete page to delete records from the students table and the related data in the Contact_number table will also be deleted. But we still have to go to the other parent table Phone and delete records from that end.

```
95  CREATE TABLE Contact_number(
96   Work VARCHAR(10),
97   Roll_number VARCHAR(15),
98   PRIMARY KEY(Roll_number,Work),
99   FOREIGN KEY(Work) REFERENCES Phone(Work) ON DELETE CASCADE ON UPDATE CASCADE,
100  FOREIGN KEY(Roll_number) REFERENCES Students(Roll_number) ON DELETE CASCADE ON UPDATE CASCADE
101 );
102
```

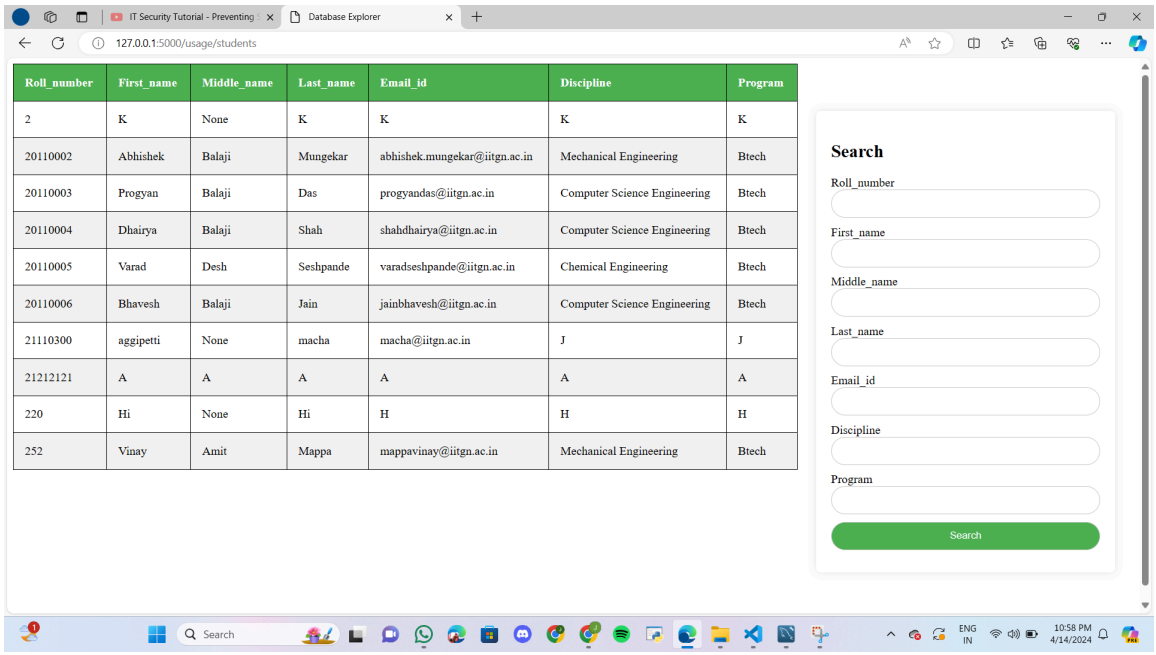
The following are screen shots to show the delete operation: let's delete the first entry here.



The below pic shows delete from the student table: Roll_number = 21

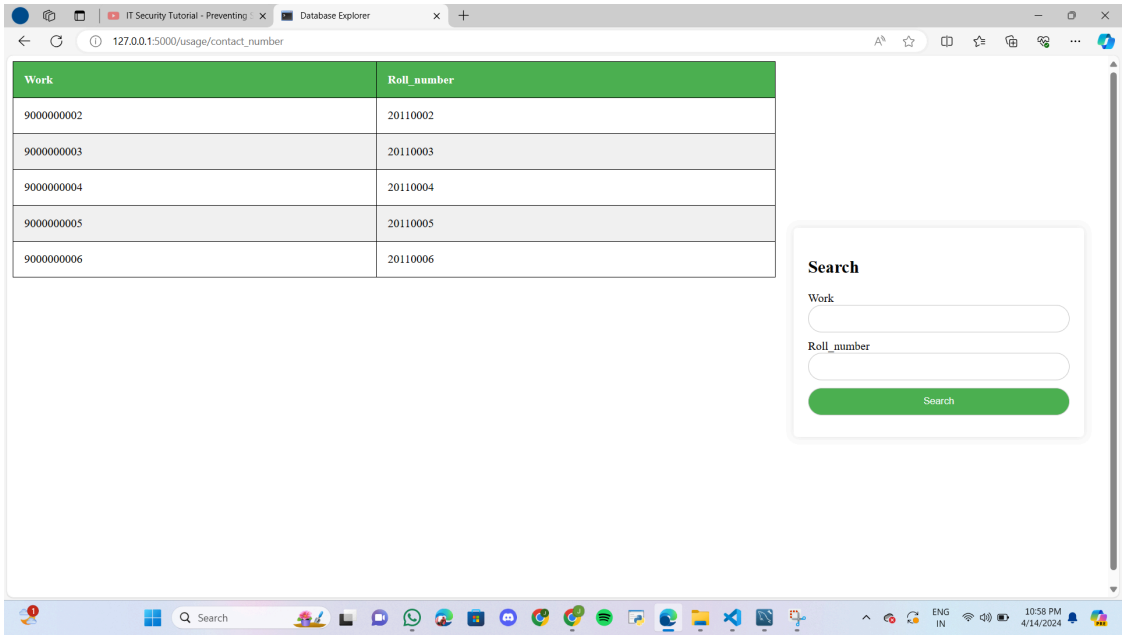


This will delete not only the students table but also in the Contact_number also. Below are the pics: No entry with Roll_number = 21. In the students table.



Roll_number	First_name	Middle_name	Last_name	Email_id	Discipline	Program
2	K	None	K	K	K	K
20110002	Abhishek	Balaji	Mungekar	abhishek.mungekar@iitgn.ac.in	Mechanical Engineering	Btech
20110003	Progyan	Balaji	Das	progyandas@iitgn.ac.in	Computer Science Engineering	Btech
20110004	Dhairya	Balaji	Shah	shahdhairya@iitgn.ac.in	Computer Science Engineering	Btech
20110005	Varad	Desh	Seshpande	varadseshpande@iitgn.ac.in	Chemical Engineering	Btech
20110006	Bhavesh	Balaji	Jain	jainbhavesh@iitgn.ac.in	Computer Science Engineering	Btech
21110300	aggipetti	None	macha	macha@iitgn.ac.in	J	J
21212121	A	A	A	A	A	A
220	Hi	None	Hi	H	H	H
252	Vinay	Amit	Mappa	mappavinay@iitgn.ac.in	Mechanical Engineering	Btech

Also no entry in the Contact_number. The earlier first row is deleted automatically due to ON DELETE CASCADE.

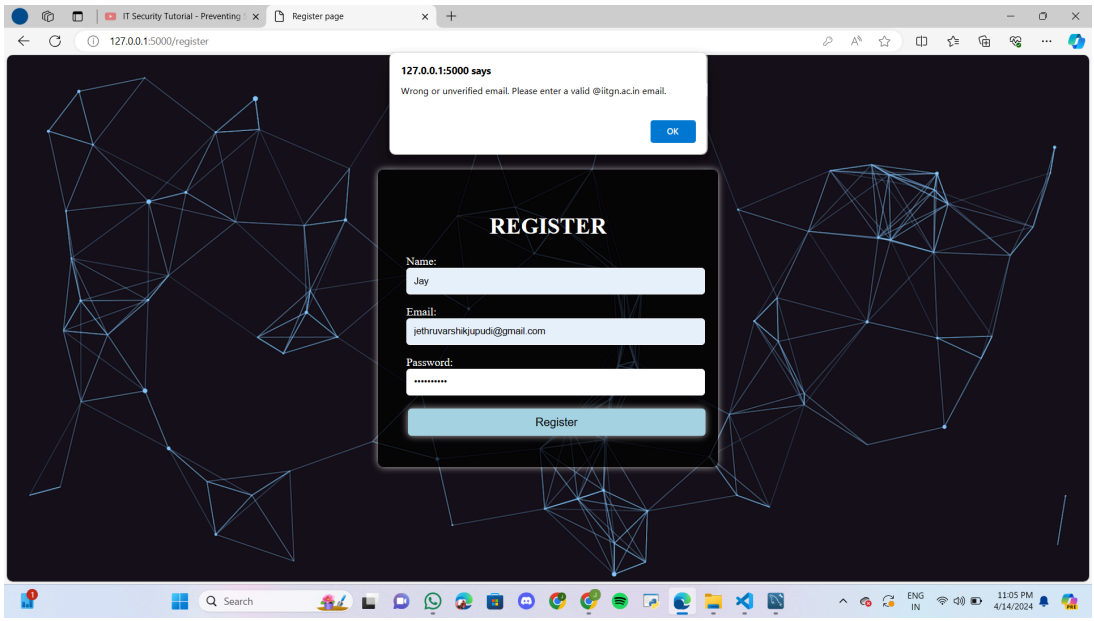


Work	Roll_number
9000000002	20110002
9000000003	20110003
9000000004	20110004
9000000005	20110005
9000000006	20110006

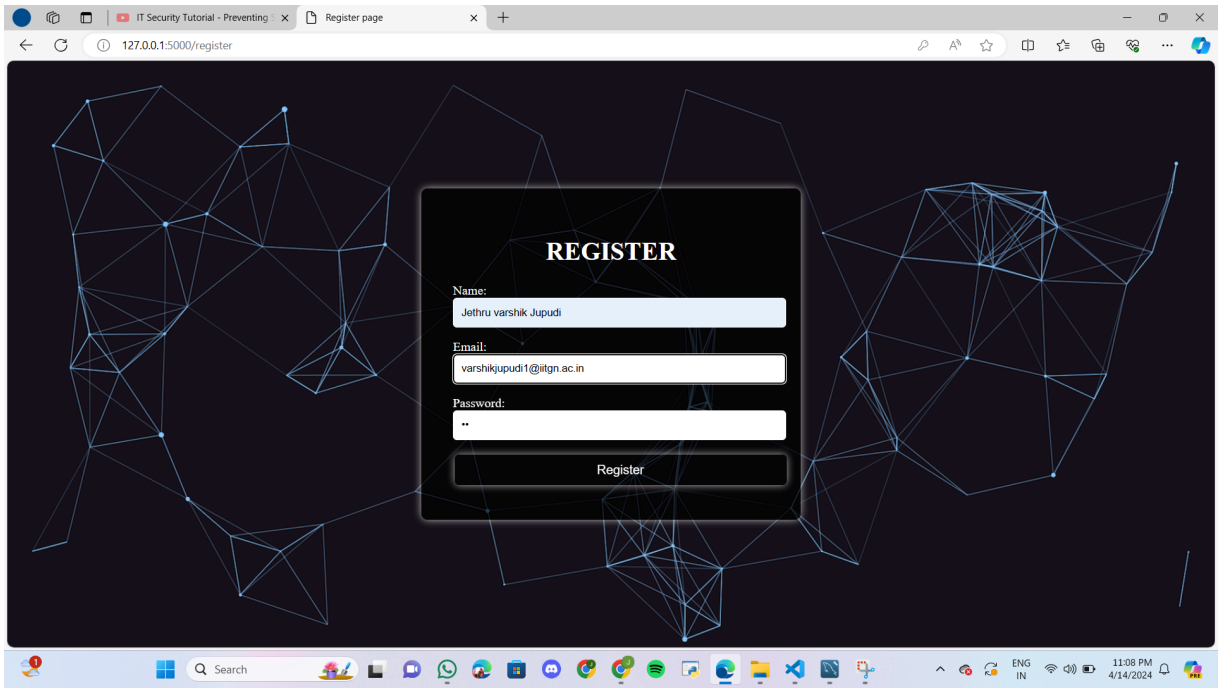
3. **<u>Adding authentication for login and registration</u>** (only IITGN user can login and register)

i) Register page:

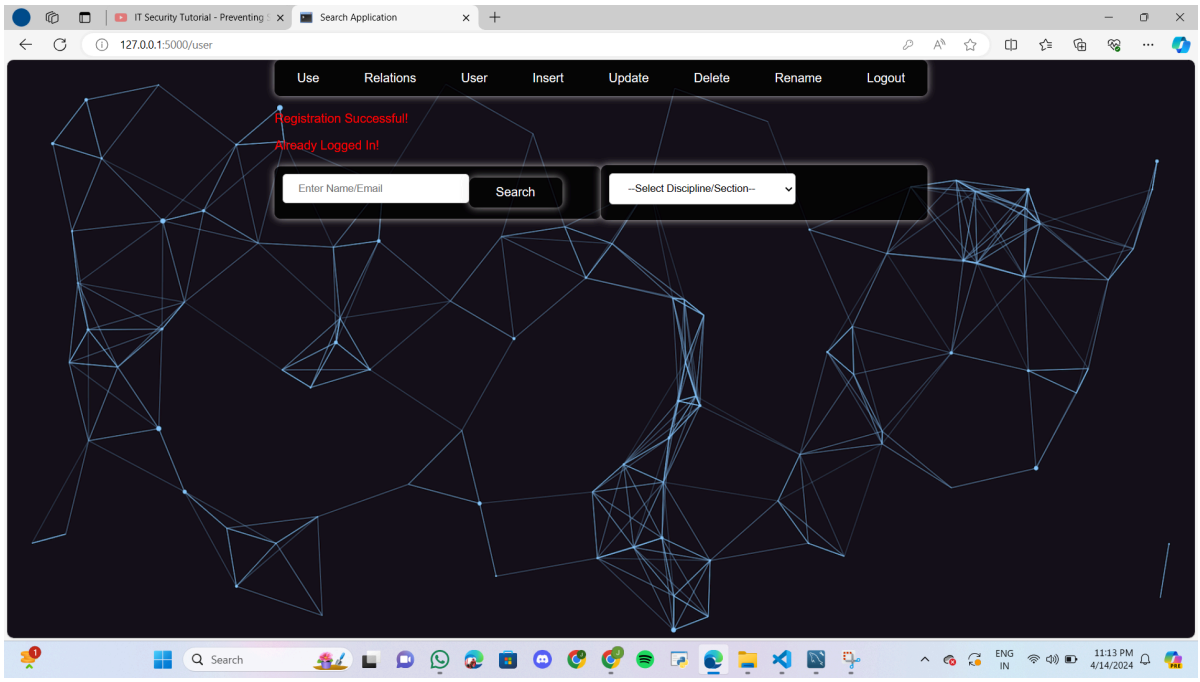
We can register only with the IITGN email id. So @iitgn.ac.in should be contained in its Email id.



Once registered, we can go and login accordingly.

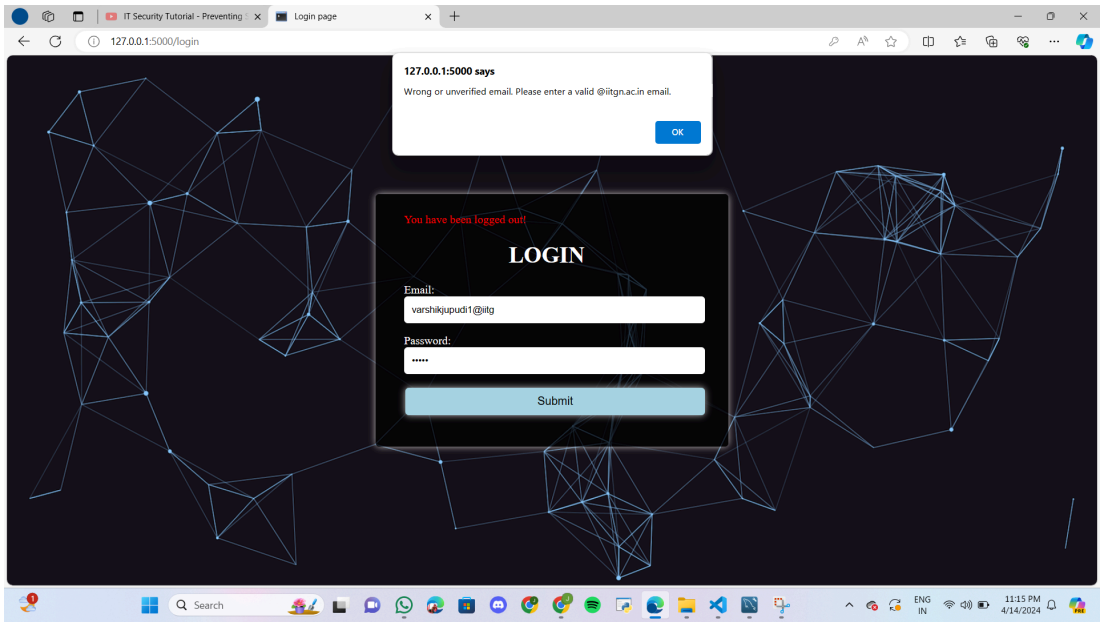


We will directly go to the User page as we are a normal user. It will be redirected to the login page first and then as the session just stored data, it will redirect further to this User page.

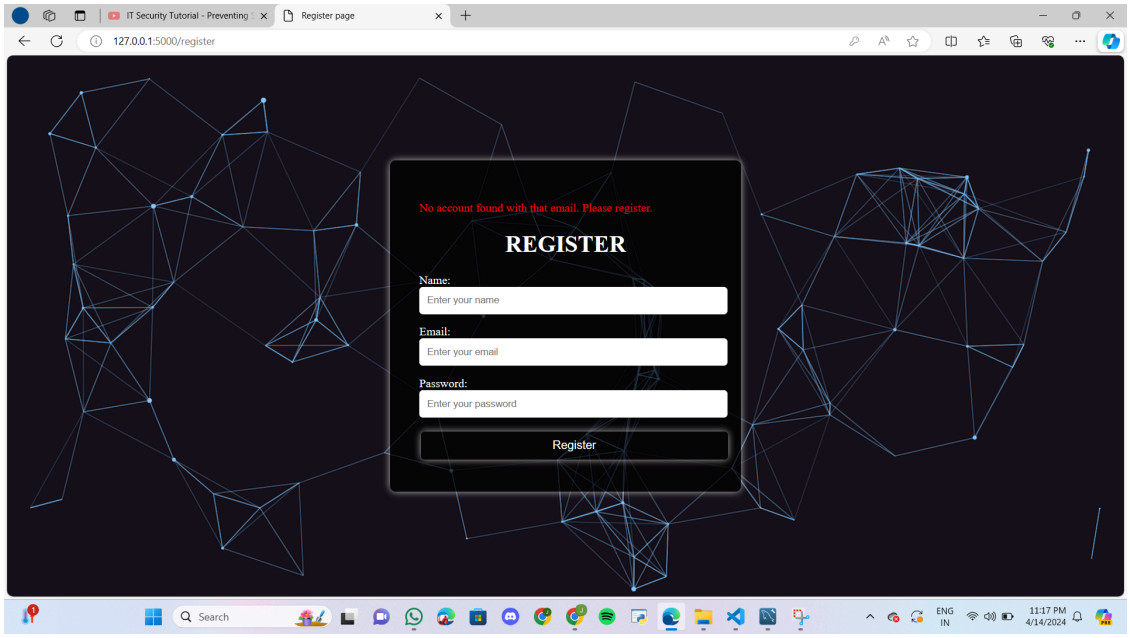


ii) Login Page:

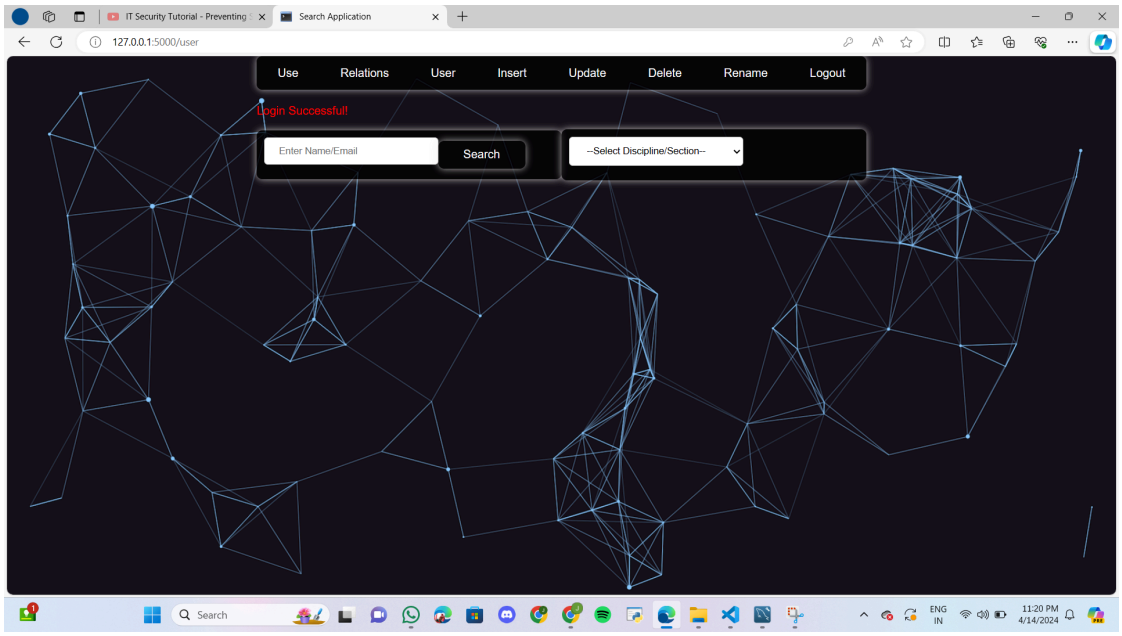
Here we should enter email and password. If not IITGN account then error:



Now if we enter our email and password, we will be redirected to the register page if not registered already: Cause we should already be registered to be able to log in.



If registered and then logged in we will land on the User page if not an administrator. If admin then we will land on the Use page.



Here is a snip of the records of the users table in the database: Password hashes are stored.

```
433 • CREATE TABLE users (
434       id INT AUTO_INCREMENT PRIMARY KEY,
435       name VARCHAR(100),
436       email VARCHAR(100) UNIQUE,
437       password_hash CHAR(64)
438   );
439 •   select * from users;
```

id	name	email	password_hash
1	Jethru varshik Jupudi	varshikjupudi@iitgn.ac.in	6b51d431df5d7f141cbececcf79edf3dd861c3b4...
3	admin	admin249@iitgn.ac.in	8bf797aa02bd65b7cbd52155f6e010be77bb3d4...
4	Sujith	sujith@iitgn.ac.in	8691dd14cb5e57c529007fa104d40864b8fcc14...
6	Kay	'or'1'='1'--@iitgn.ac.in	767c9cf8c44d67f35a7b24651fb2e0df7cd83cfcf...
8	Pseudoadmin	pa@iitgn.ac.in	a665a45920422f9d417e4867efdc4fb8a04a1f3f...
11	Jethru varshik Jupudi	varshikjupudi1@iitgn.ac.in	5994471abb01112afcc18159f6cc74b4f511b998...
NULL	NULL	NULL	NULL

3.3 Responsibility of G1 & G2:

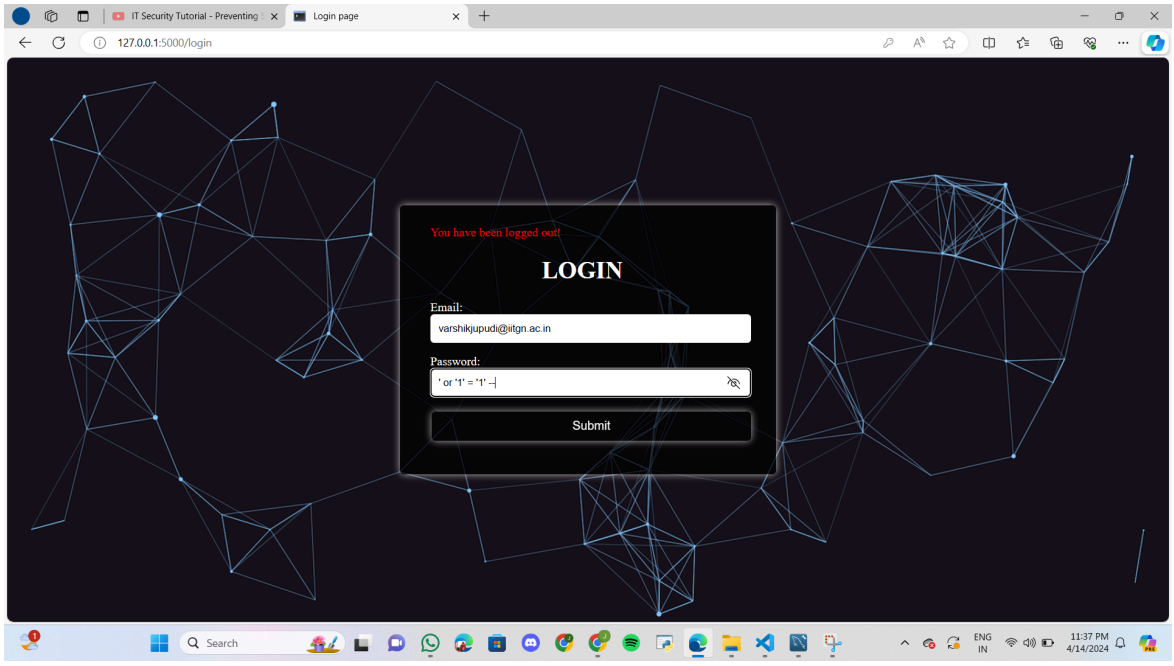
1. **SQL Injection**

Normal login page:

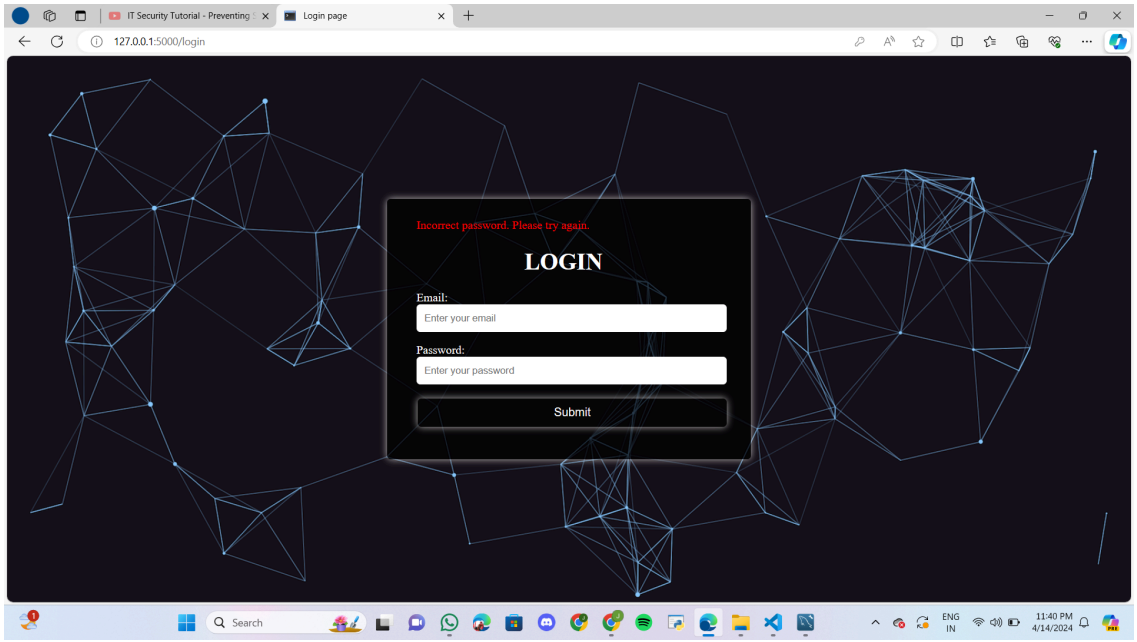
Email ID : varshikjupudi@iitgn.ac.in

Password: 12

Trying to inject our way without knowing the password: ‘’ or ‘1’ = ‘1’ – <sql comment>



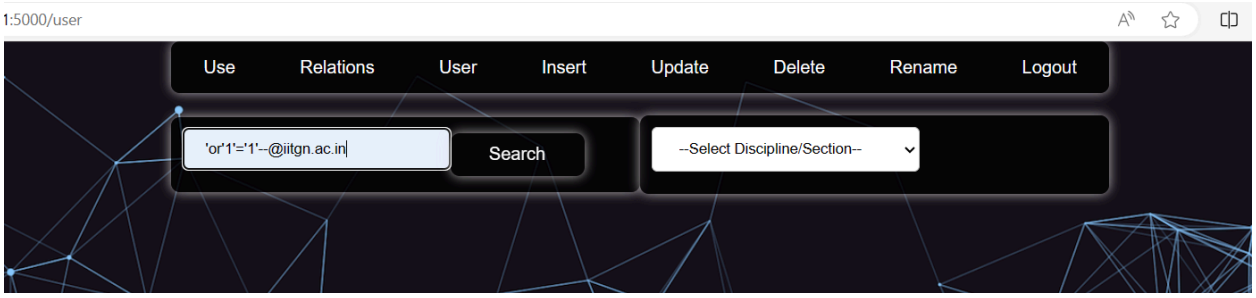
Result:



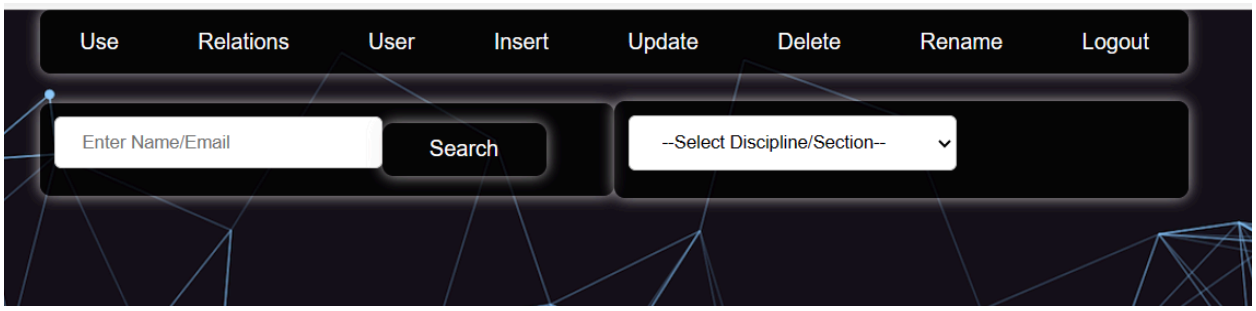
We won't be able to login. This is because in normal cases we might get injected but here passwords are stored in hash form so the stored hash won't be the same as this sql injected query part. hash(12) not equal to hash (' or '1' = '1' –)

```
59
60      # Check if the password is correct
61      if hashlib.sha256(password.encode()).hexdigest() != password_hash:
62          flash("Incorrect password. Please try again.")
63          return redirect(url_for("login"))
64
```

Another SQL injection prevention is at the Search operation in the User page.



Here we might think that all the emails will be valid so all the records will be visible but this won't happen. But no visible changes. Cause of the way in which the query is structured.



The query becomes:

```
query + " WHERE Name LIKE %s OR Email LIKE %s ORDER BY Name", (f"%{search_term}%", f"%{search_term}%")
```

Query: query + " WHERE Name LIKE ‘ or ‘1’ = ‘1’ – OR Email LIKE ‘ or ‘1’ = ‘1’ – ORDER BY Name".

This is all grouped/binded as one entity under %s , so no injection..

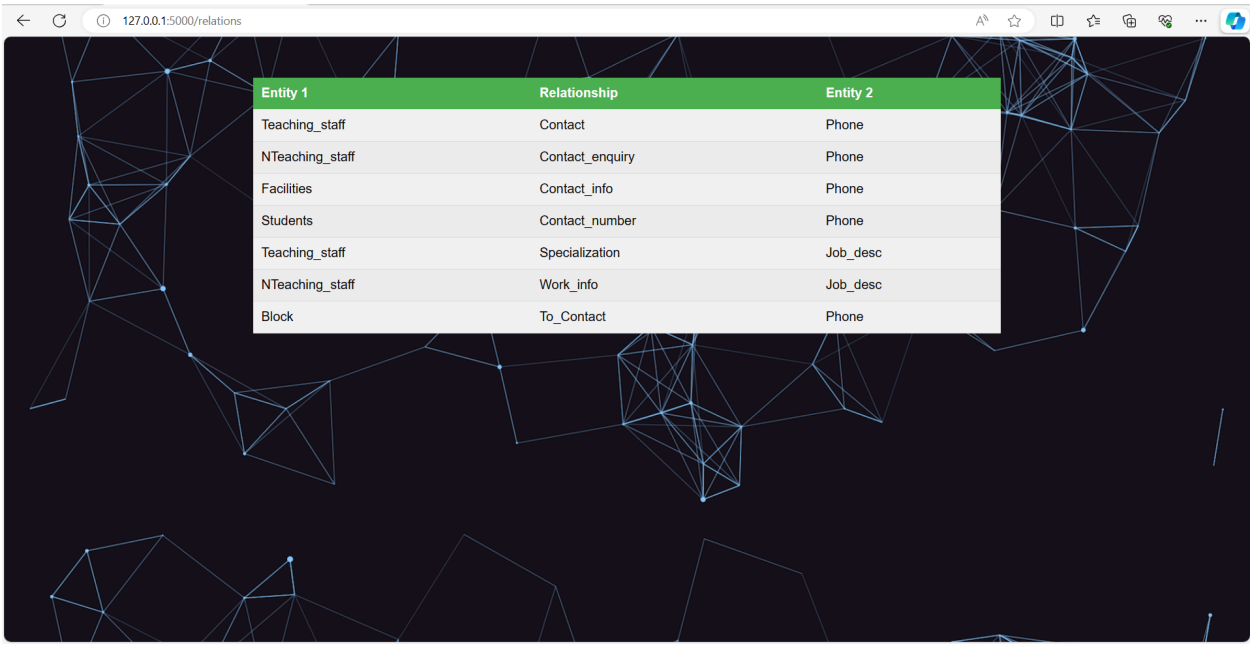
```
279         # Search by Name/Email
280         if 'search_term' in request.form:
281             search_term = request.form['search_term']
282             cur.execute(query + " WHERE Name LIKE %s OR Email LIKE %s ORDER BY Name", (f"%{search_term}%", f"%{search_term}%"))
283             data.extend(cur.fetchall())
284
```

2. **relations and their constraints**

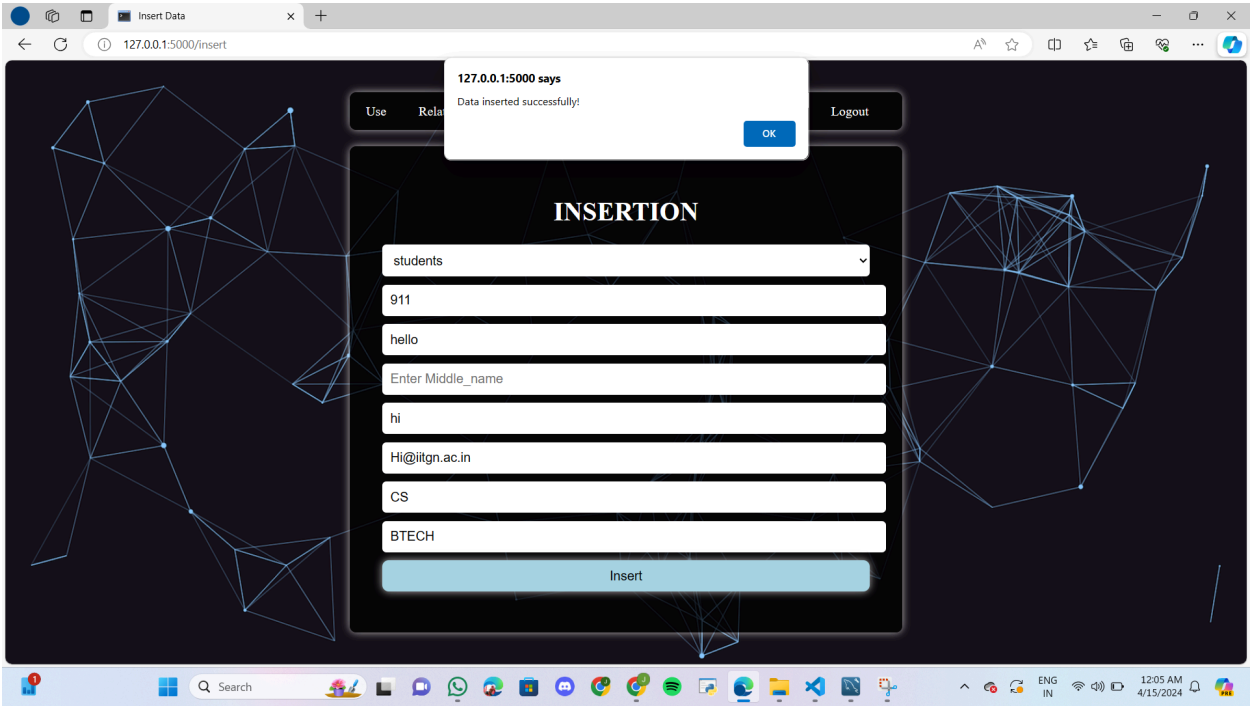
No particular changes are made to the database except the changes related to the foreign keys, ON DELETE CASCADE ON UPDATE CASCADE.

The following page relations page static, shows the relations present in the database as per assignment 1.(ONLY admin can access it).

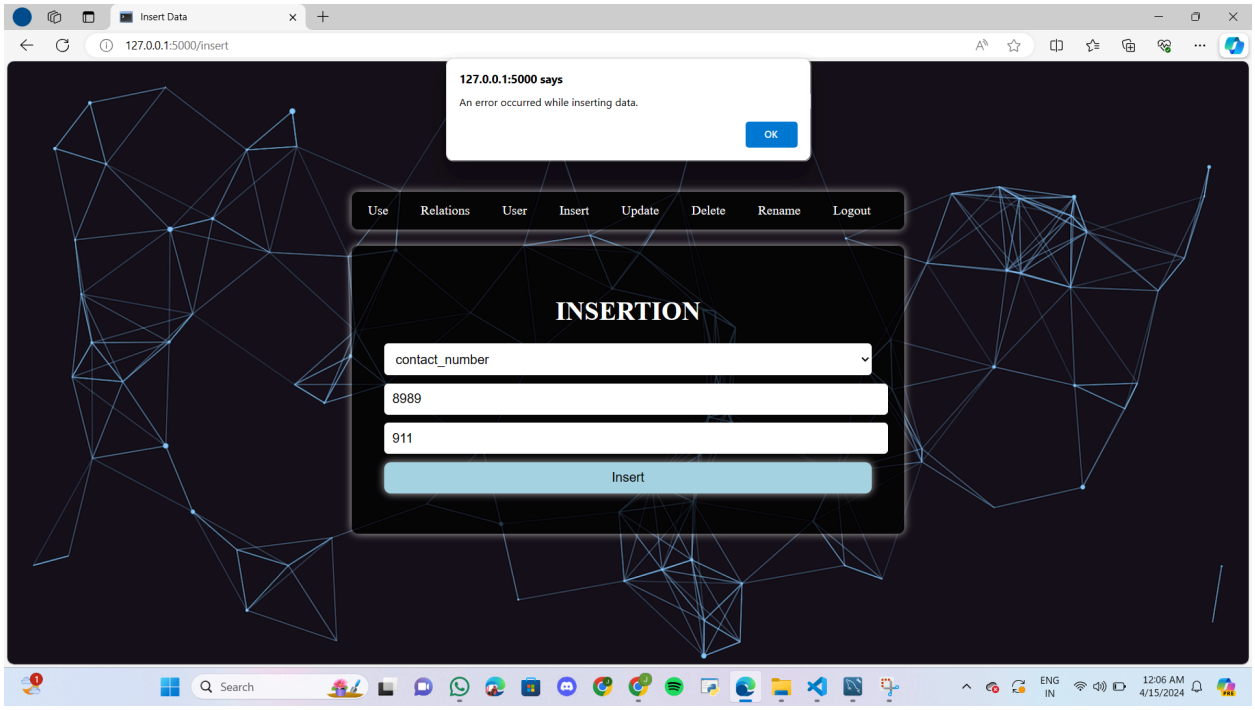
We can check these by insertions as admin.



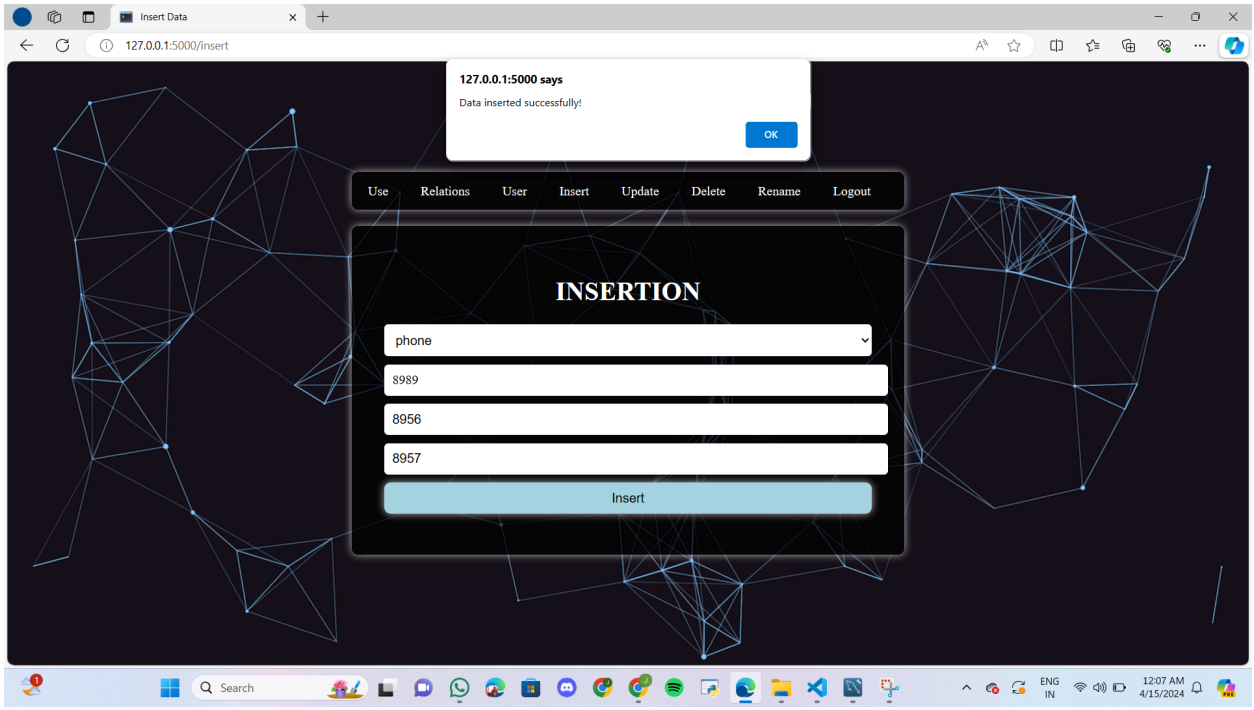
Here insert Roll_number: 911 student information.



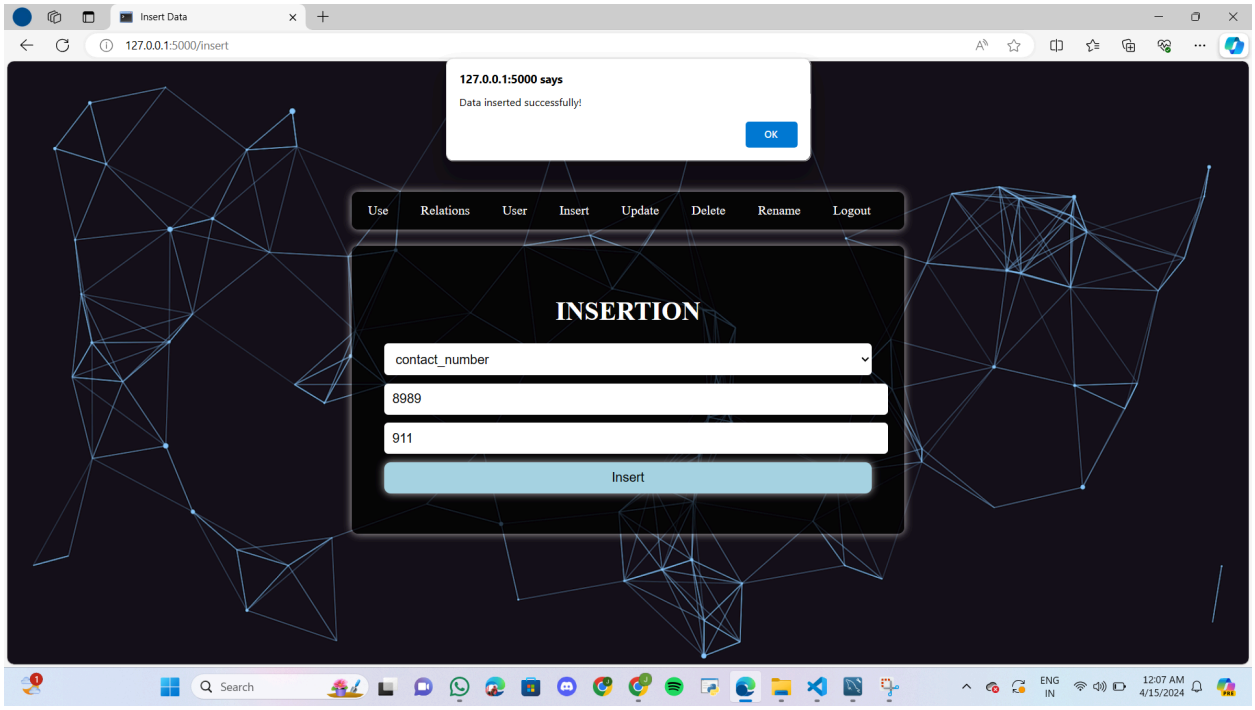
Now let's enter the Contact_number entity child of Students, Phone tables. We will get an error message, cause we did not form a relation or data at Phone table relating to this. With Work = 8989.



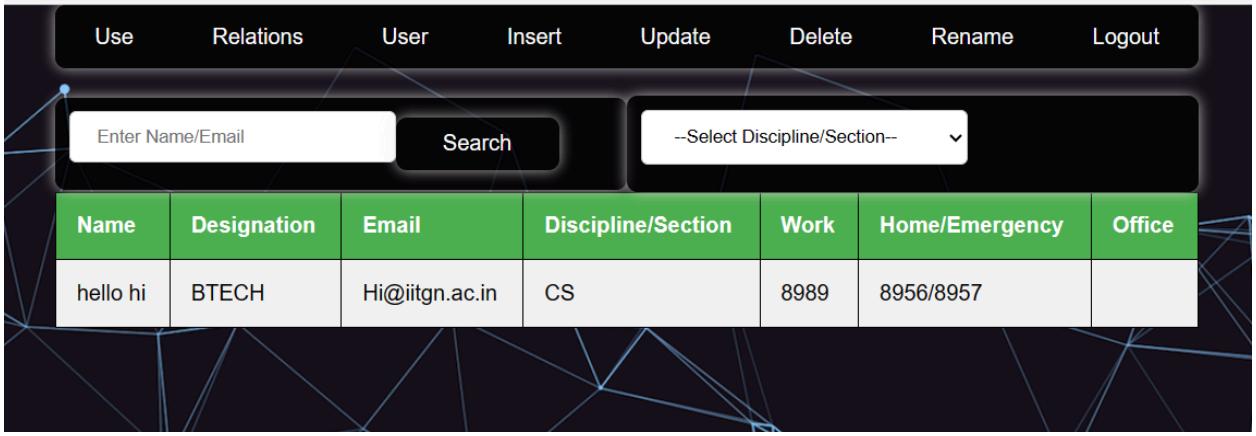
So first let's insert Phone.Work = 8989, then some Home emergency numbers.



Now try to do the insertion to the Contact_number entity child of Students, Phone tables. Work = 8989, Roll_number = 911. Now we are able to successfully insert. This shows the relationship.



Once this is done, go to the User page and do the search by name: hello, to get his entire data.



We can directly go to the Sql code to check the relations also.
As For constraints there are many NOT NULL constraints as Last_name in students table. They remain as they were in assignment 2. Same can be checked in the sql file.

Relations:

Entity 1	Relationship	Entity 2
Teaching_staff	Contact	Phone
NTeaching_staff	Contact_enquiry	Phone
Facilities	Contact_info	Phone
Students	Contact_number	Phone
Teaching_staff	Specialization	Job_desc
NTeaching_staff	Work_info	Job_desc
Block	To_Contact	Phone

CONTRIBUTIONS:

Group 1 Members:

Uday Kumar:

- Involved in improving design of the front-end user interface.
- Contributed innovative ideas and actively participated in group discussions.
- Worked on the suggestions given in the second feedback.
- Assisted in documenting the whole file.

Sujith:

- Contributed to front-end development, focusing on user experience and interface design.
- I also contributed to answering the questions in section 3.3.
- Worked on the suggestions given in the second feedback.
- Assisted the other team members of G2 to complete the responsibility of G1 and G2 (3.3 section).

Ajith:

- Participated in designing front-end elements and user interactions.
- Assisted the teammates of G2 with the final draft of the responsibility of the G2 section.
- Assisted in drafting the whole document.

Keerthi:

- Contributed to front-end development, focusing on user experience and interface design

- Assisted the other team members of G1 to complete the responsibility of G1 and G2 (3.3 section).
- Actively engaged in the responsibility of G1 and provided creative and original ideas.
- Assisted in completing tasks related to G1 and G2 responsibilities.

Pardheev sai:

- Contributed to front-end development, interface design, creating tables while using sql.
- Assisted the other team members of G1.

Group 2 Members:

Jethru:

- Guided and motivated the whole team from the beginning to the end of the assignment.
- Worked on back-end development regarding requested adjustments in feedback.
- Actively engaged in the responsibility of G2 and provided creative and original ideas.
- Made sure that every team member gets enough to work on.

Sudharshan:

- I worked on the back-end development and performed the update, delete, and rename operations.
- Actively engaged in the responsibility of G2 and provided creative and original ideas.
- Assisted the other team members in completing the responsibility of G1 and G2 (3.3 section).
- Assisted in documenting the whole file.

Vinay Marka:

- Contributed to the back-end development regarding requested adjustments in feedback.
- Engaged in brainstorming sessions to determine the essential information needed.
- I also contributed to answering the last section 3.3.
- Assisted in documenting the whole file.

Hima Sagar:

- I worked on the back-end development, primarily performing the update, delete, and rename operations.
- Assisted the other team members in completing the responsibility of G1 and G2 (3.3 section).
- Also assisted in documenting the responsibility of the G1 and G2 sections.
- Assisted the teammates in documenting the whole file.

Rudreshwar:

- Worked on the back-end development regarding requested adjustments in feedback.
- Assisted in completing tasks related to G1 and G2 responsibilities.
- Participated in discussions to ensure all necessary information was identified and included.
- Responsible for answering the last section.

In our team, we ensured each person had a fair share in the project, showing our commitment to fairness and collaboration. Whether in Group 1 or Group 2, everyone played their part, ensuring that our collective effort led to a fruitful outcome.