CS 432: Databases

Assignment-4: Developing the DBMS

Group Members

Aishwarya Omar	20110008
Akshat Shrivastava	20110009
Amaan Ansari	20110011
Mahesh Dange	20110050
Dhakad Bhagat Singh	20110055
Govardhan Ingale	20110070
Ronak Hingonia	20110170
Ojas Washimkar	20110234
Yash Adhiya	20110235
Yash Kokane	20110237
Shubham Kumar	19110137

Tasks

3.1 Responsibility of G1:

1. *The G1 takes two feedbacks from the stakeholders, one initial feedback (on or before 30th March 11:59 PM), and then makes relevant changes as suggested per the first feedback, then final feedback post changes. The write-up/documentation should have screenshots before the first feedback, after the first feedback, and after the second feedback.*

   Feedbacks for the database were collected from Dr. Deepa Shah, as well as the administrative staff at the dispensary.

   As shared in the forms, these were the particular responses from the feedbacks. Please do note that we are only summarizing the constructive feedback, which we have worked on in the website.

- **Initial Feedback:**
  1) A request was made to add a search button in the database website, in order to quickly search any required data. This was necessary, in particular, for the prescription table, where the doctor needs to quickly refer to a patient's previous prescription history.
  2) After reviewing the tables, a request was made to modify the tables, as there were a few redundant columns, as well as a few columns had to be added, such as columns for complaints, investigation, diagnosis, references etc in the "prescription" table.
  3) Different previliges were to be given to different users: Initially, all tables were visible to all users (doctors, administrators and patients).
  4) Foreign keys (which generally consisted of staff ids, order id, patient id, etc) in multiple tables, such as prescription, vendors, purchase orders etc were irrelevant, and we were requested to replace these numbers with the actual names or values instead of the ids.

5) For the medicine dosage, a particular format had to be implemented - instead of writing in words, the prescriptions had to be in the format x-y-z, where x corresponded to the number of tablets in the morning, y to the tablets after lunch, and z for tablets after dinner.
6) If possible, we were told to add an undo button, in case an entry was modified or deleted accidentally.
7) Upon deletion and updating, tables with foreign keys were facing issues, which had to be resolved.

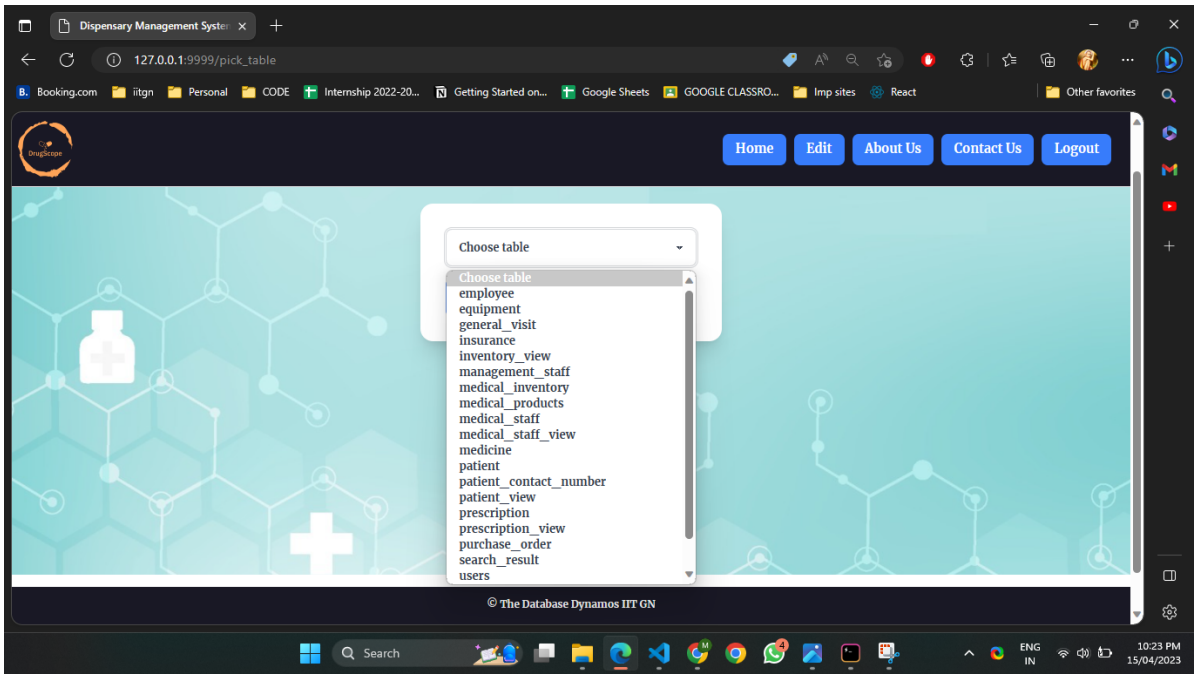
- **Second feedback:**
    8) A "Contact us" button was to be added, so that the patients or other users can get in contact with the administrators.
    9) A query for day-wise medicine use had to be implemented. This can be in the form of the prescriptions in a day. This issue can be solved by simple adding a search bar for the dates in the prescription table.

2. *Attach screenshots of different views [along with a write-up on their privileges] of the database as seen by different classes of users.*

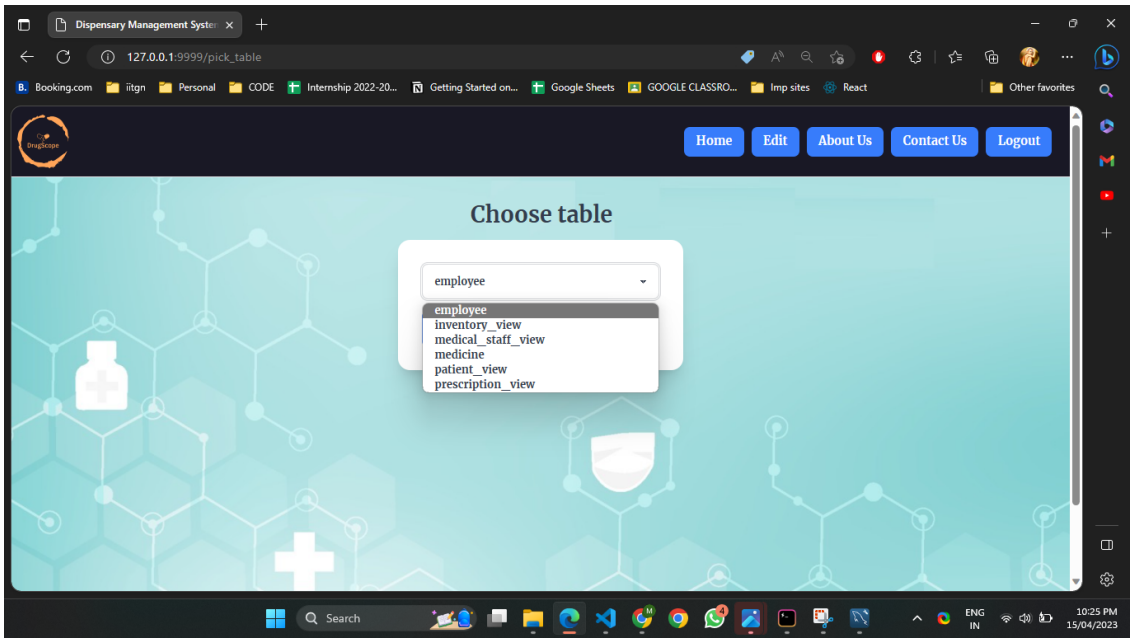
- Privileges were added for ensuring the right users can access only particular tables in the website. Earlier admin, doctors, and patients have access to the complete database system. Now we have limited the databases to doctors and patients. As told by stakeholders, the admin should be able to access the complete database. Doctors should be able to access only databases related to their work. And the patient should be only able to see the history.

- **<u>Views of different users:</u>**

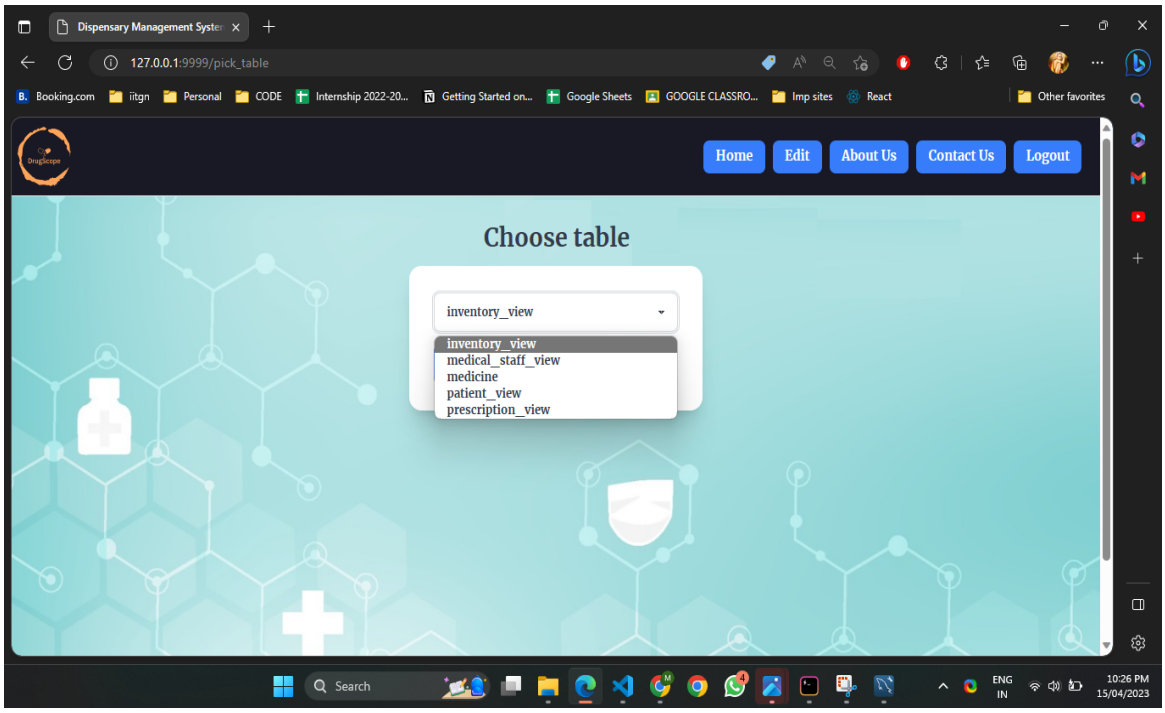
1. Admin



2. Doctor



3. Patient



In addition, the modify/insert/delete privileges were also modified as per the requirements.

- Only the administrator has full privileges to insert, modify, delete, and values.
- The doctor can only modify or delete values. However, he can use the filter button to find the required values.
- The patient does not have any privileges beyond accessing the tables and using the search button. The earlier patient was able to do all the operations as that admin. Now, the patient cannot delete, update, or insert a database.
- If the patient or the doctor tries to do operations that are not permitted, it will show the message “Action not permitted !”

Action not permitted!

← BACK + INSERT

FILTER

medical_staff_view

	STAFF_ID	NAME	ROLE	QUALIFICATION	SPECIALIZATION	LICENSE_ISSUE_DATE	HOME_CONTACT	CONTACT_NUMBER
	1001	John Smith	Doctor	MD	Cardiology	2015-06-01	555-555-5555	1234567890
	1002	Mary Johnson	Nurse	BSN	Pediatrics	2018-01-01	555-555-5555	0987654321
	1004	Sarah Davis	Doctor	MD	Dermatology	2017-08-01	555-555-5555	3456789012
	1007	Lisa Garcia	Nurse	BSN	Emergency	2020-02-01	555-555-5555	6789012345
	1008	Michael Chen	Doctor	MD	Oncology	2014-05-01	555-555-5555	7890123456

3.2 Responsibility of G2:

1. *Concurrent multi-user access: Multiple users with different roles can access and update the database concurrently. In such a scenario, the same item can not be updated by two different users. For example, locks can be applied to tables in MySQL.*

Concurrent multi-user access in the MySQL database can be given by using **table locks** in order to ensure that only one user can update a specific item at a time. Here we have used it on the **users** table. Here is the code snippet example that we can perform on **users** table according to our database-

```sql
-- CODE SNIPPET TO IMPLEMENT TABLE LOCKS
-- Start a transaction
START TRANSACTION;

-- Obtain a lock on the table to prevent other users from updating the same item
LOCK TABLES users WRITE;

-- Update the item in the table
UPDATE users SET password = '000000' WHERE id = 4;

-- Release the lock on the table
UNLOCK TABLES;

-- Commit the transaction
COMMIT;
```

In the above code, we initiate a transaction by writing the command **start transation**. This makes sure that all the changes that we are doing in the our database are executed atomically. then, we hamper the **write** right on the **users** table by writing down the command **lock tables**. This lock makes sure that only one user can alter/change/update the table at a time. Then, we alter/change/update the item in the table. Then, ultimately, we release the lock by writing the command **unlock tables**. Finally, we commit the transaction using the **commit** statement.

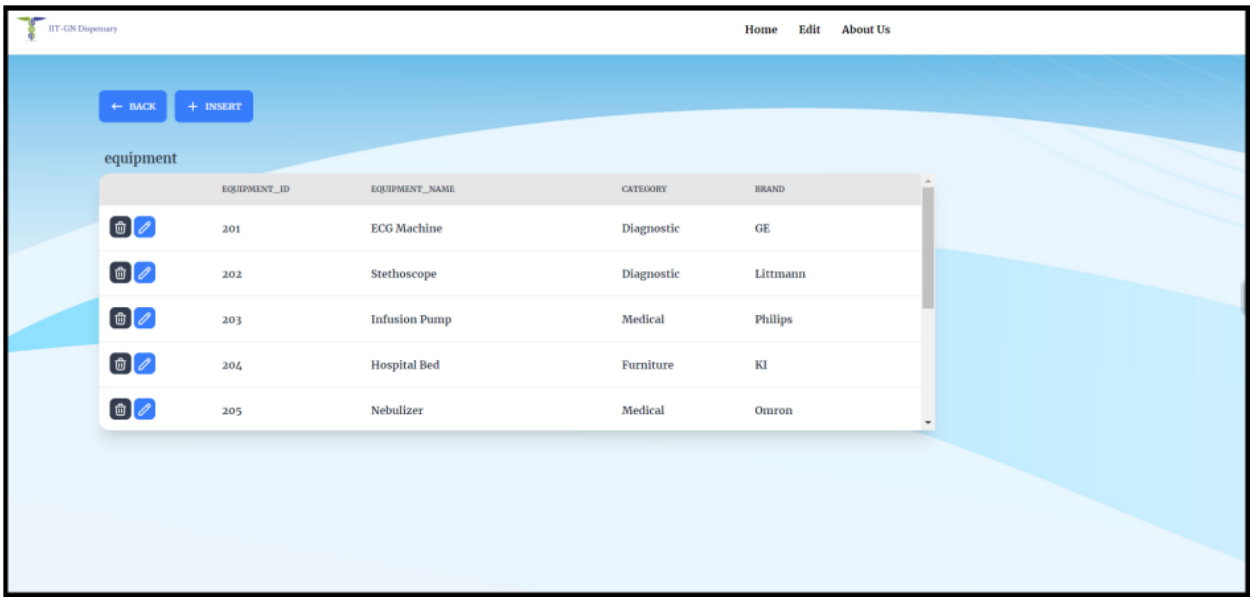
2. *Implement the changes in the database as per the feedback received from stakeholders.*

- **Implemented Solutions:**

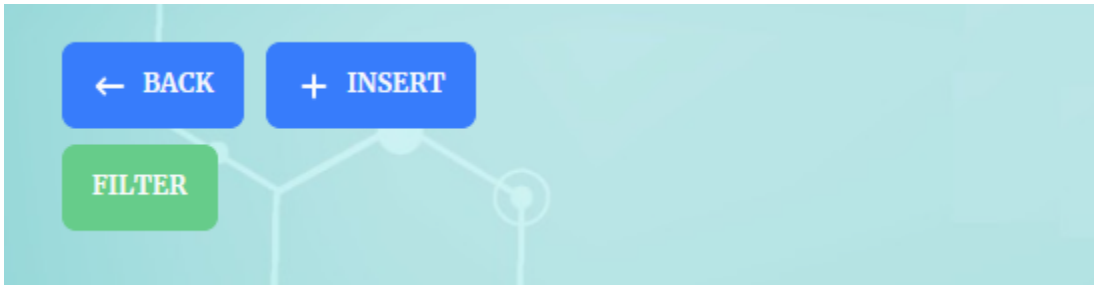
1) **A search button was implemented in the website in the top left corner.** This allowed the user to filter out the values from any selected column and show only those values. This helped us solve problems (1) and (6)

- **Search button:**

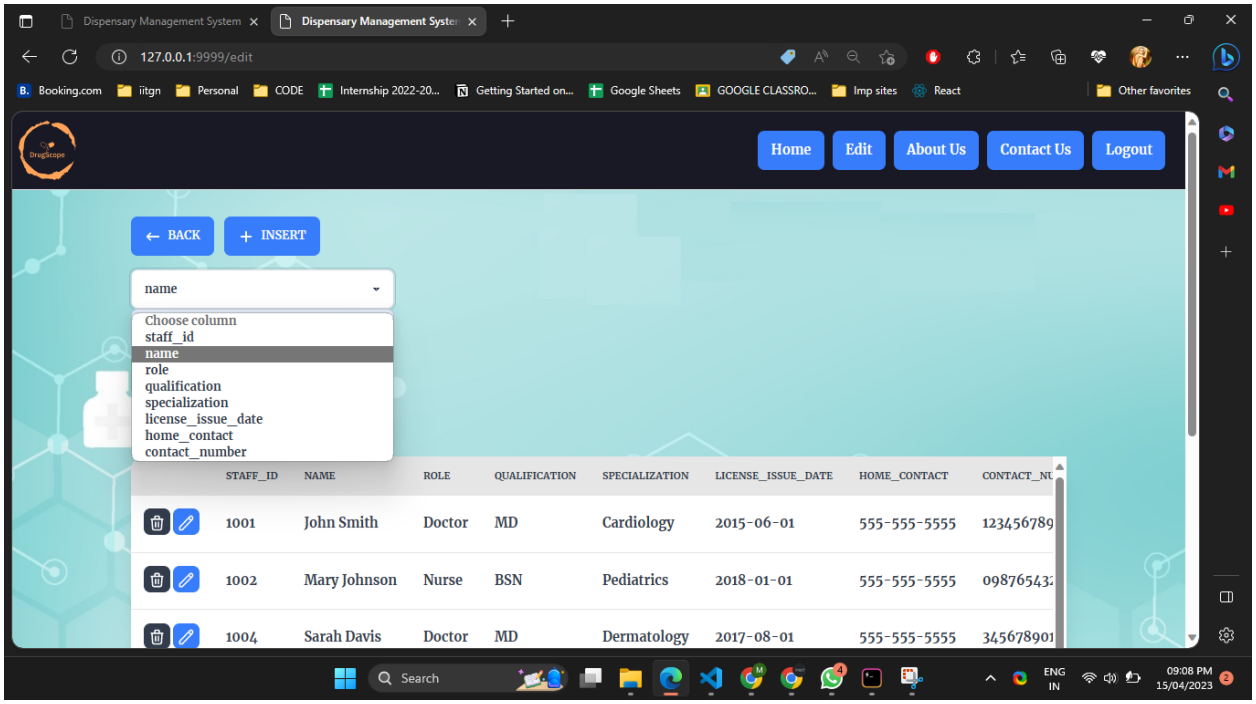
  Before Feedback:



  After Feedback:



The user can select any tuple. For example, to select the Staff_ID 1004. We first select the column and search the 10005. It will provide us with the tuple with Staff_ID 1004.



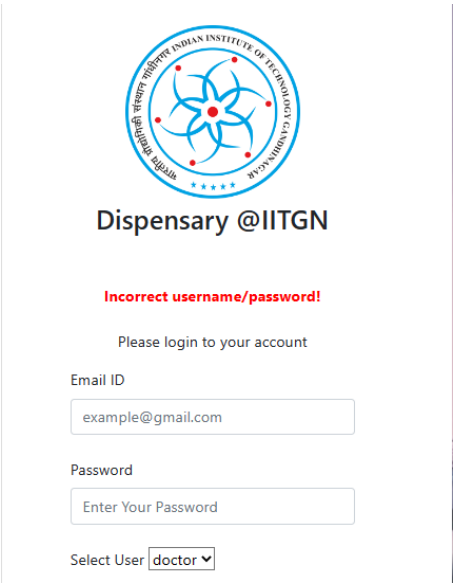
Query to select Staff_ID 1004:

Result:

2) **The prescription table in the MySQL database was modified such that it included the required tables, as well as changes to the purchase order table were made**. The correct format for the dosage in the medicines table was also implemented. This solved the problem (2) and problem (4)

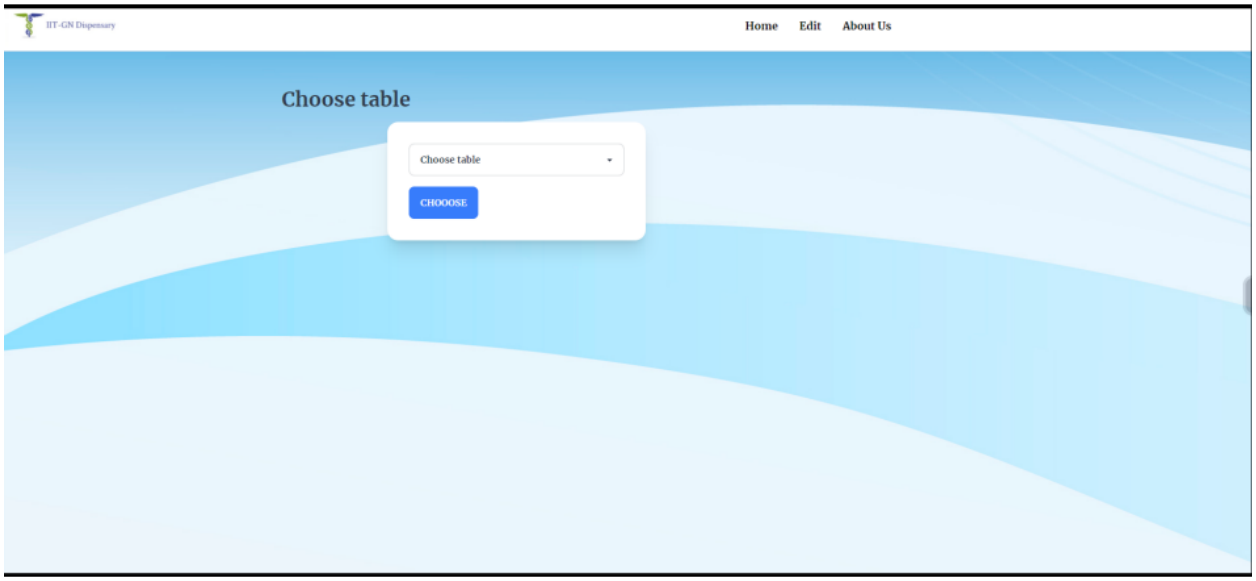
3) Stakeholders told us to show the message if an unknown user/admin/doctor tries to login to the website. We have updated the user login page. They also told us to have the logout button. We have also added the logout button in the navbar. Before feedback the message was not shown of unknown user log in.

After Feedback-



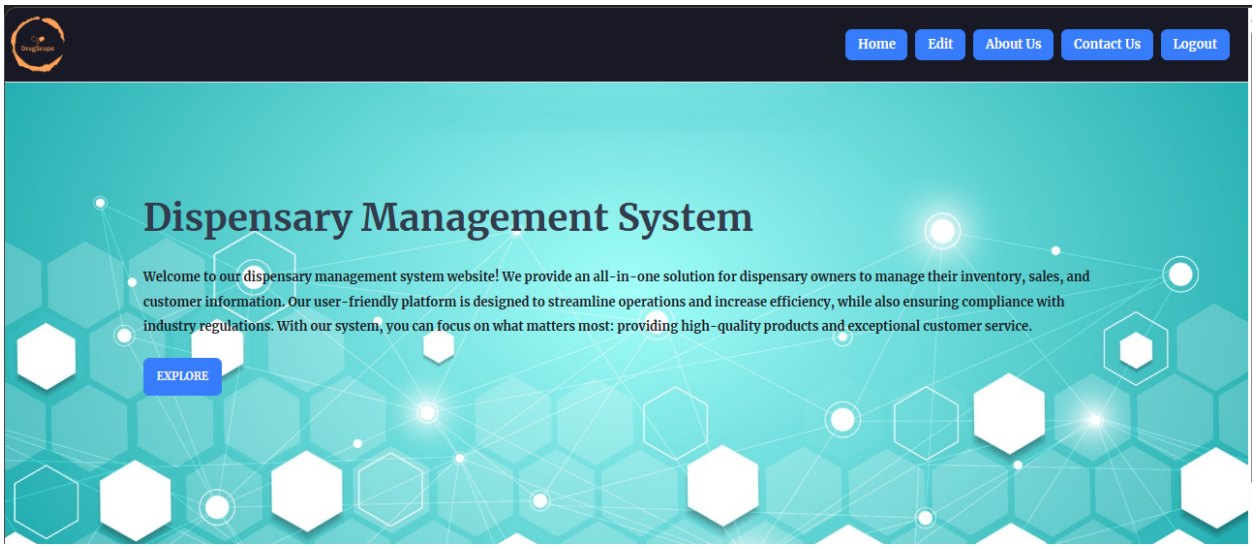
5. Logout Button:

Before Feedback-



After Feedback-

The stakeholders suggests us to make a way so that the user can logout from the website. So taking care of this feedback we made a “logout” button, by pressing this button, the user directly jump to the home-page.



6. Adding Views to eliminate foreign keys

Views were made for prescription, Inventory, Patient and Medical Staff, to eliminate the problem of foreign keys as discussed earlier. Although these views cannot be edited, we made sure that the admins had access to all the tables, so that if changes had to be made, they can be made through the original tables.

MySQL code for the views

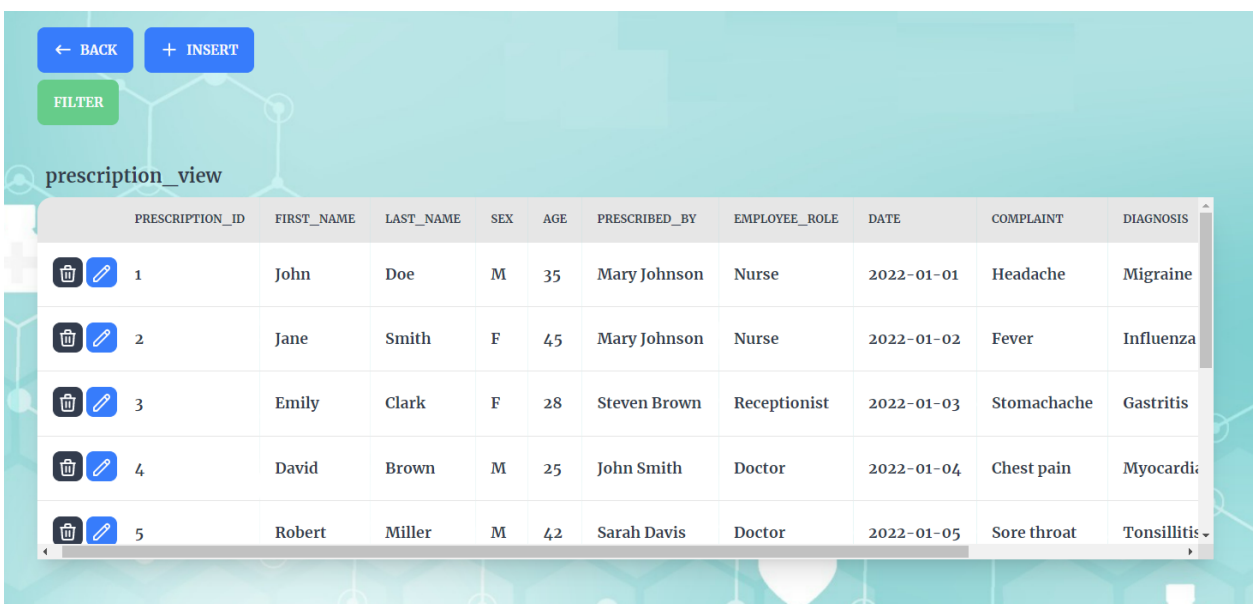
```sql
CREATE VIEW Prescription_View AS
SELECT
    Prescription.prescription_id,
    Patient.first_name as 'First_Name', Patient.last_name as 'Last_Name', Patient.gender as 'Sex',
    Patient.age as 'Age', Employee.name AS 'Prescribed_By', Employee.occupation AS 'Employee_Role',
    Prescription.date, Prescription.complaint, Prescription.diagnosis, Medical_Products.item_name AS 'Prescribed_Product'
FROM Prescription
JOIN Medical_Products USING(medical_id)
JOIN Employee USING(staff_id)
JOIN Patient USING(patient_code);


CREATE VIEW Inventory_view AS
SELECT mp.medical_id, mp.item_name, mp.cost, mi.stocks, mi.availability
FROM Medical_inventory mi
JOIN Medical_Products mp ON mi.medical_id = mp.medical_id;

CREATE VIEW Patient_view AS
SELECT Patient.*, Patient_contact_number.contact_number
FROM Patient
LEFT JOIN Patient_contact_number
ON Patient.patient_code = Patient_contact_number.patient_code;

CREATE VIEW medical_staff_view AS
SELECT ms.staff_id, e.name, ms.role, ms.qualification, ms.specialization, ms.license_issue_date, ms.home_contact, e.contact_number
FROM Medical_staff ms
LEFT JOIN Employee e ON ms.staff_id = e.staff_id;
```

For example, here is how the prescription view looks. Here, the medical_id was replaced with the name of the medicine, the Employee_id was replaced by the name of the prescriber, and the id of the patient was replaced by the first name and last name.

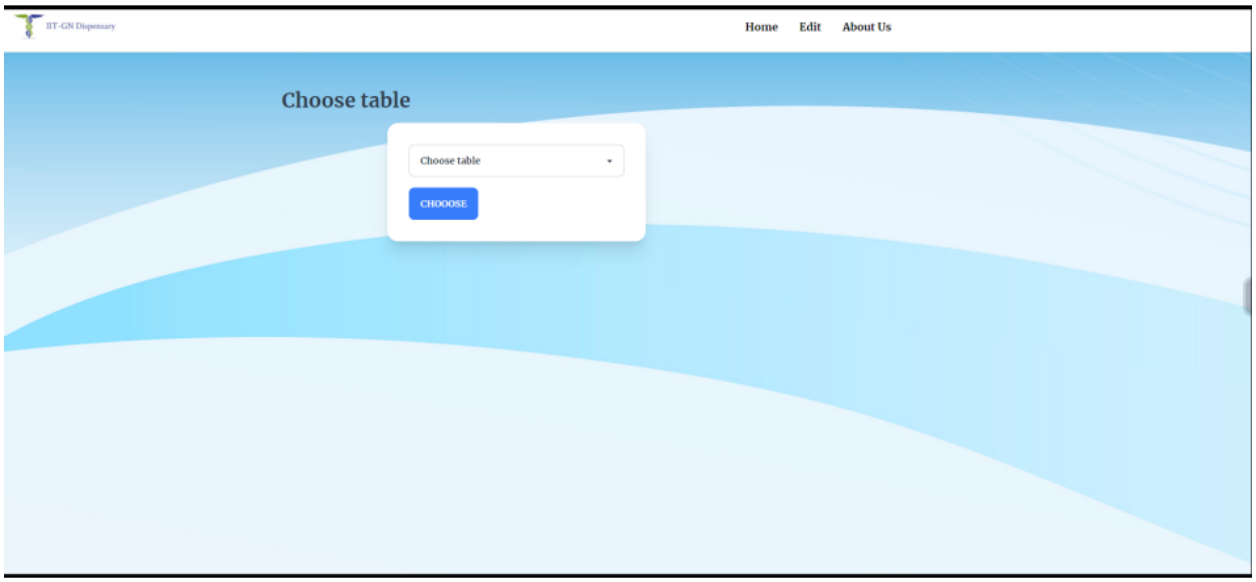


PRESCRIPTION_ID	FIRST_NAME	LAST_NAME	SEX	AGE	PRESCRIBED_BY	EMPLOYEE_ROLE	DATE	COMPLAINT	DIAGNOSIS
1	John	Doe	M	35	Mary Johnson	Nurse	2022-01-01	Headache	Migraine
2	Jane	Smith	F	45	Mary Johnson	Nurse	2022-01-02	Fever	Influenza
3	Emily	Clark	F	28	Steven Brown	Receptionist	2022-01-03	Stomachache	Gastritis
4	David	Brown	M	25	John Smith	Doctor	2022-01-04	Chest pain	Myocardi
5	Robert	Miller	M	42	Sarah Davis	Doctor	2022-01-05	Sore throat	Tonsillitis

Similar views were implemented for patient, medical staff and inventory.

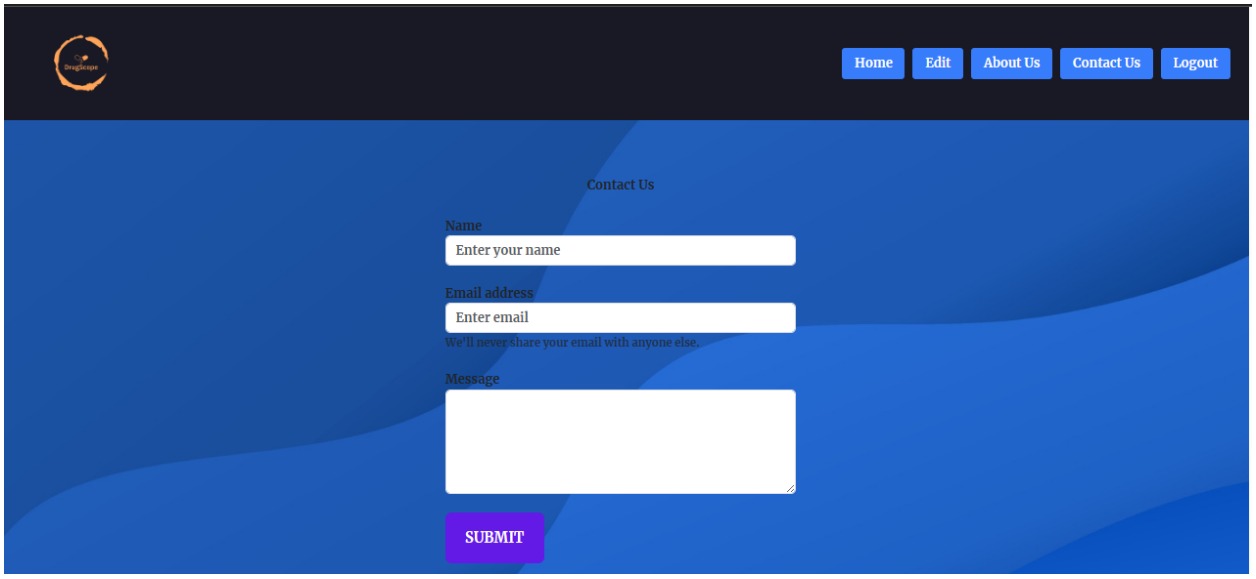
7) Contact Us button:

Before Feedback-

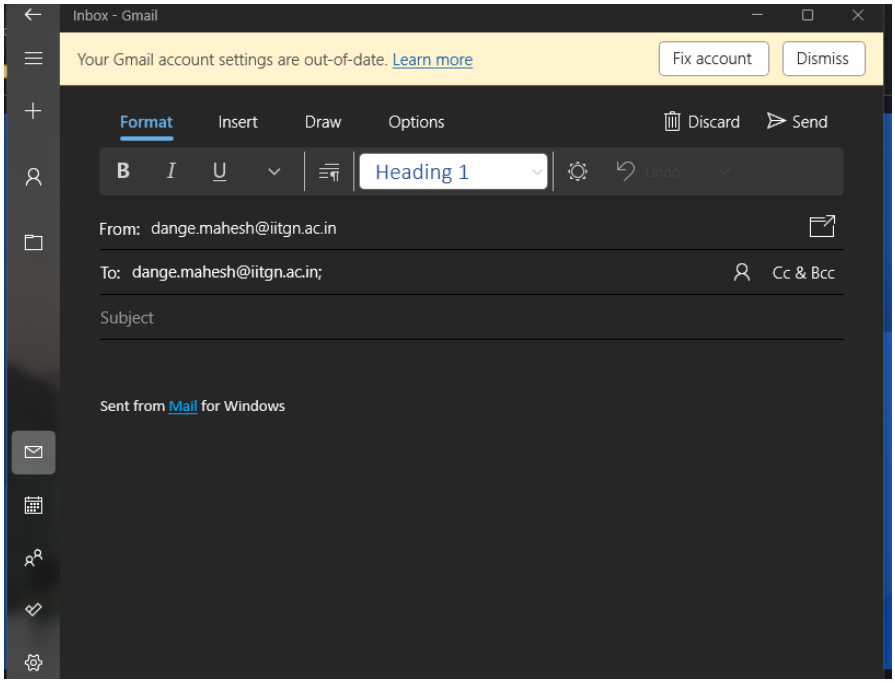


After Feedback-

We talked to the stakeholders, and they suggested us that everyone using the website should be able to contact the authorities, so we update our website by adding new “contact-us” button. By using this button, the user can get in touch with the admin.



After pressing the submit button, all the information will send to the admin through an Email (to the dispensary, as of now we have added the email of one of our group members). Here is the snapshot after clicking on the submit button.



8) In the medicine table, the required format of Recommended dosage was added.

This was according to the feedback given by the stakeholders we changed the format from the text to the “x-y-z” and x,y and z represents the number of tablets recommended to the patient in the morining, after lunch and after dinner respectively.

medicine

MEDICAL_ID	FORM	ACTIVE_INGREDIENTS	RECOMMENDED_DOSAGE	MANUFACTURING_DATE	SHELF_LIFE	EX
10001	Tablet	Acetylsalicylic acid	1-0-1	2021-01-01	3 years	2
10002	Capsule	Penicillin	1-1-1	2022-02-01	2 years	2
10003	Injection	Morphine	0-0-1	2021-07-01	1 year	2
10004	Tablet	Serotonin-norepinephrine reuptake inhibitors	1-1-0	2022-01-01	2 years	2

9) Resolving deletion and update errors in previous assignment.

Upon deletion and updating, tables with foreign keys were facing issues, which had to be resolved. This was a problem associated with the violation of the relational integrity of the table, and was solved in most cases by simply adding “ON UPDATE CASCADE ON DELETE CASCADE” to each table.

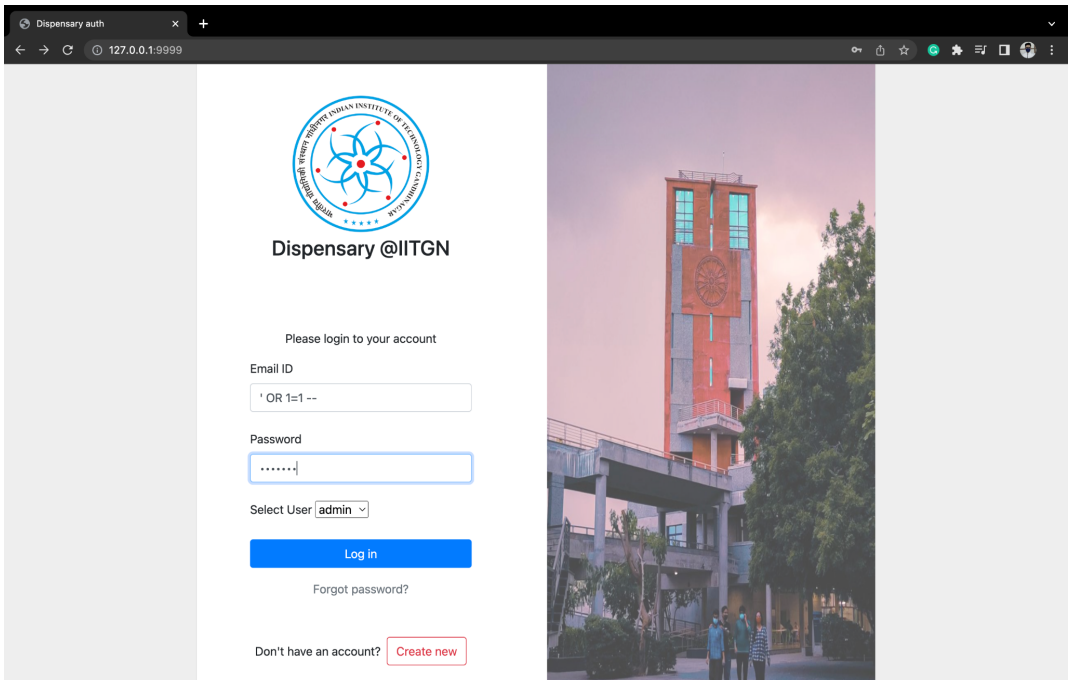
```
CREATE TABLE Medical_inventory (
    medical_id INT PRIMARY KEY,
    stocks INT NOT NULL DEFAULT 0,
    availability VARCHAR(50) NOT NULL DEFAULT 'available',
    FOREIGN KEY (medical_id) REFERENCES Medical_Products(medical_id) ON DELETE CASCADE ON UPDATE CASCADE
);
```

3.3 Responsibility of G1 & G2:

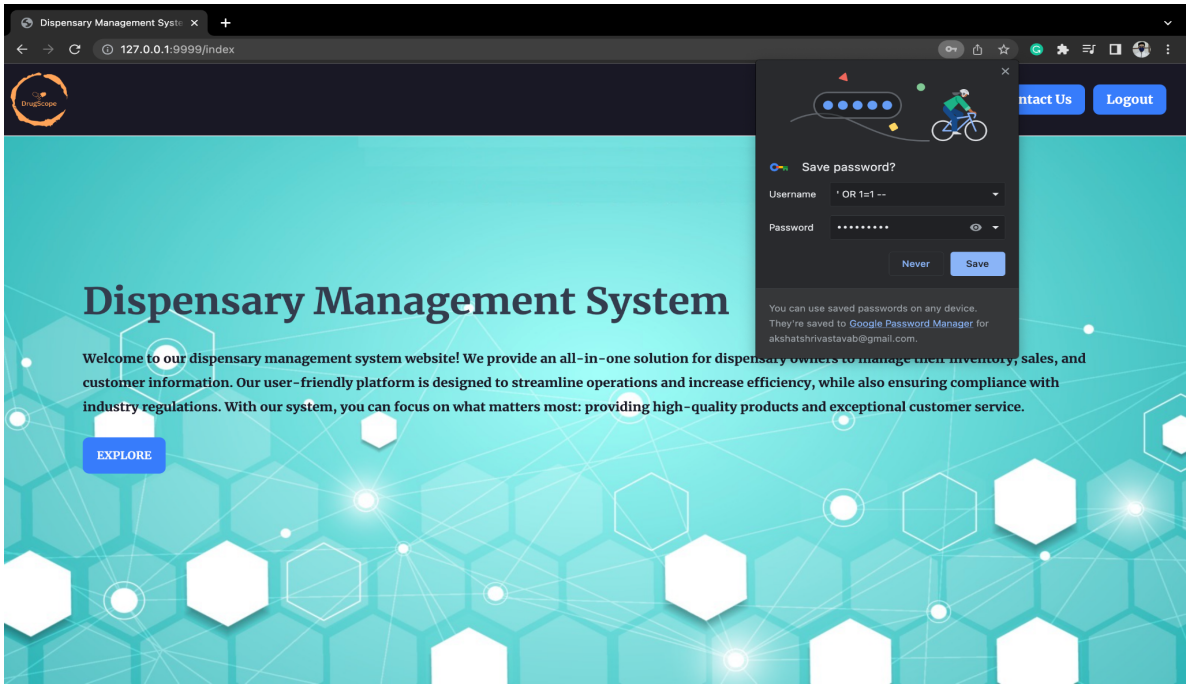
1. *Documentation and screenshots of a total of 2 attacks [SQL Injection and XSS] performed and the defenses against those attacks.*

- **SQL Injection Attack:**

The user who has not been given access to the website can log in to the dispensary app. An example of a user who is not authorized is given below :



The unknown user can login successfully to the website.



The query which leads to the attack is given below.

```
if (authority == "patient"):
    # cursor.execute('SELECT * FROM users WHERE email = % s AND password = % s AND role=%s',
    #                (useremail, password, "patient",))
    cursor.execute(f"SELECT * FROM users WHERE email = '{useremail}' AND password = '{password}' AND role = 'patient'")
    account = cursor.fetchone()
```

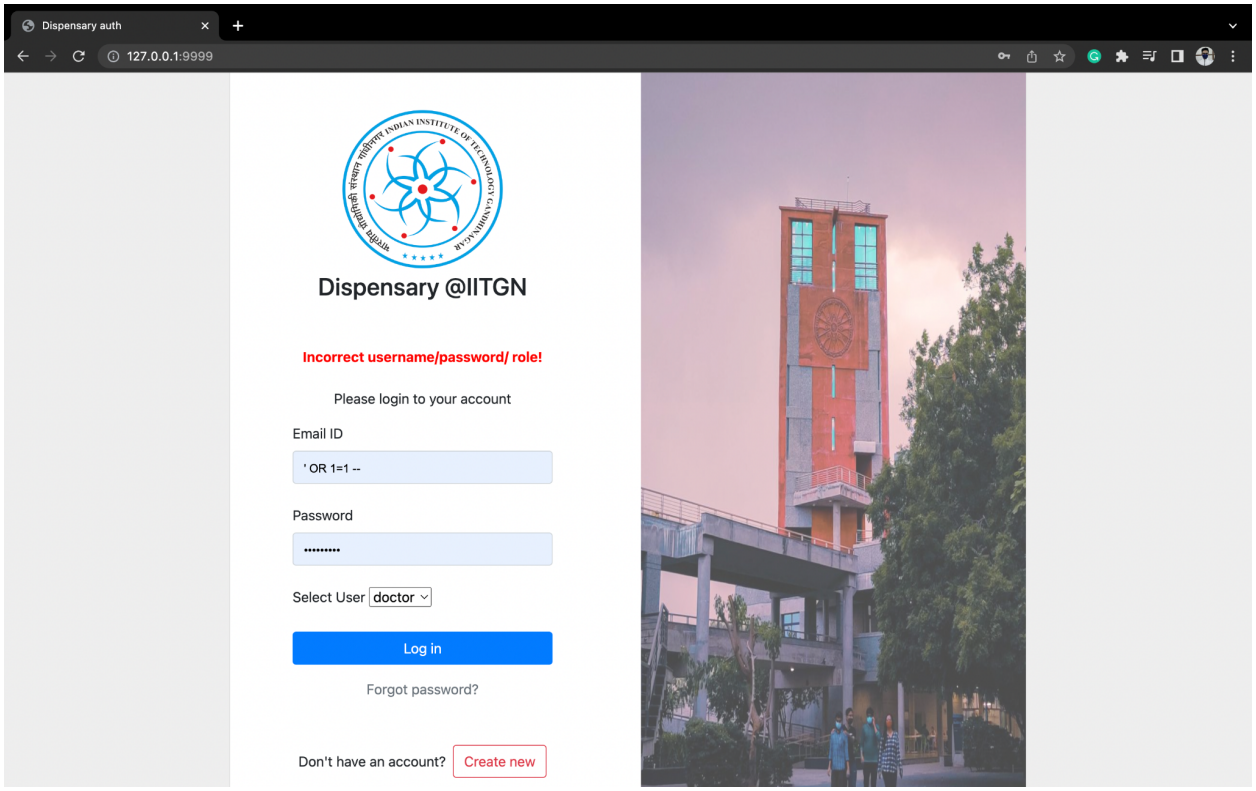
```
elif (authority == "doctor"):
    # cursor.execute('SELECT * FROM users WHERE email = % s AND password = % s AND role=%s',
    #                (useremail, password, "doctor",))
    cursor.execute(f"SELECT * FROM users WHERE email = '{useremail}' AND password = '{password}' AND role = 'doctor'")
    account = cursor.fetchone()
```

The updated query for defense is given below:

```
if (authority == "patient"):
    cursor.execute('SELECT * FROM users WHERE email = % s AND password = % s AND role=%s',
                   (useremail, password, "patient",))
```

```
elif (authority == "doctor"):
    cursor.execute('SELECT * FROM users WHERE email = % s AND password = % s AND role=%s',
                   (useremail, password, "doctor",))
```

After updating the SQL query, the unknown user cannot login into the website.



In the above Email-ID, after OR, 1=1 is true for all conditons and "--" will comment the next line, by which were are able to enter our website.

- **Cross-Site Scripting (XSS) Attack:**

In Flask, we can use the **Markup class** to safely escape **HTML** and **prevent XSS attacks**. Here's a code snippet example for the same:

```
#XSS PREVENTION EXAMPLE
@app.route('/')
def XSS_query():
    user_input = '<script>alert("XSS attack!")</script>'
    safe_input = Markup.escape(user_input)
    return render_template('blank.html', safe_input=safe_input)
```

In this, we have a route that returns an HTML template called blank.html. We have a variable **user_input**which contains a potential **Cross-Site Scripting (XSS)** attack in the form of a **script tag**. In order to prevent the XSS attack, we can use the **Markup.escape()** method in order to safely escape the input provided by **user_input** and then, store it in a new variable **safe_input.** Then, we pass this **safe_input** to the template using Flask's render function.

Thus, we can safely show or bypass the script tag.

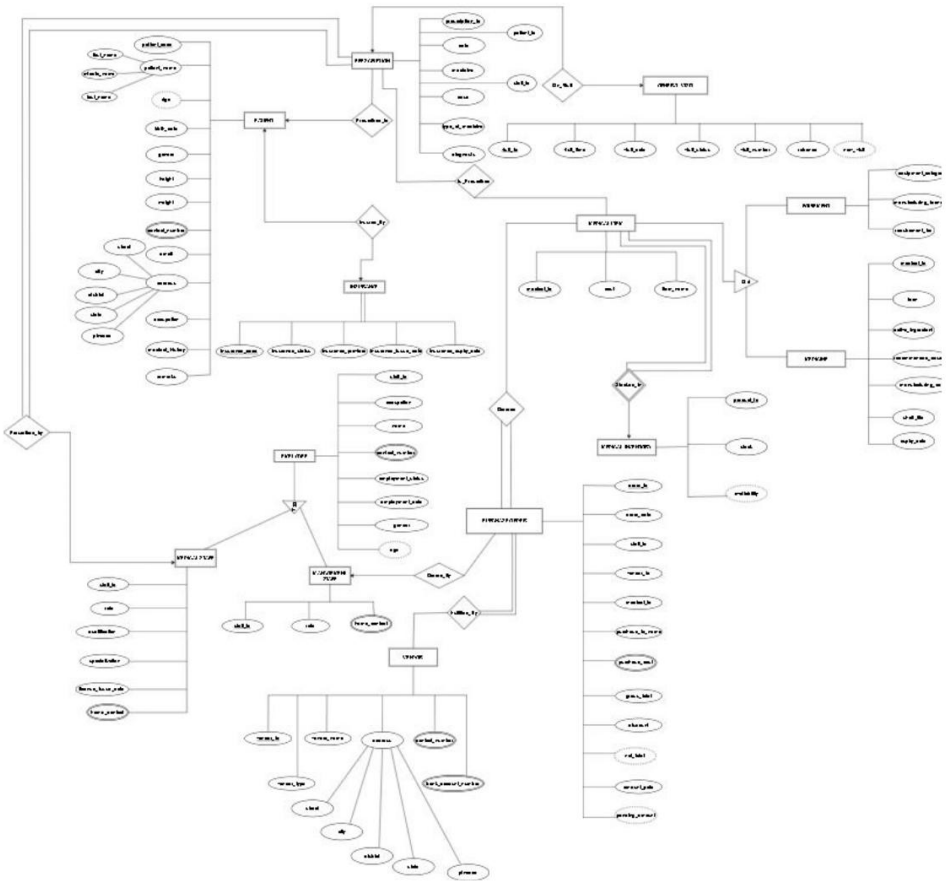
- **Cross-Site Request Forgery (CSRF) Attack:** This vulnerability (or attack) can occur when an attacker tricks a user into unknowingly performing an action on a website, by exploiting the user's existing session with the website

2. *Show that all the relations and their constraints, finalized after the second feedback, are present and valid as per the ER diagram constructed in Assignment 1.*

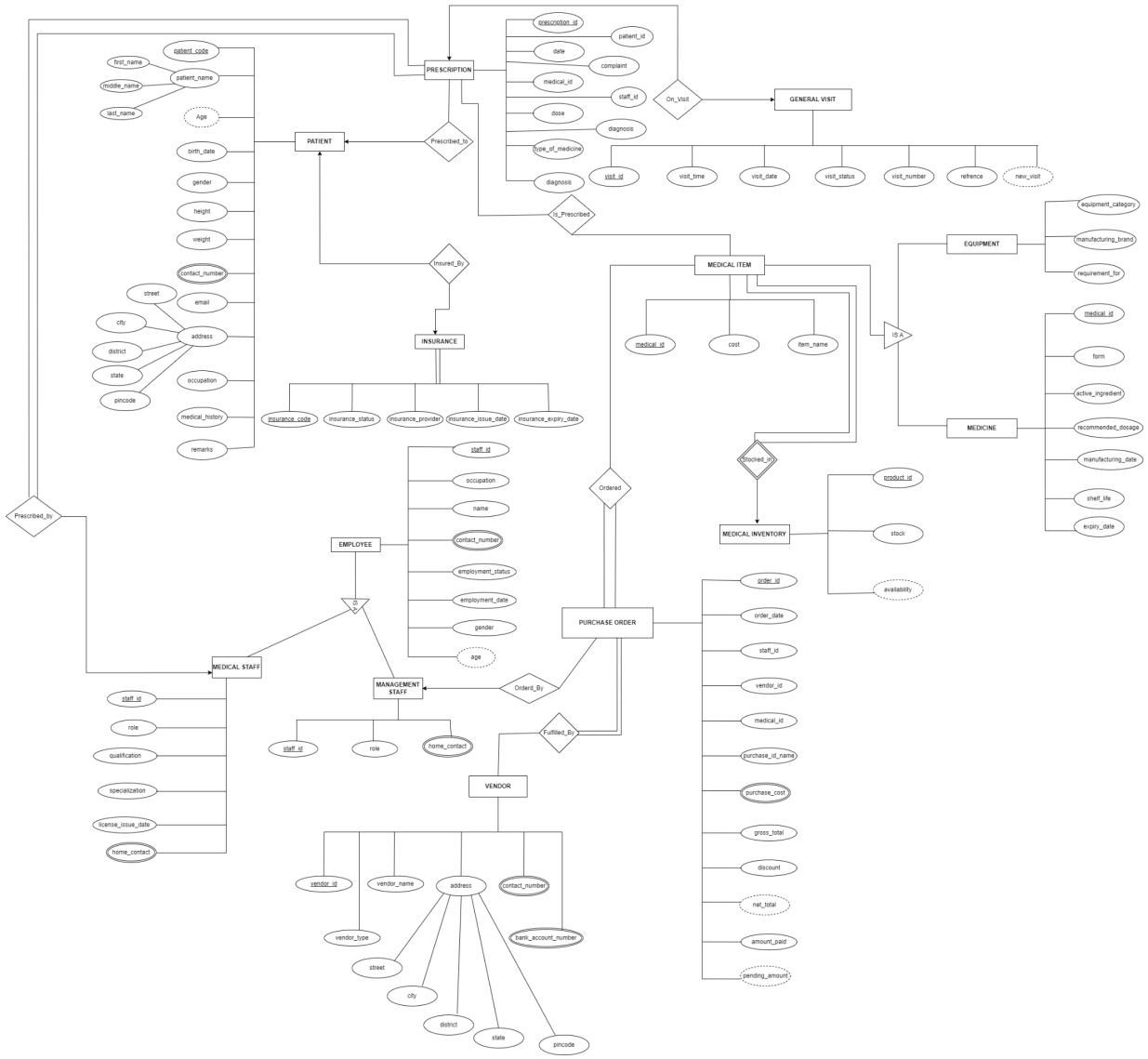
   Throughout the making of the database, we made sure that all the relations and constraints in the relational schema were maintained in the database. Almost all the attributes and entities were maintained in the database, with minor changes to a few attributes. For example, the prescription table was modified after the first feedback, to include additional attributes.

   No additional changes to the relations were made in the database. This can also be viewed in the updated ER Diagram which mirrors the relations in the database as per the time of the submission.

Before the feedback:



After the feedback:



Contribution:

Team Member	Contribution
1. Aishwarya Omar	SQL Injections.
2. Akshat Shrivastava	Frontend development according to feedback and debugging, backend development, did SQL injection and defense, managing privileges for different users and Report documentation. (G2)
3. Amaan Ansari	–dropped–
4. Mahesh Dange	Meetings with the stakeholders, Frontend Development, and Backend Development after taking the feedback from the stakeholders, Report Documentation.
5. Dhakad Bhagat Singh	Report documentation, SQL injections, Finding vulnerabilities in the web application
6. Govardhan Ingale	Feedback, foreign key removal
7. Ronak Hingonia	Meetings with stakeholders, documentation/formatting of the report. Undo feature attempted.
8. Ojas Washimkar	Report documentation, SQL injections, Finding vulnerabilities in the web application
9. Yash Adhiya	Foreign Key removal.
10. Yash Kokane	Meetings with stakeholders, feedback form submission, foreign key removal, modifying MySQL file as per requirements, adding privileges for different users, documentation of report.

11. Shubham Kumar	Report Documentation, Helped in SQL injection