



CO527: Advanced Database System

Organ Transplantation Management System

Made By:

E/14/010

E/14/028

E/14/065

Title: Organ Transplantation Management System

Problem Statement:

Organ transplantation is a medical procedure in which an organ is removed from one body and placed in the body of a recipient, to replace a damaged or missing organ. The donor and recipient may be at the same location, or organs may be transported from a donor site to another location.

Organ Donation and Procurement Organizations play a pivotal role in today's medical institutions. Such organizations are responsible for the evaluation and procurement of organs for organ transplantation. These organizations represent the front-line of organ procurement, having direct contact with the hospital and the family of a recently deceased donor. The work of such organizations includes to identify the best candidates for the available organs and to coordinate with the medical institutions to decide on each organ recipient. They are also responsible for educating the public to increase the awareness of and participation in the organ donation process. Also, it keeps track of all transplantation operations carried till date.

The Organ Donation and Procurement Network Management System is a database management system that uses database technology to construct, maintain and manipulate various kinds of data about a person's donation or procurement of a particular organ. It maintains a comprehensive medical history and other critical information like blood group, age, etc of every person in the database design. In short, it maintains a database containing statistical information regarding network of organ donation and procurement of different countries.

Organ Wastage is a major issue that can only be solved by having a proper database of all Patient and Donors in a well-formed way, that can be processed easily.

Records of donor and patients are created when a person donates or procures an organ from a Medical Institution. Records may include the following information: -

1. Personal Information
2. Medical History
3. Medical insurance, if any
4. Allergies to any medicine, if any
5. The need for an organ presently
6. Medical Insurance provided by any private or government insurers.
7. Address

This record serves a variety of purposes and is critical to the proper functioning of Organ Donation and Procurement Network, especially in today's complicated health care

environment. These records provide statistical information regarding the number of organs needed and available at a particular point of time. It is essential for planning, evaluating and coordinating organ donation and procurement.

Our aim to create a solution that effectively deals with the problems of finding donors and also providing Statistical data of the transplants that can help the government to form better rules and regulations.

Basic Steps in Implementation:

- Every user has an account with can only be registered by a government certified hospital, which will keep all the information as defined in Problem Statement.
- Only Hospitals are eligible to request for a donation or procurement transaction.
- Government organizations will keep a watch on the pairing of donors and Patients and can approve a transplantation operation if all the rules are satisfied.
- Collecting Statistical Data through the history of Transplantation Transaction

Technologies Used:

- MYSQL
- HTML
- CSS
- Python
- Flask

ER Analysis: Identifying Entity Sets and Relationship Sets:

Entity Sets:

- 1. User**
 1. User ID
 2. Name
 3. Date of birth
 4. Phone Number (multi-valued)
 5. Medical Insurance
 6. Medical History
 7. Address
- 2. Patient**
 1. Patient_ID
 2. Organ Required
 3. Reason of procurement
 4. User_ID (foreign key)
- 3. Donor**
 1. Donor_ID
 2. Organ Donated
 3. Reason of donation
 4. User_ID (foreign key)
- 4. Organ Available**
 1. Organ_ID

2. Organ Name
3. Donor_ID (foreign key)
- 5. Organization**
 1. Organization ID
 2. Organization Name
 3. Location
 4. Government approved organization or not
 5. Phone Number (multi-valued)
- 6. Doctor**
 1. Doctor ID
 2. Doctor Name
 3. Phone Number (multi-valued)
- 7. Organization Head**
 1. Head Name
 2. Date of Joining
 3. Term Length

Relationship Sets:

1. **Donates** – The act of donation of an organ from a donor
 1. Date – Date of donation
2. **Procures** - The act of procuring an organ by the patient
3. **Transaction**
 1. Date of transaction
 2. Status – whether the surgery was successful or not
4. **Organ Donated** -The organ donated by a donor, which is then stored in Organ_available table.
5. **Attended By** -The transplantation performed by doctor – procuring an organ from a donor and transplanting it to the patient by surgery.
6. **Registers** - Donor is registered in which organization
7. **Works in** – The organization where the doctor works.
8. **Headed By** – The organization is headed by which person

Security:

```
mydb_admin = mysql.connector.connect(
    host = app.config['MYSQL_DATABASE_HOST'],
    port = app.config['MYSQL_DATABASE_PORT'],
    user = app.config['MYSQL_DATABASE_USER'],
    password = app.config['MYSQL_DATABASE_PASSWORD'],
    database = app.config['MYSQL_DATABASE_DB']
)
mycursor_admin = mydb_admin.cursor(buffered=True)

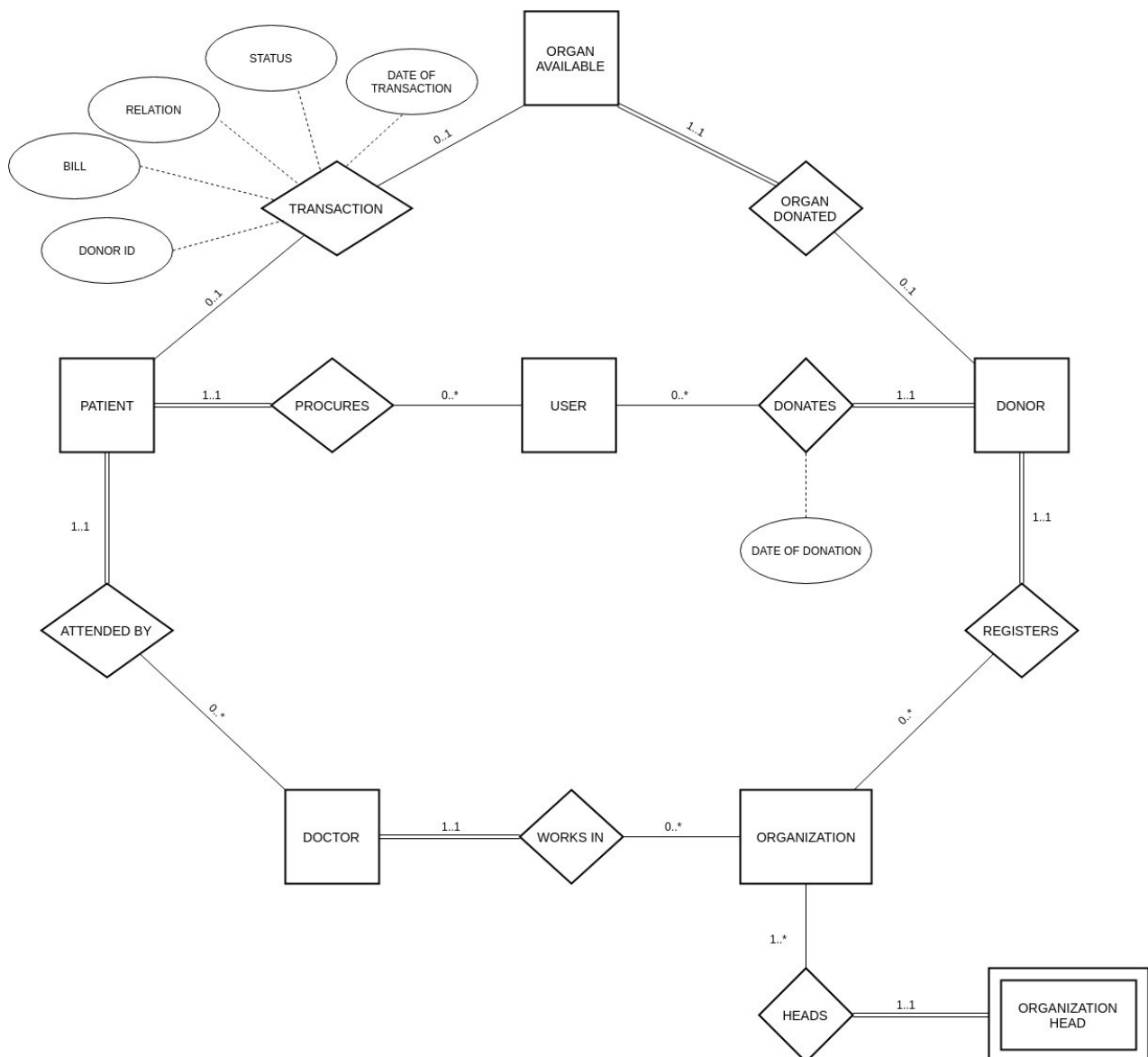
# for security reasons establish another connection for managers
# with only select, update, insert privileges
query = "CREATE USER IF NOT EXISTS 'manager'@'localhost' IDENTIFIED BY 'manager@123'"
mycursor_admin.execute(query)

# set the privileges for manager access database user
# doesn't allow to delete any record
query = "GRANT INSERT, SELECT, UPDATE ON *.* TO 'manager'@'localhost'"
mycursor_admin.execute(query)

mydb_manager = mysql.connector.connect(
    host = app.config['MYSQL_DATABASE_HOST'],
    port = app.config['MYSQL_DATABASE_PORT'],
    user = "manager",
    password = "manager@123",
    database = app.config['MYSQL_DATABASE_DB']
)
mycursor_manager = mydb_manager.cursor(buffered=True)
```

In the figure, you can see I have established two database user connections for each different access roles we have in the system so far. Basically, the supper admin and the manager roles are there. Manager doesn't have access to delete any record. The database user we are using to manipulate data has privileges on insert, select & update operations. Therefore, if any use logged as manager uses the that limited privileged database connection. For admins, system uses the connection has full privileges.

ER DIAGRAM



Comparison with other data models:

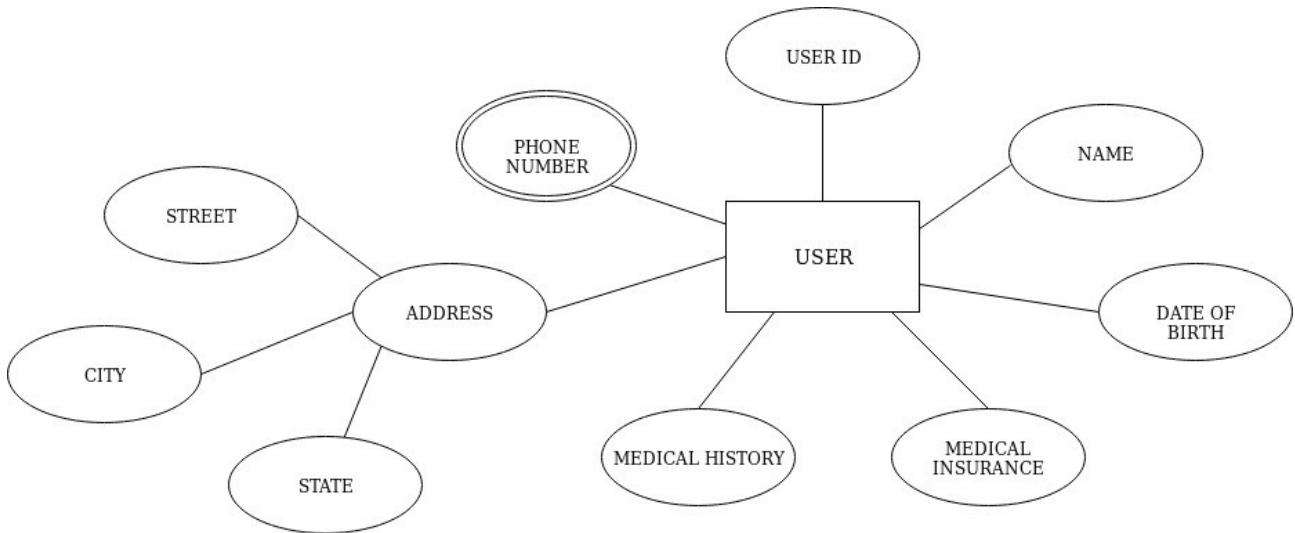
MySQL is a relational database that is based on tabular design whereas NoSQL is non-relational in nature with its document-based design. The detailed database model is required before creating the database in MySQL whereas no detailed modeling is need in the case of NoSQL database types.

MySQL is a relational database whereas NoSQL is more of design-based database type such like MongoDB. MySQL is being used with a standard query language called SQL and NoSQL like databases misses a standard query language. MySQL like a relational database can provide a performance issue for a large bunch of data, hence require optimization of queries whereas NoSQL databases like MongoDB are good at performance even with the dataset is huge.

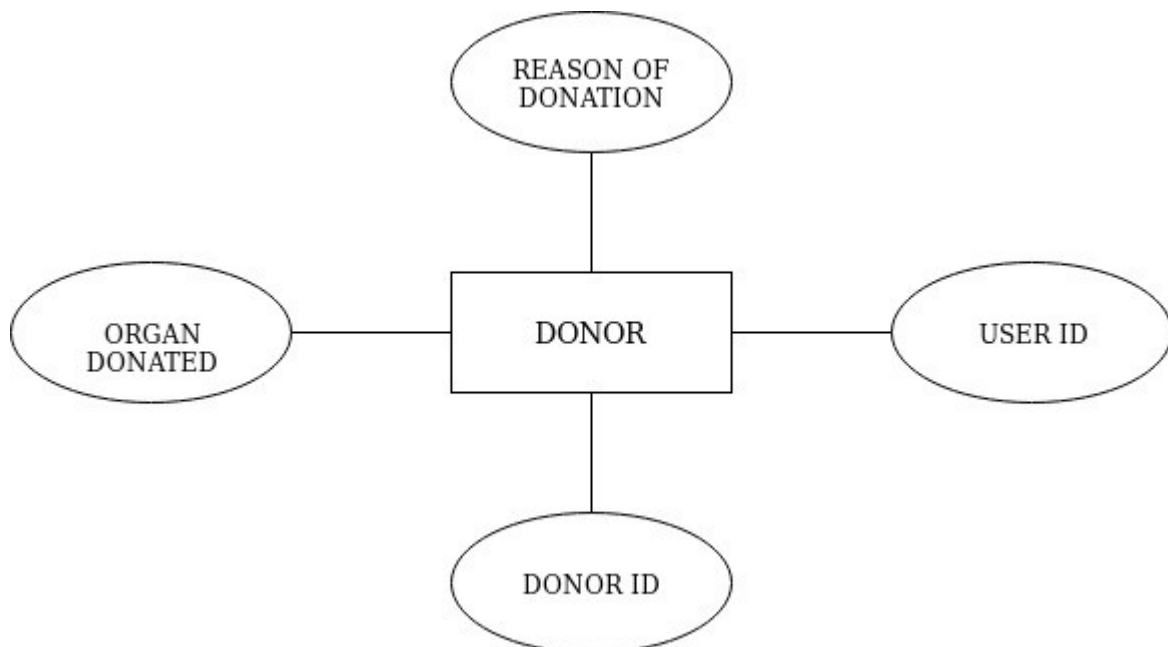
Therefore, we decided to use MySQL for the database implementation because when analyzing the relations among the tables we have more relations with more foreign keys.

Entity Sets

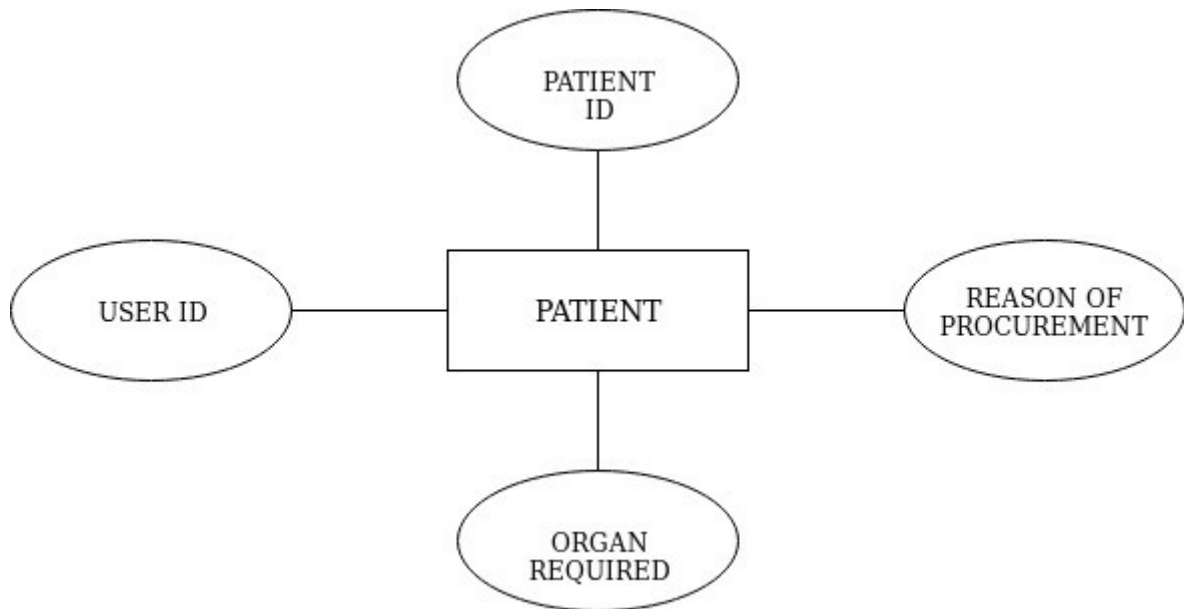
1) User -



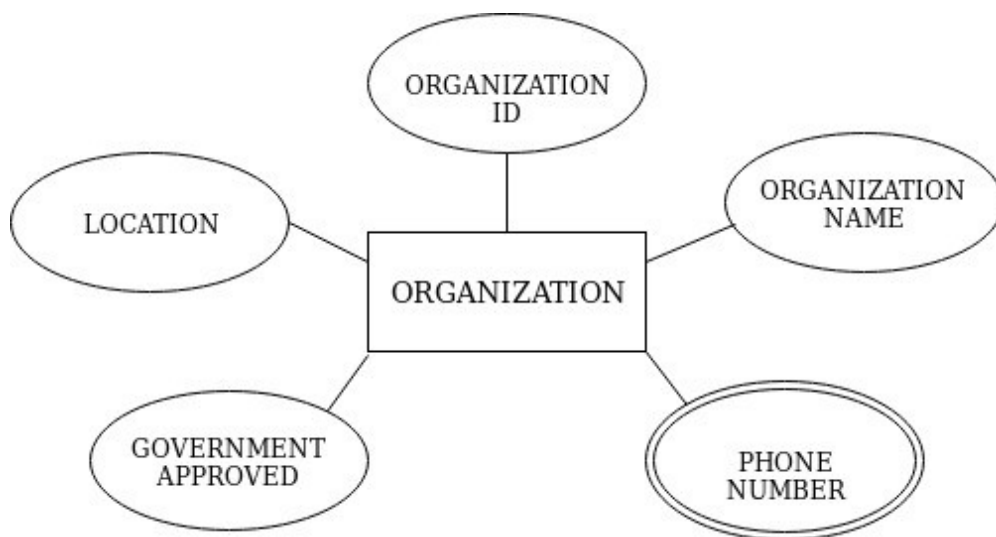
2) Donor



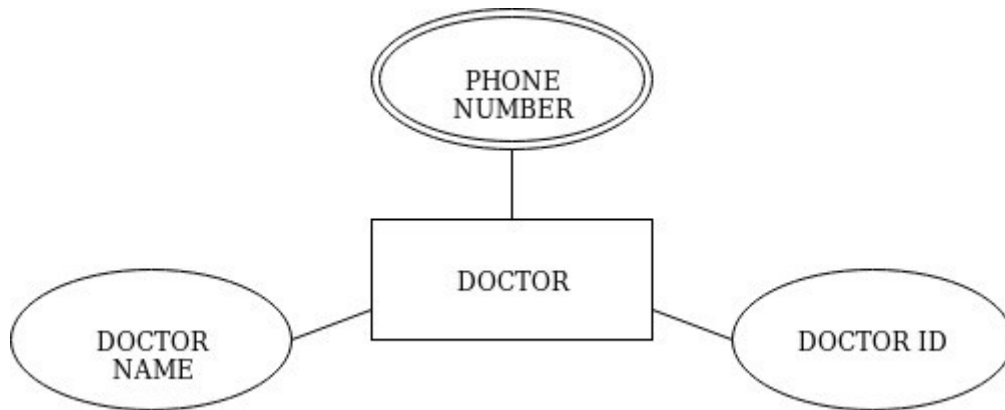
3) Patient



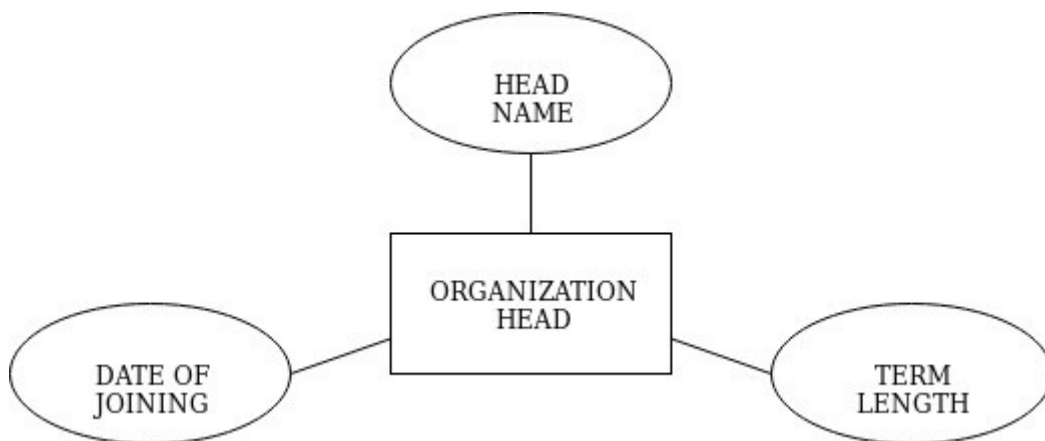
4) Organization



6) Doctor



7) Organization Head



Tables and their Functional Dependencies :-

1) **User**(User_ID, Name, Date _of_birth, Medical_Insurance, Medical_History, Street, City)

FD={User_ID \rightarrow Name, Date _of_birth, Medical Insurance, Medical History, Street, City}

2) **User_phone_no**(User_ID, phone_no)

FD={User_ID \rightarrow phone_no}

{User_ID} is foreign key constraint

3) **Patient**(Patient_ID, organ_req, reason_of_procurement, Doctor_ID, User_ID)

FD={Patient_ID, organ_req \rightarrow reason_of_procurement, Doctor_ID, User_ID}

{User_ID, Doctor_ID} are foreign key constraints

4) **Donor**(Donor_ID, organ_donated, reason_of_donation, Organization_ID, User_ID)

FD={Donor_ID, organ_donated \rightarrow reason_of_donation, Organization_ID, User_ID}

{User_ID, Organization_ID} are foreign key constraints

5) **Organ Available**(Organ_ID, Organ_name, Donor_ID)

FD={Organ_ID \rightarrow Organ_name, Donor_ID}

{Donor_ID} is foreign key constraint

6) **Transaction**(Patient_ID, Organ_ID, Donor_ID, Date_of_transaction, Status)

FD={Patient_ID, Organ_ID \rightarrow Donor_ID, Date_of_transaction, Status}

{Patient_ID, Donor_ID} are foreign key constraints

7) Organization(Organization_ID, Organization_name, Location, Government_approved)

FD={Organization_ID -> Organization_name, Location, Government_approved}

8) Organization_phone_no(Organization_ID, phone_no)

FD={Organization_ID -> phone_no}

{Organization_ID} are foreign key constraints

9) Doctor(Doctor_ID, Doctor_name, Department_name, Organization_id)

FD={Doctor_ID -> Doctor_name, Organization_id}

{Organization_ID} is foreign key constraint

10) Doctor_phone_no(Doctor_ID, phone_no)

FD={Doctor_ID -> phone_no}

{Doctor_ID} is foreign key constraint

11) Organization_head(Organization_ID, Employee_ID, Name, Date_of_joining, Term_length)

FD={Organization_ID, Employee_ID -> Name, Date_of_joining, Term_length}

Triggers

The following triggers are added to create a log of actions done on database. The logs are added to the log table.

1) Trigger for adding Donor information to Log table.

```
delimiter //  
create trigger ADD_DONOR_LOG after  
insert  
on Donor  
for each row  
begin  
insert into log values  
(now(), concat("Inserted new Donor",  
cast(new.Donor_Id as char)));  
end // delimiter  
;
```

2) Trigger for adding “Update” action information in Log table.

```
create trigger UPD_DONOR_LOG after  
update  
on Donor  
for each row  
begin  
insert into log values  
(now(), concat("Updated Donor Details",  
cast(new.Donor_Id as char)));  
end // delimiter  
;
```

3) Trigger for adding “Delete” action information in Log table.

```
create trigger DEL_DONOR_LOG after  
delete  
on Donor  
for each row  
begin  
insert into log values  
(now(), concat("Deleted Donor ",  
cast(old.Donor_Id as char))); end //  
delimiter ;
```

4) Trigger for adding “Add patient” action information in Log table

```
create trigger ADD_PATIENT_LOG after
insert
on Patient for
each row begin
insert into log values
(now(), concat("Inserted new Patient ",
cast(new.Patient_Id as char))); end //
delimiter ;
```

5) Trigger for adding “Update information” action information in Log table

```
create trigger UPD_PATIENT_LOG after
update
on Patient for
each row begin
insert into log values
(now(), concat("Updated Patient Details ",
cast(new.Patient_Id as char)));
end // delimiter
;
```

6) Trigger for adding “Delete information” action information in Log table

```
create trigger DEL_PATIENT_LOG after
delete
on Donor
for each row
begin
insert into log values
(now(), concat("Deleted Patient ",
cast(old.Donor_Id as char)));
end // delimiter
;
```

7) Trigger for adding “Add transaction” action information in Log table

```
create trigger ADD_TRANSACTION_LOG after
insert
on Transaction for
each row begin
insert into log values
(now(), concat("Added Transaction :: Patient ID : ",
cast(new.Patient_ID as char), "; Donor ID :
",cast(new.Donor_ID as char))); end //
delimiter ;
```

Query Optimized Using Indexes

```
CREATE INDEX user_id_index ON User (User_ID);
```

```
CREATE INDEX patient_id_organ_req_index ON Patient (Patient_ID, organ_req);
```

```
CREATE INDEX donor_id_organ_donated_index ON Donor (Donor_ID,
organ_donated);
```

```
CREATE INDEX organization_head_index ON Organization_head(Organization_ID,
Employee_ID);
```

```
CREATE INDEX transaction_index ON Transaction(Donor_ID, Status);
```

```
CREATE INDEX patient_id_user_id_idx ON Patient(User_ID, Patient_ID);
```

```
CREATE INDEX donor_id_user_id_idx ON Donor(User_Id,Donor_ID);
```

Backups

```
MYSQLDUMP -u root -p otms > otms_backup.sql
```

Transactions

1) Whenever a donor is added to the Donor Table, a corresponding organ must be added to the Organ_available table. So the two insert commands must be atomic. We have created the following transaction for this purpose

```
-- 1. start a new transaction START
TRANSACTION;

-- 2. insert into Donor table
INSERT INTO Donor values ( _, _, _, _, _ );

-- 3. insert into Organ_available table INSERT INTO
Organ_available ( _, _ );

-- 4. commit changes
COMMIT;
```

2) Whenever a transaction takes place, the record corresponding to that Organ_ID must be deleted from Organ_available table. So the insert and delete commands must be atomic. We have created the following transaction for this purpose.

```
-- 1. start a new transaction START
TRANSACTION;

-- 2. insert into Donor table
INSERT INTO Transaction values ( _, _, _, _,
_ );

-- 3. delete from Organ_available table
DELETE FROM Organ_available where Organ_ID = _;

-- 4. commit changes
COMMIT;
```

Procedure to run

Procedure to run on your computer:

The Project Uses:

1. MySQL
2. HTML 5
3. Python
4. Flask Framework
5. CSS
6. Bootstrap
7. JavaScript
8. Ajax

Steps to run:

Step 1. Making the database:

Import **otms_create_database.sql** to create the database & tables.

Then go to the database and import **otms_insert_data.sql** to it.

Step 2. Make sure do the changes (password, port...) in config.py to your MySQL.

```
class OTMSConfig(object):  
    MYSQL_DATABASE_USER = 'user'  
    MYSQL_DATABASE_PASSWORD = 'password'  
    MYSQL_DATABASE_DB = 'database_name'  
    MYSQL_DATABASE_PORT = 'port'  
    MYSQL_DATABASE_HOST = 'host'
```

Step 3. Run main.py.

Step 4. Go to localhost:/5000 on browser.

Admin

Username - admin

Password -admin

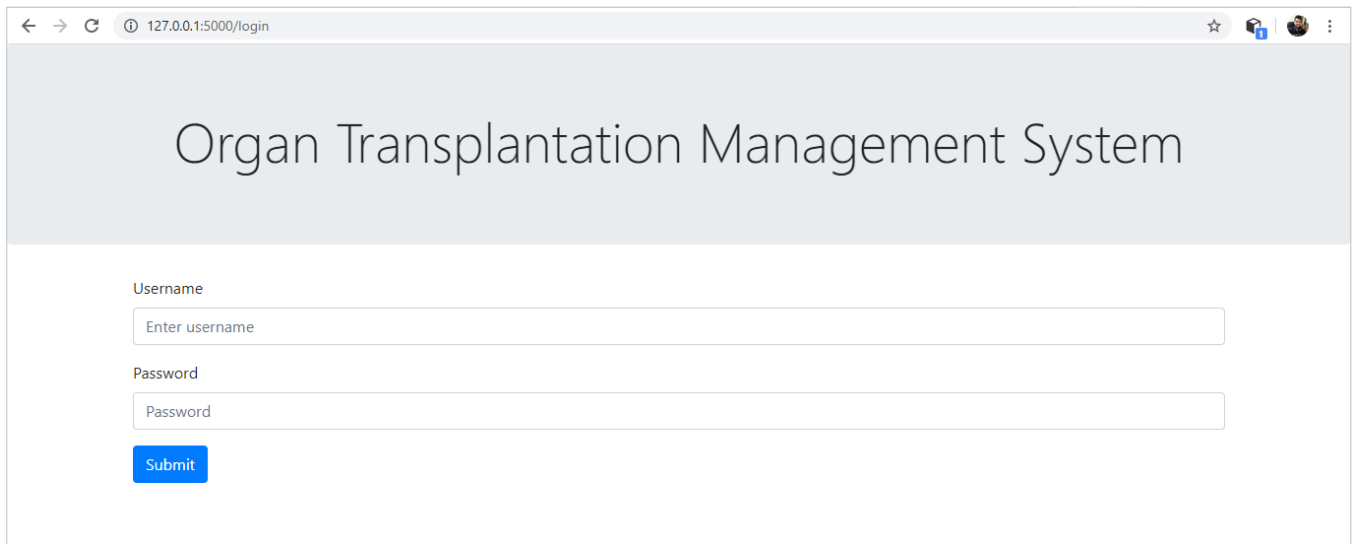
Manager

Username - manager

Password - manager

Screenshots

1) Login Page



A screenshot of a web browser showing the login page of the 'Organ Transplantation Management System'. The browser's address bar displays '127.0.0.1:5000/login'. The page has a light gray header with the system name. Below the header, there are two input fields: 'Username' with a placeholder 'Enter username' and 'Password' with a placeholder 'Password'. A blue 'Submit' button is positioned below the password field.

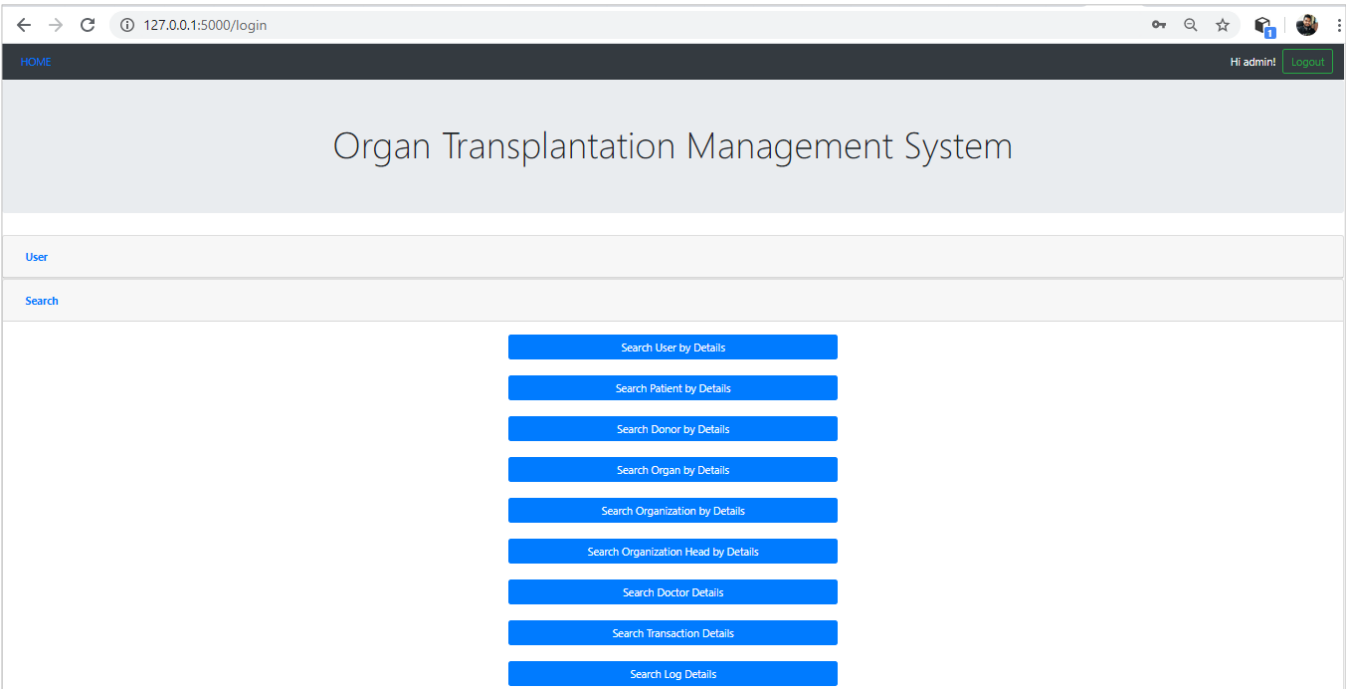
Two access roles admin and manager. Admin are allowed to do all operations. Managers have no access to delete records.

2) Main Page – GUI

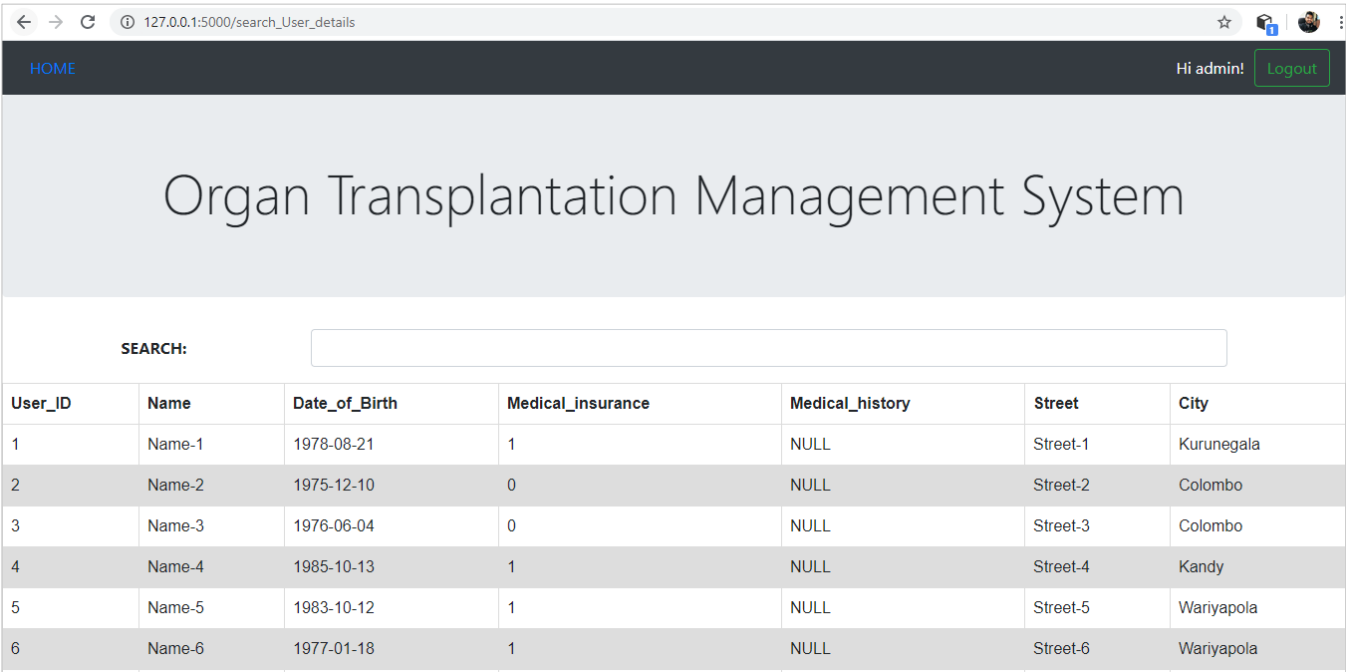


A screenshot of the main page of the 'Organ Transplantation Management System'. The browser's address bar shows '127.0.0.1:5000'. The page features a dark gray top navigation bar with a 'HOME' link on the left and a user greeting 'Hi admin!' with a green 'Logout' button on the right. The main content area has a light gray header with the system name. Below this, there is a vertical list of seven menu items: 'User', 'Search', 'Add', 'Update', 'Remove', and 'Statistics', each on a separate light gray background row.

3) Main Page – Drop Down Menu



4) Searching Option



6) Data visualization using matplotlib in Python. (save data into images and simply visualize them)

