# BLOOD BANK MANAGEMENT SYSTEM

A PROJECT REPORT

*Submitted by*

**Jalaj Gupta [RA2111027010148]**

*Under the Guidance of*

***S. Sindhu***

Assistant Professor, Department of Data Science and Business Systems

*in partial fulfillment of the requirements for the degree of*

**BACHELOR OF TECHNOLOGY**

**in**

**COMPUTER SCIENCE AND ENGINEERING**

**with Specialization in Big Data Analytics**


**DEPARTMENT OF DATA SCIENCE AND BUSINESS SYSTEMS**

**SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**

**KATTANKULATHUR – 603203**

**MAY 2024**

SRM INSTITUTE OF SCIENCE & TECHNOLOGY
COLLEGE OF ENGINEERING & TECHNOLOGY
S.R.M. NAGAR, KATTANKULATHUR – 603203

## BONAFIDE CERTIFICATE

Certified that this project report **BLOOD BANK MANAGEMENT SYSTEM** is the bonafide work of **Jalaj Gupta [RA2111027010148]** of III Year/VI Sem B.Tech (BDA) who carried out the mini project work under my supervision for the course 18CSC303J- Database Management systems in Data Science and Business systems department, school of Computing, SRM Institute of Science and Technology during the academic year 2023-2024 (Even sem)

**Date: 13/05/2024**

**Faculty Incharge**

**S.Sindhu**
Assistant Professor
Data Science and Business Systems
SRMIST -KTR

**HEAD OF DEPARTMENT**

**Dr. M Lakshmi**
Professor
Data Science and Business Systems
SRMIST -KTR

# ACKNOWLEDGEMENT

We express our humble gratitude to **Dr C. Muthamizhchelvan**, Vice-Chancellor, SRM Institute of Science and Technology, for the facilities extended for the project work and his continued support.

We extend our sincere thanks to Dean-CET, SRM Institute of Science and Technology, **Dr T.V.Gopal**, for his invaluable support.
We wish to thank Dr Revathi Venkataraman, Professor & Chairperson, School of Computing, SRM Institute of Science and Technology, for her support throughout the project work.

We are incredibly grateful to our Head of the Department, **Dr. M.Lakshmi** , Professor, Department of Data Science and Business Systems SRM Institute of Science and Technology for her suggestions and encouragement at all the stages of the project work.

Our inexpressible respect and thanks to our guide, **S. Sindhu**, Assistant Professor, Department of Data Science and Business Systems, SRM Institute of Science and Technology, for providing us with an opportunity to pursue our project under her mentorship. She provided us with the freedom and support to explore the research topics of our interest. Her passion for solving problems and making a difference in the world has always been inspiring.

We sincerely thank the Data Science and Business Systems Department, staff and students, SRM Institute of Science and Technology, for their help during our project. Finally, we would like to thank parents, family members, and friends for their unconditional love, constant support, and encouragement.

# ABSTRACT

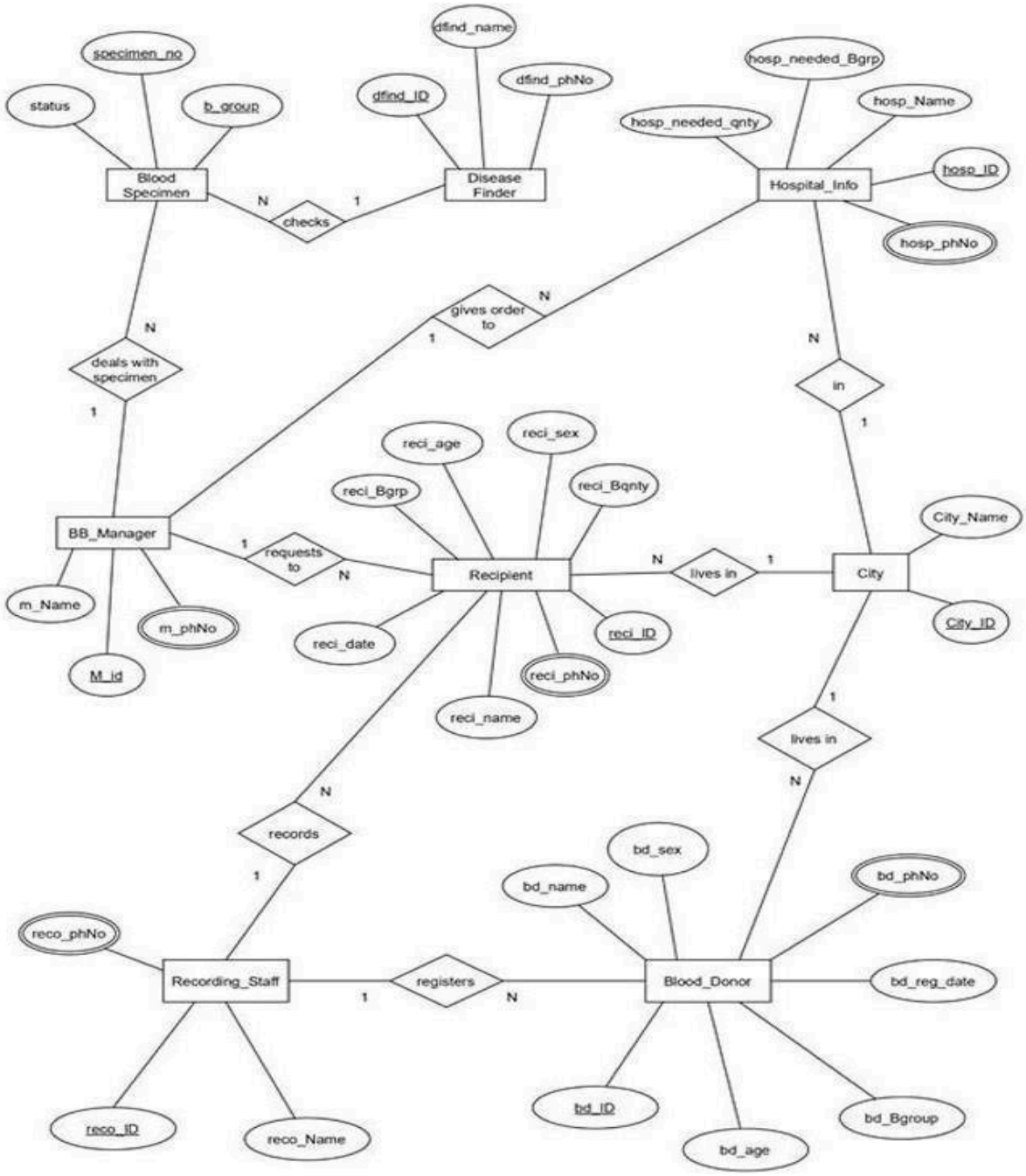
This project aims to develop a Blood Bank Management System. A Blood Bank Management System can be used in any clinic, hospital, labs, or any emergency situation which requires blood units for survival. Our system can be used to find required types of blood in emergency situations from either blood banks or even blood donors. The current system uses a grapevine communication for finding blood in cases of emergency, may it be by a donor or blood bank. The intention of proposing such a system is to abolish the panic caused during an emergency due to unavailability of blood.

# TABLE OF CONTENTS

# INTRODUCTION

Blood banks collect, store and provide collected blood to the patients who are in need of blood. The people who donate blood are called ‘donors’. The banks then group the blood which they receive according to the blood groups. They also make sure that the blood is not contaminated. The main mission of the blood bank is to provide the blood to the hospitals and health care systems which saves the patient’s life. No hospital can maintain the health care system without pure and adequate blood. The major concern each blood bank has is to monitor the quality of the blood and monitor the people who donates the blood, that is ‘donors. But this is a tough job. The existing system will not satisfy the need of maintaining quality blood and keeping track of donors. To overcome all these limitations we introduced a new system called ‘Blood Donation Management System’. The ‘Blood Bank Management System’ allows us to keep track of quality of blood and also keeps track of available blood when requested by the acceptor. The existing systems are Manual systems which are time consuming and not so effective. ‘Blood Bank Management system’ automates the distribution of blood. This database consists of thousands of records of each blood bank. By using this system searching the available blood becomes easy and saves a lot of time than the manual system. It will hoard, operate, recover and analyze information concerned with the administrative and inventory management within a blood bank. This system is developed in a manner that it is manageable, time effective, cost effective, flexible and much manpower is not required.

# ER DIAGRAM


dfind_name
specimen_no
dfind_phNo
hosp_needed_Bgrp
status
b_group
dfind_ID
hosp_Name
hosp_needed_qnty
Blood Specimen
N
checks
1
Disease Finder
Hospital_Info
hosp_ID
hosp_phNo
gives order to
N
1
N
deals with specimen
N
in
1
1
reci_age
reci_sex
reci_Bgrp
reci_Bqnty
City_Name
BB_Manager
1
requests to
N
Recipient
N
lives in
1
City
m_Name
m_phNo
reci_date
reci_ID
City_ID
M_id
reci_phNo
reci_name
1
lives in
N
N
records
bd_sex
1
bd_name
bd_phNo
reco_phNo
Recording_Staff
1
registers
N
Blood_Donor
bd_reg_date
reco_ID
reco_Name
bd_ID
bd_Bgroup
bd_age

### 1.1 Entities and Attributes:

The project encompasses entities such as City, Disease, Finder, Blood Recording, Staff, Hospital, and Name. Each entity is associated with specific attributes that capture relevant information, ensuring comprehensive data coverage and facilitating efficient data management. For example, the City entity may have attributes such as name, population, and geographical location, while the Disease entity may include attributes like name, symptoms, and treatment options.

### 1.2 Relationships:

Through relationships depicted in the project, connections between entities are established, illustrating how data interacts within the system. For instance, Staff members work at specific Hospitals, and Hospitals may have multiple Blood Recordings associated with them. Additionally, Finders may be associated with certain Cities for disease tracking purposes. These relationships enable effective data flow and management within the system, facilitating tasks such as disease monitoring, blood recording, and staffing allocation.

The ER diagram provides a clear and comprehensive overview of the Tourism Management System's data model. It serves as a vital tool for system designers, developers, and stakeholders, guiding the implementation of the database architecture and facilitating efficient data management processes. By emphasizing clarity, organization, and normalization, the ER diagram contributes to the creation of a robust and user-friendly platform for managing tourism activities effectively.

## INFORMATION OF ENTITIES

In total we have eight entities and information of each entity is mentioned below:-

**Blood_Donor:** (Attributes – bd_ID, bd_name, bd_sex, bd_age, bd_Bgroup, bd_reg_date, bd_phNo)
The donor is the person who donates blood, on donation a donor id (bd_ID) is generated and used as the primary key to identify the donor information. Other than that name, age , sex , blood group, phone number and registration dates will be stored in the database under Blood_Donor entity.

**Recipient:** (Attributes – reci_ID, reci_name, reci_age, reci_Bgrp,)
The Recipient is the person who receives blood from a blood bank, when blood is given to a recipient a recipient ID (reci_ID) is generated and used as the primary key for the recipient entity to identify blood recipients information. Along with its name ,age, sex, blood group (needed), blood quantity(needed) , phone number, and registration dates are also stored in the database under the recipient entity.

**BB_Manager:** (Attributes – m_ID, m_Name, m_phNo)
The blood bank manager is the person who takes care of the available blood samples in the blood bank, he is also responsible for handling blood requests from recipients and hospitals. Blood manager has a unique identification number (m_ID) used as primary key along with name and phone number of blood bank manager will be stored in the database under BB_Manager entity.

**Recording_Staff :** (Attributes – reco_ID, reco_Name, reco_phNo)
The recording staff is a person who registers the blood donor and recipients and the Recording_Staff entity has reco_ID which is the primary key along with recorder's name and recorder's phone number will also be stored in the database under Recording_Staff entity.

**BloodSpecimen :** (Attributes – specimen_number, b_group , status)
In the database, under the Blood Specimen entity we will store the information of blood samples which are available in the blood bank. In this entity

specimen_number and b_group together will be the primary key along with a status attribute which will show if the blood is contaminated or not.

**DiseaseFinder :** (Attributes - dfind_ID, dfind_name, dfind_PhNo)
In the database , under the DiseaseFinder entity we will store the information of the doctor who checks the blood for any kind of contamination. To store that information we have a unique identification number (dfind_ID) as the primary key. Along with the name and phone number of the doctor will also be stored under the same entity.

**Hospital_Info :** (Attributes – hosp_ID, hosp_name, hosp_needed_Bgrp, hosp_needed_Bqnty)
In the database, under the Hospital_Info entity we will store the information of hospitals. In this hosp_ID and hosp_needed_Bgrp together makes the primary key. We will store the hospital name and the blood quantity required at the hospital.

**City:** (Attributes- city_ID, city_name)
This entity will store the information of cities where donors, recipients and hospitals are present. A unique identification number (City_ID) will be used as the primary key to identify the information about the city. Along with ID city names will also be stored under this entity.

# RELATIONSHIP BETWEEN ENTITIES

**City and Hospital_Info:**

Relationship = “in”

Type of relation = 1 to many

Explanation = A city can have many hospitals in it. One hospital will belong in one city.

**City and Blood_Donor:** Relationship = “lives in” Type of relation = 1 to many

Explanation = In a city, many donors can live. One donor will belong to one city.

**City and Recipient:** Relationship = “lives in” Type of relation = 1 to many

Explanation = In a city, many recipients can live. One recipient will belong to one city.

**Recording_Staff and Donor:** Relationship = “registers” Type of relation = 1 to many

Explanation = One recording staff can register many donors. One donor will register with one recording officer.

**Recording_Staff and Recipient:**

Relationship = “records” Type of relation = 1 to many

Explanation = One recording staff can record many recipients. One recipient will be recorded by one recording officer.

**Hospital_Info and BB_Manager:** Relationship = “gives order to” Type of relation = 1 to many

Explanation = One Blood bank manager can handle and process requests from many hospitals. One hospital will place a request on the blood bank manager.

**BB_Manager and Blood Specimen:** Relationship = “dealers with specimen” Type of relation = 1 to many

Explanation = One Blood bank manager can manage many blood specimens and one specimen will be managed by one manager.

**Recipient and BB_Manager:** Relationship = “requests to” Type of relation = 1 to many

Explanation = One recipient can request blood to one manager and one manager can handle requests from many recipients.

# Proposed work details

## CREATION OF DATABASE TABLES

The proposed work consists of 7 tables that are interconnected. The team members work on tables and keep updating them by implementing queries.

The structure and function of each table are described below:

1. **Donor Details table:**

This table stores comprehensive information about blood donors. It includes fields such as Donor ID, Donor Name, Contact Number, Email Address, Age, Gender, Blood Type, and Address. The Donor ID serves as the primary key for the table.

- Donor ID: int, auto-increment, NOT NULL
- Donor Name: varchar(50), NOT NULL
- Donor Number: varchar(10), NOT NULL
- Donor Mail: varchar(50)
- Donor Age: int(60), NOT NULL
- Donor Gender: varchar(10), NOT NULL
- Donor Blood: varchar(10), NOT NULL
- Donor Address: varchar(100), NOT NULL

**TABLE STRUCTURE:**

| Field | Type | Null | Key | Default | Extra |
|---|---|---|---|---|---|
| admin_id | int(10) | NO | PRI | NULL | auto_increment |
| admin_name | varchar(50) | NO | | NULL | |
| admin_username | varchar(50) | NO | UNI | NULL | |
| admin_password | varchar(50) | NO | | NULL | |

**2. Admin Info table:**

This table stores information about administrators. It includes fields such as Admin ID, Admin Name, Username, and Password. The Admin ID serves as the primary key, and the Username field has a unique constraint.

- Admin ID: int(10), NOT NULL, UNIQUE, auto-increment
- Admin Name: varchar(50), NOT NULL
- Admin Username: varchar(50), NOT NULL, UNIQUE
- Admin Password: varchar(50), NOT NULL

| Field | Type | Null | Key | Default | Extra |
|---|---|---|---|---|---|
| blood_id | int | NO | PRI | NULL | auto_increment |
| blood_group | varchar(10) | NO | | NULL | |

**3. Blood table:**

This table stores information about different blood groups. It includes fields such as Blood ID and Blood Group. The Blood ID serves as the primary key for the table.

- Blood ID: int, auto-increment, NOT NULL
- Blood Group: varchar(10), NOT NULL

Table Structure

| Field | Type | Null | Key | Default | Extra |
|---|---|---|---|---|---|
| contact_id | int | NO | PRI | NULL | auto_increment |
| contact_address | varchar(100) | NO | | NULL | |
| contact_mail | varchar(50) | NO | | NULL | |
| contact_phone | varchar(100) | NO | | NULL | |

**4. Pages table:**

This table stores information about website pages. It includes fields such as Page ID, Page Name, Page Type, and Page Data. The Page ID serves as the primary key, and the Page Type field has a unique constraint.

- Page ID: int, NOT NULL, auto-increment, UNIQUE
- Page Name: varchar(255), NOT NULL
- Page Type: varchar(50), NOT NULL, UNIQUE
- Page Data: longtext, NOT NULL

Table Structure :

| Field | Type | Null | Key | Default | Extra |
|---|---|---|---|---|---|
| query_id | int | NO | PRI | NULL | auto_increment |
| query_name | varchar(100) | NO | | NULL | |
| query_mail | varchar(120) | NO | | NULL | |
| query_number | char(11) | NO | | NULL | |
| query_message | longtext | NO | | NULL | |
| query_date | timestamp | NO | | NULL | |
| query_status | int | YES | | NULL | |

**5. Contact Info table:**

This table stores contact information for the blood bank website. It includes fields such as Contact ID, Address, Email, and Phone Number. The Contact ID serves as the primary key for the table.

- Contact ID: int, auto-increment, NOT NULL
- Contact Address: varchar(100), NOT NULL
- Contact Mail: varchar(50), NOT NULL
- Contact Phone: varchar(100), NOT NULL

Table Structure:

| Field | Type | Null | Key | Default | Extra |
|---|---|---|---|---|---|
| donor_id | int | NO | PRI | NULL | auto_increment |
| donor_name | varchar(50) | NO | | NULL | |
| donor_number | varchar(10) | NO | | NULL | |
| donor_mail | varchar(50) | YES | | NULL | |
| donor_age | int | NO | | NULL | |
| donor_gender | varchar(10) | NO | | NULL | |
| donor_blood | varchar(10) | NO | | NULL | |
| donor_address | varchar(100) | NO | | NULL | |

### 6. Contact Query table:

This table stores information about queries submitted by users. It includes fields such as Query ID, Name, Email, Contact Number, Message, Date of Query, and Status. The Query ID serves as the primary key for the table.

- Query ID: int, auto-increment, NOT NULL
- Query Name: varchar(100), NOT NULL
- Query Mail: varchar(120), NOT NULL
- Query Number: char(11), NOT NULL
- Query Message: longtext, NOT NULL
- Query Date: timestamp, NOT NULL, DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
- Query Status: int(11), DEFAULT NULL

Table Structure:

```
+---------------+-------------+------+-----+---------+----------------+
| Field         | Type        | Null | Key | Default | Extra          |
+---------------+-------------+------+-----+---------+----------------+
| page_id       | int         | NO   | PRI | NULL    | auto_increment |
| page_name     | varchar(255)| NO   |     | NULL    |                |
| page_type     | varchar(50) | NO   | UNI | NULL    |                |
| page_data     | longtext    | NO   |     | NULL    |                |
+---------------+-------------+------+-----+---------+----------------+
```

### 7. Query Stat table:

This table stores information about the status of queries. It includes fields such as ID and Query Type. The ID serves as the primary key for the table.

- ID: int, NOT NULL, UNIQUE
- Query Type: varchar(45), NOT NULL

Table Structure:

```
+---------------+-------------+------+-----+---------+----------------+
| Field         | Type        | Null | Key | Default | Extra          |
+---------------+-------------+------+-----+---------+----------------+
| id            | int         | NO   | PRI | NULL    |                |
| query_type    | varchar(45) | NO   |     | NULL    |                |
+---------------+-------------+------+-----+---------+----------------+
```

## COMPLEX QUERIES

1. **Create a View of recipients and donors names having the same blood group registered on the same date and the name of the recording staff name**.

CREATE VIEW Blood_Recipient_SameBGrp AS select Blood_Donor.bd_name,Recipient.reci_name,reco_Name from Recording_Staff inner join Blood_Donor on Recording_Staff.reco_ID = Blood_Donor.reco_ID inner join Recipient on Recording_Staff.reco_ID = Recipient.reco_ID where Blood_Donor.bd_Bgroup = Recipient.reci_Brgp and Blood_Donor.bd_reg_date = Recipient.reci_reg_date; select* from Blood_Recipient_SameBGrp;

OUTPUT:

| bd_name | reci_name | reco_Name |
|---|---|---|
| shweta | Peter | Walcot |

**2. Show the blood specimen verified by disease finder shivam which are pure (status=1).**

Select specimen_number,b_group from BloodSpecimen,DiseaseFinder WHERE BloodSpecimen.dfind_ID= DiseaseFinder.dfind_ID AND dfind_name='shivam' AND status=1;

**Output:**

| specimen_number | b_group |
|---|---|
| 1012 | O- |
| 1013 | B- |

3. **Show the pure blood specimen handled by BB_Manager who also handles a recipient needing the same blood group along with the details of the BB_Manager and Recipient**.

select
BB_Manager.M_id,mName,Recipient.reci_name,
Recipient.reci_Brgp,BloodSpecimen.b_group
from
BB_Manager,Recipient,BloodSpecimen where Recipient.M_id = BloodSpecimen.M_id and Recipient.reci_Brgp = BloodSpecimen.b_group and Recipient.M_id = BB_Manager.M_id and status = 1; Output:

| M_id | mName | reci_name | reci_Brgp | b_group |
|---|---|---|---|---|
| 101 | shivank | Peter | B+ | B+ |
| 102 | shwetanshu | akhil | AB+ | AB+ |

**4. Show the donors having the same blood groups required by the recipient staying in the same city along with recipient details.**

Select bd_ID,bd_name,reci_ID,reci_name FROM Blood_Donor,Recipient
WHERE bd_Bgroup=reci_Brgp AND Blood_Donor.City_ID= Recipient.City_ID;
**Output:**

| bd_ID | bd_name | reci_ID | reci_name |
|---|---|---|---|
| 150013 | Shivank | 10003 | akhil |
| 150013 | Shivank | 10010 | Marsh |
| 150014 | shweta | 10004 | Parker |
| 150014 | shweta | 10008 | Swathi |
| 150017 | Mike | 10005 | jojo |

**5. Display the information of Hospital_Info_1 handled by BB_Manager whose ID is 103:**

Select hosp_ID,hosp_name , City_ID, HOspital_Info_1.M_id from Hospital_Info_1,BB_Manager where BB_Manager.M_id=Hospital_Info_1.M_id and BB_Manager.M_id=103;

**OUTPUT:-**

| hosp_ID | hosp_name | City_ID | M_id |
|---|---|---|---|
| 2 | CleavelandClinic | 1200 | 103 |
| 3 | NYU | 1300 | 103 |
| 5 | Charlton | 1800 | 103 |

## ADDING A CONSTRAINT

**6.Add Primary key constraint to the column M_ID of bb_manager table**

```
1    CREATE TABLE BB_Manager
2    ( M_id int NOT NULL PRIMARY KEY,
3     mName varchar(100) NOT NULL,
4     m_phNo bigint
5    );
```

**OUTPUT:-**

```
mysql>  select* from bb_manager;
+------+------------+------------+
| M_id | mName      | m_phNo     |
+------+------------+------------+
|  101 | shivank    | 9693959671 |
|  102 | shwetanshu | 9693959672 |
+------+------------+------------+
2 rows in set (0.01 sec)
```

**7. Add Foreign key constraint to the column M_id of bb_manager table references bb_manager table.**

```
1    CREATE TABLE BloodSpecimen
2    ( specimen_number int NOT NULL,
3     b_group varchar(10) NOT NULL,
4     status int,
5     dfind_ID int NOT NULL,
6     M_id int NOT NULL,
7    primary key (specimen_number,b_group),
8    FOREIGN KEY(M_id) REFERENCES Manager(M_id),
9    FOREIGN KEY(dfind_ID) REFERENCES DiseaseFinder(dfind_ID)
10   );
```

**OUTPUT:-**

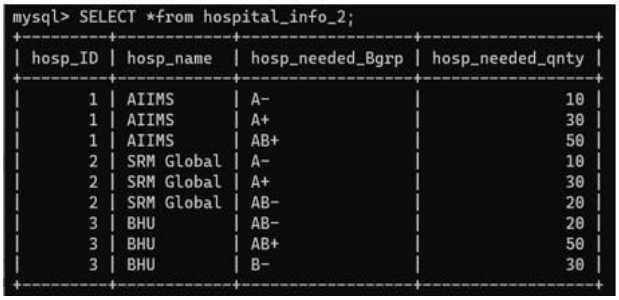
```
mysql>  select *from  hospital_info_2;
+---------+------------+------------------+------------------+
| hosp_ID | hosp_name  | hosp_needed_Bgrp | hosp_needed_qnty |
+---------+------------+------------------+------------------+
|       1 | AIIMS      | A+               |               40 |
|       1 | AIIMS      | AB+              |               60 |
|       2 | SRM Global | A+               |               40 |
|       2 | SRM Global | AB-              |               30 |
|       3 | BHU        | AB-              |               30 |
|       3 | BHU        | AB+              |               60 |
|       3 | BHU        | B-               |               40 |
+---------+------------+------------------+------------------+
```

## BASIC SELECT STATEMENTS

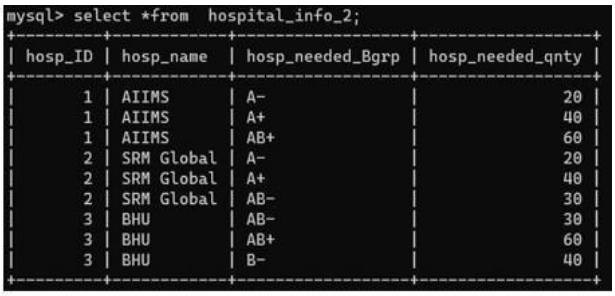
**8. Update all the records of *hospital_info_2* table by increasing 10 units of blood quantity.**

```
SELECT * FROM hospital_info_2 LIMIT 100;
UPDATE Hospital_Info_2
SET hosp_needed_qnty = hosp_needed_qnty + 10;
```

**OUTPUT:-**



```
mysql> SELECT *from hospital_info_2;
+---------+-----------+-----------------+------------------+
| hosp_ID | hosp_name | hosp_needed_Bgrp | hosp_needed_qnty |
+---------+-----------+-----------------+------------------+
|       1 | AIIMS     | A-              |               10 |
|       1 | AIIMS     | A+              |               30 |
|       1 | AIIMS     | AB+             |               50 |
|       2 | SRM Global | A-             |               10 |
|       2 | SRM Global | A+             |               30 |
|       2 | SRM Global | AB-            |               20 |
|       3 | BHU       | AB-             |               20 |
|       3 | BHU       | AB+             |               50 |
|       3 | BHU       | B-              |               30 |
+---------+-----------+-----------------+------------------+
```



```
mysql> select *from  hospital_info_2;
+---------+-----------+-----------------+------------------+
| hosp_ID | hosp_name | hosp_needed_Bgrp | hosp_needed_qnty |
+---------+-----------+-----------------+------------------+
|       1 | AIIMS     | A-              |               20 |
|       1 | AIIMS     | A+              |               40 |
|       1 | AIIMS     | AB+             |               60 |
|       2 | SRM Global | A-             |               20 |
|       2 | SRM Global | A+             |               40 |
|       2 | SRM Global | AB-            |               30 |
|       3 | BHU       | AB-             |               30 |
|       3 | BHU       | AB+             |               60 |
|       3 | BHU       | B-              |               40 |
+---------+-----------+-----------------+------------------+
```

**9. Delete the records from *hospital_info_2* table table where the salary is less than 30.**

```
SELECT * FROM hospital_info_2 LIMIT 100;
DELETE FROM Hospital_Info_2
WHERE hosp_needed_qnty < 30;
```

**OUTPUT:-**

```
mysql>  select *from  hospital_info_2;
+---------+-----------+-----------------+------------------+
| hosp_ID | hosp_name | hosp_needed_Bgrp | hosp_needed_qnty |
+---------+-----------+-----------------+------------------+
|       1 | AIIMS     | A+              |               40 |
|       1 | AIIMS     | AB+             |               60 |
|       2 | SRM Global | A+             |               40 |
|       2 | SRM Global | AB-            |               30 |
|       3 | BHU       | AB-             |               30 |
|       3 | BHU       | AB+             |               60 |
|       3 | BHU       | B-              |               40 |
+---------+-----------+-----------------+------------------+
```

**10. Display each name of the hospital as “AB+” blood group**

```
SELECT * FROM hospital_info_2 LIMIT 100;
SELECT DISTINCT hosp_name
FROM hospital_info_2
WHERE hosp_needed_Bgrp = 'AB+';
```

**OUTPUT:-**

```
+-----------+
| hosp_name |
+-----------+
| AIIMS     |
| BHU       |
+-----------+
```

**IMPLEMENTATION OF TRIGGERS**

Here, The trigger will backup the data that is deleted, Inserted and updated inside the Bb_manager table then the data is backup in bb_manager table

```
CREATE DEFINER=`root`@`localhost` TRIGGER `bb_backup` AFTER INSERT
ON
`bb_manager` FOR EACH ROW BEGIN
-- Insert the deleted row into the backup table INSERT INTO bb_backup (mName, m_phNo) VALUES (NEW.mName, NEW.m_phNo);
END
```

**IMPLEMENTATION OF CURSORS**

Here , We created cursors for getting all the data whose age is greater than 20 from Blood donor

```
CREATE DEFINER=root@localhost PROCEDURE donors() BEGIN
-- Declare variables
DECLARE done BOOLEAN DEFAULT FALSE;
DECLARE donor_name VARCHAR(100);
-- Declare cursor for the query
   DECLARE bd_cursor CURSOR FOR SELECT bd_name FROM blood_donor WHERE bd_age > 20;
-- Declare handler for when no more rows are available
DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
-- Open the cursor OPEN bd_cursor
-- Create temporary table to hold the data
   CREATE TEMPORARY TABLE temp_blood_donors (donor_name VARCHAR(100));
-- Loop through the result set blood_donor_loop: LOOP
FETCH bd_cursor INTO donor_name; IF done THEN
   LEAVE blood_donor_loop; END IF;
-- Insert each row into the temporary table
```

```
    INSERT INTO temp_blood_donors (donor_name) VALUES (donor_name);
END LOOP blood_donor_loop;
-- Close the cursor CLOSE bd_cursor;
-- Print the data from the temporary table SELECT * FROM temp_blood_donors;
-- Drop the temporary table
    DROP TEMPORARY TABLE IF EXISTS temp_blood_donors; END
cursor
```

# Analyzing the pitfalls, identifying the dependencies, and Applying Normalizations

Normalization is a database design technique used to organize data in a relational database efficiently, reducing redundancy and dependency. The normalization process involves dividing large tables into smaller, related tables and defining relationships between them. This helps to minimize data redundancy and ensures data integrity. There are several normal forms, each with its own set of rules. The most commonly used normal forms are:

**First Normal Form (1NF):**
In 1NF, each column in a table must contain atomic (indivisible) values, and each row must be unique. It eliminates repeating groups and ensures that each field contains only one value. Example: Splitting a column containing multiple values into separate columns or rows.

**Second Normal Form (2NF):**
In 2NF, a table must be in 1NF, and all non-key attributes must be fully functional dependent on the primary key. This means that no non-key attribute is dependent on only a portion of the primary key. Example: Breaking a table into multiple tables, with each table containing a single theme, and using foreign keys to establish relationships between them.

**Third Normal Form (3NF):**
In 3NF, a table must be in 2NF, and no non-key attribute should depend on another non-key attribute (transitive dependency). Each non-key attribute must depend only on the primary key. Example: Removing attributes that depend on other non-key attributes rather than the primary key.

**Boyce-Codd Normal Form (BCNF):**
BCNF is a stricter form of 3NF, where every determinant is a candidate key. It ensures that there are no non-trivial dependencies between attributes that are not candidate keys. Example: Ensuring that each determinant in a table is a candidate key, or decomposing the table further until BCNF is achieved.

**Fourth Normal Form (4 NF):**
4NF deals with multivalued dependencies, ensuring that there are no non-trivial multivalued dependencies between attributes. Example: Decomposing tables to remove multivalued dependencies.

**Fifth Normal Form (5 NF):**
5NF deals with join dependencies, ensuring that a database is free from certain types of redundancy related to join operations. Example: Ensuring that all join dependencies are satisfied, and no redundant data exists as a result of join operations.

## NORMALIZATION OF BLOOD BANK DATABASE

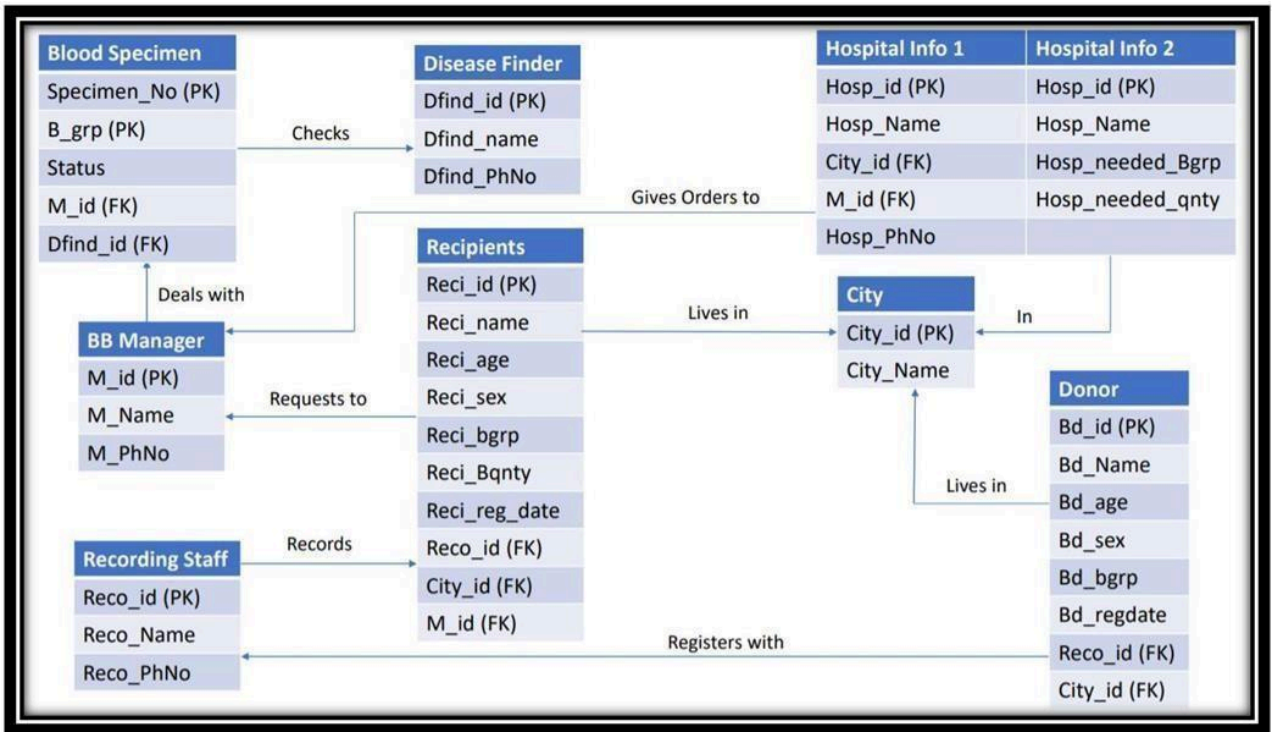
Hospital_Info ( hosp_Id, hosp_Name, hosp_phNo, hosp_needed_Bgrp, hosp_needed_qty, city_id, m_id)
{hosp_Id}= > {hosp_Name, hosp_phNo, city_id, m_id}

{hosp_Id, hosp_needed_Bgrp } = > hosp_needed_qty (functional dependency exists)

The table is in its first normal form. Since every non-primary key attribute is not fully functionally dependent on the primary key of the table, this table is not in second normal form. Hence we have to split the table. Hospital_1 (hosp_Id, hosp_phNo, hosp_Name, city_id, m_id)

Hospital_2 (hosp_Id, hosp_needed_Bgrp, hosp_needed_qty) Now it is in second normal form. The table is in its third normal form.

## RELATION SCHEMA AFTER NORMALIZATION



# Implementation of Concurrency Control and Recovery Mechanisms

Concurrency Control is the management procedure that is required for controlling concurrent execution of the operations that take place on a database. In a multi-user system, multiple users can access and use the same database at one time, which is known as the concurrent execution of the database. It means that the same database is executed simultaneously on a multi-user system by different users.

While working on the database transactions, there occurs the requirement of using the database by multiple users for performing different operations, and in that case, concurrent execution of the database is performed.The thing is that the simultaneous execution that is performed should be done in an interleaved manner, and no operation should affect the other executing operations, thus maintaining the consistency of the database. Thus, on making the concurrent execution of the transaction operations, there occur several challenging problems that need to be solved.

**Problems with Concurrent Execution:**

In a database transaction, the two main operations are READ and WRITE operations. So, there is a need to manage these two operations in the concurrent execution of the transactions as if these operations are not performed in an interleaved manner, the data may become inconsistent. So, the following problems occur with the Concurrent Execution of the operations:

**Lost Update Problems (W - W Conflict):**

The problem occurs when two different database transactions perform the read/write operations on the same database items in an interleaved manner (i.e., concurrent execution) that makes the values of the items incorrect hence making the database inconsistent.

**Dirty Read Problems (W-R Conflict)**:

The dirty read problem occurs when one transaction updates an item of the database, and somehow the transaction fails, and before the data gets rolled back, the updated database item is accessed by another transaction. There comes the Read-Write Conflict between both transactions.

**Unrepeatable Read Problem (W-R Conflict)**

Also known as Inconsistent Retrievals Problem that occurs when in a transaction, two different values are read for the same database item.

**Recovery Mechanisms:**

**Write-Ahead Logging (WAL):**

A technique where changes are first recorded in the log before being written to the database itself. This ensures that the log reflects the latest state of the database, allowing for crash recovery by replaying the log.

**Redo and Undo Logging:**

Redo logging involves reapplying the changes recorded in the log to bring the database up to the point of failure, while undo logging involves reversing the effects of incomplete transactions to restore the database to a consistent state.

**Savepoints:**

Intermediate points within a transaction where the state of the database is saved. If a failure occurs, the transaction can be rolled back to the last savepoint to minimize the amount of work that needs to be undone.

such tasks.

# CODE

```
<?php session_start();

isset($_POST['login'])){ include('../includes/connection.php');

    $query = "select id,email,password,name from patients where email =
'$_POST[email]' AND password = '$_POST[password]'";

$query_run = mysqli_query($connection,$query); if(mysqli_num_rows($query_run)){

$_SESSION['email'] = $_POST['email']; while($row =

mysqli_fetch_assoc($query_run)){

$_SESSION['name'] = $row['name'];

$_SESSION['uid'] = $row['id'];

}

cho "<script type='text/javascript'> window.location.href =

'patient_dashboard.php';

</script>";

}

else{

echo "<script type='text/javascript'>

alert('Please enter correct email and password.'); window.location.href =

'login.php';

</script>";

}

}

?>

<!DOCTYPE html>
```

```
<html>
<body>
<div class="row">
<div class="col-md-6 m-auto">
<br><center><h4><u>List of all Donors</u></h4><br></center>
<table class="table">
<thread>
<th>S.No</th>
<th>Donor ID</th>
<th>Donor Name</th>
<th>Donor Email</th>
<th>Mobile No</th>
<th>Action</th>
</thead>
<?php
session_start(); include('../includes/connection.php');
$query = "select * from donors";
$query_run = mysqli_query($connection,$query);
$sno = 1;
while($row = mysqli_fetch_assoc($query_run)){
?>
<tr>
<td><?php echo $sno; ?></td>
<td><?php echo $row['id']; ?></td>
<td><?php echo $row['name']; ?></td>
<td><?php echo $row['email']; ?></td>
<td><?php echo $row['mobile']; ?></td>
                            <td><a class="btn btn-sm btn-success"
href="edit_donor.php?did=<?php echo $row['id']; ?>">Edit</a> <a class="btn btn-sm
btn-danger" href="delete_donor.php?did=<?php echo $row['id']; ?>">Delete</a></td>
</tr>
<?php
$sno+;
}
?>
</table>
</div>
</div>
</body>
</html>
<html lang="en">

<head>

<meta charset="UTF-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```php
<?php session_start();
if isset($_POST['login'])){ include('../includes/connection.php');
    $query = "select id,email,password,name from patients where email =
'$_POST[email]' AND password = '$_POST[password]'";
$query_run = mysqli_query($connection,$query); if(mysqli_num_rows($query_run)){
$_SESSION['email'] = $_POST['email']; while($row =
mysqli_fetch_assoc($query_run)){
$_SESSION['name'] = $row['name'];
$_SESSION['uid'] = $row['id'];
}
echo "<script type='text/javascript'> window.location.href =
'patient_dashboard.php';
</script>";
}
else{
echo "<script type='text/javascript'>
alert('Please enter correct email and password.'); window.location.href =
'login.php';
</script>";
}
}
?>
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Patient Login</title>
<!-- Bootstrap files -->
<link rel="stylesheet" href="../bootstrap/css//bootstrap.min.css">
<script src="../bootstrap/js/bootstrap.min.js"></script>
<!-- External CSS file -->
<link rel="stylesheet" href="../css/styles.css">
</head>
<body>
<nav class="navbar navbar-expand-lg navbar-dark bg-danger">
<a class="navbar-brand" href="index.php">Blood Bank Management System</a>
        <button class="navbar-toggler" type="button" data-toggle="collapse" data-
target="#navbarNav" aria-controls="navbarNav" aria-expanded="false" aria-
label="Toggle navigation">
<span class="navbar-toggler-icon"></span>
</button>
```

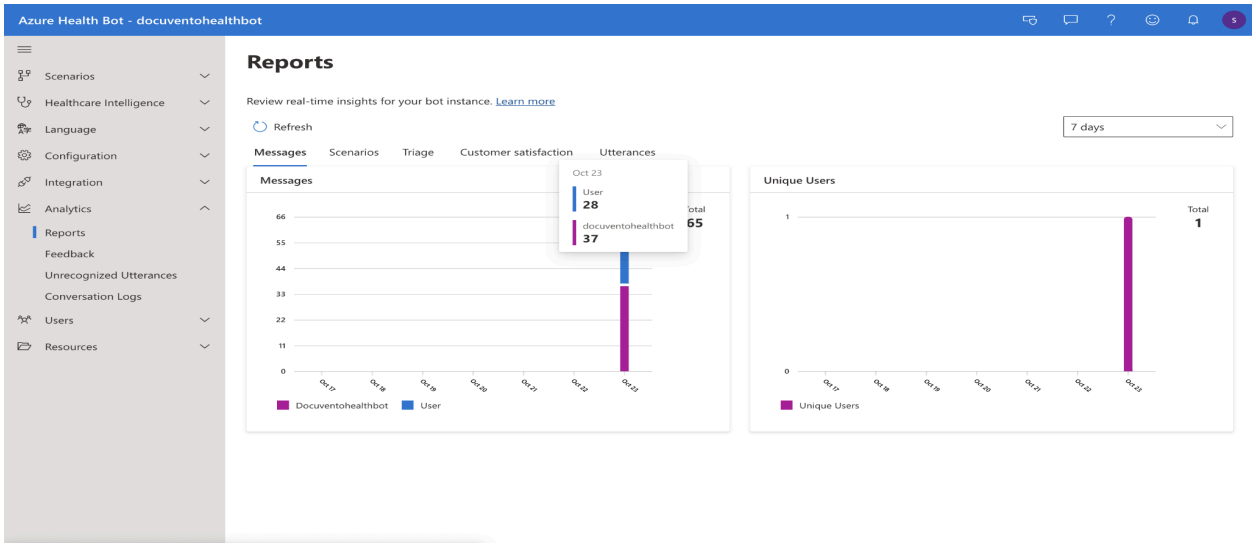
```php
<?php session_start();
if(isset($_SESSION['email'])){ include('../includes/connection.php');
$query = "select * from requests where patient_id = $_SESSION[uid]";
$query_run = mysqli_query($connection,$query);
$total_request = mysqli_num_rows($query_run);

$query = "select * from requests where patient_id = $_SESSION[uid] AND status =
1";
$query_run = mysqli_query($connection,$query);
$request_acc = mysqli_num_rows($query_run);

$query = "select * from requests where patient_id = $_SESSION[uid] AND status =
2";
$query_run = mysqli_query($connection,$query);
$request_rej = mysqli_num_rows($query_run);

$query = "select * from requests where patient_id = $_SESSION[uid] AND status =
1";
$query_run = mysqli_query($connection,$query);
$blood_requested = 0;
while($row = mysqli_fetch_assoc($query_run)){
$blood_requested = $blood_requested + number_format($row['no_units']);
}
if(isset($_POST['request_blood'])){
$query = "insert into requests
values(null,$_SESSION[uid],'$_POST[units]','$_POST[bgroup]','$_POST[reason]',0)"
;
$query_result = mysqli_query($connection,$query); if($query_result){
echo "<script type='text/javascript'> alert('Request submitted
successfully...');
window.location.href = 'patient_dashboard.php';
</script>";
}
else{
echo "<script type='text/javascript'> alert('Error...Plz try again.');
window.location.href = 'patient_dashboard.php';
</script>";
}
}
?>
<!DOCTYPE html>
<html lang="en">
<head>
```

```php
<?php session_start();
if(isset($_SESSION['email'])){ include('../includes/connection.php');
$query = "select * from requests where patient_id = $_SESSION[uid]";
$query_run = mysqli_query($connection,$query);
$total_request = mysqli_num_rows($query_run);

$query = "select * from requests where patient_id = $_SESSION[uid] AND status =
1";
$query_run = mysqli_query($connection,$query);
$request_acc = mysqli_num_rows($query_run);

$query = "select * from requests where patient_id = $_SESSION[uid] AND status =
2";
$query_run = mysqli_query($connection,$query);
$request_rej = mysqli_num_rows($query_run);

$query = "select * from requests where patient_id = $_SESSION[uid] AND status =
1";
$query_run = mysqli_query($connection,$query);
$blood_requested = 0;
while($row = mysqli_fetch_assoc($query_run)){
$blood_requested = $blood_requested + number_format($row['no_units']);
}
if(isset($_POST['request_blood'])){
$query = "insert into requests
values(null,$_SESSION[uid],'$_POST[units]','$_POST[bgroup]','$_POST[reason]',0)"
;
$query_result = mysqli_query($connection,$query); if($query_result){
echo "<script type='text/javascript'> alert('Request submitted
successfully...');
window.location.href = 'patient_dashboard.php';
</script>";
}
else{
echo "<script type='text/javascript'> alert('Error...Plz try again.');
window.location.href = 'patient_dashboard.php';
</script>";
}
}
?>
<!DOCTYPE html>
<html lang="en">
<head>
```

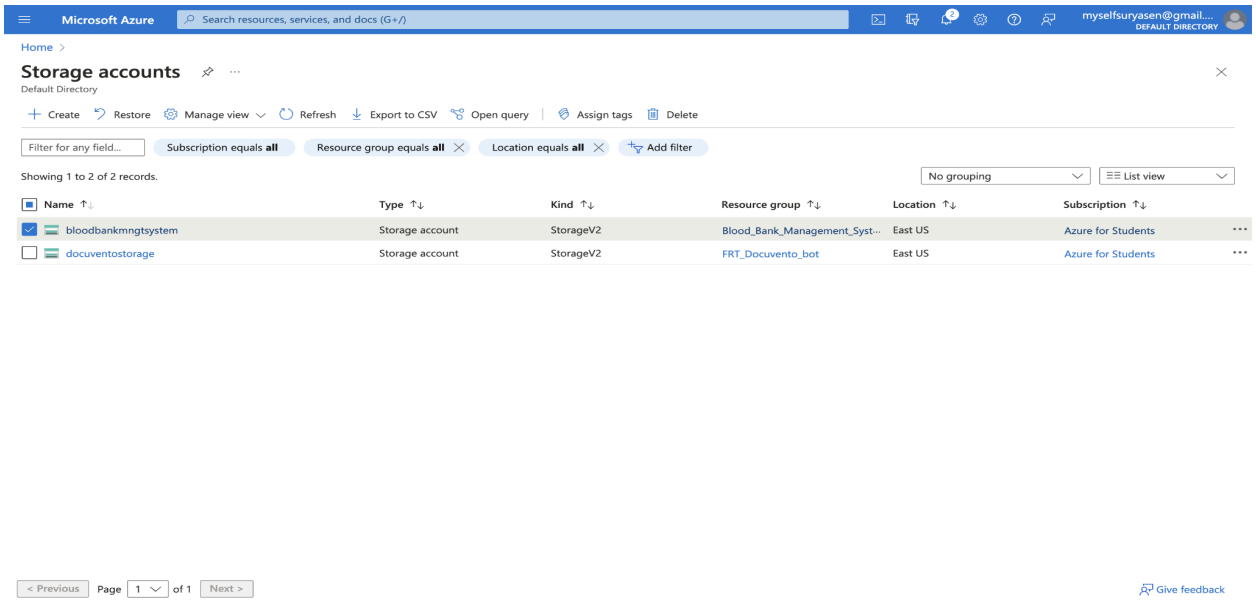
# Result and Discussion

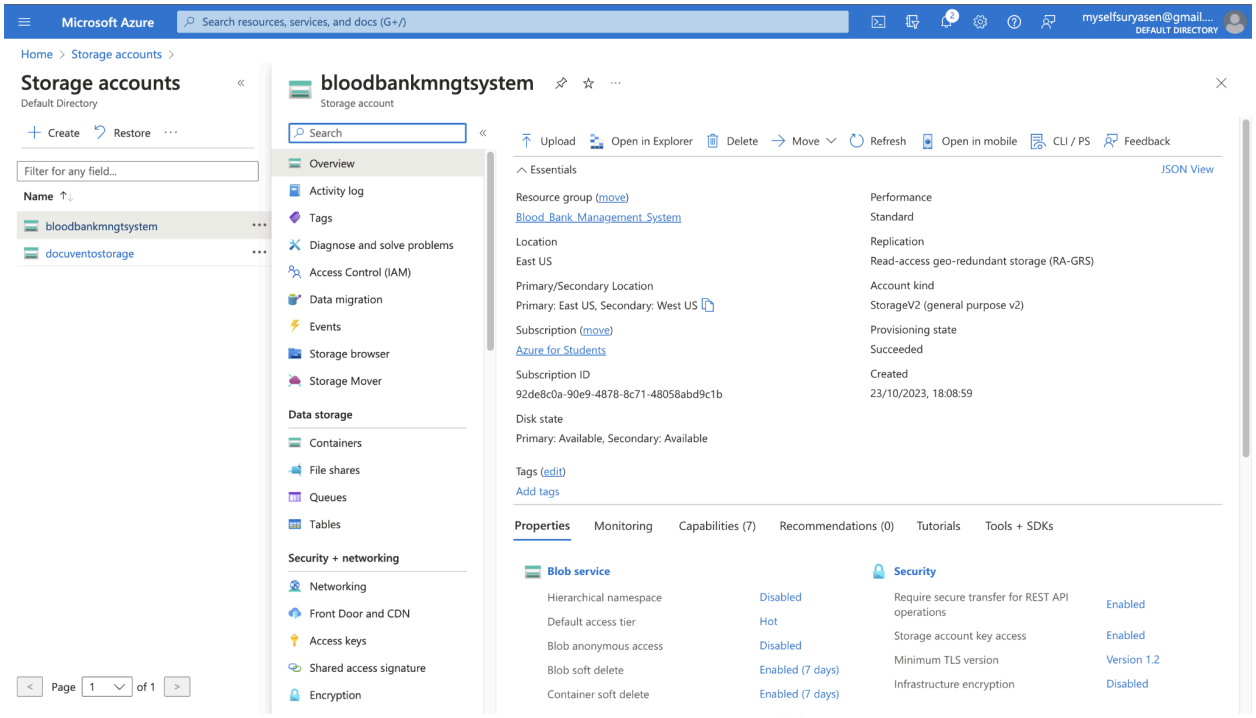
## *Screenshots*

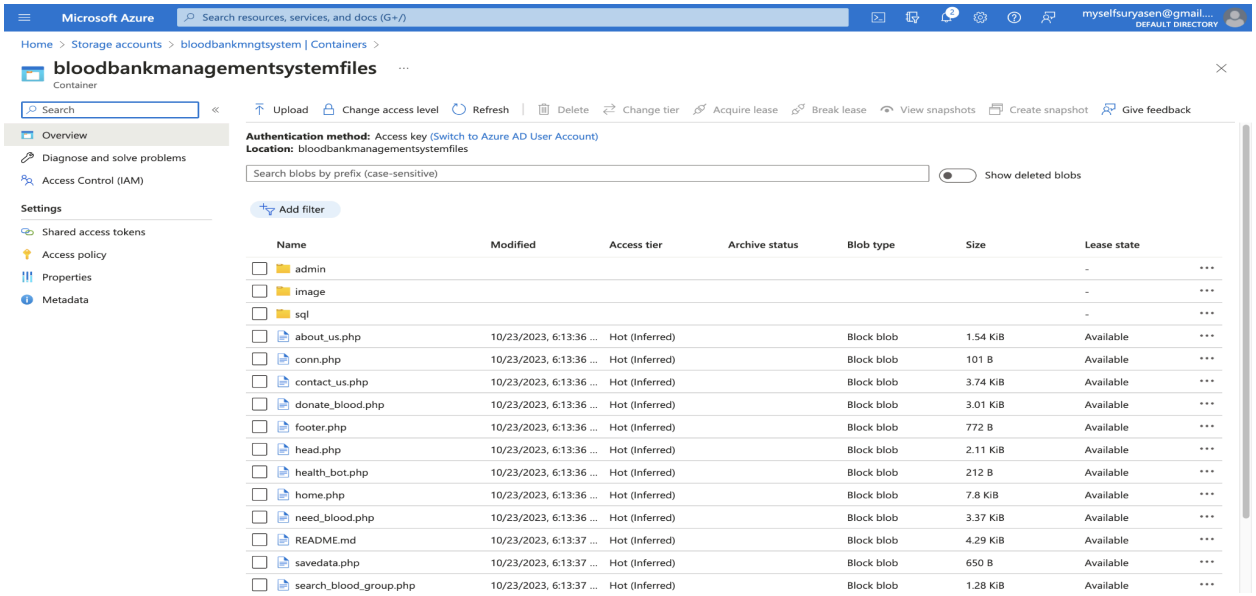
## Azure Health Bot Metrics



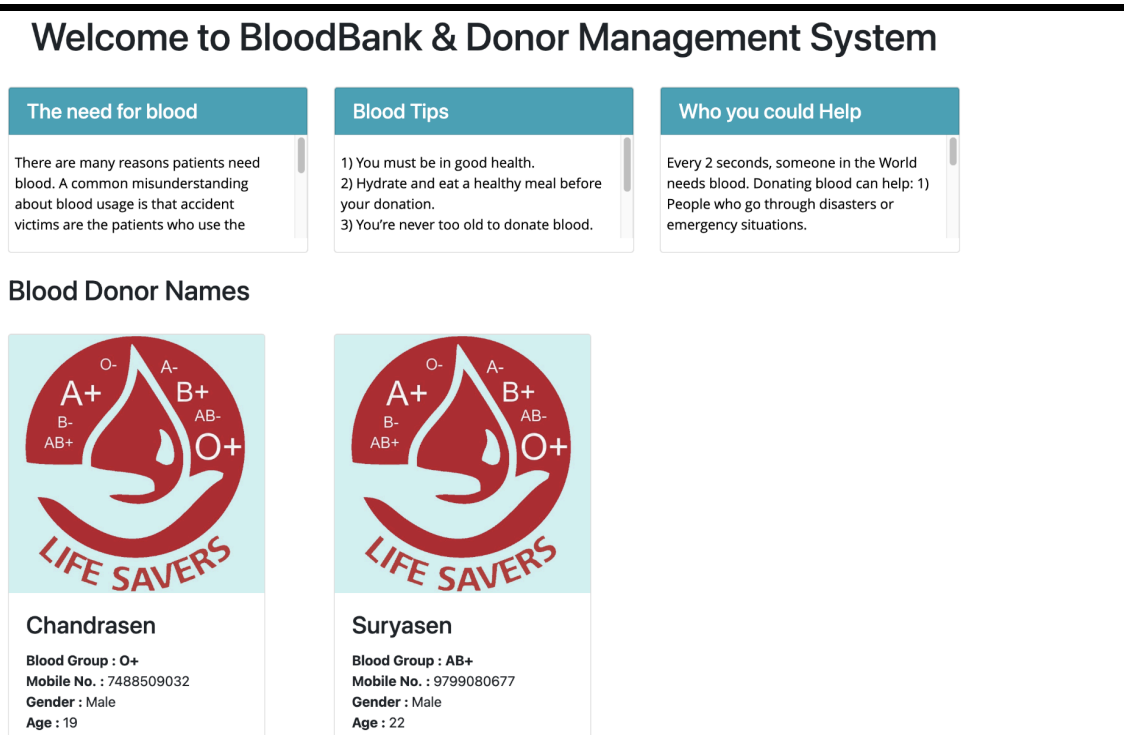

## Azure Storage Accounts and Containers



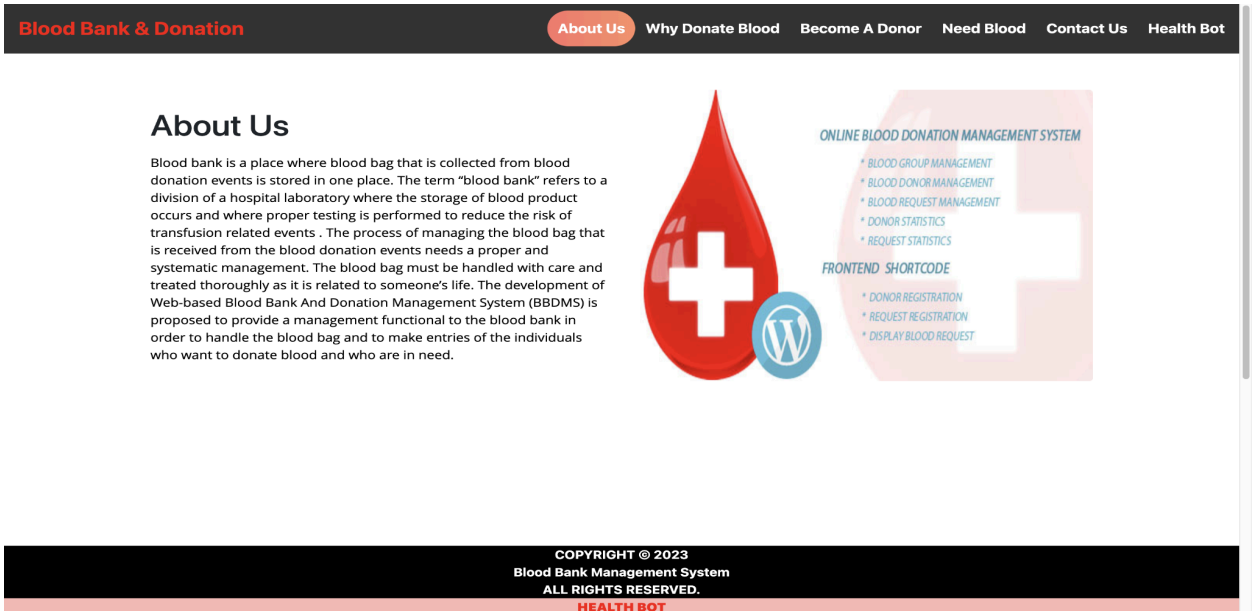

Microsoft Azure
Search resources, services, and docs (G+/)
myselfsuryasen@gmail....
DEFAULT DIRECTORY
Home > Storage accounts >
Storage accounts
Default Directory
Create
Restore
Filter for any field...
Name
bloodbankmngtsystem
docuventostorage
Page 1 of 1
bloodbankmngtsystem
Storage account
Search
Overview
Activity log
Tags
Diagnose and solve problems
Access Control (IAM)
Data migration
Events
Storage browser
Storage Mover
Data storage
Containers
File shares
Queues
Tables
Security + networking
Networking
Front Door and CDN
Access keys
Shared access signature
Encryption
Upload
Open in Explorer
Delete
Move
Refresh
Open in mobile
CLI / PS
Feedback
Essentials
JSON View
Resource group (move)
Blood_Bank_Management_System
Location
East US
Primary/Secondary Location
Primary: East US, Secondary: West US
Subscription (move)
Azure for Students
Subscription ID
92de8c0a-90e9-4878-8c71-48058abd9c1b
Disk state
Primary: Available, Secondary: Available
Performance
Standard
Replication
Read-access geo-redundant storage (RA-GRS)
Account kind
StorageV2 (general purpose v2)
Provisioning state
Succeeded
Created
23/10/2023, 18:08:59
Tags (edit)
Add tags
Properties
Monitoring
Capabilities (7)
Recommendations (0)
Tutorials
Tools + SDKs
Blob service
Hierarchical namespace
Disabled
Default access tier
Hot
Blob anonymous access
Disabled
Blob soft delete
Enabled (7 days)
Container soft delete
Enabled (7 days)
Security
Require secure transfer for REST API operations
Enabled
Storage account key access
Enabled
Minimum TLS version
Version 1.2
Infrastructure encryption
Disabled


Microsoft Azure
Search resources, services, and docs (G+/)
myselfsuryasen@gmail....
DEFAULT DIRECTORY
Home > Storage accounts > bloodbankmngtsystem | Containers >
bloodbankmanagementsystemfiles
Container
Search
Overview
Diagnose and solve problems
Access Control (IAM)
Settings
Shared access tokens
Access policy
Properties
Metadata
Upload
Change access level
Refresh
Delete
Change tier
Acquire lease
Break lease
View snapshots
Create snapshot
Give feedback
Authentication method: Access key (Switch to Azure AD User Account)
Location: bloodbankmanagementsystemfiles
Search blobs by prefix (case-sensitive)
Show deleted blobs
Add filter
Name
Modified
Access tier
Archive status
Blob type
Size
Lease state
admin
image
sql
about_us.php 10/23/2023, 6:13:36 ... Hot (Inferred) Block blob 1.54 KiB Available
conn.php 10/23/2023, 6:13:36 ... Hot (Inferred) Block blob 101 B Available
contact_us.php 10/23/2023, 6:13:36 ... Hot (Inferred) Block blob 3.74 KiB Available
donate_blood.php 10/23/2023, 6:13:36 ... Hot (Inferred) Block blob 3.01 KiB Available
footer.php 10/23/2023, 6:13:36 ... Hot (Inferred) Block blob 772 B Available
head.php 10/23/2023, 6:13:36 ... Hot (Inferred) Block blob 2.11 KiB Available
health_bot.php 10/23/2023, 6:13:36 ... Hot (Inferred) Block blob 212 B Available
home.php 10/23/2023, 6:13:36 ... Hot (Inferred) Block blob 7.8 KiB Available
need_blood.php 10/23/2023, 6:13:36 ... Hot (Inferred) Block blob 3.37 KiB Available
README.md 10/23/2023, 6:13:37 ... Hot (Inferred) Block blob 4.29 KiB Available
savedata.php 10/23/2023, 6:13:37 ... Hot (Inferred) Block blob 650 B Available
search_blood_group.php 10/23/2023, 6:13:37 ... Hot (Inferred) Block blob 1.28 KiB Available

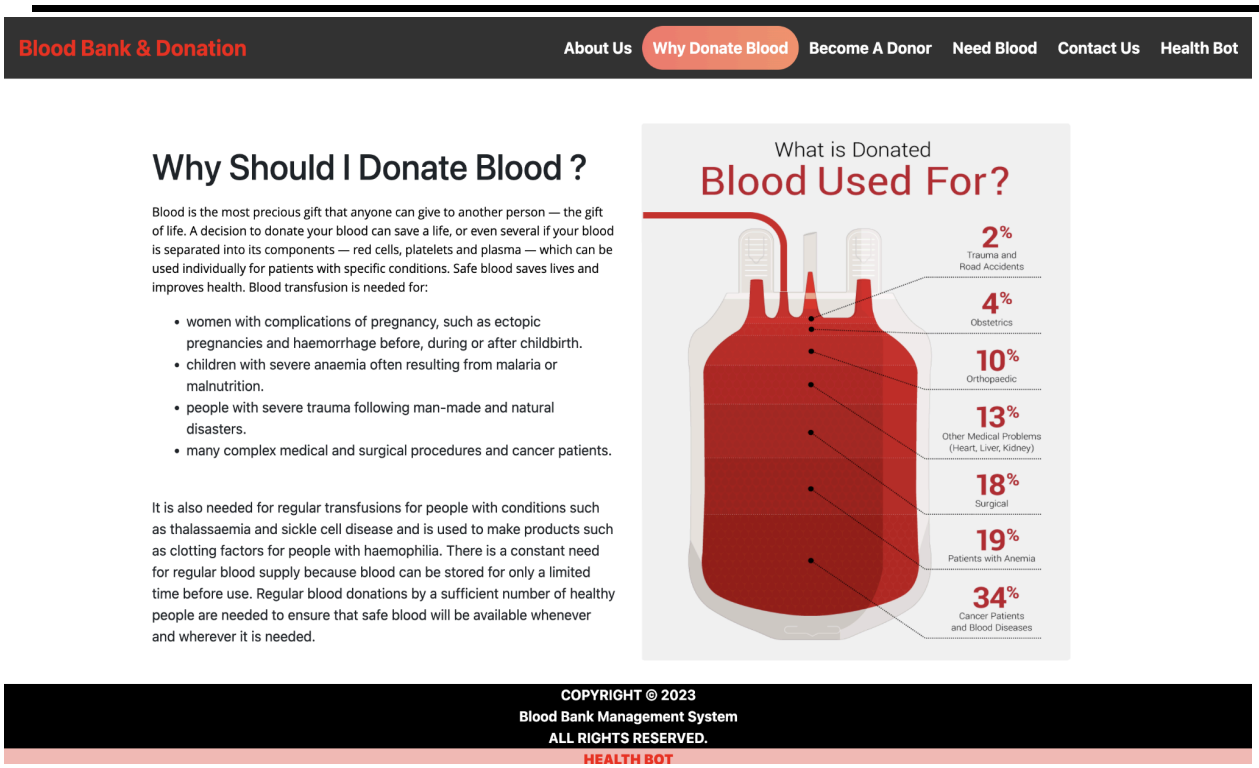
# HomeScreen Page

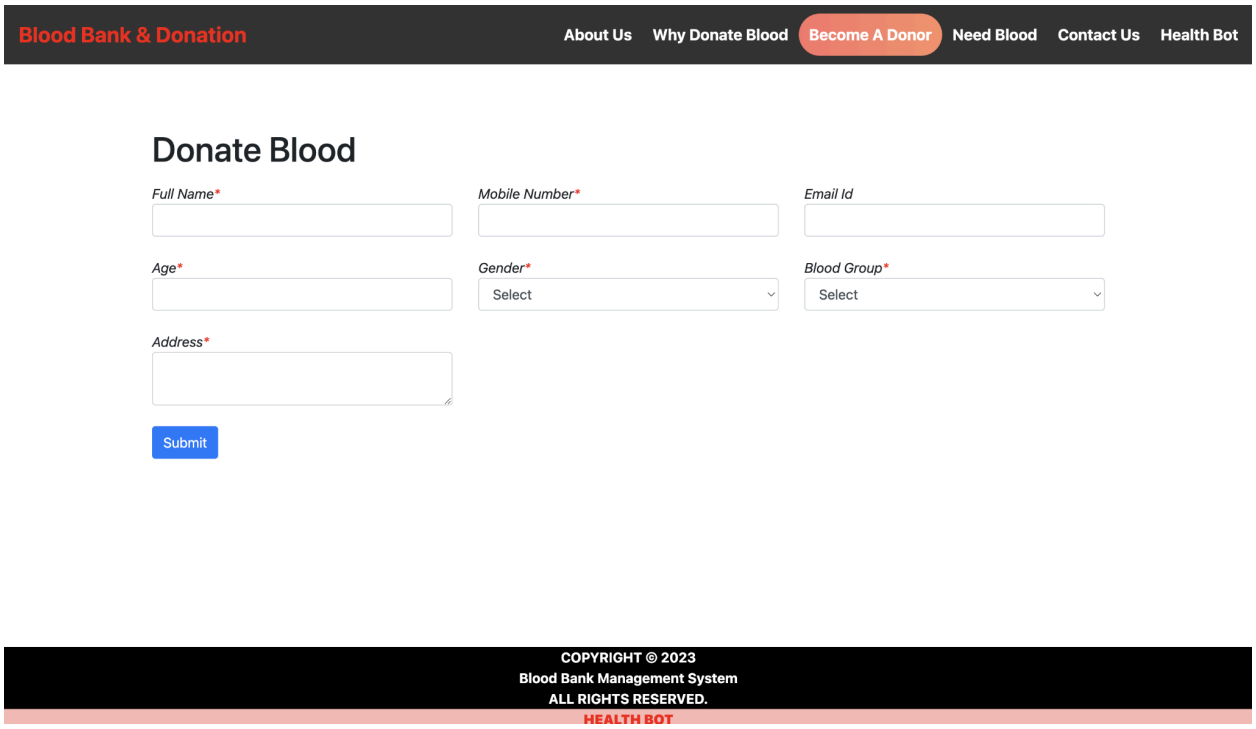
# About Page



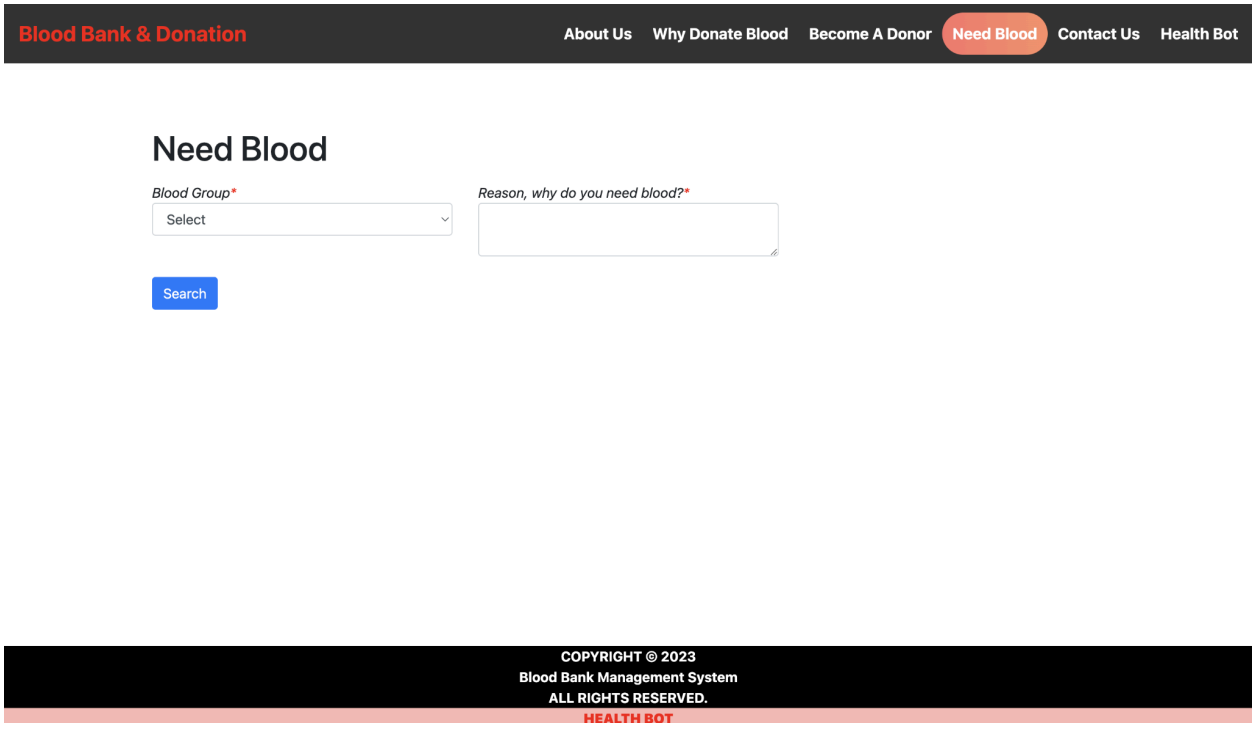

# Why Donate Blood Page

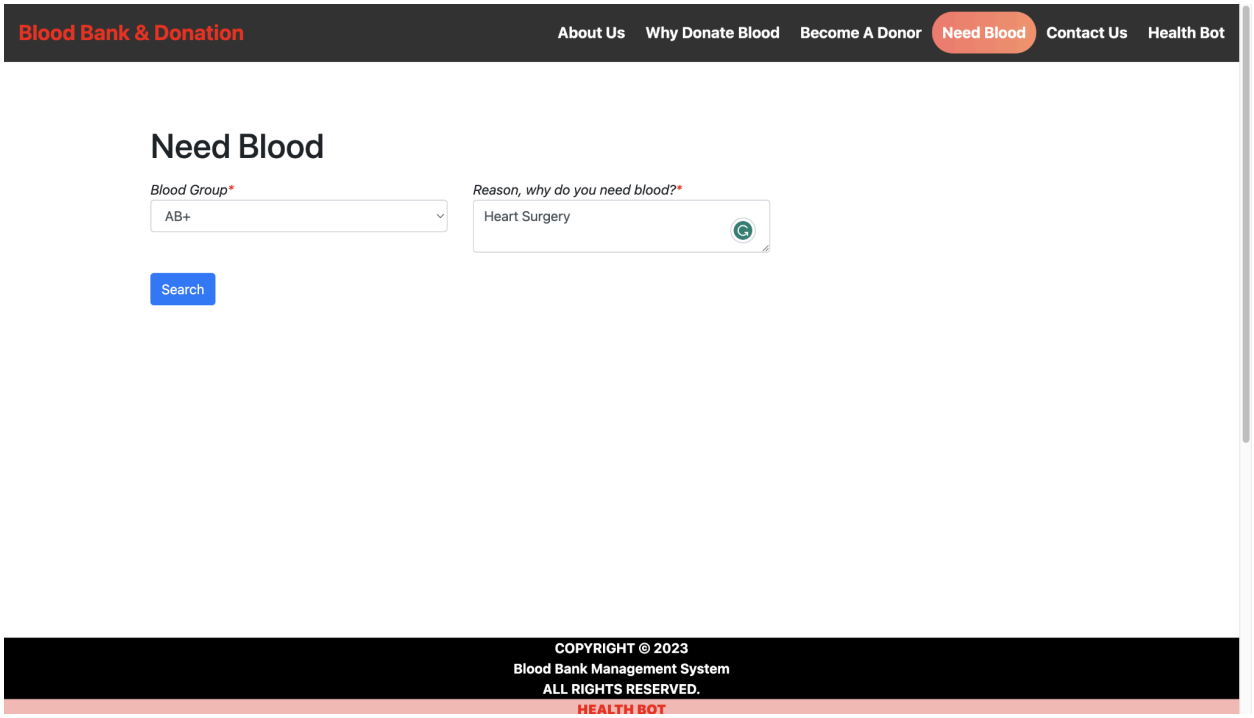


## Become a Donor Page

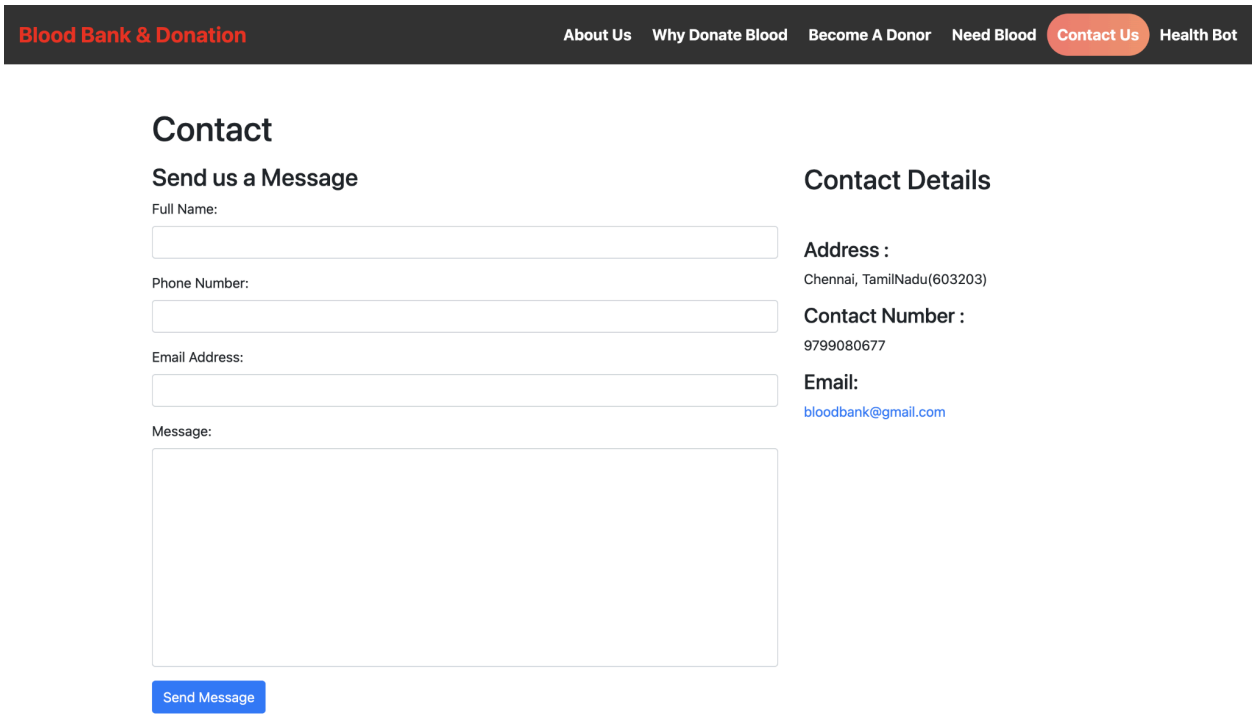



## Need Blood Page

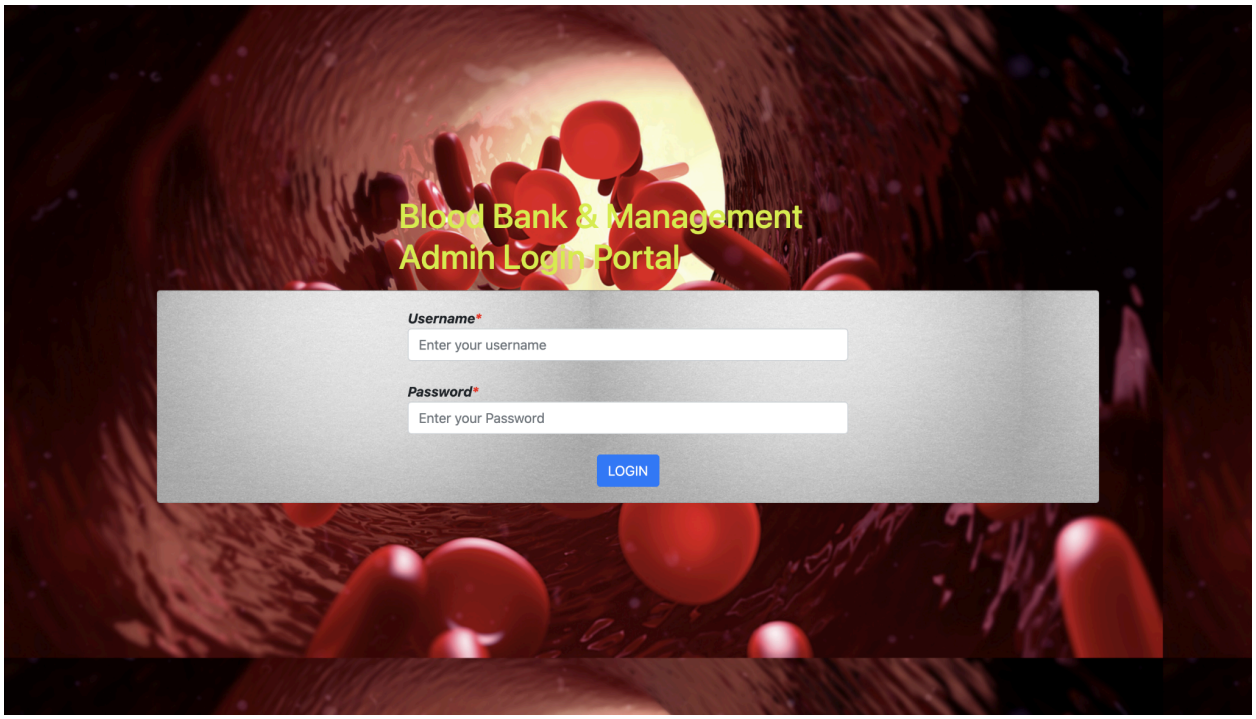




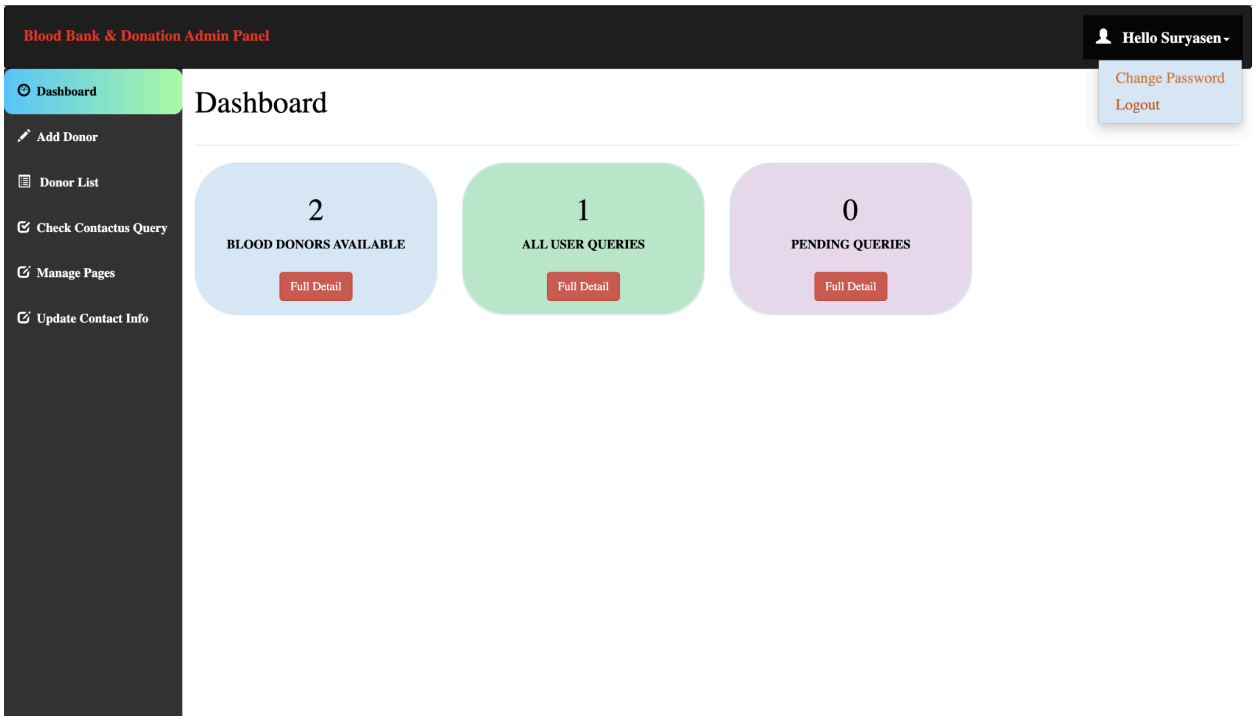

## Contact Us Page



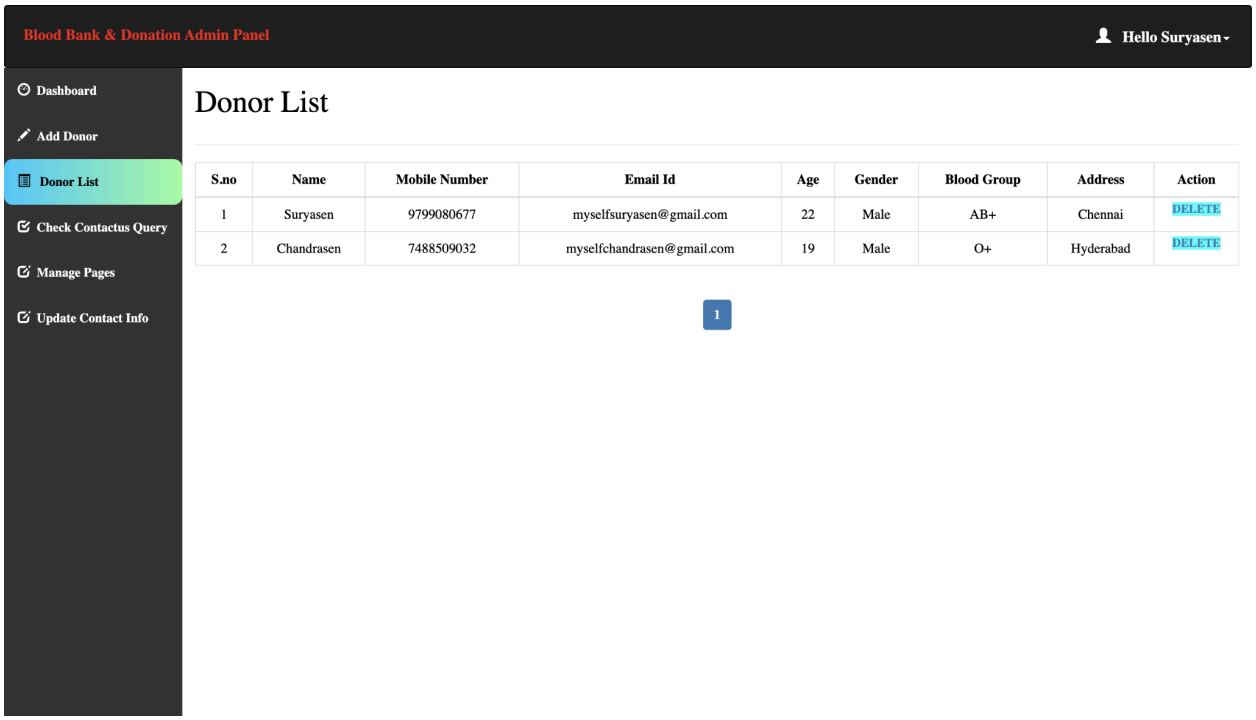
## Admin Panel



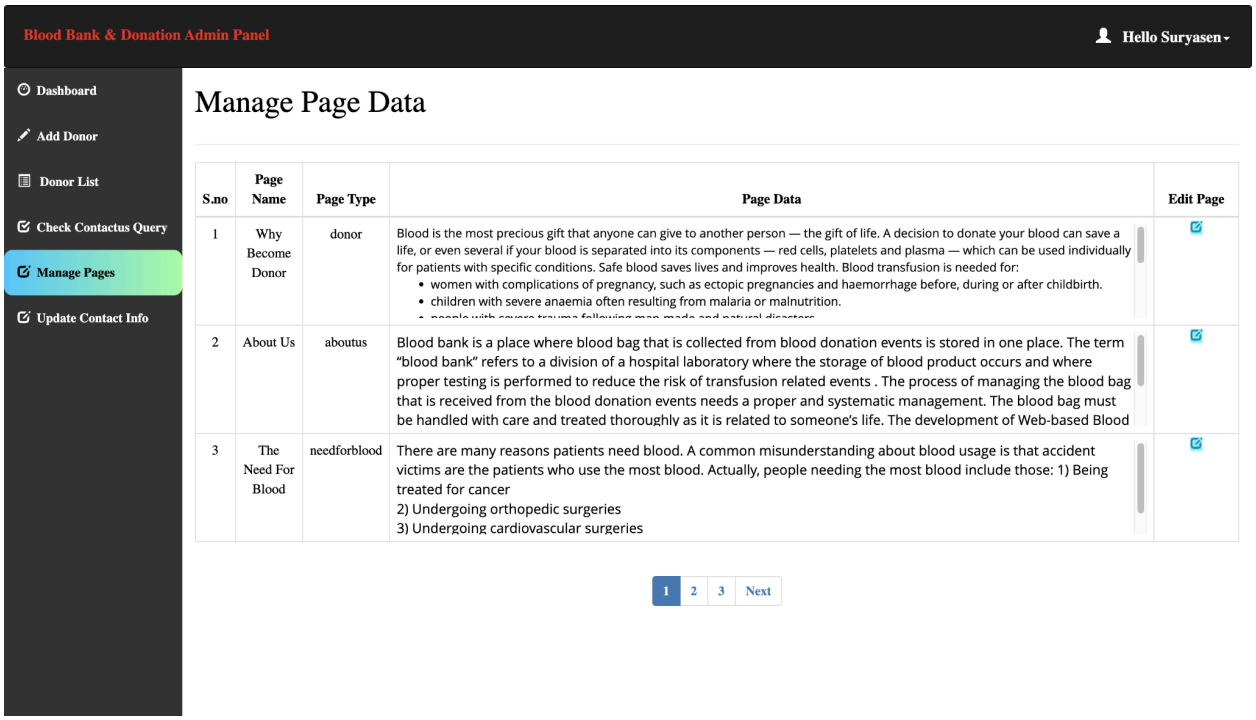

## Dashboard Page



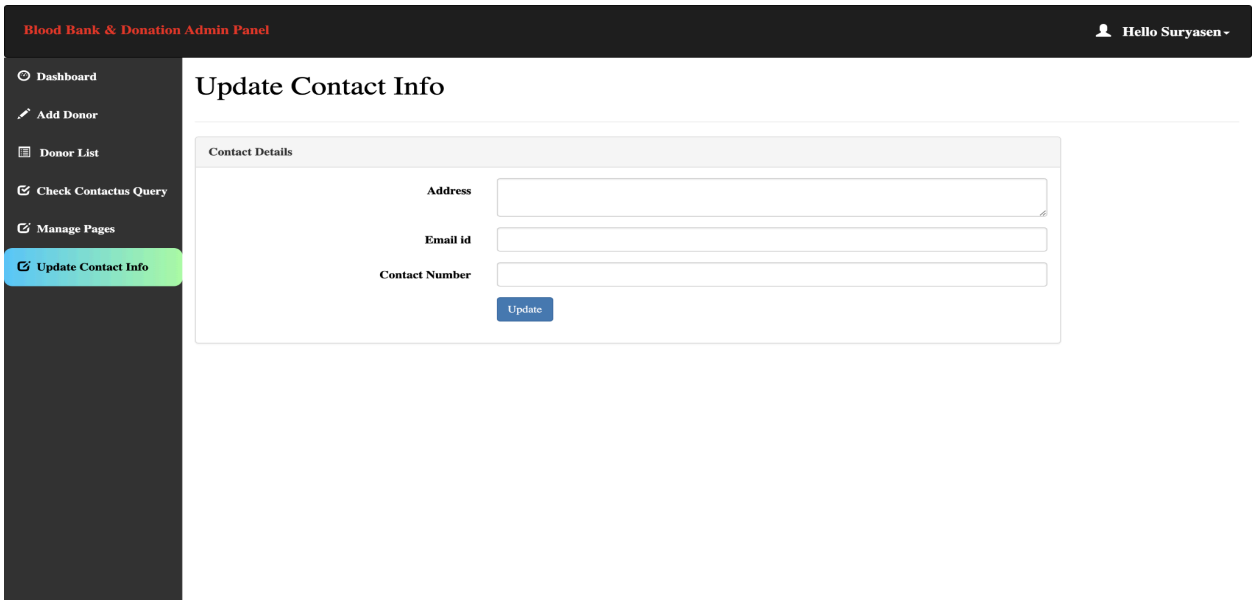
# Donor List Page




# Manage Page Data Page



## Update Contact Info Page




## Languages Used:

1. HTML
2. CSS
3. JavaScript
4. jQuery
5. PHP
6. MySQL
7. Software used

## Software Used:

1. Text editor (any)
2. Web browser (any)
3. Xampp local server

## CONCLUSION

Prior to this project, a general study of the blood bank management system was conducted from recent research of various authors and facts were gathered which helped to uncover the misfits that the system was facing. After proper analysation of these problems, a solution was then developed in order to meet up the needs of a more advanced system. This system is known as the centralized blood bank repository which helped in eliminating all the problems that the previous systems were facing. With this system, Blood banks/ Centers, Hospitals, Patients and Blood donors will be brought together to enjoy a large number of functionalities and access a vast amount of information, thereby making blood donation and reception a lot easier and faster.

Before implementing the database, in the design phase, we explored various features, operations of a blood bank to figure out required entities, attributes and the relationship among entities to make an efficient Entity Relationship Diagram(ERD). After analyzing all the requirements, I have created our ERD and then converted the ERD to a relational model and normalized the tables. Using SQL Server I have created the tables for my database and inserted some sample values in the tables. Finally, I have executed sample queries on the database to check its performance to retrieve useful information accurately and speedily.

## FUTURE WORK

In light of the current development in computing where everything is moving to cloud technology, our CBBR system is developed with the future in mind and it is therefore scalable and can easily be transformed into a cloud server that various blood banks can tap into and get required data and utilize various functionalities. On a short-term basis however, we are looking into SMS integration, where alerts and notifications will be sent to users' mobile phones.

# REFERENCES


1. "A Decision Support System for Blood Bank Inventory Management: A Simulation Approach" by G. Lanzarone, M. Papa, and A. Pizzuti. Published in the Journal of Medical Systems.
2. "Database Management Systems in Blood Bank Automation " by S. Dabke, M. Mahajan, and A. Pachpute. Published in the International Journal of Advanced Research in Computer Science.SQL for Mere Mortals" by John L. Viescas and Michael J. Hernandez.
3. https://www.w3schools.com/sql/sql_unique.asp
   https://www.w3schools.com/sql/sql_join.asp
   https://www.tutorialspoint.com/plsql/index.htm
4. https://www.tutorialspoint.com/plsql/plsql_exceptions.h tm
5. https://www.geeksforgeeks.org/sql-ddl-dql-dml-dcl-tcl- commands/
6. https://www.geeksforgeeks.org/sql-trigger-student- database/
7. https://www.w3schools.com/sql/sql_view.asp
8. https://docs.microsoft.com/en-us/sql/sql- server/?view=sql-server-ver15
9. “Database System Concepts" by Abraham Silberschatz, Henry F. Korth, and S. Sudarshan.
10. "Fundamentals of Database Systems" by Ramez Elmasri and Shamkant B. Navat