Blood Bank Management System

Name	Abdul Rahman Anas A
Roll no	7376221BM104
Department	Biomedical Engineering
Language Used	SQL
Problem Statement	Automate blood inventory, ensure quality, and improve tracking efficiency.

1. Introduction

1.1. Purpose:

The Blood Bank Management System aims to simplify and automate the processes of blood collection, storage, distribution, and monitoring. It ensures hospitals and patients have access to clean and safe blood while minimizing the challenges and delays of manual operations. By efficiently managing information about donors and recipients, the system enhances data traceability and supports better decision-making. Additionally, it offers real-time updates on blood inventory, enabling hospitals to quickly obtain the required blood types. Designed to be scalable, cost-effective, and user-friendly, the system overcomes the shortcomings of traditional methods, improving both operational efficiency and the overall quality of healthcare services.

1.2. Scope of Project:

The Blood Bank Management System aims to automate essential functions like registering donors, monitoring blood quality, managing inventory, and tracking recipients. It streamlines blood request processing from hospitals, ensuring timely availability of necessary blood types. The system includes features for managing donor and recipient data, tracking blood quality, and producing real-time reports. It is designed to scale, accommodating large databases and multiple locations. By replacing manual processes, it enhances operational efficiency, reduces errors, and improves healthcare delivery. Additionally, the system can be integrated with hospital management systems for seamless operations.

2. System Overview:

2.1. Users:

1. **Blood Donors**: Individuals who register and donate blood.
2. **Recipients**: Patients or individuals in need of blood.
3. **Blood Bank Managers**: Responsible for overseeing blood inventory, quality, and hospital requests.
4. **Hospital Staff**: Request blood supplies and track availability.
5. **Recording Staff**: Register donor and recipient details into the system.
6. **Disease Checkers**: Medical personnel who ensure blood quality and test for contamination.
7. **System Administrators**: Manage and maintain the system infrastructure.

2.2 Features:

1. **Donor Management**: Covers registration, management, and tracking donor history and eligibility.

2. **Recipient Management**: Describes managing recipient data, blood requirements, and matching with available inventory.

3. **Blood Inventory Management**: Focuses on tracking blood samples, including their status (pure or contaminated) and expiration.

4. **Quality Control**: Ensures blood is properly tested and certified, while unsuitable samples are recorded and disqualified.

5. **Hospital Integration**: Details how the system integrates with hospitals for blood requests and distribution tracking.

6. **Reporting and Analytics**: Includes the ability to generate detailed reports for inventory, donor activity, and distribution, providing insights for operational efficiency

7. **User-Friendly Interface**: Ensures easy navigation and workflow management for different types of users.

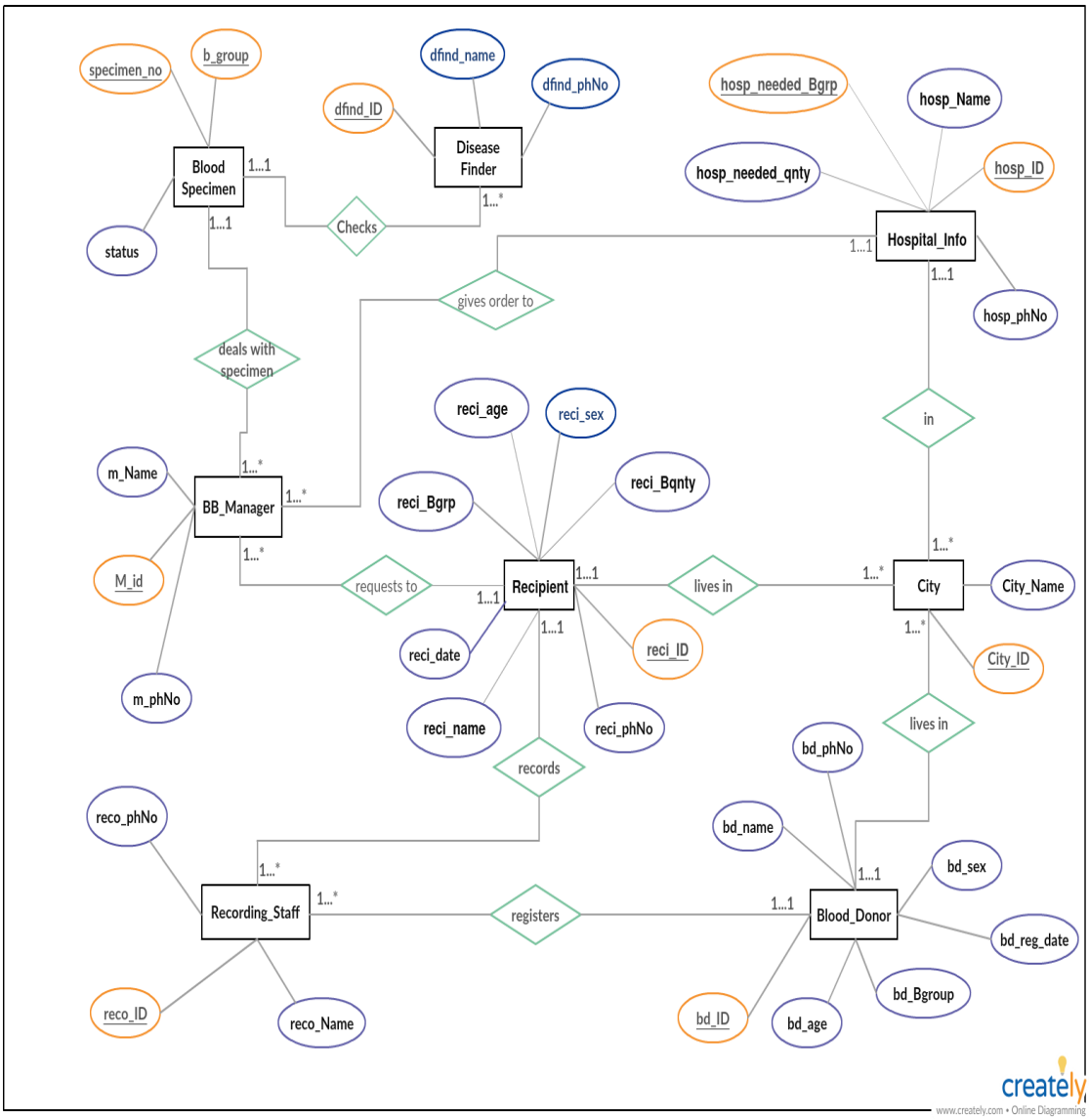
8. **Secure Database Management**: Stresses the importance of data security and access control.

9. **Automation and Alerts**: Highlights the automation of key processes, such as reminders and low inventory alerts.

10. **Scalability**: Ensures that the system is designed for expansion, supporting multiple locations and integrating with hospital management systems**.**

ER DIAGRAM USING CREATLY AND RELATION BETWEEN THE ENTITIES

3.1 Functional Requirements:



1. **User Authentication and Authorisation:**

- Secure login for system users (staff, administrators, etc.).
- Role-based access control to restrict functionalities based on user roles.

2. **Donor Management**:

- Register new donors and update donor profiles.
- Maintain records of donor eligibility, blood group, and contact details.

- Notify donors about upcoming donation opportunities.

3. **Recipient Management**:
- Register recipients and record their blood requirements.
- Match recipient requests with available blood inventory.

4. **Blood Inventory Management**:
- Maintain records of blood samples by type, quantity, and status.
- Update inventory in real-time after donations, testing, or distribution.
- Manage expiry dates and contaminated samples.

5. **Blood Quality Testing**:
- Record results of blood tests conducted by medical staff.
- Flag contaminated samples and mark them as unavailable.

6. Hospital Management:
- Enable hospitals to request specific blood types and quantities.
- Track and fulfill hospital requests efficiently.

7. **Reporting and Analytics**:
- Generate reports on donor activity, blood inventory, and hospital requests.
- Provide data insights for decision-making and process improvement.

8. **Alerts and Notifications**:
- Notify staff about low inventory levels and expiring blood samples.
- Send reminders to donors for next eligible donation dates.

9. **Data Security and Backup**:

- Ensure secure storage of sensitive donor and recipient data.
- Regularly back up the database to prevent data loss.

10. **Scalability and Integration**:

- Support multi-location blood banks with centralized access.
- Integrate with hospital management systems or national health networks.

3.2. Non-Functional Requirements:

Performance:

- The system should handle concurrent access by multiple users without performance degradation.
- Must process requests (e.g., blood search, donor registration) within 2 seconds.

Scalability:

- The system should support scalability to accommodate increased data (e.g., larger donor and recipient databases).
- Capable of managing multiple blood bank locations under one system.

Availability:

- Ensure 99.9% uptime to provide uninterrupted access to users, especially in emergencies.
- Include failover mechanisms to maintain service during server outages.

Usability:

- Provide an intuitive, user-friendly interface for donors, recipients, and staff.
- Include clear navigation and minimal learning curve for first-time users.

Security:

- Use encryption for sensitive data such as donor and recipient personal information.

- Implement role-based access control to restrict unauthorized access.

Data Integrity:

- Ensure accuracy and consistency of data stored in the database.
- Implement validation checks for all user inputs.

Reliability:

- Ensure the system operates correctly under all defined conditions.
- Provide accurate data retrieval and processing without errors.

Maintainability:

- System code should be modular and well-documented for easier maintenance and updates.
- Regular updates should be possible without affecting system operations.

Backup and Recovery:

- Implement regular automatic backups to prevent data loss.
- Ensure data can be restored within 30 minutes in case of failure.

Compliance:

- Adhere to relevant data protection and healthcare regulations, such as GDPR or HIPAA.

Compatibility:

- Ensure compatibility with multiple platforms and devices (e.g., web browsers, mobile devices).
- Support integration with third-party applications like hospital management systems.

INFORMATION OF ENTITIES

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

1. **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 primary key to identify the donor information. Other than that name, age , sex , blood group, phone number and registration dates will be stored in database under Blood_Donor entity.

2. **Recipient: (Attributes – reci_ID, reci_name, reci_age, reci_Bgrp, reci_Bqnty , reci_sex, reci_reg_date, reci_phNo)**

   The Recipient is the person who recivies blood from blood bank, when blood is given to a recipient a rericipient ID (reci_ID) is generated and used as primary key for the recipient entityto indentify blood recipients information. Along with it name ,age, sex, blood group (needed), blood quantity(needed) , phone number, and registration dates are also stored in the data base under recipient entity.

3. **BB_Manager: (Attributes – m_ID, m_Name, m_phNo)**

   The blood bank manager is the person who takes care of the avaible blood samples in the blood bank, he is also resposible for handaling blood requests from recipients and hospitals. Blood manager has a unique indentfication number (m_ID) used as primary key along with name and phone number of blood bank manager will be stored in data base under BB_Manager entity.

4. **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 enitity has reco_ID which is primary key along with recoder's name and recodrer's phone number will also be stored in the data base under Recording_Staff entity.

5. **BloodSpecimen : (Attributes – specimen_number, b_group , status)**

   In data base, under BloodSpecimen 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 primary key along with status attribute which will show if the blood is contaminated on not.

6. **DiseaseFinder : (Attributes - dfind_ID, dfind_name, dfind_PhNo)**

   In data base , under DiseaseFinder entity we will store the information of the doctor who checksthe blood for any kind of contaminations. To store that information we have unique identification number (dfind_ID) as primary key. Along with name and phone number of the doctor will also

be stored under same entitity.

7. Hospital_Info : (Attributes – hosp_ID, hosp_name, hosp_needed_Bgrp, hosp_needed_Bqnty)

   In the data base, under Hospital_Info entity we will store the information of hospitals. In this hosp_ID and hosp_needed_Bgrp toether makes the primary key. We will store hospital name and the blood quantity reqiured at the hospital.

8. city: (Attributes- city_ID, city_name)

   This entity will store the information of cities where donors, recipients and hospitals are present. A unique indentification number (City_ID) will be used as primary key to indefiy the information about the city. Along with ID city names will also be stored under this entity.

RELATIONSHIP BETWEEN ENTITIES

9. **City and Hospital_Info:**

   Relationship = "in"
   Type of relation = 1 to many
   Explanation = A city can have many hospital in it. One hospital will belong in one city.

1. **City and Blood_Donor:**

   Relationship = "lives
   in" Type of relation =
   1 to many
   Explanation = In a city, many donor can live. One donor will belong to one city.

2. **City and Recipient:**

   Relationship = "lives
   in" Type of relation =
   1 to many
   Explanation = In a city, many recipient can live. One recipient will belong to one city.

3. **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.

4. **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.

5. **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 request to on blood bank manager.

6. **BB_Manager and Blood Specimen:**

   Relationship = "deales with specimen" Type of relation = 1 to many

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

7. **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.

8. **Disease_finder and Blood Specimen:**

Relationship =
"checks", Type of
relation = 1 to many
Explanation = A disease finder can check many blood samples. One blood sample ischecked by one disease finder.

Explanation = One Blood bank manager can handle and process requests from manyhospitals. One hospital will place request to on blood bank manager.

9. BB_Manager and Blood Specimen:

Relationship = "deales with
specimen"Type of relation =
1 to many
Explanation = One Blood bank manager can manage many blood specimen and onespecimen will be managed by one manager.

10. 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 canhandle requests from many recipients.

11. Disease_finder and Blood Specimen:

Relationship =
"checks", Type of
relation = 1 to many
Explanation = A disease finder can check many blood samples. One blood sample ischecked by one disease finder.

RELATIONAL SCHEMAS

Attribute Name	Description	Type
bd_id	Blood Donor's Id	int
bd_Name	Blood Donor's Name	varchar
bd_age	Blood Donor's Age	int
bd_sex	Blood Donor's Sex	char
bd_bgrp	Blood Donor's blood group	varchar
bd_regdate	Registration Date of Donor	date
reco_id	Id of Recording Staff	int
city_id	City Id	int

- The relationship with Recording staff and Donor is 1 to many. That's why primary key ofRecording staff is used as a foreign key in Donor.

- The relationship with City and Donor is 1 to many. That's why primary key of City is used as aforeign key in Donor.

Recipient Table:

Attributes Name	Description	Type
reci_id	Recipient's Id	int
reci_Name	Recipient's Name	varchar
reci_age	Recipient's age	int
reci_sex	Recipient's sex	char
reci_bgrp	Recipient's blood group	varchar
reci_bqnty	Recipient's blood quantity	int
reci_reg_date	Recipient's registration	date

	date	
reco_id	Recording Staff's Id	int
city_id	City's unique Id	int
M_id	Blood Bank Manager's Id	int

- The relationship with Recording staff and Blood Recipient is 1 to many. That's why primary key of Recording staff is used as a foreign key in Blood Recipient.

- The relationship with City and Blood Recipient is 1 to many. That's why primary key of City is used as a foreign key in Blood Recipient.

- The relationship with Blood Bank Manager and Blood Recipient is 1 to many. That's why primary key of Blood Specimen is used as a foreign key in Blood Recipient.

City Table:

Attributes Name	**Description**	**Type**
city_id	City's unique id	int
city_name	City's name	varchar

- The relationship between City and Recipients, Donor, Hospital info are all of 1 to many. So that's why primary key of City is used as a foreign key in Recipients, Donor and Hospital info.

Recording Staff Table:

Attributes Name	**Description**	**Type**
reco_id	Recording Staff's id	int
reco_name	Recording Staff's Name	Varchar
reco_PhNo	Recording Staff's Phone number	bigint

- The relationship between Recording Staff and Blood Donor, Recipients are all of 1 to many. That's why the primary key of Recording staff is used as a foreign key in Donor and Recipient.

Blood Specimen Table:

Attributes Name	**Description**	**Type**
specimen_No	Blood Sample's unique id	int
b_grp	Blood Group	varchar
status	Whether blood is pure or not?	int
M_id	Blood Bank Manager's id	int
dfind_id	Disease Finder's unique id	int

- The relationship with Disease finder and Blood Specimen is 1 to many. That's why primary key ofDisease finder is used as a foreign key in Blood Specimen.

- The relationship with Blood Bank manager and Blood Specimen is 1 to many. That's why primarykey of Blood Bank manager is used as a foreign key in Blood Specimen

.

Disease Finder Table:

Attributes Name	**Description**	**Type**
dfind_id	Disease Finder's unique id	Int
dfind_name	Disease Finder's name	varchar
dfind_phNo	Disease Finder's phone number	bigint

- The relationship with Disease finder and Blood Specimen is of 1 to many. Therefore, theprimary key of Disease finder is used as a foreign key in Blood Specimen.

Blood Bank Manager Table:

Attributes Name	**Description**	**Type**
M_id	Blood Bank Manager's id	int
m_name	Blood Bank Manager's name	varchar
m_phNo	Blood Bank Manager's phoneno	bigint

- The relationship between Blood Bank Manager and Blood Specimen, Recipient, Hospital info are all of 1 to many. So therefore, the primary key of Blood Bank Manager is used as a foreignkey in Blood Specimen, Recipient and Hospital info.

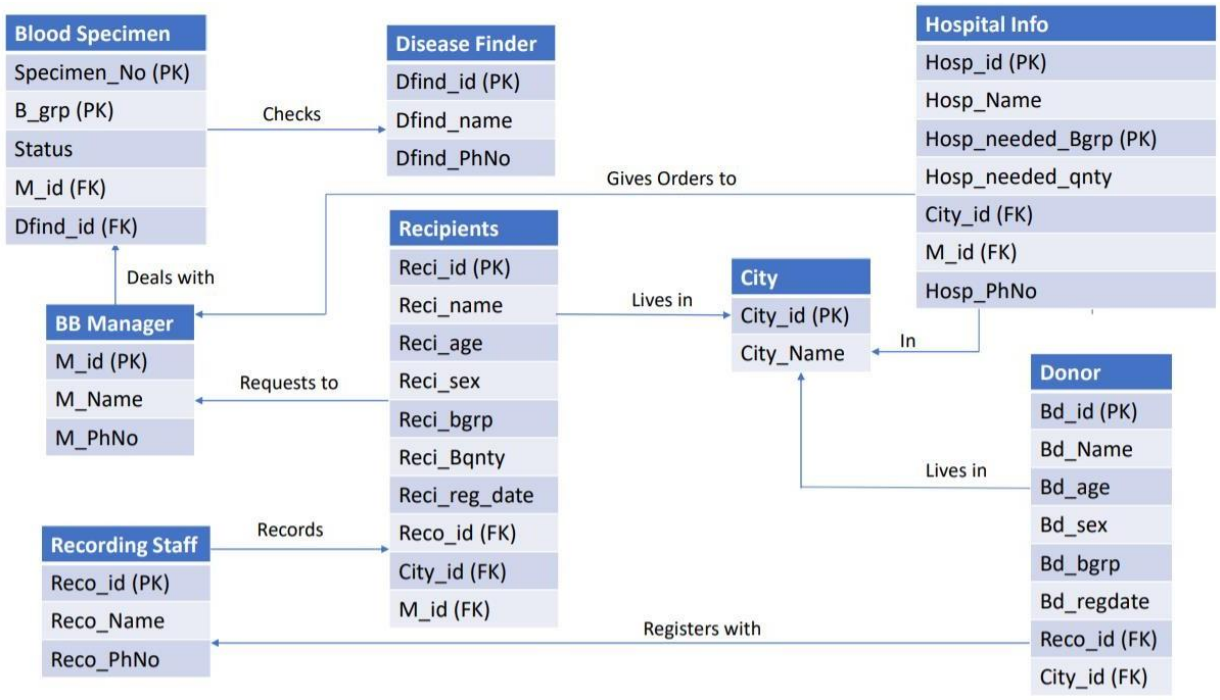
Hospital info Table:

Attributes Name	**Description**	**Type**
hosp_id	Hospital's unique id	int
hosp_name	Hospital's name	varchar
hosp_needed_Bgrp	Blood group needed by hospital	varchar
hosp_needed_qnty	Quantity of blood group needed	int
city_id	City's unique id	int
M_id	Blood Bank Manger's id	int

- The relationship with City and Hospital info is 1 to many. That's why primary key of City is usedas a foreign key in Hospital info.

- The relationship with Blood Bank Manager and Hospital info is 1 to many. That's why primary key of Blood Bank manager is used as a foreign key in Hospital info.

ER DIAGRAM WITH TABLES



NORMALIZATION

Normalization Rule

Normalization rules are divided into the following normal forms:

1. First Normal Form
2. Second Normal Form
3. Third Normal Form

First Normal Form (1NF)

For a table to be in the First Normal Form, it should follow the following 4 rules:

1. It should only have single (atomic) valued attributes/columns.
2. Values stored in a column should be of the same domain
3. All the columns in a table should have unique names.
4. And the order in which data is stored, does not matter.

Second Normal Form (2NF)

For a table to be in the Second Normal Form,

1. It should be in the First Normal form.
2. And, it should not have Partial Dependency.

Third Normal Form (3NF)

A table is said to be in the Third Normal Form when,

1. It is in the Second Normal form.
2. And, it doesn't have Transitive Dependency. [3][6]

Normalization of Blood Bank database:

1. **Blood_Donor (bd_Id, bd_name, bd_phNo bd_sex, bd_age, bd_reg_date, bd_Bgroup, reco_ID,City_ID)**

{bd_Id} = > {bd_name} (functional dependency exists, because two different bd_namedo not correspond to the same bd_Id).
{bd_ID} = > {bd_sex} (functional dependency exists).
{bd_ID} = > {bd_age} (functional dependency exists).
{bd_ID} = > {bd_reg_date} date (functional dependency exists).
{bd_ID} = > {reco_id} (functional dependency exists).

{bd_ID} = > {city_id} (functional dependency exists).
{bd_ID} = > {bd_Bgroup} (functional dependency exists).

As the attributes of this table does not have sub attributes, it is in first normal form. Because every non-primary key attribute is fully functionally dependent on the primary key of the table and it is already in first normal form, this table is now in second normal form. Since the table is in second normal form and no non-primary key attribute is transitively dependent on the primary key, the table is now in 3NF.

2. **City (city_id , city_name)**

{city_id}= > {city_name}

The table is in first normal form. The table is in second normal form. The table is in third normal form.

3. **Recording_staff (reco_name, reco_ID, reco_phNo)**

{reco_id} = > {reco_name} (functional dependency exists).
{reco_id} = > {reco_phNo} (functional dependency exists).

The table is in first normal form. The table is in second normal form. The table is in third normal form.

4. **Blood_recipient (reci_Id, reci_sex, reci_phNo, reci_age, reci_date, reci_name, reci_Bqnty, reci_Bgrp, reco_id, city_id, m_id)**

{reci_Id} = > {reci_sex} (functional dependency exists).
{reci_Id} = > {reci_age} (functional dependency exists).
{reci_Id} = > {reci_date} (functional dependency exists).
{reci_Id} = > {reci_name} (functional dependency exists).
{reci_Id} = > {reci_bqnty} (functional dependency exists).

{reci_Id} = > {reci_Bgrp} (functional dependency exists).
{reci_Id} = > {reco_id} (functional dependency exists).
{reci_Id} = > {city_id} (functional dependency exists).
{reci_Id} = > {m_id} (functional dependency exists).

The table is in first normal form. The table is in second normal form. The table is in third normal form.

5. **Blood Specimen (b_group, specimen_no, status, dfind_id, m_id)**

{b_group, specimen _no} = > {status} (functional dependency exists).
{b_group, specimen _no} = > {dfind _id} (functional dependency exists).
{b_group, specimen _no} = > {m_id} (functional dependency exists).

The table is in first normal form. The table is in second normal form. The table is in third normal form.

6. **Disease_finder (dfind_id, dfind_name, dfind_PhNo)**

{ dfind_id } = > { dfind_name }
{ dfind_id } = > { dfind_PhNo } (functional dependency exists).

The table is in first normal form. The table is in second normal form. The table is in third normal form.

7. **BB_manager (M_id, m_name, m_phNo)**

{M_id} = >{m_name}
{M_id} = > {m_phNo} (functional dependency exists)

The table is in first normal form. The table is in second normal form. The table is in third normal form.

8. 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 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 third normal form.

TABLES AFTER NORMALIZATION

BB_Manager:

Result Grid | Filter Rows:

M_id	mName	m_phNo
106	Steve	4694959671
107	Jason	4695959671
108	Stella	4663959671
109	Monika	4673959671
110	John	4693859671
102	Jack	4693959671
103	Peter	4693959601
104	Mark	4693959677
105	Jason	4693957671

Disease_finder:

dfind_ID	dfind_name	dfind_PhNo
16	Ram	4693804123
17	Swathi	4693804223
18	Gautham	4693804323
19	Ashwin	4693804423
20	Yash	4693804523
NULL	NULL	NULL

City:

Result Grid | Filter Rows:

City_ID	City_name
1200	Austin
1300	Irving
1400	Houston
1500	Richardson
1600	Plano
1700	Frisco
1800	Arlington
1900	San Antonio
2000	Tyler
NULL	NULL

Blood_specimen:

Result Grid | Filter Rows: | Edit:

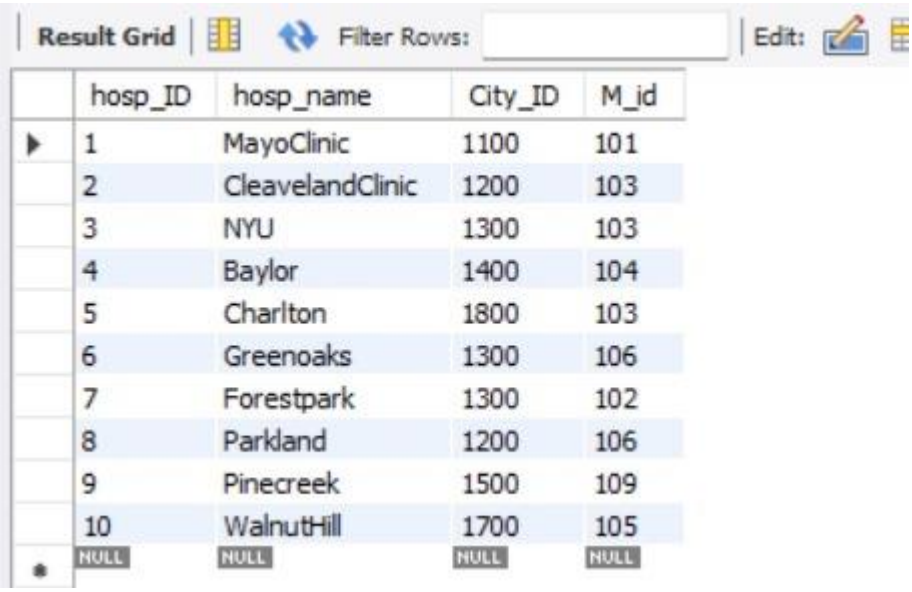
specimen_number	b_group	status	dfind_ID	M_id
1001	B+	1	11	101
1002	O+	1	12	102
1003	AB+	1	11	102
1004	O-	1	13	103
1005	A+	0	14	101
1006	A-	1	13	104
1007	AB-	1	15	104
1008	AB-	0	11	105
1009	B+	1	13	105
1010	O+	0	12	105
1011	O+	1	13	103
1012	O-	1	14	102
1013	B-	1	14	102
1014	AB+	0	15	101
NULL	NULL	NULL	NULL	NULL

Blood_Donor:

Result Grid | Filter Rows: | Edit: | Export/Import:

bd_ID	bd_name	bd_age	bd_sex	bd_Bgroup	bd_reg_date	reco_ID	City_ID
150011	Pat	29	M	O+	2015-07-19	101412	1300
150021	Shyam	42	F	A-	2015-12-24	101412	1300
150121	Dan	44	M	AB+	2015-08-28	101212	1200
150221	Mark	25	M	B+	2015-12-17	101212	1100
160011	Abdul	35	F	A+	2016-11-22	101212	1100
160031	Mike	33	F	AB-	2016-02-06	101212	1400
160091	Carrol	24	M	B-	2016-10-15	101312	1500
160101	Smith	22	M	O+	2016-01-04	101312	1200
160301	Elisa	31	F	AB+	2016-09-10	101312	1200
160401	Mark	29	M	O-	2016-12-17	101212	1200

Hospital_info_2:

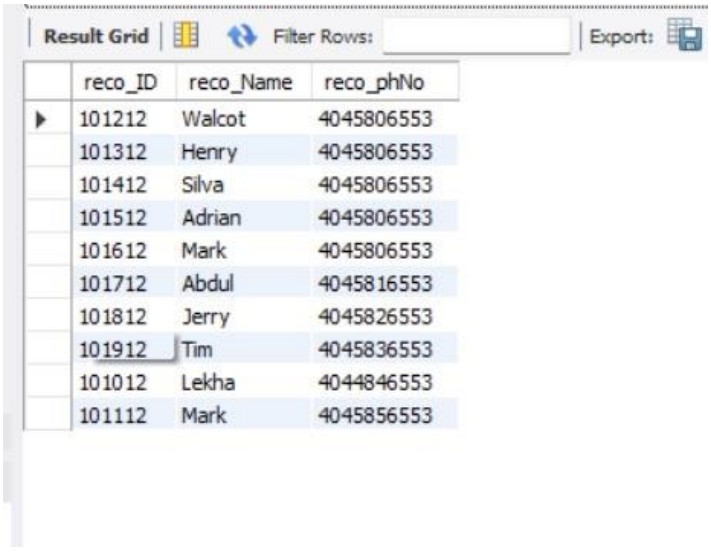


hosp_ID	hosp_name	City_ID	M_id
1	MayoClinic	1100	101
2	CleavelandClinic	1200	103
3	NYU	1300	103
4	Baylor	1400	104
5	Charlton	1800	103
6	Greenoaks	1300	106
7	Forestpark	1300	102
8	Parkland	1200	106
9	Pinecreek	1500	109
10	WalnutHill	1700	105
NULL	NULL	NULL	NULL

Hospital_info_1:

hosp_ID	hosp_name	hosp_needed_Bgrp	hosp_needed_qnty
1	MayoClinic	A+	20
1	MayoClinic	AB+	0
1	MayoClinic	A-	40
1	MayoClinic	B-	10
1	MayoClinic	AB-	20
2	CleavelandClinic	A+	40
2	CleavelandClinic	AB+	20
2	CleavelandClinic	A-	10
2	CleavelandClinic	B-	30
2	CleavelandClinic	B+	0
2	CleavelandClinic	AB-	10
3	NYU	A+	0
3	NYU	AB+	0
3	NYU	A-	0
3	NYU	B-	20
3	NYU	B+	10
3	NYU	AB-	0
4	Baylor	A+	10
5	Charlton	B+	30
4	Baylor	A-	40
7	Forestpark	B-	40
8	Parkland	B+	10
9	Pinecreek	AB-	20

Recording_staff:



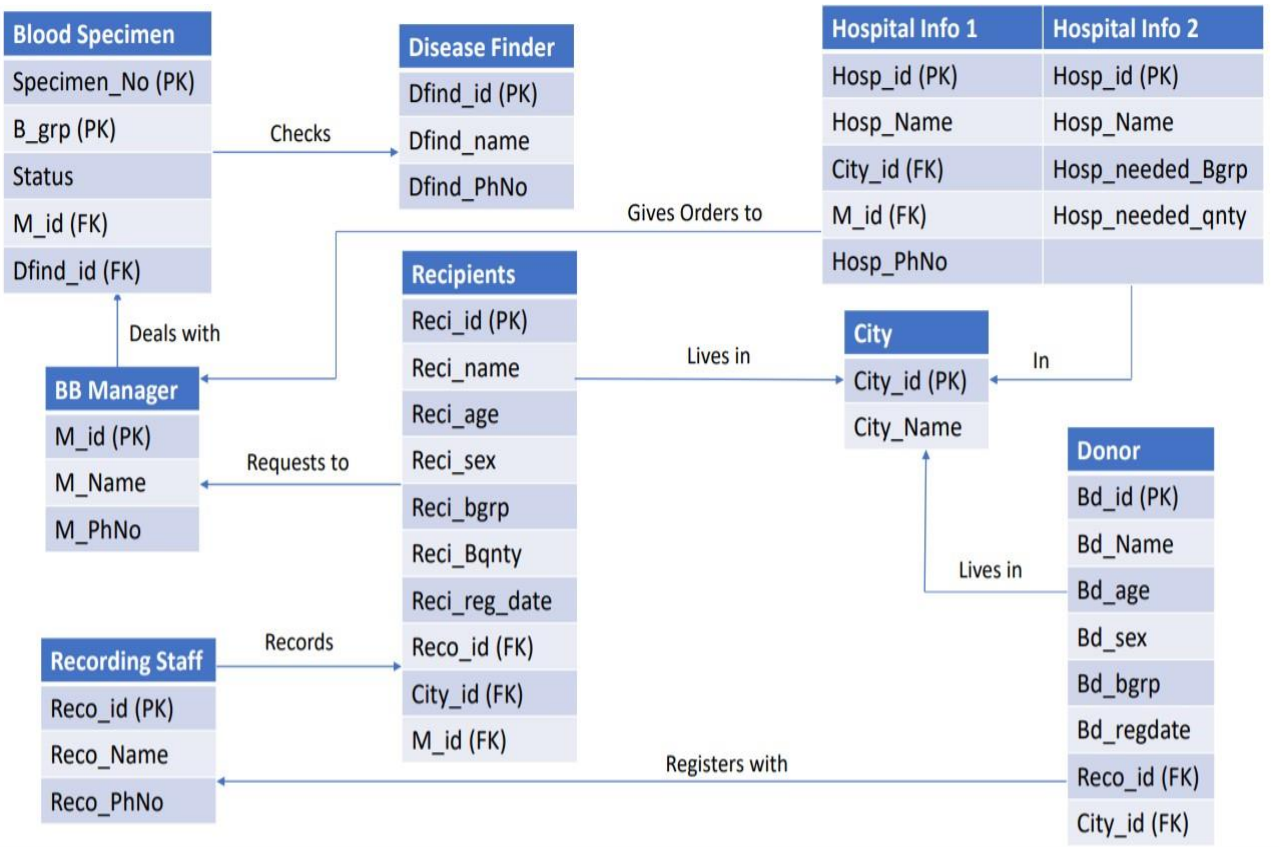
reco_ID	reco_Name	reco_phNo
101212	Walcot	4045806553
101312	Henry	4045806553
101412	Silva	4045806553
101512	Adrian	4045806553
101612	Mark	4045806553
101712	Abdul	4045816553
101812	Jerry	4045826553
101912	Tim	4045836553
101012	Lekha	4044846553
101112	Mark	4045856553

Recipient:

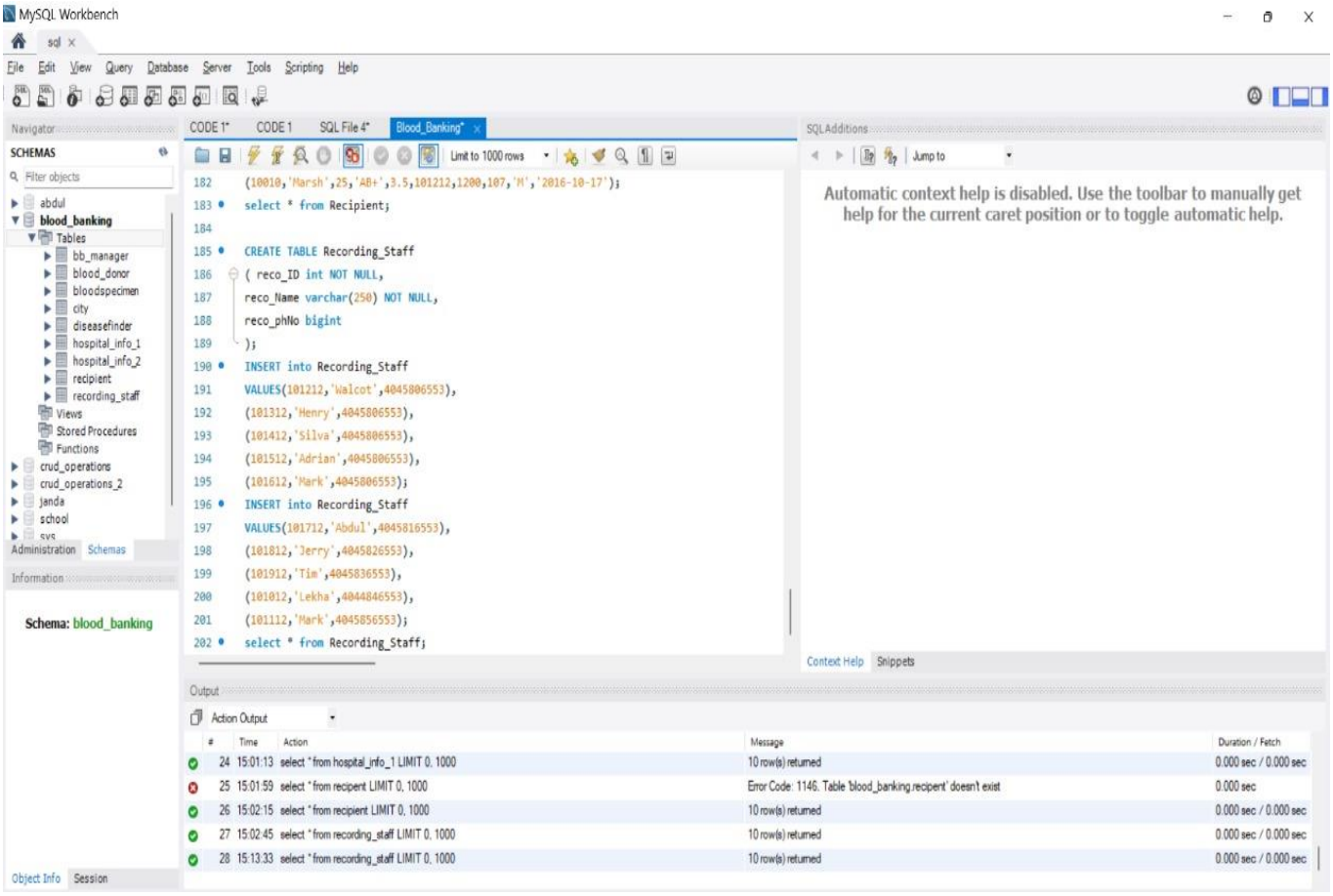
Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

reci_ID	reci_name	reci_age	reci_Brgp	reci_Bqnty	reco_ID	City_ID	M_id	reci_sex	reg_date
10001	Mark	25	B+	1.5	101212	1100	101	M	2015-12-17
10002	Dan	60	A+	1	101312	1100	102	M	2015-12-16
10003	Steve	35	AB+	0.5	101312	1200	102	M	2015-10-17
10004	Parker	66	B+	1	101212	1300	104	M	2016-11-17
10005	Jason	53	B-	1	101412	1400	105	M	2015-04-17
10006	Preetham	45	O+	1.5	101512	1500	105	M	2015-12-17
10007	Swetha	22	AB-	1	101212	1500	101	F	2015-05-17
10008	Swathi	25	B+	2	101412	1300	103	F	2015-12-14
10009	Lance	30	A+	1.5	101312	1100	104	M	2015-02-16
10010	Marsh	25	AB+	3.5	101212	1200	107	M	2016-10-17
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

ER DIAGRAM AFTER NORMALIZATION



SQL IMPLEMENTATION



CREATE TABLE BB_Manager
(M_id int NOT NULL,
mName varchar(250) NOT NULL,
m_phNo bigint
);
INSERT into BB_Manager
VALUES(102,'Jack', 4693959671),
(103,'Peter', 4693959601),
(104,'Mark', 4693959677),
(105,'Jason', 4693957671);
INSERT into BB_Manager
VALUES(106,'Steve', 4694959671),

```
(107,'Jason', 4695959671),
(108,'Stella', 4663959671),
(109,'Monika', 4673959671),
(110,'John', 4693859671);
select * from BB_Manager ;

CREATE TABLE Blood_Donor
( bd_ID int NOT NULL,
bd_name varchar(250) NOT NULL,
bd_age varchar(250),
bd_sex varchar(250),
bd_Bgroup varchar(10),
bd_reg_date date,
reco_ID int NOT NULL,
City_ID int NOT NULL -- CONSTRAINT bdID_pk PRIMARY KEY (bd_ID)
);
INSERT into Blood_Donor
VALUES(150221,'Mark',25,'M','B+','2015-12-17',101212,1100),
(160011,'Abdul',35,'F','A+','2016-11-22',101212,1100),
(160101,'Smith',22,'M','O+','2016-01-04',101312,1200),
(150011,'Pat',29,'M','O+','2015-07-19',101412,1300),
(150021,'Shyam',42,'F','A-','2015-12-24',101412,1300),
(150121,'Dan',44,'M','AB+','2015-08-28',101212,1200),
(160031,'Mike',33,'F','AB-','2016-02-06',101212,1400),
(160301,'Elisa',31,'F','AB+','2016-09-10',101312,1200),
(160091,'Carrol',24,'M','B-','2016-10-15',101312,1500),
(160401,'Mark',29,'M','O-','2016-12-17',101212,1200);
select * from Blood_Donor;
CREATE TABLE BloodSpecimen
```

```
( specimen_number int primary key ,
b_group varchar(10) NOT NULL,
status int,
dfind_ID int NOT NULL,
M_id int NOT NULL
);
INSERT into BloodSpecimen
VALUES(1001, 'B+', 1,11,101),
(1002, 'O+', 1,12,102),
(1003, 'AB+', 1,11,102),
(1004, 'O-', 1,13,103),
(1005, 'A+', 0,14,101),
(1006, 'A-', 1,13,104),
(1007, 'AB-', 1,15,104),
(1008, 'AB-', 0,11,105),
(1009, 'B+', 1,13,105),
(1010, 'O+', 0,12,105),
(1011, 'O+', 1,13,103),
(1012, 'O-', 1,14,102),
(1013, 'B-', 1,14,102),
(1014, 'AB+', 0,15,101);
use blood_banking;
select * from bloodspecimen;

CREATE TABLE City
( City_ID int primary key ,
 City_name varchar(250) NOT NULL
);
INSERT into City
```

```
VALUES(1200,'Austin'),
(1300,'Irving'),
(1400,'Houston'),
(1500,'Richardson');
INSERT into City
VALUES(1600,'Plano'),
(1700,'Frisco'),
(1800,'Arlington'),
(1900,'San Antonio'),
(2000,'Tyler');

select * from City;

CREATE TABLE DiseaseFinder
( dfind_ID int primary key,
 dfind_name varchar(250) NOT NULL,
 dfind_PhNo bigint
);
INSERT into DiseaseFinder
VALUES(11,'Peter',4693804223),
(12,'Park',4693804223),
(13,'Jerry',4693804223),
(14,'Mark',4693804223),
(15,'Monika',4693804223);
INSERT into DiseaseFinder
VALUES(16,'Ram',4693804123),
(17,'Swathi',4693804223),
(18,'Gautham',4693804323),
(19,'Ashwin',4693804423),
```

```
(20,'Yash',4693804523);
select * from DiseaseFinder;

CREATE TABLE Hospital_Info_1
( hosp_ID int primary key,
 hosp_name varchar(250) NOT NULL,
 City_ID int NOT NULL,
 M_id int NOT NULL
);
INSERT into Hospital_Info_1
VALUES(1,'MayoClinic',1100,101),
(2,'CleavelandClinic',1200,103),
(3,'NYU',1300,103);
INSERT into Hospital_Info_1
VALUES(4,'Baylor',1400,104),
(5,'Charlton',1800,103),
(6,'Greenoaks',1300,106),
(7,'Forestpark',1300,102),
(8,'Parkland',1200,106),
(9,'Pinecreek',1500,109),
(10,'WalnutHill',1700,105);
select * from Hospital_Info_1;

CREATE TABLE Hospital_Info_2
( hosp_ID int not null,
 hosp_name varchar(250) NOT NULL,
 hosp_needed_Bgrp varchar(10),
 hosp_needed_qnty int);
```

```
INSERT into Hospital_Info_2
VALUES(1,'MayoClinic','A+',20),
(1,'MayoClinic','AB+',0),
(1,'MayoClinic','A-',40),
(1,'MayoClinic','B-',10),
(1,'MayoClinic','AB-',20),
(2,'CleavelandClinic','A+',40),
(2,'CleavelandClinic','AB+',20),
(2,'CleavelandClinic','A-',10),
(2,'CleavelandClinic','B-',30),
(2,'CleavelandClinic','B+',0),
(2,'CleavelandClinic','AB-',10),
(3,'NYU','A+',0),
(3,'NYU','AB+',0),
(3,'NYU','A-',0),
(3,'NYU','B-',20),
(3,'NYU','B+',10),
(3,'NYU','AB-',0);
INSERT into Hospital_Info_2
VALUES(4,'Baylor','A+',10),
(5,'Charlton','B+',30),
(4,'Baylor','A-',40),
(7,'Forestpark','B-',40),
(8,'Parkland','B+',10),
(9,'Pinecreek','AB-',20);
select * from Hospital_Info_2;
drop table Hospital_Info_2;
select * from Hospital_Info_2;
```

```
use blood_banking;
CREATE TABLE Recipient
( reci_ID int primary key,
reci_name varchar(255) NOT NULL,
reci_age varchar(255),
reci_Brgp varchar(255),
reci_Bqnty float,
reco_ID int NOT NULL,
City_ID int NOT NULL,
M_id int NOT NULL,
reci_sex varchar(259),
reg_date date
);

INSERT into Recipient
VALUES(10001,'Mark',25,'B+',1.5,101212,1100,101,'M','2015-12-17'),
(10002,'Dan',60,'A+',1,101312,1100,102,'M','2015-12-16'),
(10003,'Steve',35,'AB+',0.5,101312,1200,102,'M','2015-10-17'),
(10004,'Parker',66,'B+',1,101212,1300,104,'M','2016-11-17'),
(10005,'Jason',53,'B-',1,101412,1400,105,'M','2015-04-17'),
(10006,'Preetham',45,'O+',1.5,101512,1500,105,'M','2015-12-17'),
(10007,'Swetha',22,'AB-',1,101212,1500,101,'F','2015-05-17'),
(10008,'Swathi',25,'B+',2,101412,1300,103,'F','2015-12-14'),
(10009,'Lance',30,'A+',1.5,101312,1100,104,'M','2015-02-16'),
(10010,'Marsh',25,'AB+',3.5,101212,1200,107,'M','2016-10-17');
select * from Recipient;

CREATE TABLE Recording_Staff
( reco_ID int NOT NULL,
```

```
reco_Name varchar(250) NOT NULL,
reco_phNo bigint
);
INSERT into Recording_Staff
VALUES(101212,'Walcot',4045806553),
(101312,'Henry',4045806553),
(101412,'Silva',4045806553),
(101512,'Adrian',4045806553),
(101612,'Mark',4045806553);
INSERT into Recording_Staff
VALUES(101712,'Abdul',4045816553),
(101812,'Jerry',4045826553),
(101912,'Tim',4045836553),
(101012,'Lekha',4044846553),
(101112,'Mark',4045856553);
select * from Recording_Staff;
```

SAMPLE SQL QUERIES

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

```
CREATE VIEW Recipients_Donors_SameBGroup AS
SELECT
   Blood_Donor.bd_name AS Donor_Name,
   Recipient.reci_name AS Recipient_Name,
   Blood_Donor.bd_Bgroup AS Blood_Group,
   Blood_Donor.bd_reg_date AS Registration_Date
FROM
```

```
   Blood_Donor
INNER JOIN
   Recipient
ON
   Blood_Donor.bd_Bgroup = Recipient.reci_Brgp
       AND Blood_Donor.bd_reg_date = Recipient.reg_date;
    SELECT * FROM Recipients_Donors_SameBGroup;
```

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

```
   SELECT
      BloodSpecimen.specimen_number,
      BloodSpecimen.b_group
   FROM
      BloodSpecimen
   INNER JOIN
      DiseaseFinder
   ON
      BloodSpecimen.dfind_ID = DiseaseFinder.dfind_ID
   WHERE
      DiseaseFinder.dfind_name = 'Mark'
          AND BloodSpecimen.status = 1;
```

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.mName AS Manager_Name,
   BB_Manager.M_id AS Manager_ID,
```

```
    Recipient.reci_name AS Recipient_Name,
    Recipient.reci_Brgp AS Blood_Group,
    BloodSpecimen.specimen_number AS Specimen_Number
FROM
    BB_Manager
INNER JOIN
    Recipient
ON
    BB_Manager.M_id = Recipient.M_id
INNER JOIN
    BloodSpecimen
ON
    BB_Manager.M_id = BloodSpecimen.M_id
    AND Recipient.reci_Brgp = BloodSpecimen.b_group
WHERE
    BloodSpecimen.status = 1; -- Only pure blood specimens
```

CONCLUSION

The constraints of the current system were adequately addressed by our project. I created a well-structured database administration system, which is a difficult task in this day and age. I used MySQL WorkBench to create a database for a blood bank. In order to create an effective Entity Relationship Diagram (ERD), we investigated a variety of features and blood bank processes during the design phase prior to database implementation. This allowed us to identify the necessary entities, properties, and relationships between the entities. I constructed our ERD after reviewing all of the criteria, normalized the data, and transformed the ERD to a relational model. I made the tables for our database using MySQL WorkBench and added some example values to them. Finally, I have executed sample queries on our database to check its performance to retrieve useful information accurately and speedily.