DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DATABASE MANAGEMENT SYSTEM - CO202

SUBMITTED TO
Mr. Rohit Beniwal

SUBMITTED BY
Vatsalya Kumar Mishra
(2K21/CO/509)
Vicky Kumar
(2K21/CO/513)

BLOOD BANK MANAGEMENT SYSTEM

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 type of blood in emergency situations from either blood bank or even blood donors.

Current system uses a grapevine communication for finding blood in cases of emergency, may it be by a donor or blood bank. The intentions of proposing such a system are to abolish the panic caused during an emergency due to unavailability of blood.

INTRODUCTION

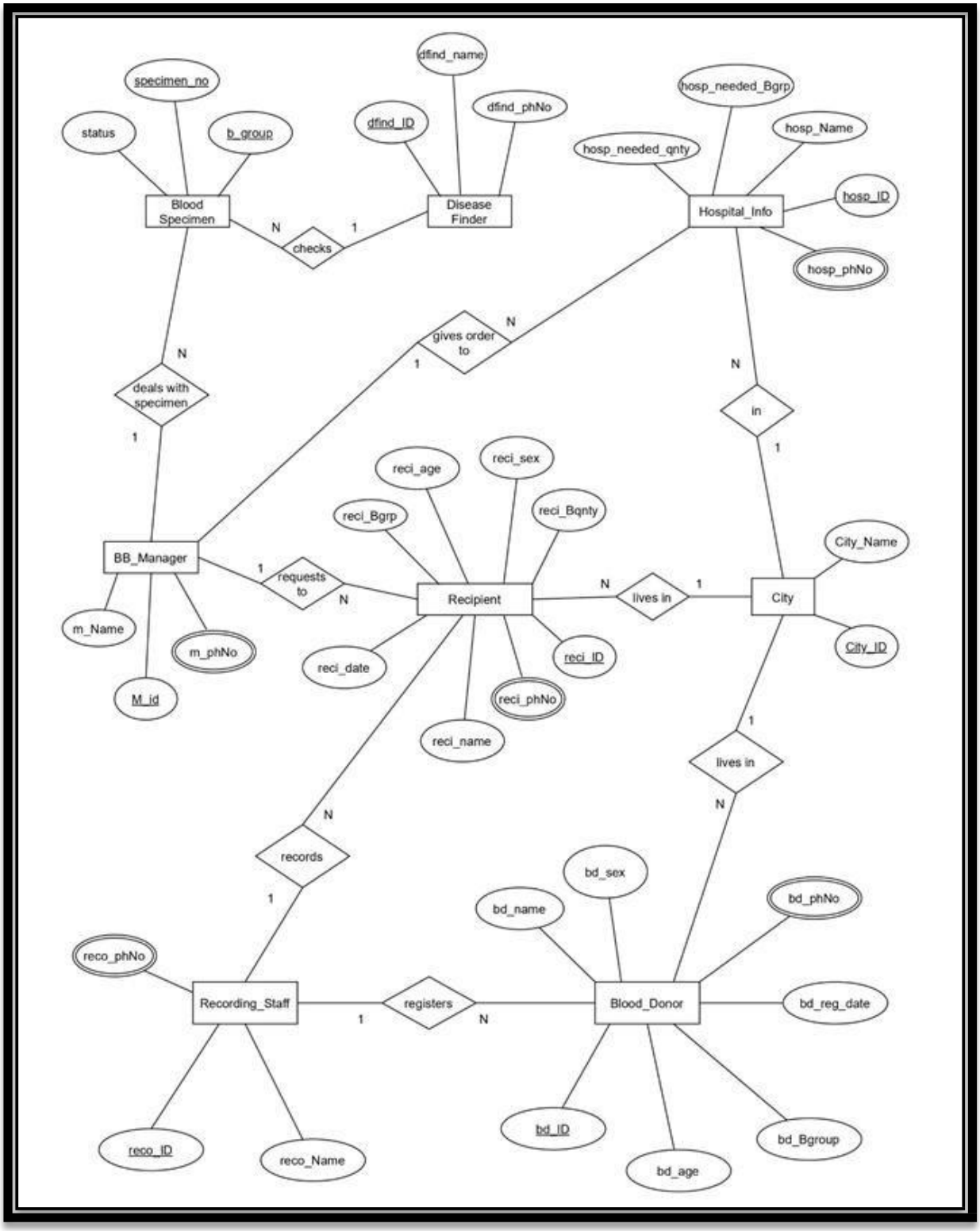
Blood banks collect, store and provide collected blood to the patients who are in need of blood. The people who donate blood are called donors. The banks then group the blood which they receive according to the blood groups. They also make sure that the blood is not contaminated. The main mission of 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 a tough job. The existing system will not satisfy the need of maintaining quality blood and keep 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 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 man power is not required.

ER DIAGRAM


dfind_name
specimen_no
hosp_needed_Bgrp
status
b_group
dfind_ID
dfind_phNo
hosp_needed_qnty
hosp_Name
Blood Specimen
N
checks
1
Disease Finder
Hospital_Info
hosp_ID
hosp_phNo
N
gives order to
N
1
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
bd_ID
bd_Bgroup
reco_ID
reco_Name
bd_age

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 receives blood from blood bank, when blood is given to a recipient a recipient ID (reci_ID) is generated and used as primary key for the recipient entity to identify 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 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 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 entity has reco_ID which is primary key along with recorder's name and recorder'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 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 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 checks the 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 entity.

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 together makes the primary key. We will store hospital name and the blood quantity required 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 identification number (City_ID) will be used as primary key to identify the information about the city. Along with ID city names will also be stored under this entity.

RELATIONSHIP BETWEEN ENTITIES

1. City and Hospital_Info:
Relationship = “in”
Type of relation = 1 to many
Explanation = A city can have many hospitals in it. One hospital willbelong in one city.

2. 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.

3. City and Recipient:
Relationship = “lives in”
Type of relation = 1 to many
Explanation = In a city, many recipients can live. One recipient willbelong to one city.

4. 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.

5. 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.

6. 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.

7. BB_Manager and Blood Specimen:
Relationship = “deals 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.

8. 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.

9. 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 is checked by one disease finder.

RELATIONAL SCHEMAS

Donor Table:

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

Recipient Table:

- 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:

- 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:

- 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:

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

Disease Finder Table:

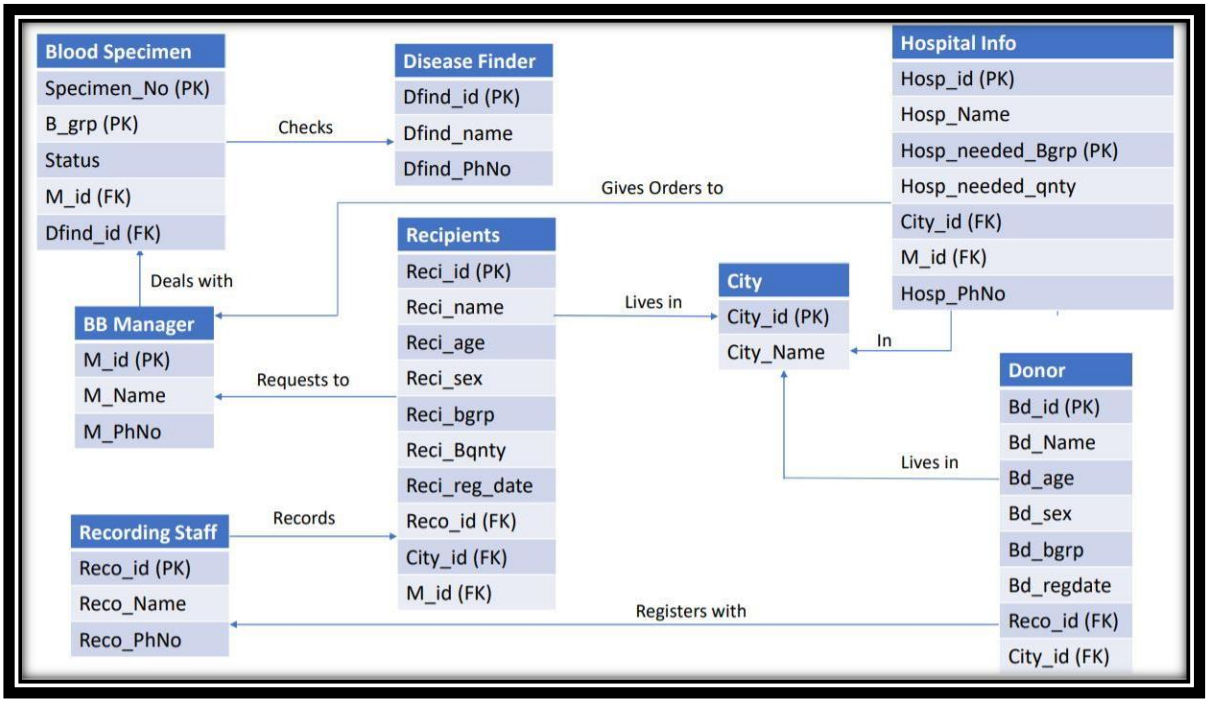
- The relationship with Disease finder and Blood Specimen is of 1 to many. Therefore, the primary key of Disease finder is used as a foreign key in Blood Specimen.

Blood Bank Manager Table:

- 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 foreign key in Blood Specimen, Recipient and Hospital info.

Hospital info Table:

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



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.

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_name do 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)

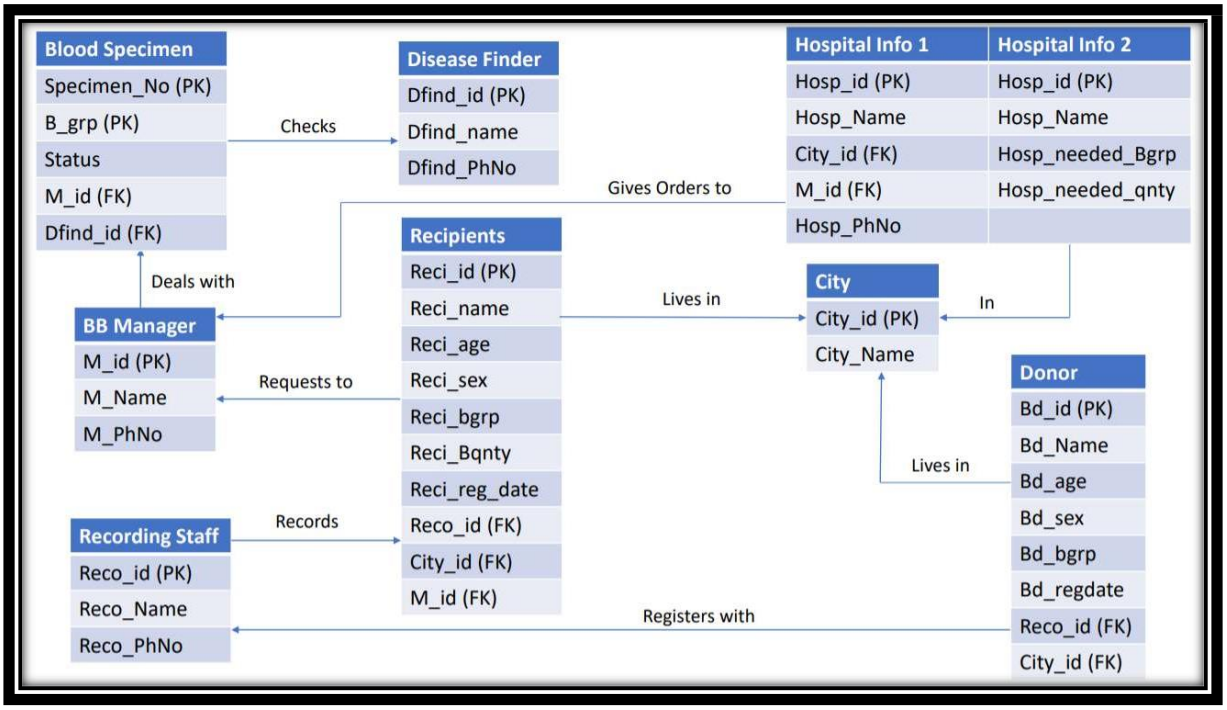
The table is in first normal form.
Since every non-primary key attribute is not fully functionally dependent on the primary key of the table, this table is not in second normal form. Hence, we have to split the table.

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.

RELATIONAL SCHEMA AFTER NORMALIZATION



SQL IMPLEMENTATION

The implementation on SQL Server is given below :-

-- Creation of 'BB_Manager' table

```
-- Creation of 'BB_Manager' table
CREATE TABLE BB_Manager (
    M_id int NOT NULL PRIMARY KEY,
    mName varchar(100) NOT NULL,
    m_phNo bigint
);

-- Value insertion
INSERT into BB_Manager VALUES
(101, 'Vatsalya', 9693959671),
(102, 'Vicky', 9693959672),
(103, 'Light', 9693959673),
(104, 'Eren', 9693959674),
(105, 'Mikasa', 9693959675),
(106, 'Goku', 9693959676),
(107, 'Itachi', 9693959677),
(108, 'Naruto', 9693959678),
(109, 'Luffy', 9693959679),
(110, 'Levi', 9693959680);

-- Display table
select * from BB_Manager;
```

```sql
CREATE DATABASE BLOODBANK;

USE BLOODBANK;

CREATE TABLE BB_Manager
( M_id int NOT NULL PRIMARY KEY,
 mName varchar(100) NOT NULL,
 m_phNo bigint
);

INSERT into BB_Manager VALUES
(101,'Vatsalya', 9693959671),
(102,'Vicky', 9693959672),
(103,'Light', 9693959673),
(104,'Eren', 9693959674),
(105,'Mikasa', 9693959675),
(106,'Goku', 9693959676),
(107,'Itachi', 9693959677),
(108,'Naruto', 9693959678),
(109,'Luffy', 9693959679),
(110,'Levi', 9693959680);

select * from BB_Manager;
```

Result Grid | Filter Rows:

M_id	mName	m_phNo
101	Vatsalya	9693959671
102	Vicky	9693959672
103	Light	9693959673
104	Eren	9693959674
105	Mikasa	9693959675
106	Goku	9693959676
107	Itachi	9693959677
108	Naruto	9693959678
109	Luffy	9693959679
110	Levi	9693959680
NULL	NULL	NULL

BB_Manager 5

-- Creation of 'Blood_Donor' table

```sql
-- Creation of 'Blood_Donor' table
CREATE TABLE Blood_Donor (
    bd_ID int NOT NULL PRIMARY KEY,
    bd_name varchar(100) NOT NULL,
    bd_age varchar(100),
    bd_sex varchar(100),
    bd_Bgroup varchar(10),
    bd_reg_date date,
    reco_ID int NOT NULL,
    City_ID int NOT NULL,
    FOREIGN KEY(reco_ID) REFERENCES Recording_Staff(reco_ID),
    FOREIGN KEY(City_ID) REFERENCES City(City_ID)
);

-- Value insertion
INSERT into Blood_Donor VALUES
(150011,'Steven',25,'M','O+','2015-07-19',101412,1100),
(150012,'Tony', 35,'M','A-','2015-12-24',101412,1100),
(150013,'Bruce',22,'M','AB+','2015-08-28',101212,1200),
(150014,'Natasha',29,'M','B+','2015-12-17',101212,1300),
(150015,'Hermoine',42,'M','A+','2016-11-22',101212,1300),
(150016,'Harry',44,'F','AB-','2016-02-06',101212,1200),
(150017,'Sherlock',33,'M','B-','2016-10-15',101312,1400),
(150018,'Logan',31,'F','O+','2016-01-04',101312,1200),
(150019,'Peter',24,'F','AB+','2016-09-10',101312,1500),
(150020,'Odinson',29,'M','O-','2016-12-17',101212,1200);

-- Display table
select * from Blood_Donor;
```

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

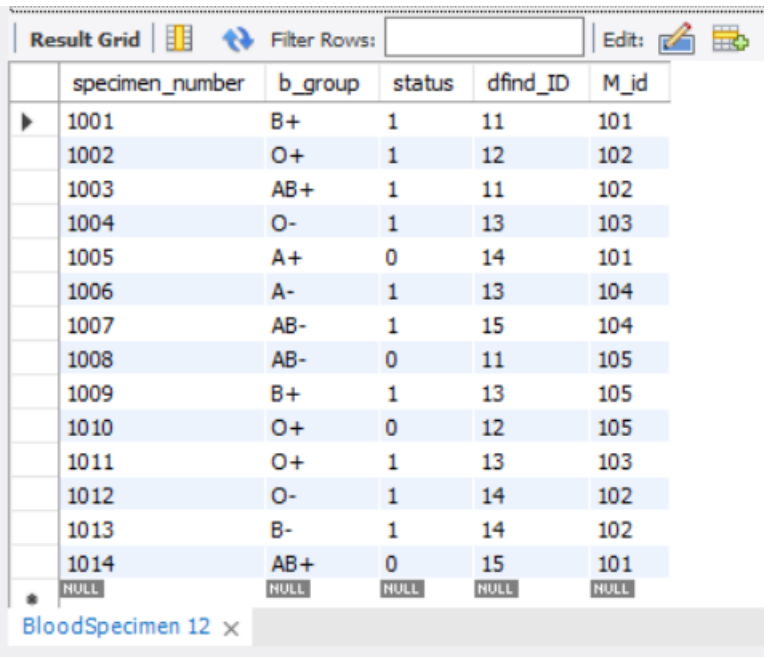
bd_ID	bd_name	bd_age	bd_sex	bd_Bgroup	bd_reg_date	reco_ID	City_ID
150011	Steven	25	M	O+	2015-07-19	101412	1100
150012	Tony	35	M	A-	2015-12-24	101412	1100
150013	Bruce	22	M	AB+	2015-08-28	101212	1200
150014	Natasha	29	M	B+	2015-12-17	101212	1300
150015	Hermoine	42	M	A+	2016-11-22	101212	1300
150016	Harry	44	F	AB-	2016-02-06	101212	1200
150017	Sherlock	33	M	B-	2016-10-15	101312	1400
150018	Logan	31	F	O+	2016-01-04	101312	1200
150019	Peter	24	F	AB+	2016-09-10	101312	1500
150020	Odinson	29	M	O-	2016-12-17	101212	1200
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Blood_Donor10 ×


```
64  CREATE TABLE Blood_Donor
65  ( bd_ID int NOT NULL PRIMARY KEY,
66   bd_name varchar(100) NOT NULL,
67   bd_age varchar(100),
68   bd_sex varchar(100),
69   bd_Bgroup varchar(10),
70   bd_reg_date date,
71   reco_ID int NOT NULL,
72   City_ID int NOT NULL,
73   FOREIGN KEY(reco_ID) REFERENCES Recording_Staff(reco_ID),
74   FOREIGN KEY(City_ID) REFERENCES City(City_ID)
75  );
76
77  INSERT into Blood_Donor VALUES
78  (150011,'Steven',25,'M','O+','2015-07-19',101412,1100),
79  (150012,'Tony',35,'M','A-','2015-12-24',101412,1100),
80  (150013,'Bruce',22,'M','AB+','2015-08-28',101212,1200),
81  (150014,'Natasha',29,'M','B+','2015-12-17',101212,1300),
82  (150015,'Hermoine',42,'M','A+','2016-11-22',101212,1300),
83  (150016,'Harry',44,'F','AB-','2016-02-06',101212,1200),
84  (150017,'Sherlock',33,'M','B-','2016-10-15',101312,1400),
85  (150018,'Logan',31,'F','O+','2016-01-04',101312,1200),
86  (150019,'Peter',24,'F','AB+','2016-09-10',101312,1500),
87  (150020,'Odinson',29,'M','O-','2016-12-17',101212,1200);
88
89  select * from Blood_Donor;
```

-- Creation of 'BloodSpecimen' table

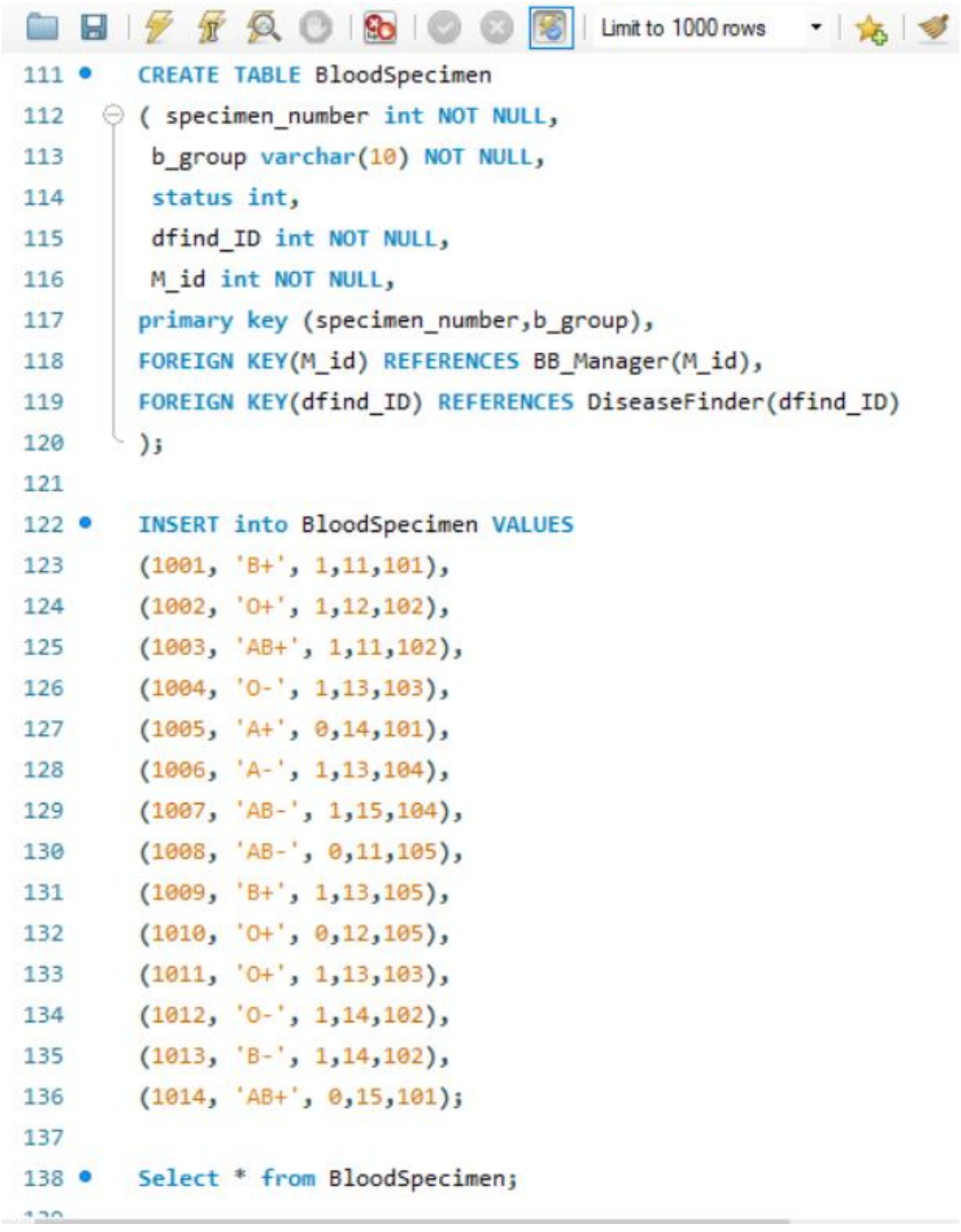
```
-- Creation of 'BloodSpecimen' table
CREATE TABLE BloodSpecimen (
    specimen_number int NOT NULL,
    b_group varchar(10) NOT NULL,
    status int,
    dfind_ID int NOT NULL,
    M_id int NOT NULL,
    primary key (specimen_number, b_group),
    FOREIGN KEY(M_id) REFERENCES BB_Manager(M_id),
    FOREIGN KEY(dfind_ID) REFERENCES DiseaseFinder(dfind_ID)
);

-- Value insertion
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);

-- Display table
Select * from BloodSpecimen;
```



Result Grid | Filter Rows: | Edit:

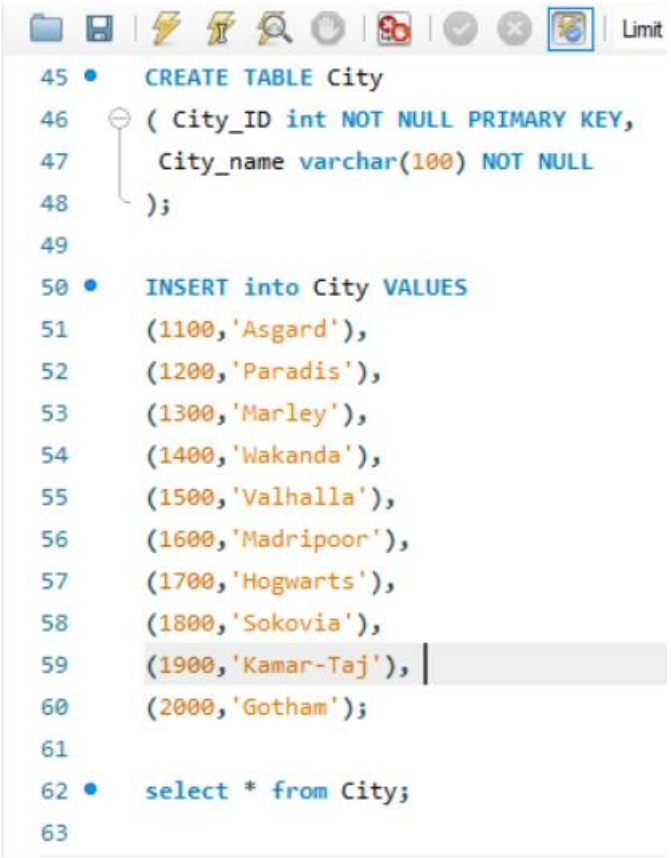
specimen_number	b_group	status	dfind_ID	M_id
1001	B+	1	11	101
1002	O+	1	12	102
1003	AB+	1	11	102
1004	O-	1	13	103
1005	A+	0	14	101
1006	A-	1	13	104
1007	AB-	1	15	104
1008	AB-	0	11	105
1009	B+	1	13	105
1010	O+	0	12	105
1011	O+	1	13	103
1012	O-	1	14	102
1013	B-	1	14	102
1014	AB+	0	15	101
NULL	NULL	NULL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

BloodSpecimen 12


Limit to 1000 rows
CREATE TABLE BloodSpecimen
(specimen_number int NOT NULL,
b_group varchar(10) NOT NULL,
status int,
dfind_ID int NOT NULL,
M_id int NOT NULL,
primary key (specimen_number,b_group),
FOREIGN KEY(M_id) REFERENCES BB_Manager(M_id),
FOREIGN KEY(dfind_ID) REFERENCES DiseaseFinder(dfind_ID)
);
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);
Select * from BloodSpecimen;

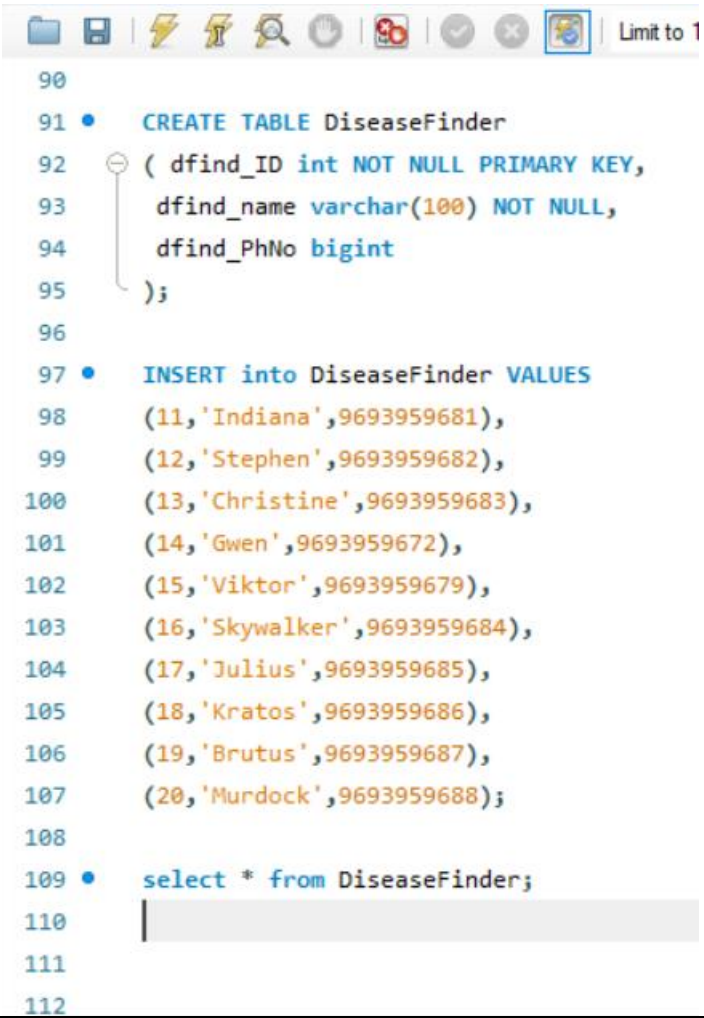
-- Creation of 'City' table

```sql
-- Creation of 'City' table
CREATE TABLE City (
    City_ID int NOT NULL PRIMARY KEY,
    City_name varchar(100) NOT NULL
);

-- Value insertion
INSERT into City
VALUES (1100, 'Asgard'),
    (1200, 'Paradis'),
    (1300, 'Marley'),
    (1400, 'Wakanda'),
    (1500, 'Valhalla'),
    (1600, 'Madripoor'),
    (1700, 'Hogwarts'),
    (1800, 'Sokovia'),
    (1900, 'Kamar-Taj'),
    (2000, 'Gotham');

-- Display table
select * from City;
```

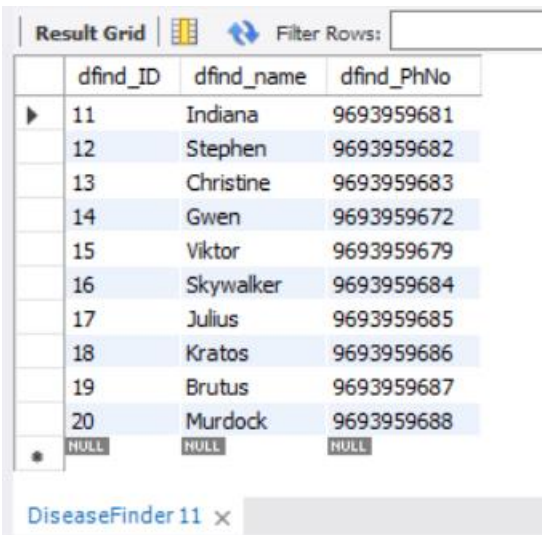


-- Creation of 'DiseaseFinder' table

```sql
-- Creation of 'DiseaseFinder' table
CREATE TABLE DiseaseFinder (
    dfind_ID int NOT NULL PRIMARY KEY,
    dfind_name varchar(100) NOT NULL,
    dfind_PhNo bigint
);

-- Value insertion
INSERT into DiseaseFinder
VALUES (11, 'Indiana', 9693959681),
    (12, 'Stephen', 9693959682),
    (13, 'Christine', 9693959683),
    (14, 'Gwen', 9693959672),
    (15, 'Viktor', 9693959679),
    (16, 'Skywalker', 9693959684),
    (17, 'Julius', 9693959685),
    (18, 'Kratos', 9693959686),
    (19, 'Brutus', 9693959687),
    (20, 'Murdock', 9693959688);

-- Display table
select * from DiseaseFinder;
```



```sql
CREATE TABLE DiseaseFinder
( dfind_ID int NOT NULL PRIMARY KEY,
 dfind_name varchar(100) NOT NULL,
 dfind_PhNo bigint
);

INSERT into DiseaseFinder VALUES
(11,'Indiana',9693959681),
(12,'Stephen',9693959682),
(13,'Christine',9693959683),
(14,'Gwen',9693959672),
(15,'Viktor',9693959679),
(16,'Skywalker',9693959684),
(17,'Julius',9693959685),
(18,'Kratos',9693959686),
(19,'Brutus',9693959687),
(20,'Murdock',9693959688);

select * from DiseaseFinder;
```



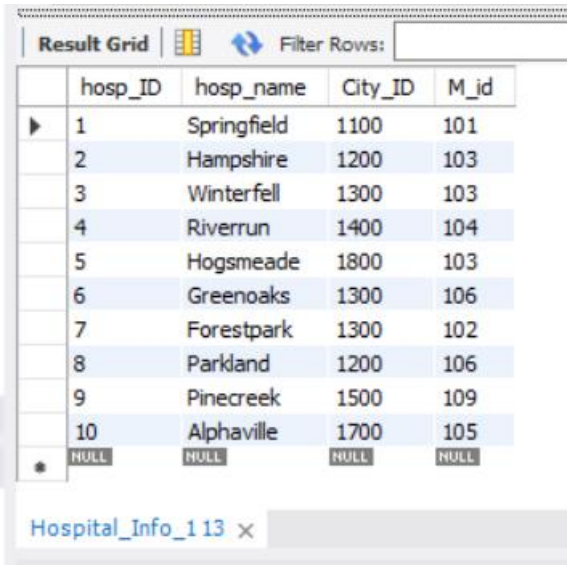
Result Grid | Filter Rows:

dfind_ID	dfind_name	dfind_PhNo
11	Indiana	9693959681
12	Stephen	9693959682
13	Christine	9693959683
14	Gwen	9693959672
15	Viktor	9693959679
16	Skywalker	9693959684
17	Julius	9693959685
18	Kratos	9693959686
19	Brutus	9693959687
20	Murdock	9693959688
NULL	NULL	NULL

DiseaseFinder 11

-- Creation of 'Hospital_Info_1' table

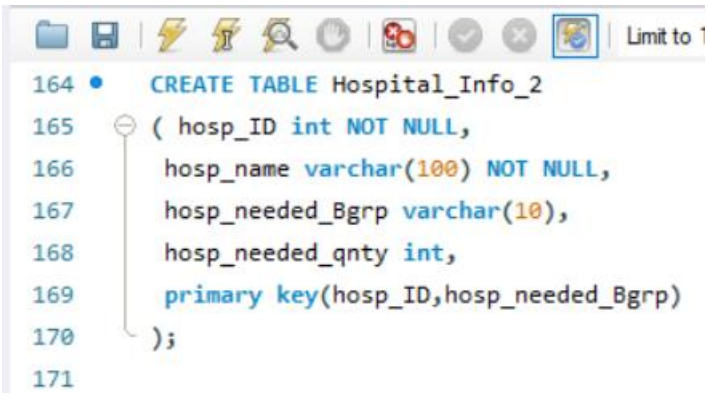
```sql
-- Creation of 'Hospital_Info_1' table
CREATE TABLE Hospital_Info_1 (
    hosp_ID int NOT NULL,
    hosp_name varchar(100) NOT NULL,
    City_ID int NOT NULL,
    M_id int NOT NULL,
    primary key(hosp_ID),
    FOREIGN KEY(M_id) REFERENCES BB_Manager(M_id),
    FOREIGN KEY(City_ID) REFERENCES City(City_ID)
);

-- Value insertion
INSERT into Hospital_Info_1
VALUES (1, 'Springfield', 1100, 101),
    (2, 'Hampshire', 1200, 103),
    (3, 'Winterfell', 1300, 103),
    (4, 'Riverrun', 1400, 104),
    (5, 'Hogsmeade', 1800, 103),
    (6, 'Greenoaks', 1300, 106),
    (7, 'Forestpark', 1300, 102),
    (8, 'Parkland', 1200, 106),
    (9, 'Pinecreek', 1500, 109),
    (10, 'Alphaville', 1700, 105);

-- Display table
select * from Hospital_Info_1;
```




```
140 •  CREATE TABLE Hospital_Info_1
141    ( hosp_ID int NOT NULL,
142     hosp_name varchar(100) NOT NULL,
143     City_ID int NOT NULL,
144     M_id int NOT NULL,
145     primary key(hosp_ID),
146     FOREIGN KEY(M_id) REFERENCES BB_Manager(M_id),
147     FOREIGN KEY(City_ID) REFERENCES City(City_ID)
148    );
149
150 •  INSERT into Hospital_Info_1 VALUES
151    (1,'Springfield',1100,101),
152    (2,'Hampshire',1200,103),
153    (3,'Winterfell',1300,103),
154    (4,'Riverrun',1400,104),
155    (5,'Hogsmeade',1800,103),
156    (6,'Greenoaks',1300,106),
157    (7,'Forestpark',1300,102),
158    (8,'Parkland',1200,106),
159    (9,'Pinecreek',1500,109),
160    (10,'Alphaville',1700,105);
161
162 •  select * from Hospital_Info_1;
163
```

-- Creation of 'Hospital_Info_2' table

```sql
-- Creation of 'Hospital_Info_2' table
CREATE TABLE Hospital_Info_2 (
    hosp_ID int NOT NULL,
    hosp_name varchar(100) NOT NULL,
    hosp_needed_Bgrp varchar(10),
    hosp_needed_qnty int,
    primary key(hosp_ID, hosp_needed_Bgrp)
);

-- Value insertion
INSERT into Hospital_Info_2
VALUES (1, 'Springfield', 'A+', 20),
    (1, 'Springfield', 'A-', 0),
    (1, 'Springfield', 'AB+', 40),
    (1, 'Springfield', 'AB-', 10),
    (1, 'Springfield', 'B-', 20),
    (2, 'Hampshire', 'A+', 40),
    (2, 'Hampshire', 'AB+', 20),
    (2, 'Hampshire', 'A-', 10),
    (2, 'Hampshire', 'B-', 30),
    (2, 'Hampshire', 'B+', 0),
    (2, 'Hampshire', 'AB-', 10),
    (3, 'Winterfell', 'A+', 0),
    (3, 'Winterfell', 'AB+', 0),
    (3, 'Winterfell', 'A-', 0),
    (3, 'Winterfell', 'B-', 20),
    (3, 'Winterfell', 'B+', 10),
    (3, 'Winterfell', 'AB-', 0),
    (4, 'Riverrun', 'A+', 10),
    (4, 'Riverrun', 'A-', 40),
    (7, 'Forestpark', 'B-', 40),
    (8, 'Parkland', 'B+', 10),
    (9, 'Pinecreek', 'AB-', 20);

-- Display table
select * from Hospital_Info_2;
```



```
174 •   INSERT into Hospital_Info_2 VALUES
175     (1,'Springfield','A+',20),
176     (1,'Springfield','A-',0),
177     (1,'Springfield','AB+',40),
178     (1,'Springfield','AB-',10),
179     (1,'Springfield','B-',20),
180     (2,'Hampshire','A+',40),
181     (2,'Hampshire','AB+',20),
182     (2,'Hampshire','A-',10),
183     (2,'Hampshire','B-',30),
184     (2,'Hampshire','B+',0),
185     (2,'Hampshire','AB-',10),
186     (3,'Winterfell','A+',0),
187     (3,'Winterfell','AB+',0),
188     (3,'Winterfell','A-',0),
189     (3,'Winterfell','B-',20),
190     (3,'Winterfell','B+',10),
191     (3,'Winterfell','AB-',0),
192     (4,'Riverrun','A+',10),
193     (4,'Riverrun','A-',40),
194     (7,'Forestpark','B-',40),
195     (8,'Parkland','B+',10),
196     (9,'Pinecreek','AB-',20);
197
198 •   select * from Hospital_Info_2;
199
```

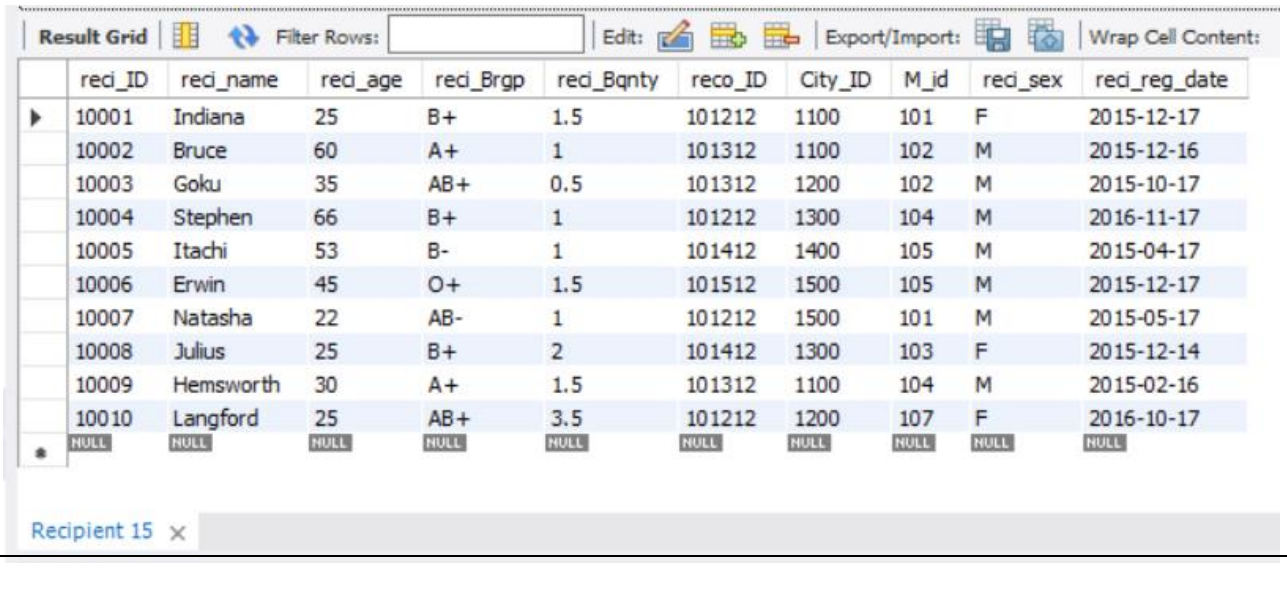
Result Grid | Filter Rows: | Edit:

hosp_ID	hosp_name	hosp_needed_Bgrp	hosp_needed_qnty
1	Springfield	A-	0
1	Springfield	A+	20
1	Springfield	AB-	10
1	Springfield	AB+	40
1	Springfield	B-	20
2	Hampshire	A-	10
2	Hampshire	A+	40
2	Hampshire	AB-	10
2	Hampshire	AB+	20
2	Hampshire	B-	30
2	Hampshire	B+	0
3	Winterfell	A-	0
3	Winterfell	A+	0
3	Winterfell	AB-	0
3	Winterfell	AB+	0
3	Winterfell	B-	20
3	Winterfell	B+	10
4	Riverrun	A-	40
4	Riverrun	A+	10
7	Forestpark	B-	40
8	Parkland	B+	10
9	Pinecreek	AB-	20
NULL	NULL	NULL	NULL

Hospital_Info_2 14 ×

-- Creation of 'Recipient' table

```
-- Creation of 'Recipient' table
CREATE TABLE Recipient (
    reci_ID int NOT NULL PRIMARY KEY,
    reci_name varchar(100) NOT NULL,
    reci_age varchar(10),
    reci_Brgp varchar(100),
    reci_Bqnty float,
    reco_ID int NOT NULL,
    City_ID int NOT NULL,
    M_id int NOT NULL,
    FOREIGN KEY(M_id) REFERENCES BB_Manager(M_id),
    FOREIGN KEY(City_ID) REFERENCES City(City_ID)
);
Alter table Recipient
ADD reci_sex varchar(100);
Alter table Recipient
ADD reci_reg_date date;

-- Value insertion
INSERT into Recipient VALUES
(10001,'Indiana',25,'B+',1.5,101212,1100,101,'F','2015-12-17'),
(10002,'Bruce',60,'A+',1,101312,1100,102,'M','2015-12-16'),
(10003,'Goku',35,'AB+',0.5,101312,1200,102,'M','2015-10-17'),
(10004,'Stephen',66,'B+',1,101212,1300,104,'M','2016-11-17'),
(10005,'Itachi',53,'B-',1,101412,1400,105,'M','2015-04-17'),
(10006,'Erwin',45,'O+',1.5,101512,1500,105,'M','2015-12-17'),
(10007,'Natasha',22,'AB-',1,101212,1500,101,'M','2015-05-17'),
(10008,'Julius',25,'B+',2,101412,1300,103,'F','2015-12-14'),
(10009,'Hemsworth',30,'A+',1.5,101312,1100,104,'M','2015-02-16'),
(10010,'Langford',25,'AB+',3.5,101212,1200,107,'F','2016-10-17');

-- Display table
select * from Recipient;
```



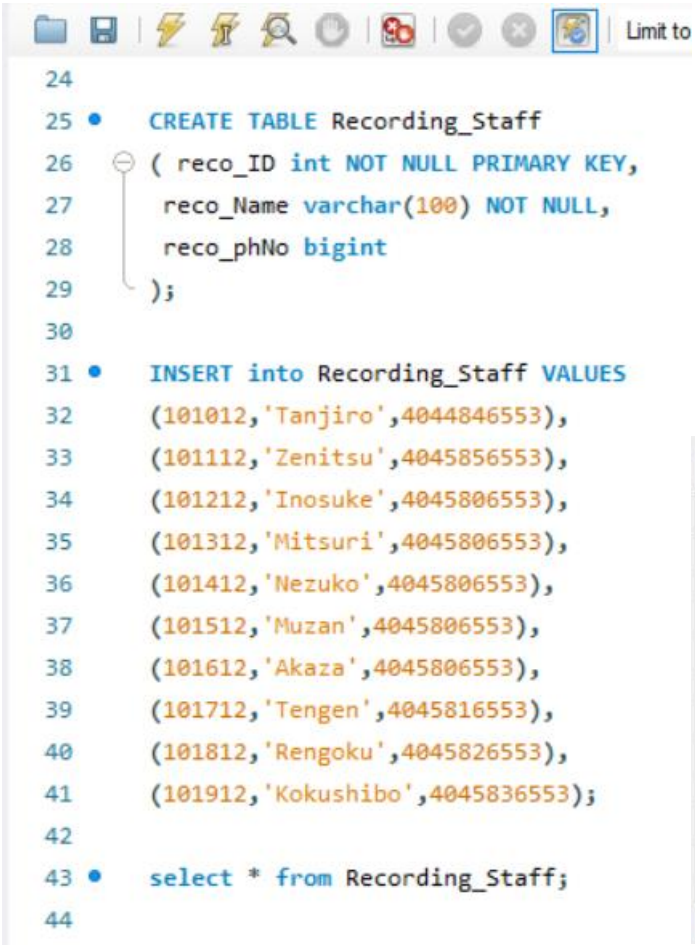
reci_ID	reci_name	reci_age	reci_Brgp	reci_Bqnty	reco_ID	City_ID	M_id	reci_sex	reci_reg_date
10001	Indiana	25	B+	1.5	101212	1100	101	F	2015-12-17
10002	Bruce	60	A+	1	101312	1100	102	M	2015-12-16
10003	Goku	35	AB+	0.5	101312	1200	102	M	2015-10-17
10004	Stephen	66	B+	1	101212	1300	104	M	2016-11-17
10005	Itachi	53	B-	1	101412	1400	105	M	2015-04-17
10006	Erwin	45	O+	1.5	101512	1500	105	M	2015-12-17
10007	Natasha	22	AB-	1	101212	1500	101	M	2015-05-17
10008	Julius	25	B+	2	101412	1300	103	F	2015-12-14
10009	Hemsworth	30	A+	1.5	101312	1100	104	M	2015-02-16
10010	Langford	25	AB+	3.5	101212	1200	107	F	2016-10-17
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

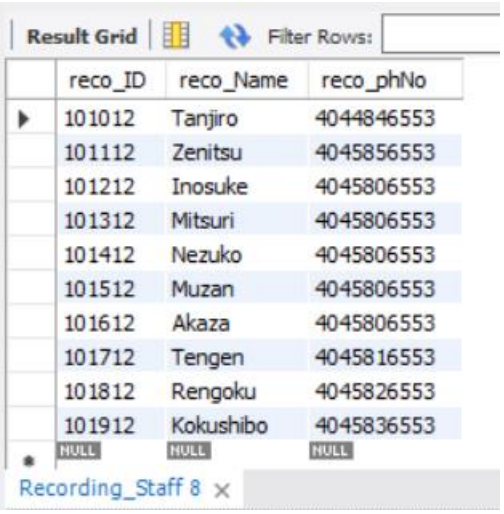

```
CREATE TABLE Recipient
( reci_ID int NOT NULL PRIMARY kEY,
 reci_name varchar(100) NOT NULL,
 reci_age varchar(10),
 reci_Brgp varchar(100),
 reci_Bqnty float,
 reco_ID int NOT NULL,
 City_ID int NOT NULL,
 M_id int NOT NULL,
 FOREIGN KEY(M_id) REFERENCES BB_Manager(M_id),
 FOREIGN KEY(City_ID) REFERENCES City(City_ID)
);

Alter table Recipient
ADD reci_sex varchar(100);

Alter table Recipient
ADD reci_reg_date date;
```

```
INSERT into Recipient VALUES
(10001,'Indiana',25,'B+',1.5,101212,1100,101,'F','2015-12-17'),
(10002,'Bruce',60,'A+',1,101312,1100,102,'M','2015-12-16'),
(10003,'Goku',35,'AB+',0.5,101312,1200,102,'M','2015-10-17'),
(10004,'Stephen',66,'B+',1,101212,1300,104,'M','2016-11-17'),
(10005,'Itachi',53,'B-',1,101412,1400,105,'M','2015-04-17'),
(10006,'Erwin',45,'O+',1.5,101512,1500,105,'M','2015-12-17'),
(10007,'Natasha',22,'AB-',1,101212,1500,101,'M','2015-05-17'),
(10008,'Julius',25,'B+',2,101412,1300,103,'F','2015-12-14'),
(10009,'Hemsworth',30,'A+',1.5,101312,1100,104,'M','2015-02-16'),
(10010,'Langford',25,'AB+',3.5,101212,1200,107,'F','2016-10-17');

select * from Recipient;
```

-- Creation of 'Recording_Staff' table

```sql
-- Creation of 'Recording_Staff' table
CREATE TABLE Recording_Staff (
    reco_ID int NOT NULL PRIMARY KEY,
    reco_Name varchar(100) NOT NULL,
    reco_phNo bigint
);

-- Value insertion
INSERT into Recording_Staff
VALUES (101012, 'Tanjiro', 4044846553),
    (101112, 'Zenitsu', 4045856553),
    (101212, 'Inosuke', 4045806553),
    (101312, 'Mitsuri', 4045806553),
    (101412, 'Nezuko', 4045806553),
    (101512, 'Muzan', 4045806553),
    (101612, 'Akaza', 4045806553),
    (101712, 'Tengen', 4045816553),
    (101812, 'Rengoku', 4045826553),
    (101912, 'Kokushibo', 4045836553);

-- Display table
select * from Recording_Staff;
```





SAMPLE SQL QUERIES

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

```
237     -- SAMPLE SQL QUERIES
238
239     -- Query 1
240 •   CREATE VIEW Blood_Recipient_SameBGrp AS
241     select Blood_Donor.bd_name,Recipient.reci_name,reco_Name from Recording_Staff
242     inner join Blood_Donor on Recording_Staff.reco_ID = Blood_Donor.reco_ID
243     inner join Recipient on Recording_Staff.reco_ID = Recipient.reco_ID
244     where Blood_Donor.bd_Bgroup = Recipient.reci_Brgp and
245     Blood_Donor.bd_reg_date = Recipient.reci_reg_date;
246
247 •   select* from Blood_Recipient_SameBGrp;
248
```

Output:

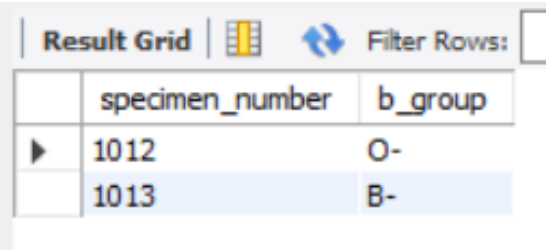
Result Grid | Filter Rows:

	bd_name	reci_name	reco_Name
▶	Natasha	Indiana	Inosuke

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

```
249     -- Query 2
250 •   Select specimen_number,b_group from BloodSpecimen,DiseaseFinder
251     WHERE BloodSpecimen.dfind_ID= DiseaseFinder.dfind_ID AND dfind_name='Gwen' AND status=1;
252
```

Output:



Result Grid | Filter Rows:

	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.**

```
253     -- Query 3
254 •   select BB_Manager.M_id,mName,Recipient.reci_name, Recipient.reci_Brgp,BloodSpecimen.b_group
255     from BB_Manager,Recipient,BloodSpecimen
256     where Recipient.M_id = BloodSpecimen.M_id
257     and Recipient.reci_Brgp = BloodSpecimen.b_group
258     and Recipient.M_id = BB_Manager.M_id and status = 1;
```

Output:

Result Grid | Filter Rows: | Export:

	M_id	mName	reci_name	reci_Brgp	b_group
▶	101	Vatsalya	Indiana	B+	B+
	102	Vicky	Goku	AB+	AB+

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

```
260     -- Query 4
261 •   Select bd_ID,bd_name,reci_ID,reci_name
262     FROM Blood_Donor,Recipient
263     WHERE bd_Bgroup=reci_Brgp AND Blood_Donor.City_ID = Recipient.City_ID;
```

Output:

Result Grid | Filter Rows:

bd_ID	bd_name	reci_ID	reci_name
150013	Bruce	10003	Goku
150013	Bruce	10010	Langford
150014	Natasha	10004	Stephen
150014	Natasha	10008	Julius
150017	Sherlock	10005	Itachi

Result 19

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

```
265     -- Query 5
266 •   Select hosp_ID,hosp_name , City_ID, HOspital_Info_1.M_id
267     from Hospital_Info_1,BB_Manager
268     where BB_Manager.M_id = Hospital_Info_1.M_id and BB_Manager.M_id = 103;
```

Output:

Result Grid | Filter Rows:

hosp_ID	hosp_name	City_ID	M_id
2	Hampshire	1200	103
3	Winterfell	1300	103
5	Hogsmeade	1800	103

CONCLUSION

Prior to this project, a general study of blood bank management system was conducted from recent researches of various authors and facts were gathered in which helped to uncover the misfits that the system was facing.
After proper analyzation 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 have 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 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.