

**ADAMA SCIENCE AND TECHNOLOGY UNIVERSITY**

**BLOOD BANK MANAGEMENT SYSTEM**

DATABASE GROUP PROJECT

1. Sumeya Akmel ugr/31260/15
2. Nebil afdelu ugr/31064/15
3. Mekhluqat abdulwehab ugr/30876/15
4. Bati jano ugr/30230/15
5. Yusuf muhammed ugr/31467/15
6. Fuad muhammed ugr/30573/15

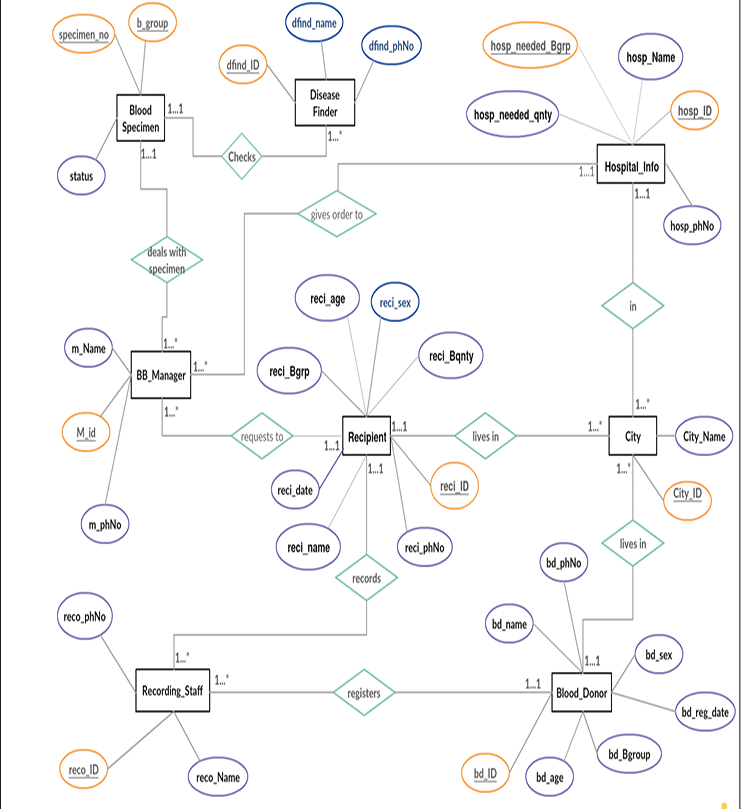
Submitted to mr. shumet

Submission date wed, sep 25

**Project description**

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 donate the blood that is ‘donors’. But this 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 USING CREATLY AND RELATION BETWEEN THE ENTITIES**

****

**Created by creatly.com**

**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.

1. **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 entity to 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.

1. **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.

1. **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.

1. **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.

1. **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 entitity.

1. **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.

1. **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**

1. **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.

2. **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.

3. **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.

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 = “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. 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**

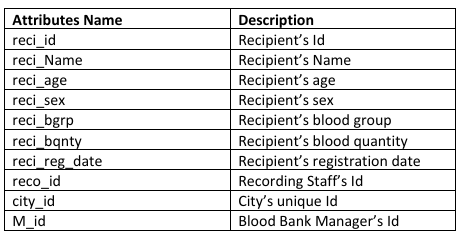
Donor Table:

|  |  |
| --- | --- |
| **Attribute name** | **description** |
| bd\_id | Blood donors id |
| Bd\_name | Blood donors name |
| Bd\_age | Blood donors age |
| Bd\_sex | Blood donors sex |
| Bd\_bgrp | Blood donors blood group |
| Bd\_regdate | Registration date of donor |
| Reco\_id | Id of recording staff |
| City\_id | City id |

• 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.

Receipent table:

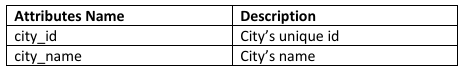
****

• 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.

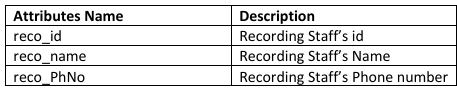
• The relationship with Blood Bank Manager and Blood Recipient is 1 to many. That’s why primary key of Blood Specimen is used as a foreign key in Blood Recipient.

City table:

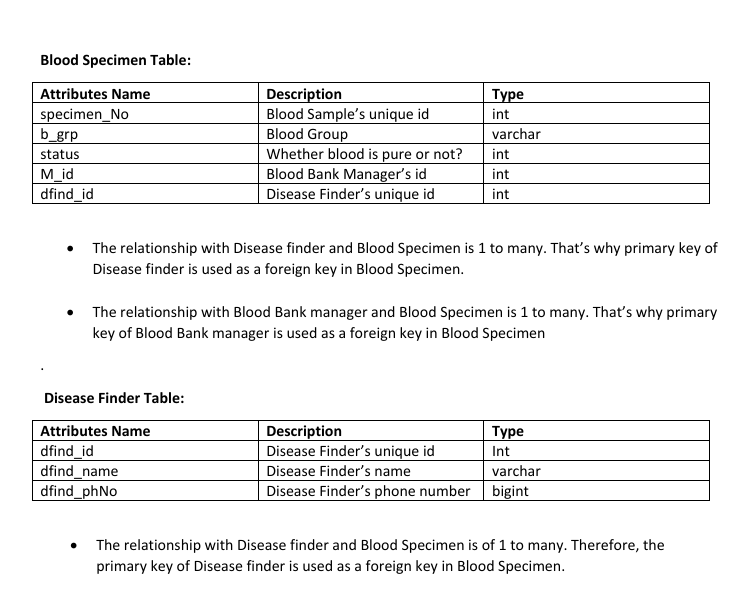
****

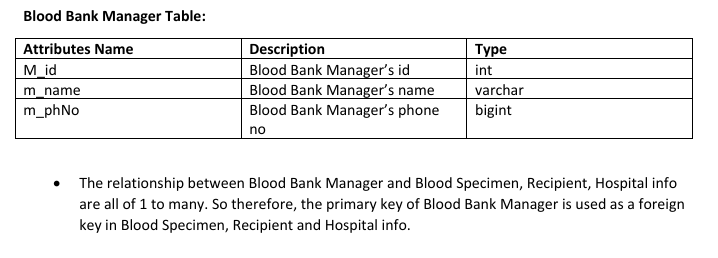
• The relationship between City and Recipients, Donor, Hospital info are all of 1 to many. So that’s why primary key of City is used as a foreign key in Recipients, Donor and Hospital info.

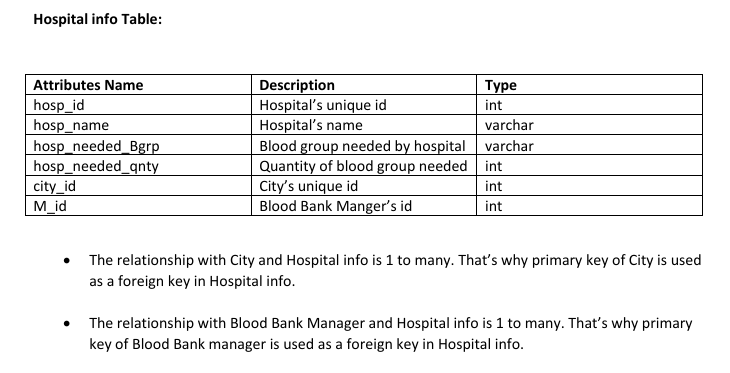
Recording Staff Table:

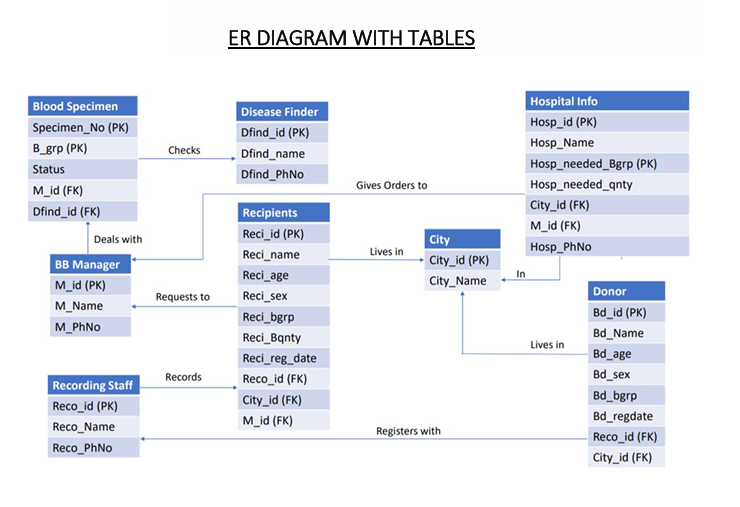
****

• 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.

****

****

****

****

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.

1. 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.

1. 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.

1. 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.

1. 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.

1. 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.

1. 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.

1. 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)

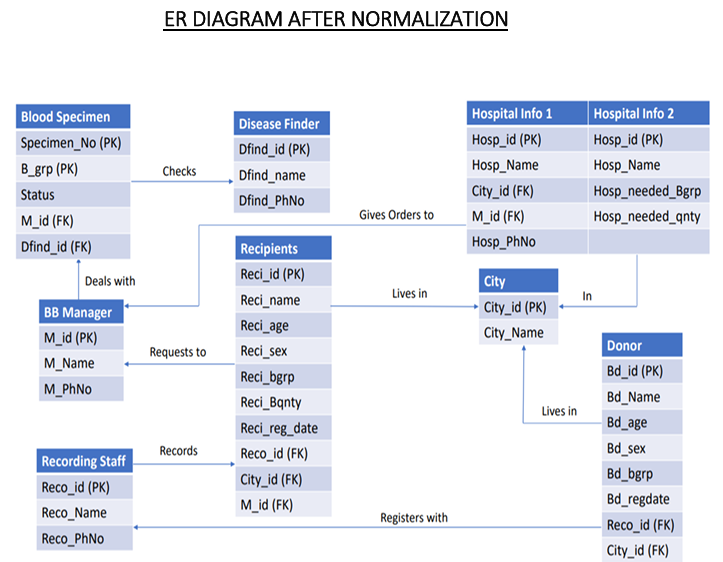
The table is in first normal form.

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.

****

**SQL IMPLEMENTATION WILL BE**

-- Drop tables if they already exist

DROP TABLE IF EXISTS BB\_Manager;

DROP TABLE IF EXISTS Blood\_Donor;

DROP TABLE IF EXISTS BloodSpecimen;

DROP TABLE IF EXISTS City;

DROP TABLE IF EXISTS DiseaseFinder;

DROP TABLE IF EXISTS Hospital\_Info\_1;

DROP TABLE IF EXISTS Hospital\_Info\_2;

DROP TABLE IF EXISTS Recipient;

DROP TABLE IF EXISTS Recording\_Staff;

-- Create table BB\_Manager

CREATE TABLE BB\_Manager (

M\_id int NOT NULL,

mName varchar(255) NOT NULL,

m\_phNo bigint

-- CONSTRAINT Mid\_pk PRIMARY KEY (M\_id)

);

-- Insert values into BB\_Manager

INSERT INTO BB\_Manager VALUES

(102,'sumeya', 4693959671),

(103,'mekhluqat', 4693959601),

(104,'nebil', 4693959677),

(105,'yusuf', 4693957671),

(106,'fuad', 4694959671),

(107,'abebe', 4695959671),

(108,'bekele', 4663959671),

(109,'selam', 4673959671),

(110,'debebe', 4693859671);

-- Create table Blood\_Donor

CREATE TABLE Blood\_Donor (

bd\_ID int NOT NULL,

bd\_name varchar(255) NOT NULL,

bd\_age int,

bd\_sex varchar(10),

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 values into Blood\_Donor

INSERT INTO Blood\_Donor VALUES

(150221,'sheleme',25,'M','B+','2015-12-17',101212,1100),

(160011,'temkin',35,'F','A+','2016-11-22',101212,1100),

(160101,'sami',22,'M','O+','2016-01-04',101312,1200),

(150011,'aman',29,'M','O+','2015-07-19',101412,1300),

(150021,'fiker',42,'F','A-','2015-12-24',101412,1300),

(150121,'Dan',44,'M','AB+','2015-08-28',101212,1200),

(160031,'sena',33,'F','AB-','2016-02-06',101212,1400),

(160301,'Elsa',31,'F','AB+','2016-09-10',101312,1200),

(160091,'miki',24,'M','B-','2016-10-15',101312,1500),

(160401,'chala',29,'M','O-','2016-12-17',101212,1200);

-- Create table BloodSpecimen

CREATE TABLE BloodSpecimen (

specimen\_number int NOT NULL PRIMARY KEY,

b\_group varchar(10) NOT NULL,

status int,

dfind\_ID int NOT NULL,

M\_id int NOT NULL

);

-- Insert values into BloodSpecimen

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);

-- Create table City

CREATE TABLE City (

City\_ID int NOT NULL,

City\_name varchar(255) NOT NULL

-- CONSTRAINT CityID\_pk PRIMARY KEY (City\_ID)

);

-- Insert values into City

INSERT INTO City VALUES

(1200,'diredawa'),

(1300,'harar'),

(1400,'addisabeba'),

(1500,'arbaminch'),

(1600,'jinka'),

(1700,'afar'),

(1800,'akaki'),

(1900,'bole'),

(2000,'cherkos');

-- Create table DiseaseFinder

CREATE TABLE DiseaseFinder (

dfind\_ID int NOT NULL,

dfind\_name varchar(255) NOT NULL,

dfind\_PhNo bigint

-- CONSTRAINT dfindID\_pk PRIMARY KEY (dfind\_ID)

);

-- Insert values into DiseaseFinder

INSERT INTO DiseaseFinder VALUES

(11,'selam',4693804223),

(12,'ababu',4693804223),

(13,'Jerry',4693804223),

(14,'Mark',4693804223),

(15,'ayna',4693804223),

(16,'iman',4693804123),

(17,'fetiya',4693804223),

(18,'sitra',4693804323),

(19,'bontu',4693804423),

(20,'kebede',4693804523);

-- Create table Hospital\_Info\_1

CREATE TABLE Hospital\_Info\_1 (

hosp\_ID int NOT NULL PRIMARY KEY,

hosp\_name varchar(255) NOT NULL,

City\_ID int NOT NULL,

M\_id int NOT NULL

);

-- Insert values into Hospital\_Info\_1

INSERT INTO Hospital\_Info\_1 VALUES

(1,'meklitclinic',1100,101),

(2,'teklehaymanothospital',1200,103),

(3,'ymranclinic',1300,103),

(4,'Baylor',1400,104),

(5,'Goh',1800,103),

(6,'abeselom',1300,106),

(7,'abrehot',1300,102),

(8,'truneshbeijing',1200,106),

(9,'Pawlos',1500,109),

(10,'mentewabclinic',1700,105);

-- Create table Hospital\_Info\_2

CREATE TABLE Hospital\_Info\_2 (

hosp\_ID int NOT NULL,

hosp\_name varchar(255) NOT NULL,

hosp\_needed\_Bgrp varchar(10),

hosp\_needed\_qnty int,

PRIMARY KEY (hosp\_ID, hosp\_needed\_Bgrp)

);

-- Insert values into Hospital\_Info\_2

INSERT INTO Hospital\_Info\_2 VALUES

(1,'meklitclinic','A+',20),

(1,'meklitclinic','AB+',0),

(1,'meklitclinic','A-',40),

(1,'meklitclinic','B-',10),

(1,'meklitclinic','AB-',20),

(2,'teklehaymanothospital','A+',40),

(2,'teklehaymanothospital','AB+',20),

(2,'teklehaymanothospital','A-',10),

(2,'teklehaymanothospital','B-',30),

(2,'teklehaymanothospital','B+',0),

(2,'teklehaymanothospital','AB-',10),

(3,'ymranclinic','A+',0),

(3,'ymranclinic','AB+',0),

(3,'ymranclinic','A-',0),

(3,'ymranclinic','B-',20),

(3,'ymranclinic','B+',10),

(3,'ymranclinic','AB-',0),

(4,'Baylor','A+',10),

(5,'abeselom','B+',30),

(4,'Baylor','A-',40),

(7,'abrehot','B-',40),

(8,'truneshbeijing','B+',10),

(9,'Pawlos','AB-',20);

-- Create table Recipient

CREATE TABLE Recipient (

reci\_ID int NOT NULL,

reci\_name varchar(255) NOT NULL,

reci\_age int,

reci\_Brgp varchar(10),

reci\_Bqnty float,

reco\_ID int NOT NULL,

City\_ID int NOT NULL,

M\_id int NOT NULL,

reci\_sex varchar(10),

reci\_reg\_date date

-- CONSTRAINT reciid\_pk PRIMARY KEY (reci\_id)

);

-- Insert values into Recipient

INSERT INTO Recipient VALUES

(10001,'chala',25,'B+',1.5,101212,1100,101,'M','2015-12-17'),

(10002,'Dan',60,'A+',1,101312,1100,102,'M','2015-12-16'),

(10003,'chube',35,'AB+',0.5,101312,1200,102,'M','2015-10-17'),

(10004,'yoseph',66,'B+',1,101212,1300,104,'M','2016-11-17'),

(10005,'coba',53,'B-',1,101412,1400,105,'M','2015-04-17'),

(10006,'sehin',45,'O-',2,101512,1500,106,'M','2015-02-28');

-- Check data in BB\_Manager

SELECT \* FROM BB\_Manager;

-- Check data in Blood\_Donor

SELECT \* FROM Blood\_Donor;

-- Check data in BloodSpecimen

SELECT \* FROM BloodSpecimen;

-- Check data in City

SELECT \* FROM City;

-- Check data in DiseaseFinder

SELECT \* FROM DiseaseFinder;

-- Check data in Hospital\_Info\_1

SELECT \* FROM Hospital\_Info\_1;

-- Check data in Hospital\_Info\_2

SELECT \* FROM Hospital\_Info\_2;

-- Check data in Recipient

SELECT \* FROM Recipient;

**QUERY OPTIMIZATION**

1. Add primary keys where they are commented out

Primary keys ensure uniqueness and improve query performance by allowing the database engine to quickly locate rows. Adding primary keys to tables like BB\_Manager, Blood\_Donor, City, DiseaseFinder, and Recipient will enhance data integrity and performance.

1. Use appropriate datatypes for phone number

Change the data type of phone numbers from bigint to varchar(15) or similar, as phone numbers are not used for arithmetic operations and can contain leading zeros or special characters.

1. Use consistent naming convention for columns and table

Ensure that column and table names follow a consistent naming convention to improve readability and maintainability of the SQL script. This does not directly impact performance but aids in long-term management.

1. Remove unnecessary select statements at the end of script

The select statements at the end of the script are used for checking data and are not necessary for the creation and insertion process. Removing them can slightly improve script execution time.

**Changes made after optimization**

**Example:**

--Create table BB\_Manager

CREATE TABLE BB\_Manager (

    M\_id int NOT NULL,

    mName varchar(255) NOT NULL,

    m\_phNo bigint

    -- CONSTRAINT Mid\_pk PRIMARY KEY (M\_id)

**);**

**After**

- Create table bb\_manager

CREATE TABLE bb\_manager (

    manager\_id int NOT NULL,

    manager\_name varchar(255) NOT NULL,

    manager\_phone varchar(15), -- Changed from bigint to varchar(15)

    CONSTRAINT manager\_id\_pk PRIMARY KEY (manager\_id)

);

**Before**

-- Insert values into Blood\_Donor

INSERT INTO Blood\_Donor VALUES

**After**

-- Batch insert values into blood\_donor

INSERT INTO blood\_donor (donor\_id, donor\_name, donor\_age, donor\_sex, donor\_blood\_group, donor\_registration\_date, recorder\_id, city\_id)

**Before**

-- Create table BloodSpecimen

CREATE TABLE BloodSpecimen (

    specimen\_number int NOT NULL PRIMARY KEY,

    b\_group varchar(10) NOT NULL,

    status int,

    dfind\_ID int NOT NULL,

    M\_id int NOT NULL

);

-- Insert values into BloodSpecimen

INSERT INTO BloodSpecimen VALUES

**After**

-- Create table blood\_specimen

CREATE TABLE blood\_specimen (

    specimen\_number int NOT NULL PRIMARY KEY,

    blood\_group varchar(10) NOT NULL,

    status int,

    disease\_finder\_id int NOT NULL,

    manager\_id int NOT NULL

);

-- Batch insert values into blood\_specimen

INSERT INTO blood\_specimen (specimen\_number, blood\_group, status, disease\_finder\_id, manager\_id)

Before

-- Create table City

CREATE TABLE City (

    City\_ID int NOT NULL,

    City\_name varchar(255) NOT NULL

    -- CONSTRAINT CityID\_pk PRIMARY KEY (City\_ID)

);

After

-- Create table city

CREATE TABLE city (

    city\_id int NOT NULL,

    city\_name varchar(255) NOT NULL,

    CONSTRAINT city\_id\_pk PRIMARY KEY (city\_id)

);

-- Batch insert values into city

INSERT INTO city (city\_id, city\_name)

Before

-- Create table DiseaseFinder

CREATE TABLE DiseaseFinder (

    dfind\_ID int NOT NULL,

    dfind\_name varchar(255) NOT NULL,

    dfind\_PhNo bigint

    -- CONSTRAINT dfindID\_pk PRIMARY KEY (dfind\_ID)

After

-- Create table disease\_finder

CREATE TABLE disease\_finder (

    disease\_finder\_id int NOT NULL,

    disease\_finder\_name varchar(255) NOT NULL,

    disease\_finder\_phone varchar(15), -- Changed from bigint to varchar(15)

    CONSTRAINT disease\_finder\_id\_pk PRIMARY KEY (disease\_finder\_id)

);

Before

-- Insert values into Hospital\_Info\_1

INSERT INTO Hospital\_Info\_1 VALUES

After

-- Batch insert values into hospital\_info\_1

INSERT INTO hospital\_info\_1 (hospital\_id, hospital\_name, city\_id, manager\_id)

And etc…

**CONCLUSION**

Our project well addressed the limitations of the existing system. We designed well organized database management system which is a challenging job in this era. We have built a database for a Blood Bank using Microsoft SQL Server. 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, we have created our ERD and then converted the ERD to relational model and normalized the tables. Using Microsoft SQL Server we have created the tables for our database and inserted some sample values in the tables. Finally, we have executed sample queries on our database to check its performance to retrieve useful information accurately and speedily and optimized our code to be efficient.