A Project Report

on

Blood Bank Management System

Course : Database Management System

Submitted By:-

Name	Seat No.	PRN
Yash Vikram Solanki	**486174**	**8022053512**
Maiitra Nischal Patel	**486148**	**8022053698**
Khushi SanjivBhai Patel	**486147**	**8022056539**

DBMS Project

TABLE OF CONTENTS

BLOOD BANK MANAGEMENT SYSTEM

DESCRIPTION:

The Blood Banking Management System is a comprehensive database management project aimed at efficiently managing the complexities of blood donation, storage, and distribution processes. This system provides a centralized platform for blood banks to streamline their operations, ensuring timely access to blood products for patients in need.

CONTENTS:

Our Blood bank management ER diagram encompasses several interconnected tables, each serving a specific purpose. Let's briefly describe the key components of this project:

Hospital:

Hospitals play a critical role in the Blood Bank Management System as they serve as the primary recipients of blood products for patient transfusions. Each hospital is represented in the database with detailed information including a unique hospital ID, name, and address. Additionally, contact information such as phone numbers and email addresses are stored in the Hospital_Contact and Hospital_Email tables respectively. Furthermore, hospitals are associated with patients through the Patient table, which records patient admissions, discharges, and blood transfusions.

Patient:

The Patient entity captures comprehensive information about individuals receiving medical treatment at hospitals. Each patient is identified by a unique patient ID within a specific hospital. The database stores patient demographics including name, age, sex, and blood type. Additionally, the Patient table records admission and discharge dates, providing a timeline of patient care. This information facilitates the efficient management of blood transfusions, ensuring that patients receive the appropriate blood products based on their medical needs. Furthermore, the Patient_Contact table stores contact information for patients, enabling hospitals to communicate important updates or follow-up care instructions.

Blood Donation Camp:

Blood donation camps serve as essential venues for collecting blood donations from voluntary donors. In the Blood Bank Management System, each camp is represented by the Blood_Donation_Camp entity, which records details such as the organizing organization, camp ID, location, start and end dates, and associated blood bank. This information helps coordinate the logistics of blood donation drives, including scheduling, staffing, and resource allocation. The Camp_Contact table stores contact information for each camp, facilitating communication with donors, volunteers, and organizers.

Donor:

The Donor entity stores detailed information about individuals who voluntarily donate blood, including a unique donor ID, name, date of birth, and age. Donor health information such as weight and blood-related parameters are recorded in the Check-ups and Blood tables respectively. Moreover, the Donor_Contact table maintains contact information for donors, facilitating communication for future donation drives or health updates. Nurses, responsible for donor screening and assistance during blood donation, are linked to donors through the Nurse table.

Blood Bank:

Blood banks serve as central hubs for the collection, processing, storage, and distribution of blood products. In the Blood Bank Management System, each blood bank is represented by the Blood_Bank entity, which stores information such as the blood bank ID, name, and location. Additionally, the entity maintains a record of available blood types and their associated costs in the Bank_Blood_Type table. This information enables hospitals to place orders for specific blood products based on patient needs and budget considerations. Effective management of blood banks ensures the timely availability of safe and compatible blood products for transfusions, contributing to patient care and medical emergencies.

ASSUMPTIONS

Let's outline the assumptions for creating the tables based on the ER diagram, including the cardinality of the relationships:

1. Check-ups:
    - Each volunteer has a unique Vol_ID. It is related to Donor_ID of Donor table.
    - Attributes include Name, Weight, DOB, and Age (Derived from DOB).
    - Relationship:
        - One-to-One: A volunteer/donor can have only one check-up (1:1).

2. Donor:
    - Each donor has a unique Donor_ID.
    - Attributes include Name, DOB, Age (Derived from DOB), Contact_Info, and Nurse_ID.
    - Relationship:
        - One-to-Many: A donor can participate in Multiple blood donations (1:N).
        - One-to-One: A donor can participate in only check-up (1:1).
        - Many-to-One: Many donors can have blood collected by a common nurse (M:1).

3. Nurse:
    - Each nurse has a unique Nurse_ID.
    - Attributes include Name, DOB, Age (Derived from DOB), Organization, Contact_Info, and Camp_ID.
    - Relationship:
        - Many-to-One: Multiple nurses can be assigned one blood donation camp (N:1).
        - One-to-Many: One nurse collects blood from multiple Donors (1:N).

4. Blood:
    - Contains information related to blood samples.
    - Attributes include Blood_Type, Haemoglobin, Red_blood_cells, White_blood_cells, Platelets, and Plazma.
    - Linked to donors via Donor_ID.
    - Relationship:
        - Many-to-One: Multiple blood sample can be donated by one donor (N:1).
        - Many-to-One: Multiple blood sample can go into one blood donation camp (N:1).

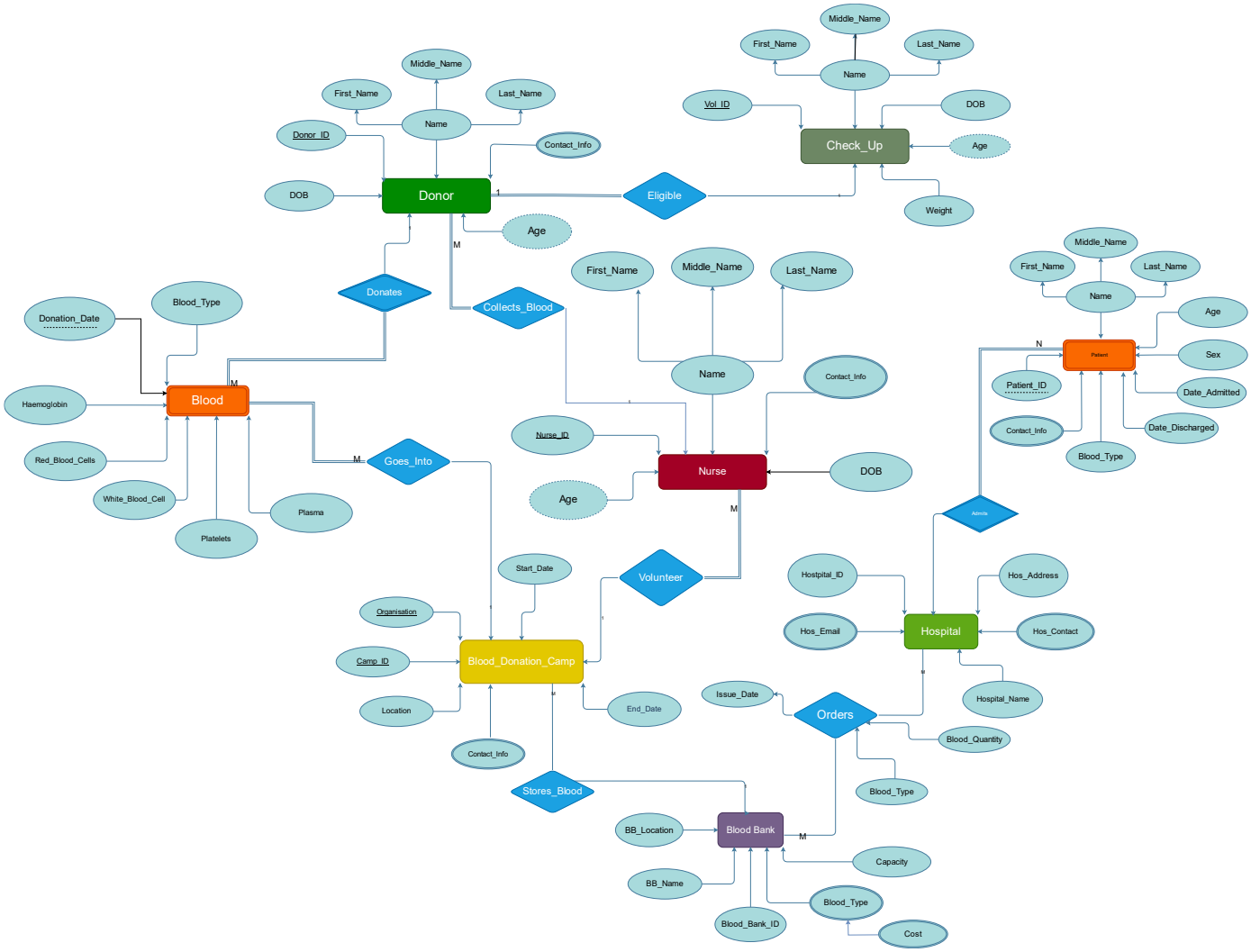
5. Blood_Donation_Camp:
    - Represents blood donation camps.
    - Each camp has a unique Organization + Camp_ID.
    - Attributes include Organization, location, Start_date, and End_date.
    - Linked to a specific Blood_Bank via Blood_Bank_ID.
    - Relationship:
        - Many-to-One: Multiple blood donation camps can be associated with one blood bank (N:1).
        - One-to-Many: One blood donation camp can have multiple nurses (1:N).
        - One-to-Many: One blood donation camp receives multiple blood samples (1:N).

6. Blood_Bank:
    - Represents blood storage and distribution centers.
    - Each blood bank has a unique Blood_Bank_ID.
    - Attributes include BB_Name (bank name), BB_Location, Blood_Type and Capacity.
    - Relationship:
        - One-to-Many: A blood bank can organize multiple blood donation camps (1:N).
        - Many-to-Many: Multiple Blood Banks can cater to orders from multiple Hospitals (N:N).

7. Hospital:
    - Represents healthcare institutions where patients receive medical treatment.
    - Each hospital has a unique Hospital_ID.
    - Attributes include Hospital_Name and Hos_Address.
    - Relationship:
        - One-to-Many: A hospital can admit multiple patients (1:N).
        - Many-to-Many: Multiple hospitals can order from multiple blood banks (N:N).

8. Patient:
    - Represents individuals seeking medical attention at hospitals.
    - Each patient has a unique Hospital_ID + Patient_ID.
    - Attributes include Name, Age, Sex, Blood_Type, Date_Admitted, and Date_Discharged.
    - Relationship:
        - Many-to-One: Each patient is admitted to one hospital (N:1).

ER-DIAGRAM


Middle_Name
First_Name
Last_Name
Name
Vol_ID
DOB
Check_Up
Age
Weight
Middle_Name
First_Name
Last_Name
Name
Donor_ID
Contact_Info
DOB
Donor
Eligible
1
Age
M
Donates
Collects_Blood
First_Name
Middle_Name
Last_Name
Name
Contact_Info
Donation_Date
Blood_Type
Blood
M
Haemoglobin
Red_Blood_Cells
White_Blood_Cell
Platelets
Plasma
M
Goes_Into
Nurse_ID
Nurse
DOB
Age
M
Start_Date
Volunteer
Organisation
Camp_ID
Location
Blood_Donation_Camp
End_Date
Contact_Info
Stores_Blood
Middle_Name
First_Name
Last_Name
Name
Age
N
Patient
Sex
Patient_ID
Date_Admitted
Contact_Info
Date_Discharged
Blood_Type
Admits
Hostpital_ID
Hos_Address
Hos_Email
Hospital
Hos_Contact
Hospital_Name
Issue_Date
Orders
Blood_Quantity
Blood_Type
BB_Location
Blood Bank
M
Capacity
BB_Name
Blood_Type
Blood_Bank_ID
Cost

TABLES BEFORE NORMALIZATION

Note : Primary key is denoted as primary key
Foreign key is denoted as foreign key
A key that is both foreign and composite key is denoted as foreign key + primary key

➔ Check-ups (Vol_ID, First_Name, Middle_Name, Last_Name, Weight, DOB)

➔ Blood_Bank (Blood_Bank_ID , BB_Name, BB_Location, Blood_Type, Cost)

➔ Blood_Donation_Camp (Orgaization , Camp_ID , location , Start_date , End_dt, Blood_Bank_ID, Contact_info)

➔ Nurse (Nurse_ID , First_Name, Middle_Name, Last_Name, DOB , Organization, Camp_ID, Contact_Info)

➔ Donor(Donor_ID, First_Name, Middle_Name, Last_Name, DOB, Nurse_ID, Contact_Info)

➔ Blood (Donor_ID,Donation_Date, Orgaization , Camp_ID, Blood_Type , Haemoglobin , Red_blood_cells , White_blood_cells , Platelets , Plazma)

➔ Hospital (Hospital_ID , Hospital_Name, Hos_Address, Contact_Info, Hos_Email)

➔ Order (Blood_Bank_ID, Hospital_ID, Issue_Date, Blood_Type, Blood_Quantity)

Patient (Hospital_ID , Patient_ID , First_Name, Middle_Name, Last_Name , Sex , Blood_Type , Date_Admitted , Date_Discharged, Contact_Info

NORMALIZATION OF TABLES

1. Check_up :

Check-ups (Vol_ID, First_Name, Middle_Name, Last_Name, Weight, DOB)

- Here this table is in 1st a normal form because every record is atomic.
- This table is in 2nd normal form because there is no partial dependency.
- It is in a 3rd normal form because there is a transitivity.
- Hence there is no need to normalize the table.

Check-ups (Vol_ID, First_Name, Middle_Name, Last_Name, Weight, DOB)

2. Blood_Bank :

Blood_Bank (Blood_Bank_ID , BB_Name, BB_Location, Blood_Type, Cost)

- Here this table is not in 1st a normal form because every record is not atomic.
- Here Blood_Type and Cost is a multivalued attribute.
- We can create different table for contact info to reduce it to the 1st normal form.
- Here Cost of Blood depends upon Blood_Type , hence transitivity exists.

Blood_Bank (Blood_Bank_ID , BB_Name , BB_Location)

Bank_Blood_Cost (Blood_Bank_ID , Blood_Type , Cost)

- This table is in 2nd normal form because there is no partial dependency.
- Because here Cost of blood in Blood_Bank_Cost table depends upon both Blood_Bank_Id and Blood_Type.
- It is in 3rd normal form because there is no transitivity.

3. Blood_Donation_Camp :

Blood_Donation_Camp (Orgaization , Camp_ID , location , Start_date , End_dt, Blood_Bank_ID, Contact_info)

- Here this table is not in 1st a normal form because every record is not atomic.
- Here Contact_Info is a multivalued attribute.
- We can create different table for contact info to reduce it to the 1st normal form.

Blood_Donation_Camp (Orgaization , Camp_ID , location , Start_date , End_dt, Blood_Bank_ID)

Camp_Contact (Organization , Camp_ID ,Contact_info)

- Blood_Donation table is not in 2nd normal form because there is a partial dependency.
- Here we have composite key Organization and Camp_ID.
- Here Blood_Bank_ID only depends upon Organization and not on Camp_ID.

Key = Organization + Camp_ID

Organization → Blood_Bank_ID

Hence we have to break given table into 2 tables.

Blood_Donation_Camp (Orgaization , Camp_ID , location , Start_date , End_date)

Organization_Blood_Bank(Organization, Blood_Bank_ID)

- It is in 3rd normal form because there is no transitivity.

Blood_Donation_Camp (Orgaization , Camp_ID , location , Start_date , End_date)

Organization_Blood_Bank(Organization, Blood_Bank_ID)

Camp_Contact (Organization , Camp_ID ,Contact_info)

4. Nurse :

Nurse (Nurse_ID , First_Name, Middle_Name, Last_Name, DOB , Organization, Camp_ID, Contact_Info)

- Here this table is not in 1st a normal form because every record is not atomic.
- Here Contact_Info is a multivalued attribute.
- We can create different table for contact info to reduce it to the 1st normal form.

Nurse (Nurse_ID , First_Name, Middle_Name, Last_Name, DOB , Organization, Camp_ID, Contact_Info)

Nurse_Contact (Nurse_ID,Contact_info)

- Both tables are in 2nd normal form because there is no partial dependency.
- Nurse is in a 3rd normal form because there is a transitivity.

Nurse (Nurse_ID , First_Name, Middle_Name, Last_Name, DOB , Organization, Camp_ID, Contact_Info)

Nurse_Contact (Nurse_ID,Contact_info)

5. Donor :

Donor(Donor_ID, First_Name, Middle_Name, Last_Name, DOB, Nurse_ID, Contact_Info)

- Here this table is not in 1st a normal form because every record is not atomic.
- Here Contact_Info is a multivalued attribute.
- We can create different table for contact info to reduce it to the 1st normal form.

Donor(Donor_ID, First_Name, Middle_Name, Last_Name, DOB, Nurse_ID)

Donor_Contact (Donor_ID , Contact_info)

- Both tables are in 2nd normal form because there is no partial dependency.
- Donor is in 3rd normal form because there is a transitivity.

6. Blood :

Blood (Donor_ID,Donation_Date, Orgaization , Camp_ID, Blood_Type , Haemoglobin , Red_blood_cells , White_blood_cells , Platelets , Plazma)

- Here this table is in 1st a normal form because every record is atomic.

- This table is in 2nd normal form because there is no partial dependency.
- It is in 3rd normal form because there is no transitivity.

7. Hospital :

Hospital (Hospital_ID , Hospital_Name , Hos_Address, Hos_Phone, Hos_Email)

- Here this table is not in 1st a normal form because every record is not atomic.
- Here Hos_Phone and Hos_Email is a multivalued attribute.
- We can create different table for contact info to reduce it to the 1st normal form.

Hospital (Hospital_ID , Hospital_Name , Hos_Address)

Hospital_Contact (Hospital_ID , Hos_Phone)

Hospital_Email (Hospital_ID , Hos_Email)

- This table is in 2nd normal form because there is no partial dependency.
- It is in 3rd normal form because there is no transitivity.

8. Order :

Order (Blood_Bank_ID, Hospital_ID, Issue_Date, Blood_Type, Blood_Quantity)

- Here this table is in 1st a normal form because every record is atomic.
- This table is in 2nd normal form because there is no partial dependency.
- It is in 3rd normal form because there is no transitivity.
- Here we have composite key = Blood_Bank_ID, Hospital_ID, Issue_Date, Blood_Type
- This table belongs to the order relation because the relation has Many to Many relationship.

9. Patient :

Patient (Hospital_ID , Patient_ID , First_Name, Middle_Name, Last_Name , Sex , Blood_Type , Date_Admitted , Date_Discharged, Contact_Info)

- Here this table is not in 1st a normal form because every record is not atomic.
- Here Contact_Info is a multivalued attribute.
- We can create different table for contact info to reduce it to the 1st normal form.

Patient (Hospital_ID , Patient_ID , First_Name, Middle_Name, Last_Name , Sex , Blood_Type , Date_Admitted , Date_Discharged, Contact_Info)

Patient_Contact (Hospital_ID , Patient_ID, Contact_Info)

- This table is in 2nd normal form because there is no partial dependency.
- It is in 3rd normal form because there is no transitivity.

TABLES AFTER NORMALIZATION

- Check-ups (Vol_ID, First_Name, Middle_Name, Last_Name, Weight, DOB)

- Blood_Bank (Blood_Bank_ID , BB_Name, BB_Location)
- Bank_Blood_Type (Blood_Bank_ID , Blood_Type , Cost)

- Blood_Donation_Camp (Orgaization , Camp_ID , location , Start_date , End_dt, Blood_Bank_ID)
- Camp_Contact (Organization , Camp_ID ,Contact_info)
- Organization_Blood_Bank(Organization, Blood_Bank_ID)

- Nurse (Nurse_ID , First_Name, Middle_Name, Last_Name, DOB , Organization, Camp_ID)
- Nurse_Contact (Nurse_ID ,Contact_info)

- Donor(Donor_ID, First_Name, Middle_Name, Last_Name, DOB, Nurse_ID)
- Donor_Contact (Donor_ID , Contact_info)

- Blood (Donor_ID,Donation_Date, Orgaization , Camp_ID, Blood_Type , Haemoglobin , Red_blood_cells , White_blood_cells , Platelets , Plazma)

- Hospital (Hospital_ID , Hospital_Name, Hos_Address)
- Hospital_Contact (Hospital_ID , Contact_Info)
- Hospital_Email (Hospital_ID , Hos_Email)

- Order (Blood_Bank_ID, Hospital_ID, Issue_Date, Blood_Type, Blood_Quantity)

- Patient (Hospital_ID , Patient_ID , First_Name, Middle_Name, Last_Name , Sex , Blood_Type , Date_Admitted , Date_Discharged)
- Patient_Contact (Hospital_ID , Patient_ID ,Contact_Info)

Creation Of Tables (DDL Commands)

1) CREATE TABLE Check_ups (

Vol_ID INT PRIMARY KEY,

First_Name VARCHAR(50),

Middle_Name VARCHAR(50),

Last_Name VARCHAR(50),

Weight FLOAT,

DOB DATE

);

2) CREATE TABLE Donor (

Donor_ID INT PRIMARY KEY ON DELETE SET NULL,

First_Name VARCHAR(50),

Middle_Name VARCHAR(50),

Last_Name VARCHAR(50),

DOB DATE,

Nurse_ID INT NOT NULL,

FOREIGN KEY (Nurse_ID) REFERENCES Nurse(Nurse_ID)

);

```
3) CREATE TABLE Donor_Contact (
    Donor_ID INT,
    Contact_info VARCHAR(100),
    PRIMARY KEY (Donor_ID),
    FOREIGN KEY (Donor_ID) REFERENCES Donor(Donor_ID)
);


4) CREATE TABLE Nurse (
    Nurse_ID INT PRIMARY KEY ON DELETE SET NULL,
    First_Name VARCHAR(50),
    Middle_Name VARCHAR(50),
    Last_Name VARCHAR(50),
    DOB DATE,
    Organization VARCHAR(100) NOT NULL,
    Camp_ID INT NOT NULL,
    FOREIGN KEY (Camp_ID,Organization) REFERENCES
Blood_Donation_Camp(Camp_ID,Organization)
);
```

```
5) CREATE TABLE Nurse_Contact (
    Nurse_ID INT,
    Contact_info VARCHAR(100),
    PRIMARY KEY (Nurse_ID),
    FOREIGN KEY (Nurse_ID) REFERENCES Nurse(Nurse_ID)
);


6) CREATE TABLE Blood (
    Donor_ID INT NOT NULL,
    Organization VARCHAR(100) NOT NULL,
    Camp_ID INT NOT NULL,
    Blood_Type VARCHAR(5),
    Haemoglobin FLOAT,
    Red_blood_cells FLOAT,
    White_blood_cells FLOAT,
    Platelets FLOAT,
    Plazma FLOAT,
    PRIMARY KEY (Donor_ID),
    FOREIGN KEY (Donor_ID) REFERENCES Donor(Donor_ID),
    FOREIGN KEY (Organization, Camp_ID) REFERENCES
Blood_Donation_Camp(Organization, Camp_ID) );
```

```
7) CREATE TABLE Blood_Donation_Camp (
    Organization VARCHAR(100),
    Camp_ID INT,
    Location VARCHAR(100),
    Start_date DATE,
    End_date DATE,
    Blood_Bank_ID INT,
    PRIMARY KEY (Organization, Camp_ID) ON DELETE SET NULL,
    FOREIGN KEY (Blood_Bank_ID) REFERENCES Blood_Bank(Blood_Bank_ID)
);
```

```
8) CREATE TABLE Camp_Contact (
    Organization VARCHAR(100),
    Camp_ID INT,
    Contact_info VARCHAR(100),
    PRIMARY KEY (Organization, Camp_ID),
    FOREIGN KEY (Organization, Camp_ID) REFERENCES
Blood_Donation_Camp(Organization, Camp_ID)
);
```

```
9) CREATE TABLE Bank_Blood_Type (
    Blood_Bank_ID INT,
    Blood_Type VARCHAR(5),
    Cost DECIMAL(10, 2),
    PRIMARY KEY (Blood_Bank_ID, Blood_Type),
    FOREIGN KEY (Blood_Bank_ID) REFERENCES Blood_Bank(Blood_Bank_ID)
);

10) CREATE TABLE Orders(
    Blood_Bank_ID INT,
    Hospital_ID INT,
    Issue_Date DATE,
    PRIMARY KEY (Blood_Bank_ID, Hospital_ID),
    FOREIGN KEY (Blood_Bank_ID) REFERENCES Blood_Bank(Blood_Bank_ID),
    FOREIGN KEY (Hospital_ID) REFERENCES Hospital(Hospital_ID)
);

11) CREATE TABLE Blood_Bank (
    Blood_Bank_ID INT PRIMARY KEY ON DELETE SET NULL,
    BB_First_Name VARCHAR(50),
```

```
    Middle_Name VARCHAR(50),
    Last_Name VARCHAR(50),
    BB_Location VARCHAR(100)
);
```

12)
```
CREATE TABLE Hospital (
    Hospital_ID INT PRIMARY KEY ON DELETE SET NULL,
    Hospital_First_Name VARCHAR(50),
    Middle_Name VARCHAR(50),
    Last_Name VARCHAR(50),
    Hos_Address VARCHAR(100)
);
```

13)
```
CREATE TABLE Hospital_Contact (
    Hospital_ID INT,
    Contact_Info VARCHAR(100),
    PRIMARY KEY (Hospital_ID),
    FOREIGN KEY (Hospital_ID) REFERENCES Hospital(Hospital_ID)
);
```

```
14) CREATE TABLE Hospital_Email (
    Hospital_ID INT,
    Hos_Email VARCHAR(100),
    PRIMARY KEY (Hospital_ID),
    FOREIGN KEY (Hospital_ID) REFERENCES Hospital(Hospital_ID)
);

15) CREATE TABLE Patient (
    Hospital_ID INT NOT NULL,
    Patient_ID INT,
    First_Name VARCHAR(50),
    Middle_Name VARCHAR(50),
    Last_Name VARCHAR(50),
    Sex CHAR(1),
    Blood_Type VARCHAR(5),
    Date_Admitted DATE,
    Date_Discharged DATE,
    PRIMARY KEY (Hospital_ID, Patient_ID),
    FOREIGN KEY (Hospital_ID) REFERENCES Hospital(Hospital_ID)
);
```

16) CREATE TABLE Patient_Contact (

Hospital_ID INT,

Patient_ID INT,

Contact_Info VARCHAR(100),

PRIMARY KEY (Hospital_ID, Patient_ID),

FOREIGN KEY (Hospital_ID, Patient_ID) REFERENCES Patient(Hospital_ID, Patient_ID)

);

Procedures

1) CREATE OR REPLACE PROCEDURE Check_Blood_Availability

(blood_type IN VARCHAR2)

IS

CURSOR blood_bank_cursor IS

SELECT bb.BB_Name

FROM Blood_Bank bb

JOIN Bank_Blood_Type bbt ON bb.Blood_Bank_ID = bbt.Blood_Bank_ID

WHERE bbt.Blood_Type = blood_type;

blood_bank_name Blood_Bank.BB_Name%TYPE;

blood_found BOOLEAN := FALSE;

BEGIN

FOR blood_bank_rec IN blood_bank_cursor LOOP

DBMS_OUTPUT.PUT_LINE('Blood bank with blood type ' || blood_type || ' is available: ' || blood_bank_rec.BB_Name);

blood_found := TRUE;

END LOOP;

EXCEPTION

```
WHEN NO_DATA_FOUND THEN

DBMS_OUTPUT.PUT_LINE('No blood banks found for blood type ' || blood_type
|| '.');

WHEN TOO_MANY_ROWS THEN

DBMS_OUTPUT.PUT_LINE('Error: Multiple blood banks found for blood type ' ||
blood_type || '.');

WHEN OTHERS THEN

DBMS_OUTPUT.PUT_LINE('An error occurred while checking blood availability.');



END;
```

2) CREATE OR REPLACE PROCEDURE get_blood_counts

```
(
    p_blood_bank_id IN NUMBER,
    p_blood_type    IN VARCHAR2
)
IS
    v_total_count   NUMBER := 0;
    v_camp_id       NUMBER;
    v_organization  VARCHAR2(100);
    v_blood_count   NUMBER;

    CURSOR camp_cursor IS
```

```
        SELECT Camp_ID, Organization

        FROM Blood_Donation_Camp

        WHERE Organization IN (SELECT Organization FROM
Organization_Blood_Bank WHERE Blood_Bank_ID = p_blood_bank_id);



    CURSOR blood_count_cursor (c_camp_id NUMBER, c_organization
VARCHAR2, c_blood_type VARCHAR2) IS

        SELECT COUNT(*) AS blood_count

        FROM Blood

        WHERE Camp_ID = c_camp_id

          AND Organization = c_organization

          AND Blood_Type = c_blood_type;



BEGIN



    FOR rec IN camp_cursor LOOP

        v_camp_id := rec.Camp_ID;

        v_organization := rec.Organization;



        OPEN blood_count_cursor(v_camp_id, v_organization, p_blood_type);

        FETCH blood_count_cursor INTO v_blood_count;



        -- Increment total count with the fetched blood count

        v_total_count := v_total_count + v_blood_count;
```

```
        IF blood_count_cursor%NOTFOUND THEN

            DBMS_OUTPUT.PUT_LINE('No data found for blood type ' ||
p_blood_type || ' in camp ID ' || v_camp_id || ' and organization ' ||
v_organization);

        ELSIF blood_count_cursor%ROWCOUNT > 1 THEN

            DBMS_OUTPUT.PUT_LINE('Error: Multiple rows found for blood type ' ||
p_blood_type || ' in camp ID ' || v_camp_id || ' and organization ' ||
v_organization);

        ELSE

            DBMS_OUTPUT.PUT_LINE('Total blood count for blood type ' ||
p_blood_type || ': ' || v_total_count);

        END IF;



        CLOSE blood_count_cursor;



    END LOOP;



EXCEPTION

    WHEN NO_DATA_FOUND THEN

        DBMS_OUTPUT.PUT_LINE('No data found for blood type ' || p_blood_type
|| ' in camp ID ' || v_camp_id || ' and organization ' || v_organization);

    WHEN TOO_MANY_ROWS THEN

        DBMS_OUTPUT.PUT_LINE('Error: Multiple rows found for blood type ' ||
p_blood_type || ' in camp ID ' || v_camp_id || ' and organization ' ||
v_organization);

    WHEN OTHERS THEN

        DBMS_OUTPUT.PUT_LINE('An error occurred while processing camps.');

END;
```

```
3) CREATE OR REPLACE PROCEDURE percentage_donation
IS
    total_participation INT;
    total_donation INT;
    donation_percentage NUMBER;
BEGIN

    SELECT COUNT(*) INTO total_participation FROM check_ups;

    SELECT COUNT(*) INTO total_donation
    FROM donor d
    JOIN blood b ON b.donor_id = d.donor_id;

    donation_percentage := (total_donation / total_participation) * 100;

    DBMS_OUTPUT.PUT_LINE('Percentage of donations out of total participation
is: ' || donation_percentage);
END;
```

```
4) CREATE OR REPLACE PROCEDURE display_available_blood_bags(
    p_Blood_Type VARCHAR2,
    p_Result OUT SYS_REFCURSOR
) IS
    l_Blood_Bank_ID Bank_Blood_Type.Blood_Bank_ID%TYPE;
    l_Blood_Bank_Name VARCHAR2(100);
    l_Blood_Type Bank_Blood_Type.Blood_Type%TYPE;
    l_Available_Bags NUMBER;
BEGIN
    OPEN p_Result FOR
    SELECT bb.Blood_Bank_ID, bb.BB_Name AS Blood_Bank_Name,
           bbt.Blood_Type, COUNT(*) AS Available_Bags
    FROM Bank_Blood_Type bbt
    JOIN Blood_Bank bb ON bbt.Blood_Bank_ID = bb.Blood_Bank_ID
    WHERE bbt.Blood_Type = p_Blood_Type
    GROUP BY bb.Blood_Bank_ID, bb.BB_Name, bbt.Blood_Type;

    DBMS_OUTPUT.PUT_LINE('Available blood bags for blood type ' ||
p_Blood_Type || ':');
    DBMS_OUTPUT.PUT_LINE('----------------------------------------');
    LOOP
        FETCH p_Result INTO
            l_Blood_Bank_ID,
            l_Blood_Bank_Name,
```

```
        l_Blood_Type,

        l_Available_Bags;

    EXIT WHEN p_Result%NOTFOUND;

    DBMS_OUTPUT.PUT_LINE('Blood Bank ID: ' || l_Blood_Bank_ID || ', Blood
Bank Name: ' || l_Blood_Bank_Name || ', Blood Type: ' || l_Blood_Type || ',
Available Bags: ' || l_Available_Bags);

  END LOOP;



  CLOSE p_Result;

END display_available_blood_bags;
```

Functions

1) CREATE OR REPLACE FUNCTION Duration_Of_Camp(

Organization IN Blood_Donation_Camp.Organization%TYPE,

Camp_ID IN Blood_Donation_Camp.Camp_ID%TYPE

)

RETURN NUMBER

IS

days NUMBER(4);

BEGIN

SELECT End_Date - Start_Date

INTO days

FROM Blood_Donation_Camp

WHERE Organization = Duration_Of_Camp.Organization

AND Camp_ID = Duration_Of_Camp.Camp_ID;

RETURN days;

EXCEPTION

WHEN NO_DATA_FOUND THEN

RETURN NULL;

WHEN OTHERS THEN

RETURN NULL;

END;

2) CREATE OR REPLACE FUNCTION FindAge(date_of_birth IN DATE)

```
RETURN NUMBER IS
BEGIN
   IF date_of_birth IS NULL THEN
      RETURN NULL;
   END IF;

   RETURN ROUND(MONTHS_BETWEEN(SYSDATE, date_of_birth) / 12);
EXCEPTION
   WHEN OTHERS THEN
      RETURN NULL;
END;
```

3) CREATE OR REPLACE FUNCTION get_camp_total_donations(

```
   p_camp_id IN Blood.Camp_ID%TYPE,
   p_organization IN Blood.Organization%TYPE
)
RETURN NUMBER
IS
   v_total_donations NUMBER;
BEGIN
   SELECT COUNT(*)
   INTO v_total_donations
   FROM Blood
```

```
    WHERE Camp_ID = p_camp_id
      AND Organization = p_organization;

    RETURN v_total_donations;
END;
```

4)

```
CREATE OR REPLACE FUNCTION get_donor_total_donations(
    p_donor_id IN Donor.Donor_ID%TYPE
)
RETURN NUMBER
IS
    v_total_donations NUMBER;
BEGIN
    SELECT COUNT(*)
    INTO v_total_donations
    FROM Blood
    WHERE Donor_ID = p_donor_id;

    RETURN v_total_donations;
END;
```

Triggers

1) CREATE OR REPLACE TRIGGER eligibility

BEFORE INSERT ON Donor

FOR EACH ROW

DECLARE

donor_weight DECIMAL(5,2);

donor_date_of_birth DATE;

donor_age INT;

BEGIN

SELECT Weight INTO donor_weight FROM Check_ups WHERE Vol_ID = :new.Donor_ID;

SELECT DOB INTO donor_date_of_birth FROM Check_ups WHERE Vol_ID = :new.Donor_Id;

donor_age := FindAge(donor_date_of_birth);

IF donor_age >= 18 AND donor_age <= 65 AND donor_weight >= 50 THEN

NULL;

ELSE

RAISE_APPLICATION_ERROR(-20000, 'Not eligible to donate blood');

END IF;

END;

/

2) CREATE OR REPLACE TRIGGER prevent_donor_deletion

BEFORE DELETE ON Donor

FOR EACH ROW

DECLARE

v_blood_count NUMBER;

BEGIN

SELECT COUNT(*)

INTO v_blood_count

FROM Blood

WHERE Donor_ID = :OLD.Donor_ID;

IF v_blood_count > 0 THEN

RAISE_APPLICATION_ERROR(-20002, 'Cannot delete Donor record as there are associated Blood records.');

END IF;

END;

/

3) CREATE OR REPLACE TRIGGER enforce_blood_bank_capacity

BEFORE INSERT ON Orders

FOR EACH ROW

DECLARE

v_current_count NUMBER;

v_capacity NUMBER;

```
BEGIN
   SELECT Capacity
   INTO v_capacity
   FROM Blood_Bank
   WHERE Blood_Bank_ID = :NEW.Blood_Bank_ID;

   SELECT COUNT(*)
   INTO v_current_count
   FROM Orders
   WHERE Blood_Bank_ID = :NEW.Blood_Bank_ID;

   IF v_current_count >= v_capacity THEN
      RAISE_APPLICATION_ERROR(-20001, 'Blood bank capacity exceeded.
Cannot insert more orders.');
   END IF;
END;
/
```

4) CREATE OR REPLACE TRIGGER check_issue_date

```
BEFORE INSERT ON Orders
FOR EACH ROW
BEGIN
   IF :NEW.Issue_Date > SYSDATE THEN
      RAISE_APPLICATION_ERROR(-20001, 'Issue date cannot be in the future');
```

```
    END IF;
END;
/


5) CREATE OR REPLACE TRIGGER check_blood_quantity
BEFORE INSERT OR UPDATE ON Orders
FOR EACH ROW
DECLARE
    v_available_quantity NUMBER;
BEGIN
    SELECT COUNT(*)
    INTO v_available_quantity
    FROM Blood b
    JOIN Blood_Bank bb ON b.Organization = bb.BB_Name
    WHERE b.Blood_Type = :NEW.Blood_Type
    AND bb.Blood_Bank_ID = :NEW.Blood_Bank_ID;


    IF v_available_quantity = 0 OR v_available_quantity < :NEW.Blood_Quantity
THEN
        RAISE_APPLICATION_ERROR(-20001, 'Requested quantity exceeds
available blood quantity for the given blood type in the Blood_Bank.');
    END IF;
END;
/
```

```
6) CREATE OR REPLACE TRIGGER check_donor_eligibility

BEFORE INSERT ON Blood

FOR EACH ROW

DECLARE

   v_last_donation_date DATE;

BEGIN

   SELECT MAX(Donation_Date)

   INTO v_last_donation_date

   FROM Blood

   WHERE Donor_ID = :NEW.Donor_ID;



   IF v_last_donation_date IS NOT NULL AND v_last_donation_date > SYSDATE
- 56 THEN

      RAISE_APPLICATION_ERROR(-20001, 'Donor is not eligible to donate blood
within 56 days of their last donation.');

   END IF;

END;

/
```

Packages

1) CREATE OR REPLACE PACKAGE BODY Blood_Bank_Package AS

```
  FUNCTION calculate_available_blood_bags(p_Blood_Bank_ID IN NUMBER)
RETURN NUMBER IS
    v_Available_Bags NUMBER;
  BEGIN
    SELECT COUNT(*)
    INTO v_Available_Bags
    FROM Bank_Blood_Type
    WHERE Blood_Bank_ID = p_Blood_Bank_ID
    GROUP BY Blood_Bank_ID;

    RETURN v_Available_Bags;
  END calculate_available_blood_bags;

  PROCEDURE print_max_blood_banks(p_Blood_Type VARCHAR2) IS
  BEGIN
    FOR r IN blood_bank_cursor(p_Blood_Type) LOOP
```

```
      DBMS_OUTPUT.PUT_LINE('Blood Bank with Max Quantity of ' ||
p_Blood_Type || ': ' || r.Blood_Bank_Name);

    END LOOP;

  END print_max_blood_banks;



END Blood_Bank_Package;




CREATE OR REPLACE PACKAGE Hospital_Package AS


  FUNCTION calculate_avg_length_of_stay(p_Hospital_ID NUMBER) RETURN
NUMBER;


  FUNCTION get_admitted_patient_count(p_Hospital_ID NUMBER, p_Start_Date
DATE, p_End_Date DATE) RETURN NUMBER;


  PROCEDURE discharge_patient(p_Hospital_ID NUMBER, p_Patient_ID
NUMBER, p_Discharge_Date DATE);


END Hospital_Package;
```

2) CREATE OR REPLACE PACKAGE BODY Hospital_Package AS

```
 FUNCTION calculate_avg_length_of_stay(p_Hospital_ID NUMBER) RETURN
NUMBER IS
   v_Total_Length NUMBER := 0;
   v_Total_Patients NUMBER := 0;
 BEGIN
   SELECT SUM(Date_Discharged - Date_Admitted)
   INTO v_Total_Length
   FROM Patient
   WHERE Hospital_ID = p_Hospital_ID
   AND Date_Discharged IS NOT NULL;

   SELECT COUNT(*)
   INTO v_Total_Patients
   FROM Patient
   WHERE Hospital_ID = p_Hospital_ID
   AND Date_Discharged IS NOT NULL;

   IF v_Total_Patients > 0 THEN
     RETURN v_Total_Length / v_Total_Patients;
```

```
    ELSE
      RETURN 0;
    END IF;
  END calculate_avg_length_of_stay;

  FUNCTION get_admitted_patient_count(p_Hospital_ID NUMBER, p_Start_Date
DATE, p_End_Date DATE) RETURN NUMBER IS
    v_Patient_Count NUMBER := 0;
  BEGIN
    SELECT COUNT(*)
    INTO v_Patient_Count
    FROM Patient
    WHERE Hospital_ID = p_Hospital_ID
    AND Date_Admitted BETWEEN p_Start_Date AND p_End_Date;

    RETURN v_Patient_Count;
  END get_admitted_patient_count;

  PROCEDURE discharge_patient(p_Hospital_ID NUMBER, p_Patient_ID
NUMBER, p_Discharge_Date DATE) IS
  BEGIN
```

```
    UPDATE Patient
    SET Date_Discharged = p_Discharge_Date
    WHERE Hospital_ID = p_Hospital_ID
    AND Patient_ID = p_Patient_ID;
  END discharge_patient;


END Hospital_Package;
```

Thank You...