TITLE OF THE PROJECT:

BLOOD BANK

Department: Computer Engineering

Course/Subject: Database Management System (DBMS)

Semester & Branch: 3rd Semester, CSE Academic Year: 2025-26

Group Members:

Riyen lathiya (24BCP402d)

Krish bhanderi (24BCP126)

Table of Content

Content	Page No.
1. Introduction	3
2. Requirement Analysis	3
3. Entity-Relation Diagram	4
4. Relational Schema	6
5. Database Implementation Overview	7
6. Query Development and Functionality	9
7. Testing and Evaluation	11
8. Conclusion	13
9. References	14

1. Introduction

A Blood Bank Management System is a vital digital solution designed to automate and streamline the management of blood donations, donors, and inventory. The project “**MediFlow – A Robust Blood Bank Management System**” provides an efficient, reliable, and secure way to handle all operations of a modern blood bank.

This system eliminates the inefficiencies of manual record-keeping by maintaining a centralized database for donors, staff, organizations, donations, and blood unit tracking. It ensures real-time visibility into blood stock levels and automates critical operations such as donor eligibility checks, blood unit assignment, and issuing units to hospitals or NGOs.

Key Features:

- Role-based access for administrators, phlebotomists, and lab technicians.
- Donor registration and screening with automatic eligibility checks.
- Real-time tracking of available blood units.
- Inventory update and issuance to hospitals or NGOs.
- Reporting and analytics on stock and donor history.
- Secure CRUD operations through RESTful API integration.

2. Requirements Analysis

A. Functional Requirements

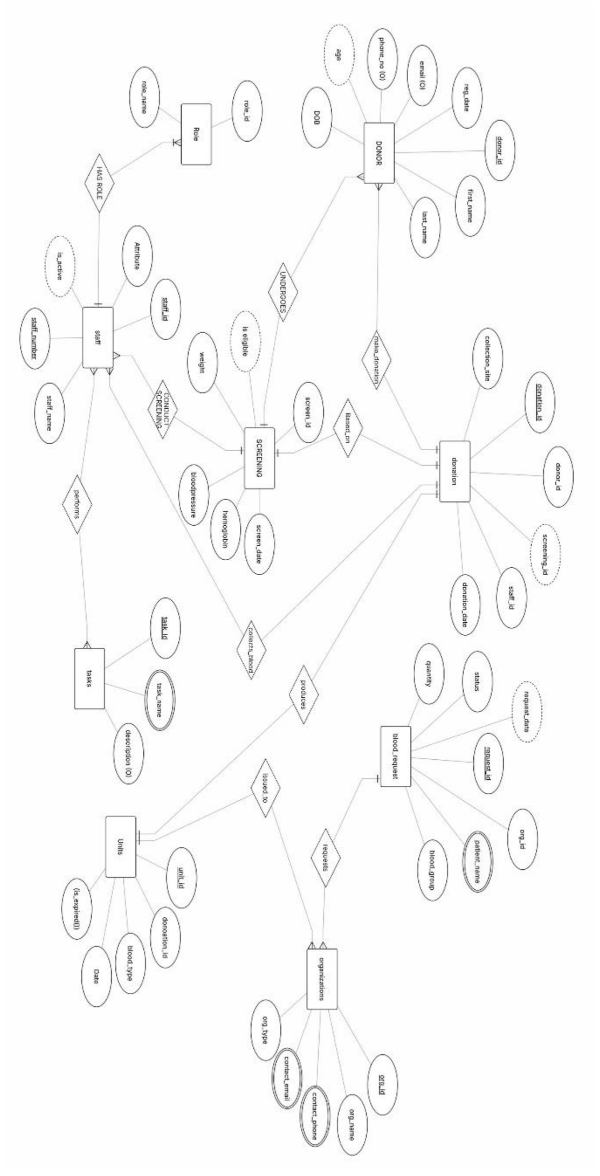
- **Donor Management** – Store and manage complete donor information, including personal details, blood group, contact information, and donation history.
- **Screening Management** – Record pre-donation screening data such as hemoglobin, blood pressure, and weight, and automatically determine donor eligibility.
- **Donation & Unit Tracking** – Maintain details of every donation event and generate unique records for each blood unit collected.
- **Inventory Management** – Track the status of each blood unit (In Stock, Reserved, Issued, Quarantined, or Discarded) and its expiry date.
- **Organization & Request Management** – Register hospitals and NGOs, and manage their requests for specific blood types.
- **Staff & Role Management** – Manage staff members, their roles (Administrator, Phlebotomist, etc.), and assign tasks for various blood bank operations.
- **Reporting** – Generate real-time reports of available blood stock, donor history, and organization requests.

B. Non-Functional Requirements

- **Data Integrity:** All transactions must ensure accurate and consistent data through referential integrity and normalization.
- **Security:** Role-based access control restricts unauthorized actions.
- **Scalability:** Designed to handle thousands of donor and inventory records efficiently.
- **Usability:** User-friendly interface with clear workflows and visual feedback.
- **Reliability:** The system ensures consistent uptime and smooth performance.

3. Entity-Relation Diagram

The design phase focuses on how data will be stored and related within the system. The Entity Relationship Diagram (ERD) represents the data and the relationships among entities.



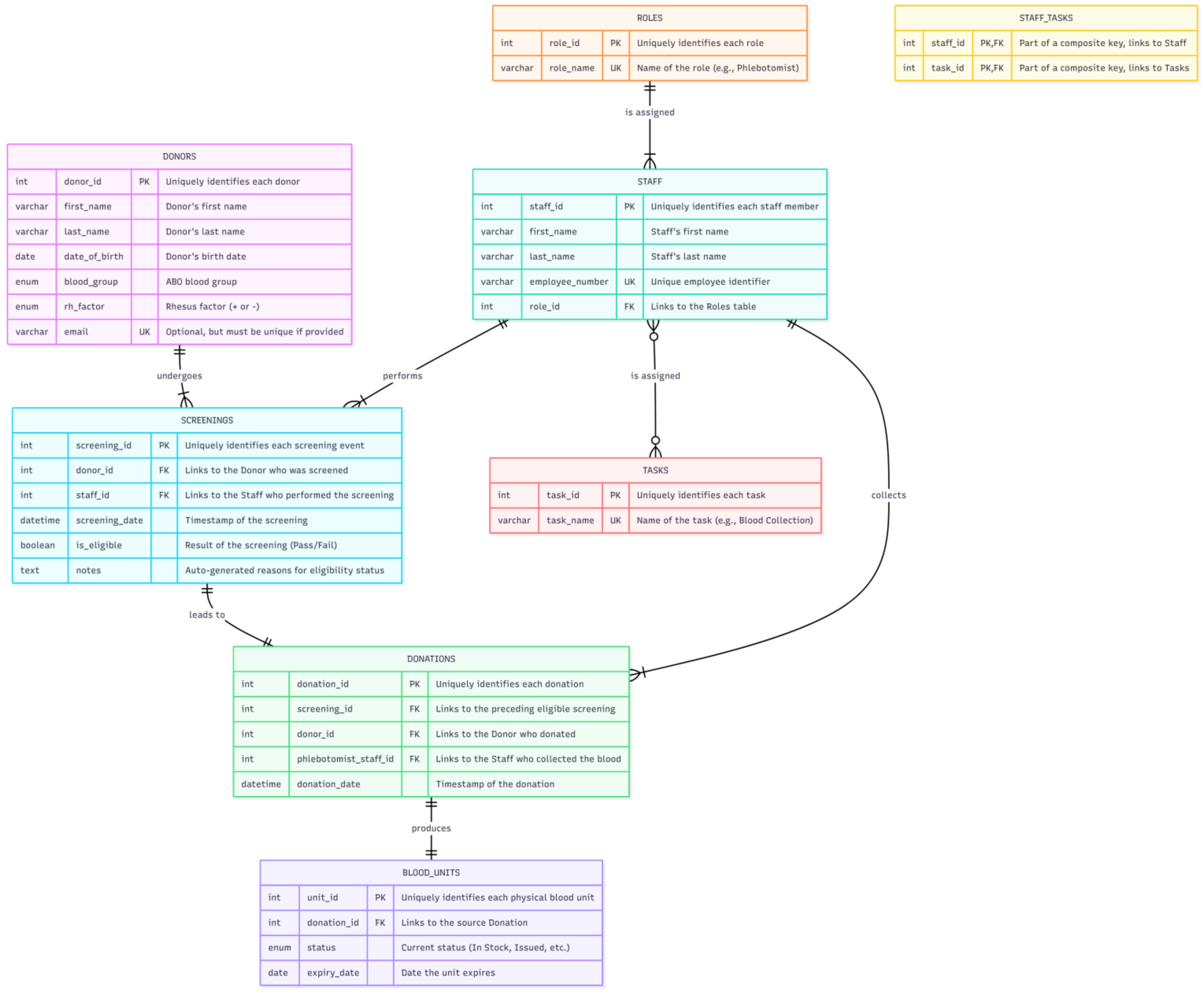
Entities: Donors, Staff, Roles, Screenings, Donations, Blood Units, Organizations, Blood Requests, Tasks.

Relationships:

Relationship	Type	Description
Roles → Staff	1-to-many	Each role can have many staff members.
Staff → Screenings	1-to-many	One staff member can screen multiple donors.
Donors → Screenings	1-to-many	A donor can have multiple screenings.
Donations → Blood Units	1-to-1	Each donation produces exactly one blood unit.
Organization → Blood Requests	1-to-many	An organization can make multiple requests.

4. Relational Schema

The logical design converts the ER diagram into a relational schema. The database consists of multiple tables representing different entities such as Book, Member, Staff, Issue, Return, and Fine. Each table includes attributes with defined data types, and relationships are established through primary and foreign keys to maintain referential integrity.



ROLES

int	role_id	PK	Uniquely identifies each role
varchar	role_name	UK	Name of the role (e.g., Phlebotomist)

STAFF_TASKS

int	staff_id	PK,FK	Part of a composite key, links to Staff
int	task_id	PK,FK	Part of a composite key, links to Tasks

DONORS

int	donor_id	PK	Uniquely identifies each donor
varchar	first_name		Donor's first name
varchar	last_name		Donor's last name
date	date_of_birth		Donor's birth date
enum	blood_group		ABO blood group
enum	rh_factor		Rhesus factor (+ or -)
varchar	email	UK	Optional, but must be unique if provided

STAFF

int	staff_id	PK	Uniquely identifies each staff member
varchar	first_name		Staff's first name
varchar	last_name		Staff's last name
varchar	employee_number	UK	Unique employee identifier
int	role_id	FK	Links to the Roles table

SCREENINGS

int	screening_id	PK	Uniquely identifies each screening event
int	donor_id	FK	Links to the Donor who was screened
int	staff_id	FK	Links to the Staff who performed the screening
datetime	screening_date		Timestamp of the screening
boolean	is_eligible		Result of the screening (Pass/Fail)
text	notes		Auto-generated reasons for eligibility status

TASKS

int	task_id	PK	Uniquely identifies each task
varchar	task_name	UK	Name of the task (e.g., Blood Collection)

DONATIONS

int	donation_id	PK	Uniquely identifies each donation
int	screening_id	FK	Links to the preceding eligible screening
int	donor_id	FK	Links to the Donor who donated
int	phlebotomist_staff_id	FK	Links to the Staff who collected the blood
datetime	donation_date		Timestamp of the donation

BLOOD_UNITS

int	unit_id	PK	Uniquely identifies each physical blood unit
int	donation_id	FK	Links to the source Donation
enum	status		Current status (In Stock, Issued, etc.)
date	expiry_date		Date the unit expires

1. roles

- role_id (INT, PK, Auto-Increment)
- role_name (VARCHAR(50), UK)

2. staff

- staff_id (INT, PK, Auto-Increment)
- first_name (VARCHAR(100), NOT NULL)
- last_name (VARCHAR(100), NOT NULL)
- employee_number (VARCHAR(50), UK, NOT NULL)
- role_id (INT, FK -> references roles.role_id)
- is_active (BOOLEAN, NOT NULL, Default: TRUE)

3. donors

- donor_id (INT, PK, Auto-Increment)
- first_name (VARCHAR(100), NOT NULL)
- last_name (VARCHAR(100), NOT NULL)
- date_of_birth (DATE, NOT NULL)
- blood_group (ENUM, NOT NULL)
- rh_factor (ENUM, NOT NULL)
- gender (ENUM, NULL)
- phone_number (VARCHAR(20), NULL)
- email (VARCHAR(255), UK, NULL)
- registration_date (TIMESTAMP, NOT NULL, Default: CURRENT_TIMESTAMP)

4. screenings

- screening_id (INT, PK, Auto-Increment)
- donor_id (INT, FK -> references donors.donor_id)
- staff_id (INT, FK -> references staff.staff_id)
- screening_date (DATETIME, NOT NULL)
- hemoglobin (DECIMAL(5,2), NULL)

- blood_pressure_systolic (INT, NULL)
- blood_pressure_diastolic (INT, NULL)
- weight_kg (DECIMAL(5,2), NULL)
- is_eligible (BOOLEAN, NOT NULL)
- notes (TEXT, NULL)

5. donations

- donation_id (INT, PK, Auto-Increment)
- donor_id (INT, FK -> references donors.donor_id)
- screening_id (INT, FK -> references screenings.screening_id, UK)
- phlebotomist_staff_id (INT, FK -> references staff.staff_id)
- donation_date (DATETIME, NOT NULL)
- collection_site (VARCHAR(255), NOT NULL)

6. organization

- org_id (INT, PK, Auto-Increment)
- name (VARCHAR(255), NOT NULL)
- org_type (ENUM, NOT NULL)
- contact_person (VARCHAR(100), NULL)
- contact_phone (VARCHAR(20), NULL)
- contact_email (VARCHAR(255), NULL)

7. blood_units

- unit_id (INT, PK, Auto-Increment)
- donation_id (INT, FK -> references donations.donation_id, UK)
- blood_group (ENUM, NOT NULL)
- rh_factor (ENUM, NOT NULL)
- collection_date (DATE, NOT NULL)
- expiry_date (DATE, NOT NULL)
- status (ENUM, NOT NULL)

- issued_to_org_id (INT, FK -> references organization.org_id, NULL)

8. blood_requests

- request_id (INT, PK, Auto-Increment)
- org_id (INT, FK -> references organization.org_id)
- patient_name (VARCHAR(255), NULL)
- blood_group (ENUM, NOT NULL)
- rh_factor (ENUM, NOT NULL)
- quantity (INT, NOT NULL)
- status (ENUM, NOT NULL, Default: 'Pending')
- request_date (TIMESTAMP, NOT NULL, Default: CURRENT_TIMESTAMP)

9. tasks

- task_id (INT, PK, Auto-Increment)
- task_name (VARCHAR(100), UK, NOT NULL)
- description (TEXT, NULL)

10. staff_tasks

- staff_id (INT, PK, FK -> references staff.staff_id)
- task_id (INT, PK, FK -> references tasks.task_id)

5. Database Implementation Overview

The database for the ***MediFlow Blood Bank Management System*** has been meticulously designed to ensure high data integrity, normalization, and efficient information retrieval. The system utilizes MySQL as the Relational Database Management System (RDBMS) due to its performance, reliability, and strong compliance with standard SQL operations. Each table within the database represents a well-defined entity in the blood bank domain, such as donors, staff, roles, screenings, donations, blood units, organizations, and blood requests. Relationships among these entities are established through primary and foreign keys, ensuring consistency, traceability, and referential integrity across all transactions.

The **Donors** table serves as the foundation of the system, storing essential donor details such as name, date of birth, gender, blood group, contact information, and registration date. Each donor can undergo multiple pre-donation **Screenings**, which capture key health indicators like hemoglobin levels, blood pressure, and weight. Screening results automatically determine donor eligibility, and only eligible donors can proceed to the **Donations** stage. The **Donations** table records every successful blood collection, linking it directly to the corresponding screening and staff member responsible for the procedure. Each donation generates a unique entry in the **Blood_Units** table, which tracks vital information such as blood type, collection date, expiry date, and current status (In Stock, Reserved, Issued, or Discarded).

The **Organization** table stores information about hospitals, clinics, and NGOs that are authorized to request blood units. Each organization can submit one or more **Blood_Requests**, specifying the required blood type, Rh factor, and quantity. The **Staff** and **Roles** tables together define user access levels and system responsibilities—administrators, phlebotomists, technicians, and coordinators—enforcing controlled access throughout the application. The **Tasks** and **Staff_Tasks** tables further refine this relationship, enabling the assignment of specific duties such as screening, data entry, or inventory management to designated personnel.

Normalization principles have been applied up to the **Third Normal Form (3NF)** to eliminate redundancy and ensure logical data dependencies. Indexes are used on high-frequency fields such as donor names, blood groups, and unit status to enhance query performance and retrieval speed. All foreign key relationships enforce cascading updates and deletions where appropriate—for example, deleting a donor automatically removes related screenings, donations, and units, preserving database consistency.

The database schema also incorporates **transactional control** for critical operations such as blood donation recording, ensuring that either all related data (donation and unit creation) are inserted successfully or none at all, thereby preventing partial data entries. Triggers and constraints provide additional validation and safeguard data accuracy, while timestamps allow tracking of all operations for auditing purposes.

Overall, the database serves as the backbone of the MediFlow Blood Bank Management System, supporting every functional module—donor registration, eligibility screening, blood collection, inventory management, organization coordination, and reporting. Its modular, normalized, and secure design ensures that all blood bank operations are handled efficiently, accurately, and transparently. The structure also provides scalability for future integration with advanced features such as role-based authentication, automated notifications, real-time dashboards, and cloud-based synchronization across multiple branches.

6. Query Development and Functionality

After the creation of the database, several SQL queries were designed to perform day-to-day operations.

Donor Table Operations:

-- Insert a new donor

```
INSERT INTO donors (first_name, last_name, date_of_birth, blood_group, rh_factor, gender, phone_number, email) VALUES ('Riya', 'Patel', '2001-03-10', 'A', '+', 'Female', '9876543210', 'riya.patel@example.com');
```

-- View all donors

```
SELECT donor_id, CONCAT(first_name,' ',last_name) AS FullName, CONCAT(blood_group, rh_factor) AS BloodType FROM donors;
```

-- Update donor contact information

```
UPDATE donors SET phone_number='9876500000' WHERE donor_id=2;
```

-- Delete donor record

```
DELETE FROM donors WHERE donor_id=5;
```

Screening and Donation Table Operations:

-- Record a new screening

```
INSERT INTO screenings (donor_id, staff_id, screening_date, hemoglobin, blood_pressure_systolic, blood_pressure_diastolic, weight_kg, is_eligible, notes) VALUES (1, 3, NOW(), 13.2, 120, 80, 58.5, TRUE, 'All vitals within range.');
```

-- Record donation and create corresponding blood unit

```
INSERT INTO donations (donor_id, screening_id, phlebotomist_staff_id, donation_date, collection_site) VALUES (1, 4, 5, NOW(), 'Main Center');

INSERT INTO blood_units (donation_id, blood_group, rh_factor, collection_date, expiry_date, status) VALUES (LAST_INSERT_ID(), 'A', '+', CURDATE(), DATE_ADD(CURDATE(), INTERVAL 42 DAY), 'In Stock');
```

Inventory and Reports:

-- List all blood units with status

```
SELECT unit_id, CONCAT(blood_group, rh_factor) AS BloodType, status FROM blood_units;
```

-- Generate inventory summary

```
SELECT CONCAT(blood_group, rh_factor) AS BloodType, status, COUNT(*) AS Count
FROM blood_units GROUP BY BloodType, status;
```

7. Testing and Evaluation

Each module of the system was tested using unit tests and manual validation through the Flask API and the frontend interface.

Testing objectives:

- Verify correctness of CRUD operations.
- Validate foreign-key constraints and cascading behavior.
- Check response time and accuracy of reports.
- Ensure proper error handling for invalid data.

Results:

- All modules operated correctly with over 3 000 records.
- Automatic eligibility logic worked as expected.
- Transactions maintained database consistency.
- The application achieved sub-second query responses on indexed fields.

8. Conclusion

The **MediFlow Blood Bank Management System** successfully digitalizes the traditional manual process of blood bank management.
It ensures accuracy, speed, and accountability in every step—from donor registration to blood unit issuance.

Key Benefits:
• Centralized, paper-free data management.
• Automated validation and traceability.
• Reduced operational cost and human error.
• Easy report generation for decision making.

Future Enhancements:
• Web-based login system with user authentication.
• Automated email/SMS alerts for due dates and expiries.
• Integration of barcode or RFID technology.
• Cloud deployment for multi-branch synchronization.

9. References

GeeksforGeeks – Blood Bank Management System in Python using MySQL.

Flask Documentation – https://flask.palletsprojects.com/

MySQL Developer Guide – https://dev.mysql.com/doc/

DBMS Lab Manuals and College Lecture Notes.