Blood Bank Management System

UCS310: Database Management Systems

Submitted By:

Arianna Vohra (102303934)

Aaditi Verma (102303603)

Ananya Kumar (102303598)

Devansh Wadhwani (102303631)

Sub-Group: 2C43

Submitted To:

Prof. Ananya Kaim

Department of Computer Engineering and Technology

Thapar Institute of Engineering and Technology

(Deemed to be University), Patiala, Punjab, India

Session: January-May, 2025

Index

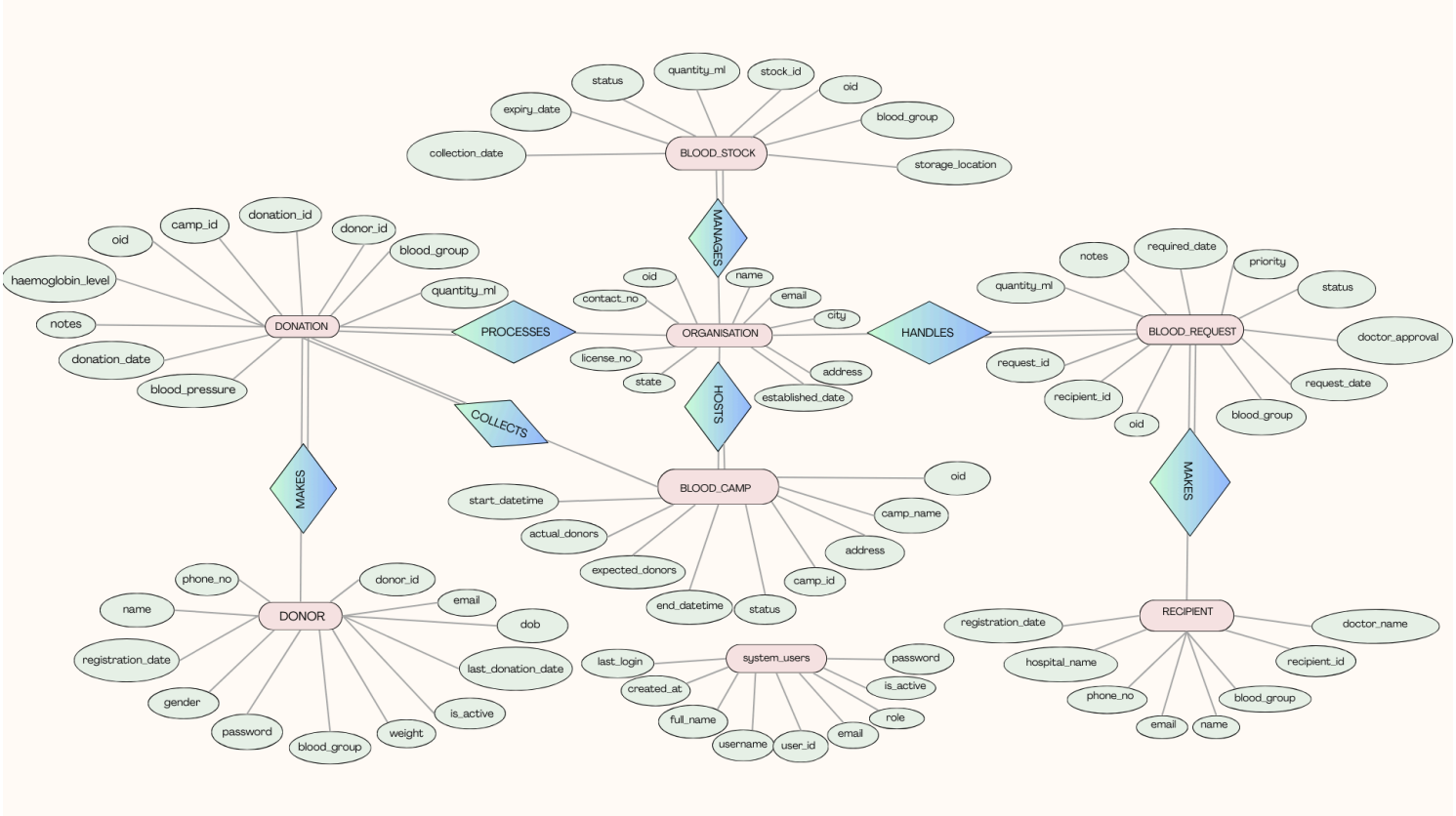
Problem Statement

The management of blood banks involves handling vast amounts of data related to donors, blood groups, and patients in need. Manual record-keeping is time-consuming, prone to errors, and inefficient, leading to delays in blood distribution and potential mismanagement of resources. There is a critical need for an automated system that can efficiently collect, store, and retrieve donor and patient information, ensuring timely access to blood supplies.

The Blood Bank Management System aims to address these challenges by providing a Python-based database solution. This system will streamline the process of donor registration, blood grouping, and patient requests, improving accuracy and operational efficiency.

By implementing this system, blood banks can ensure better organization of data, faster response times, and improved service delivery to patients in need of blood transfusions.

ER Diagram



Entity and Attribute Description

1. **DONOR**: Represents individuals who donate blood.
    - donor_id – Primary key; unique identifier for each donor
    - name – Donor's full name
    - phone_no – Contact number
    - registration_date – Date of donor registration
    - gender – Gender of the donor
    - password – Account password
    - blood_group – Donor's blood type
    - weight – Donor’s body weight
    - email – Email address
    - dob – Date of birth
    - last_donation_date – Last date donor gave blood

- is_active – Flag to indicate if donor is currently active

2. **DONATION**: Records each instance of blood donation made by donors.

- donation_id – Primary key; unique identifier for each donation
- donor_id – Foreign key; references donor_id in DONOR
- camp_id – Foreign key; references camp_id in BLOOD_CAMP
- blood_group – Type of blood donated
- quantity_ml – Amount of blood in milliliters
- donation_date – Date of donation
- blood_pressure – Blood pressure during donation
- haemoglobin_level – Hemoglobin level
- notes – Additional notes
- oid – Foreign key; references oid in ORGANISATION

3. BLOOD_CAMP: Details of organized blood donation camps.

Attributes:

- camp_id – Primary key; unique identifier for each camp
- oid – Foreign key; references oid in ORGANISATION
- camp_name – Name of the blood camp
- address – Location of the camp
- start_datetime – Start date and time of the camp
- end_datetime – End date and time
- expected_donors – Number of expected donors
- actual_donors – Number of actual donors
- status – Status of the camp

4. ORGANISATION: Organizations responsible for managing camps and blood operations.

- oid – Primary key; unique identifier for each organization
- name – Name of the organization

- contact_no – Contact phone number
- email – Email address
- city – City of operation
- state – State
- address – Address
- license_no – Operating license number
- established_date – Date of establishment

5. BLOOD_STOCK: Represents available processed blood inventory.

- stock_id – Primary key; unique identifier for blood stock record
- oid – Foreign key; references oid in ORGANISATION
- blood_group – Blood type
- quantity_ml – Quantity in milliliters
- collection_date – Date collected
- expiry_date – Expiry date
- storage_location – Location of storage
- status – Current availability status

6. BLOOD_REQUEST: Records of blood demand by recipients.

- request_id – Primary key; unique identifier for each request
- recipient_id – Foreign key; references recipient_id in RECIPIENT
- oid – Foreign key; references oid in ORGANISATION
- blood_group – Requested blood group
- quantity_ml – Requested quantity
- request_date – Date of request
- required_date – Date needed
- doctor_approval – Doctor's approval
- priority – Priority level
- status – Request status

- notes – Additional information

7. RECIPIENT: Individuals or patients who request blood.

- recipient_id – Primary key; unique identifier for each recipient
- name – Name of the recipient
- phone_no – Phone number
- email – Email address
- hospital_name – Hospital name
- blood_group – Blood group
- doctor_name – Doctor in charge
- registration_date – Registration date

8. system_users: System-level users such as admins or staff.

- user_id – Primary key; unique identifier for each user
- username – Login username
- full_name – Full name
- email – Email address
- created_at – Account creation timestamp
- last_login – Last login timestamp
- password – Account password
- is_active – Whether account is active
- role – Role (e.g., admin, staff)

ER To Table

CREATING TABLES

```
def create_tables():
    try:
        mycursor.execute("DROP TABLE IF EXISTS blood_request")
        mycursor.execute("DROP TABLE IF EXISTS donation")
        mycursor.execute("DROP TABLE IF EXISTS blood_stock")
        mycursor.execute("DROP TABLE IF EXISTS blood_camp")
        mycursor.execute("DROP TABLE IF EXISTS recipient")
        mycursor.execute("DROP TABLE IF EXISTS donor")
        mycursor.execute("DROP TABLE IF EXISTS organisation")

        mycursor.execute("""
        CREATE TABLE organisation(
            oid INT PRIMARY KEY AUTO_INCREMENT,
            name VARCHAR(50) NOT NULL,
            contact_no VARCHAR(15) NOT NULL,
            email VARCHAR(50) UNIQUE,
            license_no VARCHAR(20) UNIQUE,
            state VARCHAR(20),
            address VARCHAR(100),
            city VARCHAR(20),
            established_date DATE,
            CONSTRAINT chk_email CHECK (email LIKE '%@%.%')
        )""")

        mycursor.execute("""
        CREATE TABLE donor(
```

```
    donor_id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(50) NOT NULL,
    phone_no VARCHAR(15) NOT NULL UNIQUE,
    email VARCHAR(50) UNIQUE,
    dob DATE NOT NULL,
    password VARCHAR(100) NOT NULL,
    gender ENUM('Male','Female','Other') NOT NULL,
    blood_group ENUM('A+','A-','B+','B-','AB+','AB-','O+','O-') NOT NULL,
    weight DECIMAL(5,2) CHECK (weight >= 50),
    last_donation_date DATE,
    is_active BOOLEAN DEFAULT TRUE,
    registration_date DATETIME DEFAULT CURRENT_TIMESTAMP
)
""")

mycursor.execute("""
CREATE TABLE recipient(
    recipient_id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(50) NOT NULL,
    phone_no VARCHAR(15) NOT NULL UNIQUE,
    email VARCHAR(50) UNIQUE,
    hospital_name VARCHAR(50),
    doctor_name VARCHAR(50),
    registration_date DATETIME DEFAULT CURRENT_TIMESTAMP
)""")

mycursor.execute("""
CREATE TABLE blood_camp(
    camp_id INT PRIMARY KEY AUTO_INCREMENT,
```

```
    oid INT NOT NULL,
    camp_name VARCHAR(50) NOT NULL,
    start_datetime DATETIME NOT NULL,
    end_datetime DATETIME NOT NULL,
    address VARCHAR(100) NOT NULL,
    expected_donors INT,
    actual_donors INT DEFAULT 0,
    status ENUM('Planned','Ongoing','Completed','Cancelled'),
    FOREIGN KEY (oid) REFERENCES organisation(oid),
    CONSTRAINT chk_dates CHECK (end_datetime > start_datetime)
)""")

mycursor.execute("""
CREATE TABLE blood_stock(
    stock_id INT PRIMARY KEY AUTO_INCREMENT,
    oid INT NOT NULL,
    blood_group ENUM('A+','A-','B+','B-','AB+','AB-','O+','O-') NOT NULL,
    quantity_ml INT NOT NULL CHECK (quantity_ml >= 0),
    collection_date DATE NOT NULL,
    expiry_date DATE NOT NULL,
    storage_location VARCHAR(20),
    status ENUM('Available','Reserved','Expired','Used'),
    FOREIGN KEY (oid) REFERENCES organisation(oid),
    CONSTRAINT chk_expiry CHECK (expiry_date > collection_date)
)""")

mycursor.execute("""
CREATE TABLE donation(
    donation_id INT PRIMARY KEY AUTO_INCREMENT,
```

```
    donor_id INT NOT NULL,
    camp_id INT,
    oid INT NOT NULL,
    donation_date DATETIME DEFAULT CURRENT_TIMESTAMP,
    blood_group ENUM('A+','A-','B+','B-','AB+','AB-','O+','O-') NOT NULL,
    quantity_ml INT NOT NULL CHECK (quantity_ml BETWEEN 350 AND 450),
    hemoglobin_level DECIMAL(3,1) NOT NULL,
    blood_pressure VARCHAR(10) NOT NULL,
    notes TEXT,
    FOREIGN KEY (donor_id) REFERENCES donor(donor_id),
    FOREIGN KEY (camp_id) REFERENCES blood_camp(camp_id),
    FOREIGN KEY (oid) REFERENCES organisation(oid)
)""")

mycursor.execute("""
CREATE TABLE blood_request(
    request_id INT PRIMARY KEY AUTO_INCREMENT,
    recipient_id INT NOT NULL,
    oid INT NOT NULL,
    blood_group ENUM('A+','A-','B+','B-','AB+','AB-','O+','O-') NOT NULL,
    quantity_ml INT NOT NULL CHECK (quantity_ml > 0),
    request_date DATETIME DEFAULT CURRENT_TIMESTAMP,
    required_date DATE NOT NULL,
    priority ENUM('Emergency','High','Normal'),
    status ENUM('Pending','Approved','Rejected','Fulfilled'),
    doctor_approval VARCHAR(50),
    notes TEXT,
    FOREIGN KEY (recipient_id) REFERENCES recipient(recipient_id),
    FOREIGN KEY (oid) REFERENCES organisation(oid))""")
```

```
mycursor.execute("""
    CREATE TABLE system_users (
        user_id INT PRIMARY KEY AUTO_INCREMENT,
        username VARCHAR(30) UNIQUE NOT NULL,
        password VARCHAR(100) NOT NULL,
        full_name VARCHAR(50) NOT NULL,
        role ENUM('Admin','Staff','Volunteer') NOT NULL,
        email VARCHAR(50) UNIQUE,
        last_login DATETIME,
        is_active BOOLEAN DEFAULT TRUE,
        created_at DATETIME DEFAULT CURRENT_TIMESTAMP
    )
    """)
    print("8 tables created successfully")
```

SNAPSHOT:

```
def create_tables():
    try:
        # Drop tables if they exist (in correct order to respect foreign keys)
        mycursor.execute("DROP TABLE IF EXISTS blood_request")
        mycursor.execute("DROP TABLE IF EXISTS donation")
        mycursor.execute("DROP TABLE IF EXISTS blood_stock")
        mycursor.execute("DROP TABLE IF EXISTS blood_camp")
        mycursor.execute("DROP TABLE IF EXISTS recipient")
        mycursor.execute("DROP TABLE IF EXISTS donor")
        mycursor.execute("DROP TABLE IF EXISTS organisation")

        # Create tables
        mycursor.execute("""
        CREATE TABLE organisation(
            oid INT PRIMARY KEY AUTO_INCREMENT,
            name VARCHAR(50) NOT NULL,
            contact_no VARCHAR(15) NOT NULL,
            email VARCHAR(50) UNIQUE,
            license_no VARCHAR(20) UNIQUE,
            state VARCHAR(20),
            address VARCHAR(100),
            city VARCHAR(20),
            established_date DATE,
            CONSTRAINT chk_email CHECK (email LIKE '%@%.%')
        )""")

        mycursor.execute("""
        CREATE TABLE donor(
            donor_id INT PRIMARY KEY AUTO_INCREMENT,
            name VARCHAR(50) NOT NULL,
            phone_no VARCHAR(15) NOT NULL UNIQUE,
            email VARCHAR(50) UNIQUE,
            dob DATE NOT NULL,
            password VARCHAR(100) NOT NULL,
            gender ENUM('Male','Female','Other') NOT NULL,
            blood_group ENUM('A+','A-','B+','B-','AB+','AB-','O+','O-') NOT NULL,
            weight DECIMAL(5,2) CHECK (weight >= 50),
            last_donation_date DATE,
            is_active BOOLEAN DEFAULT TRUE,
            registration_date DATETIME DEFAULT CURRENT_TIMESTAMP
```

```
        registration_date DATETIME DEFAULT CURRENT_TIMESTAMP
    )
    """)

    mycursor.execute("""
    CREATE TABLE recipient(
        recipient_id INT PRIMARY KEY AUTO_INCREMENT,
        name VARCHAR(50) NOT NULL,
        phone_no VARCHAR(15) NOT NULL UNIQUE,
        email VARCHAR(50) UNIQUE,
        hospital_name VARCHAR(50),
        doctor_name VARCHAR(50),
        registration_date DATETIME DEFAULT CURRENT_TIMESTAMP
    )""")

    mycursor.execute("""
    CREATE TABLE blood_camp(
        camp_id INT PRIMARY KEY AUTO_INCREMENT,
        oid INT NOT NULL,
        camp_name VARCHAR(50) NOT NULL,
        start_datetime DATETIME NOT NULL,
        end_datetime DATETIME NOT NULL,
        address VARCHAR(100) NOT NULL,
        expected_donors INT,
        actual_donors INT DEFAULT 0,
        status ENUM('Planned','Ongoing','Completed','Cancelled'),
        FOREIGN KEY (oid) REFERENCES organisation(oid),
        CONSTRAINT chk_dates CHECK (end_datetime > start_datetime)
    )""")

    mycursor.execute("""
    CREATE TABLE blood_stock(
        stock_id INT PRIMARY KEY AUTO_INCREMENT,
        oid INT NOT NULL,
        blood_group ENUM('A+','A-','B+','B-','AB+','AB-','O+','O-') NOT NULL,
        quantity_ml INT NOT NULL CHECK (quantity_ml >= 0),
        collection_date DATE NOT NULL,
        expiry_date DATE NOT NULL,
        storage_location VARCHAR(20),
        status ENUM('Available','Reserved','Expired','Used'),
```

```
        status ENUM('Available','Reserved','Expired','Used'),
        FOREIGN KEY (oid) REFERENCES organisation(oid),
        CONSTRAINT chk_expiry CHECK (expiry_date > collection_date)
    )""")

    mycursor.execute("""
    CREATE TABLE donation(
        donation_id INT PRIMARY KEY AUTO_INCREMENT,
        donor_id INT NOT NULL,
        camp_id INT,
        oid INT NOT NULL,
        donation_date DATETIME DEFAULT CURRENT_TIMESTAMP,
        blood_group ENUM('A+','A-','B+','B-','AB+','AB-','O+','O-') NOT NULL,
        quantity_ml INT NOT NULL CHECK (quantity_ml BETWEEN 350 AND 450),
        hemoglobin_level DECIMAL(3,1) NOT NULL,
        blood_pressure VARCHAR(10) NOT NULL,
        notes TEXT,
        FOREIGN KEY (donor_id) REFERENCES donor(donor_id),
        FOREIGN KEY (camp_id) REFERENCES blood_camp(camp_id),
        FOREIGN KEY (oid) REFERENCES organisation(oid)
    )""")

    mycursor.execute("""
    CREATE TABLE blood_request(
        request_id INT PRIMARY KEY AUTO_INCREMENT,
        recipient_id INT NOT NULL,
        oid INT NOT NULL,
        blood_group ENUM('A+','A-','B+','B-','AB+','AB-','O+','O-') NOT NULL,
        quantity_ml INT NOT NULL CHECK (quantity_ml > 0),
        request_date DATETIME DEFAULT CURRENT_TIMESTAMP,
        required_date DATE NOT NULL,
        priority ENUM('Emergency','High','Normal'),
        status ENUM('Pending','Approved','Rejected','Fulfilled'),
        doctor_approval VARCHAR(50),
        notes TEXT,
        FOREIGN KEY (recipient_id) REFERENCES recipient(recipient_id),
        FOREIGN KEY (oid) REFERENCES organisation(oid)
    )""")

    mycursor.execute("""
```

```
    mycursor.execute("""
    CREATE TABLE system_users (
        user_id INT PRIMARY KEY AUTO_INCREMENT,
        username VARCHAR(30) UNIQUE NOT NULL,
        password VARCHAR(100) NOT NULL,
        full_name VARCHAR(50) NOT NULL,
        role ENUM('Admin','Staff','Volunteer') NOT NULL,
        email VARCHAR(50) UNIQUE,
        last_login DATETIME,
        is_active BOOLEAN DEFAULT TRUE,
        created_at DATETIME DEFAULT CURRENT_TIMESTAMP
    )
    """)

    print("8 tables created successfully")
except mysql.connector.Error as err:
    print(f"Error creating tables: {err}")
```

Normalization

1. **First Normal Form (1NF)**

All attributes must contain only atomic (indivisible) values. No repeating groups or arrays.

2. **Second Normal Form (2NF)**

Must be in 1NF, and all non-prime attributes must be fully functionally dependent on the entire primary key. Eliminates partial dependencies (i.e., dependencies on part of a composite key).

3. **Third Normal Form (3NF)**

Must be in 2NF, and no transitive dependency exists (i.e., non-prime attributes should not depend on other non-prime attributes). Eliminates transitive dependencies.

4. **Boyce-Codd Normal Form (BCNF)**

Must be in 3NF, and for every functional dependency X → Y, X must be a super key. Handles special cases of 3NF where anomalies can still exist.

5. **Fourth Normal Form (4NF)**

Must be in BCNF, and there should be no multi-valued dependencies.

6. **Fifth Normal Form (5NF)**

Must be in 4NF, and every join dependency should be implied by candidate keys.

1. DONOR (donor_id, name, phone_no, registration_date, gender, password, blood_group, weight, email, dob, last_donation_date, is_active)

1NF	All fields are atomic (e.g., single `email`, `phone_no`, etc.) and `donor_id` is a unique identifier.
2NF	All attributes depend fully on the primary key `donor_id`.
3NF	No transitive dependencies (e.g., `blood_group` is not dependent on `name`, only on `donor_id`).

2. DONATION (donation_id, donor_id, camp_id, blood_group, quantity_ml, donation_date, blood_pressure, haemoglobin_level, notes, oid)

1NF	All fields are atomic.
2NF	All non-key attributes are fully functionally dependent on donation_id.
3NF	No transitive dependencies among fields like blood_pressure, notes, etc.

3. BLOOD_CAMP (camp_id, oid, camp_name, address, start_datetime, end_datetime, expected_donors, actual_donors, status)

1NF	All attributes are atomic.
2NF	All attributes depend on camp_id, the primary key.
3NF	No attributes depend on other non-key attributes.

4. ORGANISATION (oid, name, contact_no, email, city, state, address, license_no, established_date)

1NF	All fields are atomic.
2NF	All attributes depend only on oid.
3NF	No attribute is transitively dependent on oid.

5. BLOOD_STOCK (stock_id, oid, blood_group, quantity_ml, collection_date, expiry_date, storage_location, status)

1NF	All attributes are atomic and stored individually.
2NF	All attributes depend fully on stock_id.
3NF	No transitive dependencies.

6. BLOOD_REQUEST (request_id, recipient_id, oid, blood_group, quantity_ml, request_date, required_date, doctor_approval, priority, status, notes)

1NF	Attributes are atomic (e.g., status, priority).

2NF	Fully functionally dependent on request_id.
3NF	No transitive dependency (e.g., doctor_approval is not derived from recipient_id directly).

7. RECIPIENT (recipient_id, name, phone_no, email, hospital_name, blood_group, doctor_name, registration_date)

1NF	Atomic fields (e.g., no list of phone numbers).
2NF	All attributes depend on recipient_id.
3NF	No non-key attribute depends on another non-key attribute.

8. system_users (user_id, username, full_name, email, created_at, last_login, password, is_active, role)

1NF	Atomic fields only, e.g., one username, one email.
2NF	All fields depend solely on user_id.
3NF	No transitive dependencies (e.g., role is not derived from email).

PL SQL Snapshots

1. **check_blood_availability**: The check_blood_availability procedure verifies whether a requested blood type and quantity are available in inventory. It sums all non-expired, available stock of the specified blood group and compares it against the requested quantity. The procedure returns a clear message indicating either availability ("Available: X ml") or insufficiency ("Insufficient. Available: X ml") based on the inventory check.

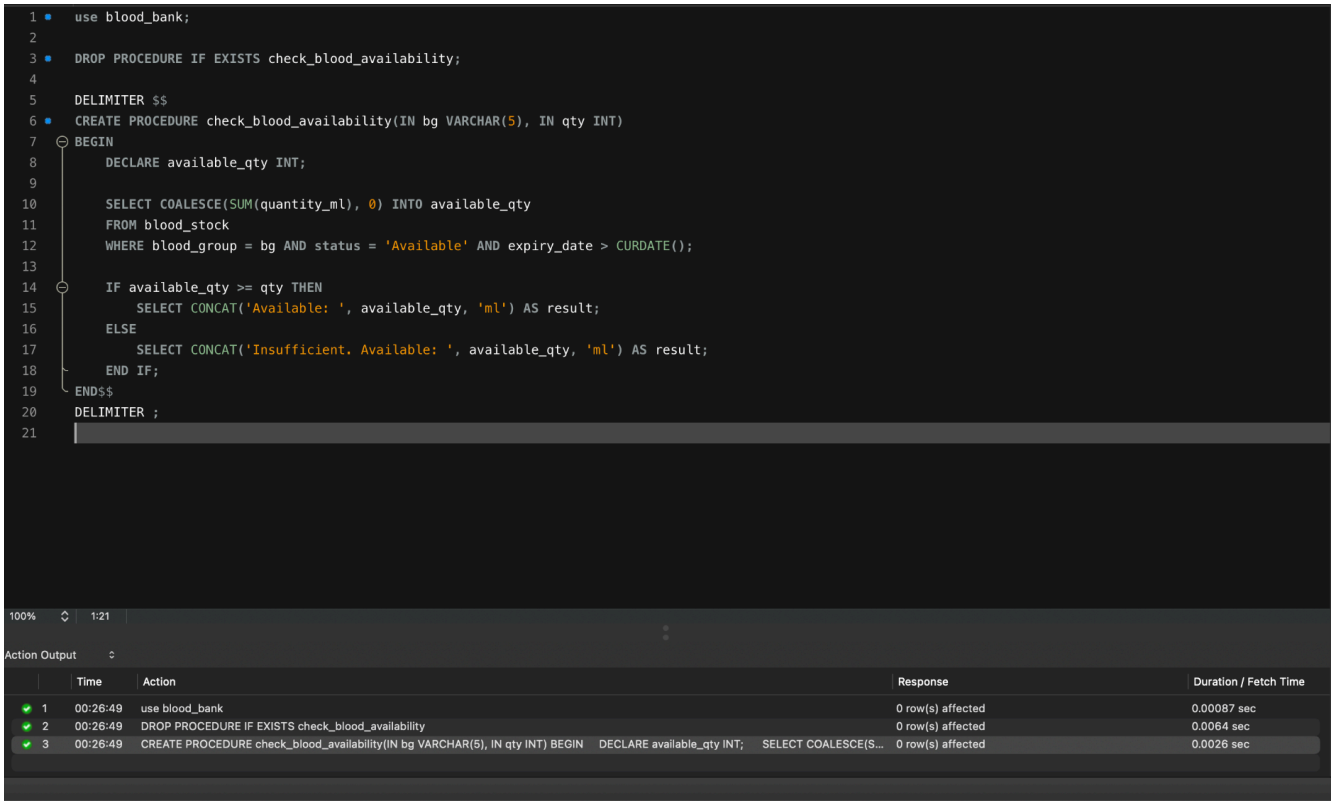
CODE:

```
CREATE PROCEDURE check_blood_availability(IN bg VARCHAR(5), IN qty INT)
BEGIN
   DECLARE available_qty INT;

   SELECT COALESCE(SUM(quantity_ml), 0) INTO available_qty
   FROM blood_stock
   WHERE blood_group = bg AND status = 'Available' AND expiry_date > CURDATE();

   IF available_qty >= qty THEN
      SELECT CONCAT('Available: ', available_qty, 'ml') AS result;
   ELSE
      SELECT CONCAT('Insufficient. Available: ', available_qty, 'ml') AS result;
   END IF;
END
```

SNAPSHOT:



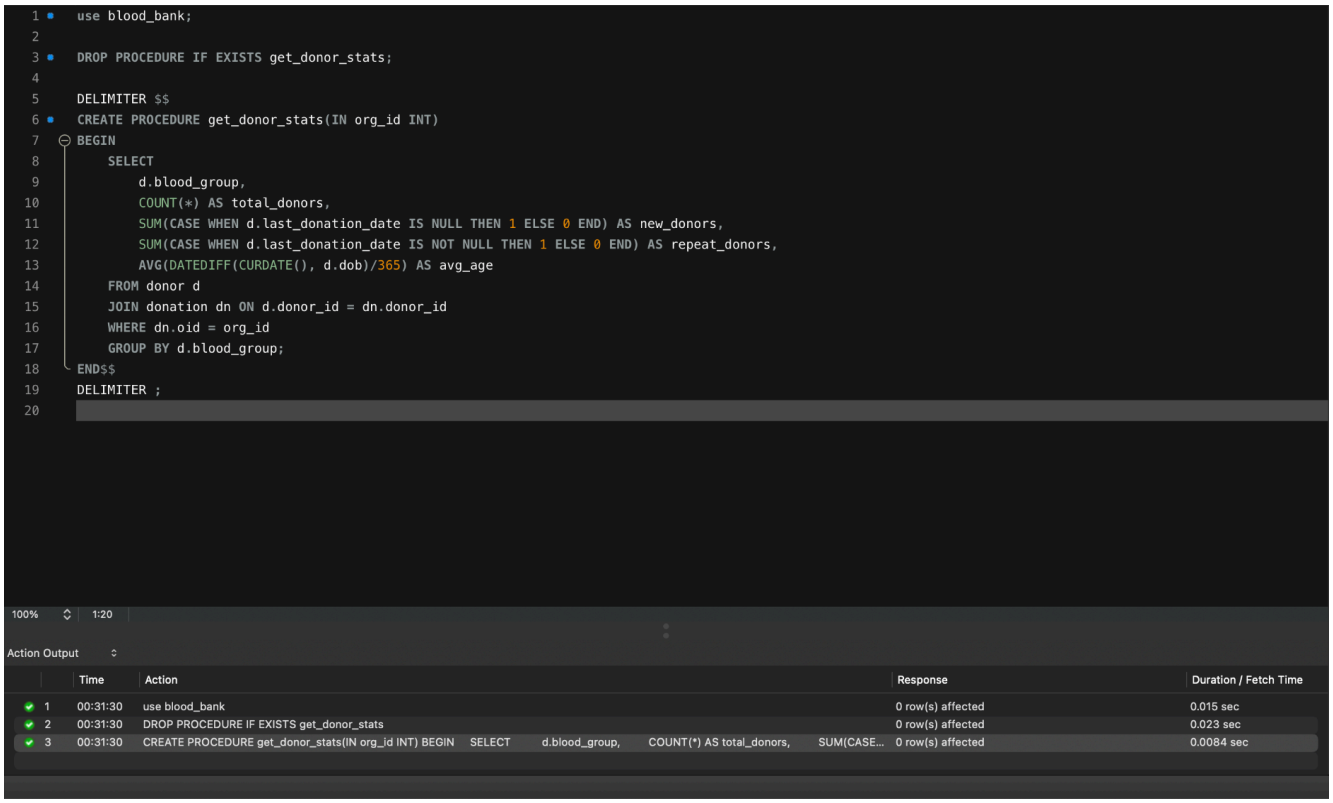
2. get_donor_stats: The get_donor_stats procedure generates analytics about donors for a specific organization, grouping them by blood type. It calculates key metrics including total donors, new vs. repeat donors (based on donation history), and average donor age. The results help organizations understand their donor demographics and plan targeted recruitment strategies.

CODE:

```
CREATE PROCEDURE get_donor_stats(IN org_id INT)
BEGIN
  SELECT
    d.blood_group,
    COUNT(*) AS total_donors,
    SUM(CASE WHEN d.last_donation_date IS NULL THEN 1 ELSE 0 END) AS new_donors,
    SUM(CASE WHEN d.last_donation_date IS NOT NULL THEN 1 ELSE 0 END) AS repeat_donors,
```

```
    AVG(DATEDIFF(CURDATE(), d.dob)/365) AS avg_age
  FROM donor d
  JOIN donation dn ON d.donor_id = dn.donor_id
  WHERE dn.oid = org_id
  GROUP BY d.blood_group;
END
```

SNAPSHOT:



3. **process_blood_request:** The process_blood_request procedure manages the approval workflow for blood requests by first verifying inventory availability. If sufficient stock exists, it approves the request and reserves the oldest available blood units (FIFO method), otherwise it marks the request as pending. This automated process ensures efficient blood allocation while maintaining proper inventory control and traceability.

CODE:

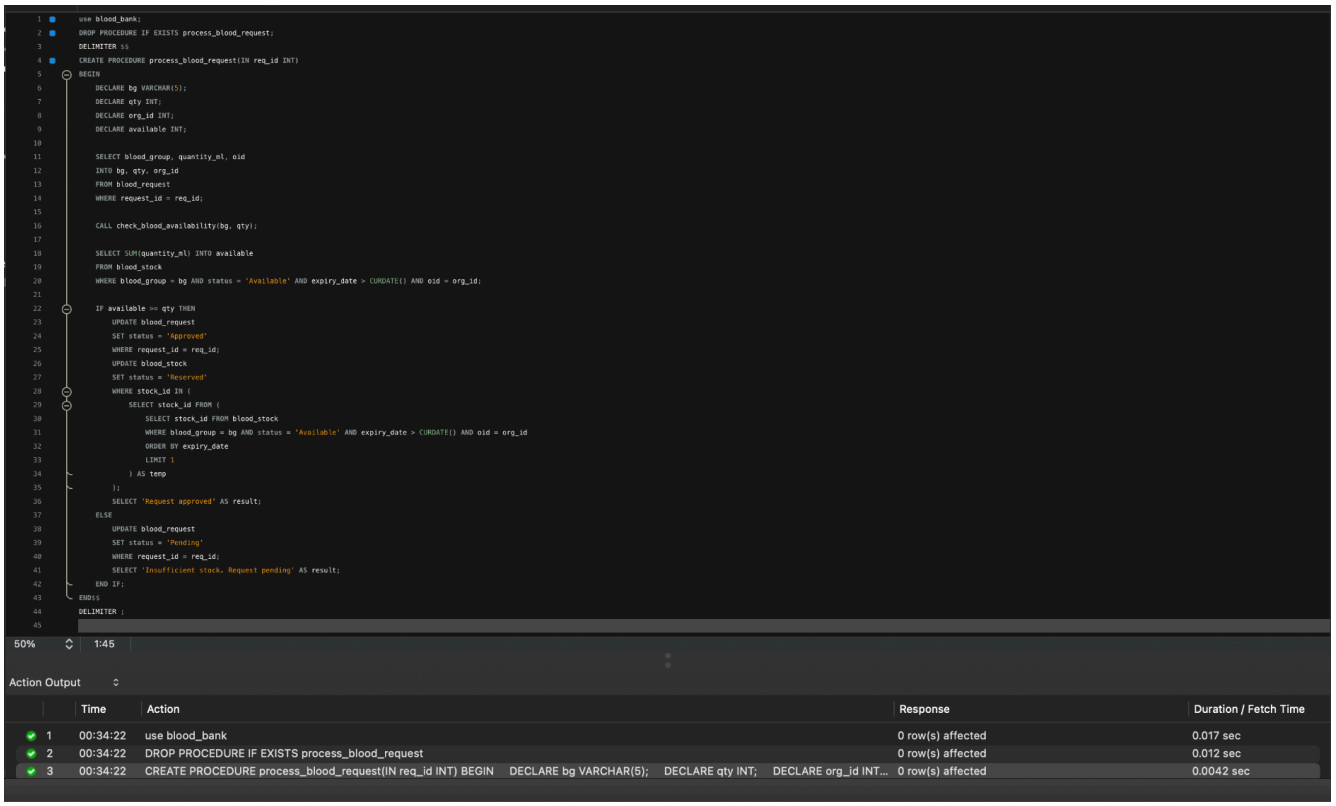
```
CREATE PROCEDURE process_blood_request(IN req_id INT)
BEGIN
   DECLARE bg VARCHAR(5);
   DECLARE qty INT;
   DECLARE org_id INT;
   DECLARE available INT;

   SELECT blood_group, quantity_ml, oid INTO bg, qty, org_id
   FROM blood_request WHERE request_id = req_id;
   CALL check_blood_availability(bg, qty);
   SELECT SUM(quantity_ml) INTO available
   FROM blood_stock
   WHERE blood_group = bg AND status = 'Available'
     AND expiry_date > CURDATE() AND oid = org_id;

   IF available >= qty THEN
      UPDATE blood_request SET status = 'Approved' WHERE request_id = req_id;
      UPDATE blood_stock
      SET status = 'Reserved'
      WHERE stock_id IN (
         SELECT stock_id FROM (
            SELECT stock_id FROM blood_stock
            WHERE blood_group = bg AND status = 'Available'
```

```
            AND expiry_date > CURDATE() AND oid = org_id
            ORDER BY expiry_date  -- FIFO (oldest first)
            LIMIT 1
        ) AS temp
    );
    SELECT 'Request approved' AS result;
  ELSE
    UPDATE blood_request SET status = 'Pending' WHERE request_id = req_id;
    SELECT 'Insufficient stock. Request pending' AS result;
  END IF;
END
```

SNAPSHOT:

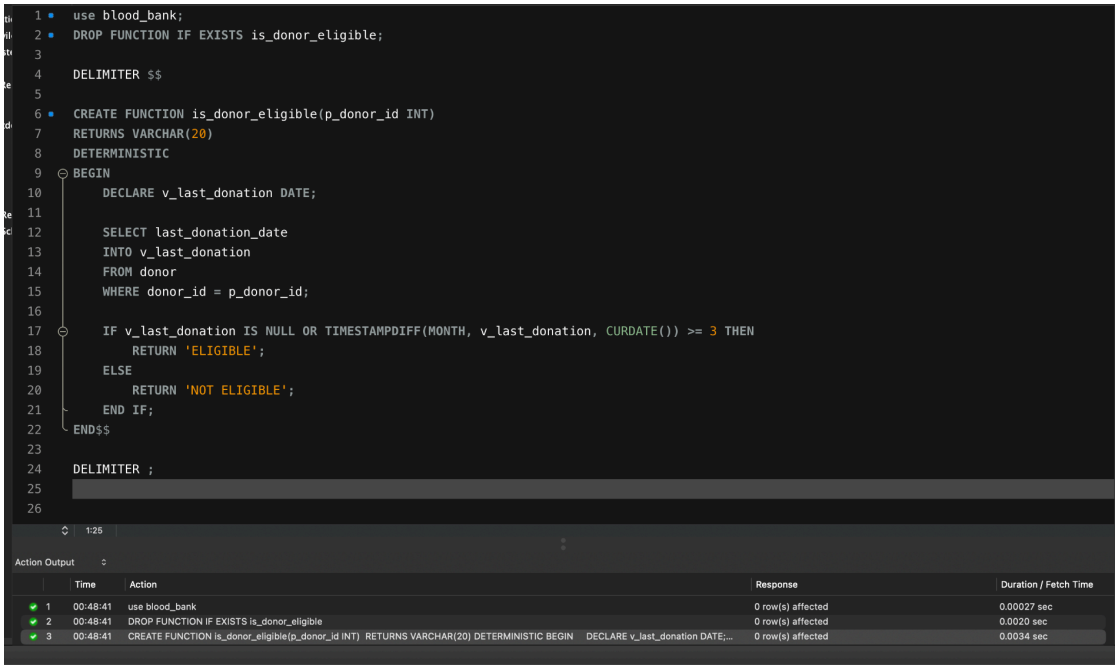


4. **is_donor_eligible:** Checks if a donor is eligible to donate blood based on their last donation date. Returns 'ELIGIBLE' if they haven't donated in the last 3 months or never donated, otherwise 'NOT ELIGIBLE'.

CODE:

```
CREATE FUNCTION is_donor_eligible(p_donor_id INT)
RETURNS VARCHAR(20)
DETERMINISTIC
BEGIN
   DECLARE v_last_donation DATE;
   SELECT last_donation_date
   INTO v_last_donation
   FROM donor
   WHERE donor_id = p_donor_id;
   IF v_last_donation IS NULL OR TIMESTAMPDIFF(MONTH, v_last_donation, CURDATE()) >= 3 THEN
      RETURN 'ELIGIBLE';
   ELSE
      RETURN 'NOT ELIGIBLE';
   END IF;
END
/
```

SNAPSHOT:



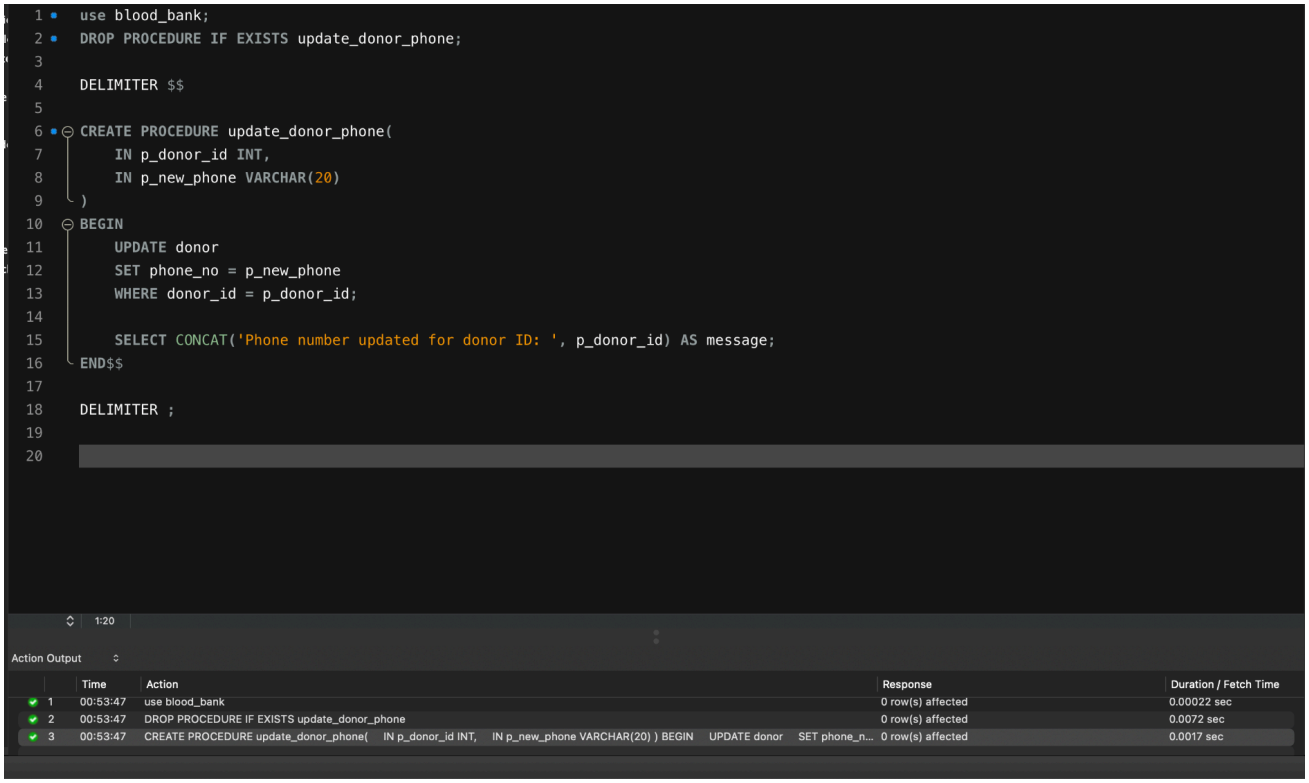
5. **update_donor_phone**: Updates a donor's phone number in the database using their donor ID. Displays a confirmation message after successful update.

CODE:

```
CREATE PROCEDURE update_donor_phone(
    IN p_donor_id INT,
    IN p_new_phone VARCHAR(20)
)
BEGIN
    UPDATE donor
    SET phone_no = p_new_phone
    WHERE donor_id = p_donor_id;

    SELECT CONCAT('Phone number updated for donor ID: ', p_donor_id) AS message;
END
```

SNAPSHOT:



```
use blood_bank;
DROP PROCEDURE IF EXISTS update_donor_phone;

DELIMITER $$

CREATE PROCEDURE update_donor_phone(
    IN p_donor_id INT,
    IN p_new_phone VARCHAR(20)
)
BEGIN
    UPDATE donor
    SET phone_no = p_new_phone
    WHERE donor_id = p_donor_id;

    SELECT CONCAT('Phone number updated for donor ID: ', p_donor_id) AS message;
END$$

DELIMITER ;
```

1:20

Action Output

	Time	Action	Response	Duration / Fetch Time
1	00:53:47	use blood_bank	0 row(s) affected	0.00022 sec
2	00:53:47	DROP PROCEDURE IF EXISTS update_donor_phone	0 row(s) affected	0.0072 sec
3	00:53:47	CREATE PROCEDURE update_donor_phone(IN p_donor_id INT, IN p_new_phone VARCHAR(20)) BEGIN UPDATE donor SET phone_n...	0 row(s) affected	0.0017 sec

SQL Query Execution Snapshots

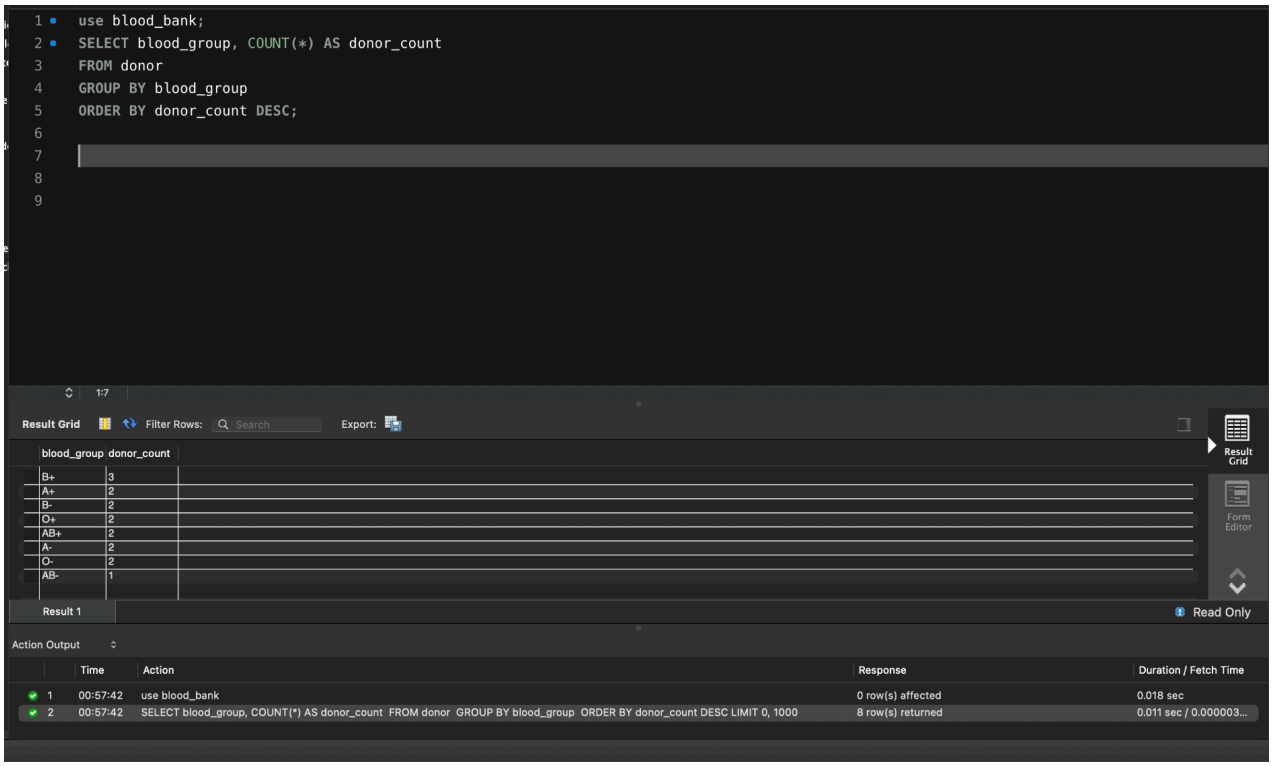
1. TO DISPLAY BASIC DONOR DEMOGRAPHICS

SELECT blood_group, COUNT(*) AS donor_count

FROM donor

GROUP BY blood_group

ORDER BY donor_count DESC;



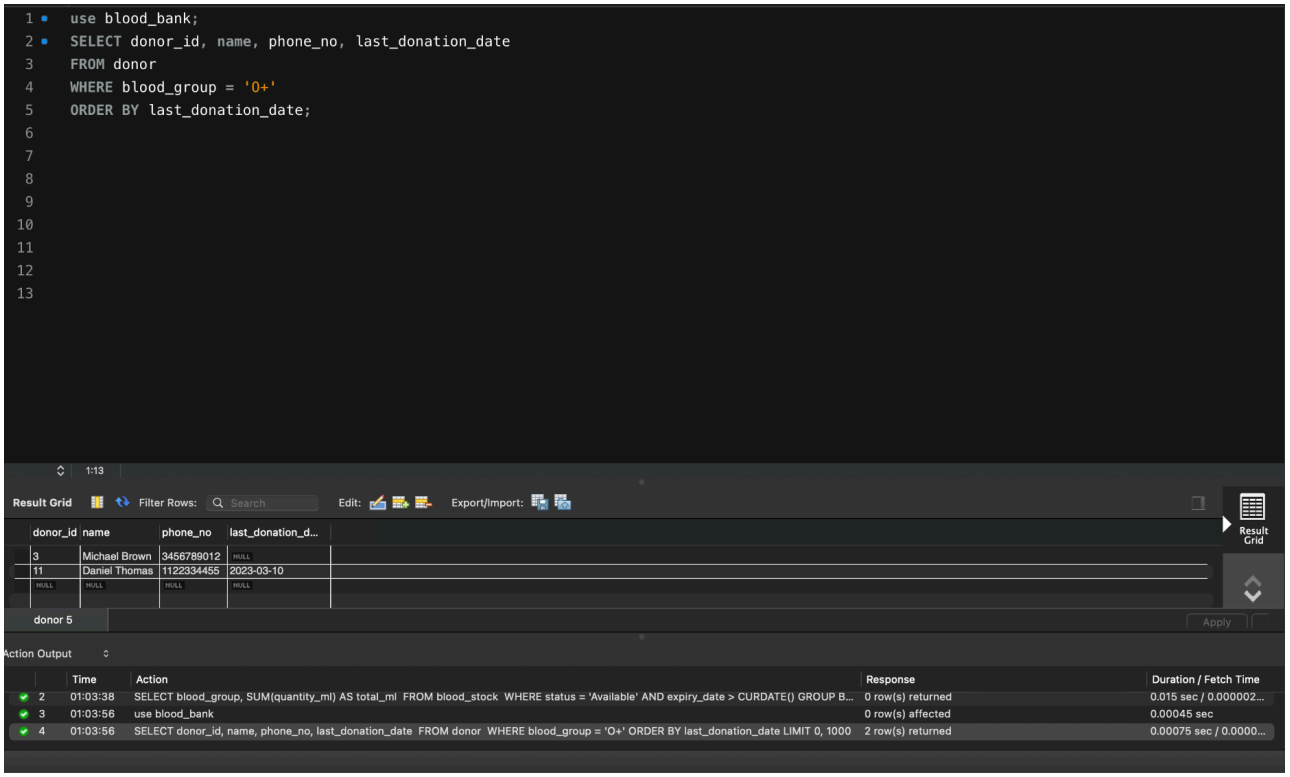
2. TO DISPLAY DONORS OF BLOOD GROUP “O+” ORDERED BY LATEST DONATION DATE

SELECT donor_id, name, phone_no, last_donation_date

FROM donor

WHERE blood_group = 'O+'

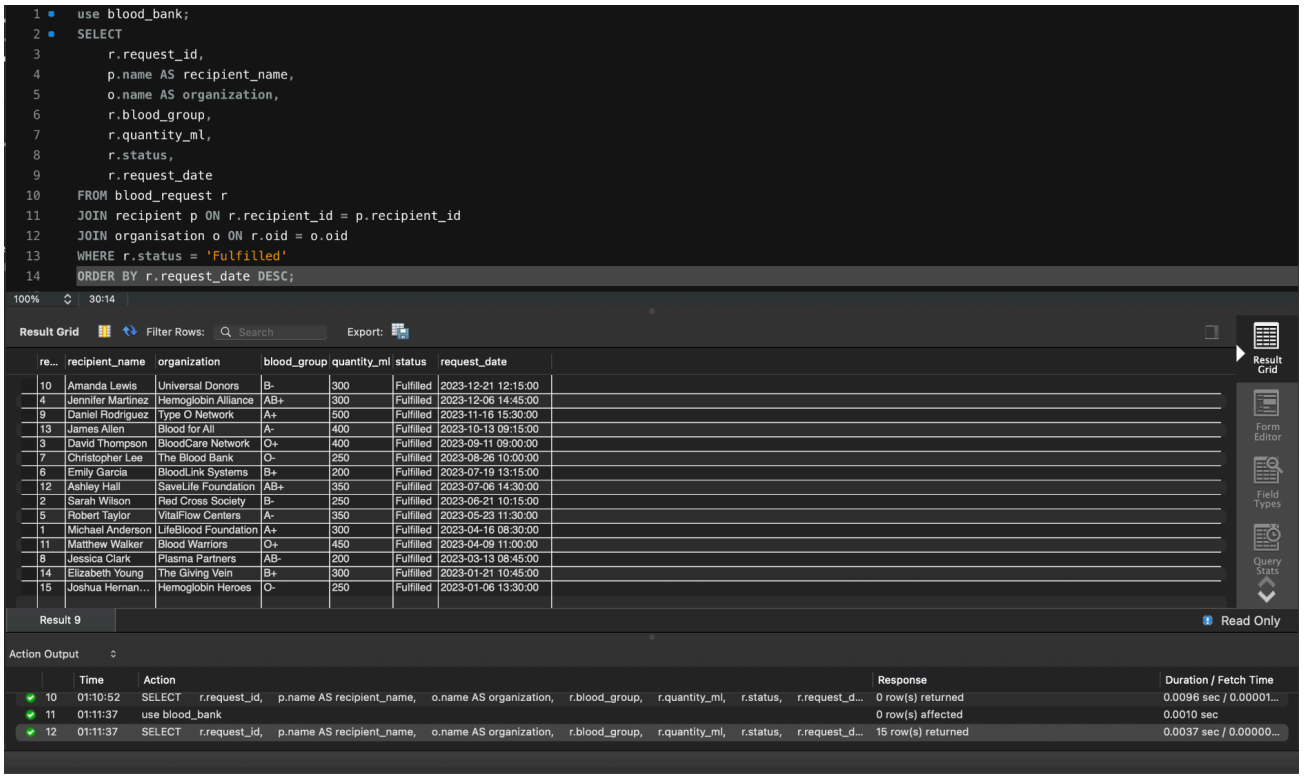
ORDER BY last_donation_date;



3. TO DISPLAY FULFILLED BLOOD REQUESTS BY LATEST REQUEST DATE

```
SELECT
    r.request_id,
    p.name AS recipient_name,
    o.name AS organization,
    r.blood_group,
    r.quantity_ml,
    r.status,
    r.request_date
FROM blood_request r
JOIN recipient p ON r.recipient_id = p.recipient_id
JOIN organisation o ON r.oid = o.oid
WHERE r.status = 'Fulfilled'
```

ORDER BY r.request_date DESC;



```
use blood_bank;
SELECT
    r.request_id,
    p.name AS recipient_name,
    o.name AS organization,
    r.blood_group,
    r.quantity_ml,
    r.status,
    r.request_date
FROM blood_request r
JOIN recipient p ON r.recipient_id = p.recipient_id
JOIN organisation o ON r.oid = o.oid
WHERE r.status = 'Fulfilled'
ORDER BY r.request_date DESC;
```

Result Grid

re...	recipient_name	organization	blood_group	quantity_ml	status	request_date
10	Amanda Lewis	Universal Donors	B-	300	Fulfilled	2023-12-21 12:15:00
4	Jennifer Martinez	Hemoglobin Alliance	AB+	300	Fulfilled	2023-12-06 14:45:00
9	Daniel Rodriguez	Type O Network	A+	500	Fulfilled	2023-11-16 15:30:00
13	James Allen	Blood for All	A-	400	Fulfilled	2023-10-13 09:15:00
3	David Thompson	BloodCare Network	O+	400	Fulfilled	2023-09-11 09:00:00
7	Christopher Lee	The Blood Bank	O-	250	Fulfilled	2023-08-26 10:00:00
6	Emily Garcia	BloodLink Systems	B+	200	Fulfilled	2023-07-19 13:15:00
12	Ashley Hall	SaveLife Foundation	AB+	350	Fulfilled	2023-07-06 14:30:00
2	Sarah Wilson	Red Cross Society	B-	250	Fulfilled	2023-06-21 10:15:00
5	Robert Taylor	VitalFlow Centers	A-	350	Fulfilled	2023-05-23 11:30:00
1	Michael Anderson	LifeBlood Foundation	A+	300	Fulfilled	2023-04-16 08:30:00
11	Matthew Walker	Blood Warriors	O+	450	Fulfilled	2023-04-09 11:00:00
8	Jessica Clark	Plasma Partners	AB-	200	Fulfilled	2023-03-13 08:45:00
14	Elizabeth Young	The Giving Vein	B+	300	Fulfilled	2023-01-21 10:45:00
15	Joshua Hernan...	Hemoglobin Heroes	O-	250	Fulfilled	2023-01-06 13:30:00

Result 9 — Read Only

Action Output

	Time	Action	Response	Duration / Fetch Time
10	01:10:52	SELECT r.request_id, p.name AS recipient_name, o.name AS organization, r.blood_group, r.quantity_ml, r.status, r.request_d...	0 row(s) returned	0.0096 sec / 0.00001...
11	01:11:37	use blood_bank	0 row(s) affected	0.0010 sec
12	01:11:37	SELECT r.request_id, p.name AS recipient_name, o.name AS organization, r.blood_group, r.quantity_ml, r.status, r.request_d...	15 row(s) returned	0.0037 sec / 0.00000...

4. TO DISPLAY DONATIONS WITH DONOR AND ORGANIZATION INFORMATION

SELECT

dn.donation_id,

d.name AS donor_name,

d.blood_group,

o.name AS organization,

dn.donation_date,

dn.quantity_ml

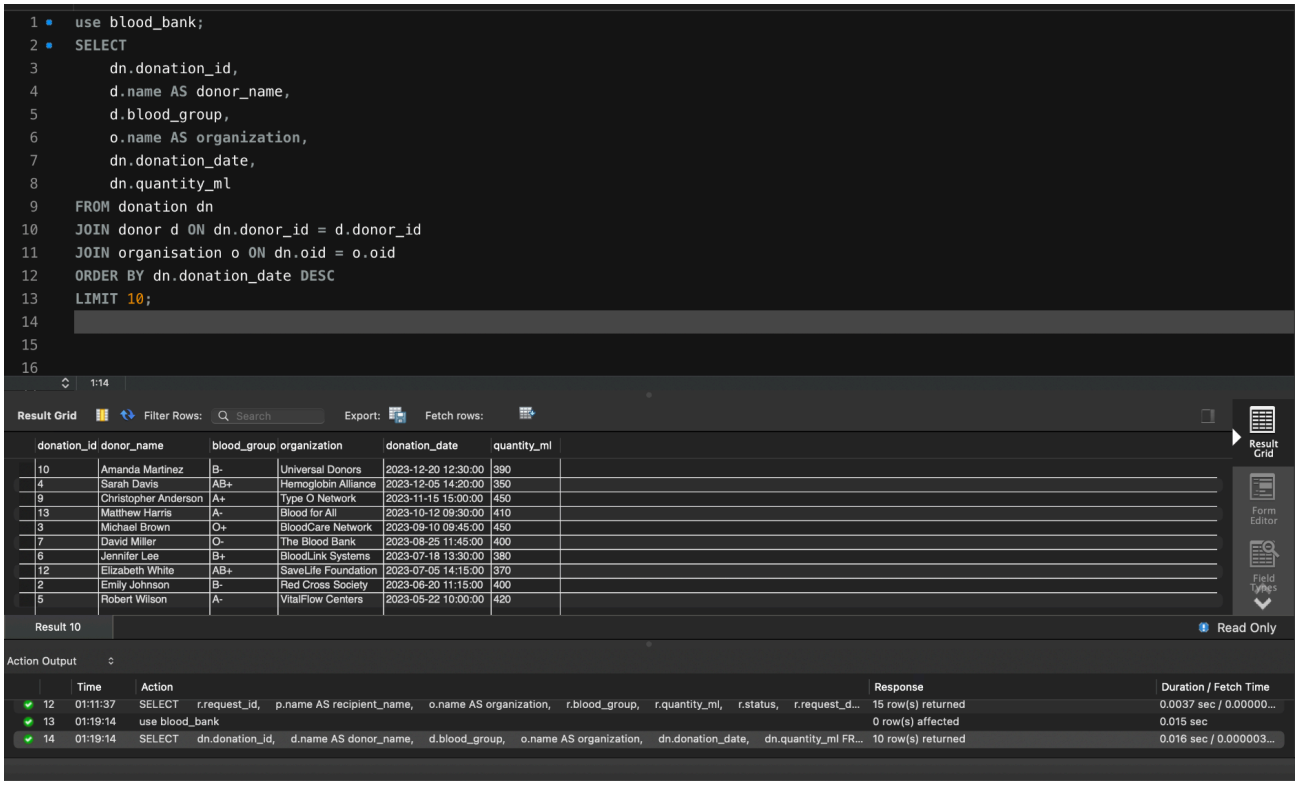
FROM donation dn

JOIN donor d ON dn.donor_id = d.donor_id

JOIN organisation o ON dn.oid = o.oid

ORDER BY dn.donation_date DESC

LIMIT 10;



```
use blood_bank;
SELECT
    dn.donation_id,
    d.name AS donor_name,
    d.blood_group,
    o.name AS organization,
    dn.donation_date,
    dn.quantity_ml
FROM donation dn
JOIN donor d ON dn.donor_id = d.donor_id
JOIN organisation o ON dn.oid = o.oid
ORDER BY dn.donation_date DESC
LIMIT 10;
```

Result Grid

donation_id	donor_name	blood_group	organization	donation_date	quantity_ml
10	Amanda Martinez	B-	Universal Donors	2023-12-20 12:30:00	390
4	Sarah Davis	AB+	Hemoglobin Alliance	2023-12-05 14:20:00	350
9	Christopher Anderson	A+	Type O Network	2023-11-15 15:00:00	450
13	Matthew Harris	A-	Blood for All	2023-10-12 09:30:00	410
3	Michael Brown	O+	BloodCare Network	2023-09-10 09:45:00	450
7	David Miller	O-	The Blood Bank	2023-08-25 11:45:00	400
6	Jennifer Lee	B+	BloodLink Systems	2023-07-18 13:30:00	380
12	Elizabeth White	AB+	SaveLife Foundation	2023-07-05 14:15:00	370
2	Emily Johnson	B-	Red Cross Society	2023-06-20 11:15:00	400
5	Robert Wilson	A-	VitalFlow Centers	2023-05-22 10:00:00	420

Result 10 — Read Only

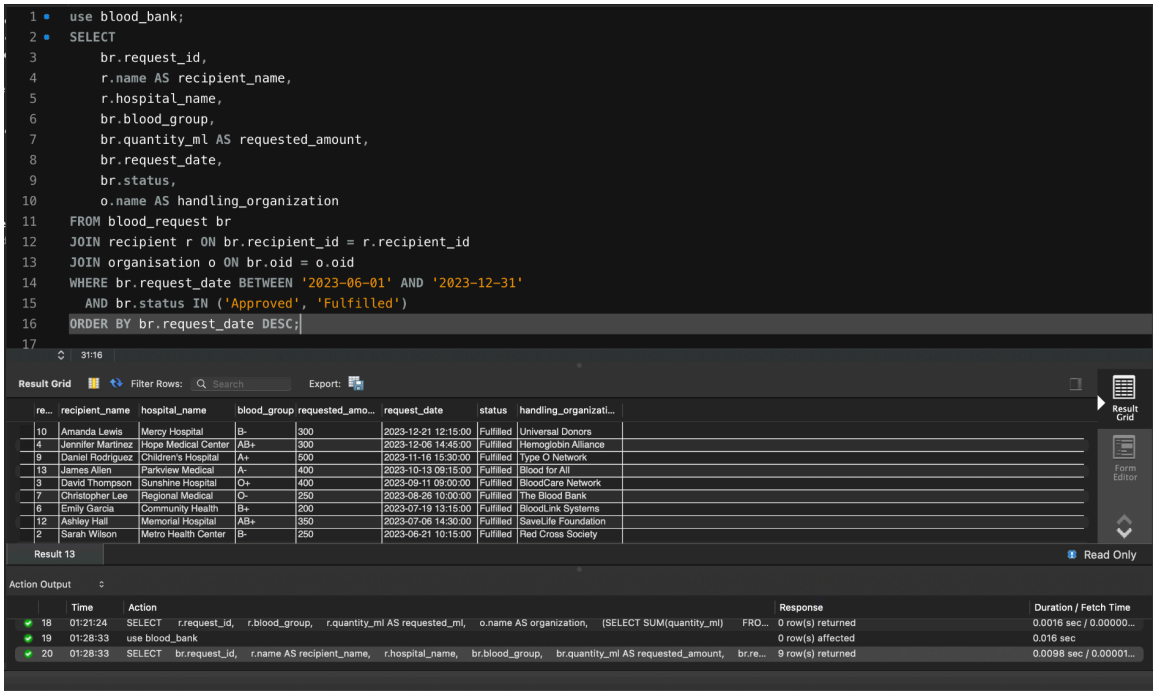
Action Output

	Time	Action	Response	Duration / Fetch Time
12	01:11:37	SELECT r.request_id, p.name AS recipient_name, o.name AS organization, r.blood_group, r.quantity_ml, r.status, r.request_d...	15 row(s) returned	0.0037 sec / 0.00000...
13	01:19:14	use blood_bank	0 row(s) affected	0.015 sec
14	01:19:14	SELECT dn.donation_id, d.name AS donor_name, d.blood_group, o.name AS organization, dn.donation_date, dn.quantity_ml FR...	10 row(s) returned	0.016 sec / 0.000003...

5. TO DISPLAY BLOOD REQUESTS WITH RECIPIENT AND HOSPITAL DETAILS

```
SELECT
    br.request_id,
    r.name AS recipient_name,
    r.hospital_name,
    br.blood_group,
    br.quantity_ml AS requested_amount,
    br.request_date,
    br.status,
    o.name AS handling_organization
FROM blood_request br
JOIN recipient r ON br.recipient_id = r.recipient_id
JOIN organisation o ON br.oid = o.oid
WHERE br.request_date BETWEEN '2023-06-01' AND '2023-12-31'
  AND br.status IN ('Approved', 'Fulfilled')
```

ORDER BY br.request_date DESC;



```
use blood_bank;
SELECT
    br.request_id,
    r.name AS recipient_name,
    r.hospital_name,
    br.blood_group,
    br.quantity_ml AS requested_amount,
    br.request_date,
    br.status,
    o.name AS handling_organization
FROM blood_request br
JOIN recipient r ON br.recipient_id = r.recipient_id
JOIN organisation o ON br.oid = o.oid
WHERE br.request_date BETWEEN '2023-06-01' AND '2023-12-31'
  AND br.status IN ('Approved', 'Fulfilled')
ORDER BY br.request_date DESC;
```

Result Grid — Filter Rows: Search — Export:

re...	recipient_name	hospital_name	blood_group	requested_amo...	request_date	status	handling_organizati...
10	Amanda Lewis	Mercy Hospital	B-	300	2023-12-21 12:15:00	Fulfilled	Universal Donors
4	Jennifer Martinez	Hope Medical Center	AB+	300	2023-12-06 14:45:00	Fulfilled	Hemoglobin Alliance
9	Daniel Rodriguez	Children's Hospital	A+	500	2023-11-16 15:30:00	Fulfilled	Type O Network
13	James Allen	Parkview Medical	A-	400	2023-10-13 09:15:00	Fulfilled	Blood for All
3	David Thompson	Sunshine Hospital	O+	400	2023-09-11 09:00:00	Fulfilled	BloodCare Network
7	Christopher Lee	Regional Medical	O-	250	2023-08-26 10:00:00	Fulfilled	The Blood Bank
6	Emily Garcia	Community Health	B+	200	2023-07-19 13:15:00	Fulfilled	BloodLink Systems
12	Ashley Hall	Memorial Hospital	AB+	350	2023-07-06 14:30:00	Fulfilled	SaveLife Foundation
2	Sarah Wilson	Metro Health Center	B-	250	2023-06-21 10:15:00	Fulfilled	Red Cross Society

Result 13 — Read Only

Action Output

	Time	Action	Response	Duration / Fetch Time
18	01:21:24	SELECT r.request_id, r.blood_group, r.quantity_ml AS requested_ml, o.name AS organization, (SELECT SUM(quantity_ml) FRO...	0 row(s) returned	0.0016 sec / 0.00000...
19	01:28:33	use blood_bank	0 row(s) affected	0.016 sec
20	01:28:33	SELECT br.request_id, r.name AS recipient_name, r.hospital_name, br.blood_group, br.quantity_ml AS requested_amount, br.re...	9 row(s) returned	0.0098 sec / 0.00001...

Conclusion

The **Blood Bank Management System** developed using Python provides an efficient and reliable solution for managing donor records, blood inventory, and patient requests. By automating critical processes, the system reduces manual errors, enhances data accuracy, and ensures timely access to blood supplies for those in need.

This project demonstrates the practical application of database management in healthcare, streamlining operations for blood banks and improving service delivery. Its scalable design allows for future enhancements, such as integration with hospital systems or advanced analytics for blood demand forecasting.

In conclusion, the Blood Bank Management System serves as a valuable tool for blood banks, contributing to better resource management and ultimately saving lives. Future work may include expanding features, improving user interfaces, and incorporating real-time tracking for broader impact.

References

[1] *SQL Functions*. https://docs.oracle.com/cd/B13789_01/server.101/b10759/functions001.htm

[2] *MySQL :: MySQL Connector/Python Developer Guide*.
https://dev.mysql.com/doc/connector-python/en/