DBMS PROJECT

TOPIC: BLOOD BANK MANAGEMENT SYSTEM

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

SUBMITTED BY:

DAKSH KUMAR NAHAR - 102103481

AARUSHI JAIN – 102103483

SELINA VARSHNEY- 102103496

VAANI DANG – 102283010

SUBMITTED TO: ARCHANA SINGH

INDEX

TOPIC	PAGE NO.
INTRODUCTION	2
ER DIAGRAM	3
ER TO TABLES	4
NORMALIZATION	8
SQL IMPLEMENTATION	11
CONCLUSION	22
REFERENCES	23

AIM

The aim of the blood bank management system is to ensure that there is a sufficient supply of safe and quality blood available to meet the needs of patients who require transfusions or other blood-related medical procedures.

INTRODUCTION

The system includes three main tables: DONOR_INFO, PATIENT_INFO, and INVENTORY. The DONOR_INFO table stores information about the blood donors such as their first and last name, donor ID, branch ID, date of birth, gender, phone number, and past medical history. Similarly, the PATIENT_INFO table stores information about patients such as their first and last name, patient ID, branch ID, date of birth, gender, phone number, and past medical history.

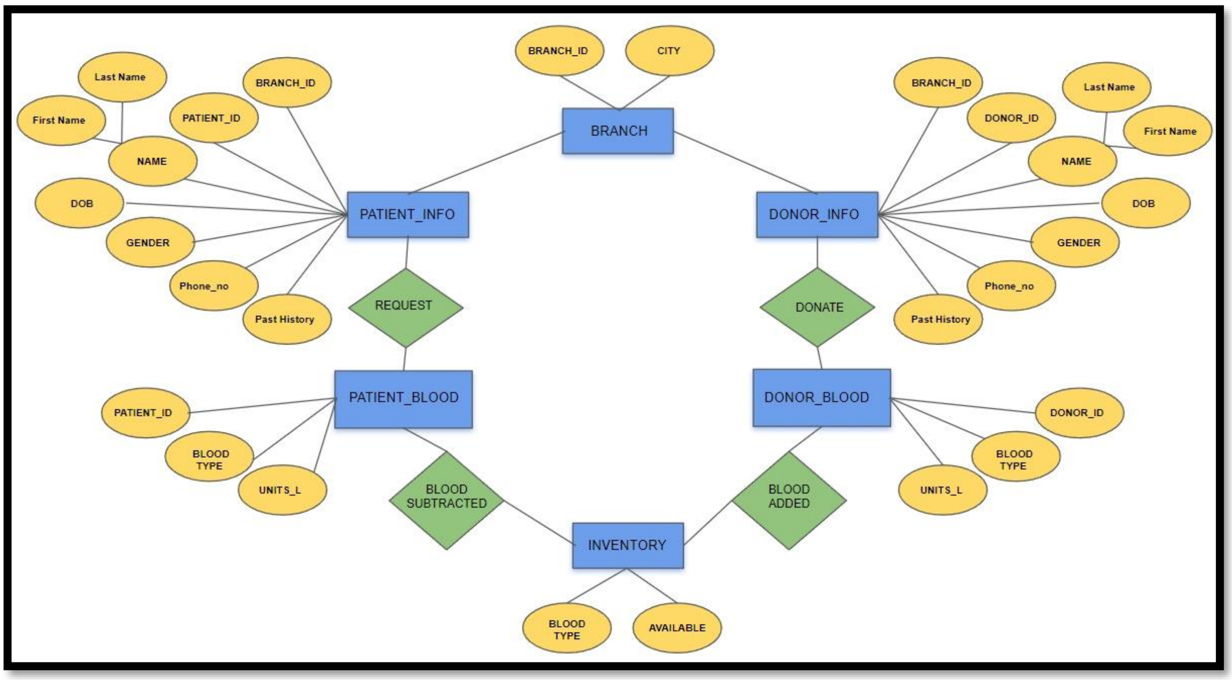
The INVENTORY table stores information about the available blood units and their types. It includes two columns: BLOOD_TYPE and AVAILABLE, where BLOOD_TYPE specifies the type of blood and AVAILABLE specifies the number of units available for that particular blood type.

Additionally, a DONOR_BLOOD table has been created to store information about the blood donated by a particular donor. It includes the donor ID, blood type, and number of units of blood donated.

A trigger named UPDATE_INVENTORY has also been created, which automatically updates the available blood units in the INVENTORY table whenever a new donation is made.

Overall, this blood bank management system provides basic functionality to store and manage information about blood donors, patients, and inventory, and automate inventory management by updating the available units of blood whenever a new donation is made.

ER DIAGRAM


BRANCH_ID
CITY
BRANCH
Last Name
BRANCH_ID
First Name
PATIENT_ID
NAME
DOB
PATIENT_INFO
GENDER
Phone_no
Past History
REQUEST
BRANCH_ID
Last Name
DONOR_ID
First Name
NAME
DOB
DONOR_INFO
GENDER
Phone_no
Past History
DONATE
PATIENT_ID
PATIENT_BLOOD
BLOOD TYPE
UNITS_L
DONOR_BLOOD
DONOR_ID
BLOOD TYPE
UNITS_L
BLOOD SUBTRACTED
BLOOD ADDED
INVENTORY
BLOOD TYPE
AVAILABLE

ER TO TABLES

1. BRANCH

COLUMN NAME	DATA TYPE	CONSTRAINTS
BRANCH_ID	Int	UNIQUE
CITY	VARCHAR(20)	

2. PATIENT_INFO

COLUMN NAME	DATA TYPE	CONSTRAINTS
First_name	**Varchar(30)**	
Last_name	**Varchar(30)**	
PATIENT_ID	**Int**	**Primary key**
BRANCH_ID	**int**	
DOB	**Date**	
GENDER	**Varchar(2)**	
Phone_no	**Number(10)**	**Unique**
Past_medical_disease	**Varchar(50)**	

3. DONOR_INFO

COLUMN NAME	DATA TYPE	CONSTRAINTS
First_name	Varchar(30)	
Last_name	Varchar(30)	
DONOR_ID	Int	Primary key
BRANCH_ID	int	
DOB	Date	
GENDER	Varchar(2)	
Phone_no	Number(10)	Unique
Past_medical_disease	Varchar(50)	

4. PATIENT_BLOOD

COLUMN NAME	DATA TYPE	CONSTRAINTS
PATIENT_ID	INT	PRIMARY KEY FOREIGN KEY
BLOOD_TYPE	VARCHAR(10)	
UNITS_L	INT	NOT NULL

5. DONOR_BLOOD

COLUMN NAME	DATA TYPE	CONSTRAINTS
DONOR_ID	INT	PRIMARY KEY FOREIGN KEY
BLOOD_TYPE	VARCHAR(10)	
UNITS_L	INT	NOT NULL

6. INVENTORY

COLUMN NAME	DATA TYPE	CONSTRAINTS
BLOOD_TYPE	VARCHAR(10)	UNIQUE
AVAILABLE	INT	

NORMALIZATION PROCESS

1. BRANCH TABLE

First Normal Form (1NF):

- The table already satisfies the requirements of 1NF, as each column contains only atomic values.

Second Normal Form (2NF):

- The table has a composite primary key consisting of BRANCH_ID and CITY.
- However, there is no non-key attribute, so the table already satisfies the requirements of 2NF.

Third Normal Form (3NF):

- The table already satisfies the requirements of 3NF, as there are no transitive dependencies between non-key attributes.

Therefore, the given table BRANCH is already normalized up to 3NF.

2. PATIENT_INFO TABLE

First Normal Form (1NF):

- The table already satisfies the requirements of 1NF, as each column contains only atomic values.

Second Normal Form (2NF):

- The table has a composite primary key consisting of PATIENT_ID and BRANCH_ID.
- However, there is only one non-key attribute, Past_medical_disease, which is functionally dependent on PATIENT_ID only. That's why the table is not of 2NF form.

3. DONOR_INFO TABLE

First Normal Form (1NF):

- The table already satisfies the requirements of 1NF, as each column contains only atomic values.

Second Normal Form (2NF):

- The table has a composite primary key consisting of DONOR_ID and BRANCH_ID.
- However, there is only one non-key attribute, Past_medical_disease, which is functionally dependent on DONOR_ID only.

4. PATIENT_BLOOD TABLE

First Normal Form (1NF):

- The table already satisfies the requirements of 1NF, as each column contains only atomic values.

Second Normal Form (2NF):

- The table has a composite primary key consisting of PATIENT_ID and BLOOD_TYPE.
- The non-key attribute UNITS_L is fully functionally dependent on the composite primary key.
- Therefore, the table is already in 2NF.

Third Normal Form (3NF):

- There are no transitive dependencies in the table, so it is already in 3NF.

Therefore, the given table PATIENT_BLOOD is already normalized to 3NF.

5. DONOR_BLOOD TABLE

First Normal Form (1NF):

- The table already satisfies the requirements of 1NF, as each column contains only atomic values.

Second Normal Form (2NF):

- The table has a composite primary key consisting of DONOR_ID and BLOOD_TYPE.
- The non-key attribute UNITS_L is fully functionally dependent on the composite primary key.
- Therefore, the table is already in 2NF.

Third Normal Form (3NF):

- There are no transitive dependencies in the table, so it is already in 3NF.

Therefore, the given table DONOR_BLOOD is already normalized to 3NF.

6. <u>INVENTORY TABLE</u>

First Normal Form (1NF):

- The given table INVENTORY is already in 1NF, as each column contains only atomic values.

Second Normal Form (2NF):

- The table has a single-column primary key, BLOOD_TYPE.
- There is only one non-key attribute, AVAILABLE, which is fully functionally dependent on BLOOD_TYPE.
- Therefore, the table is already in 2NF.

Third Normal Form (3NF):

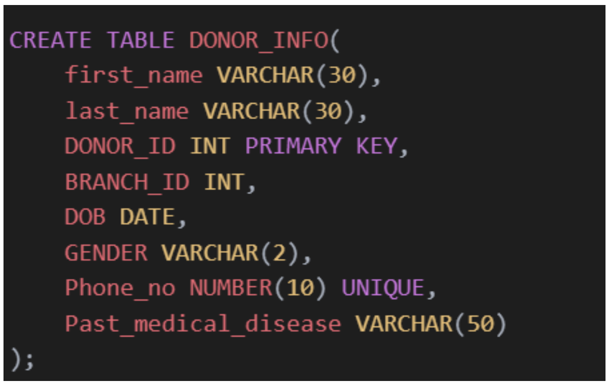
- There are no transitive dependencies in the table, so it is already in 3NF.

Therefore, the given table INVENTORY is already normalized to 3NF.

PL/SQL COMMANDS:

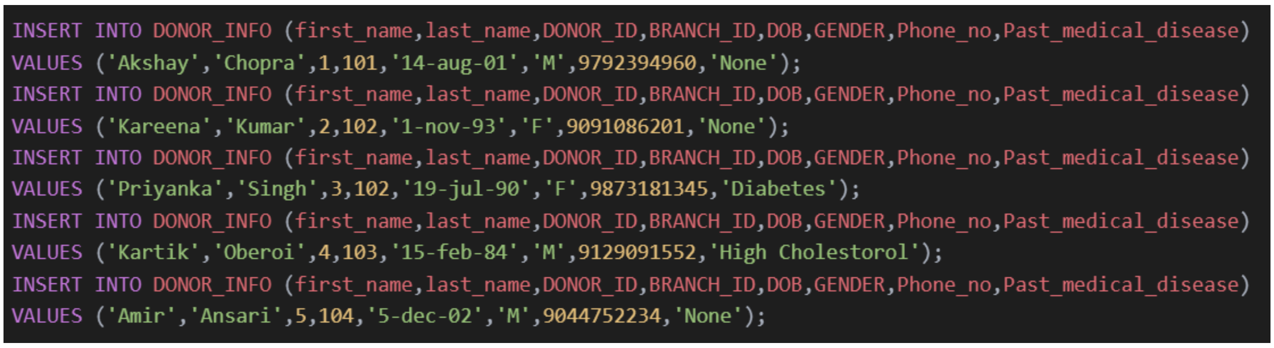
DONOR_INFO TABLE

Creating a table DONOR_INFO HAVING Details of all Donors:



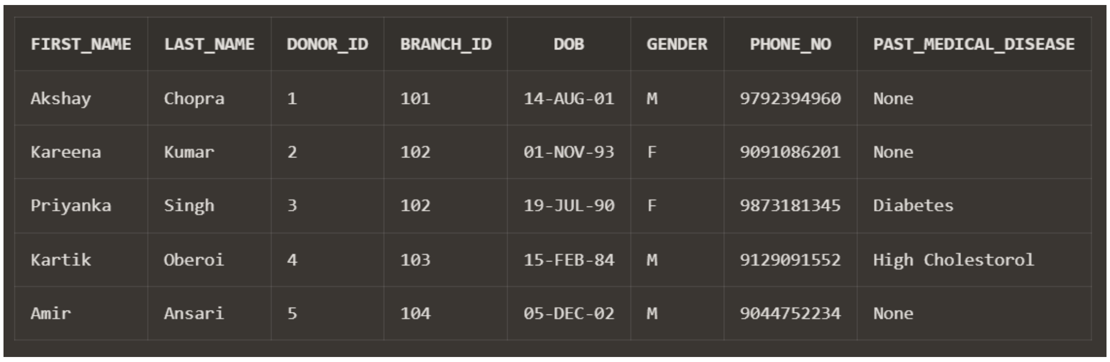
```
CREATE TABLE DONOR_INFO(
    first_name VARCHAR(30),
    last_name VARCHAR(30),
    DONOR_ID INT PRIMARY KEY,
    BRANCH_ID INT,
    DOB DATE,
    GENDER VARCHAR(2),
    Phone_no NUMBER(10) UNIQUE,
    Past_medical_disease VARCHAR(50)
);
```

Inserting values in DONORS_INFO:



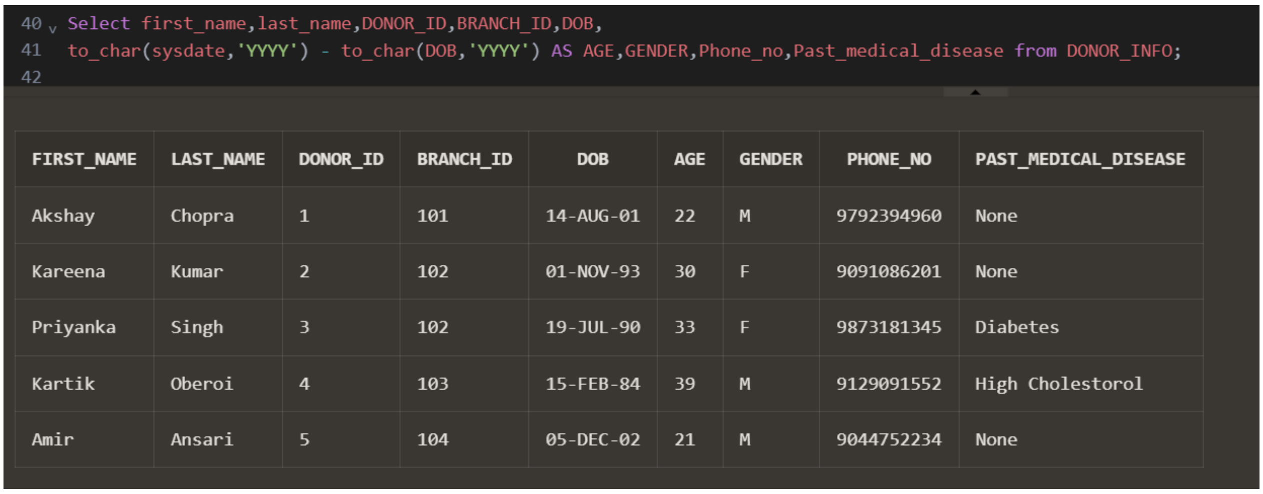
```
INSERT INTO DONOR_INFO (first_name,last_name,DONOR_ID,BRANCH_ID,DOB,GENDER,Phone_no,Past_medical_disease)
VALUES ('Akshay','Chopra',1,101,'14-aug-01','M',9792394960,'None');
INSERT INTO DONOR_INFO (first_name,last_name,DONOR_ID,BRANCH_ID,DOB,GENDER,Phone_no,Past_medical_disease)
VALUES ('Kareena','Kumar',2,102,'1-nov-93','F',9091086201,'None');
INSERT INTO DONOR_INFO (first_name,last_name,DONOR_ID,BRANCH_ID,DOB,GENDER,Phone_no,Past_medical_disease)
VALUES ('Priyanka','Singh',3,102,'19-jul-90','F',9873181345,'Diabetes');
INSERT INTO DONOR_INFO (first_name,last_name,DONOR_ID,BRANCH_ID,DOB,GENDER,Phone_no,Past_medical_disease)
VALUES ('Kartik','Oberoi',4,103,'15-feb-84','M',9129091552,'High Cholestorol');
INSERT INTO DONOR_INFO (first_name,last_name,DONOR_ID,BRANCH_ID,DOB,GENDER,Phone_no,Past_medical_disease)
VALUES ('Amir','Ansari',5,104,'5-dec-02','M',9044752234,'None');
```

Displaying the DONOR_INFO TABLE:



FIRST_NAME	LAST_NAME	DONOR_ID	BRANCH_ID	DOB	GENDER	PHONE_NO	PAST_MEDICAL_DISEASE
Akshay	Chopra	1	101	14-AUG-01	M	9792394960	None
Kareena	Kumar	2	102	01-NOV-93	F	9091086201	None
Priyanka	Singh	3	102	19-JUL-90	F	9873181345	Diabetes
Kartik	Oberoi	4	103	15-FEB-84	M	9129091552	High Cholestorol
Amir	Ansari	5	104	05-DEC-02	M	9044752234	None

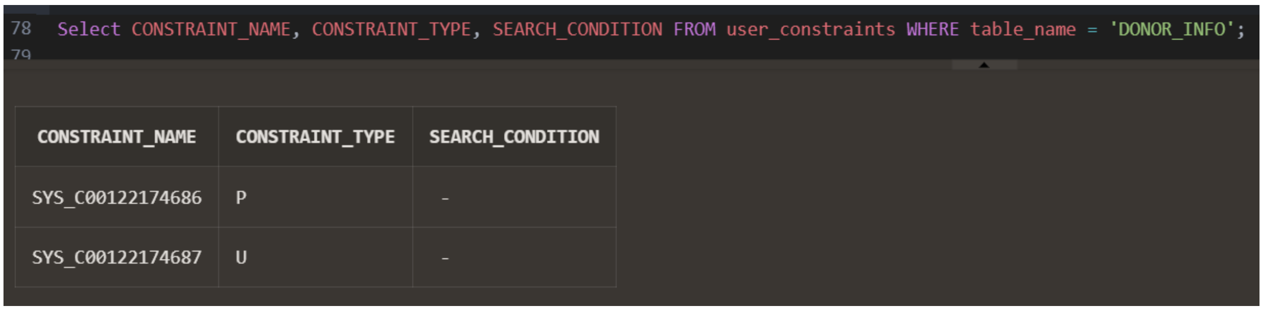
Deriving Age as an attribute:



```
Select first_name,last_name,DONOR_ID,BRANCH_ID,DOB,
to_char(sysdate,'YYYY') - to_char(DOB,'YYYY') AS AGE,GENDER,Phone_no,Past_medical_disease from DONOR_INFO;
```

FIRST_NAME	LAST_NAME	DONOR_ID	BRANCH_ID	DOB	AGE	GENDER	PHONE_NO	PAST_MEDICAL_DISEASE
Akshay	Chopra	1	101	14-AUG-01	22	M	9792394960	None
Kareena	Kumar	2	102	01-NOV-93	30	F	9091086201	None
Priyanka	Singh	3	102	19-JUL-90	33	F	9873181345	Diabetes
Kartik	Oberoi	4	103	15-FEB-84	39	M	9129091552	High Cholestorol
Amir	Ansari	5	104	05-DEC-02	21	M	9044752234	None

Displaying constraint details:

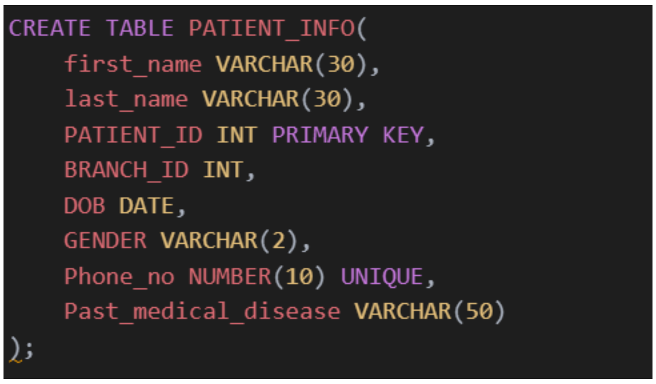


```
Select CONSTRAINT_NAME, CONSTRAINT_TYPE, SEARCH_CONDITION FROM user_constraints WHERE table_name = 'DONOR_INFO';
```

CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION
SYS_C00122174686	P	-
SYS_C00122174687	U	-

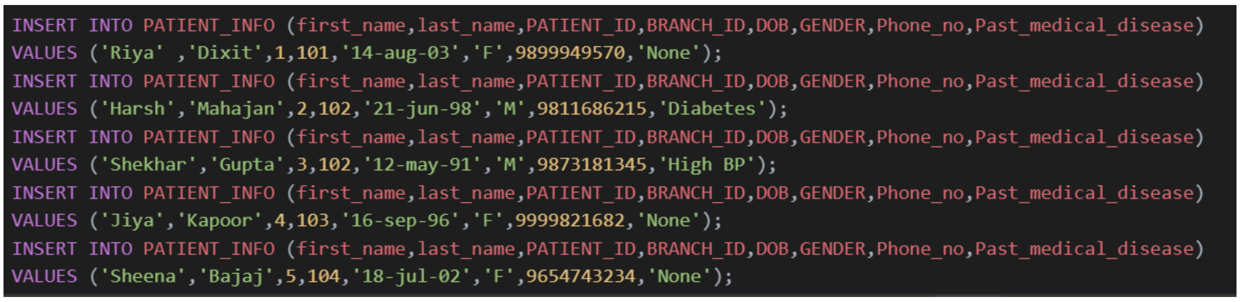
PATIENT_INFO TABLE:

Creating a table PATIENT_INFO HAVING Details of all Patients:



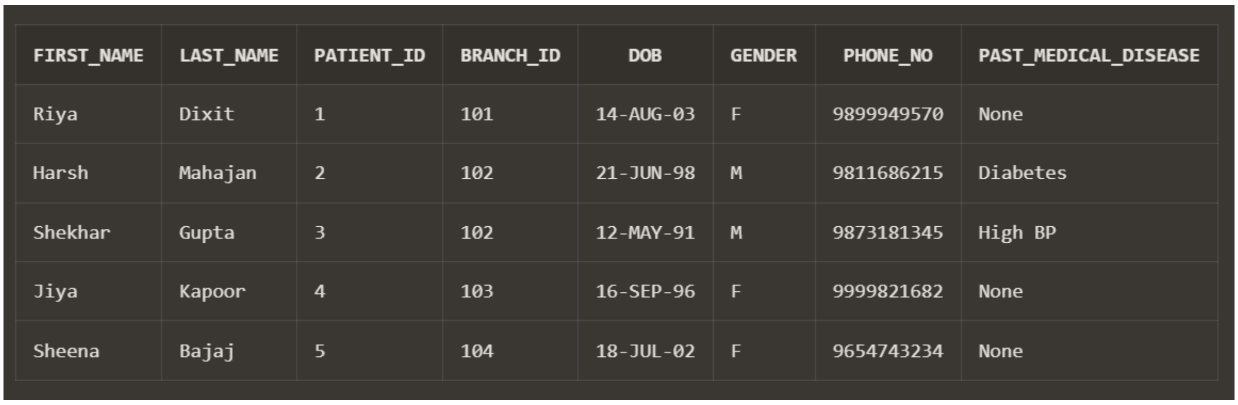
```
CREATE TABLE PATIENT_INFO(
    first_name VARCHAR(30),
    last_name VARCHAR(30),
    PATIENT_ID INT PRIMARY KEY,
    BRANCH_ID INT,
    DOB DATE,
    GENDER VARCHAR(2),
    Phone_no NUMBER(10) UNIQUE,
    Past_medical_disease VARCHAR(50)
);
```

Inserting values in PATIENTS_INFO:



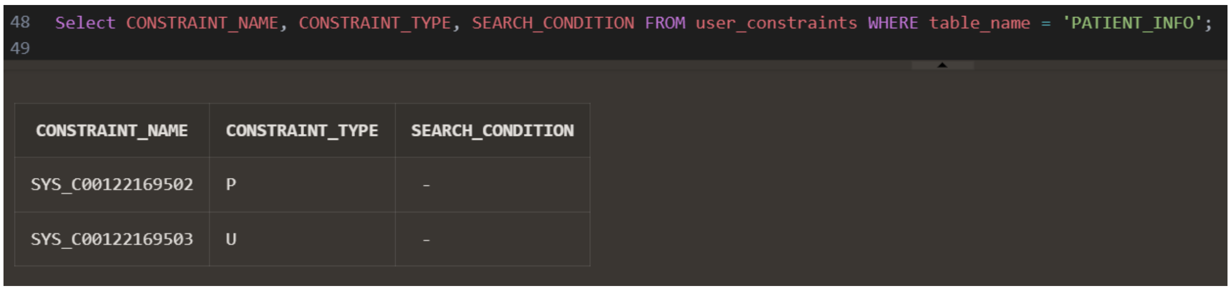
```sql
INSERT INTO PATIENT_INFO (first_name,last_name,PATIENT_ID,BRANCH_ID,DOB,GENDER,Phone_no,Past_medical_disease)
VALUES ('Riya'  ,'Dixit',1,101,'14-aug-03','F',9899949570,'None');
INSERT INTO PATIENT_INFO (first_name,last_name,PATIENT_ID,BRANCH_ID,DOB,GENDER,Phone_no,Past_medical_disease)
VALUES ('Harsh','Mahajan',2,102,'21-jun-98','M',9811686215,'Diabetes');
INSERT INTO PATIENT_INFO (first_name,last_name,PATIENT_ID,BRANCH_ID,DOB,GENDER,Phone_no,Past_medical_disease)
VALUES ('Shekhar','Gupta',3,102,'12-may-91','M',9873181345,'High BP');
INSERT INTO PATIENT_INFO (first_name,last_name,PATIENT_ID,BRANCH_ID,DOB,GENDER,Phone_no,Past_medical_disease)
VALUES ('Jiya','Kapoor',4,103,'16-sep-96','F',9999821682,'None');
INSERT INTO PATIENT_INFO (first_name,last_name,PATIENT_ID,BRANCH_ID,DOB,GENDER,Phone_no,Past_medical_disease)
VALUES ('Sheena','Bajaj',5,104,'18-jul-02','F',9654743234,'None');
```

Displaying the PATIENT_INFO TABLE:



FIRST_NAME	LAST_NAME	PATIENT_ID	BRANCH_ID	DOB	GENDER	PHONE_NO	PAST_MEDICAL_DISEASE
Riya	Dixit	1	101	14-AUG-03	F	9899949570	None
Harsh	Mahajan	2	102	21-JUN-98	M	9811686215	Diabetes
Shekhar	Gupta	3	102	12-MAY-91	M	9873181345	High BP
Jiya	Kapoor	4	103	16-SEP-96	F	9999821682	None
Sheena	Bajaj	5	104	18-JUL-02	F	9654743234	None

Displaying constraint details:

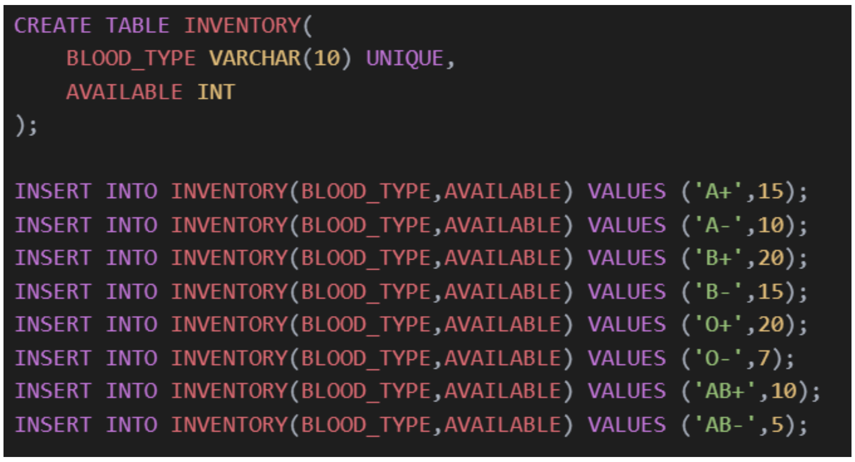


```sql
Select CONSTRAINT_NAME, CONSTRAINT_TYPE, SEARCH_CONDITION FROM user_constraints WHERE table_name = 'PATIENT_INFO';
```

CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION
SYS_C00122169502	P	-
SYS_C00122169503	U	-

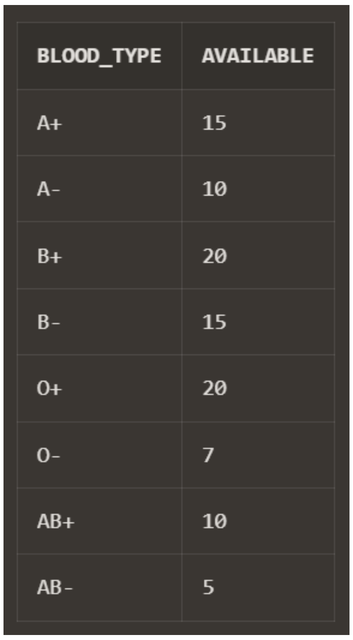
INVENTORY TABLE

Creating a table INVENTORY showing the blood units available in the blood bank, and inserting values in it:



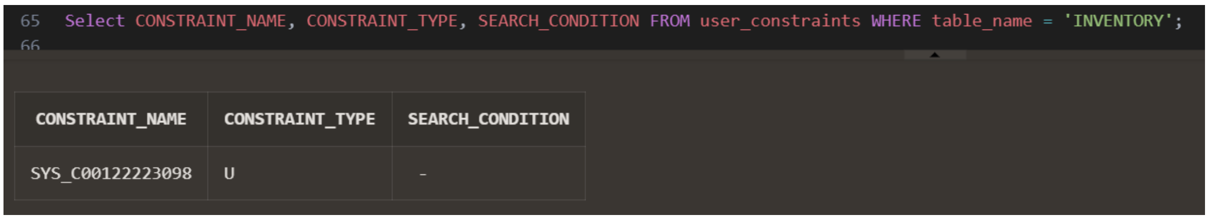
```
CREATE TABLE INVENTORY(
    BLOOD_TYPE VARCHAR(10) UNIQUE,
    AVAILABLE INT
);

INSERT INTO INVENTORY(BLOOD_TYPE,AVAILABLE) VALUES ('A+',15);
INSERT INTO INVENTORY(BLOOD_TYPE,AVAILABLE) VALUES ('A-',10);
INSERT INTO INVENTORY(BLOOD_TYPE,AVAILABLE) VALUES ('B+',20);
INSERT INTO INVENTORY(BLOOD_TYPE,AVAILABLE) VALUES ('B-',15);
INSERT INTO INVENTORY(BLOOD_TYPE,AVAILABLE) VALUES ('O+',20);
INSERT INTO INVENTORY(BLOOD_TYPE,AVAILABLE) VALUES ('O-',7);
INSERT INTO INVENTORY(BLOOD_TYPE,AVAILABLE) VALUES ('AB+',10);
INSERT INTO INVENTORY(BLOOD_TYPE,AVAILABLE) VALUES ('AB-',5);
```

Displaying INVENTORY Table:



BLOOD_TYPE	AVAILABLE
A+	15
A-	10
B+	20
B-	15
O+	20
O-	7
AB+	10
AB-	5

Displaying Constraints:

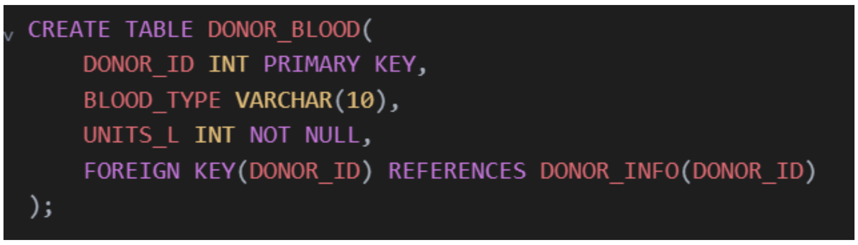


```
Select CONSTRAINT_NAME, CONSTRAINT_TYPE, SEARCH_CONDITION FROM user_constraints WHERE table_name = 'INVENTORY';
```

CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION
SYS_C00122223098	U	-

DONOR_BLOOD TABLE:

Creating DONOR_BLOOD table to give details about the donor's blood group

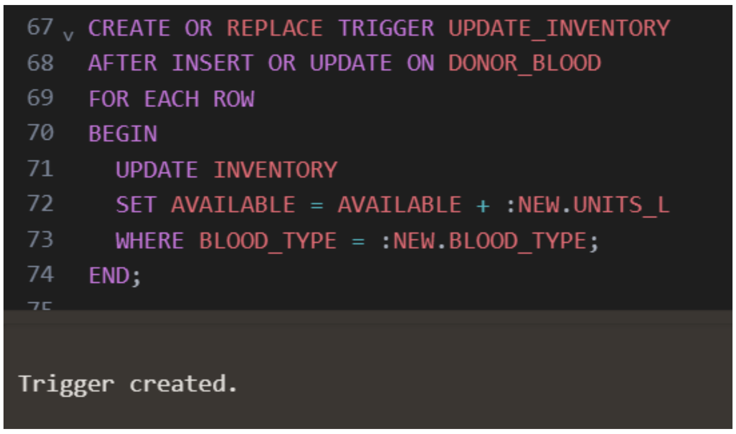


```
CREATE TABLE DONOR_BLOOD(
    DONOR_ID INT PRIMARY KEY,
    BLOOD_TYPE VARCHAR(10),
    UNITS_L INT NOT NULL,
    FOREIGN KEY(DONOR_ID) REFERENCES DONOR_INFO(DONOR_ID)
);
```

USING PL/SQL COMMAND TRIGGER:

In PL/SQL, a trigger is a special kind of stored procedure that is automatically executed in response to certain events, such as insertions, updates, or deletions of data in a database table. A trigger is defined to perform an action in response to an event that occurs on a particular table or view.

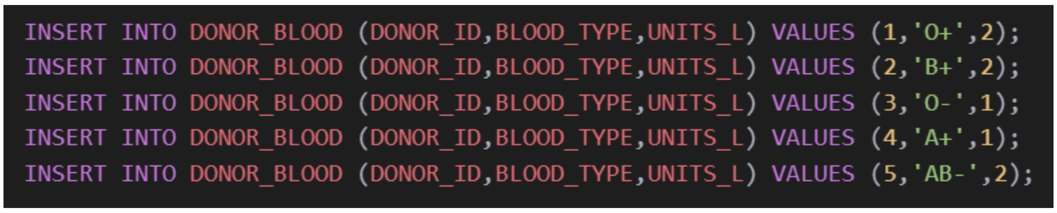
Here we have used Trigger to update the inventory evertime a donation is made:



```
CREATE OR REPLACE TRIGGER UPDATE_INVENTORY
AFTER INSERT OR UPDATE ON DONOR_BLOOD
FOR EACH ROW
BEGIN
  UPDATE INVENTORY
  SET AVAILABLE = AVAILABLE + :NEW.UNITS_L
  WHERE BLOOD_TYPE = :NEW.BLOOD_TYPE;
END;
```

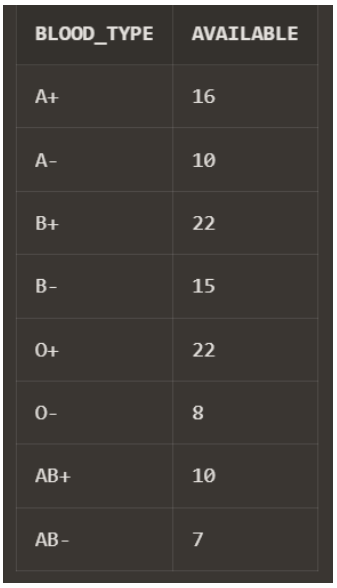
```
Trigger created.
```

Now, Making Insertions in the DONOR_BLOOD Table:



```
INSERT INTO DONOR_BLOOD (DONOR_ID,BLOOD_TYPE,UNITS_L) VALUES (1,'O+',2);
INSERT INTO DONOR_BLOOD (DONOR_ID,BLOOD_TYPE,UNITS_L) VALUES (2,'B+',2);
INSERT INTO DONOR_BLOOD (DONOR_ID,BLOOD_TYPE,UNITS_L) VALUES (3,'O-',1);
INSERT INTO DONOR_BLOOD (DONOR_ID,BLOOD_TYPE,UNITS_L) VALUES (4,'A+',1);
INSERT INTO DONOR_BLOOD (DONOR_ID,BLOOD_TYPE,UNITS_L) VALUES (5,'AB-',2);
```

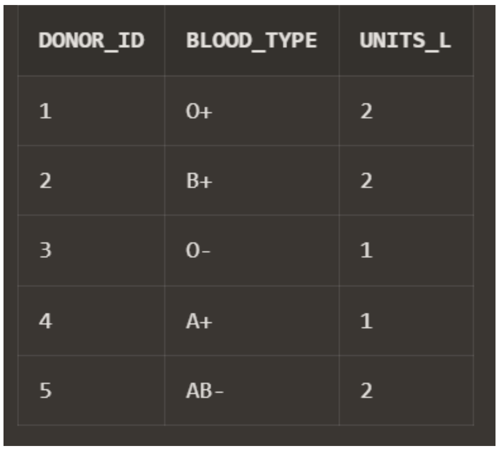
Trigger will now be invoked. Checking the inventory values now:



BLOOD_TYPE	AVAILABLE
A+	16
A-	10
B+	22
B-	15
O+	22
O-	8
AB+	10
AB-	7

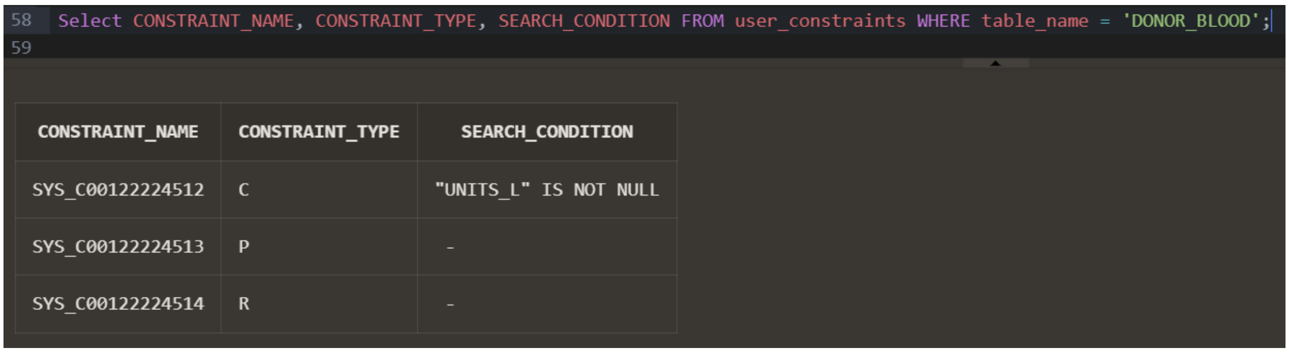
We see the INVENTORY table has been updated according to the donations made.

Displaying the DONOR_BLOOD Table:



DONOR_ID	BLOOD_TYPE	UNITS_L
1	O+	2
2	B+	2
3	O-	1
4	A+	1
5	AB-	2

Display constraints:

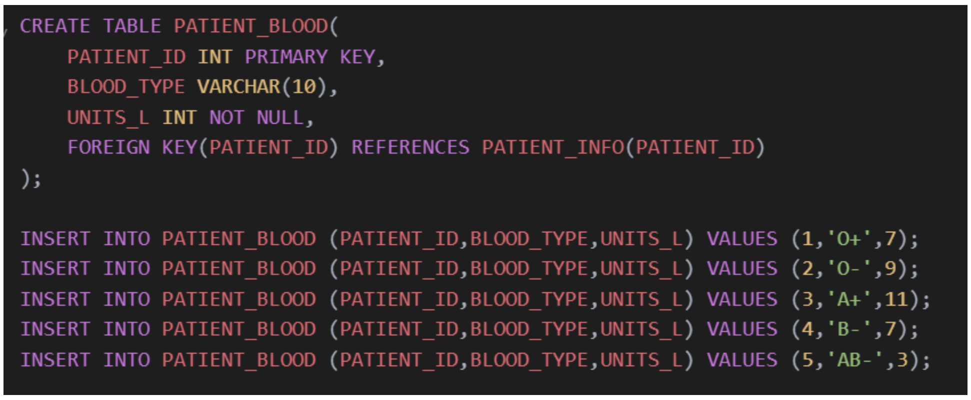


```
Select CONSTRAINT_NAME, CONSTRAINT_TYPE, SEARCH_CONDITION FROM user_constraints WHERE table_name = 'DONOR_BLOOD';
```

CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION
SYS_C00122224512	C	"UNITS_L" IS NOT NULL
SYS_C00122224513	P	-
SYS_C00122224514	R	-

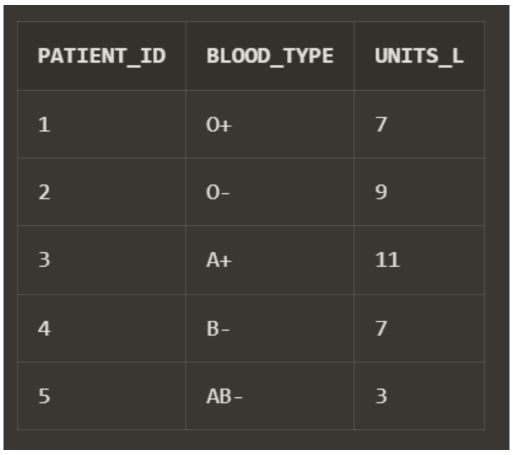
PATIENT_BLOOD TABLE:

Creating DONOR_BLOOD table to give details about the patient's blood group and how many units of blood is required, and inserting those values:



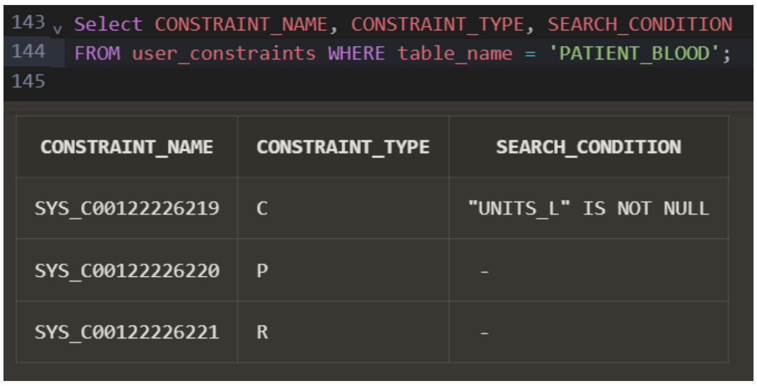
```
CREATE TABLE PATIENT_BLOOD(
    PATIENT_ID INT PRIMARY KEY,
    BLOOD_TYPE VARCHAR(10),
    UNITS_L INT NOT NULL,
    FOREIGN KEY(PATIENT_ID) REFERENCES PATIENT_INFO(PATIENT_ID)
);

INSERT INTO PATIENT_BLOOD (PATIENT_ID,BLOOD_TYPE,UNITS_L) VALUES (1,'O+',7);
INSERT INTO PATIENT_BLOOD (PATIENT_ID,BLOOD_TYPE,UNITS_L) VALUES (2,'O-',9);
INSERT INTO PATIENT_BLOOD (PATIENT_ID,BLOOD_TYPE,UNITS_L) VALUES (3,'A+',11);
INSERT INTO PATIENT_BLOOD (PATIENT_ID,BLOOD_TYPE,UNITS_L) VALUES (4,'B-',7);
INSERT INTO PATIENT_BLOOD (PATIENT_ID,BLOOD_TYPE,UNITS_L) VALUES (5,'AB-',3);
```

Displaying the PATIENT_BLOOD Table:



PATIENT_ID	BLOOD_TYPE	UNITS_L
1	O+	7
2	O-	9
3	A+	11
4	B-	7
5	AB-	3

Displaying constraints:



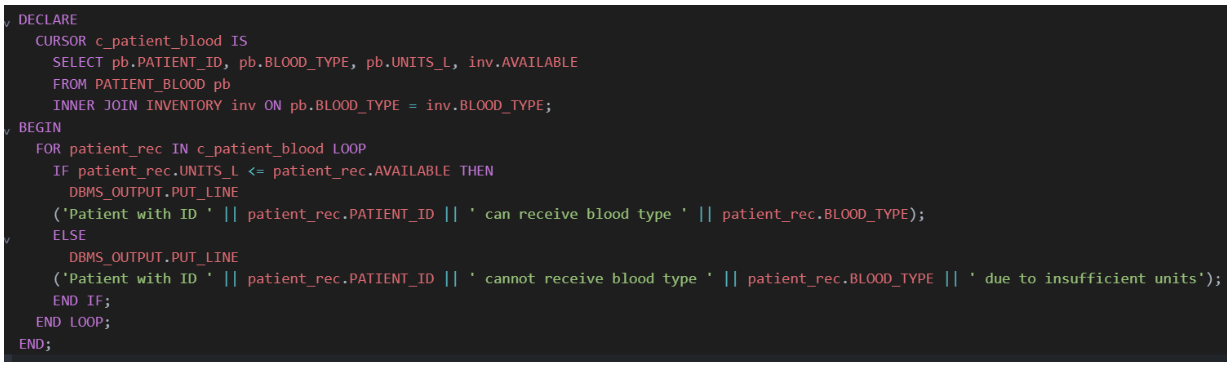
```
Select CONSTRAINT_NAME, CONSTRAINT_TYPE, SEARCH_CONDITION
FROM user_constraints WHERE table_name = 'PATIENT_BLOOD';
```

CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION
SYS_C00122226219	C	"UNITS_L" IS NOT NULL
SYS_C00122226220	P	-
SYS_C00122226221	R	-

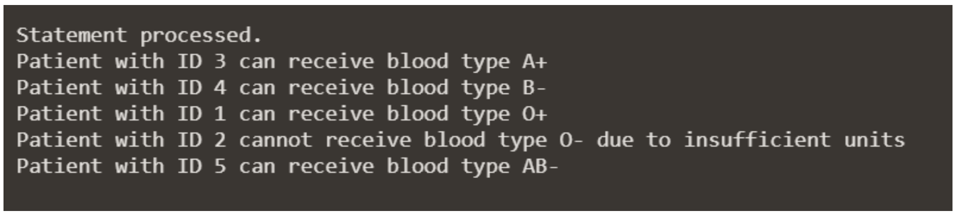
USING PL/SQL COMMAND CURSOR:

In PL/SQL, a cursor is a mechanism that enables you to traverse the rows returned by a SELECT statement one at a time. Cursors are useful when you need to process the result set of a query row by row, or when you need to update or delete specific rows in a result set.

Here, Using Cursor to check whether the units of bloods that the patient is requesting is even available in the blood bank:



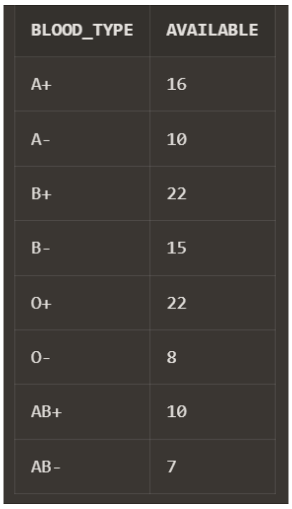
```
DECLARE
  CURSOR c_patient_blood IS
    SELECT pb.PATIENT_ID, pb.BLOOD_TYPE, pb.UNITS_L, inv.AVAILABLE
    FROM PATIENT_BLOOD pb
    INNER JOIN INVENTORY inv ON pb.BLOOD_TYPE = inv.BLOOD_TYPE;
BEGIN
  FOR patient_rec IN c_patient_blood LOOP
    IF patient_rec.UNITS_L <= patient_rec.AVAILABLE THEN
      DBMS_OUTPUT.PUT_LINE
    ('Patient with ID ' || patient_rec.PATIENT_ID || ' can receive blood type ' || patient_rec.BLOOD_TYPE);
    ELSE
      DBMS_OUTPUT.PUT_LINE
    ('Patient with ID ' || patient_rec.PATIENT_ID || ' cannot receive blood type ' || patient_rec.BLOOD_TYPE || ' due to insufficient units');
    END IF;
  END LOOP;
END;
```



```
Statement processed.
Patient with ID 3 can receive blood type A+
Patient with ID 4 can receive blood type B-
Patient with ID 1 can receive blood type O+
Patient with ID 2 cannot receive blood type O- due to insufficient units
Patient with ID 5 can receive blood type AB-
```

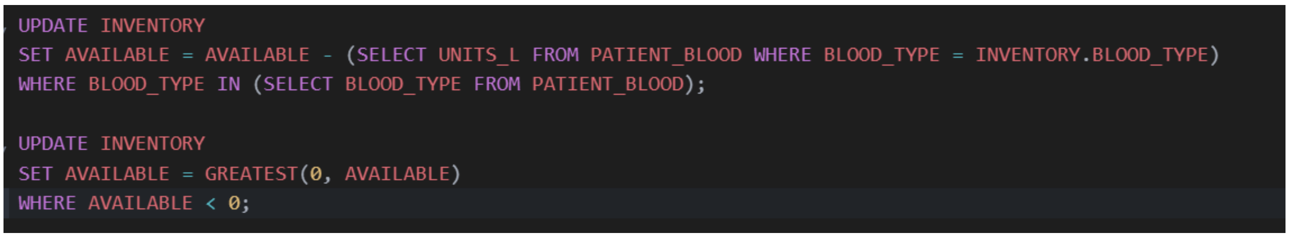
Running a Query to update Inventory as per the units needed by the patient:

Originally the inventory, after donation update was:



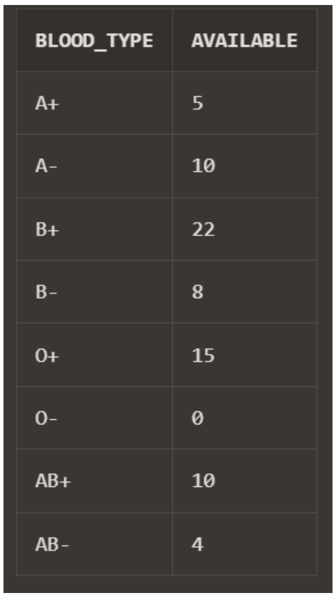
BLOOD_TYPE	AVAILABLE
A+	16
A-	10
B+	22
B-	15
O+	22
O-	8
AB+	10
AB-	7

Using a SQL query:



```
UPDATE INVENTORY
SET AVAILABLE = AVAILABLE - (SELECT UNITS_L FROM PATIENT_BLOOD WHERE BLOOD_TYPE = INVENTORY.BLOOD_TYPE)
WHERE BLOOD_TYPE IN (SELECT BLOOD_TYPE FROM PATIENT_BLOOD);

UPDATE INVENTORY
SET AVAILABLE = GREATEST(0, AVAILABLE)
WHERE AVAILABLE < 0;
```

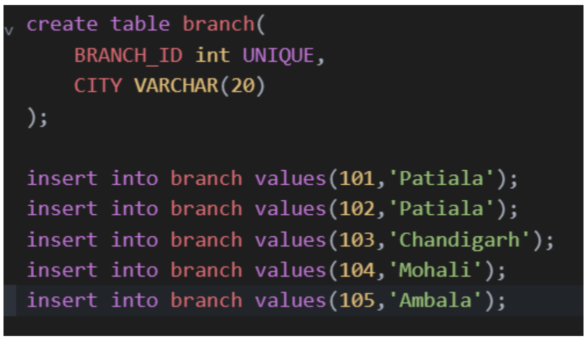
After updating, the inventory table becomes:



BLOOD_TYPE	AVAILABLE
A+	5
A-	10
B+	22
B-	8
O+	15
O-	0
AB+	10
AB-	4

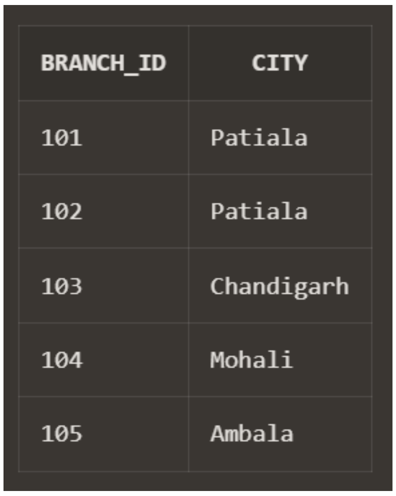
branch TABLE

Creating Table branch to give details of the blood donation centre, and inserting values in it:



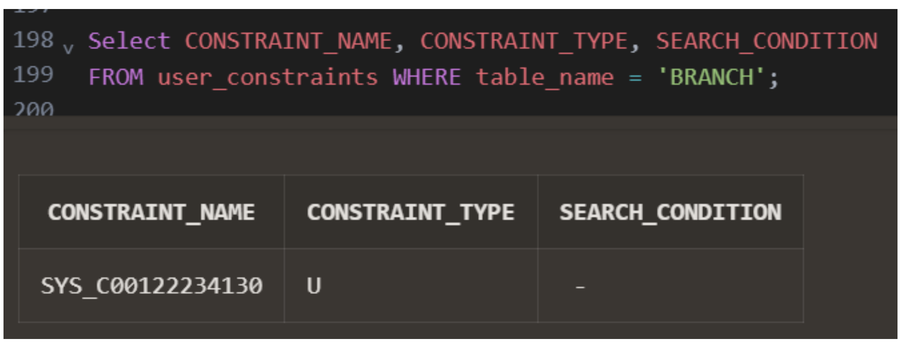
```
create table branch(
    BRANCH_ID int UNIQUE,
    CITY VARCHAR(20)
);

insert into branch values(101,'Patiala');
insert into branch values(102,'Patiala');
insert into branch values(103,'Chandigarh');
insert into branch values(104,'Mohali');
insert into branch values(105,'Ambala');
```

Displaying the table:



BRANCH_ID	CITY
101	Patiala
102	Patiala
103	Chandigarh
104	Mohali
105	Ambala

Displaying constraints:



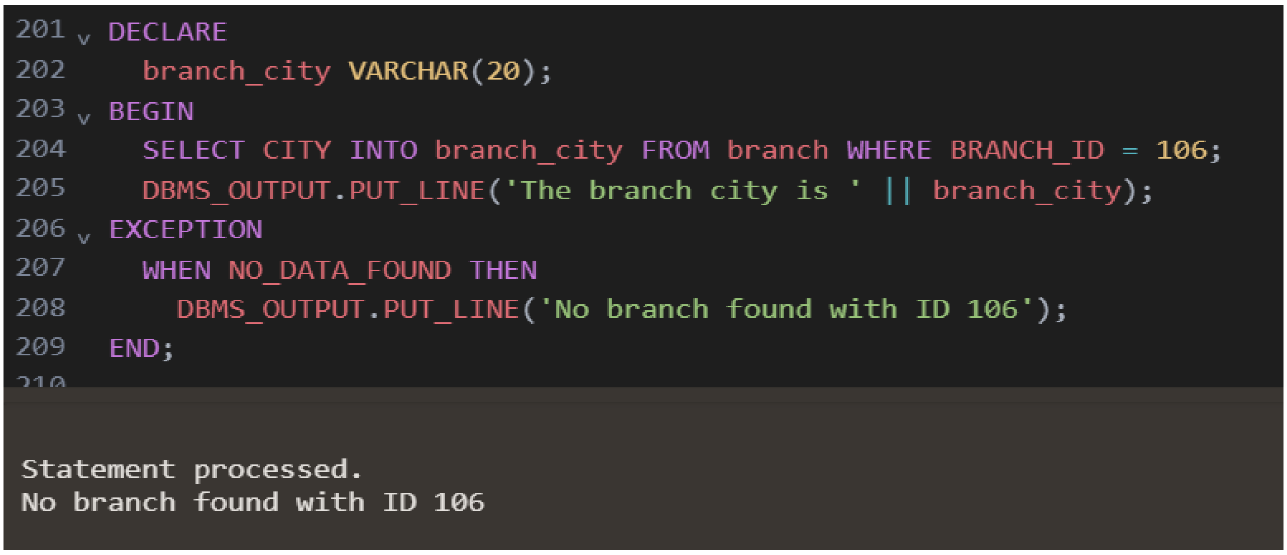
```
Select CONSTRAINT_NAME, CONSTRAINT_TYPE, SEARCH_CONDITION
FROM user_constraints WHERE table_name = 'BRANCH';
```

CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION
SYS_C00122234130	U	-

USING PL/ SQL COMMAND EXCEPTION HANDLING

FOR NO DATA FOUND

Exception handling is the process of dealing with runtime errors and unexpected events that occur during the execution of PL/SQL programs. PL/SQL provides several types of exceptions that allow you to handle errors and exceptions gracefully, without having to stop the execution of the program abruptly. Examples are **NO_DATA_FOUND**, **TOO_MANY_ROWS**, **INVALID_CURSOR** etc.



```
DECLARE
  branch_city VARCHAR(20);
BEGIN
  SELECT CITY INTO branch_city FROM branch WHERE BRANCH_ID = 106;
  DBMS_OUTPUT.PUT_LINE('The branch city is ' || branch_city);
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('No branch found with ID 106');
END;
```

```
Statement processed.
No branch found with ID 106
```

CONCLUSION

We designed well organized database management system ensuring an adequate supply of safe blood for patients in need.We have built a database for a Blood Bank using Oracle live 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 Oracle live 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.

REFERNCES

- https://livesql.oracle.com/apex/livesql/file/tutorial_BUE7ZQDL5G1XC855LKTWNACDO.html
- https://www.techonthenet.com/sql/update.php
- https://asktom.oracle.com
- https://stackoverflow.com