INDIVIDUAL REPORT

In this scenario, I have chosen a fictitious organization of a Blood Bank System called ‘Ek Rakhta’ for managing the blood bags that are received from the Blood Donation events. The business model which I have implemented is developed to keep a track of all the information that is been collected after the Blood Donation Event and supplying the blood to the Hospitals that ask for it. Many a times, there are situations where people lose their lives due to unavailability of blood, or due to poor blood management practices, so ‘Ek Rakhta’ has taken the initiative of making sure that there is always a good amount of blood available in the hospitals and there is also a proper management for blood requirements.

Every month, this organization holds Blood Donation Events which is one of the way to increase the blood stock. This model can assist the information of the blood bag during its handling, right from the donation till the hospital needs for patients. There are many functions taken up by the Blood Bank Management System, but we are just focussing on the Blood Donation side area for increasing the system performance, reliability and throughput. ‘Ek Rakhta’ is looking for a database management system where we can maintain all the records such as the list of donors, quantity of blood that is been donated, the hospitals that are associated with the blood bank and the patient demands for a particular blood group. Thus, there is a need to design the database management system for this organization.

There are some assumptions we need to take into consideration:

1. The database should have different section for donor and the donor can donate blood number of times after a particular span of time.
2. The donor has to register with the Donor Blood Bank before donating the blood.
3. The amount of blood donated by the donor is also noted.
4. The Blood Bank has the record of the different types of blood groups and shows its status, whether available or not.
5. The Hospital asks the Blood Bank for its specific blood group requirements.
6. The Hospital maintains the records of its Patients, their blood group and the amount of blood required by them.
7. The phone number of the Hospital can be Null.
8. The ‘Donor_Blood_Bank’ is a linking table, that has both the donorID and the bloodID stored in it.

What is RDBMS? What is the use of RDBMS for our Blood Bank?

A Relational Database is structured, that means the data is organized in tables. Most of the times, the data within these tables have relationships or dependencies with one another. In this scenario, I have used SQLite to organize the data in tables and then I have used it to execute queries, retrieve data, and edit data by updating, deleting or creating new records. The SQLite works by linking information from multiple tables through the use of ‘keys’. So we can link the tables using the primary key and foreign key to allow easy sorting between the tables. Here in this example, we have some primary keys which are unique like donorID, bloodID etc. and also the foreign keys to iterate through the tables. Relational database can be well used for the structured data of donors that have donated blood, blood bank where the blood is available, so it is very well suited for this organization.

The Normalization feature of Relational database can reduce space by eliminating data redundancy. Like in the Blood table, using the Donor ID we can see the Name of the donor by simply referencing and by avoiding duplicate or multiple columns of data.

Using SQLite allows us to delete and add new donors to the table, and that data will be saved consistently in the database, without any duplicate entries, so Relational database also allows us to keep the data consistent. Simply storing the data in a spreadsheet can cause anomalies, so the assigned person must update the database in a timely manner, otherwise it can tamper the data.

What is Redis Database? How can Redis prove useful for our Blood Bank?

Redis is an open source, in-memory data structure store, used as a key-value database, where keys are used to uniquely identify the associated values. So, we have used Redis for the same reason that it stores the values in the form of keys and values i.e., every key has its individual value. This individual key is called as the primary key for this database.

The Blood Bank sometimes has data showing orders which can be null, so clearly this data cannot be structured since all some orders also would not be placed. As Relational database does not work with unstructured data, we introduce the use of non-relational database. Thus, Redis will be an ideal choice for the Blood Bank.

The data in the Redis keys are fetched from the SQL table and then they are converted into hash keys. These hash keys contain the attributes of the table in the form of keys and values.

So the Redis database, is made up of hash keys and every single Index key comprises of the details from the Blood Bank Table for example orders, status etc. It stores the data in the form of keys and values and not in table form as that in the case of Relational database. This makes it easy for sorting and computing through the data.

As Redis does not need fixed structures and fixed attributes, it shows the Flexibility property of the database. Consider in this example, if the hospital does not want to give their phone number or the orders in the Blood Bank show null, if we had used tables they would have blank entries thus making the tables inefficient. So in those cases, non-relational databases is recommended since it can store unstructured data.

Also, as Redis is open-source, it is very cost-effective unlike the Relational database that require licenses.

POLYGLOT PERSISTENCE:

Polyglot Persistence is the concept to understand that when storing data, it is best to use multiple data storage technologies, which depend on how the data is being used by the application or component. For example, consider a case when we use more than 1 database storage technology to store the data, it shows polyglot persistence. Here we have used both SQL and Redis to store the data received from the donor and to display the data from the blood bank respectively. We would be using Python to demonstrate the Polyglot Persistence.

How have I used polyglot persistence in the case of Blood Bank?

Initially, I have used the Blood bank table, I have selected the data having its ‘status’ as ‘Available’. Then I have converted it to a Data Frame using Pandas and I have converted the Blood Bank ID as the Index key, followed by converting it to a Python Dictionary. The Index Key is then supplied to the pipeline function of the Redis database and is converted into Hash keys.

Further I have defined four functions, bloodAvailability(), update_unavail(), update_avail() and update_bloodBank().

The bloodAvailability() function checks the ‘status’ of the blood available with the Blood bank, whether the blood is ‘Available’ or ‘Unavailable’, if the ‘status’ is ‘Available’, it will return ‘True’ otherwise it returns ‘False’.

The update_unavail() function will set the ‘status’ = ‘Unavailable’ if the bloodAvailability() function gives the output as ‘Available’, it will also update the redis key value with ‘Available’ status.

The update_avail() function will set the ‘status’ = ‘Available’ if the bloodAvailability() function gives the output as ‘Unavailable’, it will also update the redis key value with ‘Unavailable’ status.

Lastly, we run the update_bloodBank() function, which considers two situations,

If the blood is available, it will run the bloodAvailability() function, which will check whether the Blood is Available or not, if it is available, it will run the update_unavail(). As a result, update_unavail() will update the status of the blood as ‘Unavailable’, and display a message ‘Blood is made Unavailable’ then afterwards it will update the hash key. Now whenever we run the run the hash key, it shows the same updated value as SQL consists for that id.

If the blood is Unavailable, it will run the bloodAvailability() function, which will check whether the Blood is Available or not, if it is Unavailable, it will run the update_avail(). As a result, update_avail() will update the status of the blood as ‘Available’, and display a message ‘Blood is made Available’ then afterwards it will update the hash key. Now whenever we run the run the hash key, it shows the same updated value as SQL consists for that id.

The Table Information with its characteristics can be shown below in Fig 1.

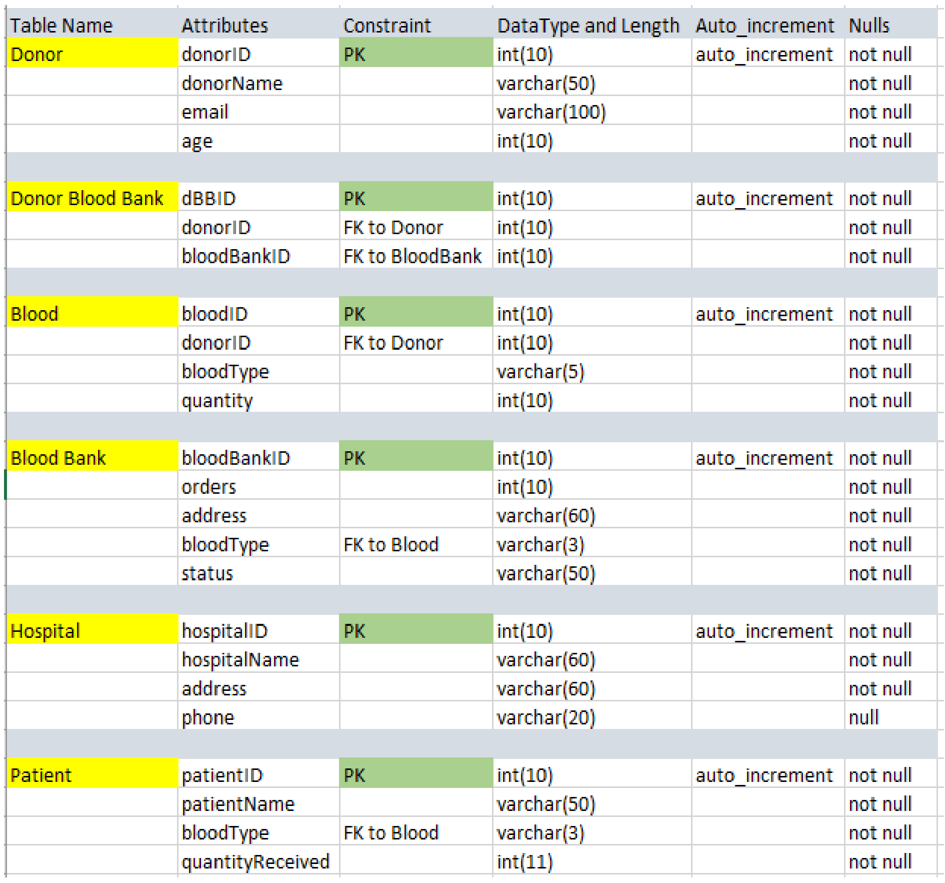

Table Name	Attributes	Constraint	DataType and Length	Auto_increment	Nulls
Donor	donorID	PK	int(10)	auto_increment	not null
	donorName		varchar(50)		not null
	email		varchar(100)		not null
	age		int(10)		not null
Donor Blood Bank	dBBID	PK	int(10)	auto_increment	not null
	donorID	FK to Donor	int(10)		not null
	bloodBankID	FK to BloodBank	int(10)		not null
Blood	bloodID	PK	int(10)	auto_increment	not null
	donorID	FK to Donor	int(10)		not null
	bloodType		varchar(5)		not null
	quantity		int(10)		not null
Blood Bank	bloodBankID	PK	int(10)	auto_increment	not null
	orders		int(10)		not null
	address		varchar(60)		not null
	bloodType	FK to Blood	varchar(3)		not null
	status		varchar(50)		not null
Hospital	hospitalID	PK	int(10)	auto_increment	not null
	hospitalName		varchar(60)		not null
	address		varchar(60)		not null
	phone		varchar(20)		null
Patient	patientID	PK	int(10)	auto_increment	not null
	patientName		varchar(50)		not null
	bloodType	FK to Blood	varchar(3)		not null
	quantityReceived		int(11)		not null

Fig 1. Table Information with Characteristics

The Entity-Relationship(ER) diagram below shows relationship and participation and cardinality between 6 entities: Donor, Donor Blood Bank, Blood, Blood Bank, Hospital and Patient.

The relationship amongst the entities can be understood by:

1. One Donor Blood Bank can register many Donors. But one Donor can be registered only in one Donor Blood Bank. This is one to many relationship.
2. Blood can be donated by only one Donor, but one Donor can donate Blood many times. This is one to many relationship.
3. Many Donor Blood banks can give Blood to many Blood Banks. Many Blood Banks can give blood to many Donor Blood banks.
4. Many Blood Banks can order Blood from many Hospitals. Many Hospitals can order blood from many Blood Banks.
5. One Hospital can give blood to many Patients in the hospital. One Patient can receive blood only from the Hospital it is admitted to.

ENTITY-RELATIONSHIP DIAGRAM:

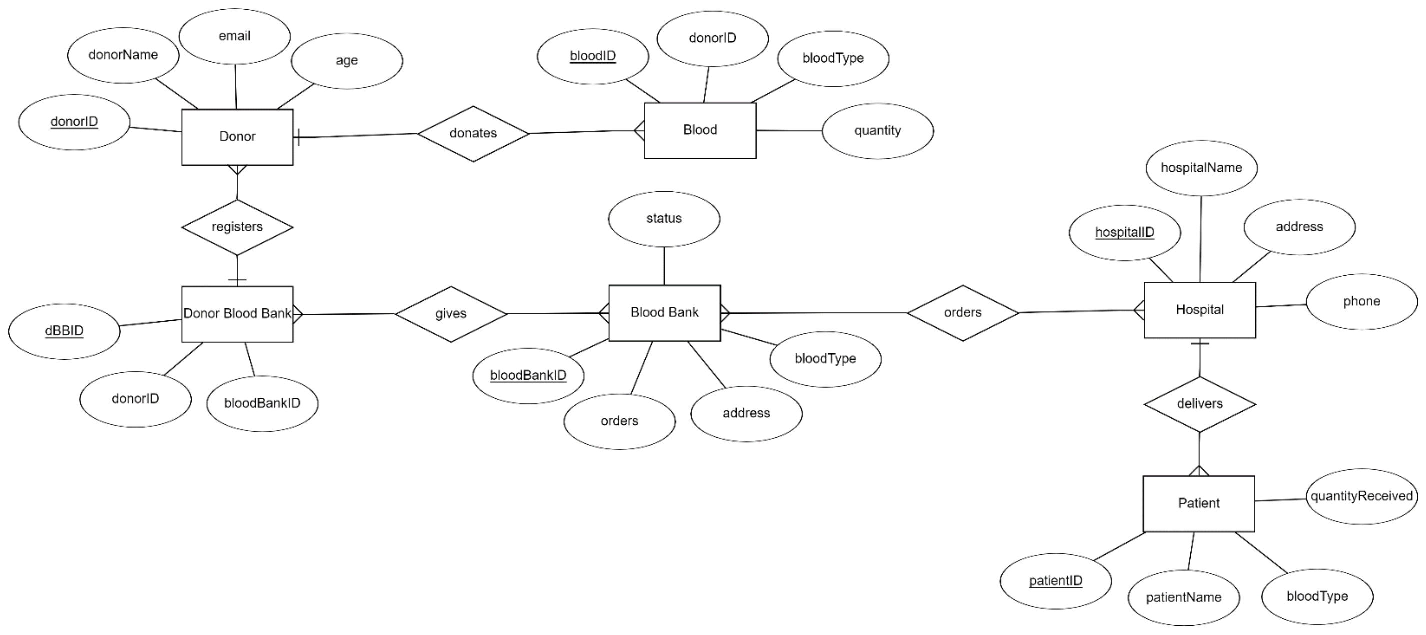

Fig 2. Entity-Relationship Diagram for Blood Bank.

Logical Database Design:

The Logical database design gives information about Primary and Foreign keys.

Donor Blood Bank (registers) Donor: 1.M Relationship
Donor Blood Bank (gives) Blood Bank: M.M Relationship
Donor Blood Bank: (dBBID, donorID, bloodBankID)
Primary key: dBBID
Foreign key: donorID reference the Donor on Update Cascade on Delete No Action.
bloodBankID reference the Blood Bank on Update Cascade on Delete No Action.

- Table is in 1st normalized form as it doesn’t contain any multi-value column(attribute)
- Table is in 2nd normalized form, it doesn’t contain any composite primary keys.
- Table is in 3rd normalized form, as non-primary key attributes cannot be determined except primary key and candidate key.

Donor (donates) Blood: 1.M Relationship
Donor (registers) Donor Blood bank: M.1 Relationship
Donor : (donorID, donorName, email, age)
Primary key: donorID
Alternate key: donorName, email

- Table is in 1st normalized form as it doesn’t contain any multi-value column(attribute)
- Table is in 2nd normalized form, it doesn’t contain any composite primary keys.
- Table is in 3rd normalized form, as non-primary key attributes cannot be determined except primary key and alternate key(i.e. donorName, email).

Blood (donates) Donor : M.1 Relationship
Blood : (bloodID, donorID, bloodType, quantity)
Primary key: bloodID
Foreign key: donorID reference the Donor on Update Cascade on Delete No Action.

- Table is in 1st normalized form as it doesn't contain any multi-value column(attribute)
- Table is in 2nd normalized form, it doesn't contain any composite primary keys.
- Table is in 3rd normalized form, as non-primary key attributes cannot be determined except primary key and candidate key.

Blood Bank (gives) Donor Blood Bank: M.M Relationship
Blood Bank (orders) Hospital: M.M Relationship
Blood Bank: (bloodBankID, orders, address, bloodType, status)
Primary key: bloodBankID
Foreign key: bloodType reference the Blood on Update Cascade on Delete No Action.

- Table is in 1st normalized form as it doesn't contain any multi-value column(attribute)
- Table is in 2nd normalized form, it doesn't contain any composite primary keys.
- Table is in 3rd normalized form, as non-primary key attributes cannot be determined except primary key and candidate key.

Hospital (orders) Blood Bank: M.M Relationship
Hospital (delivers) Patient: 1.M Relationship
Hospital: (hospitalID, hospitalName, address, phone)
Primary key: hospitalID
Alternate key: hospitalName, address.

- Table is in 1st normalized form as it doesn't contain any multi-value column(attribute)
- Table is in 2nd normalized form, it doesn't contain any composite primary keys.
- Table is in 3rd normalized form, as non-primary key attributes cannot be determined except primary key and alternate key(i.e. hospitalName, address).

Patient (delivers) Hospital: M.1 Relationship
Patient: (patientID, patientName, bloodType, quantityReceived)
Primary key: patientID
Foreign key: bloodType reference the Blood on Update Cascade on Delete No Action.

- Table is in 1st normalized form as it doesn't contain any multi-value column(attribute)
- Table is in 2nd normalized form, it doesn't contain any composite primary keys.
- Table is in 3rd normalized form, as non-primary key attributes cannot be determined except primary key and candidate key.

Whenever we try to delete any of the record from the parent table, without deleting it from the child table, then it would not allow us to delete the entry directly. This is the functionality of 'Delete No Action on Update Cascade'. Thus, we have to firstly delete the child table entry where parent attribute is used as a foreign key. Then only would we be able to delete record from the parent table.