BLOOD BANK DATABASE MANAGEMENT SYSTEM

ASSOCIATED WITH BLOOD WARRIORS DAYTON

Final Report

Data Added, SQL Queries Created and Final Report

by

Vamshidher Reddy K

May 03, 2022

Database Proposal (1)

Description of Database:

Blood Warriors Dayton is a nonprofit organization to help Thalassemia kids that is based in Dayton, Ohio. Sandeep K who is the CEO of this organization requires a database for tracking and managing her data. Currently Mr. Krishna, One of the crucial team members keeps track of data by a haphazardly, lackadaisical approach of Excel files, handwritten and computerized documentation, and receipts. Inevitably, whenever early and monthly reports are required, it takes him an exorbitant amount of time to complete. Krishna expressed distress over the time wasted during gathering this information and subsequently asked me to design and implement a database to help facilitate faster reporting.

Mr. Krishna desires a database that is extremely user-friendly; so that at a glance he can see each monthly report. Some of the data that he currently stores is as follows: blood donations collected, amount of blood needed for each monthly project, amount of blood need to raise, and balance. Some of this information is put on his letterhead, and the year's donations for that particular donor are itemized. Additionally, every donor receives an itemized list of donations given. These sponsors then submit this document when filing for their income tax returns.

In addition, Mrs. Doe expressed interest in exporting some information in order to generate mailing address labels, which she uses monthly for hundreds of sponsors when mailing out her activity reports & thank you letters. However, almost every time I speak with her or her colleagues, I find different information that needs to be tracked. So, I expect the database to change and grow through this process of developing a DB for her.

Therefore, for the scope of this class and to prevent project requirement creep I am going to lock the database into these following functions and requirements.

- Doctor examining the donor’s details and eligibility for blood donation.
- Donor donating the type of blood.
- Blood bank collecting the blood units.
- Blood Bank updating and storing the type of blood and number of units collected
- Blood Bank delivering the blood units to the patient.

Description of Entities:

Here is a current list of entities and their relationships in regards to their foreign keys.

- doctor entity:

Description: the doctor's entity is designed to track individual doctor's information such as name, identification number, primary phone number and resident city which is considered as address. Phone number and email address could be multivalued attributes however the CEO does not want this. He only tracks one phone number and one email for a given doctor. For a better understanding of the information that is tracked for the entity see the logical schema below.

doctor (doctor_id, doctor_name, doctor_phno, doctor_add)

- donor entity:

Description: the donor's entity is designed to track individual donor's information such as identification number, primary phone number, gender, iron content, date of birth, weight, blood pressure (for which mean artery pressure is considered) and resident city. Phone number could be multivalued attributes however the CEO does not want this. He only tracks one phone number and one email for a given donor. For a better understanding of the information that is tracked for the entity see the logical schema below.

donor (donor_id, donor_name, donor_gender, donor_address, donor_ph_no, iron_content, DOB, weight, blood_pressure, *doctor_doctor_id*)

FK *doctor_doctor_id ~ doctor*

- patient entity:

Description: the patient's entity is designed to track individual patient's information such as name, identification number, primary phone number, the hospital address in which the

patient has been admitted, and resident city which is considered as address. Phone could be multivalued attributes however the CEO does not want this. He only tracks one phone number for a given patient. For a better understanding of the information that is tracked for the entity see the logical schema below.

patient (patient_id, patient_name, patient_ph_no, patient_add, hosp_add)

- blood_bank entity:

Description: the bloodbank's entity is designed to track individual bloodbank's information such as name, identification number and the city in which the bloodbank is present. For a better understanding of the information that is tracked for the entity see the logical schema below.

blood_bank (blood_bank_id, blood_bank_name, blood_bank_add)

- doctor_examines_donor entity:

Description: the doctor_examines_donor's entity is designed to break up the many to many relations between the doctor entity and the donor entity. Its main goal is to maintain the data of the patient's information and the doctor's information whoever tested them. For a better understanding of the information that is tracked for the entity see the logical schema below.

doctor_examines_donor (doctor_doctor_id, donor_donor_id)

FK *doctor_doctor_id ~ doctor*

FK *donor_donor_id ~ donor*

- blood entity:

Description: the blood entity is designed to maintain the track of the blood information between donor entity and the blood_bank entity. Its main goal is to track the blood type donated by the donor, along with the donor's information and the blood_bank's information in which it has been stored. For a better understanding of the information that is tracked for the entity see the logical schema below.

blood (donor_donor_id, blood_type, blood_bank_blood_bank_id)

FK *donor_donor_id ~ donor*

FK *blood_bank_blood_bank_id ~ blood_bank*

- blood_delivery entity:

Description: the blood_delivery entity is designed to break up the many to many relations between the blood_bank entity and the patient entity. Its main goal is to maintain the data of the blood_bank's information from which the blood has been delivered and the patient's information to whichever the blood has been delivered. For a better understanding of the information that is tracked for the entity see the logical schema below.

blood_delivery (patient_patient_id, blood_bank_blood_bank_id)

FK *patient_patient_id ~ donor*

FK *blood_bank_blood_bank_id ~ blood_bank*

Relationships EER diagram

Moreover, here is an EER diagram that shows the relationships between entities and their attributes. Note: all arrows indicate relationship; an arrow pointing to an entity indicates a one to many relationships. All entities in this database have a one to many relationships and are hopefully normalized.

Description of Transactions

List of Tasks Performed on Database:

- Doctor examines the donor's details and eligibility for blood donation.
- Donor donates the type of blood.
- Blood bank collects the blood units.
- Blood Bank updates the type of blood and number of units collected
- Blood Bank delivers the blood units to the patient

ER Model (2)

Description

The database is designed for a non-profit organization. It will be used to keep track of its volunteers including their doctors, donors, and their dependent patients. In addition, the database will store information on the donors as well as their blood donations, including which project their donation was allocated to.

When a donor joins the organization, their health condition and contact information will be stored in the database along with their date of birth, weight and iron content. In addition, the database will keep track of whether this donor is eligible to donate the blood which is tested by the doctor. Blood information will also be stored, including donor who donated it and who managed to test the donor. If a volunteer has an expense related to a project that information is also stored in the database, including whose expense it was and which project it is related to. For those donors giving to the organization, their contact information is stored in addition to a record of all donations, and which blood bank's and patient's those donations were allocated to.

Entities

doctor {doctor_id, doctor_name, doctor_phno, doctor_add}
doctor_examines_donor {doctor_doctor_id, donor_donor_id }
donor {donor_id, donor_name, donor_gender, donor_address, donor_ph_no, iron_content, DOB, weight, blood_pressure, *doctor_doctor_id*}
blood {donor_donor_id, blood_type, blood_bank_blood_bank_id }
blood_bank {blood_bank_id, blood_bank_name, blood_bank_add}
blood_delivery (patient_patient_id, blood_bank_blood_bank_id)
patient {patient_id, patient_name, patient_ph_no, patient_add, hosp_add}

Here will be a brief overview of some of the crucial assumptions about Blood Warriors Dayton database.

1. The **doctor** entity is designed to store information about doctors. In the Blood Warriors Dayton database, the doctor entity stores the information about the doctor id, name, doctor's contact number and the residing city.

   There is an N: M relationships called **doctor_examines_donor** between **doctor** and **donor** and in this relationship **doctor** and **donor** has total participation.

   a. Explanations and Assumptions:
   i. N: M: A **doctor** can test many **donors**, and many **donors** can be tested by many **doctors**.

      1. *Total participation: There is total participation between **doctor** and **donor** because every **doctor** is allocated to a **donor**.* Blood Warriors Dayton *cannot indefinitely keep a donor. It must get allocated out to one of the **doctors**.*

2. The **doctor_examines_donor** entity is designed to test donors that were registered for a **donor** by certified **doctor**. The relationships that the **doctor_examines_donor** has between **donor** and **doctor** will be explained in each of their paragraphs (see number 1 for **doctor** and see number 3 for **donor**).
3. The **donor** entity is designed to track specific information for a given **donor**. Each tuple in the **donor** entity inherits key information about itself; for example, donor's name, birthdate, and so on. The relationship between doctor and donor is explained in 1.

   There is also a 1:1 relationship called **donates** between **donor** and **blood.**

   a. Explanations and Assumptions:
   i. 1:1: A **donor** can donate only one type to **blood** and a particular blood can belong to one donor with all other attributes like iron content considered.

      1. *Total Participation: Every blood must have a donor.*
      2. *Partial Participation: Not every donor donates blood.*

4. **blood** entity is designed to track all donations that have been given to Blood Warriors Dayton by their **donors**. This entity is a weak entity type and inherits from the neighbor entity specific attributes. The entity has a relationship not just with **donors** but with **blood banks** as well. The relationship between donors and blood has already been defined in 2.

   There is an N: M relationships called **stored in** between **blood** and **blood_banks** and in this relationship **blood** has total participation while blood_bank only has partial participation.

   a. Explanations and Assumptions:

   i. N: 1: Many **blood** samples can be stored in single **blood_bank**, and not all **blood_bank's** have one or more blood

      1. *Total participation: There is total participation between **blood** and **blood_bank** because every **blood** is allocated to a **blood_bank.** Blood Warriors Dayton cannot indefinitely keep a donation. It must get allocated out to one of the **blood_banks**.*
      2. *Partial participation: Meanwhile, not every **blood_bank** necessarily gets a **blood sample**. There are many **blood_bank's** that Blood Warriors Dayton has the difficulty to fulfill the requirement. Additionally, often **blood** is not preemptively planned and collected unless available. During this stage, the **blood_banks** are not allocated any blood requirement.*

5. **blood_bank** entity is designed to track current, future or past donations of blood that Blood Warriors Dayton has completed. It has multiple relationships between the **blood** entity which is discussed in 3.

   It also has a N:M relationship called **blood_delivery** between **blood_bank** entity and patient entity.

   a. Relationships, Explanations, and Assumptions:

i. N: M: A **blood_bank** and **patient** have a **blood_delivery** relationship. This is because a given blood_bank can have many **blood** items and many **blood** items can be donated to different **patients**.

0. *Partial Participation: Moreover, not every blood_bank necessarily has a blood that is required by the patient.*
1. *Total Participation: Every blood item must belong to some patient.*

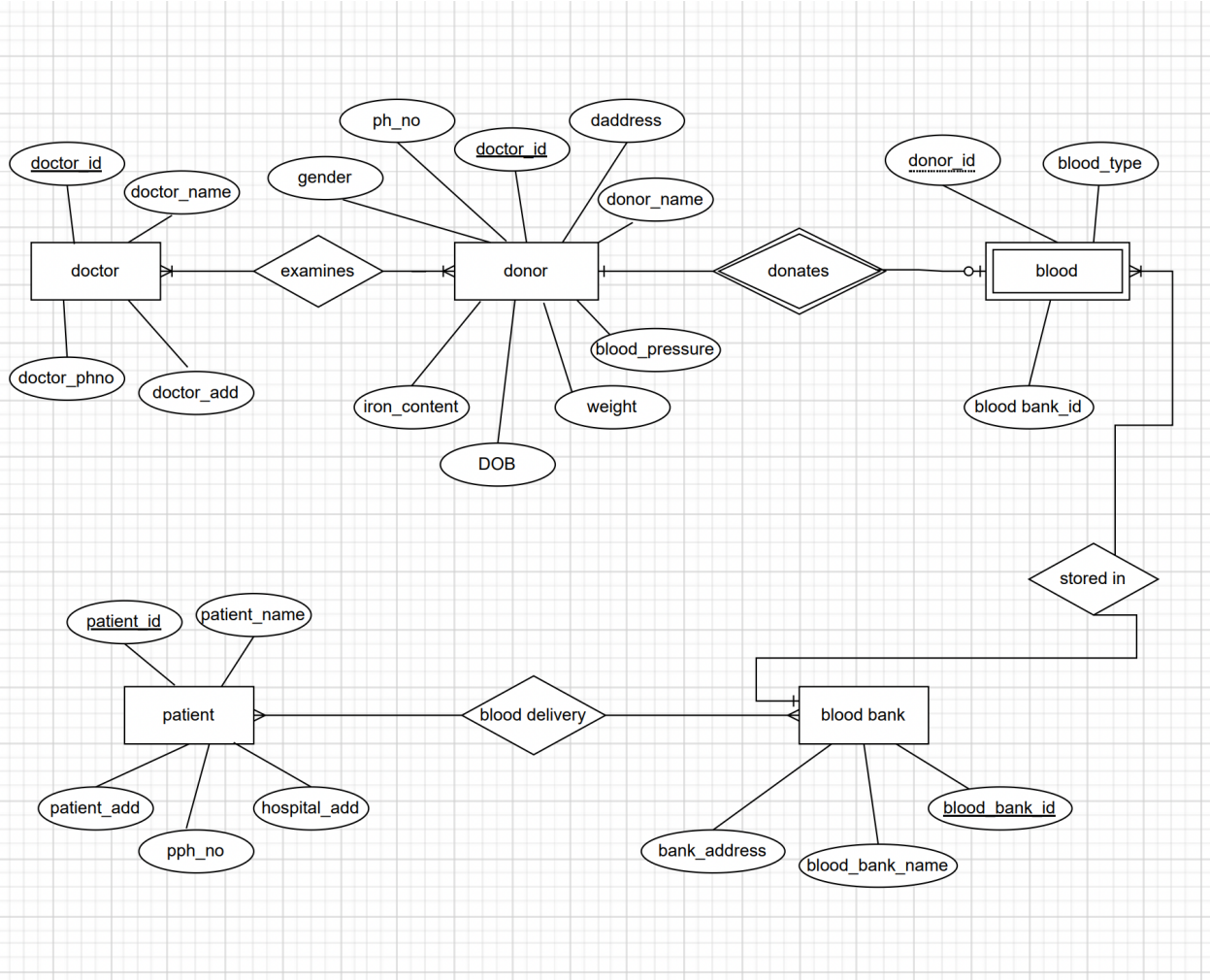
6. The **blood_delivery** entity is designed to track blood items that were being received from **blood banks** which later used for a **patient** as per the requirement. The relationships that the **blood_delivery** item has between **blood banks** and **patient** will be explained in each of their paragraphs (see number 5)
7. The **patient** entity is designed to track any **patient's** information like name, contact number, residing city along with the hospital in which the patient has been admitted. It also has a N:M relationship called **blood_delivery** between **blood_bank** entity and **patient** entity as discussed in the point 5.

a. Relationships, Explanations, and Assumptions:

i. N:M: A **blood_bank** and **patient** have a **blood_delivery** relationship. This is because a given blood_bank can have many **blood** items and many **blood** items can be donated to different **patients**.

0. *Partial Participation: Moreover, not every blood_bank necessarily has a blood that is required by the patient.*
1. *Total Participation: Every blood item must belong to some patient.*

Diagram

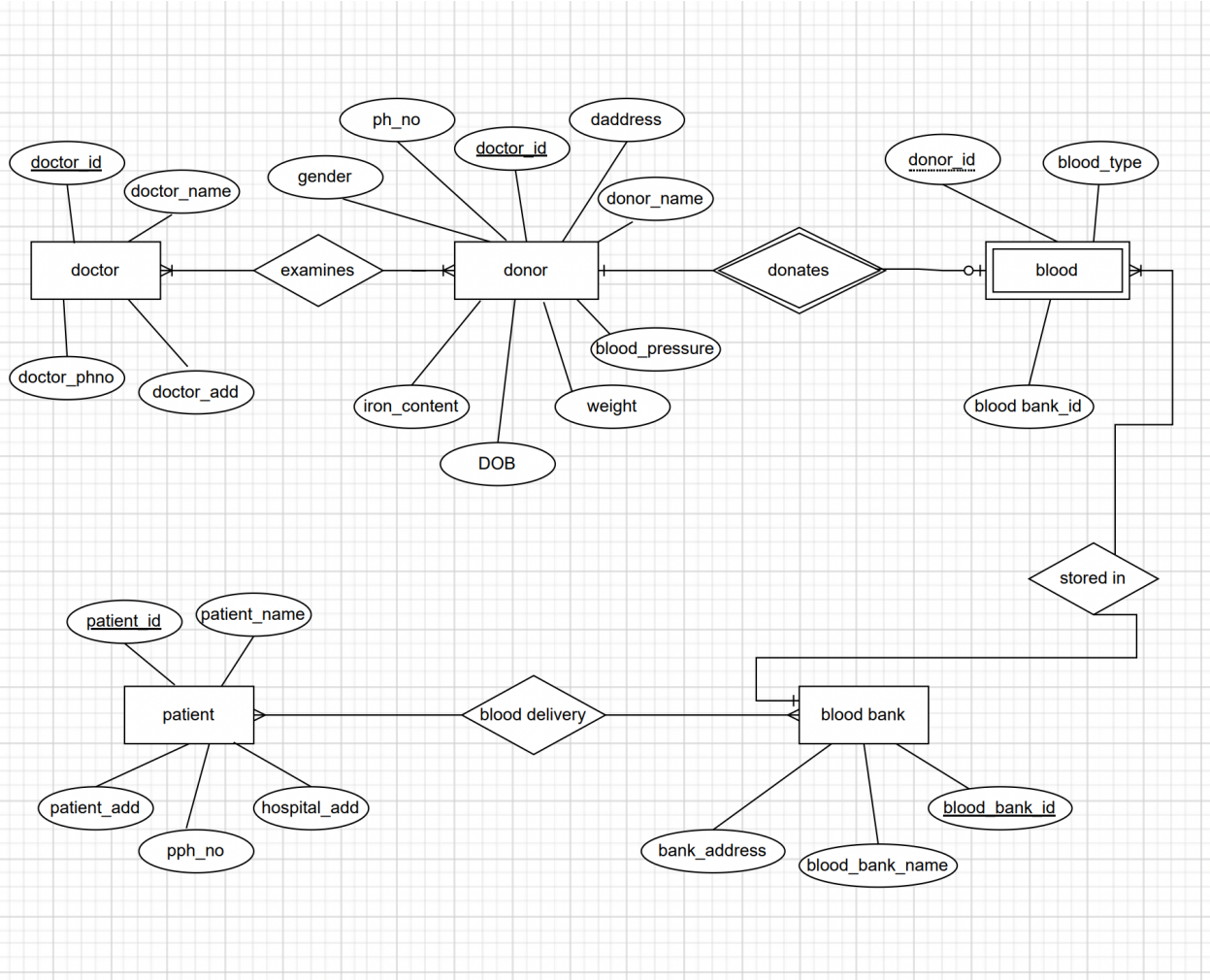

doctor_id
doctor_name
doctor
doctor_phno
doctor_add
examines
ph_no
doctor_id
daddress
gender
donor_name
donor
iron_content
weight
DOB
blood_pressure
donates
donor_id
blood_type
blood
blood bank_id
stored in
patient_id
patient_name
patient
patient_add
hospital_add
pph_no
blood delivery
blood bank
bank_address
blood_bank_name
blood_bank_id

Relational Schema-Normalization (3)

Step 1

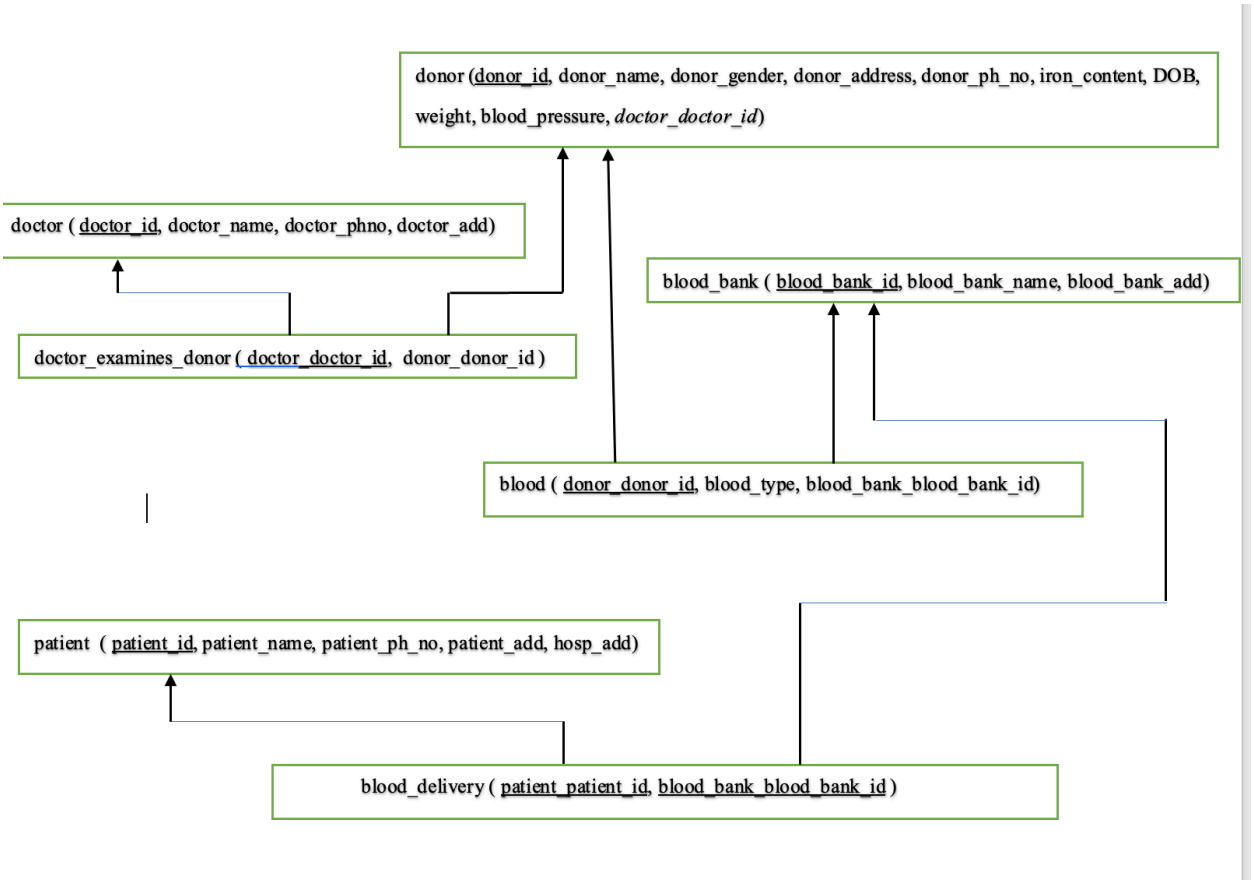
In this section of our project we are changing the EER diagram to the relational model. In order to facilitate easy grading we will show the EER model and then the relational model side by side.

- EER model:



Relational model:

This is our un-normalized relational model.



Step 2

In this section we will normalize each relationship and list each relationship's functional dependencies. In order to speed up the process of listing each relationships functional dependencies we are going to let the first attribute in a table be represented by the alphabet letter A, the next attribute in the relation will be represented by B and this process will continue for every attribute in the relation. For example: the Doctor's relation will be represented by the following alphabet characters

Doc (A|B|C|D|E|F|G|H)

where **Doc** represents the relationship and {**ABCD**} are the set attributes in the same order they appear in the relation.

First Normal Form

All the relations in the above relational model are already in first normal form. This is because every relation listed above has no multivalued attributes (i.e., atomic) and each relation has a primary key as well as a correct domain type.

The Doctor Relation

Doctor (doctor_id | doctor_name | doctor_phno | doctor_address)

Let the Doctor relation's attributes equal the letters {ABCD}

R (A|B|C|D)

List of Functional Dependency:

A --> BCD

BCNF: This relation is in BCNF because there are no trivial functional dependencies.

3NF: We are obviously in 3NF because we are in BCNF and every non-prime attribute of the Person relation is non-transitively dependent on every key in the Person relation.

2NF: We are also obviously in 2NF because we are in 3NF and every nonprime attribute is dependent on a proper subset of every or any candidate key in this relation.

The Donor Relation

Donor (donor_id, donor_name, donor_gender, donor_address, donor_ph_no, iron_content, DOB, weight, blood_pressure, *doctor_doctor_id*)

Let the Donor's relation's attributes equal the letters {ABCDEFGHIJ}

R (A|B|C|D|E|F|G|H|I|J)

List of Functional Dependency:

A --> BCDEFGHIJ

BCNF: This relation is Not in BCNF because there are trivial functional dependencies. However, in order to maintain the desired dependencies this relation has we do not actually want to break the table up into two separate relations. (**This comes from talking to the professor and he told us not to break this table up**). Therefore, in order to maintain these dependencies, we will not normalize this relation to BCNF.

3NF: We are obviously in 3NF because we are in BCNF and every non-prime attribute of the Person relation is non-transitively dependent on every key in the Person relation.

2NF: We are also obviously in 2NF because we are in 3NF and every nonprime attribute is dependent on a proper subset of every or any candidate key in this relation.

The Blood Relation

blood (donor_donor_id, blood_type, blood_bank_blood_bank_id)

Let the Blood relation's attributes equal the letters {ABCDEFGHIJ}

R (A|B|C)

List of Functional Dependency:

A --> BC

BCNF: This relation is in BCNF because there are no trivial functional dependencies.

3NF: We are obviously in 3NF because we are in BCNF and every non-prime attribute of the Person relation is non-transitively dependent on every key in the Person relation.

2NF: We are also obviously in 2NF because we are in 3NF and every nonprime attribute is dependent on a proper subset of every or any candidate key in this relation.

The Bloodbank Relation

blood_bank (blood_bank_id, blood_bank_name, blood_bank_add)

Let the Bloodbank's relation's attributes equal the letters {ABC}

R (A|B|C)

List of Functional Dependency:

A --> BC

BCNF: This relation is in BCNF because there are no trivial functional dependencies.

3NF: We are obviously in 3NF because we are in BCNF and every non-prime attribute of the Person relation is non-transitively dependent on every key in the Person relation.

2NF: We are also obviously in 2NF because we are in 3NF and every nonprime attribute is dependent on a proper subset of every or any candidate key in this relation.

The Patient Relation

patient (patient_id, patient_name, patient_ph_no, patient_add, hosp_add)

Let the Patient's relation's attributes equal the letters {ABCDE}

R (A|B|C|D|E)

List of Functional Dependency:

A --> BCDE

BCNF: This relation is in BCNF because there are no trivial functional dependencies.

3NF: We are obviously in 3NF because we are in BCNF and every non-prime attribute of the Person relation is non-transitively dependent on every key in the Person relation.

2NF: We are also obviously in 2NF because we are in 3NF and every nonprime attribute is dependent on a proper subset of every or any candidate key in this relation.

The Examines Relation

doctor_examines_donor (doctor_doctor_id, donor_donor_id)

Let the examines relation's attributes equal the letters {AB}

R (A|B)

List of Functional Dependency:

A --> B

BCNF: This relation is in BCNF because there are no trivial functional dependencies.

3NF: We are obviously in 3NF because we are in BCNF and every non-prime attribute of the Person relation is non-transitively dependent on every key in the Person relation.

2NF: We are also obviously in 2NF because we are in 3NF and every nonprime attribute is dependent on a proper subset of every or any candidate key in this relation.

The Blood_Delivery Relation

blood_delivery (patient_patient_id, blood_bank_blood_bank_id)

Let the examines relation's attributes equal the letters {ABC}

R (A|B|C)

List of Functional Dependency:

A --> BC

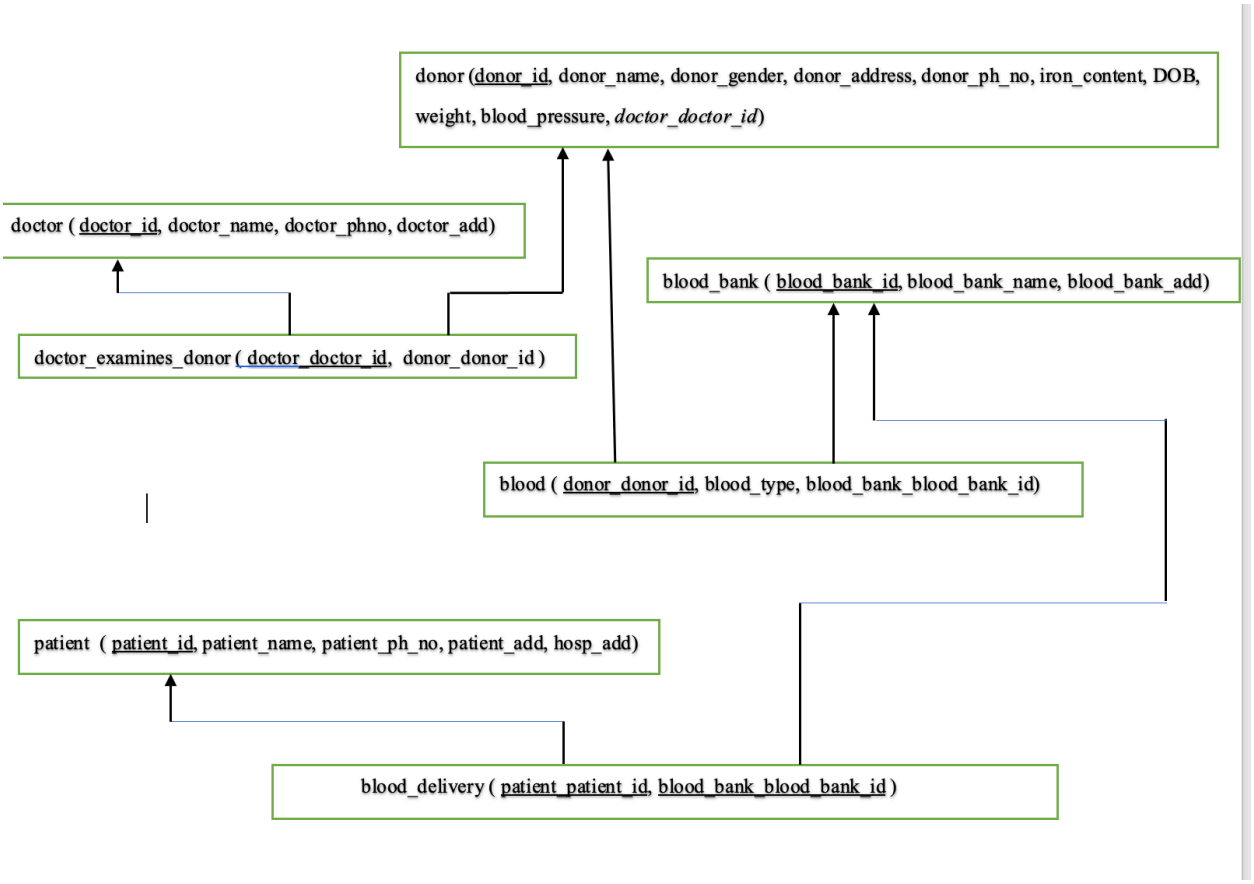
BCNF: This relation is in BCNF because there are no trivial functional dependencies.

3NF: We are obviously in 3NF because we are in BCNF and every non-prime attribute of the Person relation is non-transitively dependent on every key in the Person relation.

2NF: We are also obviously in 2NF because we are in 3NF and every nonprime attribute is dependent on a proper subset of every or any candidate key in this relation.

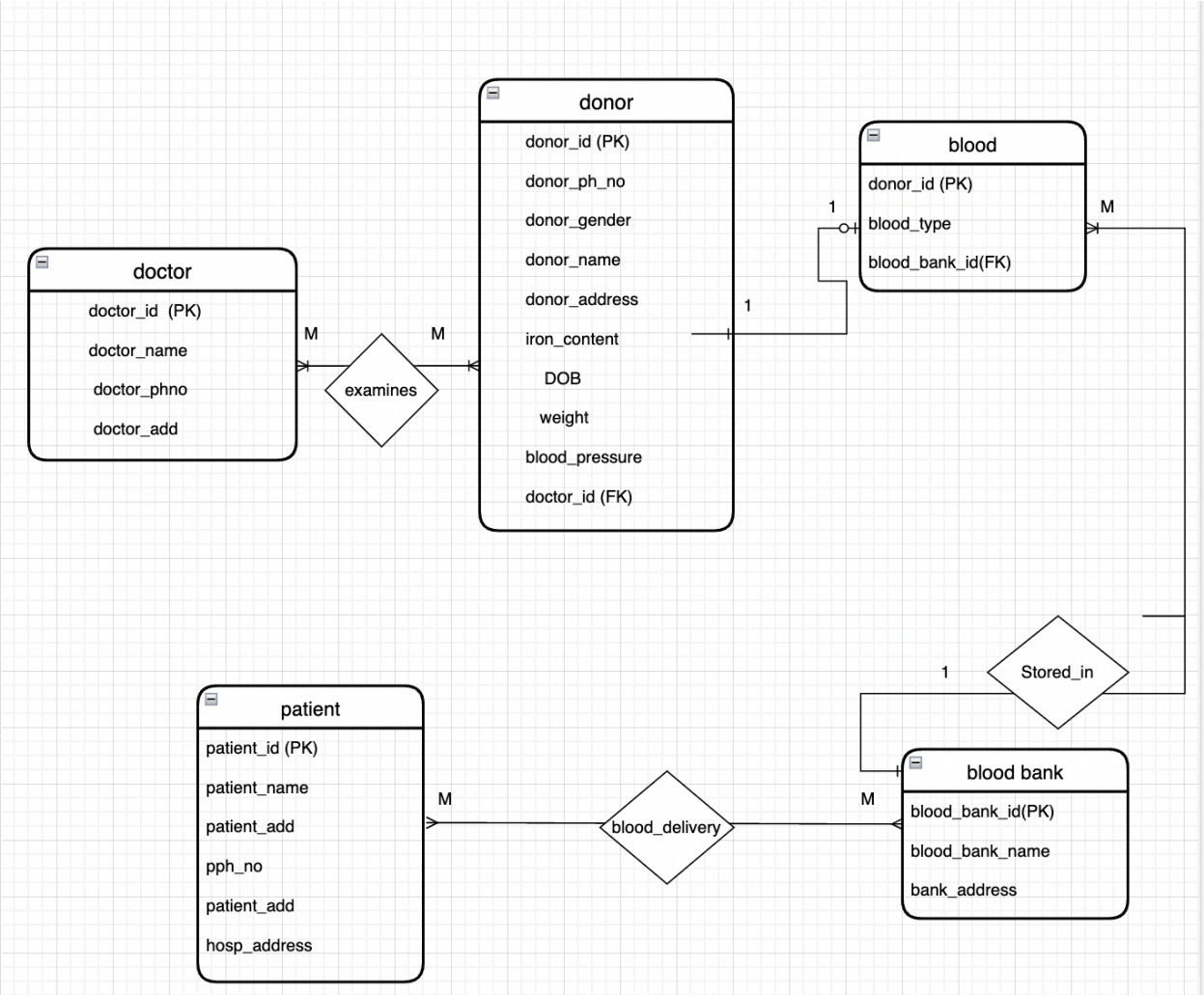
Normalized Relational model:

Here, normalized relational model is similar to the Relational model before normalization as shown in page 15

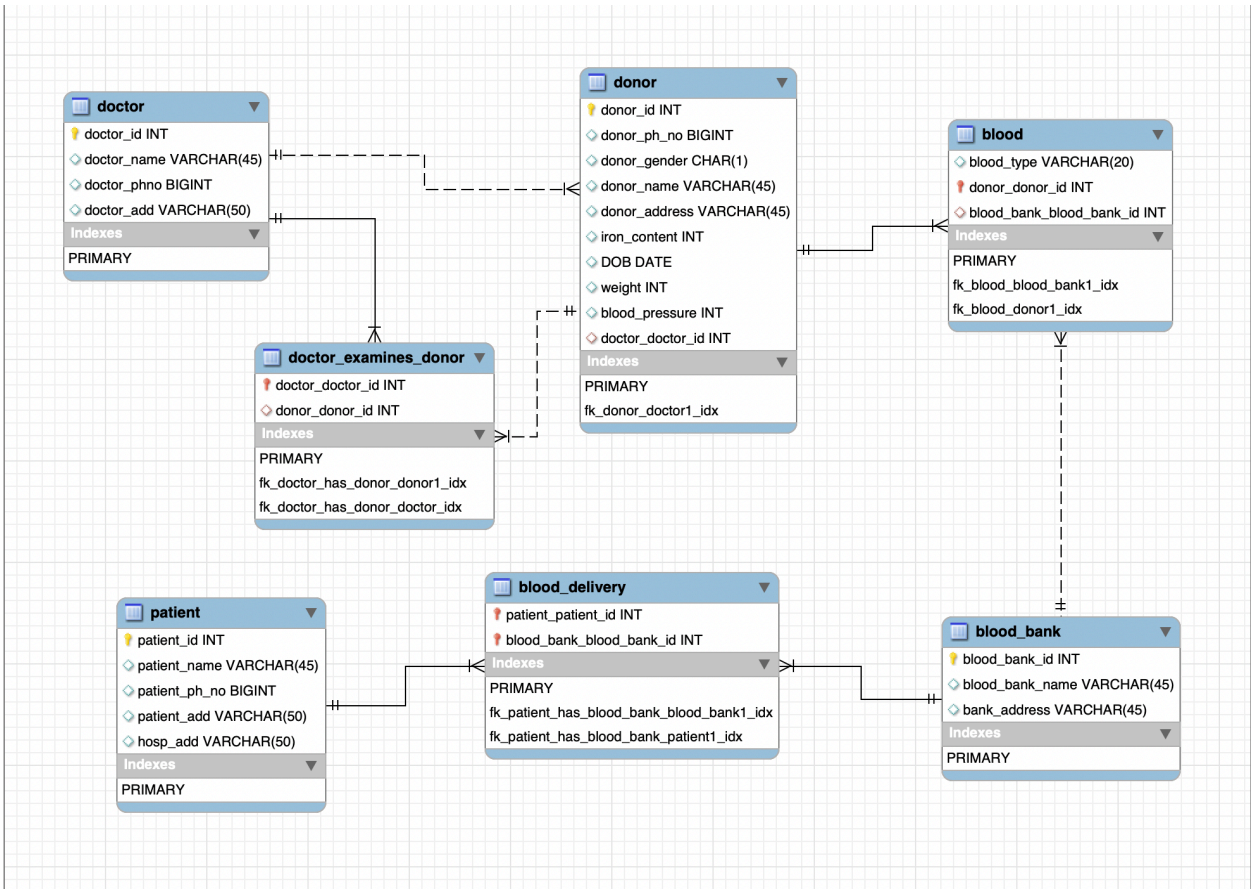


Logical and Relational Models (4)

Logical Model:



Relational Model



Physical Model

```
-- MySQL Workbench Forward Engineering
SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,
FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE
,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTIO
N';
-- -----------------------------------------------------
-- Schema bloodbank
-- -----------------------------------------------------
-- -----------------------------------------------------
-- Schema bloodbank
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `bloodbank` DEFAULT CHARACTER SET
utf8 ;
USE `bloodbank` ;
-- -----------------------------------------------------
-- Table `bloodbank`.`doctor`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `bloodbank`.`doctor` (
  `doctor_id` INT NOT NULL AUTO_INCREMENT,
  `doctor_name` VARCHAR(45) NULL,
  `doctor_phno` BIGINT NULL,
  `doctor_add` VARCHAR(50) NULL,
  PRIMARY KEY (`doctor_id`))
ENGINE = InnoDB;
-- -----------------------------------------------------
-- Table `bloodbank`.`blood_bank`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `bloodbank`.`blood_bank` (
  `blood_bank_id` INT NOT NULL,
  `blood_bank_name` VARCHAR(45) NULL,
  `bank_address` VARCHAR(45) NULL,
  PRIMARY KEY (`blood_bank_id`))
ENGINE = InnoDB;
-- -----------------------------------------------------
-- Table `bloodbank`.`blood`
-- -----------------------------------------------------
```

```
CREATE TABLE IF NOT EXISTS `bloodbank`.`blood` (
  `blood_type` VARCHAR(20) NOT NULL,
  `donor_id` INT NULL,
  `blood_bank_id` INT NULL,
  `blood_bank_blood_bank_id` INT NOT NULL,
  PRIMARY KEY (`blood_type`, `blood_bank_blood_bank_id`),
  INDEX `fk_blood_blood_bank1_idx` (`blood_bank_blood_bank_id`
ASC) VISIBLE,
  CONSTRAINT `fk_blood_blood_bank1`
    FOREIGN KEY (`blood_bank_blood_bank_id`)
    REFERENCES `bloodbank`.`blood_bank` (`blood_bank_id`)
    ON DELETE CASCADE
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
-- -----------------------------------------------------
-- Table `bloodbank`.`donor`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `bloodbank`.`donor` (
  `donor_id` INT NOT NULL,
  `donor_ph_no` BIGINT NULL,
  `donor_gender` CHAR(1) NULL,
  `donor_name` VARCHAR(45) NULL,
  `donor_address` VARCHAR(45) NULL,
  `iron_content` INT NULL,
  `DOB` DATE NULL,
  `weight` INT NULL,
  `blood_pressure` INT NULL,
  `doctor_id` INT NULL,
  `blood_blood_type` VARCHAR(20) NOT NULL,
  PRIMARY KEY (`donor_id`, `blood_blood_type`),
  INDEX `fk_donor_blood1_idx` (`blood_blood_type` ASC) VISIBLE,
  CONSTRAINT `fk_donor_blood1`
    FOREIGN KEY (`blood_blood_type`)
```

```
REFERENCES `bloodbank`.`blood` (`blood_type`)
    ON DELETE CASCADE
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
-- -----------------------------------------------------
-- Table `bloodbank`.`patient`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `bloodbank`.`patient` (
  `patient_id` INT NOT NULL,
  `patient_name` VARCHAR(45) NULL,
  `patient_ph_no` BIGINT NULL,
  `patient_add` VARCHAR(50) NULL,
  `hosp_add` VARCHAR(50) NULL,
  PRIMARY KEY (`patient_id`))
ENGINE = InnoDB;
-- -----------------------------------------------------
-- Table `bloodbank`.`doctor_examines_donor`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `bloodbank`.`doctor_examines_donor` (
  `doctor_doctor_id` INT NOT NULL,
  `donor_donor_id` INT NOT NULL,
  PRIMARY KEY (`doctor_doctor_id`, `donor_donor_id`),
  INDEX `fk_doctor_has_donor_donor1_idx` (`donor_donor_id` ASC)
VISIBLE,
  INDEX `fk_doctor_has_donor_doctor_idx` (`doctor_doctor_id`
ASC) VISIBLE,
  CONSTRAINT `fk_doctor_has_donor_doctor`
    FOREIGN KEY (`doctor_doctor_id`)
    REFERENCES `bloodbank`.`doctor` (`doctor_id`)
    ON DELETE CASCADE
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_doctor_has_donor_donor1`
    FOREIGN KEY (`donor_donor_id`)
    REFERENCES `bloodbank`.`donor` (`donor_id`)
    ON DELETE CASCADE
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
-- -----------------------------------------------------
-- Table `bloodbank`.`blood_delivery`
-- -----------------------------------------------------
```

```
CREATE TABLE IF NOT EXISTS `bloodbank`.`blood_delivery` (
  `patient_patient_id` INT NOT NULL,
  `blood_bank_blood_bank_id` INT NOT NULL,
  PRIMARY KEY (`patient_patient_id`,
`blood_bank_blood_bank_id`),
  INDEX `fk_patient_has_blood_bank_blood_bank1_idx`
(`blood_bank_blood_bank_id` ASC) VISIBLE,
  CONSTRAINT `fk_patient_has_blood_bank_patient1`
    FOREIGN KEY (`patient_patient_id`)
    REFERENCES `bloodbank`.`patient` (`patient_id`)
    ON DELETE CASCADE
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_patient_has_blood_bank_blood_bank1`
    FOREIGN KEY (`blood_bank_blood_bank_id`)
    REFERENCES `bloodbank`.`blood_bank` (`blood_bank_id`)
    ON DELETE CASCADE
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;

ALTER TABLE `bloodbank`.`donor`
DROP FOREIGN KEY `fk_donor_doctor1`;
ALTER TABLE `bloodbank`.`donor`
ADD CONSTRAINT `fk_donor_doctor1`
  FOREIGN KEY (`doctor_doctor_id`)
  REFERENCES `bloodbank`.`doctor` (`doctor_id`)
  ON DELETE NO ACTION
  ON UPDATE CASCADE;
```

Adding Data, SQL Queries and Final Report (5)

In order to facilitate ease of use our team decided to utilize only one Oracle account to create, make changes, and run queries against our database. After creating the database, we populated it with data by importing excel spreadsheets through SQL Developer. We created multiple queries including INSERT, UPDATE, and DELETE as well as other queries that a typical user would want to run in order to retrieve useful information in the database.

Data in Tables

This section contains screen captions of each relation and its data.

1.Doctor Table

doctor_id	doctor_name	doctor_ph...	doctor_add
1100	David Willy	874874776	Centerville
1101	Md. Abdullah	675896989	Miami
1102	Sky Wagner	675890823	Springfield
1103	Alex Bill	657381828	Wilmington
1104	Paola Oritz	657381828	Nordale
1105	Michelle Choe	657381828	Patterson
1106	Samantha Bing	758584283	Bellaire
1107	Ross Gellar	987253465	Fairborne
1108	Najjia Mahmoud	647767764	Beavercreek
1109	NULL	355356764	Wayne
1110	Virat Dev	878748728	Kettering
1111	Liam Livingston	646797764	Hamilton
1112	Sam Billings	748974764	Dixie
1113	Morgan stanley	984904847	Shroyer
1114	Ravi Shastri	288747766	Vandalia
1115	Bret Lee	747477743	Oakwood
1116	David Warner	467816478	Miamisburg
1117	Jessie Pinkman	NULL	Springfield
1118	Carlo Pavesi	895925967	Centerville
1119	Nedo Nadi	275925825	Bellaire
1120	Bradley Wiggins	566868638	Wayne
1121	Sun Yang	344649498	Nordale
1122	Bud Houser	789692493	Dixie
1123	Nafiya Naz	264641684	Nordale
NULL	NULL	NULL	NULL

2.Donor Table

donor_id	donor_ph_no	donor_gender	donor_name	donor_address	iron_conte...	DOB	weight	blood_pressu...	doctor_doctor...
8091	557257278	F	Nanmi Lee	Wayne	45	1994-09-27	135	120	1100
8092	747474376	M	John Kepler	Patterson	47	1997-02-23	129	115	1101
8093	437347367	M	Bill Joey	Dixie	41	1196-10-29	122	134	1102
8094	983493879	F	Rachel Greene	Nordale	47	1987-12-13	143	117	1115
8095	894898494	F	Phoebe Buffay	Wilmington	46	1998-07-25	129	123	1104
8096	894959358	M	Chandler Bing	Fairborn	39	1998-10-06	119	114	1105
8097	243542534	M	Aron Paul	Miamisburg	41	1992-05-30	128	133	1106
8098	879879869	F	Monica Gellar	Clifton	43	1991-11-23	145	113	1107
8099	494878744	M	NULL	Centerville	45	1997-12-31	127	111	1108
8100	970697006	F	Anna Delvy	Irving	44	1993-05-18	118	121	1109
8101	949384348	M	Simon Livieve	Thorpe	42	1994-01-19	141	119	1110
8102	656552535	M	Walter White	Stewart	46	1196-10-29	130	119	1111
8103	998987989	M	Darth Vader	Utah	40	1999-08-19	136	117	1112
8104	542542235	M	Chris Evans	Volkenand	43	1986-09-15	139	112	1113
8105	778678576	M	Tyler Durden	Medford	41	1989-10-19	132	120	1114
8106	767747657	M	Sherlock Hol…	Constantia	47	1196-10-29	119	120	1115
8107	454254544	M	Antoine Doinel	Alberta	39	1991-11-23	142	119	1116
8108	847585838	M	John Wick	Nordale	43	1988-12-12	136	111	1109
8109	NULL	F	Scarlette Joh…	Pike	47	1993-11-11	124	119	1111
8110	647264727	F	Jessica Holmes	Columbus	41	1990-08-26	164	116	1104
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

3.Blood Table

blood_ty...	donor_donor_id	blood_bank_blood_bank...
A+	8091	2308
A-	8092	2314
B+	8093	2309
B-	8094	2315
AB+	8095	2312
AB-	8096	2317
O+	8097	2312
O-	8098	2320
A+	8099	2302
A-	8100	2310
B+	8101	2314
AB+	8102	2304
AB-	8103	2305
B+	8104	2306
A+	8105	2307
A-	8106	2313
O+	8107	2314
O-	8108	2315
B-	8109	2312
AB-	8110	2301
NULL	NULL	NULL

4.Bloodbank Table

blood_bank_id	blood_bank_name	bank_address
2301	Houchin	Beavercreek
2302	Cleveland cord	Wilmington Ave
2303	Talecris	Salem Ave
2304	Versiti	Bellaire Ave
2305	BioLife	Wayne Ave
2306	Huntington	Hamilton Ave
2307	Houchin	Dixmyth Ave
2308	Octopharma	Monteith Ave
2309	American Red Cross	Clifton Ave
2310	Community Tissue	Nordale Ave
2311	Prebel	Linden Ave
2312	Hoxworth	Oakdale Ave
2313	CompuNet	Monument Ave
2314	LifeSouth	Watervliet Ave
2315	Card Blood	Poe Ave
2316	BioLife	Wayne Ave
2317	Talecris	Linden Ave
2318	Octopharma	Monteith Ave
2319	Versiti	Beavercreek
2320	Prebel	Dixmyth Ave
NULL	NULL	NULL

5.Patient Table

patient_id	patient_name	patient_ph_...	patient_add	hosp_add
142510	Michael Phelps	673734889	Wyoming	Spencer
142511	Larisa Latynina	747247248	Tennyson	Harisson
142512	Paavo Nurmi	737477828	Genessee	Buxton
142513	Carl Lewis	748429892	Arlene	Kemper
142514	Birgit Fischer	265624764	Denlinger	Herbert
142515	Ray Ewry	978968796	Ferguson	Keowee
142516	Nikolai Andrianov	909302389	Salem	Helena
142517	Bonnie Blair	392323237	Wilkinson	Stanley
142518	Wu Minxia	782782782	Wawona	Berwyck
142519	Dara Torres	882482499	Philadelphia	Kehler
142520	Ma Long	482828474	Earlham	Timber
142521	Dawn Fraser	478247827	Cornell	Gipsy
142522	Sven Fischer	742929299	Wesleyan	Miller
142523	Olga Korbut	783788484	Prescott	Webster
142524	Libby Trickett	788278478	Brumbaugh	Jefferson
142525	Murray Rose	839222848	Thorain	Hillcrest
142526	Janet Evans	844284828	Kipling	Marson
142527	Valentin Muratov	274274827	Natalie	Allerton
142528	Kathrin Boron	848274878	Free Pike	Hudson
142529	Lisa Leslie	784786748	Hosbrook	Belmonte
NULL	NULL	NULL	NULL	NULL

6.Examines Table

doctor_doctor...	donor_donor_id
1119	8091
1118	8092
1117	8093
1116	8094
1115	8095
1114	8096
1113	8097
1112	8098
1111	8099
1110	8100
1122	8100
1109	8101
1108	8102
1107	8103
1106	8104
1105	8105
1104	8106
1103	8107
1102	8108
1120	8108
1101	8109
1100	8110
1121	8110
NULL	NULL

7.BloodDelivery Table

patient_patient_id	blood_bank_blood_bank_id
142510	2301
142529	2302
142528	2303
142527	2304
142526	2305
142525	2306
142524	2307
142523	2308
142522	2309
142521	2310
142520	2311
142519	2312
142518	2313
142517	2314
142516	2315
142515	2316
142514	2317
142513	2318
142512	2319
142511	2320
NULL	NULL

SQL Select Queries

This section contains all the queries that we ran on our database.

1. List of all female donor's id along with the blood type donated and the address of the blood bank stored in with iron content more than 45

SQL QUERY:

```
1
2 •  select b.donor_donor_id,
3           b.blood_type,
4           b.blood_bank_blood_bank_id
5    from  blood b
6    inner join blood_bank k on k.blood_bank_id = b.blood_bank_blood_bank_id
7    inner join donor d on d.donor_id = b.donor_donor_id AND d.donor_gender = 'F' WHERE d.iron_content > 40;
8
```

RESULTS:

donor_donor_id	blood_type	blood_bank_blood_bank...
8091	A+	2308
8094	B-	2318
8095	AB+	2312
8098	O-	2320
8100	A-	2310
8109	B-	2312
8110	AB-	2301

2.List of all the blood samples and the number of samples of each blood type

SQL QUERY:

```
1
2 •   select blood_type AS Samples, count(blood_type)
3     from  blood group by blood_type;
4
```

RESULTS:

Samples	count(blood_type)
A+	3
A-	3
B+	3
B-	2
AB+	2
AB-	3
O+	2
O-	2

3.List of all the patient's id and the patient's name and the blood bank name present in the same address location of the hospital that the patient is residing.

SQL QUERY:

```
1 •   SELECT patient_id,
2            patient_name,
3            blood_bank_name
4     FROM bloodbank.patient,
5          bloodbank.blood_bank
6     WHERE bank_address = hosp_add;
7
```

RESULTS:

patient_id	patient_name	blood_bank_na...
142524	Libby Trickett	Houchin
142528	Kathrin Boron	Octopharma
142517	Bonnie Blair	Hoxworth
142512	Paavo Nurmi	Card Blood
142528	Kathrin Boron	Octopharma
142524	Libby Trickett	Versiti

4.List of all the donor's names and the doctor's name that tested their eligibility to donate blood.

SQL QUERY:

```
SELECT  donor_name, doctor_name
FROM donor, doctor
WHERE doctor.doctor_id = donor.doctor_doctor_id
        AND donor_name IS NOT NULL
        AND doctor_name IS NOT NULL
ORDER BY donor_name ASC;

```

RESULTS:

donor_name	doctor_name
Antoine Doinel	David Warner
Aron Paul	Samantha Bing
Bill Joey	Sky Wagner
Chandler Bing	Michelle Choe
Chris Evans	Morgan stanley
Darth Vader	Sam Billings
Jessica Holmes	Paola Oritz
John Kepler	Md. Abdullah
Monica Gellar	Ross Gellar
Nanmi Lee	David Willy
Phoebe Buffay	Paola Oritz
Rachel Greene	Bret Lee
Scarlette John…	Liam Livingston
Sherlock Holm…	Bret Lee
Simon Livieve	Virat Dev
Tyler Durden	Ravi Shastri
Walter White	Liam Livingston

SQL Insert Queries

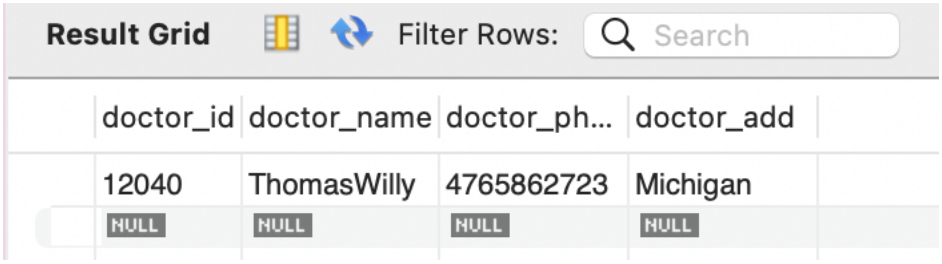
INSERT SQL QUERY:

```
1  INSERT INTO `doctor`
2  VALUES (12040,'ThomasWilly',4765862723,'Michigan');
```

Action Output

	Time	Action	Response	Duration / Fetch Time
1	14:13:16	INSERT INTO `doctor` VALUES (12040,'ThomasWilly',4765862723,'Michigan')	1 row(s) affected	0.0084 sec

RESULTS:



SQL Update Queries

UPDATE SQL QUERY:

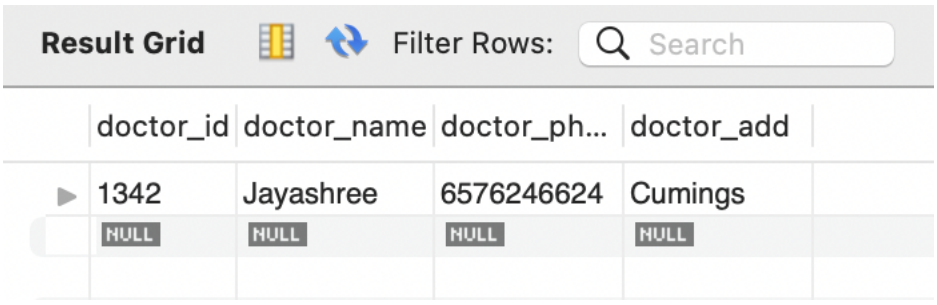
```
1 •  UPDATE doctor
2    SET doctor_id = 1342,
3          doctor_name = 'Jayashree',
4          doctor_phno = 6576246624,
5          doctor_add = 'Cumings'
6    WHERE doctor_id = 1110;
7
```

100% 1:7

Action Output

	Time	Action	Response	Duration / Fetch Time
1	14:20:45	UPDATE doctor SET doctor_id = 1342, doctor_name = 'Jayashree',...	0 row(s) affected Row...	0.00072 sec

RESULTS:



SQL Delete Queries

Delete SQL QUERY:

```
DELETE
FROM donor
WHERE donor.doctor_doctor_id = (SELECT doctor_id
                                FROM doctor
                                WHERE doctor.doctor_add
                                IN ('Fairborne' 'Springfield' 'Kettering' 'Miami'
                                'Hamilton' 'Wayne' 'Dixie' 'Bellaire'));
```

100% 33:8

Action Output

	Time	Action	Response	Duration / Fetch Time
1	14:33:07	DELETE FROM donor WHERE donor.doctor_doctor_id = (SELECT docto...	0 row(s) affected	0.00082 sec

RESULTS:

```
SELECT *
FROM bloodbank.donor
WHERE donor.doctor_doctor_id = (SELECT doctor.doctor_id
                                FROM bloodbank.doctor
                                WHERE doctor.doctor_add
                                IN('Fairborne' 'Springfield' 'Kettering' 'Miami' 'Hamilton' 'Wayne' 'Dixie' 'Bellaire' ));
```

100% 21:2

Result Grid Filter Rows: Search Edit: Export/Import:

donor_id	donor_ph_no	donor_gender	donor_name	donor_address	iron_conte...	DOB	weight	blood_pressu...	doctor_doctor...
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Conclusion :

After completing the project, we learned many things throughout the different parts of the project. Setting up a proper ER diagram is essential to having a functional database. We also learned the importance of creating meaningful relations so that data can be retrieved from the database in a useful manner. When learning about normalization it was important to remember there are times when normalizing to BCNF simply does not make sense for your database. We also learned how to use many of the data developer tools offered by Oracle. Including, being able to create a database with tables and relationships and to be able to successfully export data.

One of the first problems we ran into when designing the database was with the way it was initially set up. The database contained many to many relationships that had to be taken care of. Another problem that we encountered was with normalization. At first, we normalized to BCNF but found that this would later cause issues as it separated data that would always be retrieved together. This would have required constant joins that would not benefit the database design. Another issue we had was with being able to export data, which we had to do some troubleshooting, to not receive any errors. Another issue we ran into was trying to import data due to some constraint violations.

Overall, I think both of us learned quite a bit from this class and feel more confident not only creating databases but managing them. In the future, this database can now be used by Blood Warriors Dayton to properly store and manage all its data. This will not only make finding the donors or fulfilling the requirement easier but allow them to add reporting where stored procedures can be created which will allow users to pick parameters based on their needs to receive important information with little or no database experience.