MEDICAL CLINIC INFORMATION SYSTEM

Database Management System

FINAL PROJECT

BY:
Naga Sreeja Kurra
500920707

TABLE OF CONTENTS

INTRODUCTION

This Medical Database System is responsible for all the relevant information within the structure of a clinic. It stores the data related to the patients and their personal health information. This system is able to function common database functions like insert, create, delete and view. The primary employees of this system are Doctors, nurses and the receptionist. The general goal of this system is to provide basic facilities for the patients and to treat them according to their health conditions.

BASIC FUNCTIONS

Some of the general potential functions of this DBMS are outlined below:

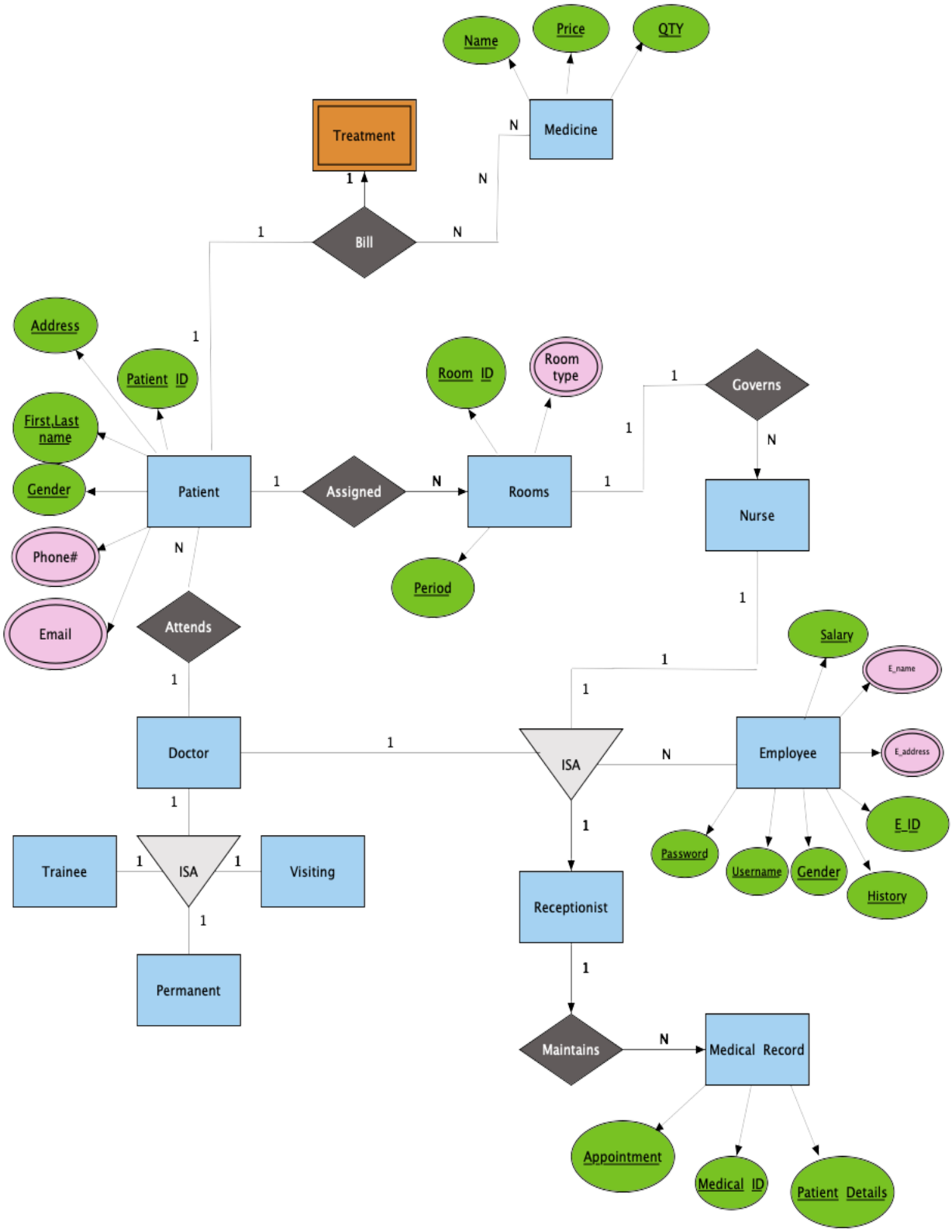
FUNCTIONS	DESCRIPTION
Patient	Add the personal details of the patient when they are new to the hospital.
Patient_check	Add the basic details of the patient on the day of the appointment.
Medical_Clinic	Updates the name, location, number of clinics of the same patient. (it also shows the information of the patients who visited the other clinics apart from ours).
Doctor	Gives the personal details of the doctor and also shows the details of the patient treated.
Receptionist	Gives only the details of the receptionist, and also helps to enter the details of the patient.
Medical_record	This function updates the appointments of the Patient and also gives the medical history of the Patient.
Medicine	This function gives the information about the doctor's prescribed medicines to the patient along with the price and quantity.
Nurse	This function gives the basic details of the nurse and the related patient she took care of.
Governs	This function gives the basic details of the room being assigned to the patient (if required)
Employee	This function is the core for the database system, it gives the overall information about the type of employees working in our clinic. Search all the relevant information by a primary key Emp_ID.

In the above functions, the primary keys are being assigned which gives the connection between each table.

The following list of entities in this DBMS and their relationships are given below:

ENTITIES - ATTRIBUTES	RELATIONSHIPS
1. PATIENT - Patient_ID - Name - Gender - Address - Email - Phone# 2. TREATMENT (WEAK) 3. MEDICINE - Medicine_ID - Name - Price - Quantity 4. ROOMS - Room_ID - Room_number - Room_type - Period 5. DOCTOR 6. NURSE 7. RECEPTIONIST 8. EMPLOYEE (includes 5,6,7) - E_name - Emp_ID - Username - Password - Gender - History - Salary - E_address (this includes the general details of the Doctor, nurse and receptionist, based on the Emp_Id, it will decide which one is logging in. 9. MEDICAL RECORD - Medical_ID - Appointment - Patient_details 10. TRAINEE (INTERN) 11. PERMANENT Entities 10 and 11, are based on the work terms, but give the same information.	- Patient(1) **manages** Medicine(N). - Doctor(1) **manages** Patient(N). - Patient(1) **books** Rooms(N). Can be given one or more rooms. - Nurse(1) **manages** Rooms(N). - Nurse(1) **manages** Patients(N). - Patient(1) **maintains** Medical_Record(1)

ER DIAGRAM


Name
Price
QTY
Medicine
Treatment
Bill
Address
Patient ID
First,Last name
Gender
Phone#
Email
Patient
Assigned
Room ID
Room type
Rooms
Period
Governs
Nurse
Attends
Doctor
ISA
Employee
Salary
E_name
E_address
E_ID
Password
Username
Gender
History
Trainee
Visiting
Permanent
Receptionist
Maintains
Medical Record
Appointment
Medical ID
Patient Details

FUNCTIONAL DEPENDENCIES

Patient Table

From the Table Patient, the attributes are:

Employee_ID
Employee_name
Gender
Address
Contact_No
Email_id

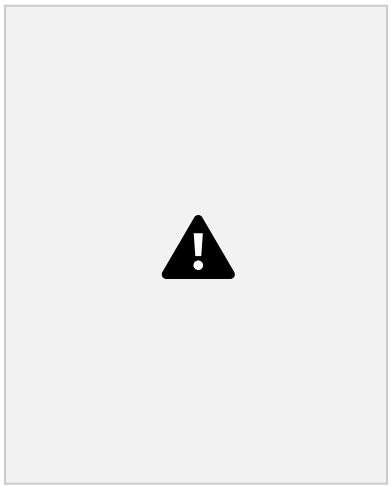
Since Employee_ID is a primary key and the functions which are dependent on the Employee_ID are Employee_name, Gender, Address, Contact_No, Email_id. These are the attributes which are the dependent ones to the Employee_ID.

So,

Employee_ID → Employee_name,Gender, Address, Contact_No, Email_id

From the above diagram, many patients will attend a doctor. So the Functional Dependencies for the above diagram will be based on

#Patient_ID → #Doctor_ID



Patient_Check Table

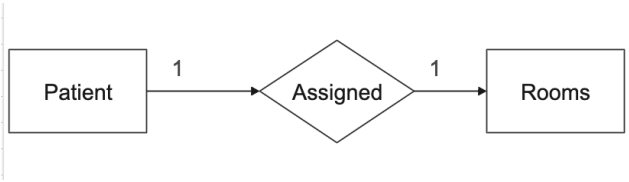
From the Table Patient_check the attributes are:

App_Id PRIMARY KEY
N_Id
P_Id
D_Id
P_HEIGHT
P_WEIGHT
P_BP

From the above table, these attributes are the common details for the Patient which are supposed to update the on-date values.
Here App_Id is the Primary key for the above table and N_Id, P_Id, D_Id, P_HEIGHT, P_WEIGHT and P_BP are dependent values for the primary key.
So,
App_Id → N_Id, P_Id, D_Id, P_HEIGHT, P_WEIGHT and P_BP.

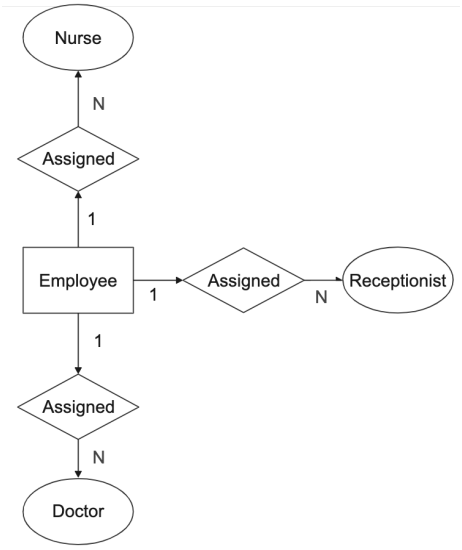
Rooms Table

From the Table Rooms the attributes are:
Room_ID PRIMARY KEY
Room Type
Period
From the above table, each room is assigned to a patient which includes Room_ID as a primary key with a room type and no. of days as period.
So, since it is a 1:1 relationship the functional dependency for the above diagram will be
#Patient_id → #Room_Id



Employee Table

From the table Employee the attributes are:
E_Id PRIMARY KEY
E_Firstname
E_Lastname
Gender
Username
Pass_word
Address
From the above diagram we can see that the Employee has a primary key name Employee_ID and it is being used based on three employees: Doctor, Receptionist and Nurse.
Each table Nurse, Receptionist and Doctor has been assigned an ID to each one of them and it is based on the employee_ID.
So,
Employee_ID → Doctor_ID, Nurse_ID, Receptionist_ID
Each employee has one to many relationships, meaning an employee can be one or more doctors, nurses and receptionists.
#Employee_ID → #Doctor_ID, #Nurse_ID, #Receptionist_ID



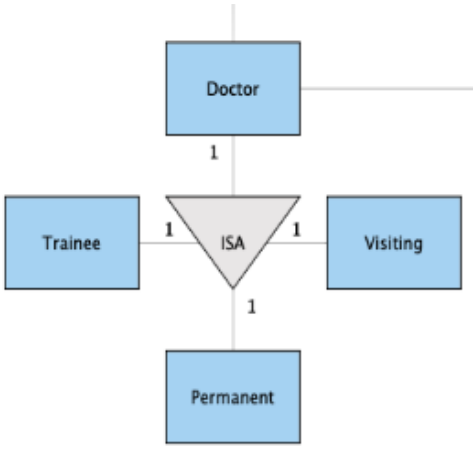
Doctor Table

From the Table Doctor the attributes are:

E_ID

The doctor table branches into 3 seperate tables called trainee, visiting and permanent, all of which contain only the employee_id attribute. This splits all doctors into specific types.

There are no functional dependencies because the table consists of a list of employee Id's signifying which employees are doctors.



Medical_Clinic Table

From the Table Medical_Clinic the attributes are:

Clinic_Number PRIMARY KEY

Clinic_Name

Clinic_Location

Clinic_Phone_Number

Here, Clinic_Number is the PRIMARY KEY which is used to find all associated clinic information: clinic name, location and phone number

So,

Clinic_Number → Clinic_Name, Clinic_Location,Clinic_Phone_Number

Receptionist Table

From the Table Receptionist the attributes are:

E_ID

No Functional Dependencies because the table consists of only E_ID. Employee_ID is used to show which employee id's are receptionists

Medical_Record Table

From the Table Medical_Record the attributes are:

Medical_Record_ID PRIMARY KEY

Appointment

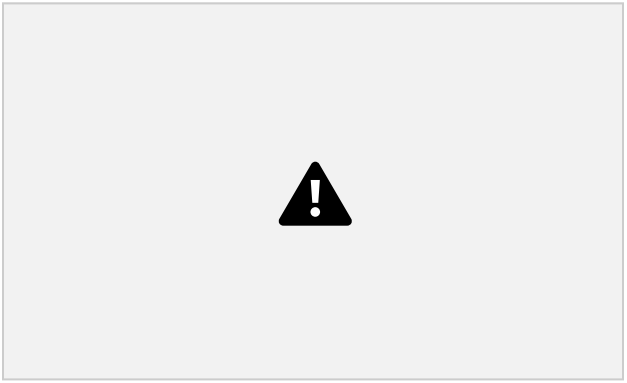
Medical_ID

Patient_Details

Here, Medical_Record_ID is the PRIMARY KEY which is used to find all associated clinic information: appointment, medical_id, patient_details

So,

Medical_Record_ID → Appointment, Medical_ID,Patient_Details



Nurse Table

From the Table Nurse the attributes are:

N_ID

Nurse_Details

The PRIMARY KEY N_ID is used to find the associated nurse information: Nurse_Details
N_Id → Nurse_Details

Governs Table

From the Table governs the attributes are:
E_ID
Room_Id
The primary key E_Id is used to associate an employee to a room (element of Room table) in a many to one relationship.
E_Id → Room_Id

Medicine Table

From the Table Medicine the attributes are:
Medicine_ID
Doctor_ID
Doctor_name
Medicine_name
Medicine_price
Quantity
The medicine forms a many to one relationship with the corresponding Doctor_ID, allowing a single doctor to prescribe multiple medicines.

The PRIMARY KEY Medicine_ID is used to find the associated information about the medicine: Doctor_ID, Medicine_name, Medicine_price, Quantity

Medicine_ID → Doctor_ID, Medicine_name, Medicine_price, Quantity

NORMALISATIONS

Patient Table

1NF

Patient

patient_ID	patient_name	Gender	Address	Contact_no	Email_ID
1001	sarah	Female	34 Twin Pines Drive	6478907867	sarah@gmail.com
1002	sree	Female	74 Crescent Avenue	9053458976	sree23@hotmail.com
1003	mark	Male	145 allen drive	4167685786	23mark@yahoo.com

2NF

Patient_Details

patient_name	Gender
sarah	Female
sree	Female
mark	Male

Patient_ContactDetails

patient_ID	Address	Contact_no	Email_ID
1001	34 Twin Pines Drive	6478907867	sarah@gmail.com
1002	74 Crescent Avenue	9053458976	sree23@hotmail.com
1003	145 allen drive	4167685786	23mark@yahoo.com

From the above tables, we can conclude that the Gender is dependent on one of the composite keys. **patient_ID** and **patient_name** are the two primary keys making the composite key. Address, contact_no and the Email address are dependent on the patient_ID and patient_name. It can be either with the patient_ID and patient_name which is associated with the other details. From the table Patient, now there are no partial dependencies from the above two tables.

3NF

Patient_contact

patient_ID	patient_name	Gender	Contact_no	Address	Email_ID
1001	sarah	Female	6478907867	34 Twin Pines Drive, Brampton	sarah@gmail.com
1002	sree	Female	9053458976	74 Crescent Avenue, Toronto	sree23@hotmail.com
1003	mark	Male	4167685786	145 allen drive, Mississauga	23mark@yahoo.com

BCNF

Patient_contact

patient_ID	patient_name
1001	sarah
1002	sree
1003	mark

patient_name	Gender	Contact_no	Address	Email_ID
sarah	Female	6478907867	34 Twin Pines Drive, Brampton	sarah@gmail.com
sree	Female	9053458976	74 Crescent Avenue, Toronto	sree23@hotmail.com
mark	Male	4167685786	145 allen drive, Mississauga	23mark@yahoo.com

From the above 3NF, **Patient_ID** and **Patient_name** are the candidate keys. **Gender, Contact_no, Email_ID** and **Address** are not the super keys and therefore they are the non-prime attributes.

{Patient_ID, Patient_name} → {Gender, Contact_no, Email_ID, Address}

Therefore, {Patient_ID, Patient_name}are the prime attributes.

From the first table, **Patient_ID** is the super key. From the second table, **Patient_name** is the super key.

Employee Table

1NF

Employee

Emp_ID	Emp_name (F & L)	Gender	Emp_type	Username	Password	Address
D101	Mark Stew	Male	Doctor	Mark	m@56	Brampton
N101	Siri D'souza	Female	Nurse	siri35	siri45j	Toronto
R101	Rahul Dravid	Male	Receptionist	Rahul	rahul3	York

2NF

Employee_Details

Emp_ID	Emp_type
D101	Doctor
N101	Nurse
R101	Receptionist

Employee_details2

Emp_name (F & L)	Username	Password
Mark Stew	Mark	m@56
Siri D'souza	siri35	siri45j
Rahul Dravid	Rahul	rahul3

Employee_details3

Emp_name (F & L)	Gender	Emp_ID	Address
Mark Stew	Male	D101	Brampton
Siri D'souza	Female	N101	Toronto
Rahul Dravid	Male	R101	York

From the above tables, we can say that there are no partial dependencies and all the tables are in 2NF. Here **Emp_ID** is the primary key for the first table and for the table Employee_details3, while **Emp_name** is the primary key.

3NF

Employee_Logindetails

Emp_ID	Emp_type	Username	Password
D101	Doctor	Mark	m@56
N101	Nurse	siri35	siri45j
R101	Receptionist	Rahul	rahul3

Employee_Personaldetails

Emp_ID	Emp_name (F & L)	Gender	Address
D101	Mark Stew	Male	Brampton
N101	Siri D'souza	Female	Toronto
R101	Rahul Dravid	Male	York

BCNF

From the table **Employee_Logindetails,**

Emp_ID	Emp_type
D101	Doctor
N101	Nurse
R101	Receptionist

Emp_ID	Username	Password
D101	Mark	m@56
N101	siri35	siri45j
R101	Rahul	rahul3

From the above 3NF, **Emp_ID** and **Emp_type** are the candidate keys. **Username** and **Password** are not the super keys and therefore they are the non-prime attributes.
{Emp_ID,Emp_type} → {Username, Password}
Therefore, {Emp_ID,Emp_type} are the prime attributes.
From the first table and the second table, **Emp_ID** is the super key. Here Emp_type cannot be the super key for the second table because it is the employee type and type of employee cannot be decided for the Username and the Password.

From the table **Employee_Personaldetails,**

Emp_ID	Emp_name (F & L)
D101	Mark Stew
N101	Siri D'souza
R101	Rahul Dravid

Emp_name (F & L)	Gender	Address
Mark Stew	Male	Brampton
Siri D'souza	Female	Toronto
Rahul Dravid	Male	York

From the above 3NF, **Emp_ID** and **Emp_name** are the candidate keys. **Gender** and **Address** are not the super keys and therefore they are the non-prime attributes.
{Emp_ID,Emp_name} → {Gender,Address}
Therefore, {Emp_ID,Emp_name} are the prime attributes.
From the first table, **Emp_ID** is the super key. From the second table, **Emp_name** is the super key. In conclusion, the above tables will be based on Emp_Id.

Patient_Check Table

1NF

Patient_check

App_ID	Emp_ID	P_Height	P_Weight	P_BP
1001	N101	178cm	59kgs	120/80
1002	N102	164cm	70kgs	131/78
1003	N101	185cm	69kgs	120/78

2NF

Patient_check1

App_ID	Emp_ID
1001	N101
1002	N102

1003	N101

Patient_check2

App_ID	P_Height	P_Weight	P_BP
1001	178cm	59kgs	120/80
1002	164cm	70kgs	131/78
1003	185cm	69kgs	120/78

From the above tables, there are no partial dependencies. Here, App_ID is the primary key for the table Patient_check1. From the table Patient_check 1, the Emp_ID is partially dependent on the App_ID because the details for the Patient are based on the table Patient_check2.

3NF

Patient_check

App_ID	P_Height	P_Weight	P_BP
1001	178cm	59kgs	120/80
1002	164cm	70kgs	131/78
1003	185cm	69kgs	120/78

In this table there are no functional dependencies.

App_ID	Patient_name	P_Height	P_Weight	P_BP
1001	Sarah	178cm	59kgs	120/80
1002	sree	164cm	70kgs	131/78
1003	mark	185cm	69kgs	120/78

From the above table there are functional dependencies, we can further get classified to boyce codd normal form.

BCNF

App_ID	Patient_name
1001	Sarah
1002	sree

1003	mark

Patient_name	P_Height	P_Weight	P_BP
Sarah	178cm	59kgs	120/80
sree	164cm	70kgs	131/78
mark	185cm	69kgs	120/78

From the above 3NF, **App_ID** and **Patient_name** are the candidate keys. **P_Height, P_Weight** and **P_BP** are not the super keys and therefore they are the non-prime attributes.
{Appp_ID,Patient_name} → {P_Height, P_Weight, P_BP}
Therefore, {App_ID,Patient_name} are the prime attributes.
From the first table, **App_ID** is the super key. From the second table, **Patient_name** is the super key.

Medical_Record Table

2NF

Medical_Record_ID	Appointment	Patient_Id	Patient_Details
1001	January 1 2021	2	Fever
1002	January 2 2021	56	Null
1003	January 5 2021	7	Cough

Nurse table is 2nf because all fields are dependent on Medical_Record_Id (Primary Key) and it's uniquely identifiable

3NF

Medical_Record_ID	Appointment	Patient_Id	Patient_Details
1001	January 1 2021	2	Fever
1002	January 2 2021	56	Null
1003	January 5 2021	7	Cough

BCNF

Medical_Record_ID	Appointment	Patient_Id	Patient_Details
1001	January 1 2021	2	Fever
1002	January 2 2021	56	Null
1003	January 5 2021	7	Cough

No changes made from 3NF to BCNF in the Medical_Record Table.
Medical_Record_ID is the prime attribute
{Medical_Record_ID} → {Appointment, Patient_ID, Patient_Details}

The reason why there are no changes between 3NF and BCNF is because **there are no overlapping candidate keys**.

Nurse Table

2NF

N_ID	E_Firstname	E_Lastname	Nurse_Details
1001	Bob	Joe	Null
1002	Sarah	Bob	Null
1003	Matt	Lee	Null

Nurse table is 2nf because all fields are dependent on N_Id (Primary Key)

3NF

N_ID	E_Name	Nurse_Details
1001	Bob Joe	Null
1002	Sarah Bob	Null
1003	Matt Lee	Null

BCNF

N_ID	E_Name
1001	Bob Joe

1002	Sarah Bob
1003	Matt Lee

N_ID	Nurse_Details
1001	Null
1002	Null
1003	Null

From the above 3NF, **N_ID** and **N_Name** are the candidate keys. **Nurse_Details** is not a prime attribute.
{N_ID, N_Name} → {Nurse_Details}
Therefore, {N_ID, N_Name} are the prime attributes.
The super key is N_ID.

Medicine Table

1NF

Doctor_ID	Doctor_Name	Medicine_ID	Medicine_Name	Medicine_PrICE	Quantity
180	Mark Stew	1001	Medicine1	100	22315
197	Matt Chin	1002	Medicine2	140	231
3	Reggie Jackson	1003	Medicine3	200	5

2NF

Medicine

Medicine_ID	Medicine_Name	Medicine_PrICE	Quantity
1001	Medicine1	100	22315
1002	Medicine2	140	231
1003	Medicine3	200	5

Medicine_Assigned

Doctor_ ID	Doctor_Name	Medicine_ID	Assigned_Patient_I D
180	Mark Stew	1001	35
197	Matt Chin	1002	36
3	Reggie Jackson	1003	657

Tables above remove partial dependencies by splitting up the original table into two. 1 table is used to track all medicines available. The other table is used to track which medications are prescribed to what patient. Primary Keys are medicine_id and doctor_id for respective tables.

3NF

Doctor_to_Medicine

Doctor_ID	Doctor_Name	Medicine_ID
180	Mark Stew	1001
197	Matt Chin	1002
3	Reggie Jackson	1003

Medicine_to_Patient

Medicine_ID	Assigned_Patient_ID
1001	35
1002	36
1003	657

The above tables were split from one to two tables when converting from 2NF to 3NF. The purpose of these two tables is to keep track of assigning medicine to patients and the doctors involved.

BCNF

Doctor_ID	Doctor_Name
180	Mark Stew
197	Matt Chin
3	Reggie Jackson

Doctor_ID	Medicine_ID
180	1001
197	1002
3	1003

Medicine_ID	Assigned_Patient_ID
1001	35
1002	36
1003	657

From the above 3NF, **Doctor_ID** and **Doctor_name** are the candidate keys. **Medicine_ID** and **Assigned_Patient_ID** are not the super keys and therefore they are the non-prime attributes.
{Doctor_ID, Doctor_name} → {Medicine_ID, Assigned_Patient_ID}
Therefore, {Doctor_ID, Doctor_name}are the prime attributes.
From the Doctor_to_Medicine table, the super key is Doctor_ID.

Medical_Clinic Table

2NF

Clinic_Number	Clinic_Name	Clinic Location	Clinic Number
31	Marques Clinic	19 Plam Drive, Toronto	1-647-892-1923

		Ontario	
23	Better Help Clinic	54 Progress Ave, Toronto Ontario	1-847-999-3923
45	Smith Clinic	39 Plam Drive, Toronto Ontario	1-347-333-1333

3NF

Clinic_Number	Clinic_Name	Head_Doctor_ID	Clinic_Location	Clinic_Number
31	Marques Clinic	180	19 Plam Drive, Toronto Ontario	1-647-892-1923
23	Better Help Clinic	197	54 Progress Ave, Toronto Ontario	1-847-999-3923
45	Smith Clinic	3	39 Plam Drive, Toronto Ontario	1-347-333-1333

This is the derived table from its corresponding 2nf form to 3nf. This table tracks the location and details corresponding to each individual clinic.

BCNF

Head_Doctors

Clinic_Number	Head_Doctor_ID
31	180
23	197
45	3

Clinic_Subinfo

Clinic_Number	Clinic_Name	Clinic_Location	Clinic_Number
31	Marques	19 Plam Drive, Toronto	1-647-892-1923

	Clinic	Ontario	
23	Better Help Clinic	54 Progress Ave, Toronto Ontario	1-847-999-3923
45	Smith Clinic	39 Plam Drive, Toronto Ontario	1-347-333-1333

In the above 3NF, **Clinic_Num** and **Clinic_Name** are the candidate keys. **Head_Doctor_ID**, **Clinic_Location** and **Clinic_Number** are the non-prime attributes.
{Clinic_Num,Clinic_Name}→ {Head_Doctor,Location,Number}
The super key in this table would be Clinic_Number, to remove the possibility for the Head_Doctor to record the Clinic_Number as well.

Doctor Table

2NF

Doctor_Number (Same as Employee#)	Ext_Number	Specialty	Doctor_Location
33	23	Surgeon	99 Albatross Dr. Setel Ontario
22	34	General Physician	67 Supers Dr. Setel Ontario
41	35	Cardiologist	31 Ulter Dr. Thunderbay Ontario

3NF

Doctor_Number (Same as Employee#)	Ext_Number	Specialty	Supervisor_Employee_ID
33	23	Surgeon	NULL
22	34	General Physician	41
41	35	Cardiologist	33

This is the derived 3NF form without any transitive dependencies. The primary key is the Doctor_Number attribute.

BCNF

Doctor_Number	Ext_Number	Specialty
33	23	Surgeon
22	34	General Physician
41	35	Cardiologist

Doctor_Number	Supervisor_Employee_ID
33	NULL
22	41
41	33

In the above 3NF, **Doctor_Num** is the candidate key. **Ext_Num**, **Specialty** and **Supervisor_Employee_ID** are the non-prime attributes.
{Doctor_Number}→ {Ext_Number,Specialty,Supervisor_Employee_ID}
The super key in this table would be Doctor_Number, to normalize the table into BCNF form, by splitting the Doctor->Supervisor relation into a separate table.

QUERIES (SQL & RELATIONAL ALGEBRA)

SQL

Select * FROM PATIENT;

RELATIONAL ALGEBRA

Patient

EMPLOYEE_ID	EMPLOYEE_NAME
1001	Sri
1002	harry

SQL

Select Employee_ID FROM patient Where employee_id=1001;

RELATIONAL ALGEBRA

$\pi_{\text{employee\_id}}$

$\sigma_{\text{employee\_id} = 1001}$ patient

EMPLOYEE_ID
1001

SQL

SELECT

* FROM medical_clinic

order by clinic_number ASC;

RELATIONAL ALGEBRA

$\tau_{\text{clinic\_number}}$ medical_clinic

CLINIC_NAME	CLINIC_NUMBER	CLINIC_LOCATION	CLINIC_PHONE_NUMBER
General4	22	toronto	1234567823
General	23	brampton	64789234
General1	30	brampton	1234567890
General2	31	markham	1234567891
General3	35	brampton	1234567892

SQL

Select * FROM doctor;

RELATIONAL ALGEBRA

Doctor

DOCTOR_NAME	EMPL_NUMBER	EXT_NUMBER	SPECIALTY	DOCTOR_LOCATION
teja	222	34	gaenec	brampton
sreeja	233	45	cardiac	toronto
sai	344	56	neuro	ottawa

SQL

Select distinct Empl_Number from doctor;

RELATIONAL ALGEBRA

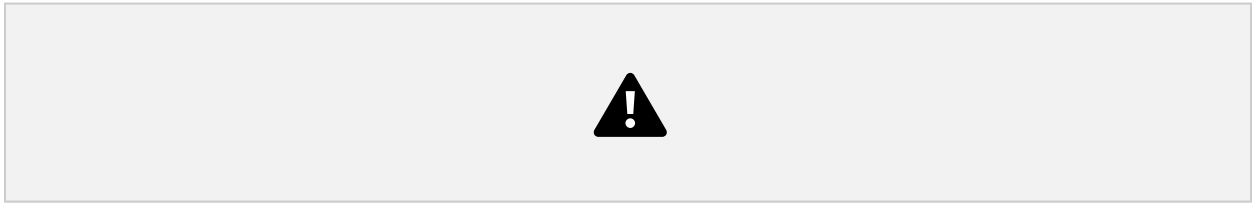
δ

$\pi_{empl\_number}\, doctor$

EMPL_NUMBER
222
233
344

SQL

```
CREATE VIEW male_employee AS
   (SELECT * FROM employee WHERE gender = 'M');
CREATE VIEW female_employee AS
   (SELECT * FROM employee WHERE gender = 'F');
```



	E_ID	E_FIRSTNAME	E_LASTNAME	GENDER	USERNAME	PASSWORD	ADDRESS
1	1	Ethan	Ting	M	Eting	password	address1
2	2	Bob	Joe	M	Bjoe	password	address2

SQL

```
SELECT COUNT(medical_record_id), patient_details
FROM medical_record
GROUP BY patient_details;
```

RELATIONAL ALGEBRA

$\gamma_{\text{patient\_details, COUNT (medical\_record\_id)}}$ medical_record

SQL

```
select count (quantity) from Medicine
having count (quantity) > 1;
```

RELATIONAL ALGEBRA

$\pi_{\text{COUNT (quantity)}}$

$\sigma_{\text{COUNT (quantity) > 1}}$

$\gamma_{\text{COUNT (quantity)}}$ medicine

COUNT(QUANTITY)
4

SQL

```
CREATE VIEW medicine_with_id_over_100 AS
SELECT MED_ID, MED_NAME
FROM MEDICINE_TABLE
WHERE MED_ID > 100;
```

	MED_ID	MED_NAME
1	222	Cefaclor
2	836	Enalapril
3	145	Regorafenib

SQL

```
CREATE VIEW nurses AS
SELECT EMP_ID, EMP_TYPE
FROM EMPLOYEE_TYPE_TABLE
WHERE EMP_TYPE = 'Nurse';
```

SQL

```
SELECT *
   FROM CLINIC_INFO
   WHERE CLINIC_ID < 10
   MINUS (SELECT * FROM CLINIC_INFO WHERE CLINIC_NAME ='North Clinic');
```

SQL

```
SELECT * FROM MEDICINE_TABLE WHERE Med_ID < 1000;
```

RELATIONAL ALGEBRA

$\sigma_{\text{med\_id} < 1000}$ medicine_table

MED_ID	MED_NAME
86	Amoxicillin
222	Cefaclor
836	Enalapril
21	Labetalol
145	Regorafenib

UNIX SHELL IMPLEMENTATION

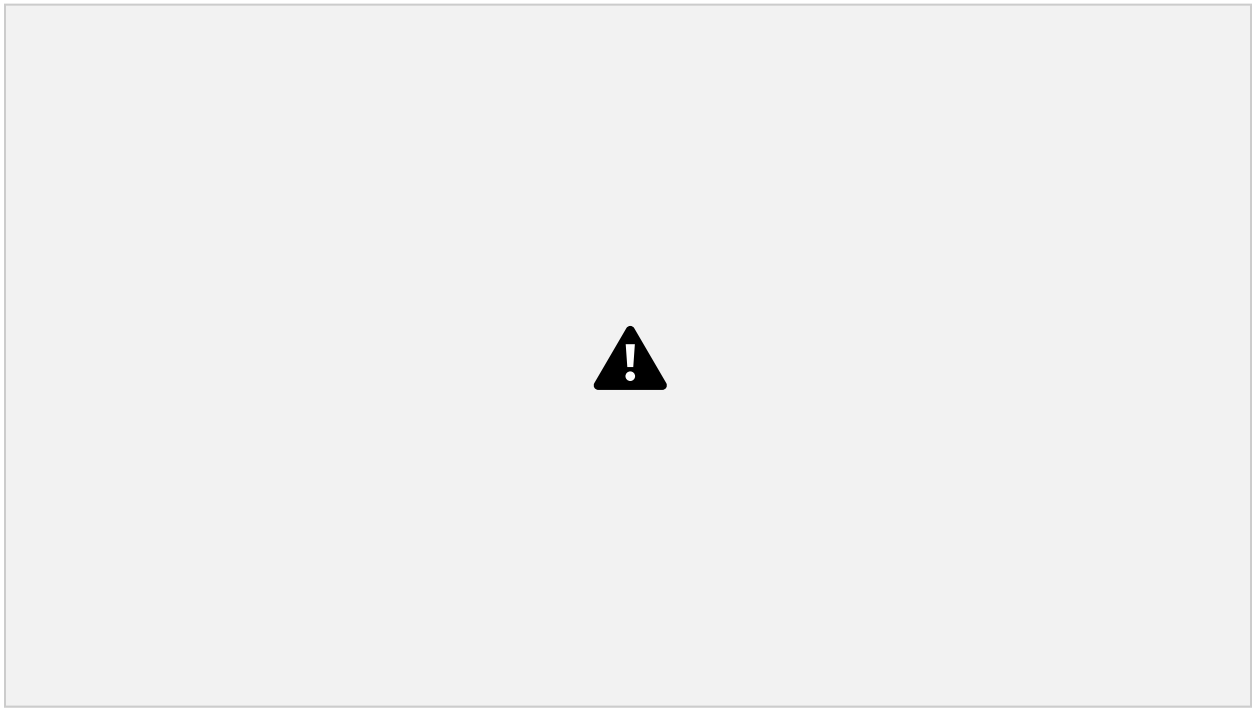
In our UNIX Shell implementation, we provide users and administrators the ability to drop, create, query and populate tables in our Clinic Information Database. These scripts also include some specific queries for the database.

Below is the code for menu.sh, create.sh,view.sh, and insert.sh

Menu.sh

```
#!/bin/sh
MainMenu() {

   while [ "$CHOICE" != "START" ]
   do

      clear
      echo "================================================================="
      echo "| Oracle All Inclusive Tool |"
      echo "| Main Menu - Select Desired Operation(s): |"
      echo "| <CTRL-Z Anytime to Enter Interactive CMD Prompt> |"
      echo "================================================================="
      echo " $IS_SELECTEDM M) View Manual"
      echo " "
      echo " $IS_SELECTED1 1) Drop Tables"
      echo " $IS_SELECTED2 2) Create Tables"
      echo " $IS_SELECTED3 3) Populate Tables"
      echo " $IS_SELECTED4 4) View Tables"
      echo " "
      echo " $IS_SELECTEDX X) Force/Stop/Kill Oracle DB"
      echo " "
      echo " $IS_SELECTEDE E) End/Exit"
      echo "Choose: "

      read CHOICE
 if [ "$CHOICE" = "0" ]
      then
         echo "Nothing Here"
      elif [ "$CHOICE" = "1" ]
     bash drop.sh
         Pause
      elif [ "$CHOICE" = "2" ]
      then
         bash create.sh
         Pause
      elif [ "$CHOICE" = "3" ]
      then
         bash insert.sh
         Pause
elif [ "$CHOICE" = "4" ]
      then
         bash view.sh
         Pause
      elif [ "$CHOICE" = "E" ]
      then
         exit
```

```
        fi
    done
#--COMMENTS BLOCK--
# Main Program
#--COMMENTS BLOCK--
ProgramStart()
{
    StartMessage
while [ 1 ]
    do
        MainMenu
    done
}
ProgramStart
```



Drop.sh

```
#export LD_LIBRARY_PATH=/usr/lib/oracle/12.1/client64/lib
sqlplus64 "Username/Password(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(Host=oracle.scs$
DROP TABLE PATIENT_CHECK1
DROP TABLE PATIENT_CHECK2
exit;
EOF
```

```
Choose:
1

SQL*Plus: Release 12.1.0.2.0 Production on Thu Oct 28 09:50:55 2021

Copyright (c) 1982, 2014, Oracle.  All rights reserved.


Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL>
Table dropped.

SQL> Disconnected from Oracle Database 11g Enterprise Edition Release 11.2.0.1.0
 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options
./menu.sh: 32: ./menu.sh: Pause: not found

=================================================================
```

Create.sh

```
!/bin/sh
#export LD_LIBRARY_PATH=/usr/lib/oracle/12.1/client64/lib
sqlplus64 "Username/Password@(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(Host=oracle.scs$
CREATE TABLE Nurse_Name_Table (ID varchar(255) REFERENCES Employee_Type_Table(Emp_ID), E_Name varchar(255))";
CREATE TABLE Doctor_Specialty (D_ID varchar(255) REFERENCES Employee_Type_Table(Emp_ID), Ext_Number varchar(255),
Specialty varchar(255));
CREATE TABLE Employee_Login_Details (Emp_ID varchar(255) REFERENCES Employee_Type_Table(Emp_ID), Username
varchar(255),Pword varchar(255))";
CREATE TABLE Employee_Personal_Details (Emp_ID varchar(255) REFERENCES Employee_Type_Table(Emp_ID),Gender
varchar(255),Address varchar(255));
exit;
EOF
```

```
Choose:
2

SQL*Plus: Release 12.1.0.2.0 Production on Thu Oct 28 09:51:13 2021

Copyright (c) 1982, 2014, Oracle.  All rights reserved.


Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL>   2    3    4    5    6    7    8    9   10
Table created.

SQL> Disconnected from Oracle Database 11g Enterprise Edition Release 11.2.0.1.0
 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options
./menu.sh: 36: ./menu.sh: Pause: not found
```

Insert.sh

```
#export LD_LIBRARY_PATH=/usr/lib/oracle/12.1/client64/lib
sqlplus64 "Username/Password(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(Host=oracle.scs$
INSERT INTO CLINIC_INFO VALUES(1,'West Clinic','1062 West Street, Toronto','7528825732')"
INSERT INTO CLINIC_INFO VALUES(2,'East Clinic','888 East Street, Waterloo','8471920594')"
INSERT INTO MEDICINE_TABLE VALUES(86, 'Amoxicillin')
INSERT INTO MEDICINE_TABLE VALUES(222, 'Cefaclor')
INSERT INTO Employee_Type_Table VALUES(1, 'Nurse')
INSERT INTO Employee_Login_Details VALUES(1, 'test', 'test')
exit;
EOF
```

```
Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL>
1 row created.

SQL>
1 row created.

SQL>
1 row created.

SQL> SQL> INSERT INTO doctor VALUES(2)
            *
ERROR at line 1:
ORA-00947: not enough values


SQL> Disconnected from Oracle Database 11g Enterprise Edition Release 11.2.0.1.0
 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options
./menu.sh: 40: ./menu.sh: Pause: not found
```

View.sh

```
!/bin/sh
#export LD_LIBRARY_PATH=/usr/lib/oracle/12.1/client64/lib
sqlplus64 "Username/Password@(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(Host=oracle.scs$
SELECT * FROM CLINIC_INFO",
SELECT * FROM MEDICAL_RECORD",
CREATE VIEW male_employees AS SELECT EMP_ID, EMP_NAME, GENDER FROM EMPLOYEE_PERSONAL_DETAILS
WHERE GENDER = 'M'",
CREATE VIEW medicine_with_id_over_100 AS SELECT MED_ID, MED_NAME FROM MEDICINE_TABLE WHERE MED_ID >
100"
exit;
EOF
```

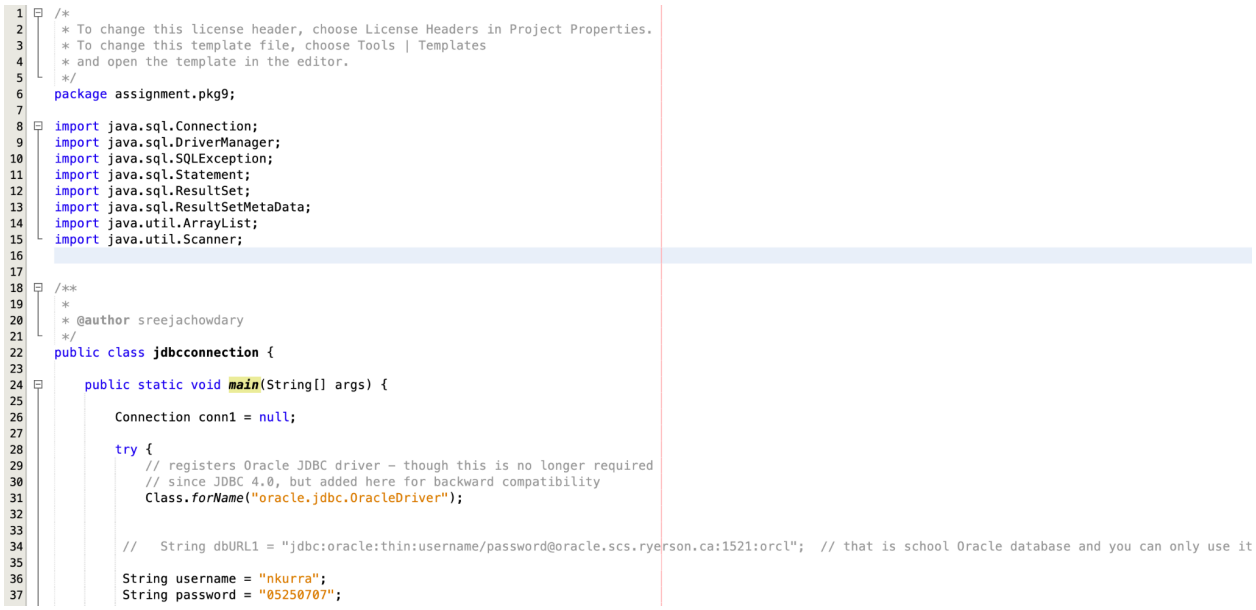
```
20210422
         2
Stomach Pain


MEDICAL_RECORD_ID
-----------------
APPOINTMENT
--------------------------
      P_ID
----------
PATIENT_DETAILS
--------------------------
                 3
20210104
         1
Cold,Cough
```

JAVA GUI SHELL IMPLEMENTATION

In our Java implementation, we designed a GUI using JAVA that would be typically used by an Employee. It is representative of Patient information into the system.

The Java Code which is being attached below has been modified according to the SQL commands and it has been designed according to the menu. The tables have been created, dropped, viewed and executed.

Below is the Java code:



```java
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package assignment.pkg9;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.util.ArrayList;
import java.util.Scanner;


/**
 *
 * @author sreejachowdary
 */
public class jdbcconnection {

    public static void main(String[] args) {

        Connection conn1 = null;

        try {
            // registers Oracle JDBC driver - though this is no longer required
            // since JDBC 4.0, but added here for backward compatibility
            Class.forName("oracle.jdbc.OracleDriver");


            //   String dbURL1 = "jdbc:oracle:thin:username/password@oracle.scs.ryerson.ca:1521:orcl";  // that is school Oracle database and you can only use it

            String username = "nkurra";
            String password = "05250707";
```

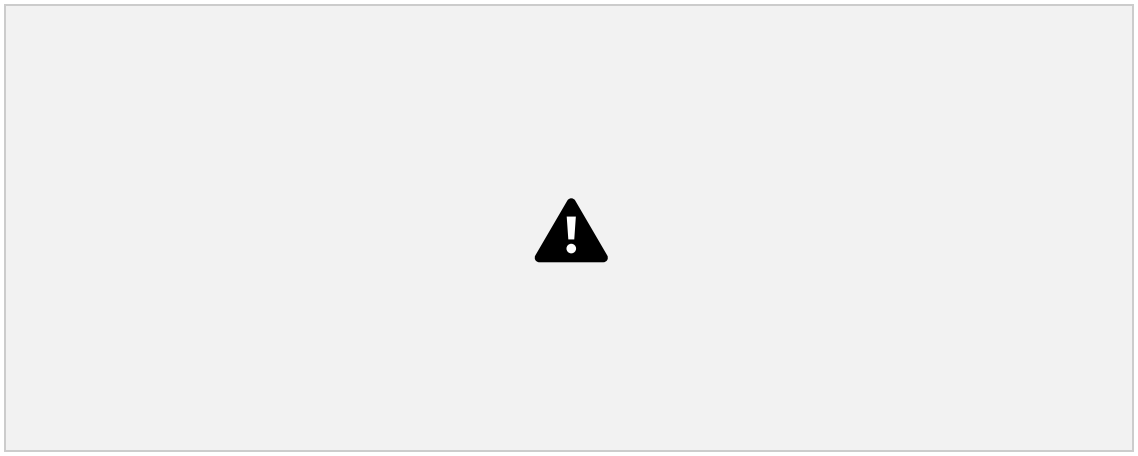
```
            String dbURL1 = "jdbc:oracle:thin:" + username + "/" + password + "@oracle.cs.ryerson.ca:1521:orcl";
                    
                    conn1 = DriverManager.getConnection(dbURL1);
            if (conn1 != null) {
                System.out.println("Connected with connection #1");
            }
                    String query = "select NAME, NUM from TESTJDBC";

                    try (Statement stmt = conn1.createStatement()) {
                        Scanner in = new Scanner(System.in);
                        char inp;
                        boolean stay = true;
                        System.out.println("1 - Create Tables\n2 - PopulateTables\n3 - Query Tables\n4 - Drop Tables\ne - Exit");
                        while (stay) {
                            System.out.print("=> ");
                            inp = in.next().charAt(0);
                            switch (inp) {
                                case '1':
                                    createTables(stmt);
                                    break;
                                case '2':
                                    populateTables(stmt);
                                    break;
                                case '3':
                                    queryTables(stmt);
                                    break;
                                case '4':
                                    dropTables(stmt);
                                    break;
                                case 'e':
                                    stay = false;
                                    System.out.println("Exiting Program");
                                    break;
                                default:
                                    System.out.println("Invalid Input");

                            }
```

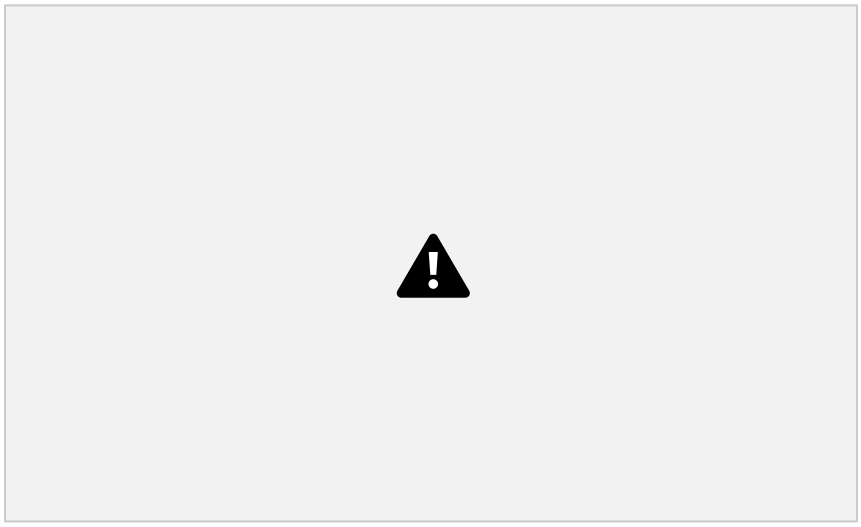
```
                    } catch (SQLException e) {
                            System.out.println("Error " + e.getErrorCode());
                            //e.printStackTrace();
                    }



    } catch (ClassNotFoundException | SQLException ex) {
    } finally {
        try {
            if (conn1 != null && !conn1.isClosed()) {
                conn1.close();
            }

        } catch (SQLException ex) {
        }
    }


}

public static void createTables(Statement stmt) throws SQLException {
    String[] queries = {/*"CREATE TABLE Nurse_Name_Table (ID varchar(255) REFERENCES Employee_Type_Table(Emp_ID), E_Name varchar(255))",
                        "CREATE TABLE Doctor_Specialty (D_ID varchar(255) REFERENCES Employee_Type_Table(Emp_ID), Ext_Number varchar(255), Specialty varchar(255))",
                        "CREATE TABLE Employee_Login_Details (Emp_ID varchar(255) REFERENCES Employee_Type_Table(Emp_ID), Username varchar(255),Pword varchar(255))",
                        "CREATE TABLE Employee_Personal_Details (Emp_ID varchar(255) REFERENCES Employee_Type_Table(Emp_ID),Gender varchar(255),Address varchar(255))
    for (String query : queries) {
        stmt.executeUpdate(query);
    }
    System.out.println("Tables Created");
}

```

```
    public static void populateTables(Statement stmt) throws SQLException {
        String[] queries = {//"INSERT INTO CLINIC_INFO VALUES(1,'West Clinic','1062 West Street, Toronto','7528825732')",
                            //"INSERT INTO CLINIC_INFO VALUES(2,'East Clinic','888 East Street, Waterloo','8471920594')",
                            "INSERT INTO MEDICINE_TABLE VALUES(86, 'Amoxicillin')",
                            "INSERT INTO MEDICINE_TABLE VALUES(222, 'Cefaclor')",
                            //"INSERT INTO Employee_Type_Table VALUES(1, 'Nurse')",
                            //"INSERT INTO Employee_Login_Details VALUES(1, 'test', 'test')"
                            };
        for (String query : queries) {
            stmt.executeUpdate(query);
        }
        System.out.println("Tables Populated");
    }

    public static void queryTables(Statement stmt) throws SQLException {
        String[] queries = {"SELECT * FROM CLINIC_INFO",
                            "SELECT * FROM MEDICAL_RECORD",
                            /*"CREATE VIEW male_employees AS SELECT EMP_ID, EMP_NAME, GENDER FROM EMPLOYEE_PERSONAL_DETAILS WHERE GENDER = 'M'",
                            "CREATE VIEW medicine_with_id_over_100 AS SELECT MED_ID, MED_NAME FROM MEDICINE_TABLE WHERE MED_ID > 100",*/};
        for (String query : queries) {
            System.out.println(query);
            ResultSet rs = stmt.executeQuery(query);
            ResultSetMetaData rsmd = rs.getMetaData();
            ArrayList<String> columns = new ArrayList<>();
            for(int i = 1; i <= rsmd.getColumnCount(); i++) {
                columns.add(rsmd.getColumnName(i));
                System.out.print(rsmd.getColumnName(i));
                if (i != rsmd.getColumnCount())
                    System.out.print(", ");
            }
            System.out.println();
            while (rs.next()) {
                for (String c : columns) {
                    System.out.print(rs.getString(c));
                    if (!columns.get(columns.size() - 1).equals(c))
                        System.out.print(", ");
                }
                System.out.println();
            }
                    }
                    System.out.println();
                }
                System.out.println();
            }
            System.out.println("Tables Queried");
        }

        public static void dropTables(Statement stmt) throws SQLException {
            String[] queries = {
                                "DROP TABLE STUDENT"
                                /*"DROP TABLE PATIENT_CHECK1",
                                "DROP TABLE PATIENT_CHECK2"*/};
            for (String query : queries) {
                stmt.executeUpdate(query);
            }
            System.out.println("Tables Dropped");
        }

    }
```

The following output will be based on the tables created, queried, dropped and inserted.

```
run:
Connected with connection #1
1 - Create Tables
2 - PopulateTables
3 - Query Tables
4 - Drop Tables
e - Exit
=> 1
Tables Created
```





```
Connected with connection #1
1 - Create Tables
2 - PopulateTables
3 - Query Tables
4 - Drop Tables
e - Exit
=> 2
Tables Populated
```

These are the views created for the assignment 9 for the GUI. The results have been shown during the demo.

CONCLUSION

Working on this Medical Clinic Information System has provided us with a solid foundation in all aspects of database design and implementation. Previously, we had not realized how important it was to represent data in a clear and concise manner. With the concepts we have learned like entity-relationship diagrams, relational schema design, functional dependencies, normalization, etc., we were able to turn various pieces of data into a useful and accessible database.

If we look into the technical aspects of the DBMS, we become familiar with SQL commands and the services provided by Oracle. Through the following assignments, we have learned how to create, drop, insert and query tables according to the user requirements. Making GUI using Java was a fun experiment and we are also familiar with how a front-end interface connects and interacts with a backend database.

This project also exercised our skills in teamwork, project management, and software development. My teammates have done a great job during this project. All of us were very helpful to each other and tried our best to do all the assigned tasks.

In conclusion, it was a pleasure working on this Medical Clinic Information System database. It truly allowed us to use the knowledge and skills acquired in this course.