Term Project

Group members:

Meet Patel

Nidheesh Kumar

Polad Paul

Adam Jusino

Uername: meet19web

Password: meet1999

A. Application background:

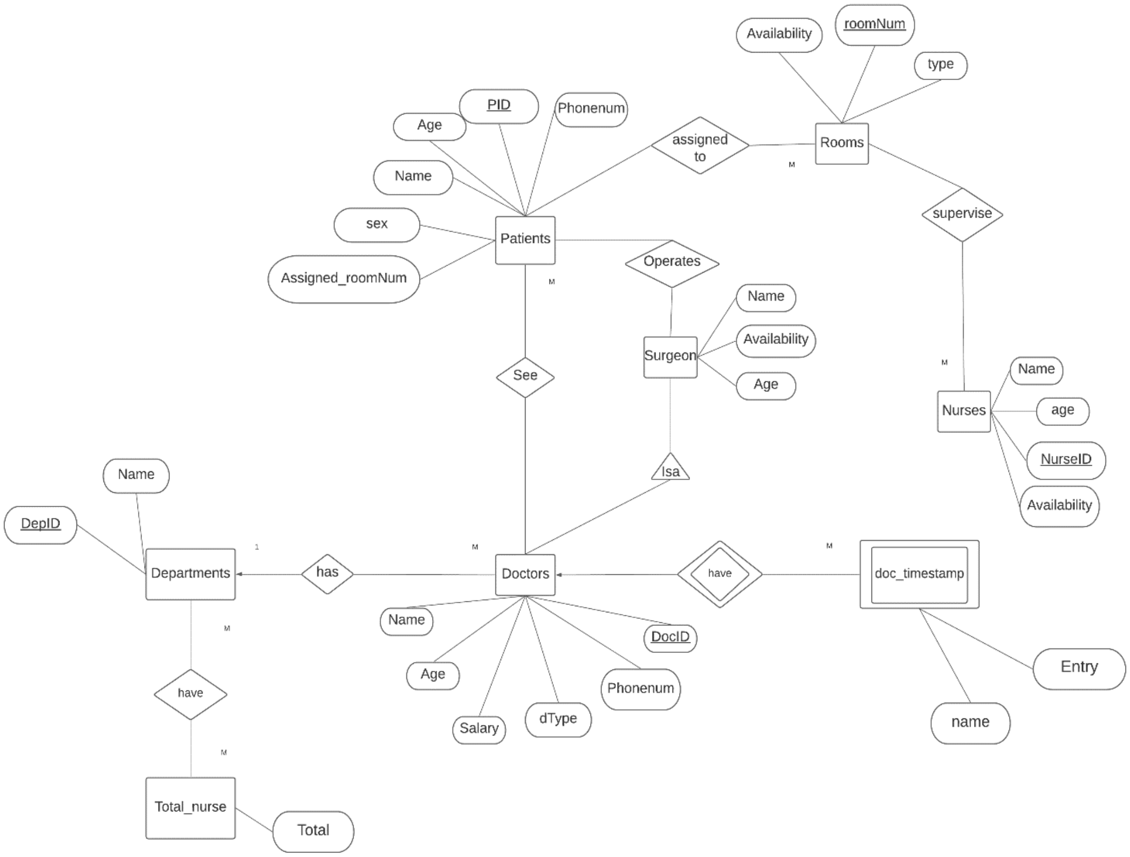
- We developed a first iteration database from a hospital model, our ER diagram reflects the relationships used in this database (shown below).
- The hospital model entity-relationship diagram depicts all of the visual instruments of database tables as well as the relationships between patients, nurses, departments, and rooms, among other things. It makes use of structured data and defines the linkages between structured data groupings in the hospital model. Departments, Doctors, Patients, Surgeons, Rooms, and Nurses are some of the primary entities in our hospital database.
- The hospital database is a good blueprint/skeletal implementation for a more advanced database. It will serve as a good first iteration for someone to build upon and create a more sophisticated database.
- The user of this database will have access to be able to input values on all of our 6 main entities mentioned earlier. The input value can be based on real scenarios from a hospital, like a patient entry log, a doctor's salary as well as timestamps, room availability and more. We have also made this system more efficient by creating views, adding constraints, indexes and triggers (each of these additions will be further explained in their section below).

Use Cases of application:

- This application provides the hospital a pace to put a patients information like their name, age, sex etc.
- The data shows the information about the availability of the rooms, doctors, and nurses.

- The data also shows all the necessary information about the doctors, like doctors name and what type of doctor it is (ex, cardiologist, radiologist, ect).
- The data provides the information about the major entities of a hospital.
- The application can also keep a record of current time stamps for doctors who join the hospital.
- The application also serves as a place for payroll, it will show how much the doctors at the hospitals make a month.
- The application will also show contact information (phone number) for doctors and patients.

B. Hospital database ER diagram:



Assumptions:

- Departments have the attributes DepID (key), and name. One department has many doctors. This is because we can have two doctors with the same occupation which would lead them to be in the same department. Also, one weak entity is total_nurse (have) is a supporting relationship. The assumption is that the departments have the total count of all of the nurses
- Doctors have the attributes Name, Age, dType, Phonenum, and DocID (key). One weak entity is doc_timestamp (has) is a supporting relationship. The assumption is that doctors have a time stamp. One doctor can see many patients. This is because one particular doctor can see many patients in one workday while one patient can go into the hospital only to be seen by one doctor. A surgeon is a doctor.
- A surgeon's attributes are Name, Availability, and Age. One surgeon can operate on many patients.
- A patient's attributes are sex, Name, Age, pID (key), and Phonenum. Many Patients can be assigned to many Rooms. This is because rooms can be inhabited by many patients and one patient can visit many rooms.
- Rooms have the attributes Availability, roomNum (key), and type. Many Nurses can supervise many rooms. Nurses have the attributes Name, age, NurseID (key), and availability.

C. Relational Schema (3NF):

3NF rule: For each functional dependencies LHS must be a candidate key or super key or RHS must be a prime attribute.

1.

- Relation: Department (<u>depID</u>, name)
- Functional Dependencies:
  - depID -> name
- Candidate Key:
  - {depID}
- Prime attribute:
  - {depID}
- Non-prime attribute:
  - {name}

English description:

This relation determines relation between depID and name where depID is a primary key and it determines name.

2.

- Relation: Doctors (docID, name, dtype, age, salary, phonenum)
- Functional Dependencies:
  - docID dtype -> name age phonenum
  - docID -> salary
  - phonenum -> docID
- Candidate key:
  - {docID dtype, phonenum dtype}
- Prime attribute:
  - {docID, dtype, phonenum}
- Non-prime attribute:
  - {name, salary, age}

English Description:

This relation determines doctors' attributes. The primary key is docID and there are three dependencies which does not violate the 3NF rules.

3.

- Relation: Patients (PID, name, age, phonenum, sex, assigned_roomnum)
- Functional Dependencies:
  - PID name -> age salary sex assigned_roomnum
  - PID -> name age phonenum sex assigned_roomnum
  - PID age -> name phonenum sex
  - age -> PID
- Candidate Key:
  - {PID, PID name, age name}
- Prime attribute:
  - {PID, name, age}

- Non-prime attribute:
  - {sex, assigned_roomnum, phonenum}

English description:

This relation determines patients' relation and attributes. The primary key is PID and each FDs do not violate the 3NF rules.

4.

- Relation: Nurses (nurseID, availability, age, name)
- Functional Dependencies:
  - nurseID -> availability age
  - nurseID name -> availability age
  - availability -> nurseID
- Candidate Keys:
  - {nurseID name, availability name}
- Prime attributes:
  - {nurseID, name, availability}
- Non-prime attributes:
  - {age}

English description:

This relation determines Nurse's relation and attributes. The primary key is nurseID and each FDs do not violate the 3NF rules.

5.

- Relation: Rooms (availability, type, roomnum)
- Functional Dependency:
  - availability -> roomnum
  - type -> roomnum
  - roomnum type -> availability
  - roomnum -> type availability
- Candidate key:
  - {roomnum type, availability type, type, availability}
- Prime attributes:
  - {roomnum, type, availability}
- Non-prime attribute:
  - { }

English description:

This relation determines rooms' relation and attributes. The primary key is roomnum and each FDs do not violate the 3NF rules.

6.

Relation: logbook (name, log_time)

Functional Dependency:

name -> log_time

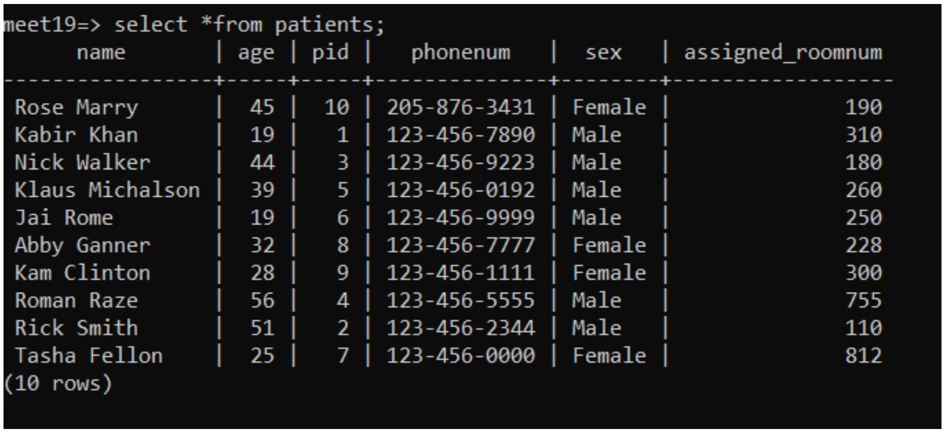
English description:

logbook is weak entity in the database so that it does not have prime attribute, so that this will be just a normal relational schema.

D. Sample Data:

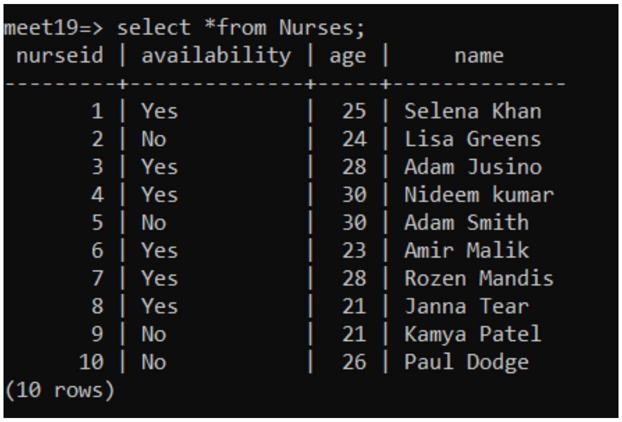
We created table for each entity with 10 tuples in each table.

Patients Table:



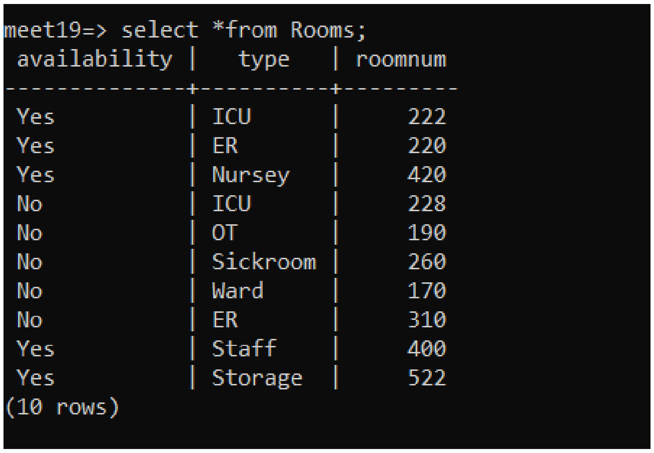
```
meet19=> select *from patients;
      name       | age | pid |   phonenum   |  sex   | assigned_roomnum
-----------------+-----+-----+--------------+--------+------------------
 Rose Marry      |  45 |  10 | 205-876-3431 | Female |              190
 Kabir Khan      |  19 |   1 | 123-456-7890 | Male   |              310
 Nick Walker     |  44 |   3 | 123-456-9223 | Male   |              180
 Klaus Michalson |  39 |   5 | 123-456-0192 | Male   |              260
 Jai Rome        |  19 |   6 | 123-456-9999 | Male   |              250
 Abby Ganner     |  32 |   8 | 123-456-7777 | Female |              228
 Kam Clinton     |  28 |   9 | 123-456-1111 | Female |              300
 Roman Raze      |  56 |   4 | 123-456-5555 | Male   |              755
 Rick Smith      |  51 |   2 | 123-456-2344 | Male   |              110
 Tasha Fellon    |  25 |   7 | 123-456-0000 | Female |              812
(10 rows)
```

Nurses Table:



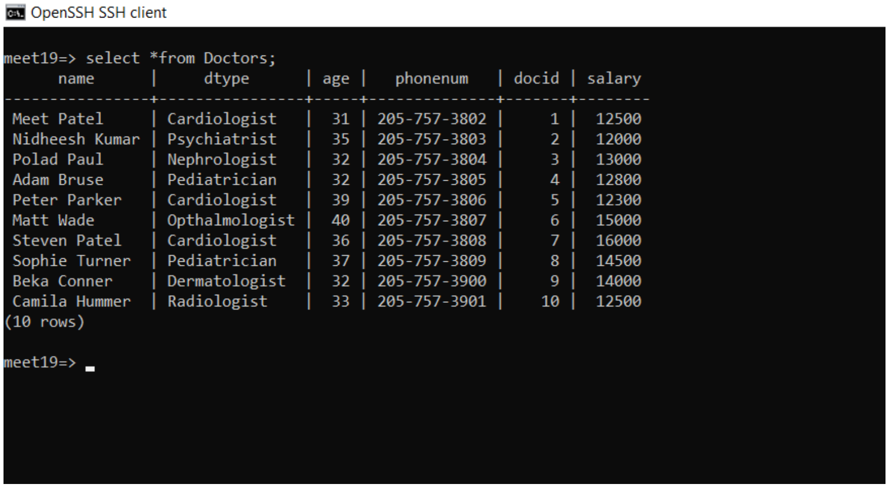
```
meet19=> select *from Nurses;
 nurseid | availability | age |     name
---------+--------------+-----+-------------
       1 | Yes          |  25 | Selena Khan
       2 | No           |  24 | Lisa Greens
       3 | Yes          |  28 | Adam Jusino
       4 | Yes          |  30 | Nideem kumar
       5 | No           |  30 | Adam Smith
       6 | Yes          |  23 | Amir Malik
       7 | Yes          |  28 | Rozen Mandis
       8 | Yes          |  21 | Janna Tear
       9 | No           |  21 | Kamya Patel
      10 | No           |  26 | Paul Dodge
(10 rows)
```

Rooms Table:



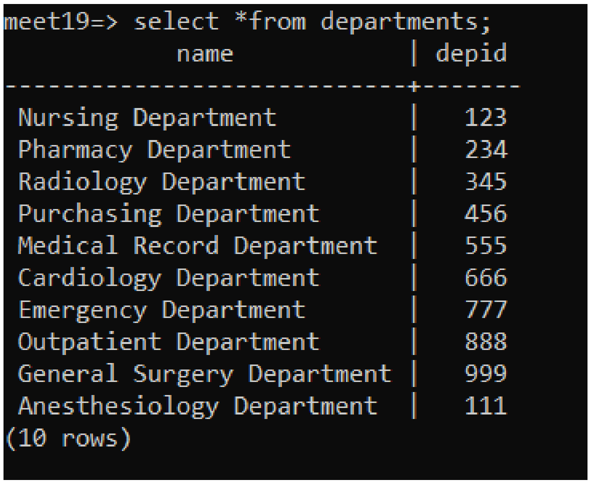
```
meet19=> select *from Rooms;
 availability |   type    | roomnum
--------------+-----------+---------
 Yes          | ICU       |     222
 Yes          | ER        |     220
 Yes          | Nursey    |     420
 No           | ICU       |     228
 No           | OT        |     190
 No           | Sickroom  |     260
 No           | Ward      |     170
 No           | ER        |     310
 Yes          | Staff     |     400
 Yes          | Storage   |     522
(10 rows)
```

Doctors Table:



```
meet19=> select *from Doctors;
      name      |     dtype      | age |   phonenum   | docid | salary
----------------+----------------+-----+--------------+-------+--------
 Meet Patel     | Cardiologist   |  31 | 205-757-3802 |     1 |  12500
 Nidheesh Kumar | Psychiatrist   |  35 | 205-757-3803 |     2 |  12000
 Polad Paul     | Nephrologist   |  32 | 205-757-3804 |     3 |  13000
 Adam Bruse     | Pediatrician   |  32 | 205-757-3805 |     4 |  12800
 Peter Parker   | Cardiologist   |  39 | 205-757-3806 |     5 |  12300
 Matt Wade      | Opthalmologist |  40 | 205-757-3807 |     6 |  15000
 Steven Patel   | Cardiologist   |  36 | 205-757-3808 |     7 |  16000
 Sophie Turner  | Pediatrician   |  37 | 205-757-3809 |     8 |  14500
 Beka Conner    | Dermatologist  |  32 | 205-757-3900 |     9 |  14000
 Camila Hummer  | Radiologist    |  33 | 205-757-3901 |    10 |  12500
(10 rows)

meet19=>
```

Department Table:



```
meet19=> select *from departments;
           name            | depid
---------------------------+-------
 Nursing Department        |   123
 Pharmacy Department       |   234
 Radiology Department      |   345
 Purchasing Department     |   456
 Medical Record Department |   555
 Cardiology Department     |   666
 Emergency Department      |   777
 Outpatient Department     |   888
 General Surgery Department |   999
 Anesthesiology Department |   111
(10 rows)
```

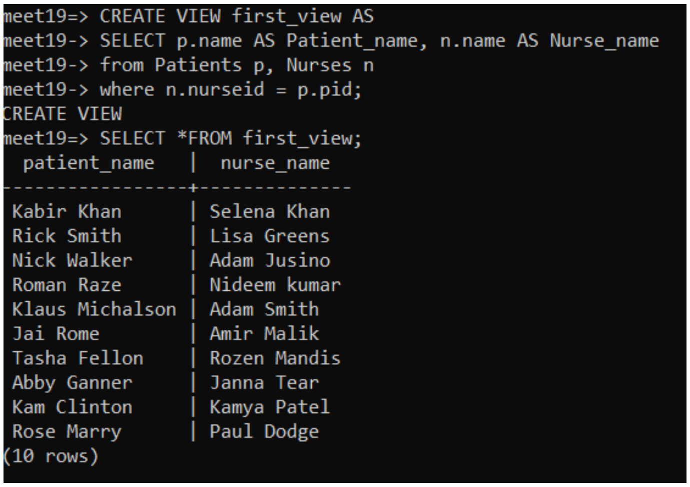
E. CREATE VIEWS:

First_view:

CREATE VIEW first_view As SELECT p.name AS patient_name, n.name AS Nurse_name

From Patients P, Nurses n Where n.nurseid = p.pid;

This query will tell use which nurse it taking care of which patient. Each nurse will have same id as the patient have.



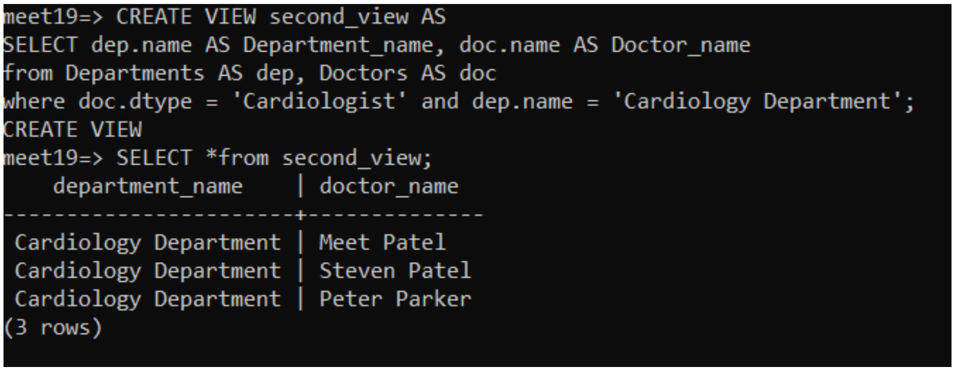
```
meet19=> CREATE VIEW first_view AS
meet19-> SELECT p.name AS Patient_name, n.name AS Nurse_name
meet19-> from Patients p, Nurses n
meet19-> where n.nurseid = p.pid;
CREATE VIEW
meet19=> SELECT *FROM first_view;
  patient_name   |  nurse_name
-----------------+-------------
 Kabir Khan      | Selena Khan
 Rick Smith      | Lisa Greens
 Nick Walker     | Adam Jusino
 Roman Raze      | Nideem kumar
 Klaus Michalson | Adam Smith
 Jai Rome        | Amir Malik
 Tasha Fellon    | Rozen Mandis
 Abby Ganner     | Janna Tear
 Kam Clinton     | Kamya Patel
 Rose Marry      | Paul Dodge
(10 rows)
```

Second_view

CREATE VIEW second_view AS SELECT dep.name AS Department_name, doc.name AS Doctor_name From departments AS dep, Doctors AS doc

Where doc.type = ‘Cardiologist’ and dep.name = ‘Cardiology Departmetn’;

This query will help us to know which doctors are in which departments, and we can find their names with it.



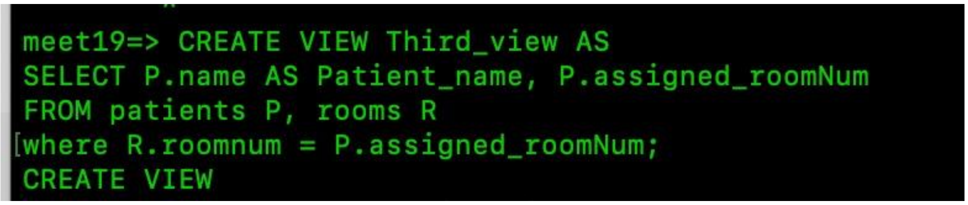
```
meet19=> CREATE VIEW second_view AS
SELECT dep.name AS Department_name, doc.name AS Doctor_name
from Departments AS dep, Doctors AS doc
where doc.dtype = 'Cardiologist' and dep.name = 'Cardiology Department';
CREATE VIEW
meet19=> SELECT *from second_view;
    department_name    | doctor_name
-----------------------+--------------
 Cardiology Department | Meet Patel
 Cardiology Department | Steven Patel
 Cardiology Department | Peter Parker
(3 rows)
```

Third view

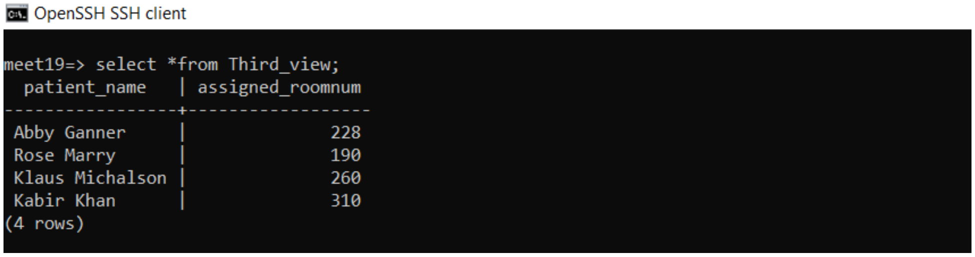
CREATE VIEW third_view As SELECT p.name AS Patient_name, P.assigned_roomNum

From patients P, rooms R Where R.roomnum = p.assigned_roomnum;

This query will help us know which room is assigned to which patients from the rooms table and the patient's table.



```
meet19=> CREATE VIEW Third_view AS
SELECT P.name AS Patient_name, P.assigned_roomNum
FROM patients P, rooms R
[where R.roomnum = P.assigned_roomNum;
CREATE VIEW
```

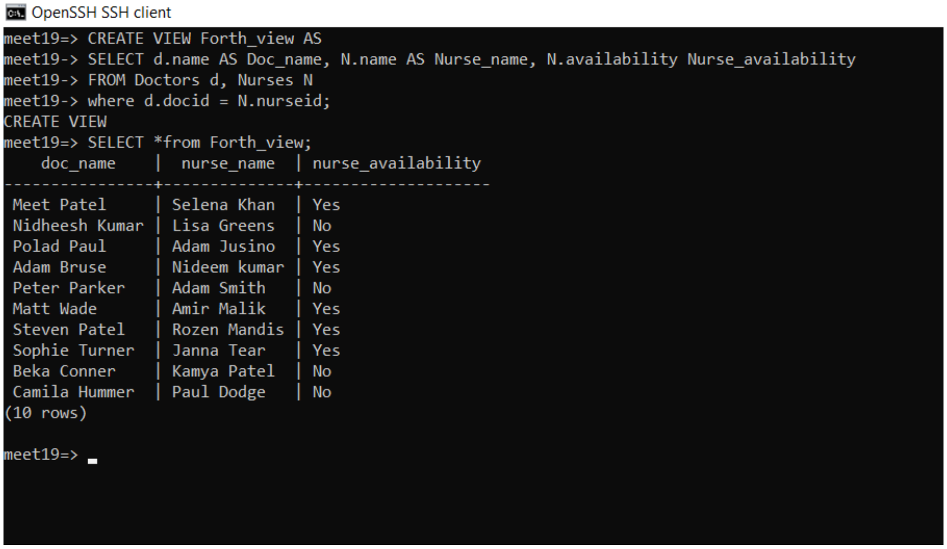


```
meet19=> select *from Third_view;
  patient_name   | assigned_roomnum
-----------------+------------------
 Abby Ganner     |              228
 Rose Marry      |              190
 Klaus Michalson |              260
 Kabir Khan      |              310
(4 rows)
```

Forth view

CREATE VIEW forth_view AS SELECT d.name AS Doc_name, N.name AS Nurse_name, N.availability Nurse_availability From Doctors d, Nurses N where d.docid = N.nurseid;

This will query help us know which nurse is working with which doctors, and it can be find by the given id to both entities which will be the same for both.



```
meet19=> CREATE VIEW Forth_view AS
meet19-> SELECT d.name AS Doc_name, N.name AS Nurse_name, N.availability Nurse_availability
meet19-> FROM Doctors d, Nurses N
meet19-> where d.docid = N.nurseid;
CREATE VIEW
meet19=> SELECT *from Forth_view;
    doc_name     |  nurse_name  | nurse_availability
-----------------+--------------+--------------------
 Meet Patel      | Selena Khan  | Yes
 Nidheesh Kumar  | Lisa Greens  | No
 Polad Paul      | Adam Jusino  | Yes
 Adam Bruse      | Nideem kumar | Yes
 Peter Parker    | Adam Smith   | No
 Matt Wade       | Amir Malik   | Yes
 Steven Patel    | Rozen Mandis | Yes
 Sophie Turner   | Janna Tear   | Yes
 Beka Conner     | Kamya Patel  | No
 Camila Hummer   | Paul Dodge   | No
(10 rows)

meet19=>
```

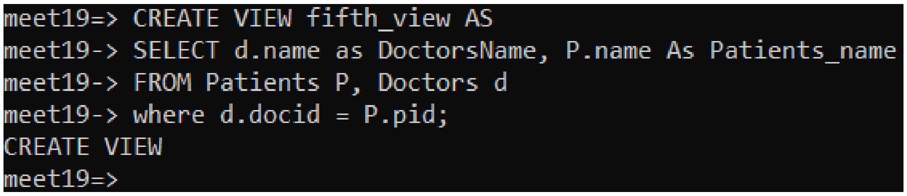
Fifth view

CREATE VIEW fifth_view AS

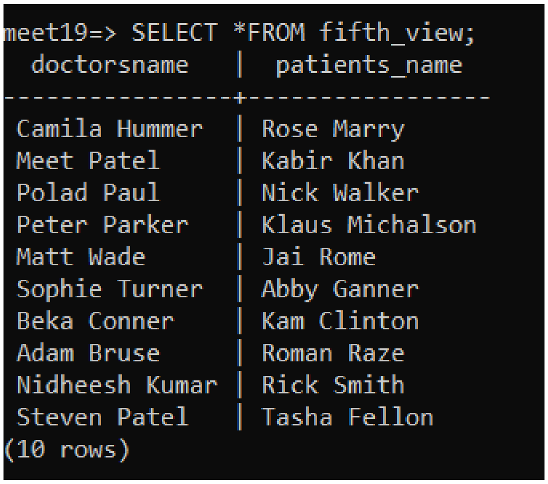
SELECT d.name as DoctorsName, P.name As Patients_name FROM Patients P, Doctors d

where d.docid = P.pid;

This query will help us find which patient is being seen by doctor, it will help user to find the doctor and patients name easily.



```
meet19=> CREATE VIEW fifth_view AS
meet19-> SELECT d.name as DoctorsName, P.name As Patients_name
meet19-> FROM Patients P, Doctors d
meet19-> where d.docid = P.pid;
CREATE VIEW
meet19=>
```



```
meet19=> SELECT *FROM fifth_view;
  doctorsname    |  patients_name
-----------------+-----------------
 Camila Hummer   | Rose Marry
 Meet Patel      | Kabir Khan
 Polad Paul      | Nick Walker
 Peter Parker    | Klaus Michalson
 Matt Wade       | Jai Rome
 Sophie Turner   | Abby Ganner
 Beka Conner     | Kam Clinton
 Adam Bruse      | Roman Raze
 Nidheesh Kumar  | Rick Smith
 Steven Patel    | Tasha Fellon
(10 rows)
```

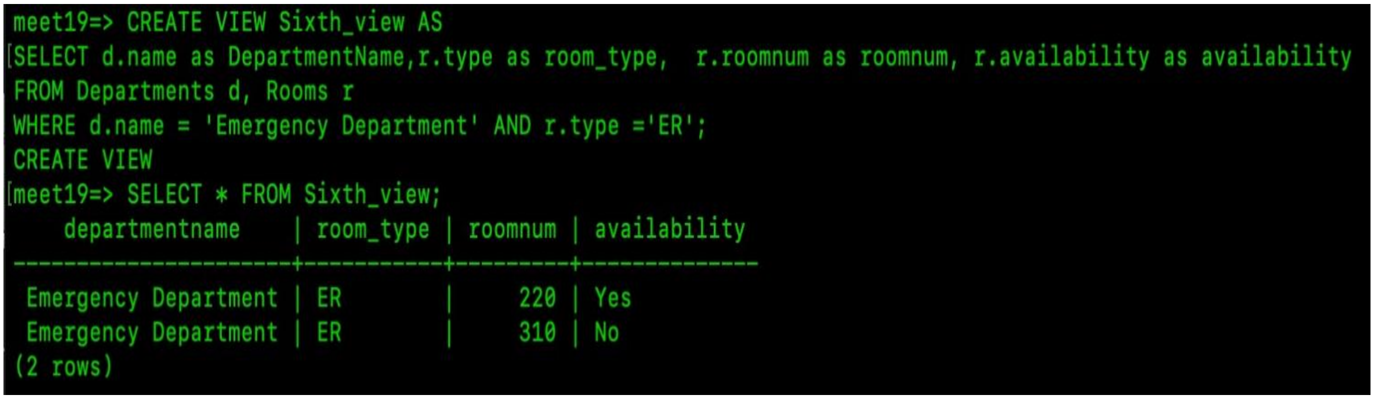
Sixth_view

CREATE VIEW sixth_view AS SELECT d.name as DepartmentName, r.type as room_type,

r.roomnum as roomnum, r.availability as availability from Departments d, Rooms r

where d.name = ‘Emergency Department’ AND r.type =’ER’;

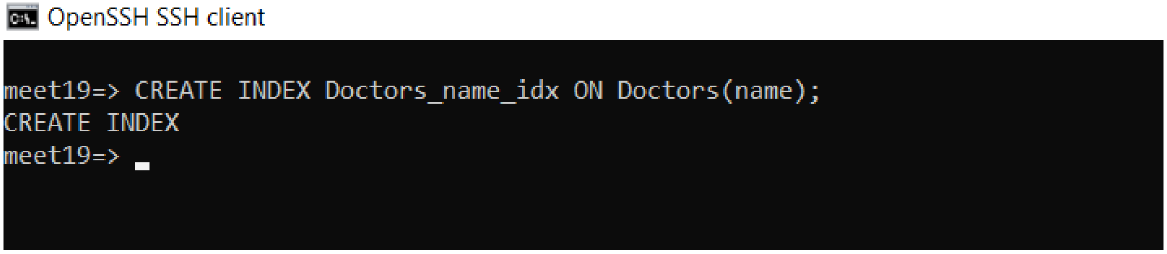
this query will help use to find the room availability, and what type of room is it from the department name from the department table. It is helpful to have the type, and availability and the department type in one query so that we can know which room is available.



```
meet19=> CREATE VIEW Sixth_view AS
[SELECT d.name as DepartmentName,r.type as room_type,  r.roomnum as roomnum, r.availability as availability
 FROM Departments d, Rooms r
 WHERE d.name = 'Emergency Department' AND r.type ='ER';
 CREATE VIEW
[meet19=> SELECT * FROM Sixth_view;
    departmentname    | room_type | roomnum | availability
----------------------+-----------+---------+--------------
 Emergency Department | ER        |     220 | Yes
 Emergency Department | ER        |     310 | No
(2 rows)
```

F. Indexes:

1.

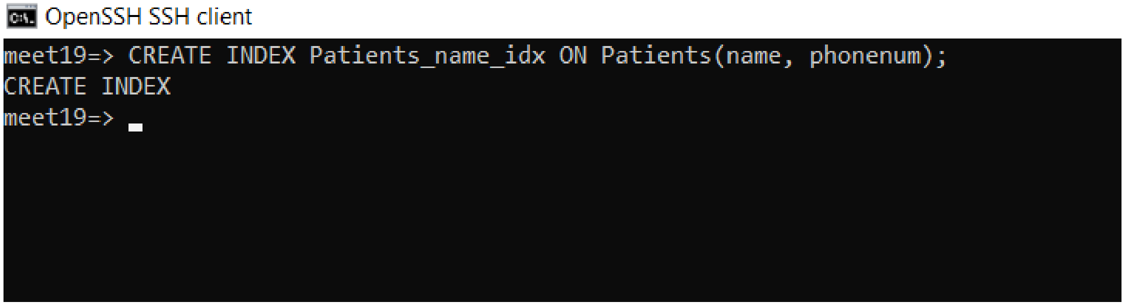


```
meet19=> CREATE INDEX Doctors_name_idx ON Doctors(name);
CREATE INDEX
meet19=> _
```

English description:

This index will help user to find the doctors name when they use the query to find the doctor’s name. whenever the user tries to something with the doctor’s name in the query, this index will help the user to get the result very quickly.

2.

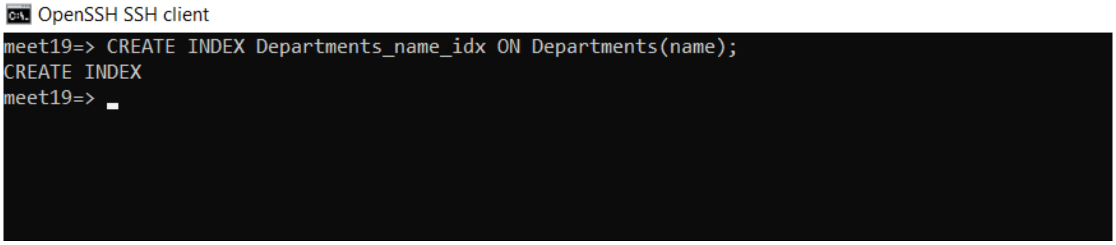


```
meet19=> CREATE INDEX Patients_name_idx ON Patients(name, phonenum);
CREATE INDEX
meet19=>
```

English description:

This index will help the user to find the patients name and phone number. When the user tries to find anything about the patients’ name or phone number, this index will help the user to find the result very quickly.

3.



```
meet19=> CREATE INDEX Departments_name_idx ON Departments(name);
CREATE INDEX
meet19=>
```

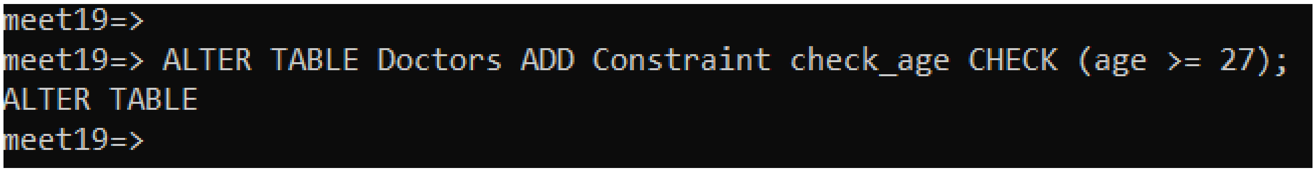
English description:

This index will help the user to find the departments name when the user tries to find anything related to department’ name, it will help the query to run quickly, and get the result.

G. CONSTRAINT:

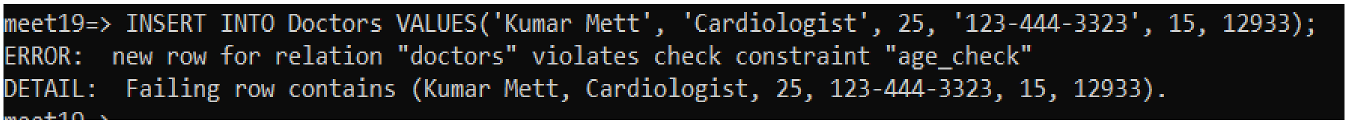
1.

Created the constraint for the doctors whose age is grater than equals to 27.



```
meet19=>
meet19=> ALTER TABLE Doctors ADD Constraint check_age CHECK (age >= 27);
ALTER TABLE
meet19=>
```

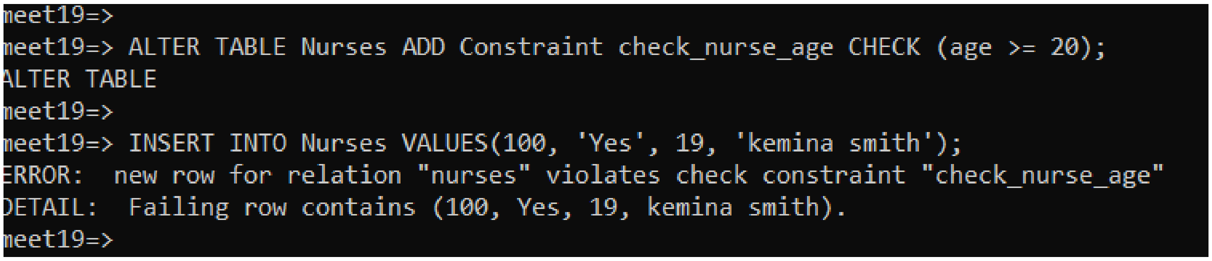
If the doctors age is below 27 or if the user accidentally inputs the age below 27, the database will give the error massage.



```
meet19=> INSERT INTO Doctors VALUES('Kumar Mett', 'Cardiologist', 25, '123-444-3323', 15, 12933);
ERROR:  new row for relation "doctors" violates check constraint "age_check"
DETAIL:  Failing row contains (Kumar Mett, Cardiologist, 25, 123-444-3323, 15, 12933).
```

2.

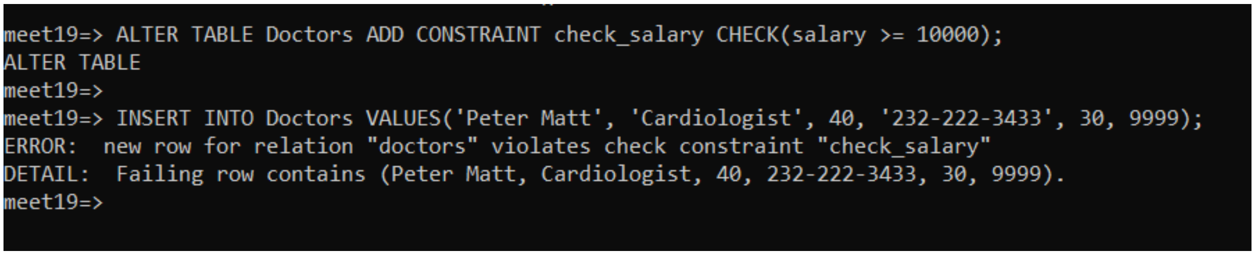
Here, we created the constraint for the nurses to have the age of 20 or above, it the user accedently input the age below 20, it will give the error message, because the hospital only hire the nurses whose age is 20 or above.



```
meet19=>
meet19=> ALTER TABLE Nurses ADD Constraint check_nurse_age CHECK (age >= 20);
ALTER TABLE
meet19=>
meet19=> INSERT INTO Nurses VALUES(100, 'Yes', 19, 'kemina smith');
ERROR:  new row for relation "nurses" violates check constraint "check_nurse_age"
DETAIL:  Failing row contains (100, Yes, 19, kemina smith).
meet19=>
```

3.

This constraint will check if the doctor's salary is below 10000, it will give the error. The minimum starting monthly salary will be 10000 for a doctor.



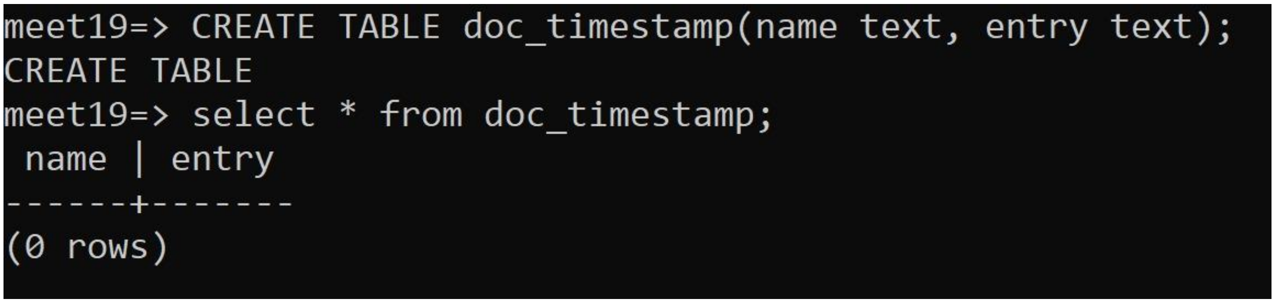
```
meet19=> ALTER TABLE Doctors ADD CONSTRAINT check_salary CHECK(salary >= 10000);
ALTER TABLE
meet19=>
meet19=> INSERT INTO Doctors VALUES('Peter Matt', 'Cardiologist', 40, '232-222-3433', 30, 9999);
ERROR:  new row for relation "doctors" violates check constraint "check_salary"
DETAIL:  Failing row contains (Peter Matt, Cardiologist, 40, 232-222-3433, 30, 9999).
meet19=>
```

H. Triggers:

1.

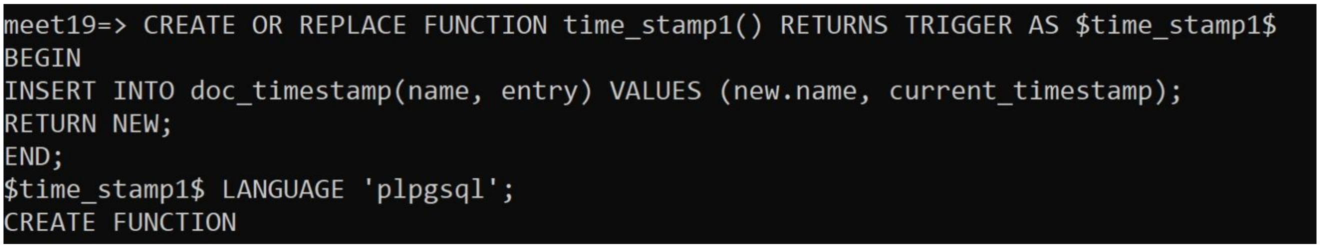
First,

we created the table of doc_timestamp for the trigger.



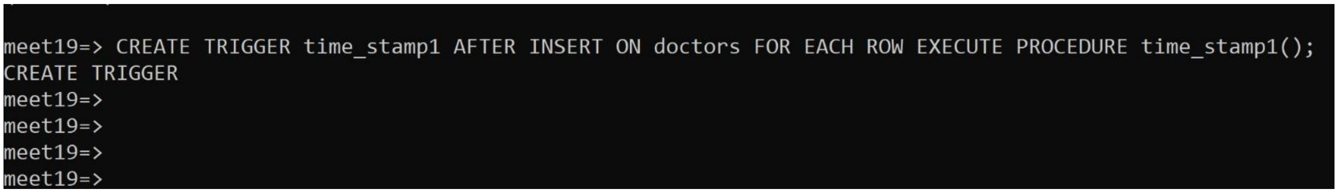
```
meet19=> CREATE TABLE doc_timestamp(name text, entry text);
CREATE TABLE
meet19=> select * from doc_timestamp;
 name | entry
------+-------
(0 rows)
```

The we created the function to help the trigger.



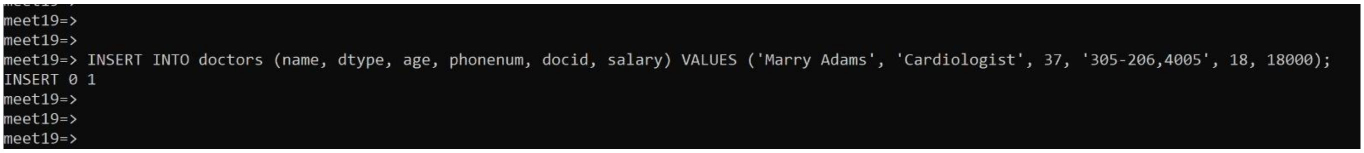
```
meet19=> CREATE OR REPLACE FUNCTION time_stamp1() RETURNS TRIGGER AS $time_stamp1$
BEGIN
INSERT INTO doc_timestamp(name, entry) VALUES (new.name, current_timestamp);
RETURN NEW;
END;
$time_stamp1$ LANGUAGE 'plpgsql';
CREATE FUNCTION
```

We created the trigger after the function.



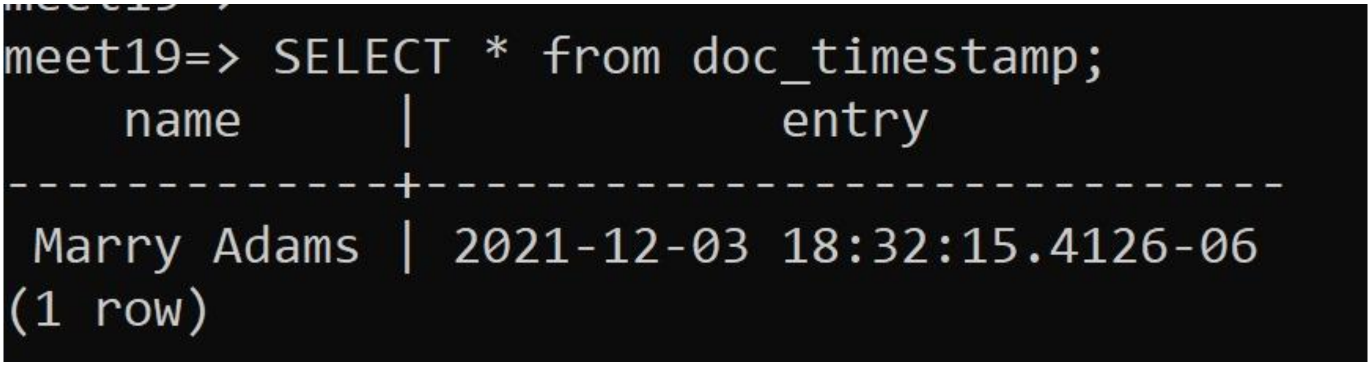
```
meet19=> CREATE TRIGGER time_stamp1 AFTER INSERT ON doctors FOR EACH ROW EXECUTE PROCEDURE time_stamp1();
CREATE TRIGGER
meet19=>
meet19=>
meet19=>
meet19=>
```

We instered a tuple into doctors table.



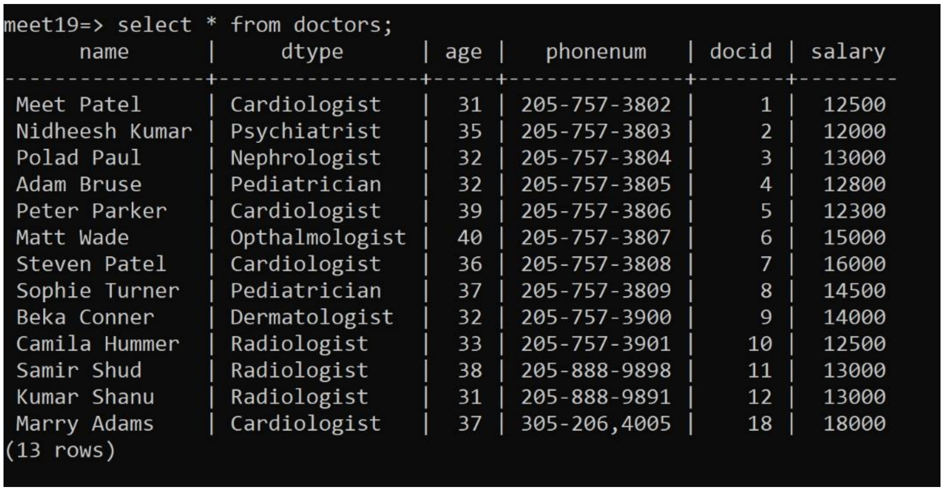
```
meet19=>
meet19=>
meet19=> INSERT INTO doctors (name, dtype, age, phonenum, docid, salary) VALUES ('Marry Adams', 'Cardiologist', 37, '305-206,4005', 18, 18000);
INSERT 0 1
meet19=>
meet19=>
meet19=>
```

The result screenshot of the doc_timestamp after inserting the tuple into doctors table using the trigger function



```
meet19=> SELECT * from doc_timestamp;
    name     |            entry
-------------+-----------------------------
 Marry Adams | 2021-12-03 18:32:15.4126-06
(1 row)
```

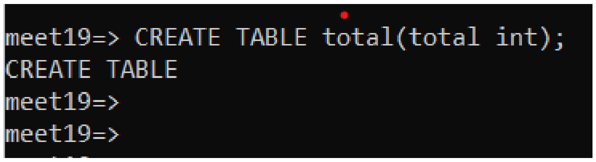
The result doctors table after the insertion of tuple using the trigger:



```
meet19=> select * from doctors;
      name      |      dtype      | age |   phonenum   | docid | salary
----------------+-----------------+-----+--------------+-------+--------
 Meet Patel     | Cardiologist    |  31 | 205-757-3802 |     1 |  12500
 Nidheesh Kumar | Psychiatrist    |  35 | 205-757-3803 |     2 |  12000
 Polad Paul     | Nephrologist    |  32 | 205-757-3804 |     3 |  13000
 Adam Bruse     | Pediatrician    |  32 | 205-757-3805 |     4 |  12800
 Peter Parker   | Cardiologist    |  39 | 205-757-3806 |     5 |  12300
 Matt Wade      | Opthalmologist  |  40 | 205-757-3807 |     6 |  15000
 Steven Patel   | Cardiologist    |  36 | 205-757-3808 |     7 |  16000
 Sophie Turner  | Pediatrician    |  37 | 205-757-3809 |     8 |  14500
 Beka Conner    | Dermatologist   |  32 | 205-757-3900 |     9 |  14000
 Camila Hummer  | Radiologist     |  33 | 205-757-3901 |    10 |  12500
 Samir Shud     | Radiologist     |  38 | 205-888-9898 |    11 |  13000
 Kumar Shanu    | Radiologist     |  31 | 205-888-9891 |    12 |  13000
 Marry Adams    | Cardiologist    |  37 | 305-206,4005 |    18 |  18000
(13 rows)
```

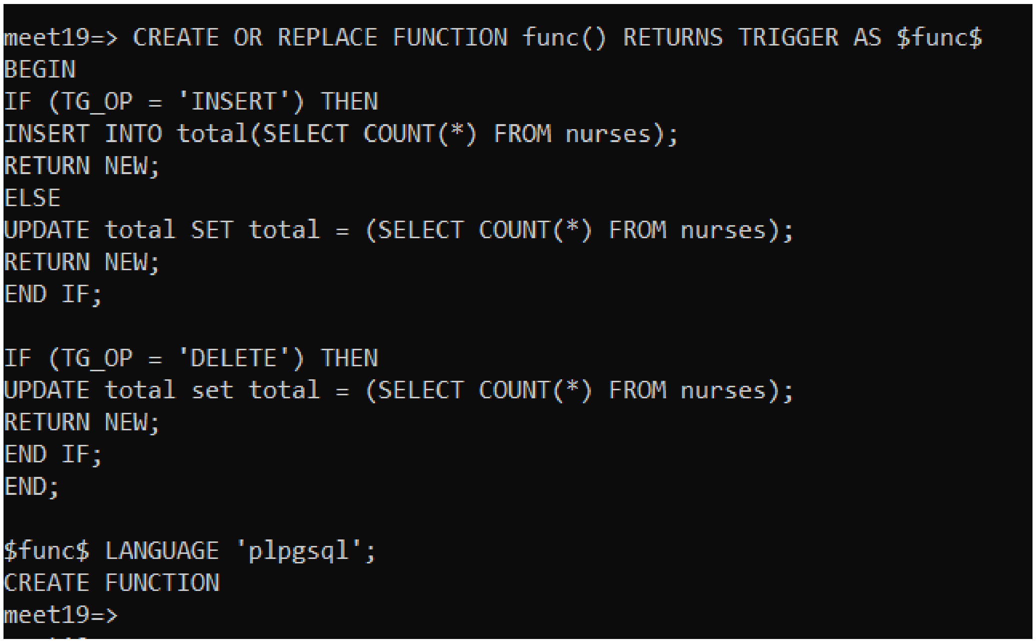
2.

First, we created a table to trigger it to find the total number of nurses in departments.



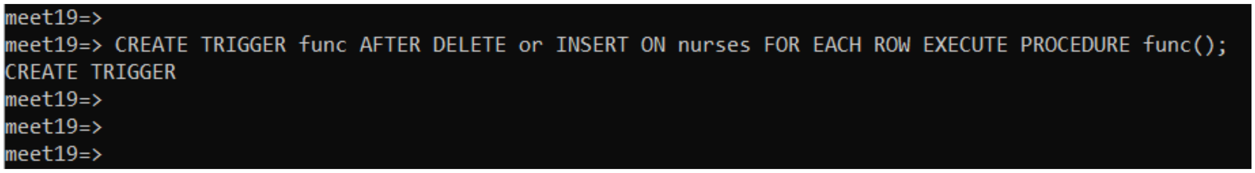
```
meet19=> CREATE TABLE total(total int);
CREATE TABLE
meet19=>
meet19=>
```

Then, we created the functions to help the trigger to perform according to the logic.



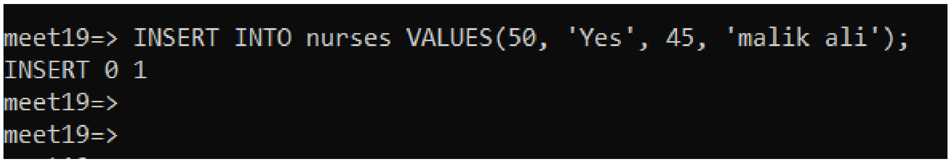
```
meet19=> CREATE OR REPLACE FUNCTION func() RETURNS TRIGGER AS $func$
BEGIN
IF (TG_OP = 'INSERT') THEN
INSERT INTO total(SELECT COUNT(*) FROM nurses);
RETURN NEW;
ELSE
UPDATE total SET total = (SELECT COUNT(*) FROM nurses);
RETURN NEW;
END IF;

IF (TG_OP = 'DELETE') THEN
UPDATE total set total = (SELECT COUNT(*) FROM nurses);
RETURN NEW;
END IF;
END;

$func$ LANGUAGE 'plpgsql';
CREATE FUNCTION
meet19=>
```

Then, we created the trigger based on the function.



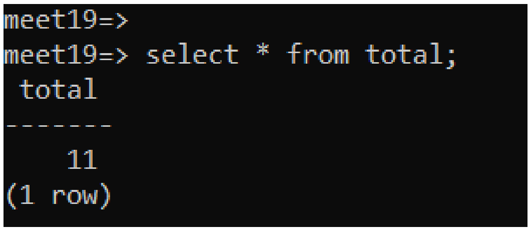
```
meet19=>
meet19=> CREATE TRIGGER func AFTER DELETE or INSERT ON nurses FOR EACH ROW EXECUTE PROCEDURE func();
CREATE TRIGGER
meet19=>
meet19=>
meet19=>
```

Then, we added the tuple to nurses table to check weather the trigger is working or not.



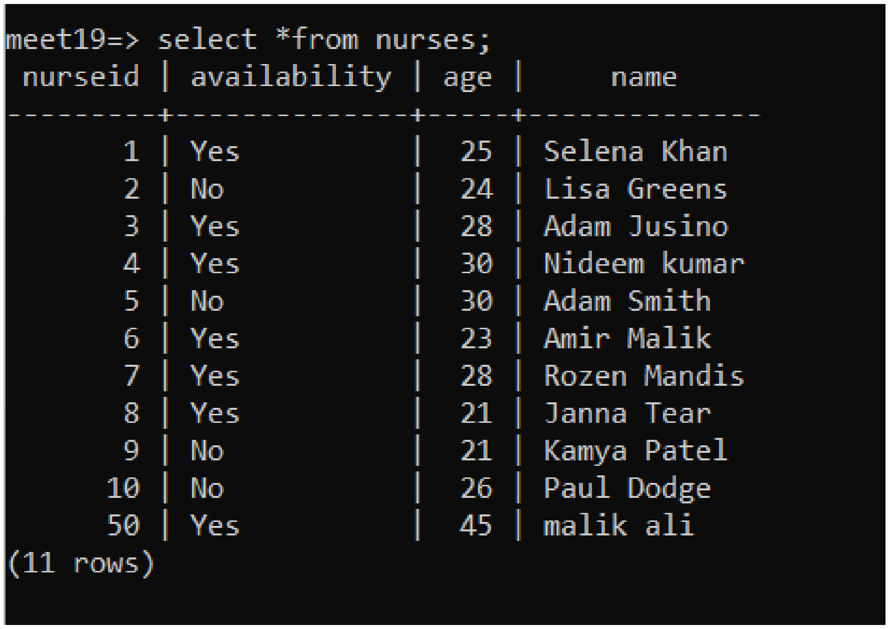
```
meet19=> INSERT INTO nurses VALUES(50, 'Yes', 45, 'malik ali');
INSERT 0 1
meet19=>
meet19=>
```

Trigger works, and the total number went 10 to 11 since we added one tuple.



```
meet19=>
meet19=> select * from total;
 total
-------
    11
(1 row)
```

Here is the last tuple in the table we added to double check it.



```
meet19=> select *from nurses;
 nurseid | availability | age |     name
---------+--------------+-----+--------------
       1 | Yes          |  25 | Selena Khan
       2 | No           |  24 | Lisa Greens
       3 | Yes          |  28 | Adam Jusino
       4 | Yes          |  30 | Nideem kumar
       5 | No           |  30 | Adam Smith
       6 | Yes          |  23 | Amir Malik
       7 | Yes          |  28 | Rozen Mandis
       8 | Yes          |  21 | Janna Tear
       9 | No           |  21 | Kamya Patel
      10 | No           |  26 | Paul Dodge
      50 | Yes          |  45 | malik ali
(11 rows)
```

We declare that we have completed this assignment completely and entirely on our own, without any consultation with others. We have read the UAB Academic Honor Code and understand that any breach of the Honor Code may result in severe penalties.
We also declare that the following percentage distribution faithfully represents individual group members' contributions to the completion of the assignment

Name	Overall Contribution (%)	Major work items completed by me	Signature or initials	Date
Meet Patel	25 %	E-R diagram, Relational Schema, Sample Data, Create Views, Indexes, Constraints	Meet Patel	12/03/2021
Adam Jusino	25 %	Assumptions, Background, Triggers, Sample Data, E-R diagram	Adam Jusino	12/03/2021
Nidheesh Kumar	25 %	Triggers, Sample Data, E-R diagram, Create Views,	Nidheesh Kadem	12/03/2021
Polad Yunisov	25 %	Constraints, Create Views, Tables, E-R diagram, Sample Data	Polad Yunisov	12/03/2021