DBMS PROJECT – PHARMACY MANAGEMENT SYSTEM

NAME: MAHAH SADIQUE
SRN: PES1201801529
SECTION : J

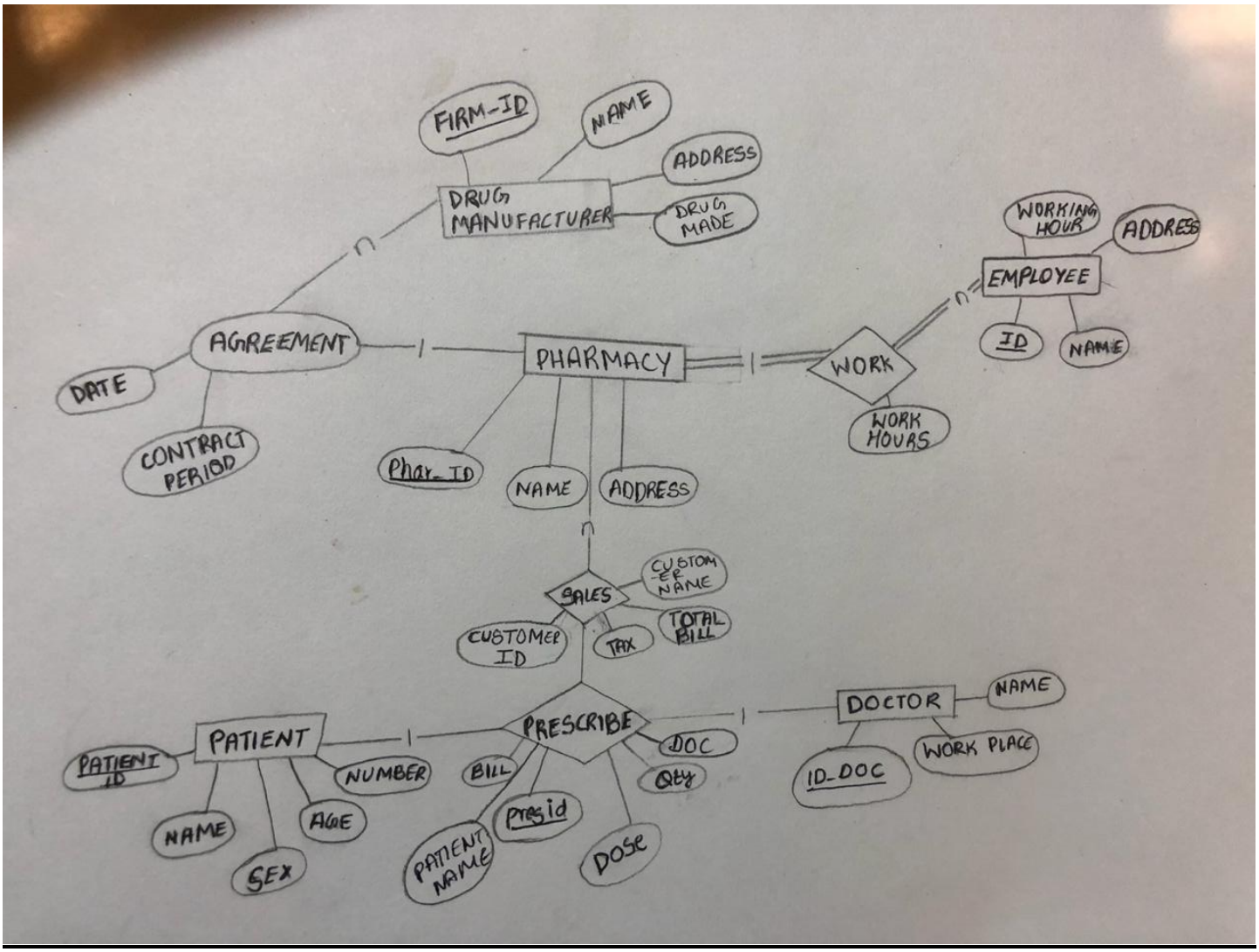
MINIWORLD : PHARMACY

A) INTRODUCTION

The miniworld that I have selected is healthcare , more specifically , the database management system of a pharmacy

The ER diagram consists of basic components of any pharmaceutical store - the pharmacy itself , the drug manufacturer/supplier, customer (here I've only included customers who purchase medicines with a prescription) , hence labelled as patients, the doctors that examine the patient , the prescription written by the doctor , an employee record , a work record specifying work hours for employee and an employee_ change table for any employee data that is updated.

E R DIAGRAM



SCHEMA NAME – pharmacy

1)Pharma :

Attributes include pharmacy ID, pharmacy name , and address , where 'phar_ID' is the primary key

2) Patient :

Attributes include patient name, age , sex, patient ID and contact number where 'idpatient' is the primary key

3) Doctor:

Attributes include doctor ID (primary key), name and work place (name of hospital and clinic they work at)

4)Drug manufacturer:

Attributes include ID(primary key), name, name of drug being produced, name of manufacturing company, address

5) Agreement:

Acts as a contract between the manufacturing company and pharmacy , Attributes include date of start of agreement, period of the contract (in years), and foreign keys referring to the IDs of 'pharma' and 'drug_manuf'

6) Employee:

Contains details of the pharmacy employee, Attributes include , employee name , address and ID

7)employee_changes:

Contains information regarding time of update, action taken and the employee data that was replaced (the old data)

8)Prescription:

Attributes include , dosage of medicine (in terms of times in a day),Qty, bill amount, name of doctor and patient , prescription ID(primary key), foreign key referring to patient ID and doctor ID

9) Work :

Attributes include work hours and foreign keys referring to pharma and employee IDs.

10)Sale:

Attributes like customer name and ID , tax amount , total bill , and foreign key referring to prescription id

SCHEMA DIAGRAM

B) FUNCTIONAL DEPENDENCIES:

(under respective table names)

1) PATIENT : idpatient → name , sex , age
2) DOCTOR : idDoctor → doc name , work place
3) PHARMACY : phar_ID →name , address
4) EMPLOYEE: emp_ID → name , address
5) PRESCRIPTION : idpres → Qty, dose
6) WORK : [emp_ID , phar _ID] → work hours

KEYS:

Primary keys :

1) idpatient
2) idDoctor
3) phar_ID
4) emp_ID
5) idpres

C) NORMALISATION :

1) **1N FORM :** While creating the tables, no non atomic values were inserted , as every value was atomic and indivisible , the database was already in the first normal form .
2) **2N FORM :** Now since it was already in first normal form , next aim was to remove all partial dependencies , initially EMPLOYEE was under PHARMACY , but since certain employee attributes , like employee address, did not depend on phar_ID , this was a partial dependency , hence the table was broken up to form a new EMPLOYEE table , with attributes such as employee ID, name , address.

3) **3N FORM :** While creating the table it was made sure that no transitive dependencies were present , where the attributes did not depend on the primary key . Although this could be violated for example : under EMPLOYEE , if we were to add a REPORTING TIME attribute based on where each employee's location so that those who lived close by had to report earlier and those lived far could report later , this attribute depends on the EMPLOYEE.address attribute and not on the primary key itself .

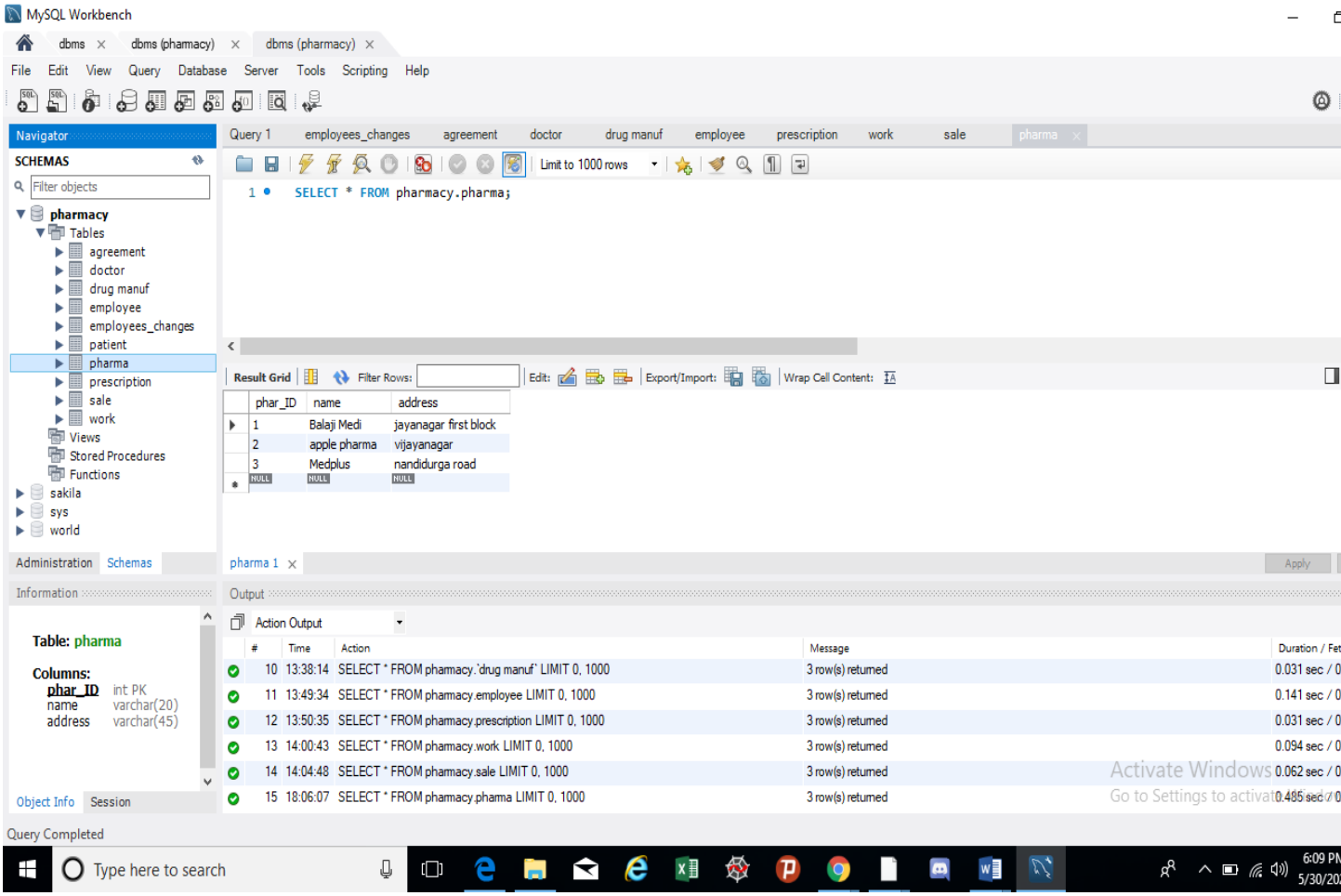
D) TABLE CREATION

Given down below are the scripts for creating the tables along with the output screenshot

All screenshots where taken using snipping tool

1) PHARMA

```
CREATE TABLE `pharmacy`.`pharma` (
  `pharma_ID` INT NOT NULL,
  `name` VARCHAR(20) NOT NULL
  `address` VARCHAR(30) NOT NULL
    PRIMARY KEY(pharma_ID);
```



2) PATIENT

CREATE TABLE `pharmacy`.`patient` (

`idpatient` INT NOT NULL,

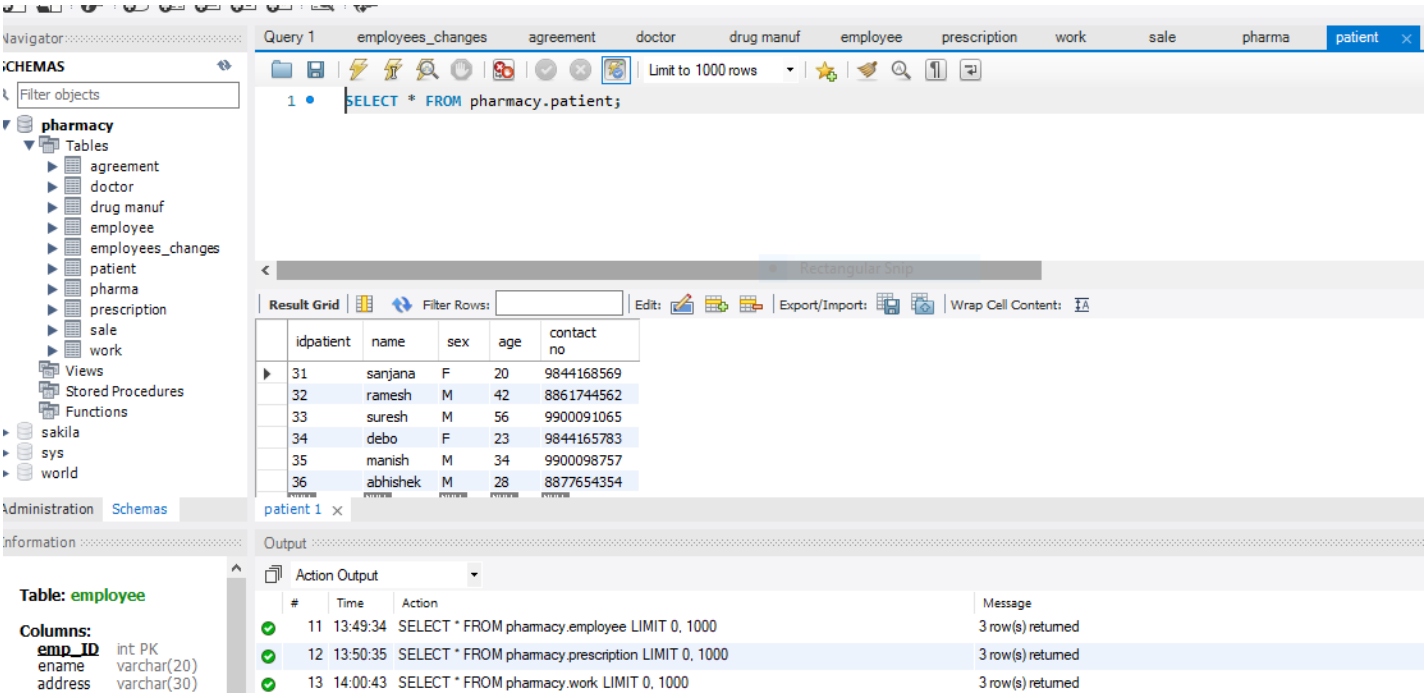
`name` VARCHAR(20) NOT NULL,

`sex` VARCHAR(5) NOT NULL,

'age' INT NOT NULL,

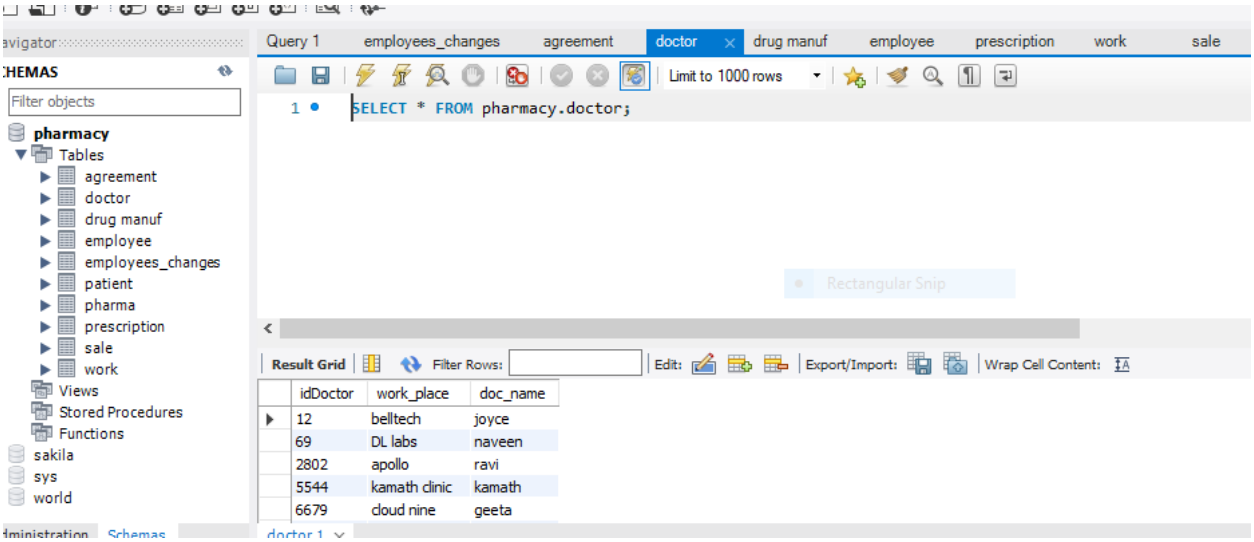
PRIMARY KEY (`idpatient`));

** in the instructions I'm aware that we are supoosed to insert full screenshots , but as you can see the clarity isn't a lot , hence i have no option but to take a rectangular ss for a better image **



3)DOCTOR

```
CREATE TABLE `pharmacy`.`doctor ` (
  `idDoctor` INT NOT NULL,
  `work_place` VARCHAR(20) NOT NULL,
  `doc_name’ VARCHAR(10) NOT NULL,
  PRIMARY KEY (`idDoctor));
```



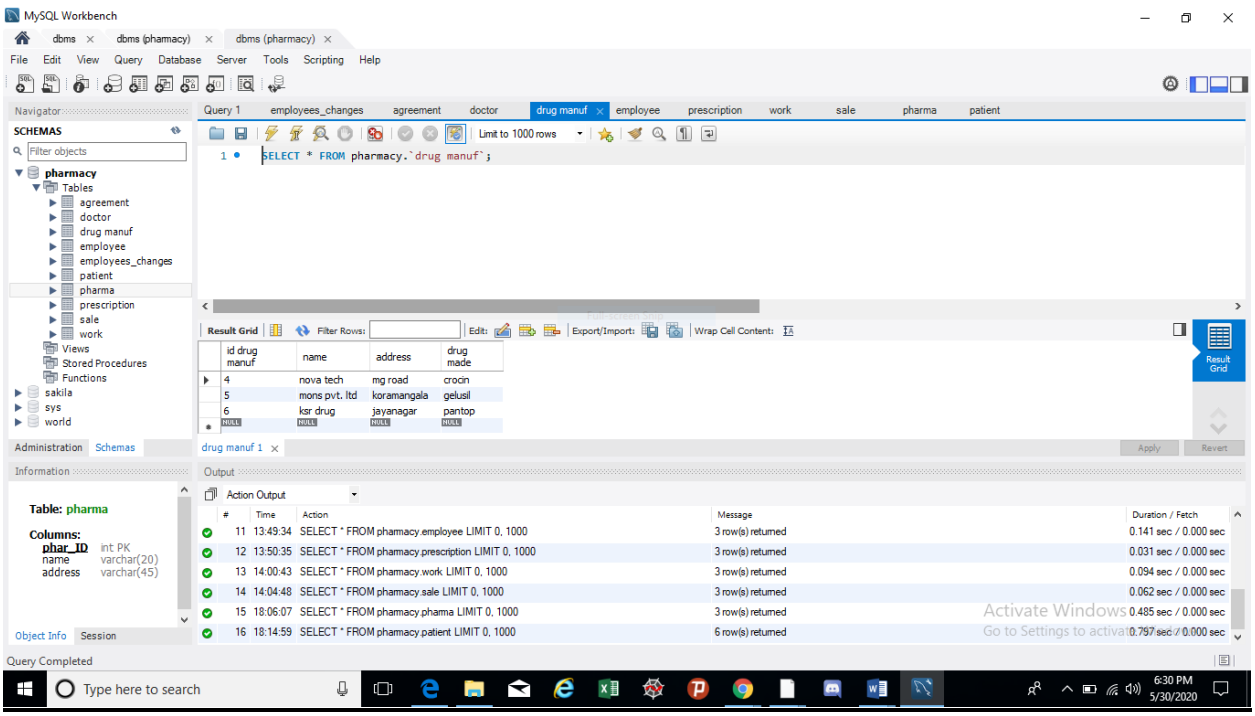
4) **DRUG_MANUF**

CREATE TABLE `pharmacy`.`drug_manuf` (

`iddrug_manuf` INT NOT NULL,

`name` VARCHAR(20) NOT NULL,

5) `address` VARCHAR(30) NOT NULL,
6) `drug made` VARCHAR(10) NOT NULL,
7) PRIMARY KEY (`iddrug_manuf`)),

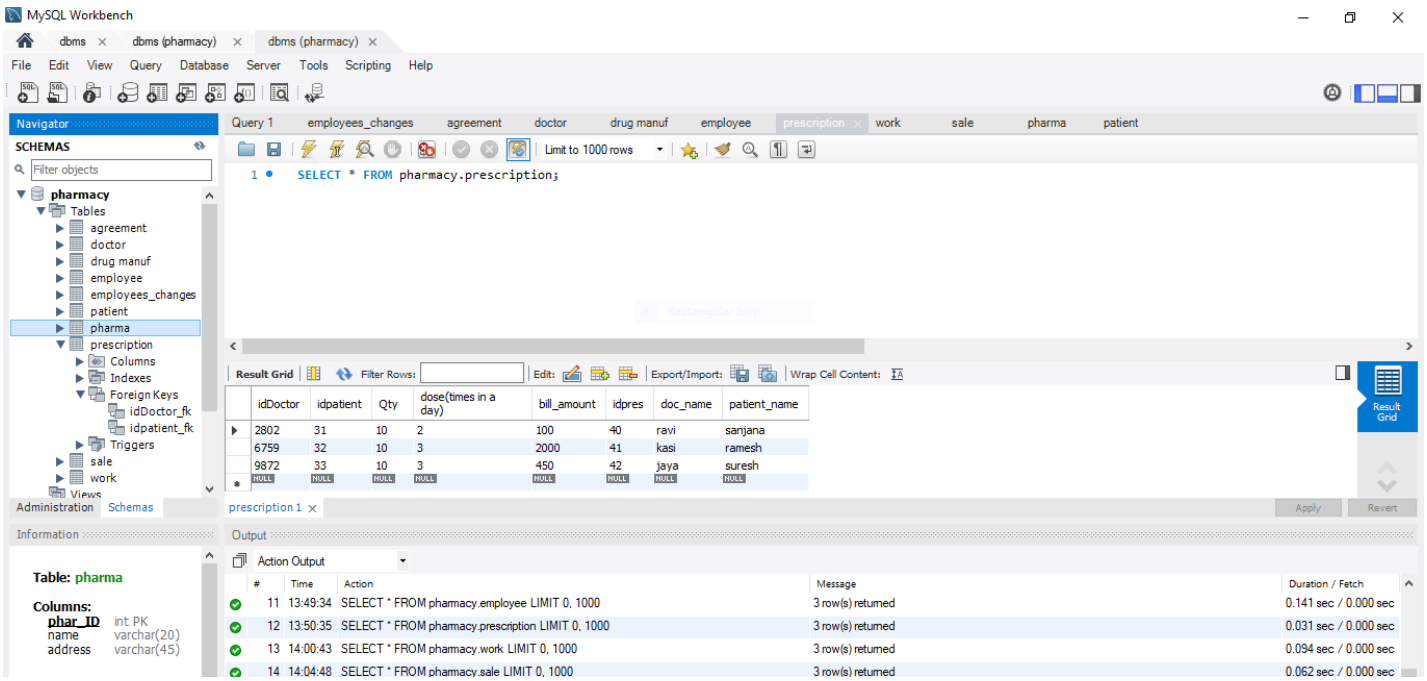


5) PRESCRIPTION

demonstrating use of constraints and creation of foreign keys

```
CREATE TABLE `pharmacy`.`prescription` (
  `idDoctor` INT NOT NULL,
  `idpatient` INT NOT NULL,
  `Qty` VARCHAR(20) NOT NULL,
  `dose(times in a day)` VARCHAR(10) NOT NULL,
  'bill_amount' INT NOT NULL
  'idpres' INT NOT NULL
'doc_name' VARCHAR(20) NOT NULL,
'patient_name' VARCHAR(20) NOT NULL
  PRIMARY KEY (`idDoctor`, `idpatient`, 'idpres'),
  INDEX `idDoctor` (`idDoctor` ASC) INVISIBLE,
  INDEX `idpatient` (`idpatient` ASC) VISIBLE,
  CONSTRAINT `idDoctor_fk`
```

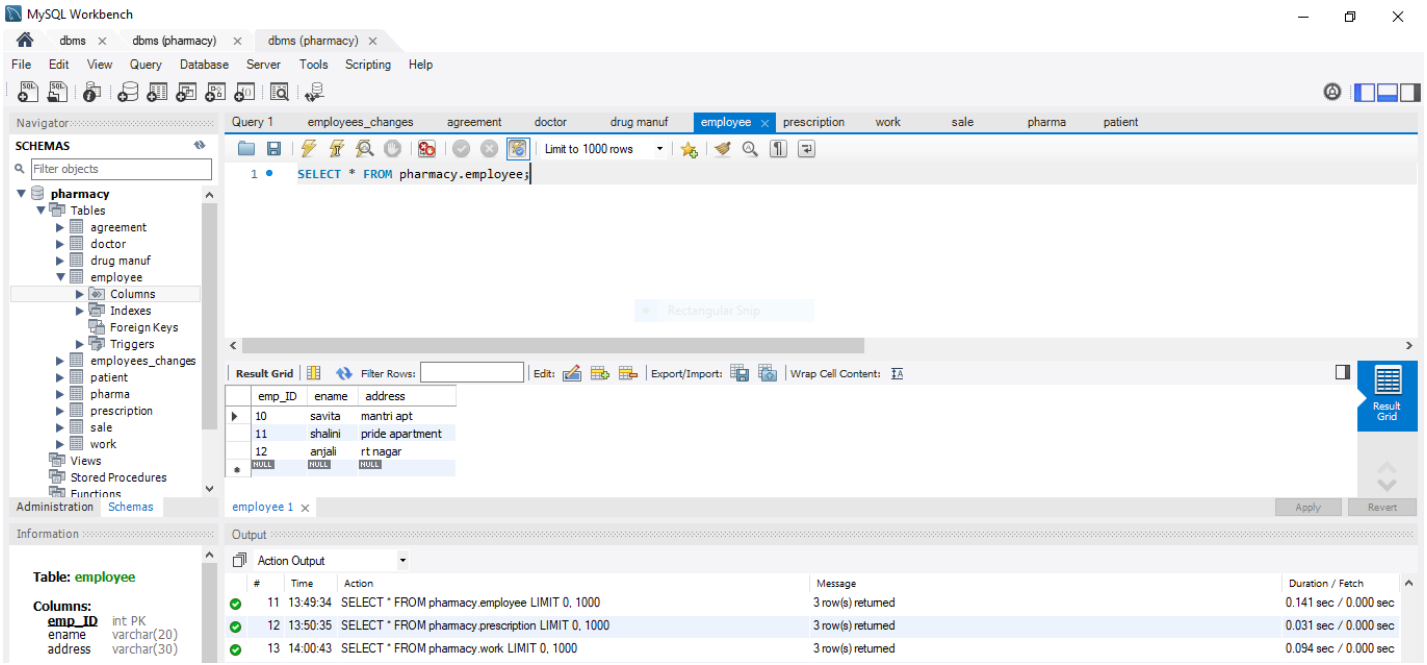
```
  FOREIGN KEY (`idDoctor`)
  REFERENCES `pharmacy`.`doctor` (`idDoctor`)

  ON DELETE CASCADE
  ON UPDATE CASCADE,
CONSTRAINT `idpatient_fk`
  FOREIGN KEY (`idpatient`)
  REFERENCES `pharmacy`.`patient` (`idpatient`)
  ON DELETE CASCADE
  ON UPDATE CASCADE);
```



6) EMPLOYEE

```
CREATE TABLE `pharmacy`.`employee` (
  `emp_ID` INT NOT NULL,
  `ename` VARCHAR(20) NOT NULL,
  `working hours` INT NOT NULL,
```

`address` VARCHAR(30) NOT NULL);



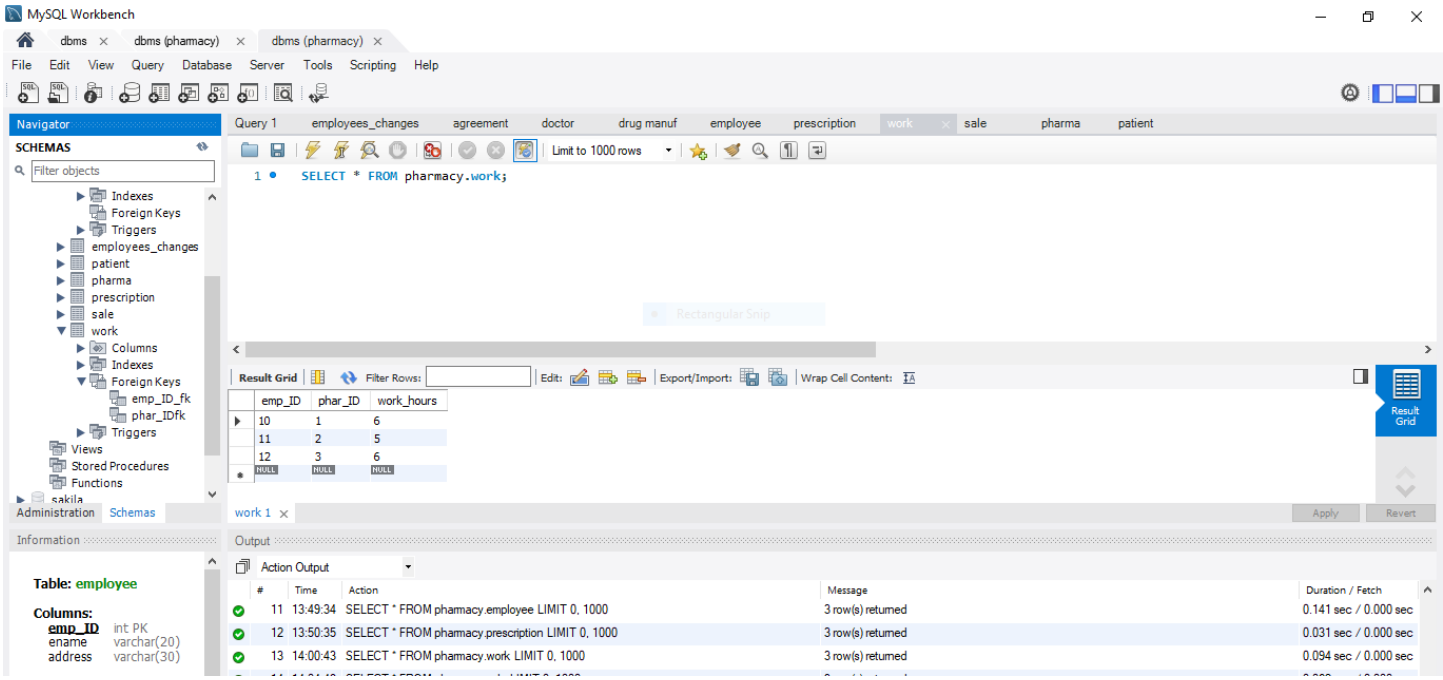
7)WORK

```
CREATE TABLE `pharmacy`.`work` (
  `work_hours` INT NOT NULL,
  `phar_ID` INT NOT NULL,
  `emp_ID` INT NOT NULL,
  PRIMARY KEY (`idpres`));
ALTER TABLE `pharmacy`.`work`
ADD CONSTRAINT `emp_ID_fk`
  FOREIGN KEY (`emp_ID`)
  REFERENCES `pharmacy`.`employee` (`emp_ID`)
  ON DELETE CASCADE
  ON UPDATE CASCADE,
ADD CONSTRAINT `phar_IDfk`
  FOREIGN KEY (`phar_ID`)
```

REFERENCES `pharmacy`.`pharma` (`phar_ID`)

ON DELETE CASCADE

ON UPDATE CASCADE;



8)AGREEMENT

CREATE TABLE `pharmacy`.`agreement` (

`date of agreement` INT NOT NULL,

`contract period` INT NOT NULL,

`id drug manuf` INT NOT NULL,

`phar_ID` INT NOT NULL,

INDEX `phar_ID` (`phar_ID` ASC) VISIBLE,

INDEX `id drug manuf` (`id drug manuf` ASC) VISIBLE,

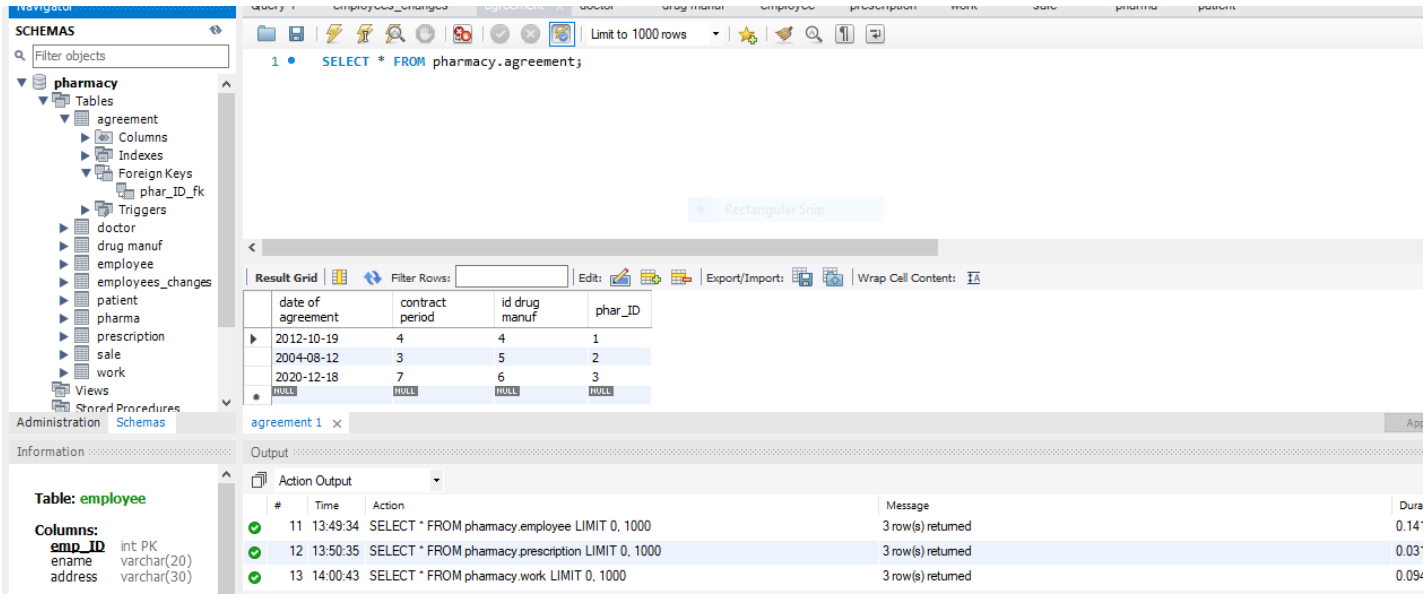
CONSTRAINT `phar_ID_fk`

FOREIGN KEY (`phar_ID`)

REFERENCES `pharmacy`.`pharma` (`phar_ID`)

ON DELETE CASCADE

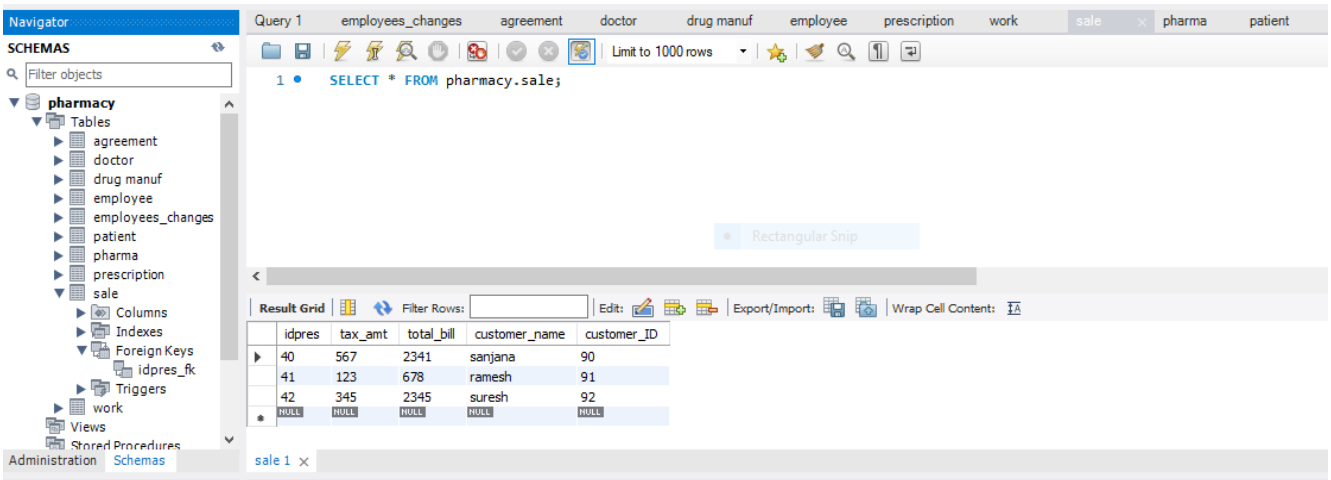
```
  ON UPDATE CASCADE,
CONSTRAINT `id drug manuf_fk`
  FOREIGN KEY (`id drug manuf`)
  REFERENCES `pharmacy`.`drug manuf` (`id drug manuf`)
  ON DELETE CASCADE
  ON UPDATE CASCADE);
```



9)SALES

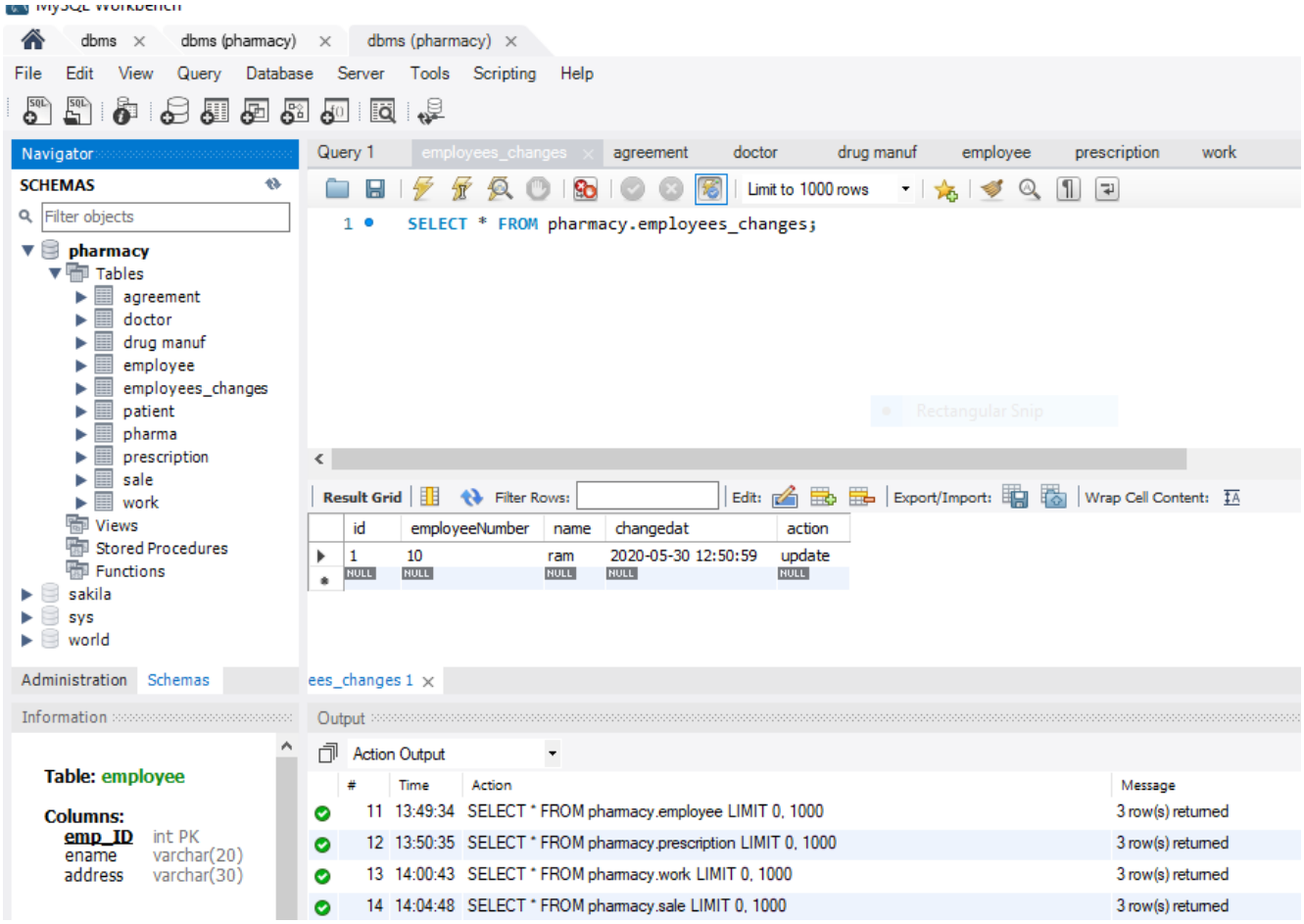
```
CREATE TABLE `pharmacy`.`sale` (
  `idpres` INT NOT NULL,
  `tax amt` INT NOT NULL,
  `total bill` INT NOT NULL,
   'customer_name' VARCHAR(20) NOT NULL,
    'customer_ID' INT NOT NULL,
  PRIMARY KEY (`idpres`));
```

```
ALTER TABLE `pharmacy`.`sale`
ADD INDEX `idpres` (`idpres` ASC) VISIBLE;
;

ALTER TABLE `pharmacy`.`sale`
ADD CONSTRAINT `idpres_fk`
  FOREIGN KEY (`idpres`)
  REFERENCES `pharmacy`.`sale` (`idpres`)
  ON DELETE CASCADE
  ON UPDATE CASCADE;
```



10) employee_changes

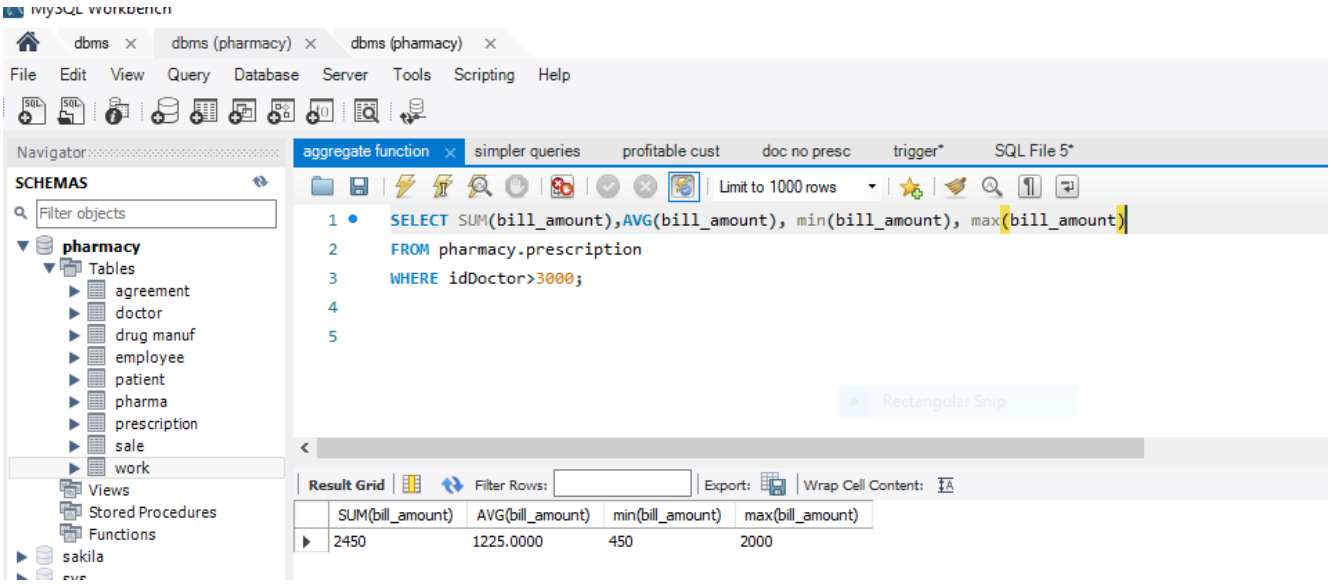
```
CREATE TABLE pharmacy.employee_changes (
    id INT AUTO_INCREMENT PRIMARY KEY,
    'employeeNumber' INT NOT NULL,
    'Name' VARCHAR(20) NOT NULL,
    'changedat' DATETIME DEFAULT NULL,
   ' action' VARCHAR(20) DEFAULT NULL
);
```



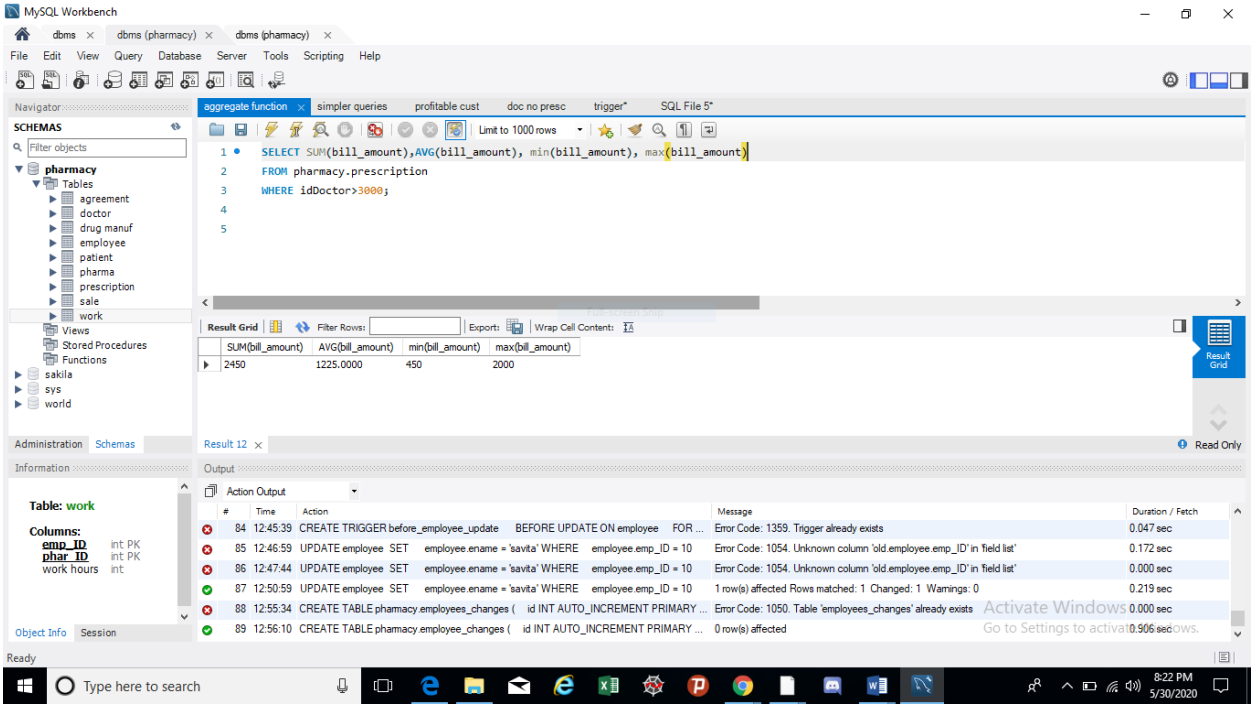
E) QUERIES

1) Created query implementing aggregate function
   ➔ This query returned the SUM , AVG , MIN & MAX of the bill amounts paid where the condition was idDoctor should be greater than 3000

```
SELECT SUM(bill_amount),AVG(bill_amount), min(bill_amount), max(bill_amount)

FROM pharmacy.prescription

WHERE idDoctor>3000;
```



Full screenshot (values are very grainy hence all results are showed in cropped screenshots) :

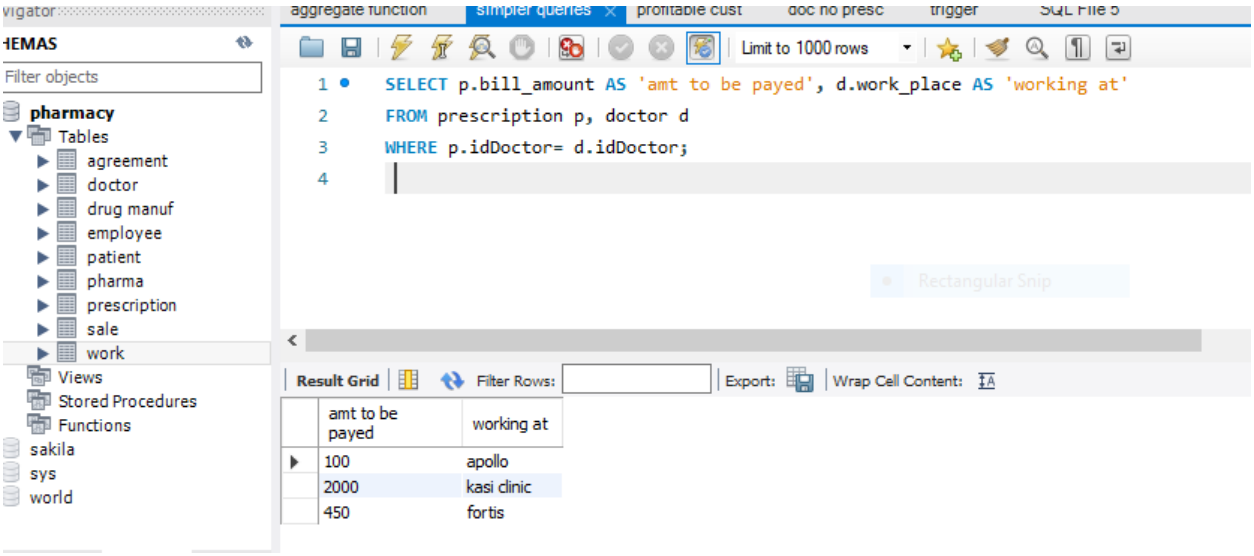


2) <u>Query extracting data from a table as a result of two tables</u>

➔ This query displays bill amount and the work place of the doctor who wrote that very prescription

SELECT p.bill_amount AS 'amt to be payed', d.work_place AS 'working at'

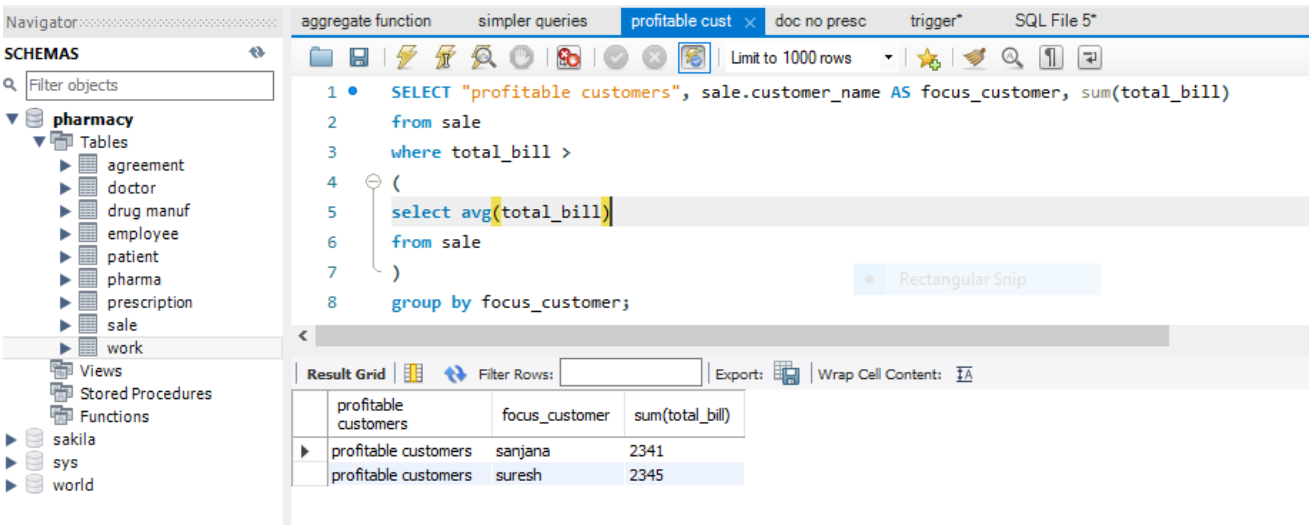
FROM prescription p, doctor d
WHERE p.idDoctor= d.idDoctor;



3) <u>Nested query</u>

➔ This query is used to show which customers are proving to be the high spenders , where their bill amount is greater than the average of all bill amounts , the customers displayed can then be taken as an audience for target marketing

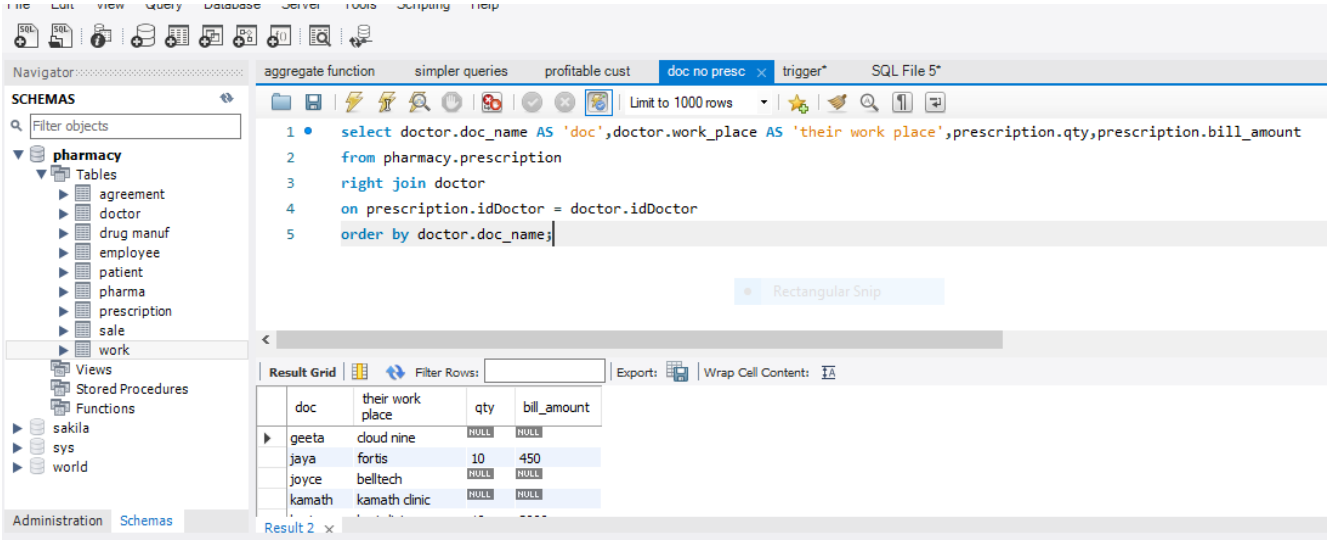
SELECT "profitable customers", sale.customer_name AS focus_customer, sum(total_bill)
from sale
where total_bill >
(
select avg(total_bill)
from sale
)
group by focus_customer;

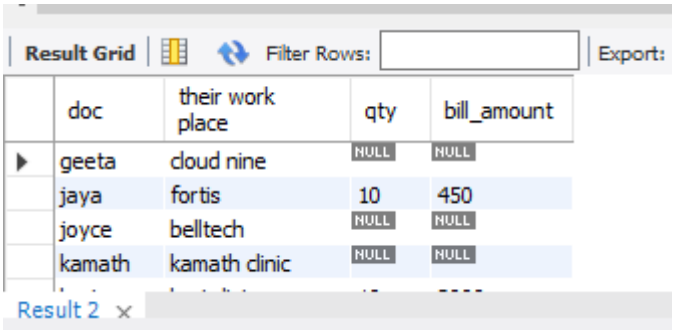


4) Nested query #2 implementing

➔ This query shows which doctors have written prescriptions and those who have NOT can be identified from the presence of NULL values under qty and bill_amount

```
select doctor.doc_name AS 'doc',doctor.work_place AS 'their work place',prescription.qty,prescription.bill_amount
from pharmacy.prescription
right join doctor
on prescription.idDoctor = doctor.idDoctor
order by doctor.doc_name;
```





F) TRIGGER

→This trigger is used to show us exactly what updates on the employee table has taken place , everytime a name is updated/changed in the employee table , then the old information corresponding to that particular employee ID is displayed , along with the date and time at which it occurred , the action taken is also mentioned .

```
CREATE TRIGGER before_employee_update
    BEFORE UPDATE ON employee
    FOR EACH ROW
 INSERT INTO employees_changes
 SET action = 'update',
     employeeNumber = OLD.emp_ID,
     name = OLD.ename,
     changedat = NOW();

     UPDATE employee
```

**To test the trigger *

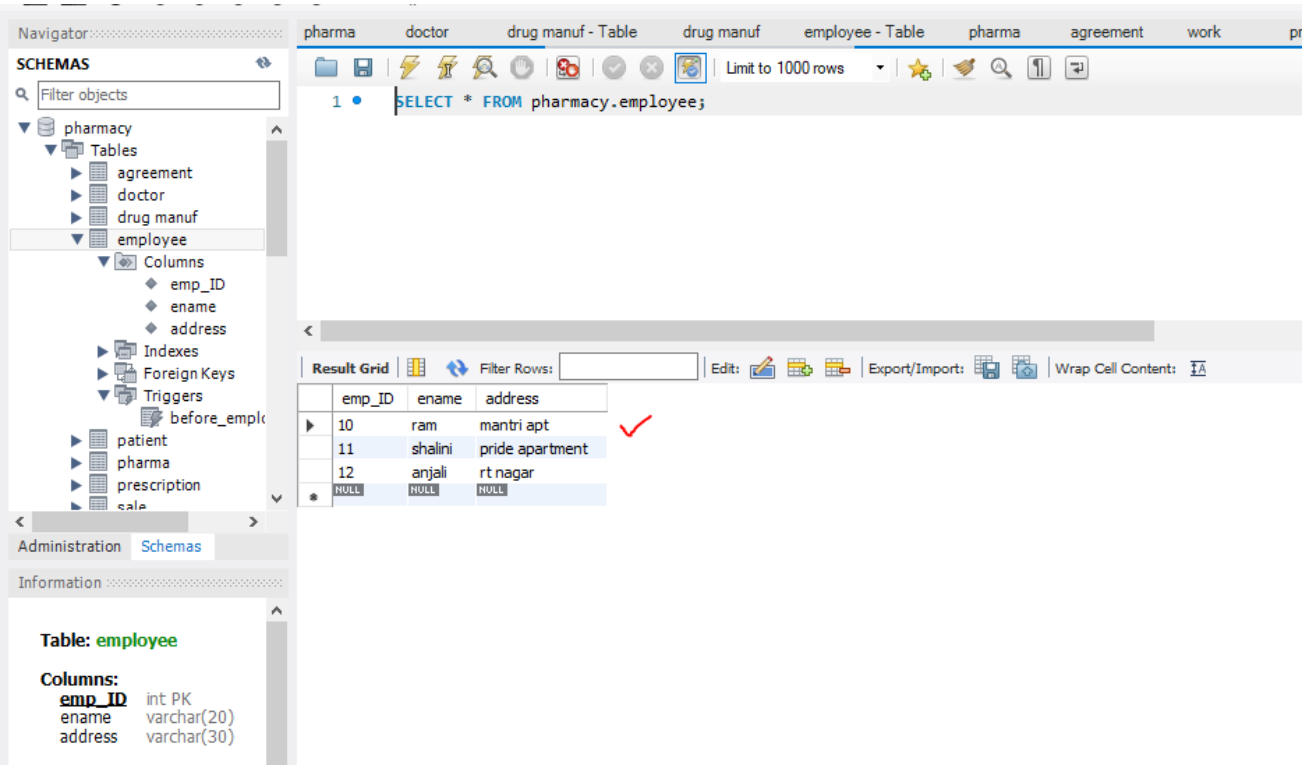
```
SET
    employee.ename = 'savita'
```

WHERE

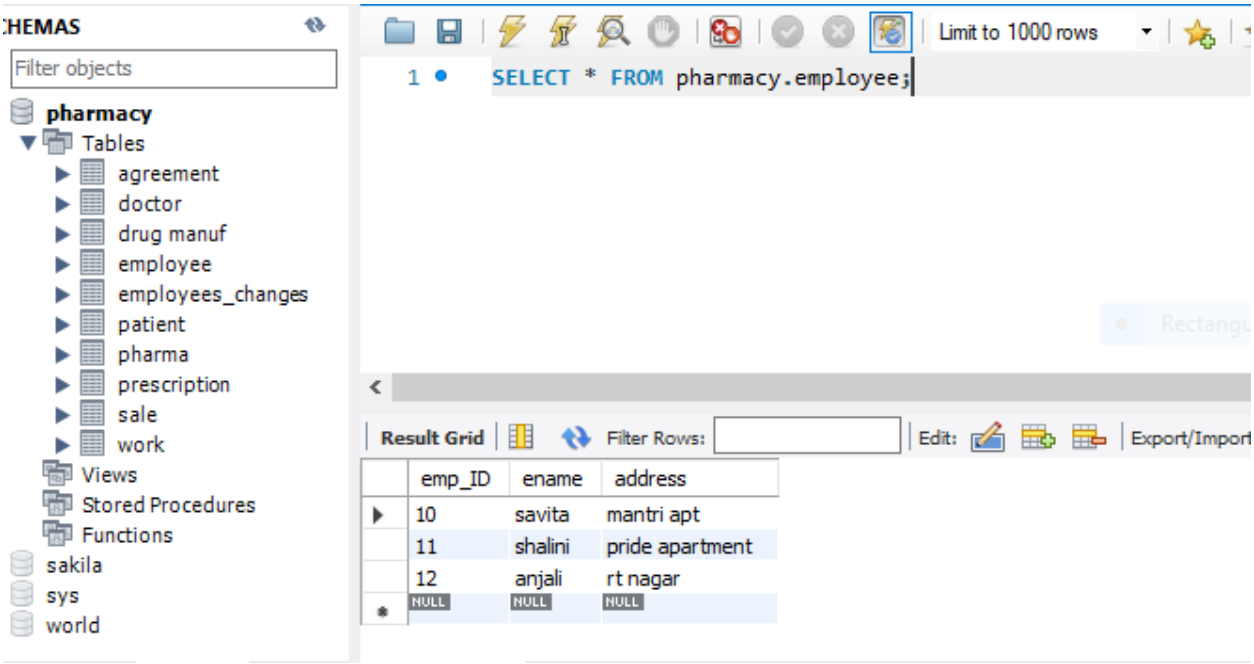
employee.emp_ID = 10;

employee name was updated from ram to savita , old name ram highlighted with red tick.

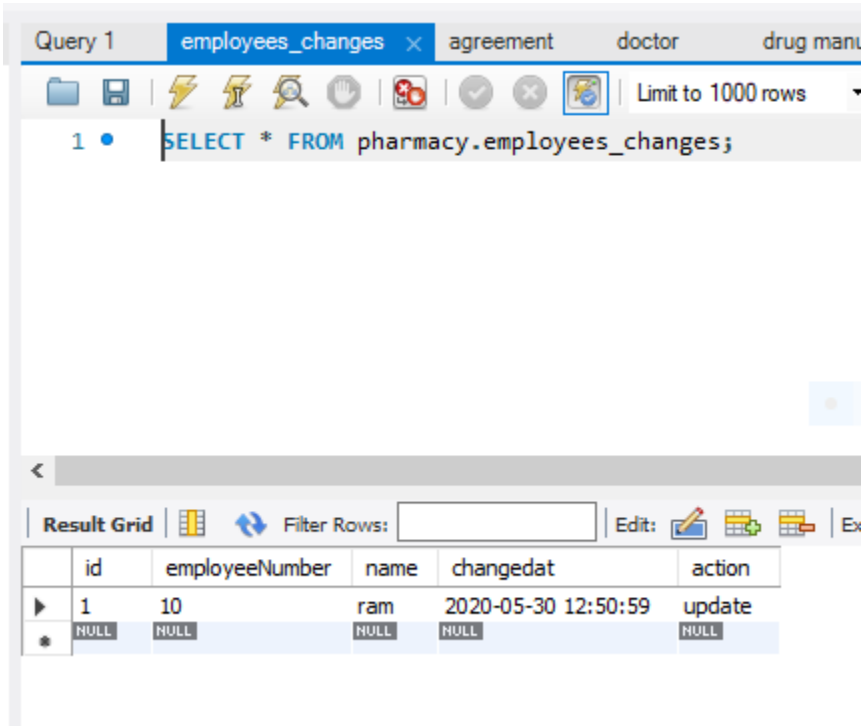
BEFORE UDDATE



AFTER UPDATE



EMPLOYEE_CHANGES , containing old employee data:



G) TEST FOR LOSSLESS JOIN PROPERTY :

Lossless join occurs when a relation is decomposed and when the divided relations would get joint , no data would be lost

Take for example the sales table containing the following data :

Id_pres	Tax_amount	Total_bill	Customer_ID	Customer_name
12	120	345	23	Lilly
13	200	509	24	Stewart
14	250	650	25	Raj

Lets split it into 2 tables:

BILL

Id_pres	Tax_amount	Total_bill	Customer_ID
12	120	345	23
13	200	509	24
14	250	650	25

CUSTOMER

Customer_ID	Customer_name
23	Lilly
24	Stewart
25	Raj

If we perform a natural join on these two tables , we get the original table with no loss in information . Therefore we can say it's a lossless join .