

## **DBMS PROJECT – PHARMACY MANAGEMENT SYSTEM**

NAME: MAHAH SADIQUE

SRN: PES1201801529

SECTION : J

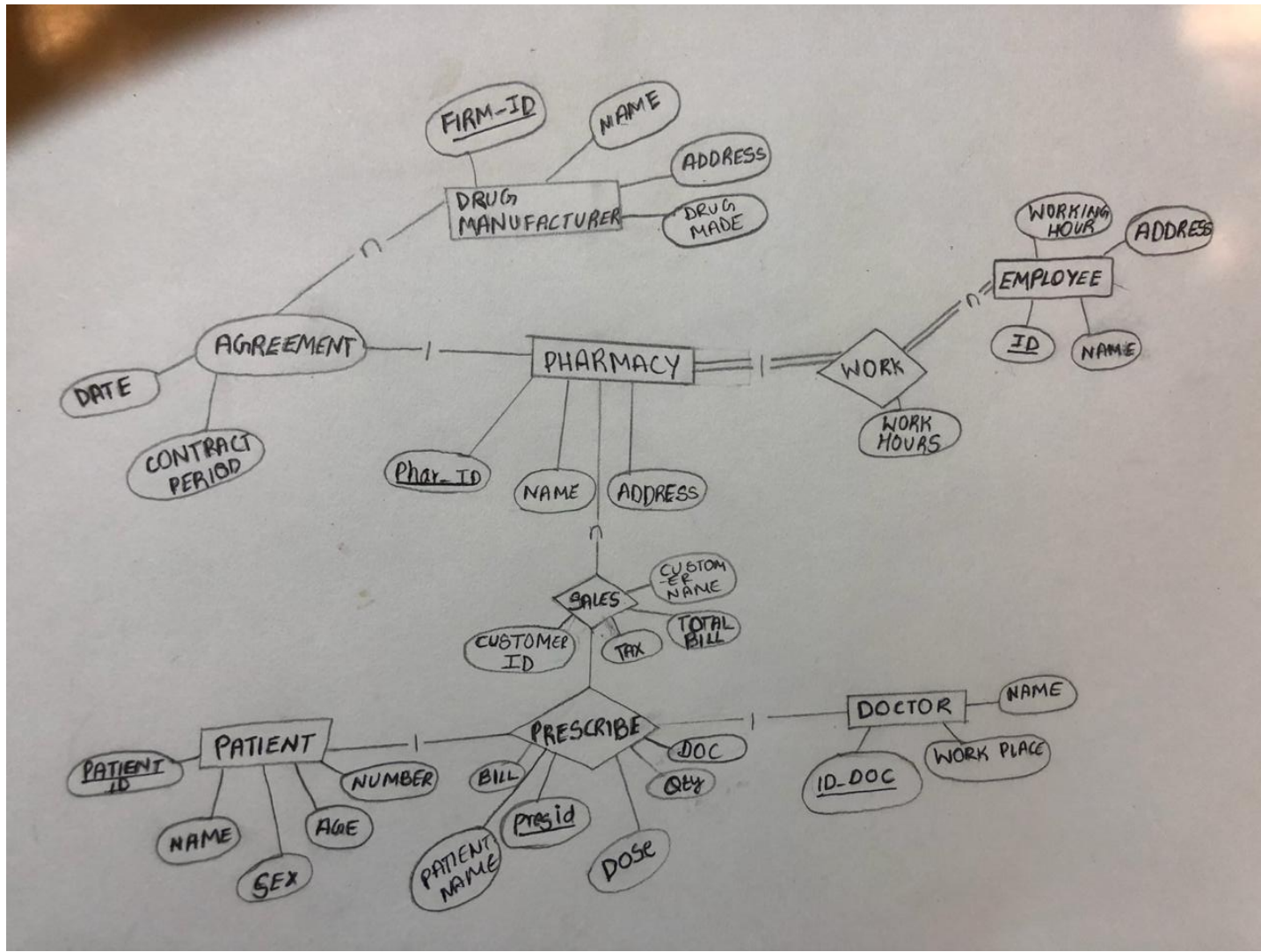
MINIWORLD : PHARMACY

### **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 .

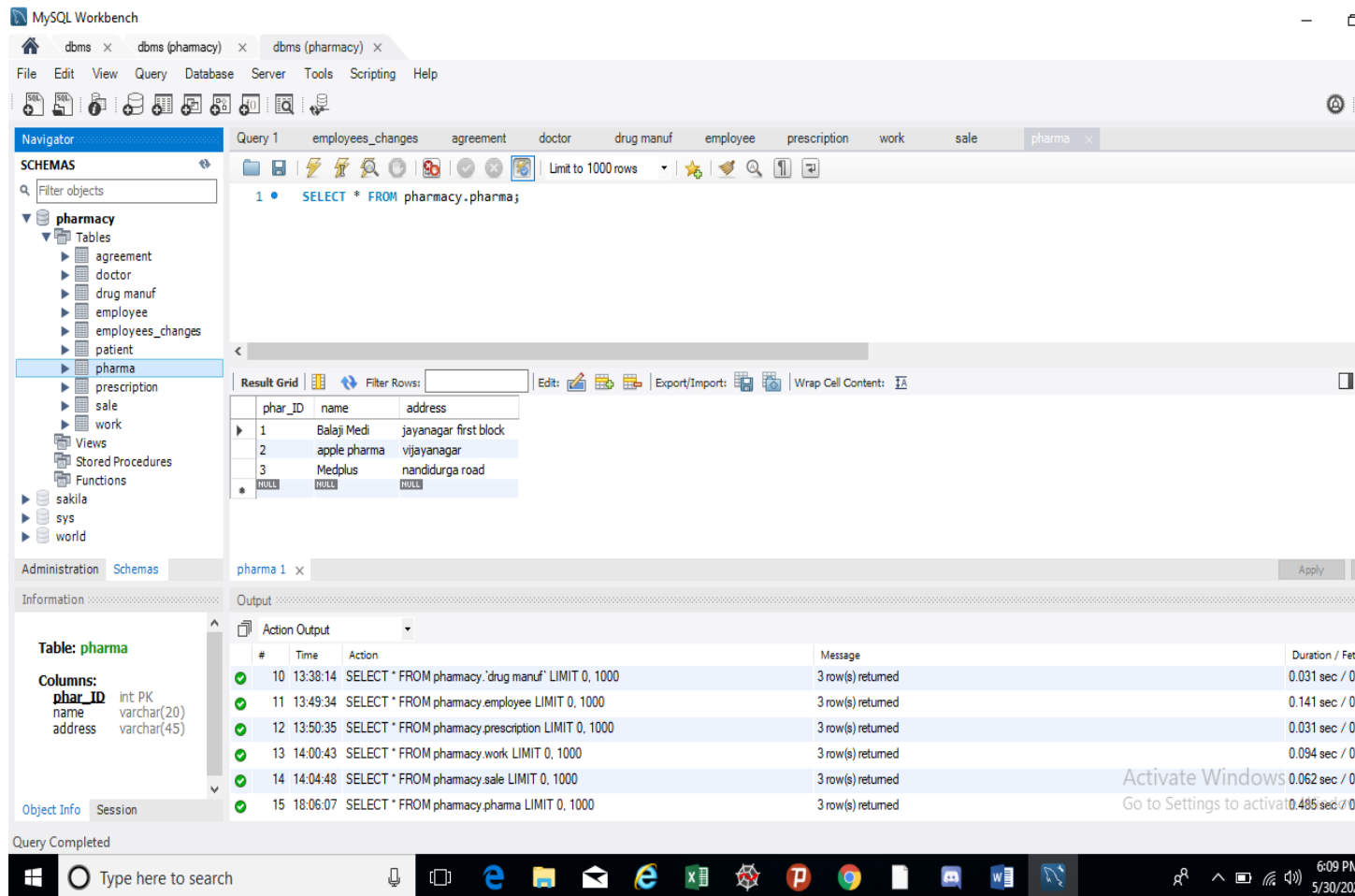
### **D) TABLE CREATION**

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 supposed 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 \*\***

Navigator: Filter objects

- pharmacy
  - Tables
    - agreement
    - doctor
    - drug manuf
    - employee
    - employees\_changes
    - patient
    - pharma
    - prescription
    - sale
    - work
  - Views
  - Stored Procedures
  - Functions
- sakila
- sys
- world

Administration Schemas

Information

Table: **employee**

Columns:

- emp\_ID int PK
- ename varchar(20)
- address varchar(30)

Query 1: employees\_changes agreement doctor drug manuf employee prescription work sale pharma patient

Limit to 1000 rows

1 • SELECT \* FROM pharmacy.patient;

Result Grid

|   | idpatient | name     | sex | age | contact no |
|---|-----------|----------|-----|-----|------------|
| ▶ | 31        | sanjana  | F   | 20  | 9844168569 |
|   | 32        | ramesh   | M   | 42  | 8861744562 |
|   | 33        | suresh   | M   | 56  | 9900091065 |
|   | 34        | debo     | F   | 23  | 9844165783 |
|   | 35        | manish   | M   | 34  | 9900098757 |
|   | 36        | abhishek | M   | 28  | 8877654354 |

patient 1 x

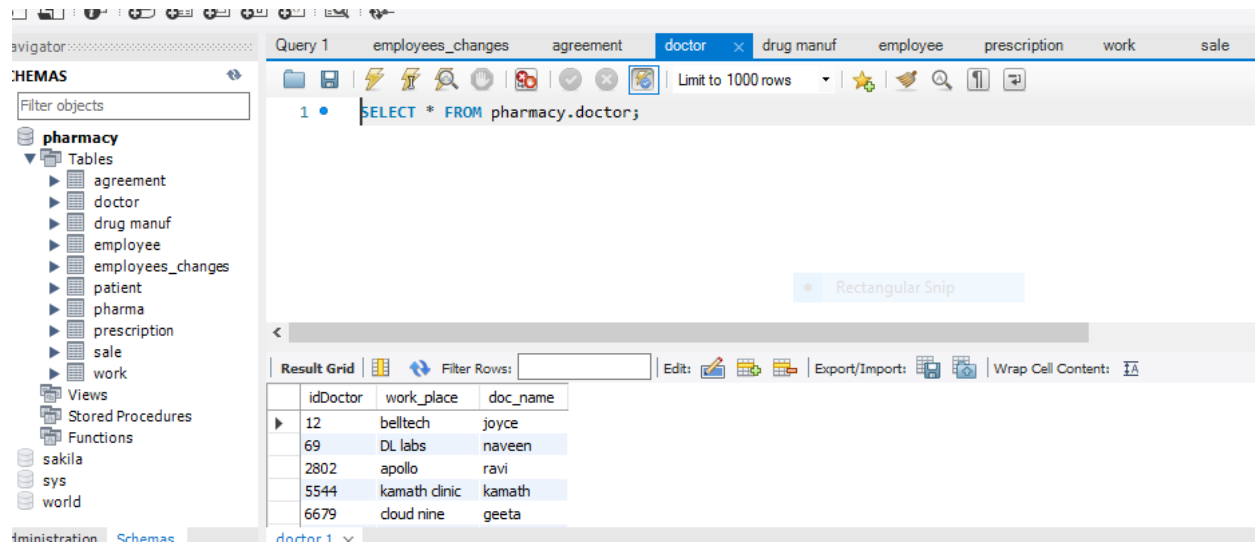
Output

Action Output

| #    | Time     | Action  | Message           |
|------|----------|---|-------------------|
| ✓ 11 | 13:49:34 | SELECT * FROM pharmacy.employee LIMIT 0, 1000     | 3 row(s) returned |
| ✓ 12 | 13:50:35 | SELECT * FROM pharmacy.prescription LIMIT 0, 1000 | 3 row(s) returned |
| ✓ 13 | 14:00:43 | SELECT * FROM pharmacy.work LIMIT 0, 1000         | 3 row(s) returned |

### 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,

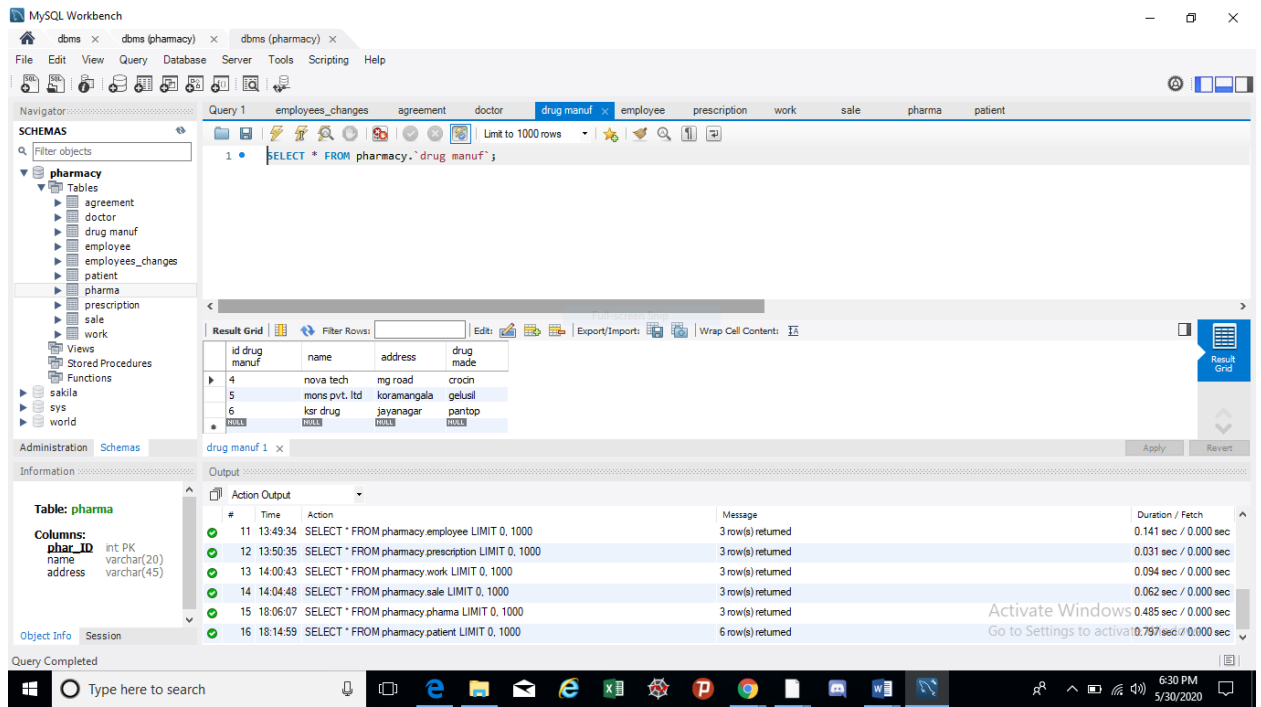
    `name` VARCHAR(20) NOT NULL,

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);
```

The screenshot shows the MySQL Workbench interface. The left sidebar displays the 'pharmacy' database schema with tables: agreement, doctor, drug\_manuf, employee, employees\_changes, patient, pharma, prescription, sale, and work. The 'pharma' table is selected, showing its columns: idDoctor, idpatient, Qty, dose(times in a day), bill\_amount, idpres, doc\_name, and patient\_name. The main query window shows a query: `SELECT * FROM pharmacy.prescription;` with the result grid displaying 3 rows of data. The bottom panel shows the 'Output' tab with a table of actions and their durations.

| #  | Time     | Action  | Message           | Duration / Fetch      |
|----|----------|---|-------------------|-----------------------|
| 11 | 13:49:34 | SELECT * FROM pharmacy.employee LIMIT 0, 1000     | 3 row(s) returned | 0.141 sec / 0.000 sec |
| 12 | 13:50:35 | SELECT * FROM pharmacy.prescription LIMIT 0, 1000 | 3 row(s) returned | 0.031 sec / 0.000 sec |
| 13 | 14:00:43 | SELECT * FROM pharmacy.work LIMIT 0, 1000         | 3 row(s) returned | 0.094 sec / 0.000 sec |
| 14 | 14:04:48 | SELECT * FROM pharmacy.sale LIMIT 0, 1000         | 3 row(s) returned | 0.062 sec / 0.000 sec |

## 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);

The screenshot shows the MySQL Workbench interface. The left sidebar displays the 'SCHEMAS' tree with the 'pharmacy' database selected. The 'Tables' list includes 'employees\_changes', 'agreement', 'doctor', 'drug\_manuf', 'employee', 'employees\_changes', 'patient', 'pharma', 'prescription', 'sale', 'work', 'Views', 'Stored Procedures', and 'Functions'. The 'employee' table is highlighted. The main query window shows a query: `SELECT * FROM pharmacy.employee;`. The 'Result Grid' displays the following data:

| emp_ID | ename  | address         |
|--------|--------|-----------------|
| 10     | savita | mantri apt      |
| 11     | shalni | pride apartment |
| 12     | anjali | rt nagar        |
| NULL   | NULL   | NULL            |

The bottom panel shows the 'Action Output' tab with the following log:

| #  | Time     | Action  | Message           | Duration / Fetch      |
|----|----------|---|-------------------|-----------------------|
| 11 | 13:49:34 | SELECT * FROM pharmacy.employee LIMIT 0, 1000     | 3 row(s) returned | 0.141 sec / 0.000 sec |
| 12 | 13:50:35 | SELECT * FROM pharmacy.prescription LIMIT 0, 1000 | 3 row(s) returned | 0.031 sec / 0.000 sec |
| 13 | 14:00:43 | SELECT * FROM pharmacy.work LIMIT 0, 1000         | 3 row(s) returned | 0.094 sec / 0.000 sec |

## 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,

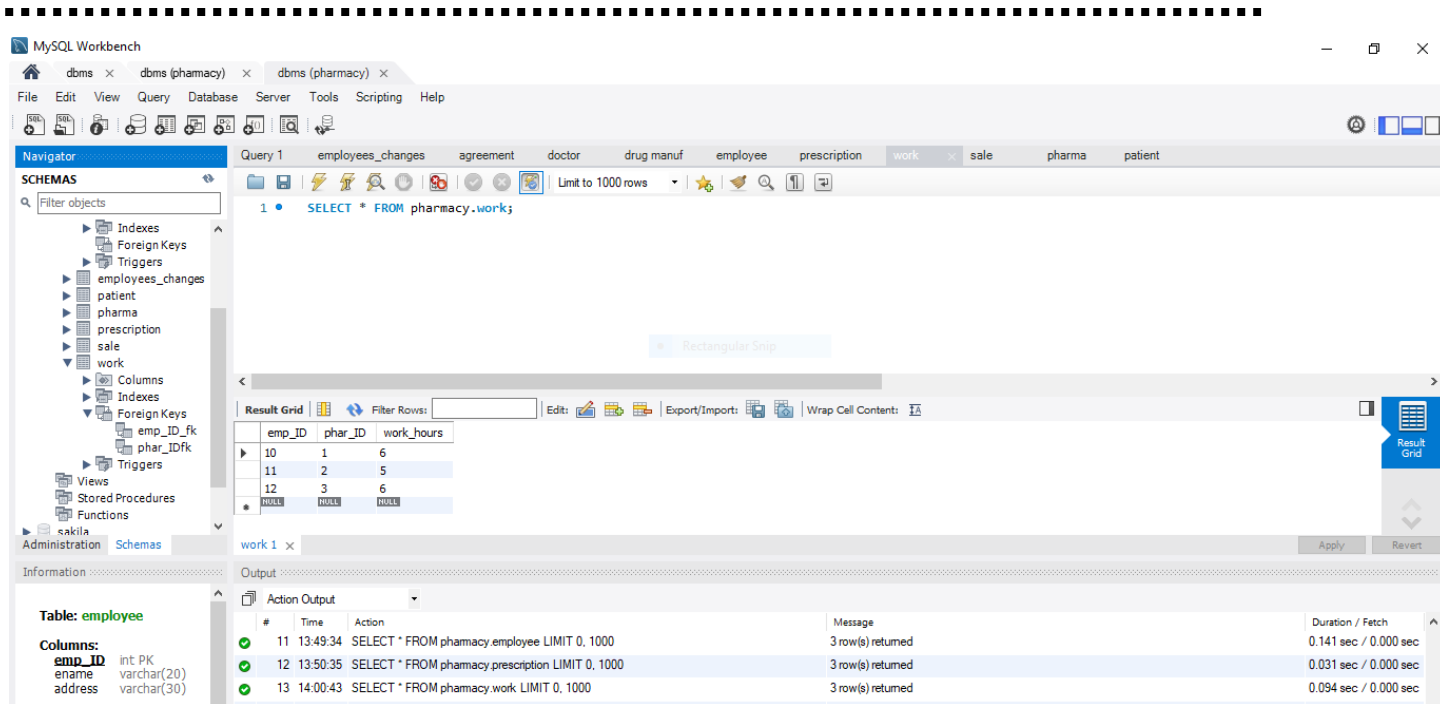
ADD CONSTRAINT `phar\_IDfk`

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);

```

Navigation: Schemas

Filter objects

pharmacy

- Tables
  - agreement
    - Columns
    - Indexes
    - Foreign Keys
      - phar\_ID\_fk
    - Triggers
  - doctor
  - drug manuf
  - employee
  - employees\_changes
  - patient
  - pharma
  - prescription
  - sale
  - work
- Views
- Stored Procedures

Administration Schemas

Information

Table: **employee**

Columns:

- emp\_ID int PK
- ename varchar(20)
- address varchar(30)

Query: 1 • SELECT \* FROM pharmacy.agreement;

Result Grid

| date of agreement | contract period | id drug manuf | phar_ID |
|-------------------|-----------------|---------------|---------|
| 2012-10-19        | 4               | 4             | 1       |
| 2004-08-12        | 3               | 5             | 2       |
| 2020-12-18        | 7               | 6             | 3       |
| NULL              | NULL            | NULL          | NULL    |

agreement 1 x

Output

| #  | Time     | Action  | Message           | Dura   |
|----|----------|---|-------------------|--------|
| 11 | 13:49:34 | SELECT * FROM pharmacy.employee LIMIT 0, 1000     | 3 row(s) returned | 0.14'  |
| 12 | 13:50:35 | SELECT * FROM pharmacy.prescription LIMIT 0, 1000 | 3 row(s) returned | 0.03'  |
| 13 | 14:00:43 | SELECT * FROM pharmacy.work LIMIT 0, 1000         | 3 row(s) returned | 0.094' |

## 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;
```

The screenshot shows a database management interface. On the left, the 'Navigator' pane displays the 'pharmacy' schema with various tables and indexes. The 'sale' table is selected, showing its columns, indexes, and foreign keys. The 'idpres\_fk' foreign key is highlighted. In the center, a query window shows the query 'SELECT \* FROM pharmacy.sale;'. Below the query, the 'Result Grid' displays the data from the 'sale' table. The grid has five columns: 'idpres', 'tax\_amt', 'total\_bill', 'customer\_name', and 'customer\_ID'. The data rows are as follows:

| idpres | tax_amt | total_bill | customer_name | customer_ID |
|--------|---------|------------|---------------|-------------|
| 40     | 567     | 2341       | sanjana       | 90          |
| 41     | 123     | 678        | ramesh        | 91          |
| 42     | 345     | 2345       | suresh        | 92          |
| *      | NULL    | NULL       | NULL          | NULL        |

## 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
);
```

The screenshot shows the MySQL Workbench interface. The left sidebar displays the 'SCHEMAS' tree with the 'pharmacy' database selected. The main window shows a query in the 'Query 1' tab: `SELECT * FROM pharmacy.employees_changes;`. Below the query, the 'Result Grid' shows a single row of data:

|   | id   | employeeNumber | name | changedat           | action |
|---|------|----------------|------|---------------------|--------|
| ▶ | 1    | 10             | ram  | 2020-05-30 12:50:59 | update |
| * | NULL | NULL           | NULL | NULL                | NULL   |

At the bottom, the 'Output' tab shows the 'Action Output' log with four entries:

| #    | Time     | Action  | Message           |
|------|----------|---|-------------------|
| ✓ 11 | 13:49:34 | SELECT * FROM pharmacy.employee LIMIT 0, 1000     | 3 row(s) returned |
| ✓ 12 | 13:50:35 | SELECT * FROM pharmacy.prescription LIMIT 0, 1000 | 3 row(s) returned |
| ✓ 13 | 14:00:43 | SELECT * FROM pharmacy.work LIMIT 0, 1000         | 3 row(s) returned |
| ✓ 14 | 14:04:48 | SELECT * FROM pharmacy.sale LIMIT 0, 1000         | 3 row(s) returned |

## E) QUERIES

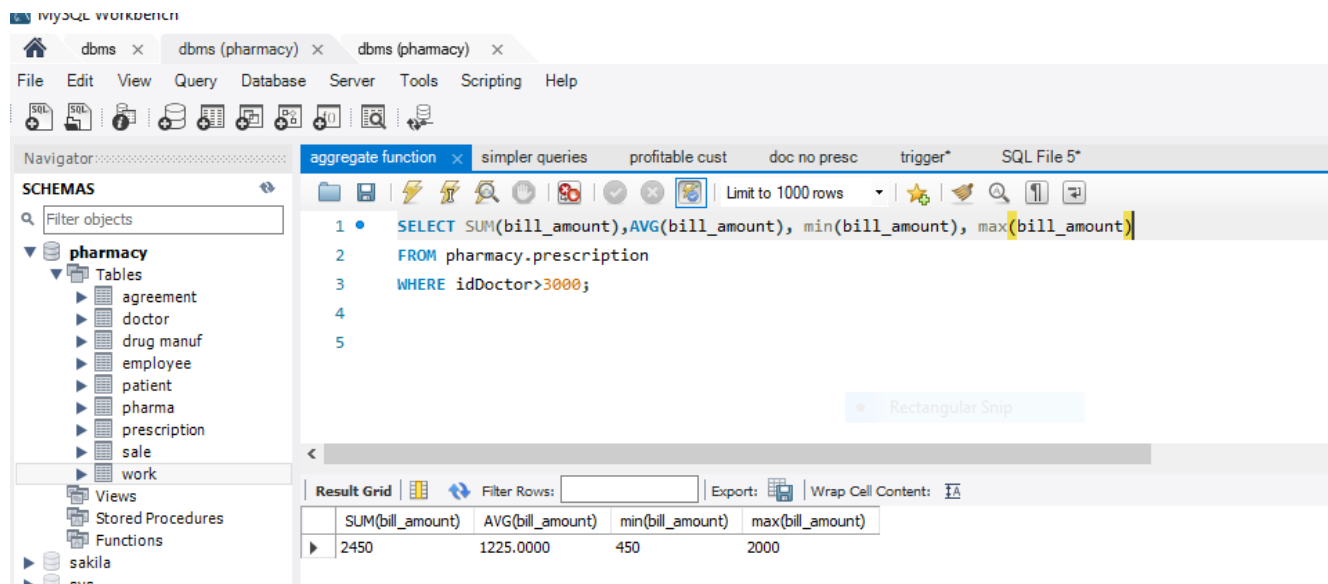
### 1) Created query implementing aggregate function

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

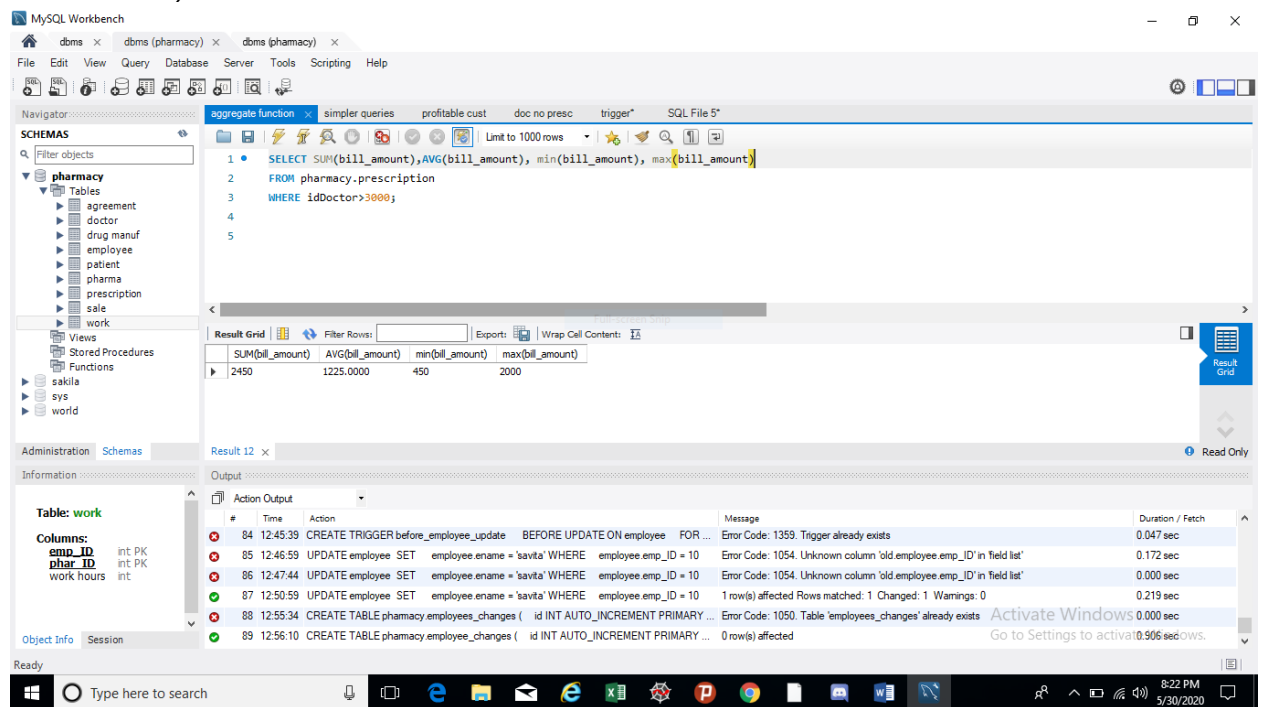
SELECT SUM(bill\_amount),AVG(bill\_amount), min(bill\_amount), max(bill\_amount)

FROM pharmacy.prescription

WHERE idDoctor>3000;



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



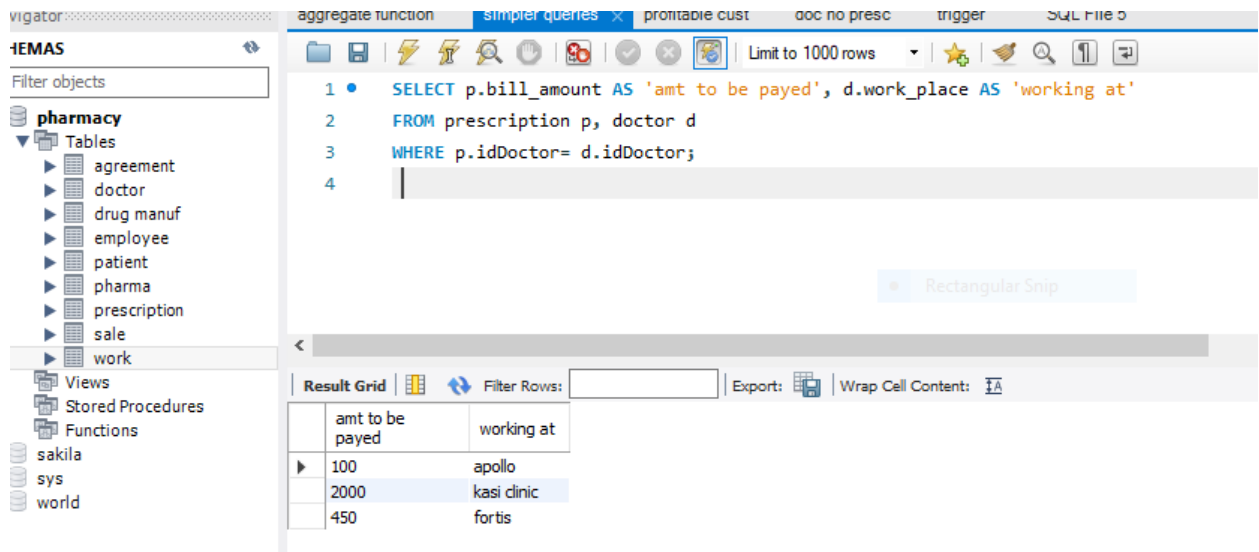
## 2) Query extracting data from a table as a result of two tables

➔ 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) Nested query

- ➔ 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;
```

The screenshot shows a SQL query in the main editor window. The query is as follows:

```

1 • SELECT "profitable customers", sale.customer_name AS focus_customer, sum(total_bill)
2   from sale
3   where total_bill >
4     (
5     select avg(total_bill)
6     from sale
7     )
8   group by focus_customer;

```

The result grid below the query shows the following data:

| profitable customers | focus_customer | sum(total_bill) |
|----------------------|----------------|-----------------|
| profitable customers | sanjana        | 2341            |
| profitable customers | suresh         | 2345            |

#### 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;

```

The screenshot shows a SQL query in the main editor window. The query is as follows:

```


1 • select doctor.doc_name AS 'doc',doctor.work_place AS 'their work place',prescription.qty,prescription.bill_amount
2   from pharmacy.prescription
3   right join doctor
4   on prescription.idDoctor = doctor.idDoctor
5   order by doctor.doc_name;

```

The result grid below the query shows the following data:

| doc    | their work place | qty  | bill_amount |
|--------|------------------|------|-------------|
| geeta  | cloud nine       | NULL | NULL        |
| jaya   | fortis           | 10   | 450         |
| joyce  | belltech         | NULL | NULL        |
| kamath | kamath clinic    | NULL | NULL        |

Result Grid


Filter Rows:

Exports

|   | doc    | their work place | qty  | bill_amount |
|---|--------|------------------|------|-------------|
| ▶ | geeta  | cloud nine       | NULL | NULL        |
|   | jaya   | fortis           | 10   | 450         |
|   | joyce  | belltech         | NULL | NULL        |
|   | kamath | kamath clinic    | NULL | NULL        |

Result 2

## **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 UPDATE\*

The screenshot shows a database management interface with a left-hand 'SCHEMAS' pane and a main 'Result Grid' pane. The 'SCHEMAS' pane shows a tree view of the database structure, including tables like 'employee' and 'pharmacy'. The 'Result Grid' pane displays the data from the 'employee' table. The query 'SELECT \* FROM pharmacy.employee;' is executed, showing three rows: (10, ram, mantri apt), (11, shalini, pride apartment), and (12, anjali, rt nagar). The first row is highlighted with a red checkmark, indicating it is the target of the update.

| emp_ID | ename   | address         |
|--------|---------|-----------------|
| 10     | ram     | mantri apt      |
| 11     | shalini | pride apartment |
| 12     | anjali  | rt nagar        |

\*AFTER UPDATE\*

HEMAS

Filter objects

- pharmacy
  - Tables
    - agreement
    - doctor
    - drug manuf
    - employee
    - employees\_changes
    - patient
    - pharma
    - prescription
    - sale
    - work
  - Views
  - Stored Procedures
  - Functions
- sakila
- sys
- world

1 • `SELECT * FROM pharmacy.employee;`

Limit to 1000 rows

Result Grid

|   | emp_ID | ename   | address         |
|---|--------|---------|-----------------|
| ▶ | 10     | savita  | mantri apt      |
|   | 11     | shalini | pride apartment |
|   | 12     | anjali  | rt nagar        |
| * | NULL   | NULL    | NULL            |

EMPLOYEE\_CHANGES , containing old employee data:

Query 1 employees\_changes agreement doctor drug manu

1 • `SELECT * FROM pharmacy.employees_changes;`

Limit to 1000 rows

Result Grid

|   | id   | employeeNumber | name | changedat           | action |
|---|------|----------------|------|---------------------|--------|
| ▶ | 1    | 10             | ram  | 2020-05-30 12:50:59 | update |
| * | NULL | NULL           | NULL | NULL                | NULL   |

### **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 .