Shanmukhi Baireddy
801309031

PROJECT – 2

Advanced features of an EMR Database System

Introduction:

An electronic medical record (EMR) is a digital version of all the information about a hospital or clinic such as medical history, diagnoses, medications, appointment dates, treatment, allergies, lab results and doctor's notes. EMRs are more than just a replacement for paper records. They effectively allow communication and coordination among members of a healthcare team for optimal patient care.

In this project, I have developed a database system for Royale Baby Children's Clinic. It is named as royale_baby_clinic. This database has been created to get an overview of this small practice of children's clinic. It is designed to manage clinic related information such as patient's information, medical records, record keeping of doctor's status and pharmacy inventory.

Scope:

The scope behind developing an EMR for children's database is to provide a comprehensive and efficient system for storing and accessing medical information for children. Moreover, an EMR for children's database can improve research and analysis of pediatric diseases and treatment outcomes.

Motivation:

Motivation behind selecting children's database is that they always require more time and specificities for their care. They also require close monitoring of parameters, personalized medication and special trained care providers who are passionate enough. That's the reasons I have opted for children's database.

Tools Used: MySQL workbench

Audit Trails:

audit_appointmentinfo: Has the logs of appointment_info table with a performed action column consisting whether the row is inserted, updated or deleted and another column with the system time stamp when the row is updated and user column which consists of name of the user updating the row.

Columns:

appointment_id	Int AI PK
patient_id	Int
nurse_id	Int
doctor_id	Int
start_datetime	Datetime
end_datetime	Datetime
room_number	Varchar(50)
Action_Performed	Varchar(50)
Action_Time	Datetime
user	Varchar(100)

Shanmukhi Baireddy
801309031

audit_bill: Has the logs of bill table with a performed action column consisting whether the row is inserted, updated or deleted and another column with the system time stamp when the row is updated and user column which consists of name of the user updating the row.

Columns:

bill_id	Int AI PK
appointment_id	Int
total_cost	Decimal(9,3)
date_	Int
Action_Performed	Varchar(50)
Action_Time	Datetime
user	Varchar(100)

audit_clinicemployee: Has the logs of clinic_employee table with a performed action column consisting whether the row is inserted, updated or deleted and another column with the system time stamp when the row is updated and user column which consists of name of the user updating the row.

Columns:

Employee_id	Int PK
First_Name	Varchar(50)
Last_Name	Varchar(50)
Designation	Varchar(50)
Salary	Decimal(9,2)
Action_Performed	Varchar(50)
Action_Time	Datetime
user	Varchar(100)

audit_healthinsurance: Has the logs of health_insurance table with a performed action column consisting whether the row is inserted, updated or deleted and another column with the system time stamp when the row is updated and user column which consists of name of the user updating the row.

Columns:

insurance_id	Int PK
insurance_provider	Varchar(100)
policy_no	Varchar(100)
plan_type	Varchar(100)
start_date	Datetime
end_date	Datetime
Action_Performed	Varchar(50)
Action_Time	Datetime
user	Varchar(100)

audit_medicalprescription: Has the logs of medical_prescription table with a performed action column consisting whether the row is inserted, updated or deleted and another column with the system time stamp when the row is updated and user column which consists of name of the user updating the row.

Shanmukhi Baireddy
801309031

Columns:

appointment_id	Int PK
medicine_id	Varchar PK
medicine_dose	Int
Action_Performed	Varchar(50)
Action_Time	Datetime
user	Varchar(100)

audit_patientinfo: Has the logs of patient_info table with a performed action column consisting whether the row is inserted, updated or deleted and another column with the system time stamp when the row is updated and user column which consists of name of the user updating the row.

Columns:

patient_id	Int PK
First_Name	Varchar(25)
Middle_Name	Int
Last_Name	Int
Gender	Datetime
Age	Datetime
Phone_number	Varchar(50)
Address	Varchar(50)
Insurance_id	Datetime
pcp	Varchar(100)
Action_Performed	Varchar(50)
Action_Time	datetime
user	Varchar(100)

audit_patientprocedure: Has the logs of patient_procedure table with a performed action column consisting whether the row is inserted, updated or deleted and another column with the system time stamp when the row is updated and user column which consists of name of the user updating the row.

Columns:

procedure_no	Int
appointment_id	Int PK
procedure_id	Int PK
date_	datetime
doctor_id	int
nurse_id	int
Action_Performed	Varchar(50)
Action_Time	Datetime
user	Varchar(100)

audit_patientsymptoms: Has the logs of patient_symptoms table with a performed action column consisting whether the row is inserted, updated or deleted and another column with the system time stamp when the row is updated and user column which consists of name of the user updating the row.

Shanmukhi Baireddy
801309031

Columns:

appointment_id	Int PK
fever	tinyint
flu	tinyint
ear_infection	tinyint
stomach_pain	tinyint
pneumonia	tinyint
toothpain	tinyint
Action_Performed	Varchar(50)
Action_Time	Datetime
user	Varchar(100)

audit_pharmacyinventory: Has the logs of pharmacy_inventory table with a performed action column consisting whether the row is inserted, updated or deleted and another column with the system time stamp when the row is updated and user column which consists of name of the user updating the row.

medicine_id	Varchar(50) PK
medicine_name	Varchar(150)
supplier	Varchar(100)
description	Varchar(300)
price_per_item	Decimal(8,2)
Action_Performed	Varchar(50)
Action_Time	Datetime
user	Varchar(100)

audit_procedureresults: Has the logs of procedure_results table with a performed action column consisting whether the row is inserted, updated or deleted and another column with the system time stamp when the row is updated and user column which consists of name of the user updating the row.

Columns:

procedure_id	Int
results	Varchar(150)
summary	Varchar(200)
Action_Performed	Varchar(50)
Action_Time	Datetime
user	Varchar(100)

audit_testprocedure: Has the logs of test_procedure table with a performed action column consisting whether the row is inserted, updated or deleted and another column with the system time stamp when the row is updated and user column which consists of name of the user updating the row.

Columns:

id	Int PK
name	Varchar(100)
description	Varchar(500)
price	Int
Action_Performed	Varchar(50)

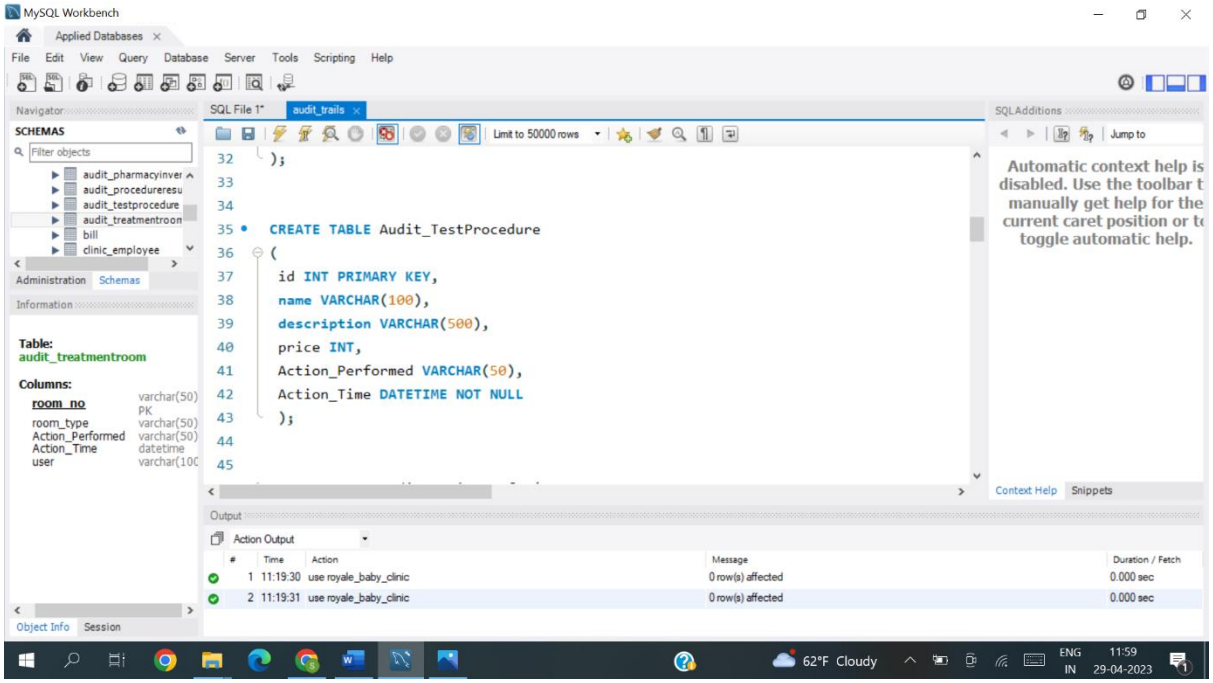
Action_Time	Datetime
user	Varchar(100)

audit_treatmentroom: Has the logs of clinic_treatment_room table with a performed action column consisting whether the row is inserted, updated or deleted and another column with the system time stamp when the row is updated and user column which consists of name of the user updating the row.

Columns:

room_no	Varchar(50) PK
room_type	Varchar(50)
Action_Performed	Varchar(50)
Action_Time	Datetime
user	Varchar(100)

Creating the Audit_TestProcedure Table:



Updating an entry in test_procedure table:

Shanmukhi Baireddy
801309031

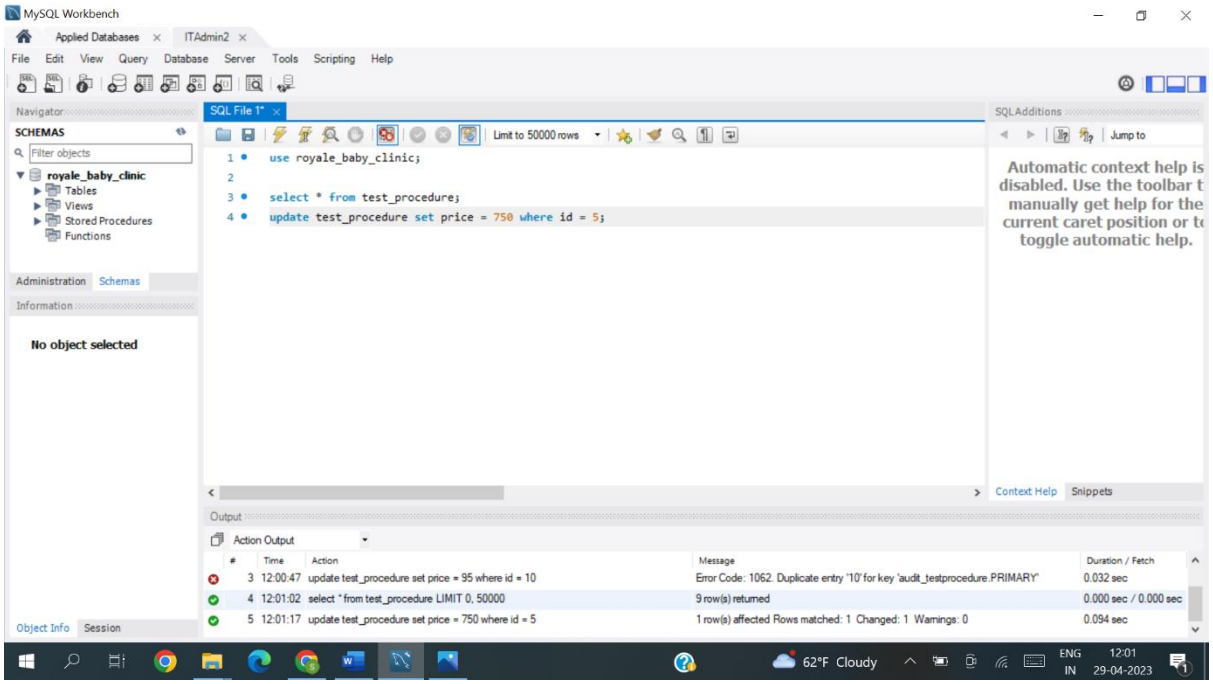
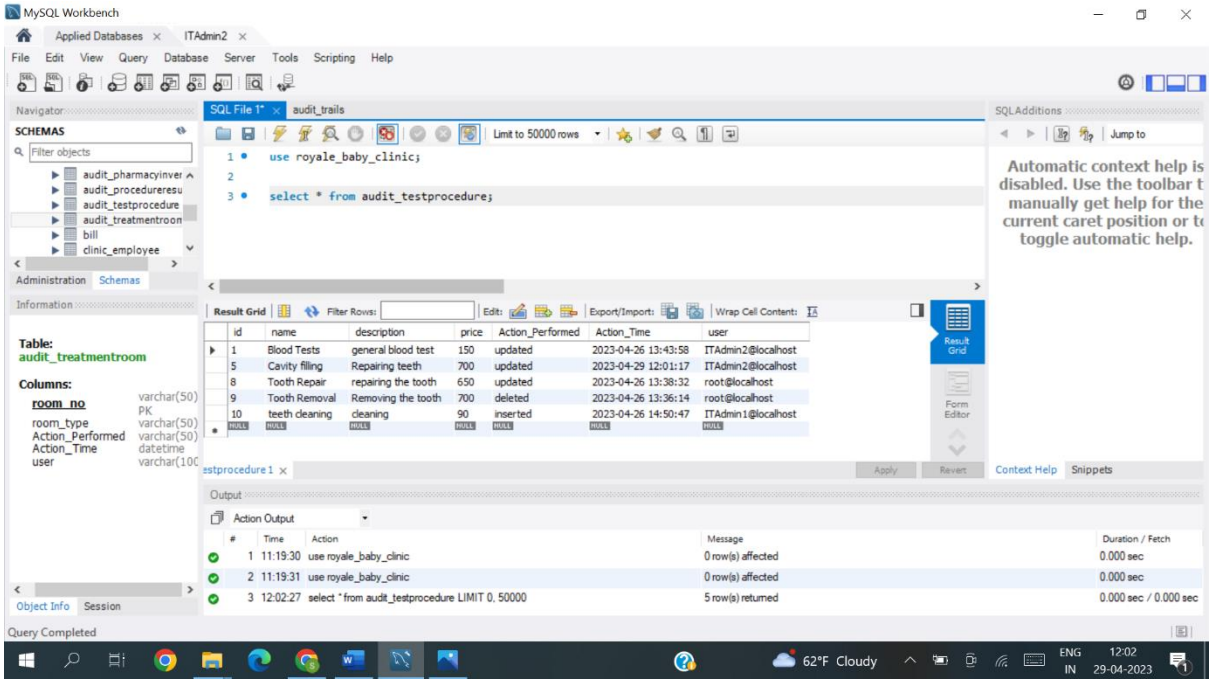
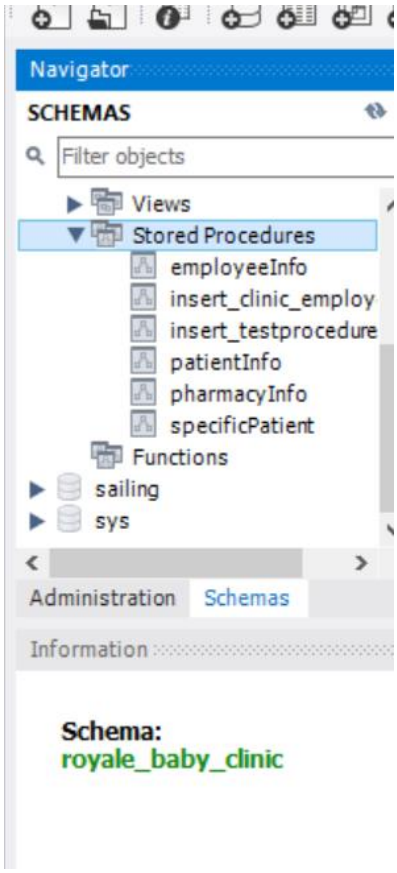


Table showing who have done changes to the test_procedure table.



Shanmukhi Baireddy
801309031

Stored Procedures:

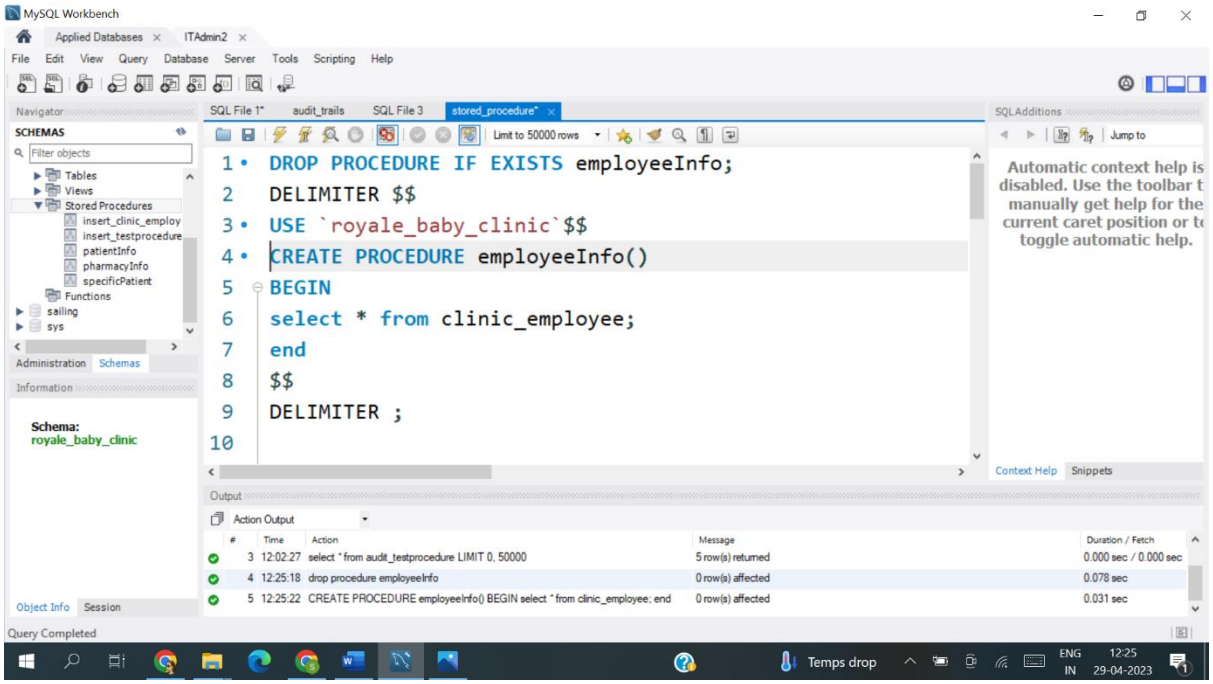


SELECT Stored Procedures:

Procedure to display information about employees in the clinic:

```
DROP PROCEDURE IF EXISTS employeeInfo;

DELIMITER $$

USE `royale_baby_clinic`$$

CREATE PROCEDURE employeeInfo()

BEGIN

select * from clinic_employee;

end

$$

DELIMITER ;
```

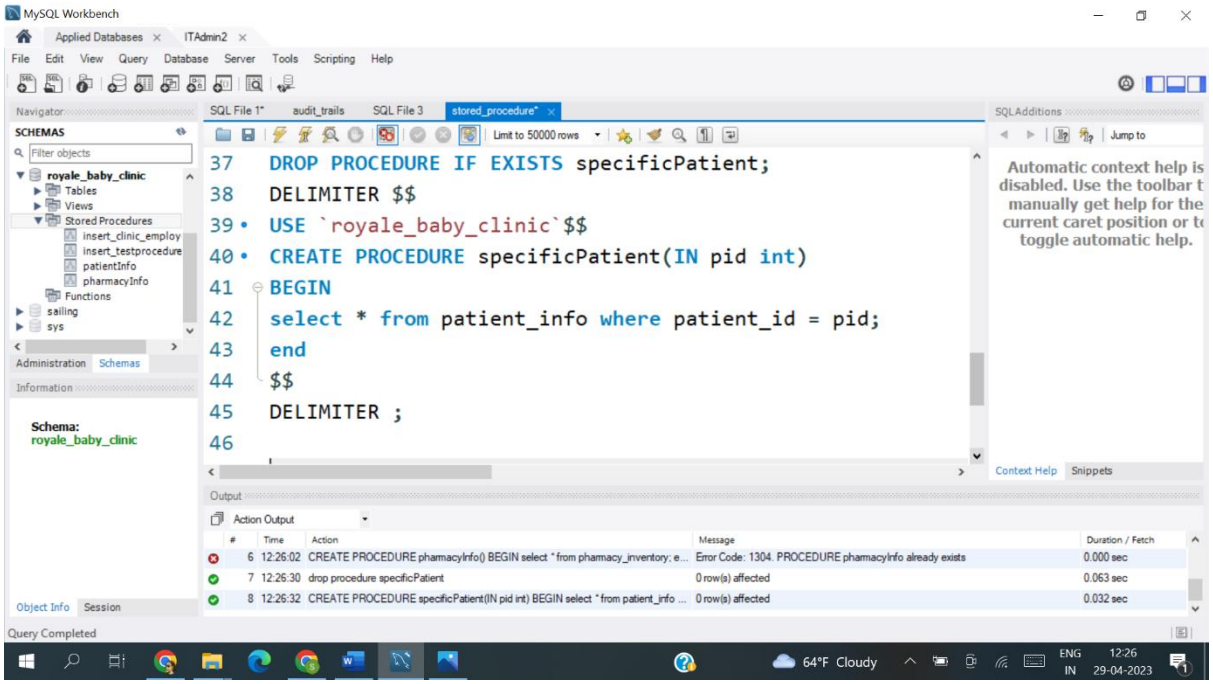
Shanmukhi Baireddy
801309031



Procedure to display information about a specific patient from patient_info table:

```
DROP PROCEDURE IF EXISTS specificPatient;

DELIMITER $$

USE `royale_baby_clinic`$$

CREATE PROCEDURE specificPatient(IN pid int)

BEGIN

select * from patient_info where patient_id = pid;

end

$$

DELIMITER ;
```

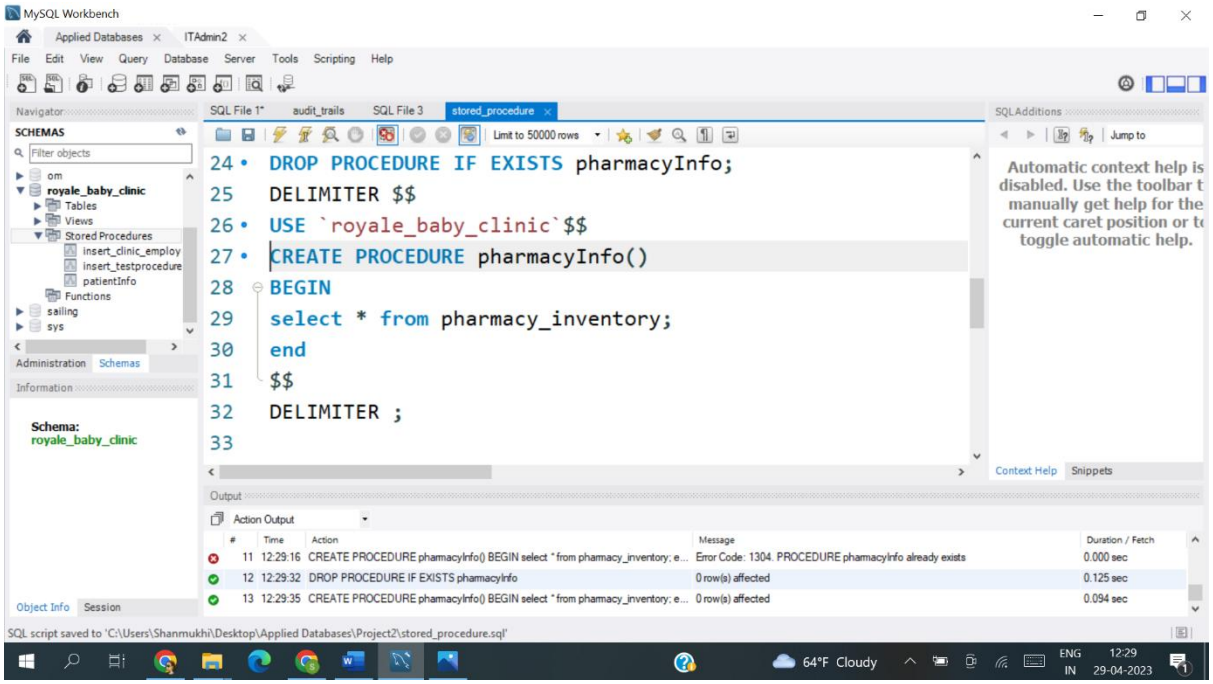
Shanmukhi Baireddy
801309031



Procedure to display information of pharmacy_inventory table:

```
DROP PROCEDURE IF EXISTS pharmacyInfo;

DELIMITER $$

USE `royale_baby_clinic`$$

CREATE PROCEDURE pharmacyInfo()

BEGIN

select * from pharmacy_inventory;

end

$$

DELIMITER ;
```

Shanmukhi Baireddy
801309031



Creating a text-based interface and calling API to use the database functionalities:

Created a text-based interface using python programming language.

Upon running the program using ‘python stored_procedures.py’ command

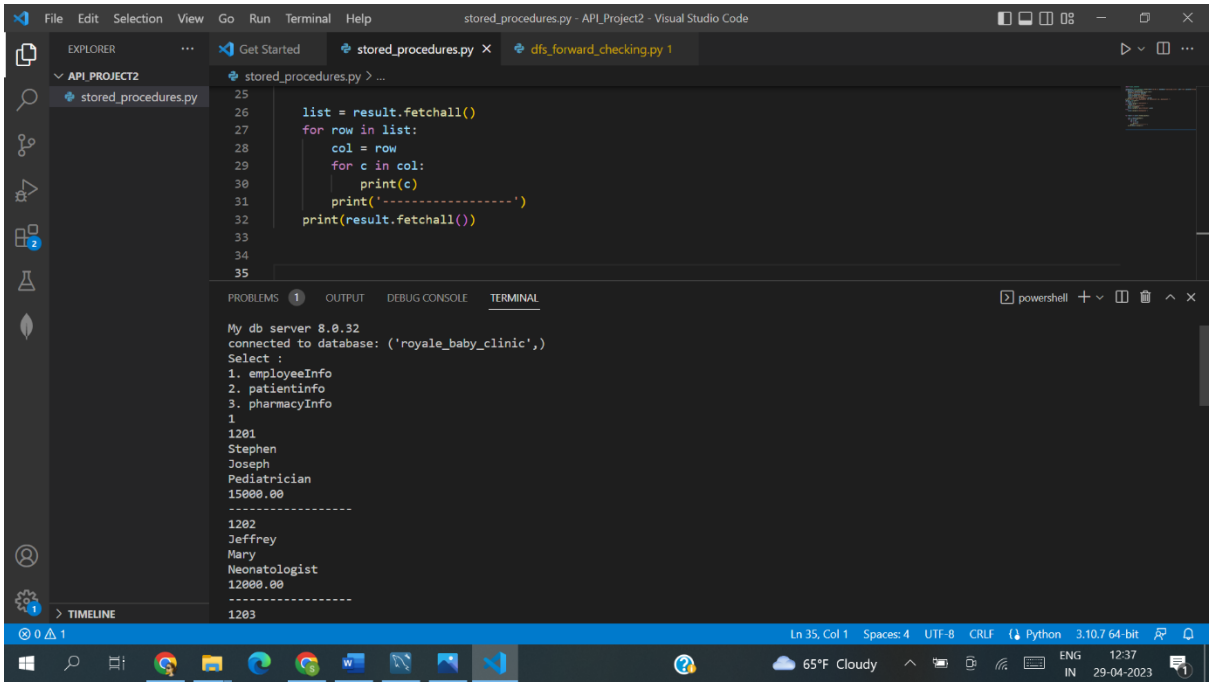
It asks to select numbers from 1-3:

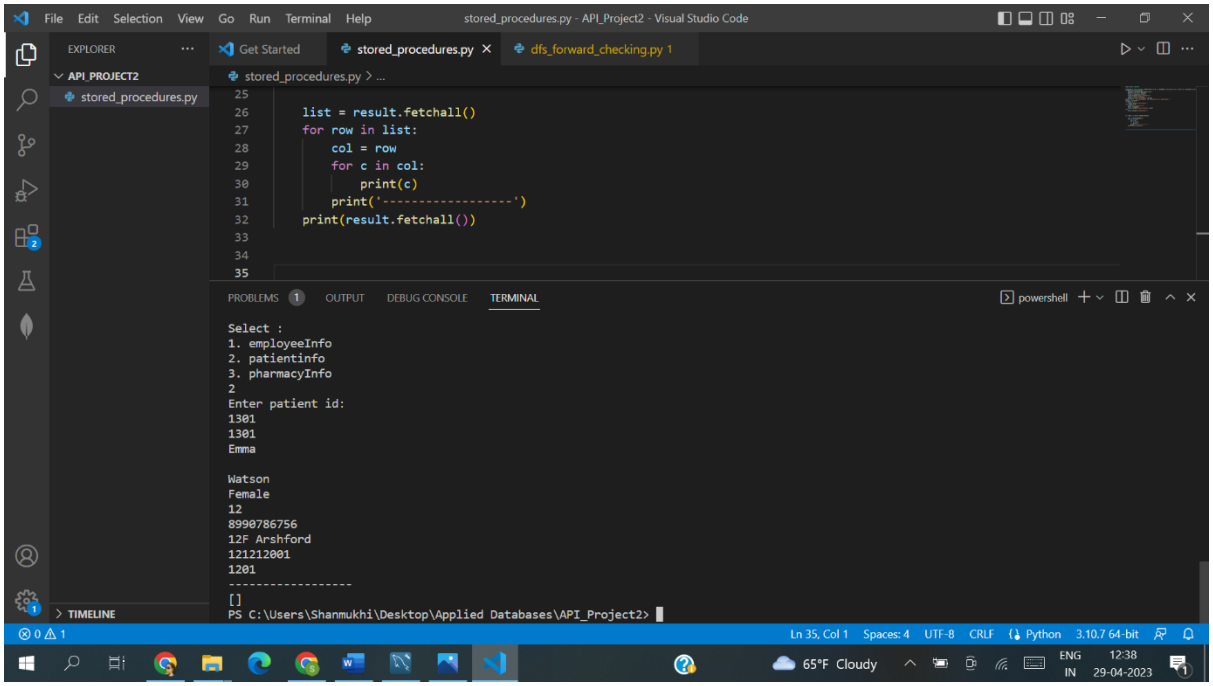
Upon selecting 1 it displays employee information of the clinic

Upon selecting 2 it asks to enter patient id and then displays information about that particular patient.

Upon selecting 3 it displays information about the pharmacy inventory of the clinic.

Shanmukhi Baireddy
801309031





Code used for creating text-based interface:

```
import mysql.connector

connection =
mysql.connector.connect(host='127.0.0.1',database='royale_baby_clinic',user='r
oot',password='Willywonka@12')
if connection.is_connected():
    db_Info = connection.get_server_info()
    print("My db server",db_Info)
    cursor = connection.cursor()
    cursor.execute("Select database();")
```

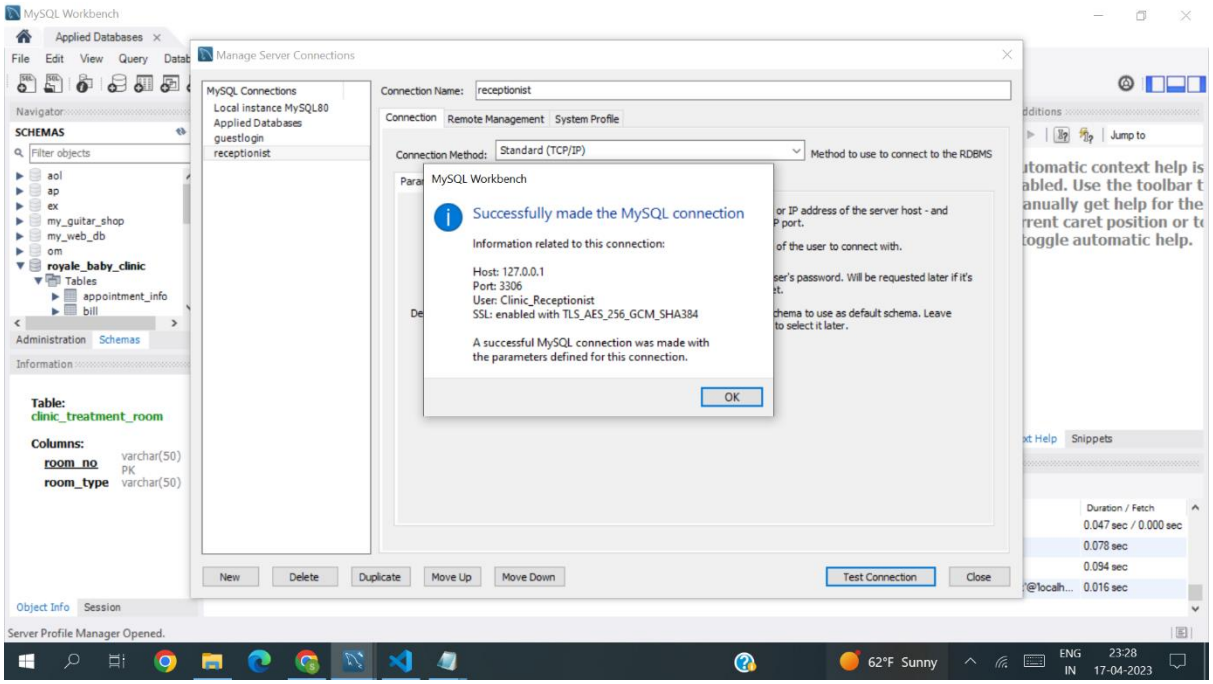
```
    record = cursor.fetchone()
    print("connected to database:",record)
print("Select : \n1. employeeInfo \n2. patientinfo \n3. pharmacyInfo ")
choice = int(input())
if choice == 1:
    cursor.callproc('employeeInfo')
elif choice == 2:
    print("Enter patient id:")
    pid = int(input())
    cursor.callproc('specificPatient',[pid])
else:
    cursor.callproc('pharmacyInfo')



for result in cursor.stored_results():

    list = result.fetchall()
    for row in list:
        col = row
        for c in col:
            print(c)
        print('------------------')
    print(result.fetchall())
```

USERS & ROLES:

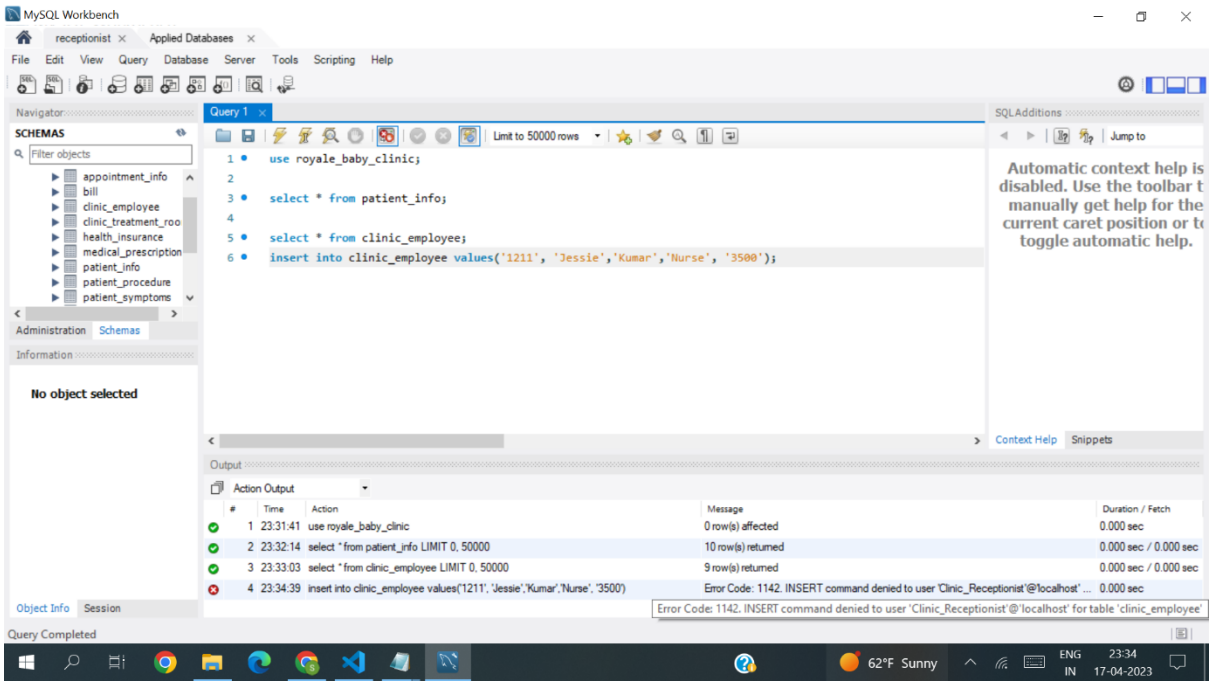
-- creating receptionist

CREATE USER Clinic_Receptionist@localhost IDENTIFIED BY 'Baby@123';

GRANT SELECT ON royale_baby_clinic.* TO Clinic_Receptionist@localhost WITH GRANT OPTION;



Created a user receptionist granting select privileges on the receptionist.



Above you can see that when the receptionist tries to insert a row into the table, an error code :1142 which is stating the permission was denied to them as they don't have the necessary privileges granted.

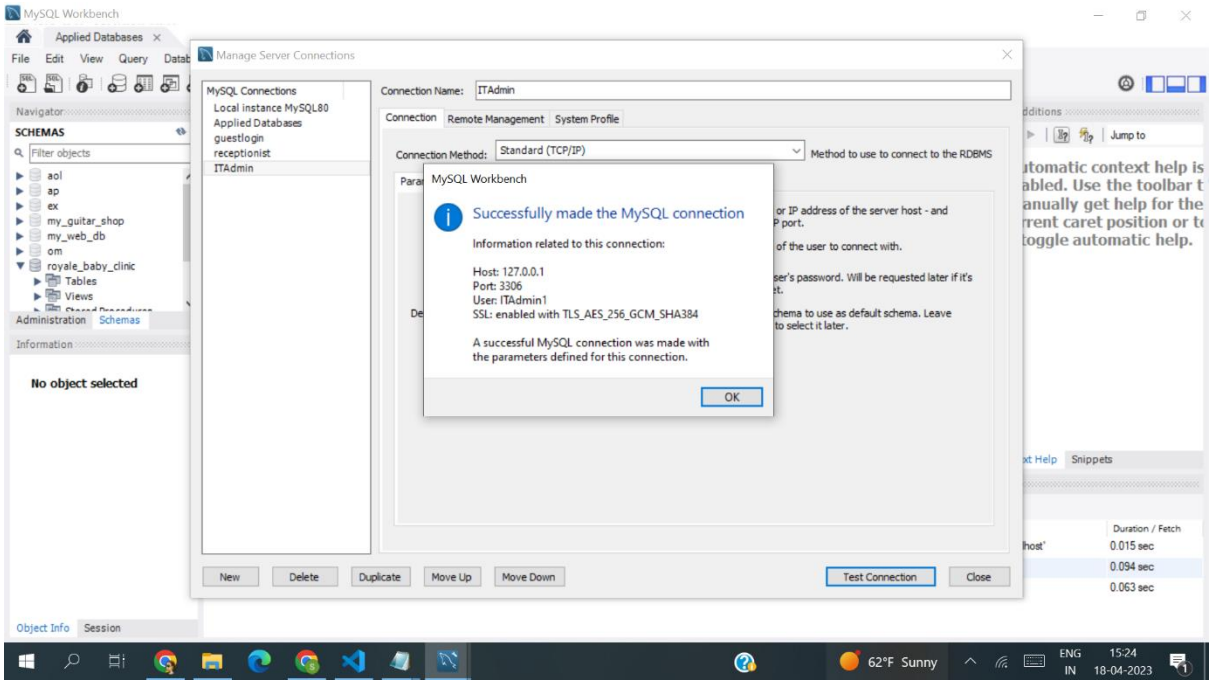
Created user ITAdmin1:

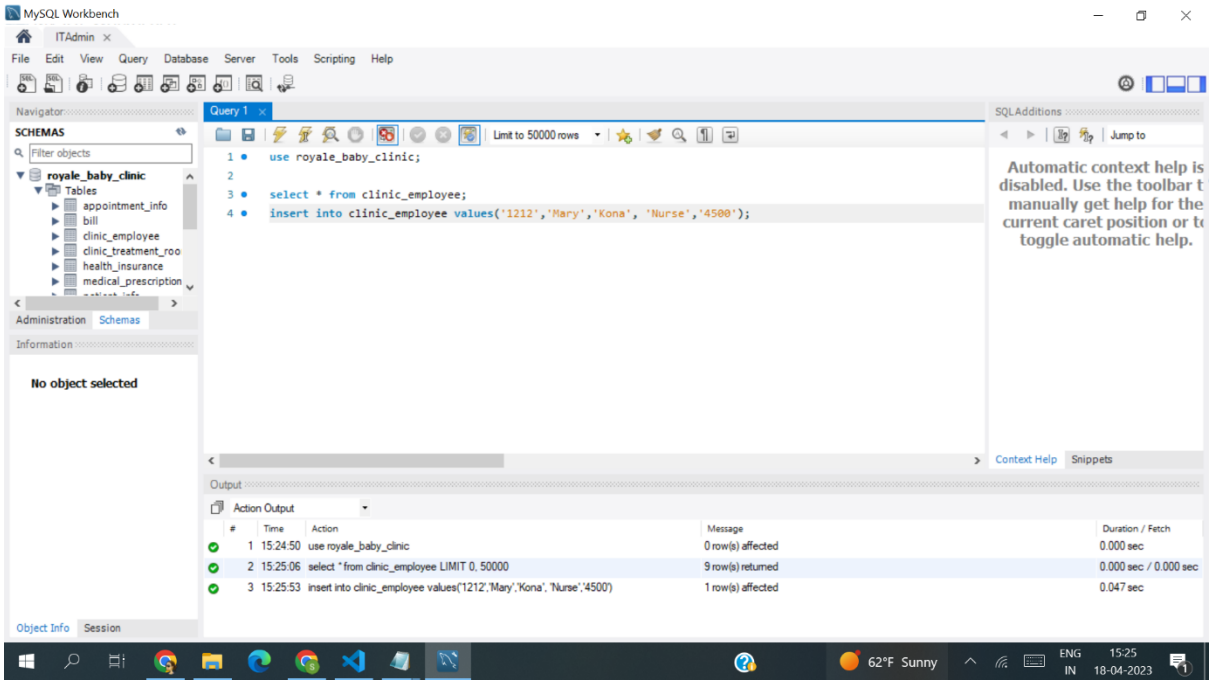
CREATE USER ITAdmin1@localhost IDENTIFIED BY 'Admin@123';

Granting all privileges to ITAdmin1 on database.

GRANT ALL PRIVILEGES ON royale_baby_clinic.* TO ITAdmin1@localhost WITH GRANT OPTION;

Shanmukhi Baireddy
801309031





From the above, the ITAdmin was able to select and insert into the tables of the database.

Created user seniordoctor:

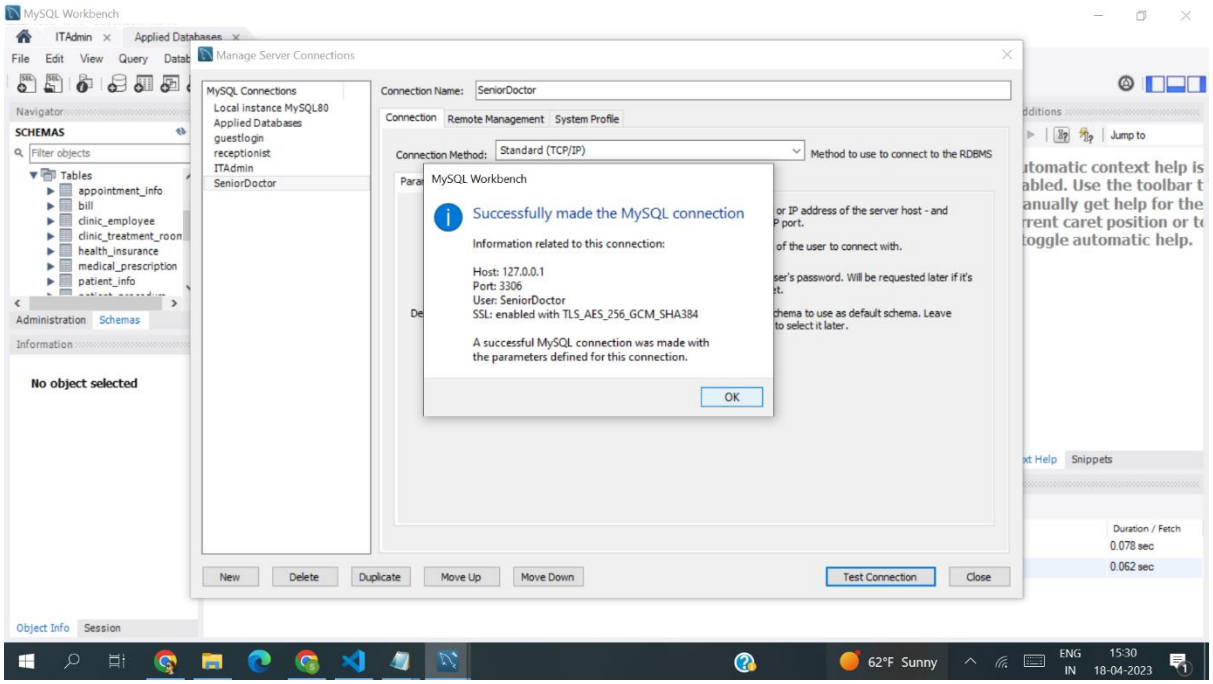
CREATE USER SeniorDoctor@localhost IDENTIFIED BY 'Doctor@123';

Granted the doctor all privileges only on medical_prescription and patient_info table.

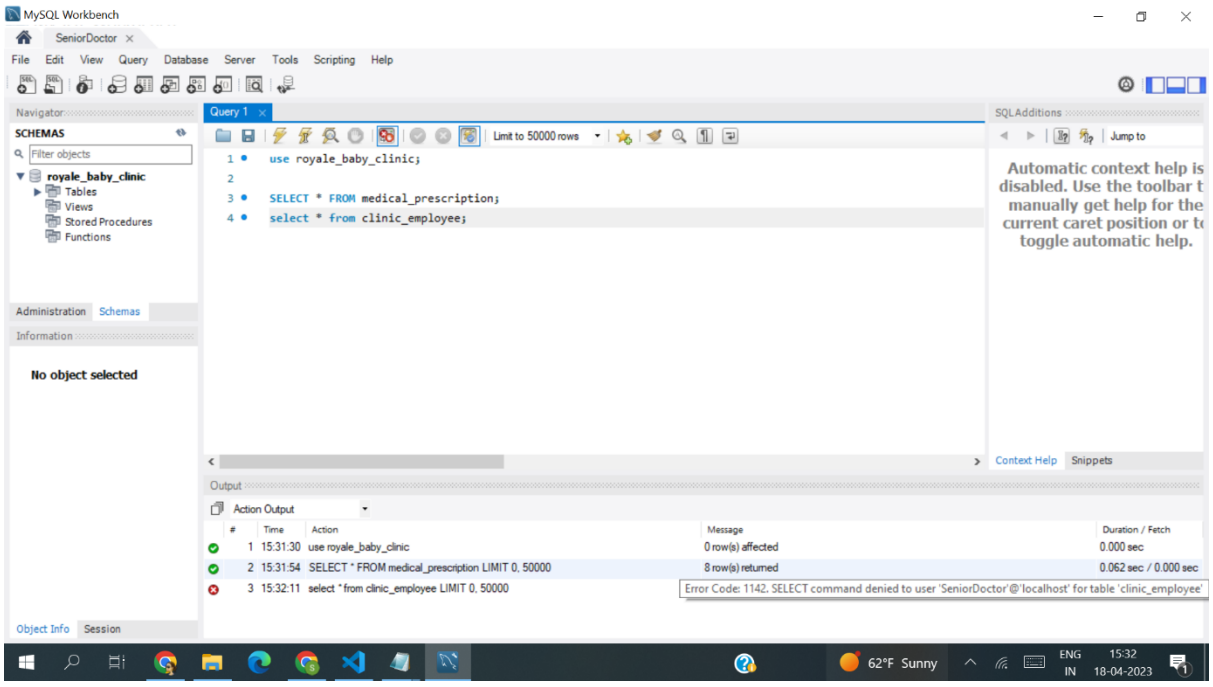
GRANT ALL PRIVILEGES ON royale_baby_clinic.medical_prescription TO SeniorDoctor@localhost WITH GRANT OPTION;

GRANT ALL PRIVILEGES ON royale_baby_clinic.patient_info TO SeniorDoctor@localhost WITH GRANT OPTION;

Shanmukhi Baireddy
801309031



Doctor trying to access clinic_employee table:



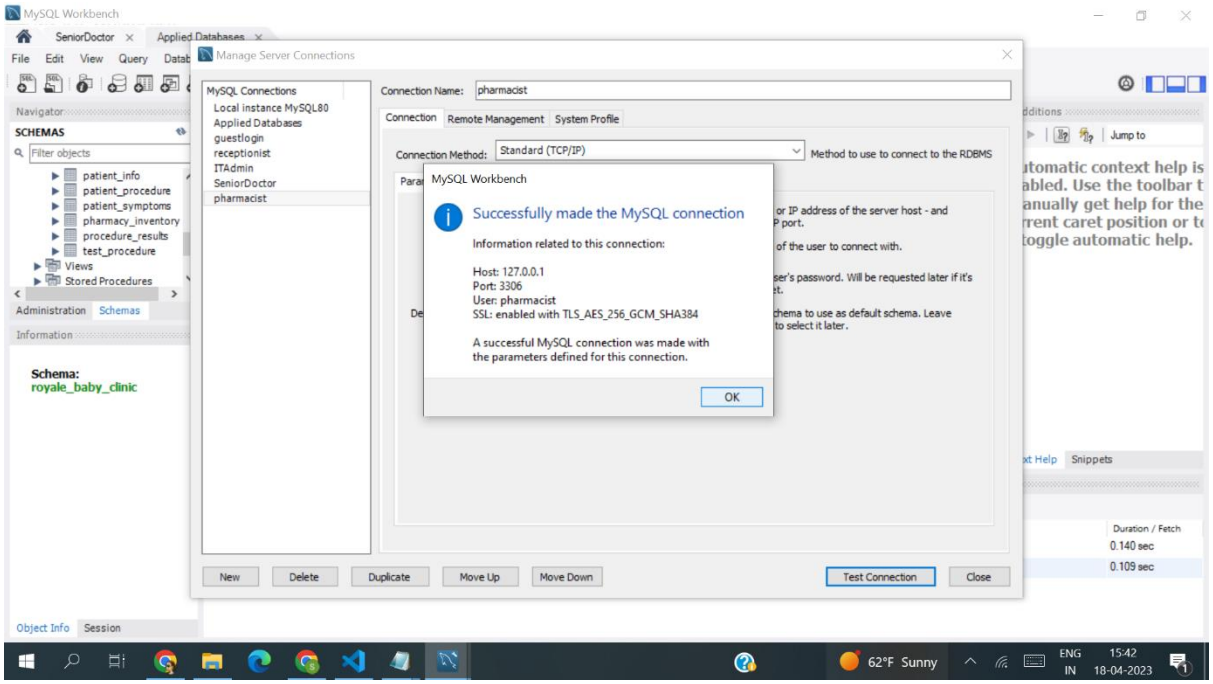
Created user pharmacist:

CREATE USER pharmacist@localhost IDENTIFIED BY 'pharma@123';

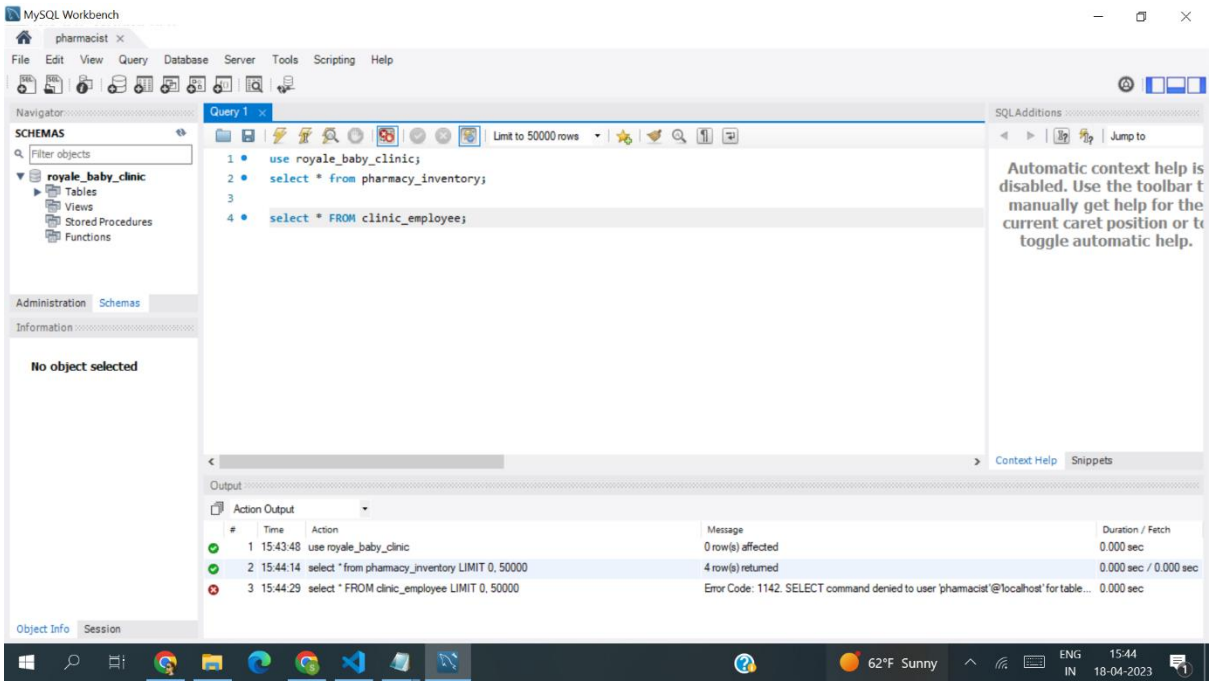
Granting the pharmacist all privileges only on pharmacy_inventory table:

GRANT ALL PRIVILEGES ON royale_baby_clinic.pharmacy_inventory TO pharmacist@localhost WITH GRANT OPTION;

Shanmukhi Baireddy
801309031



Pharmacist trying to access another table but is denied:

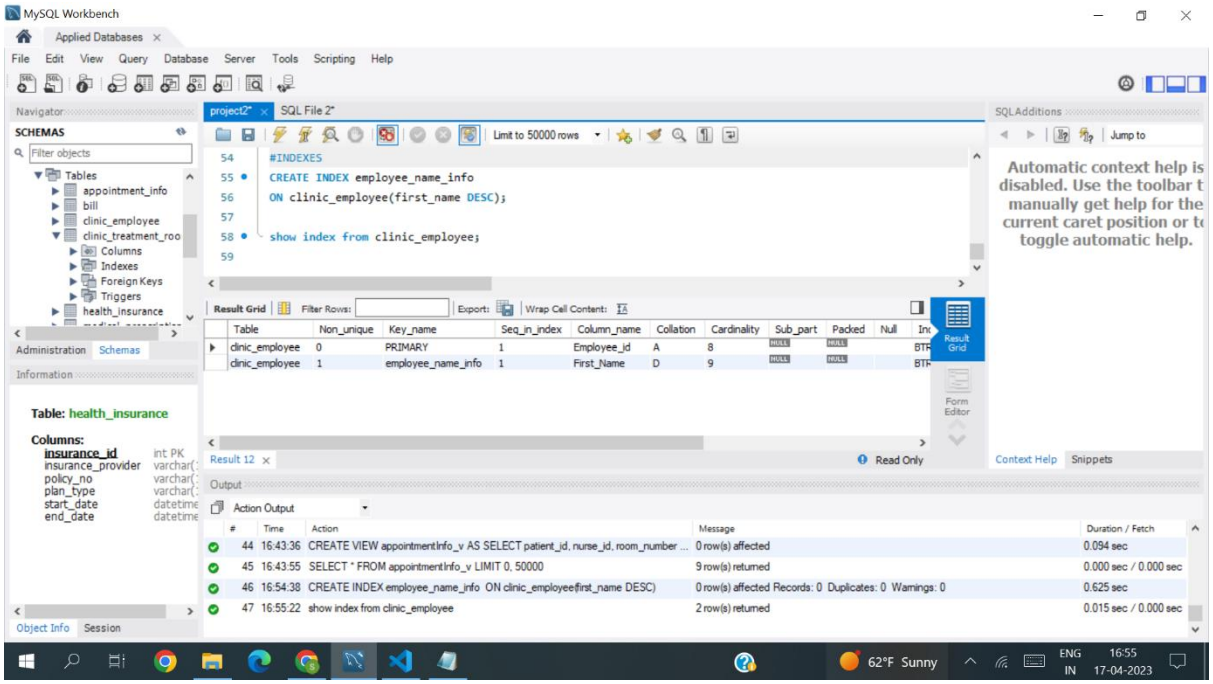


INDEXES:

Index to Recognizing Clinic Employee by first name:

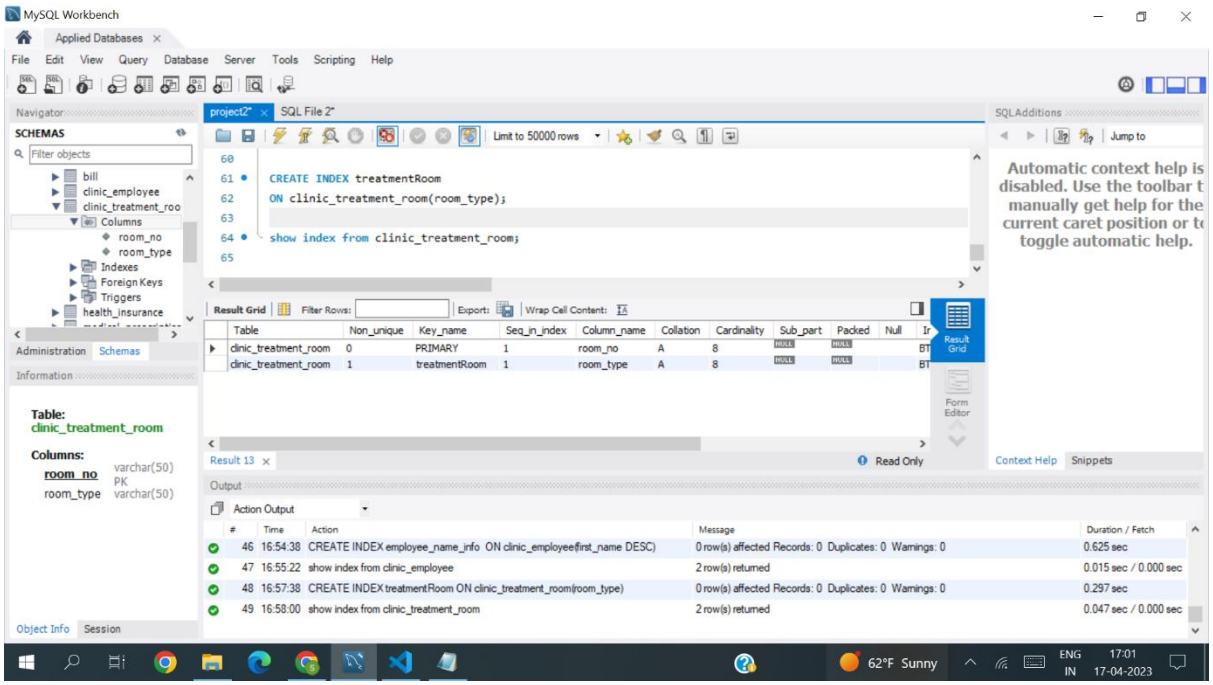
CREATE INDEX employee_name_info ON clinic_employee(first_name DESC);

Shanmukhi Baireddy
801309031



Index to recognize treatment room by room type:

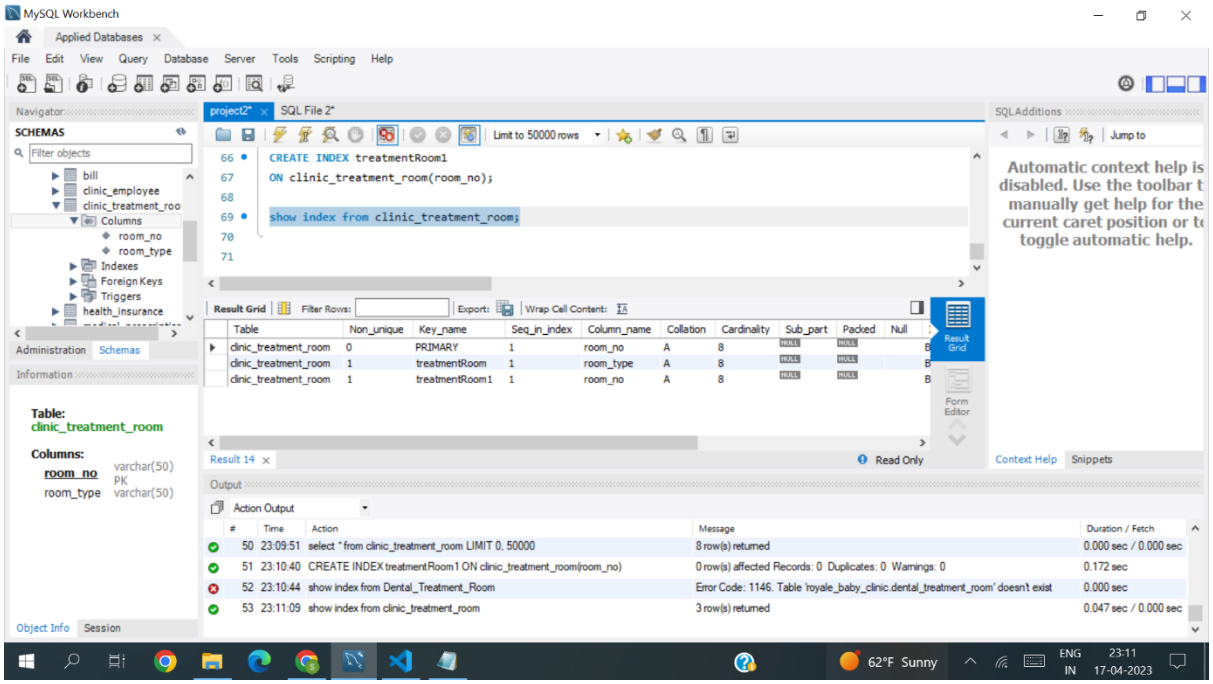
CREATE INDEX treatmentRoom ON clinic_treatment_room(room_type);



Index to recognize treatment room by room number:

CREATE INDEX treatmentRoom ON clinic_treatment_room(room_no);

Shanmukhi Baireddy
801309031

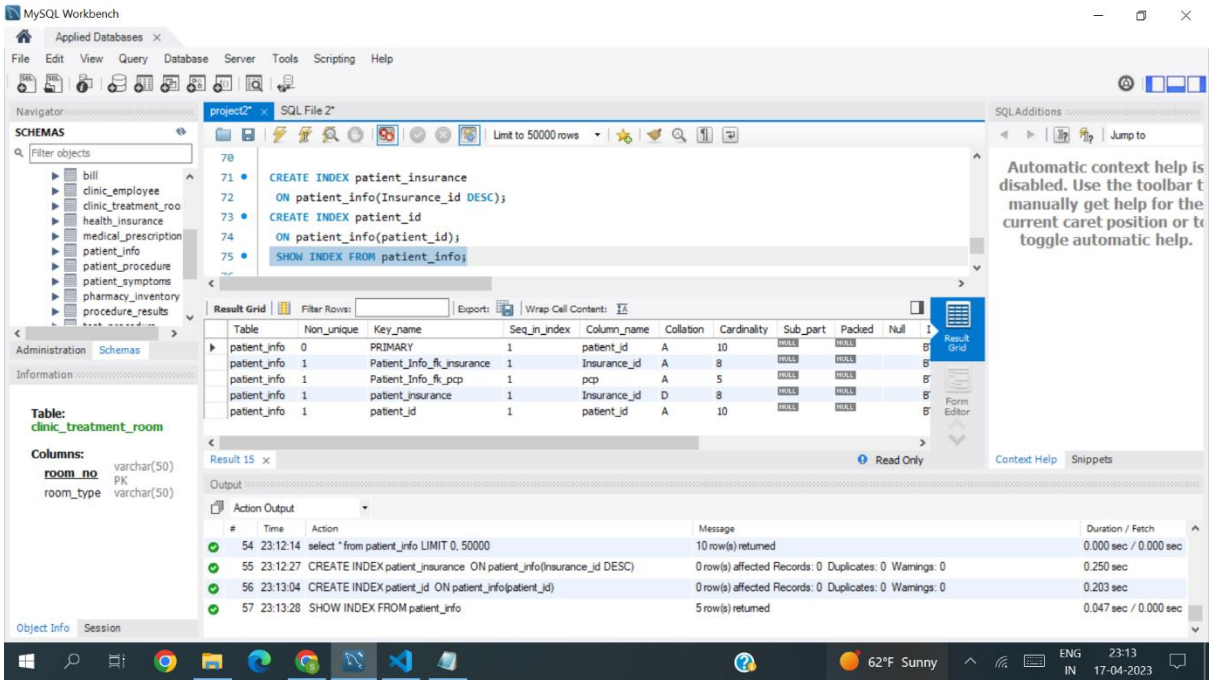


Index to recognize patient by insurance_id:

CREATE INDEX patient_insurance ON patient_info(Insurance_id DESC);

Index to recognize patient by patient id:

CREATE INDEX patient_id ON patient_info(patient_id);

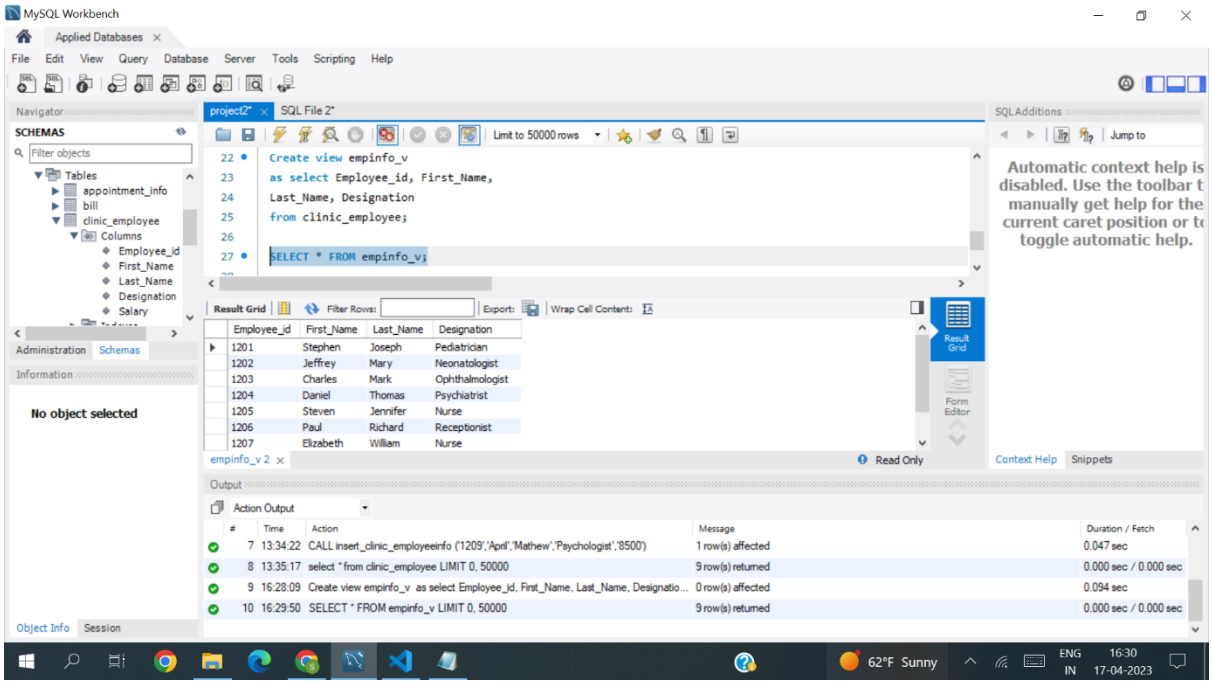


Shanmukhi Baireddy
801309031

VIEWS:

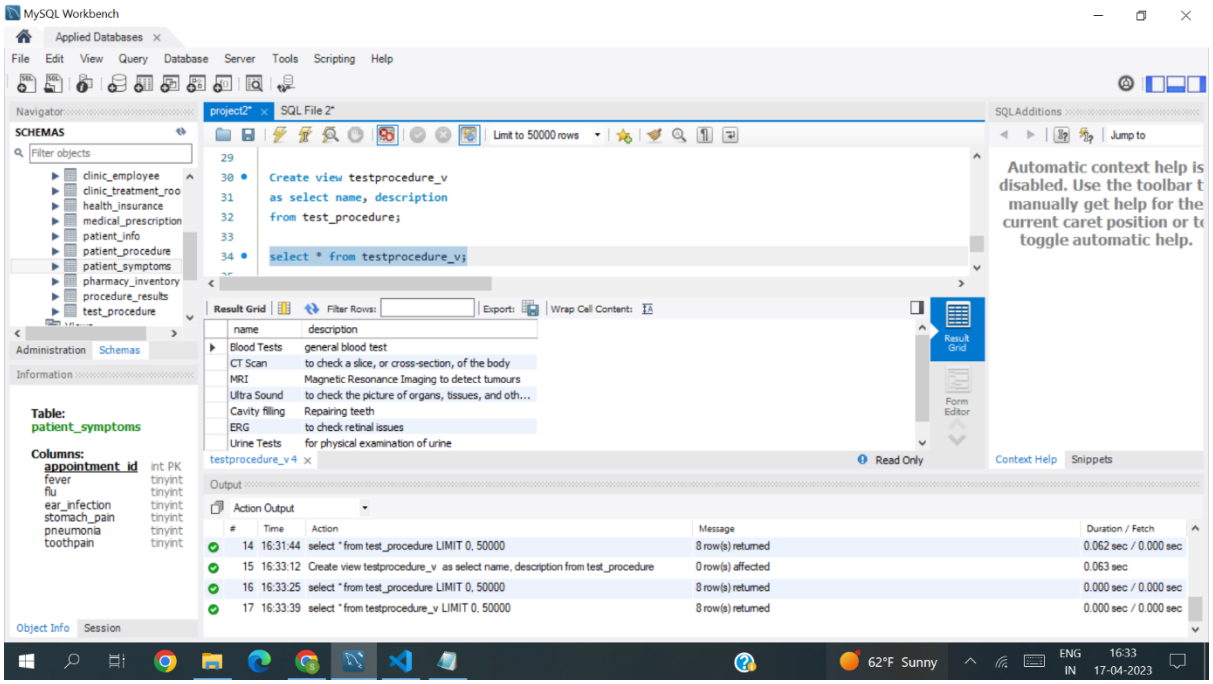
Creating a view to get the employee id, first name, last name and designation of the employee:

Create view empinfo_v as select Employee_id,First_Name, Last_Name, Designation from clinic_employee;



Creating a view to get the name and description of the test procedure:

Create view testprocedure_v as select name, description from test_procedure;



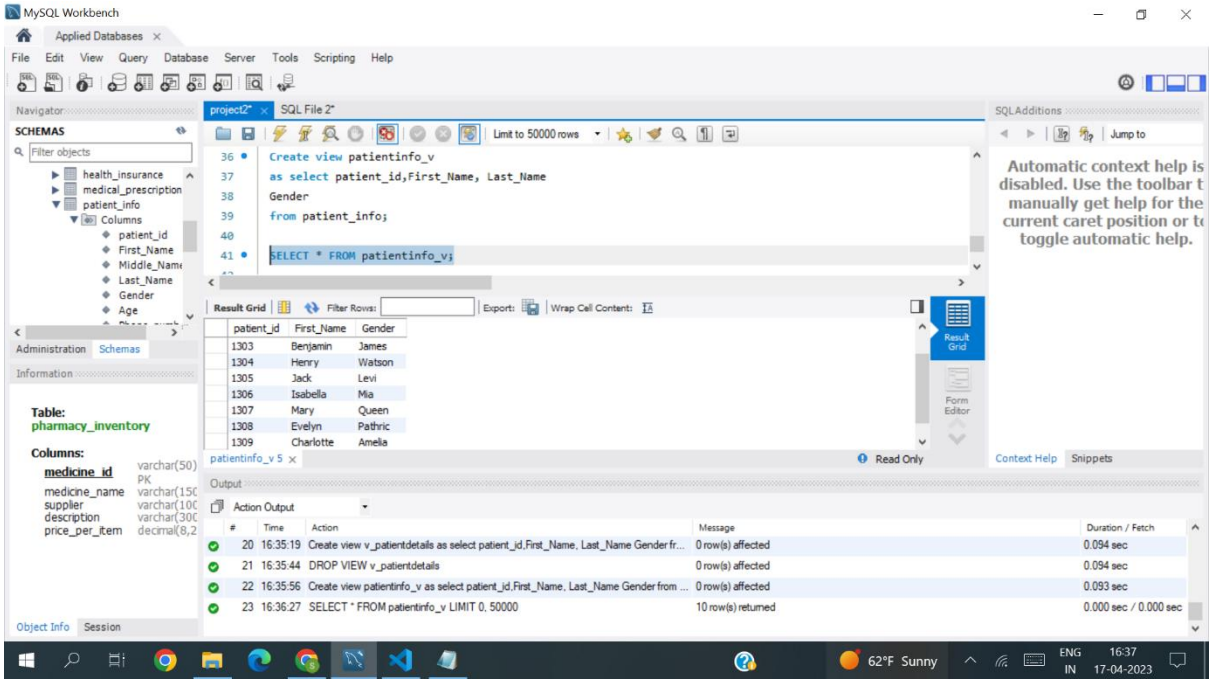
Shanmukhi Baireddy
801309031

Creating view to get the id, first name, last name and gender of the patient:

Create view patientinfo_v

as select patient_id, First_Name, Last_Name, Gender

from patient_info;

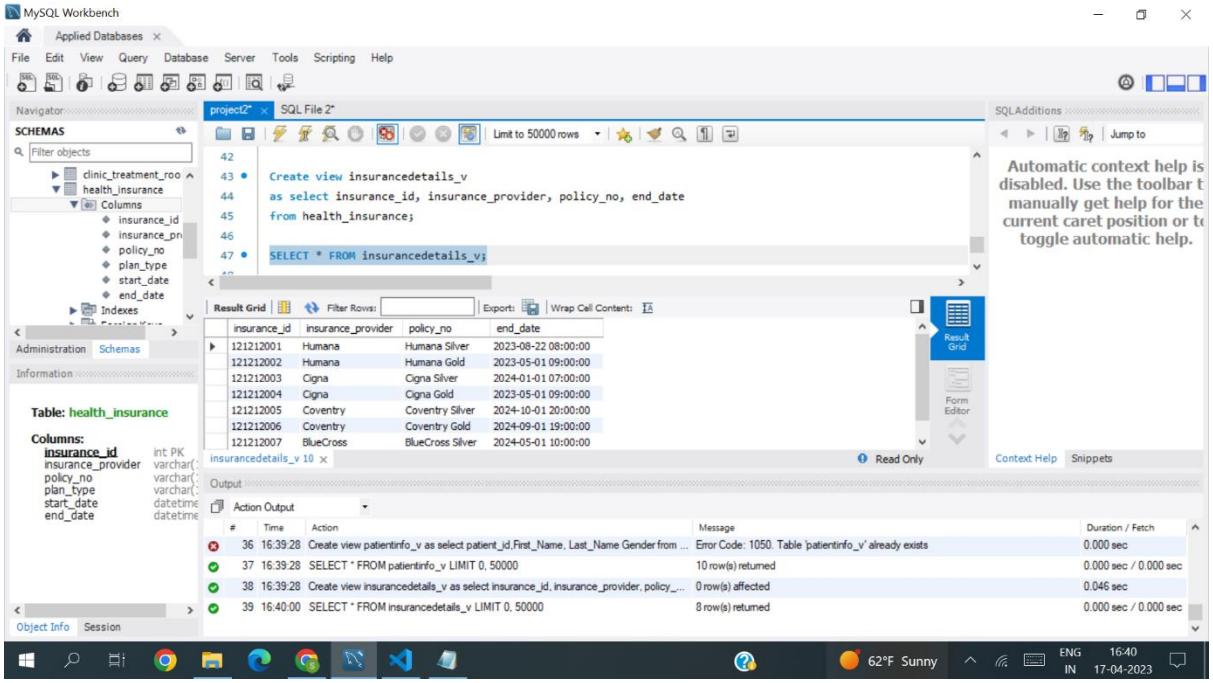


Creating a view to get the id, policy provider, policy number and end date of health insurance:

Create view insurancedetails_v as

select insurance_id, insurance_provider, policy_no, end_date

from health_insurance;

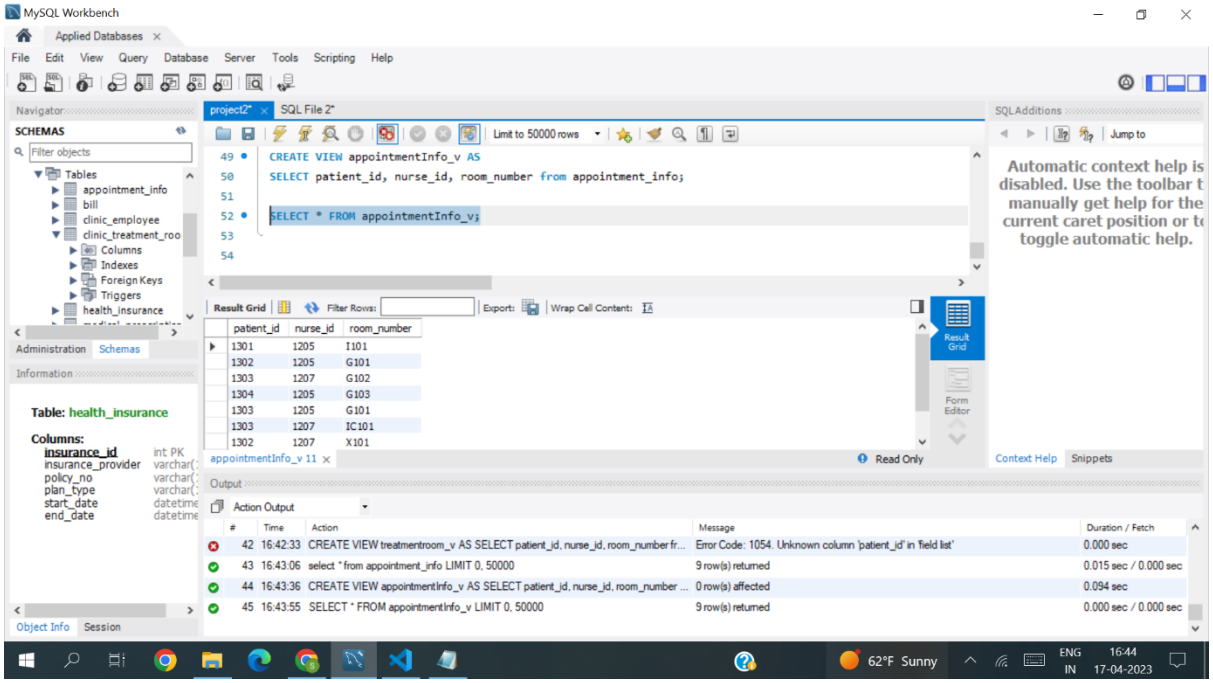


Shanmukhi Baireddy
801309031

Creating view to get patient id, nurse id and room number of an appointment:

Create view appointmentInfo_v as

select patient_id, nurse_id, room_number from appointment_info;



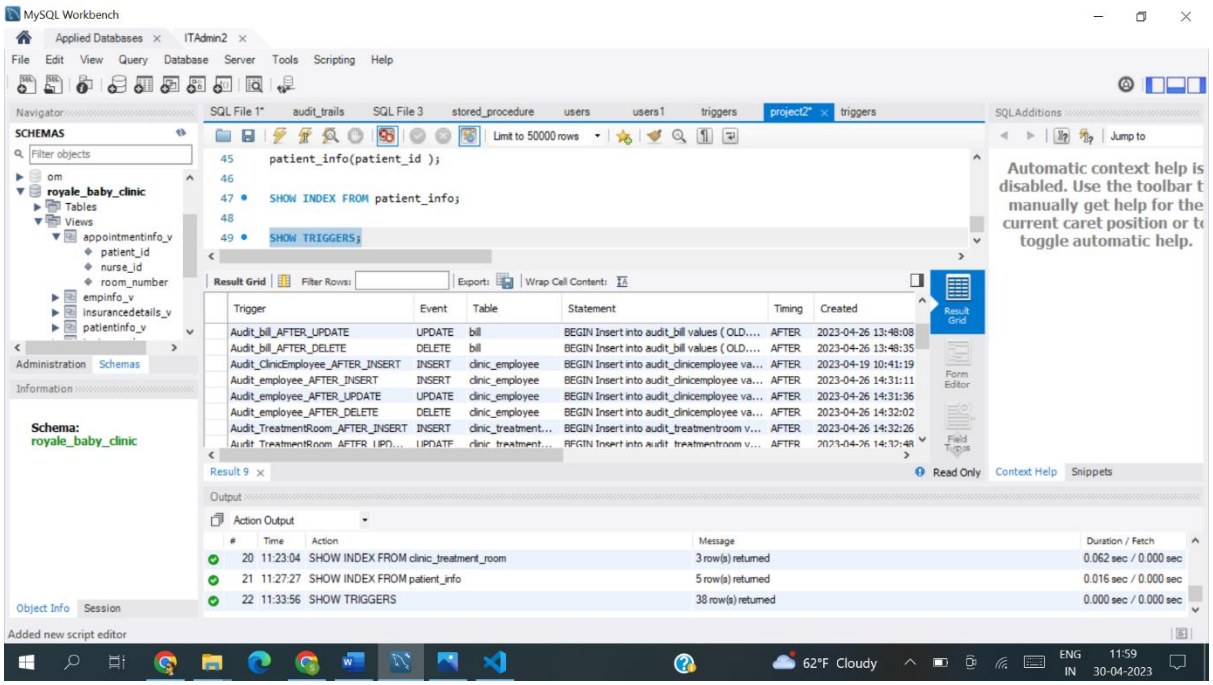
TRIGGERS:

Created triggers for all tables.

Audit_Appoinment_AFTER_INSERT
Audit_Appointment_AFTER_UPDATE
Audit_Appointment_AFTER_DELETE
Audit_Bill_AFTER_INSERT
Audit_Bill_AFTER_UPDATE
Audit_Bill_AFTER_DELETE
Audit_employee_AFTER_INSERT
Audit_employee_AFTER_UPDATE
Audit_employee_AFTER_DELETE
Audit_TreatmentRoom_AFTER_INSERT
Audit_TreatmentRoom_AFTER_UPDATE
Audit_TreatmentRoom_AFTER_DELETE
Audit_HealthInsurance_AFTER_INSERT
Audit_HealthInsurance_AFTER_UPDATE
Audit_HealthInsurance_AFTER_DELETE
Audit_medicalprescription_AFTER_INSERT
Audit_medicalprescription_AFTER_UPDATE
Audit_medicalprescription_AFTER_DELETE
Audit_patientinfo_AFTER_INSERT

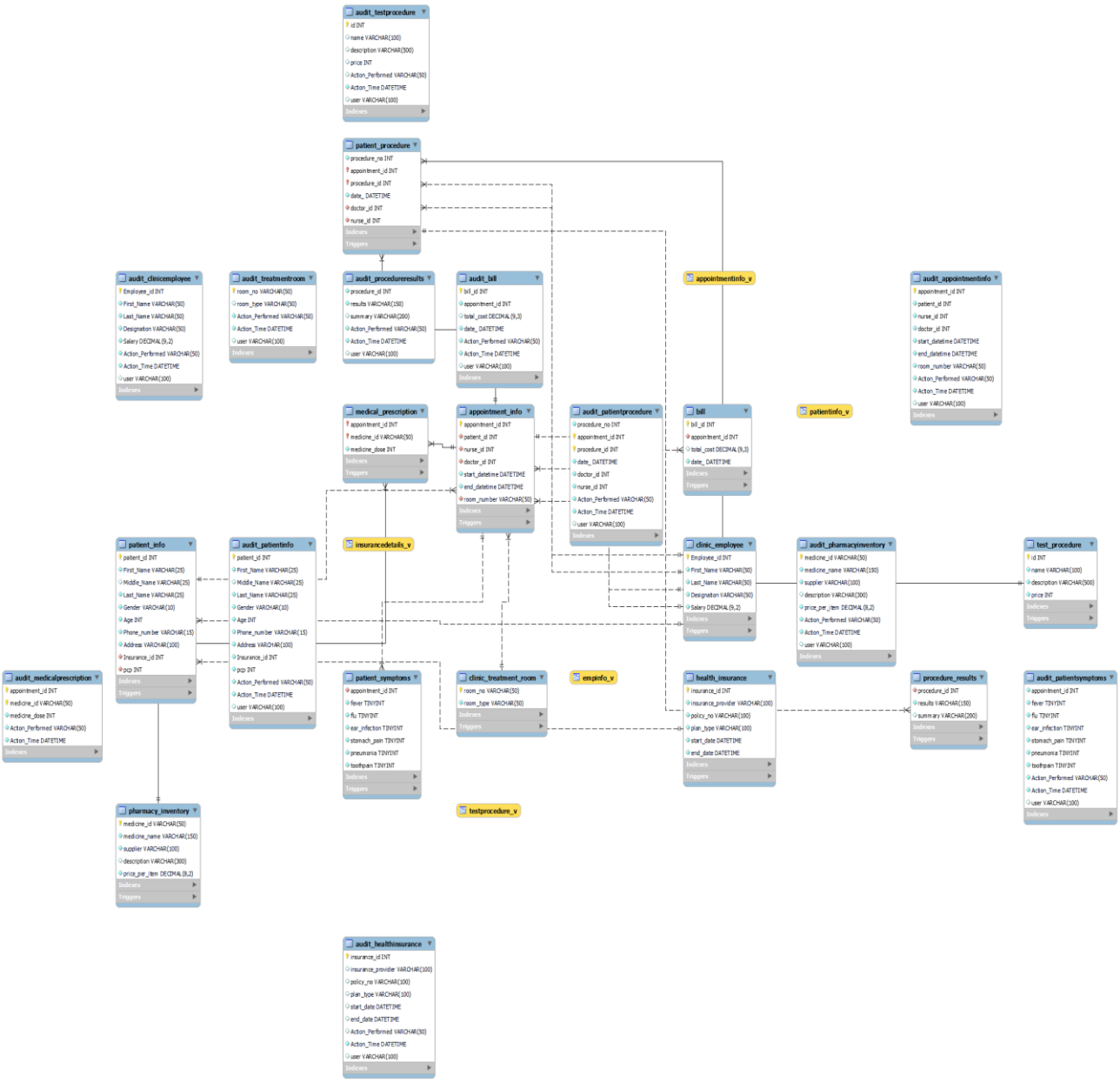
Shanmukhi Baireddy
801309031

Audit_patientinfo_AFTER_UPDATE
Audit_patientinfo_AFTER_DELETE
Audit_patientprocedure_AFTER_INSERT
Audit_patientprocedure_AFTER_UPDATE
Audit_patientprocedure_AFTER_DELETE
Audit_patientsymptoms_AFTER_INSERT
Audit_patientsymptoms_AFTER_INSERT
Audit_patientsymptoms_AFTER_UPDATE
Audit_patientsymptoms_AFTER_DELETE
Audit_pharmacyinventory_AFTER_INSERT
Audit_pharmacyinventory_AFTER_UPDATE
Audit_pharmacyinventory_AFTER_DELETE
Audit_procedureresults_AFTER_INSERT
Audit_procedureresults_AFTER_UPDATE
Audit_procedureresults_AFTER_DELETE
Audit_testprocedure_AFTER_INSERT
Audit_testprocedure_AFTER_UPDATE
Audit_testprocedure_AFTER_DELETE



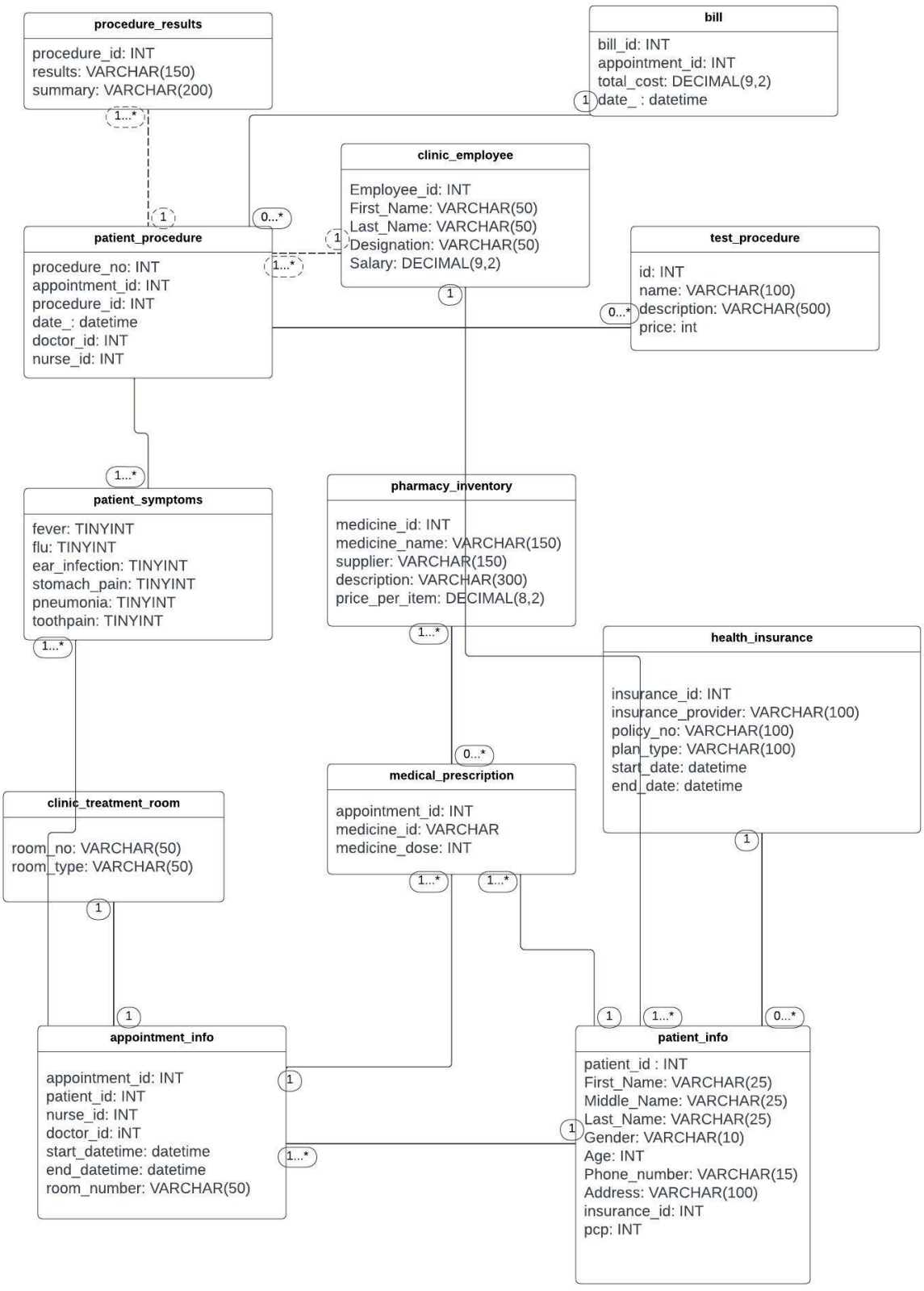
Shanmukhi Baireddy
801309031

ER diagram:



Shanmukhi Baireddy
801309031

UML Diagram:



Shanmukhi Baireddy
801309031

- All tables in database are in BCNF form as all the values in the tables are atomic with unique column names and data in each column belong to same domain. The tables don't have any partial dependencies or transitive dependencies. All the tables satisfy the third normal form.
- For example, if we take a table named medical_prescription it has attributes appointment_id, medicine_id and medicine_dose. Here, appointment_id and medicine_id together act as primary or candidate key. So, appointment_id and medicine_id together will lead to identify the medicine_dose. The medicine_dose itself will not lead to either of medicine_id or appointment_id. That means no prime attribute is dependent on non-prime attribute.

References:

Murach MySQL TextBook, Course lectures, Python information on google.