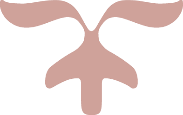


APPLIED DATABASE PROJECT 2

Advanced Features of an EMR Database System



## DECEMBER 11, 2019

ITIS 6120 - Applied Databases

**by**

Abhimanyu Singh Bisht

# Scope

Develop and maintain database system that serves as the backend of a simple EMR system that captures a small portion of relevant information of student clinic of UNCC university. Gather information which is digitally equivalent to paper records. Utilizing the gathered data for the treatment provided and maintaining medical history about the patient.

# Requirements Specification

The purpose of the Electronic Medical Records (EMRs) database is the systematized collection of patient health information and provider information in a digital format that can be created, gathered, managed, and consulted by authorized clinicians. By using this system, the outcome will be more predictable, the process will be more efficient, and the overall healthcare system will be more effective.

This project is in continuation of the previous work done. In the advance version of Electronic Medical Records (EMRs) database we have added additional features to facilitate more robust and efficient application development and to support security, privacy, audit trail and other regulatory requirements of an electronic medical records.

This document provides insight into additional tables, columns, stored procedures, triggers, indexes, views and other database functions that fulfill the new requirements.

For database security that seeks methods to protect stored data from intrusions, improper modifications and unauthorized disclosure of private information of patients. Therefore, the Database Design is composed of user authentication only for authorized users to access the database to make the security and privacy of database effective.

In order to provide documentary evidence of the sequence of activities that have affected at any time a specific operation, procedure, or event took place like updating the address of patient, for that we have added one more feature to enable audit trail.

# Create Additional Tables

CREATE TABLE `receptionist\_information` (

`receptionist\_id` int(11) PRIMARY KEY NOT NULL AUTO\_INCREMENT,

`receptionist\_name` varchar(50) NOT NULL,

`contact\_number` varchar(50) DEFAULT NULL

);

CREATE TABLE `supplier` (

`supplier\_id` varchar(50) PRIMARY KEY NOT NULL,

`supplier\_name` varchar(50) NOT NULL,

`supplier\_address` varchar(50) DEFAULT NULL,

`contact\_number` varchar(50) DEFAULT NULL

);

CREATE TABLE `patient\_information\_audit` (

`patient\_id` int(11) NOT NULL,

`patient\_first\_name` varchar(50) NOT NULL,

`patient\_last\_name` varchar(50) NOT NULL,

`patient\_address` varchar(50) NOT NULL,

`patient\_zip\_code` varchar(20) NOT NULL,

`patient\_phone` varchar(50) NOT NULL,

`action\_type` varchar(50) NOT NULL,

`action\_by` varchar(50) NOT NULL,

`action\_date` datetime NOT NULL

);

CREATE TABLE `medicine\_audit` (

`medicine\_name` varchar(50) NOT NULL,

`medicine\_price\_USD` decimal(10,2) NOT NULL,

`supplier\_id` varchar(50) NOT NULL,

`action\_type` varchar(50) NOT NULL,

`action\_by` varchar(50) NOT NULL,

`action\_date` datetime NOT NULL

);

# Inserting Data to the Additional Tables

INSERT INTO receptionist\_information VALUES (901, 'Alyssa', 9876543210 ),

(902, 'Kevin', 6543217890),

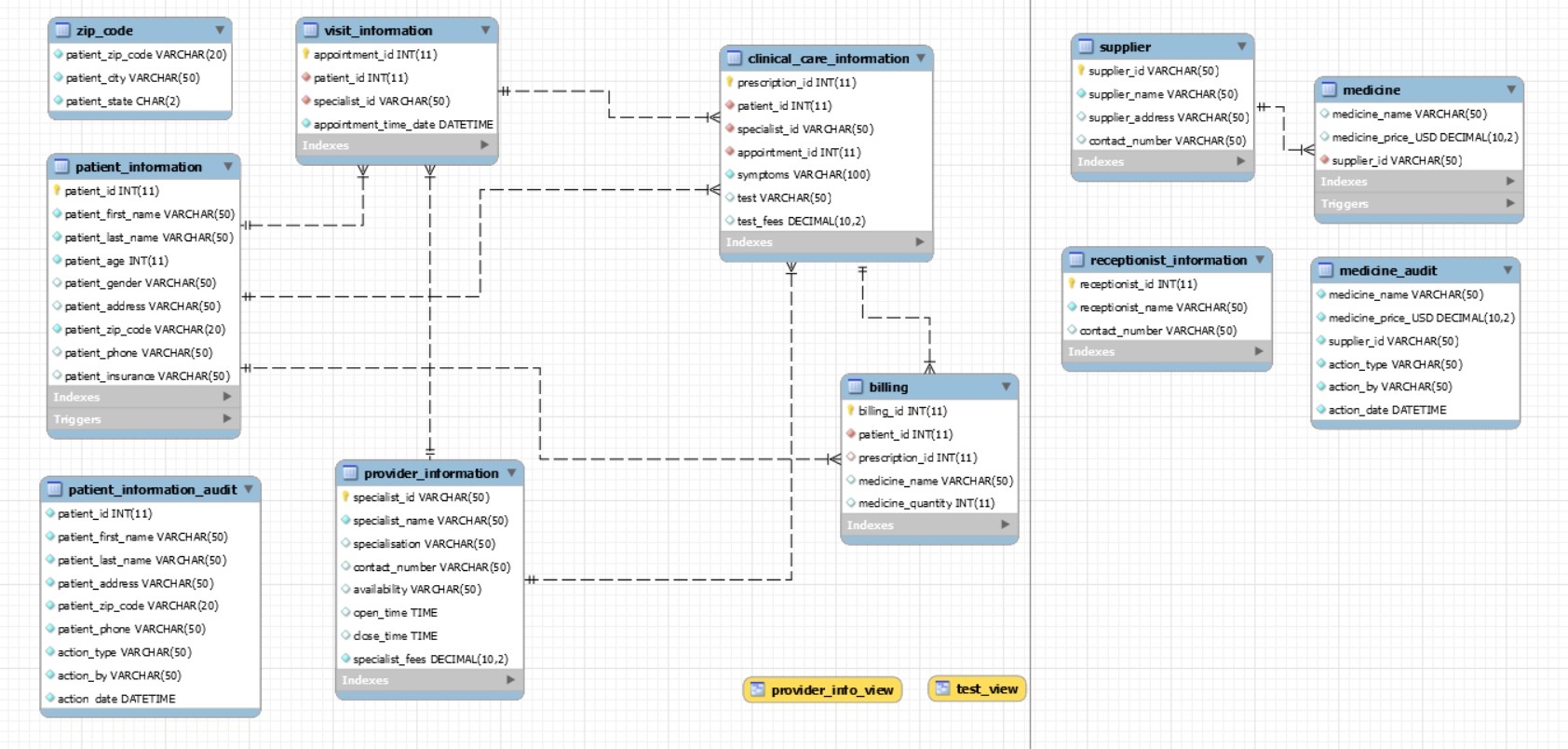
(903, 'Rishi', 3216598740 );

INSERT INTO supplier VALUES

('X', 'Get Well', 'Charlotte', 9876543210 ), ('Y', 'Wallgreens', 'Concord', 6543217890);

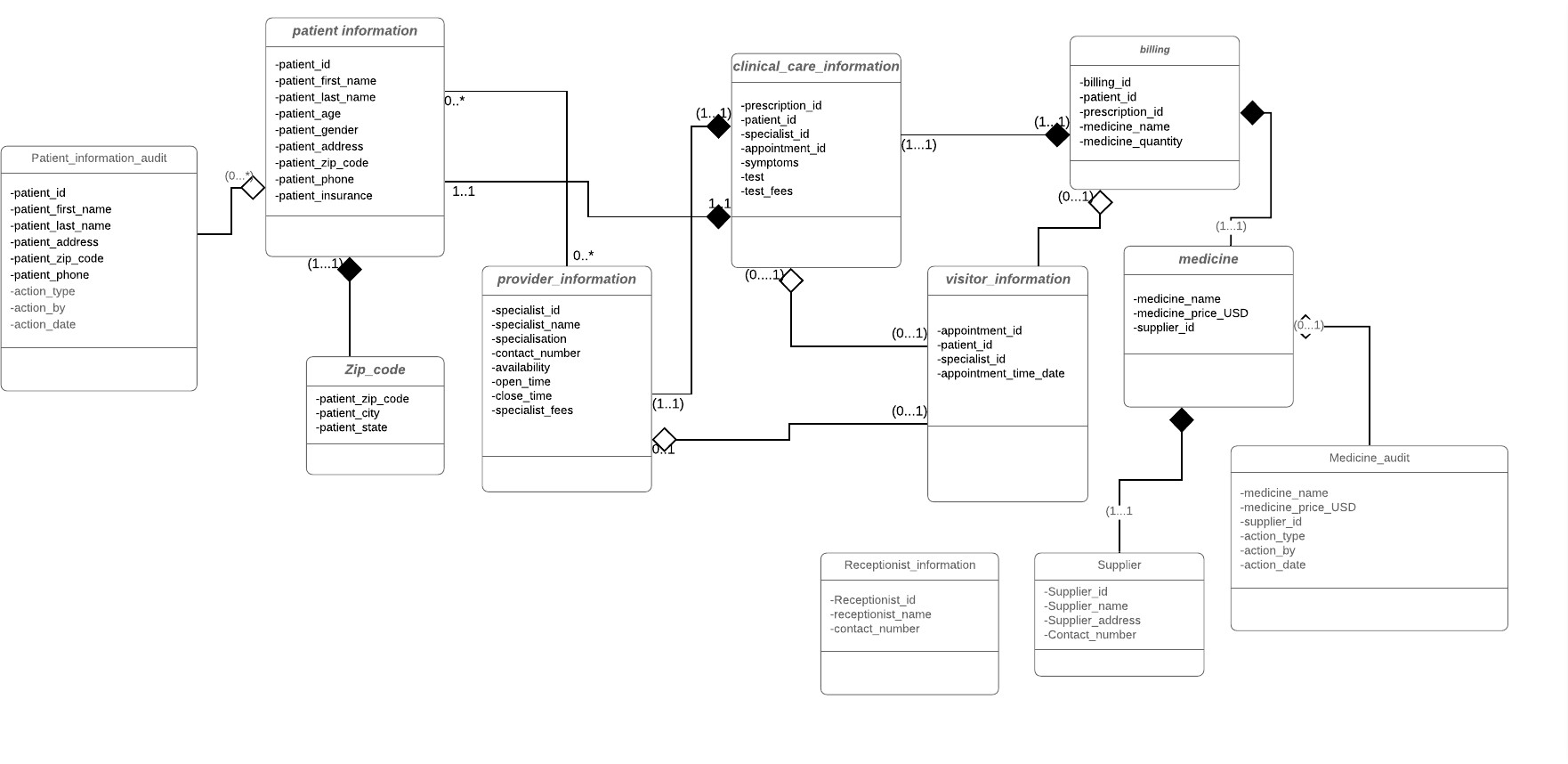
# ER DIAGRAM

An Entity Relationship Diagram (ERD) is a snapshot of data structures. An Entity Relationship Diagram shows entities (tables) in a database and relationships between tables within that database. Below is a snapshot of the database emr\_db5



# UML DIAGRAM

A UML diagram is a diagram based on the Unified Modeling Language with the purpose of visually representing a system along with its main actors, roles, actions, artifacts or classes, in order to better understand, alter, maintain, or document information about the system.



# Stored Procedure

A stored procedure is a database object that contains a block of procedural SQL code. We can use stored procedures to modify the data that's stored within a database. Using stored procedure, we can execute an insert, update and delete operations. Stored procedures are executed using CALL function.

## Select statement using procedure

Drop procedure if exists API\_patient\_information; DELIMITER //

CREATE PROCEDURE API\_patient\_information (

patient\_id\_para int

)

BEGIN

SELECT patient\_id, concat(patient\_first\_name, ' ' , patient\_last\_name) as FullName,

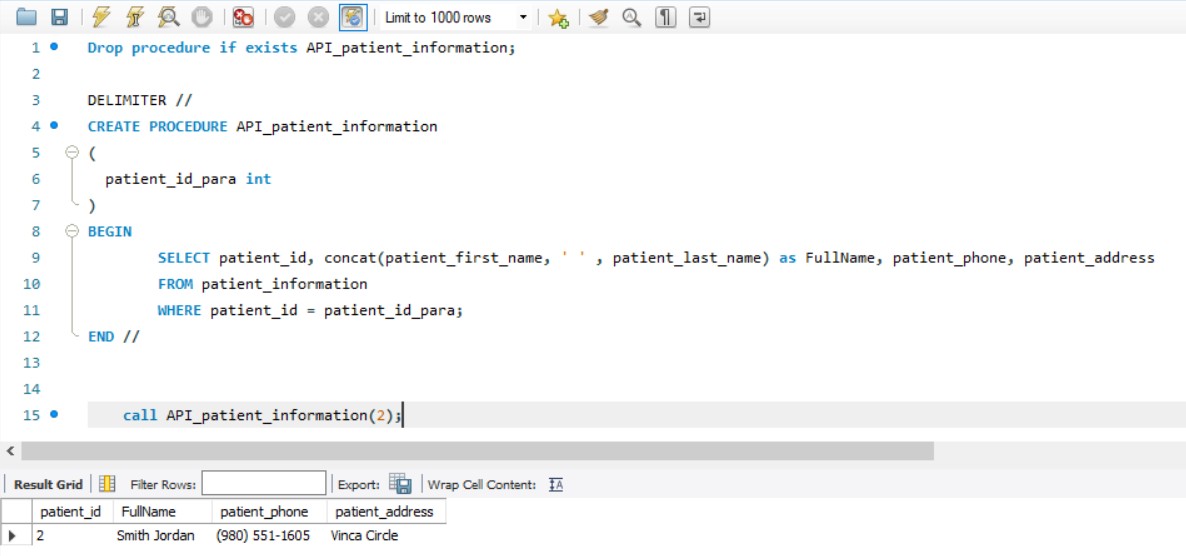
patient\_phone, patient\_address FROM patient\_information

WHERE patient\_id = patient\_id\_para; END //

## Scenario

*Select Patient Id, Patient Full Name, Patient Phone and Patient Address of the patient with id 2 using Stored Procedure and Call function.*

call API\_patient\_information(2);



## Insert statement using procedure

DROP PROCEDURE IF EXISTS API\_insert\_billing; DELIMITER //

CREATE PROCEDURE API\_insert\_billing (

billing\_id\_para int, patient\_id\_para int, prescription\_id\_para int,

medicine\_name\_para varchar(100), medicine\_quantity\_para int

)

BEGIN

DECLARE sql\_error TINYINT DEFAULT FALSE;

DECLARE CONTINUE HANDLER FOR SQLEXCEPTION

SET sql\_error = TRUE;

START TRANSACTION;

insert into billing(billing\_id,patient\_id,prescription\_id,medicine\_name,medicine\_quantity) values(billing\_id\_para,patient\_id\_para,prescription\_id\_para,medicine\_name\_para,medicine\_quantity\_para);

IF sql\_error = FALSE THEN COMMIT;

ELSE

ROLLBACK;

END IF;

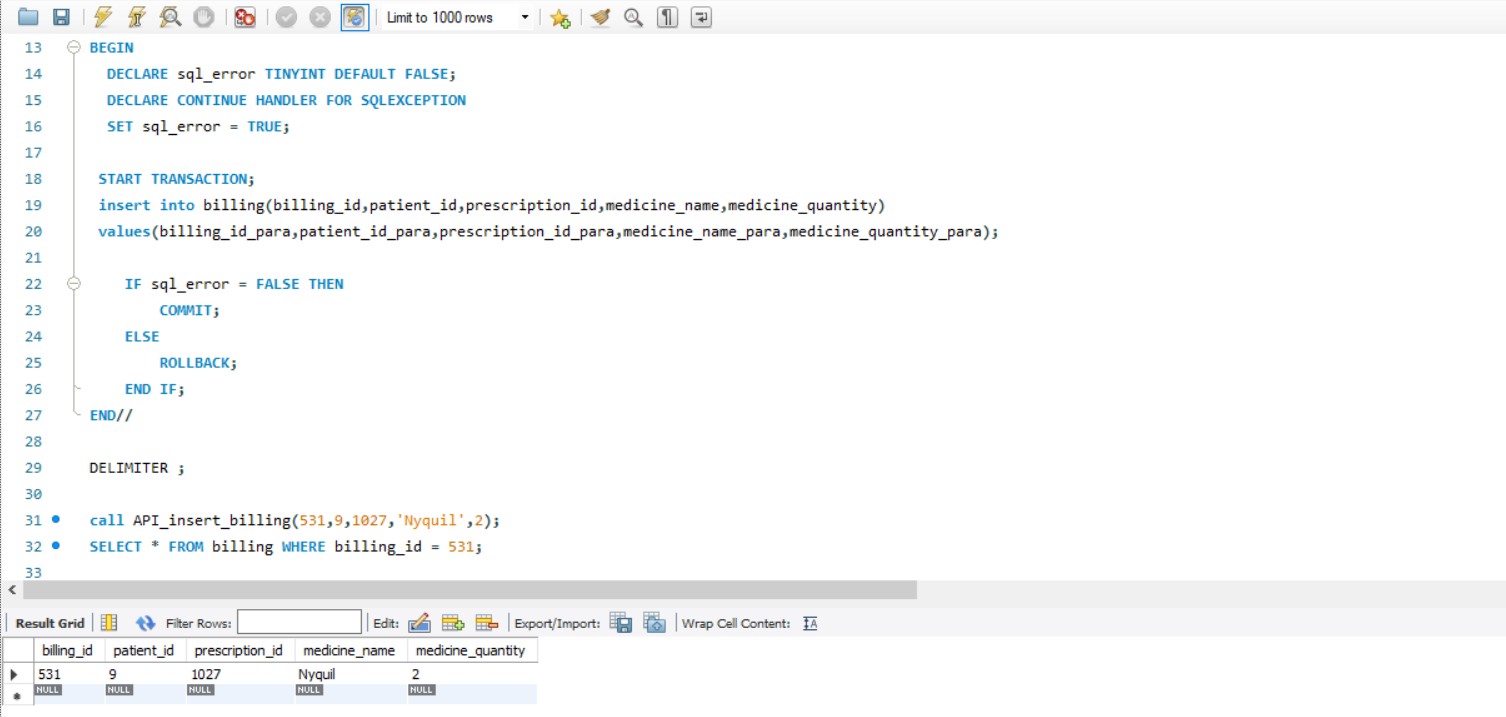
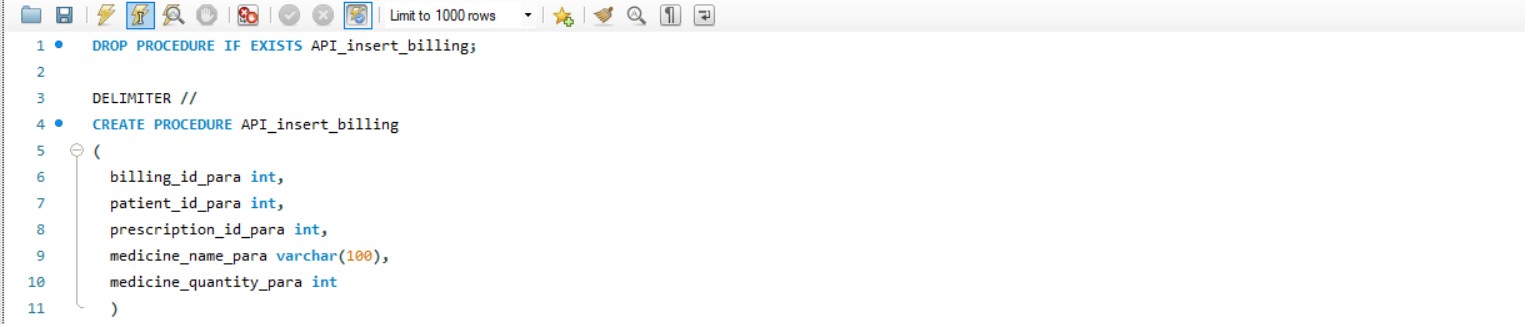
END//

DELIMITER ;

## Scenario

*Insert a new billing for patient id 9 with prescription id 1027 and 2 quantities of Nyquil medicine.*

call API\_insert\_billing(531,9,1027,'Nyquil',2); SELECT \* FROM billing WHERE billing\_id = 531;



## Update statement using procedure

DROP PROCEDURE IF EXISTS API\_update\_provider\_information; DELIMITER //

CREATE PROCEDURE API\_update\_provider\_information (

Specialist\_id\_parameter varchar(50), Contact\_number\_parameter VARCHAR(100)

)

BEGIN

DECLARE sql\_error TINYINT DEFAULT FALSE;

DECLARE CONTINUE HANDLER FOR SQLEXCEPTION

SET sql\_error = TRUE;

START TRANSACTION;

Update provider\_information

set contact\_number = contact\_number\_parameter where specialist\_id = specialist\_id\_parameter;

IF sql\_error = FALSE THEN

COMMIT;

ELSE

ROLLBACK;

END IF;

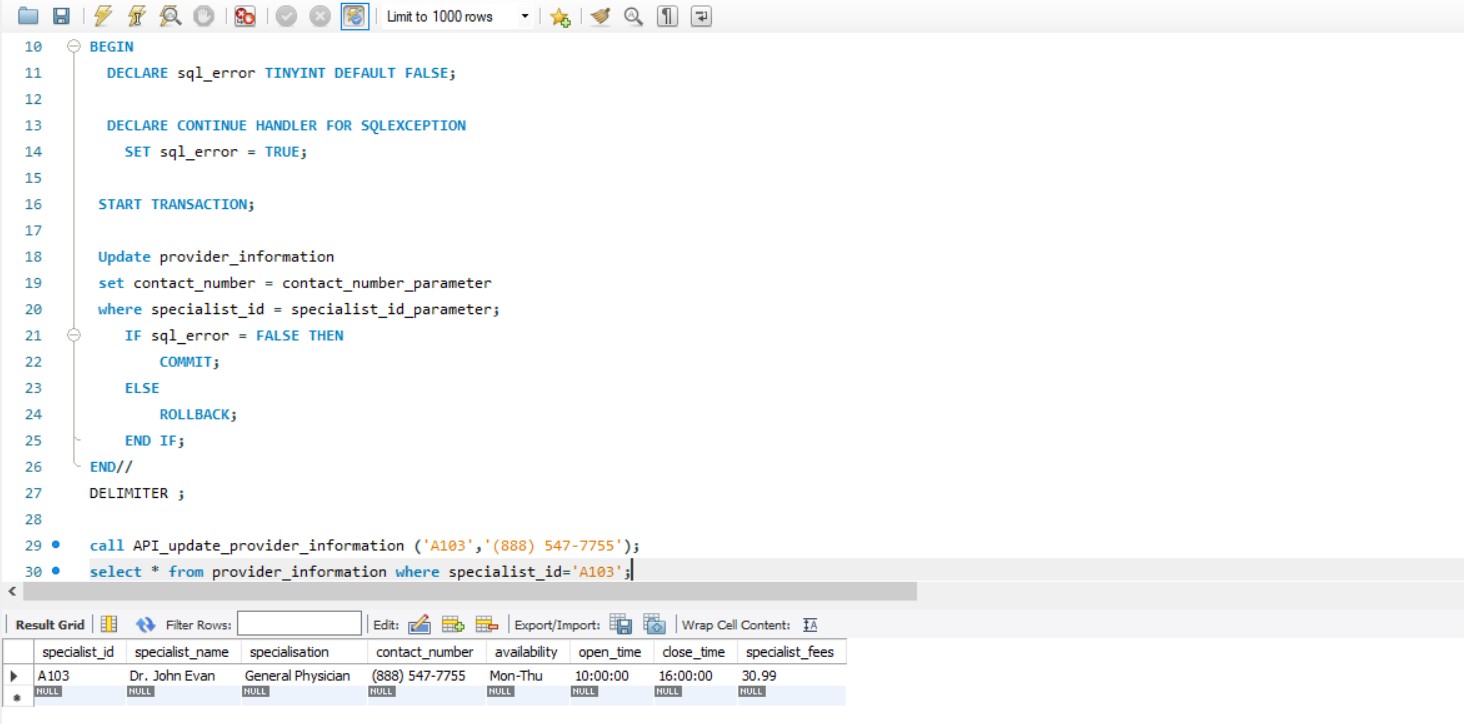
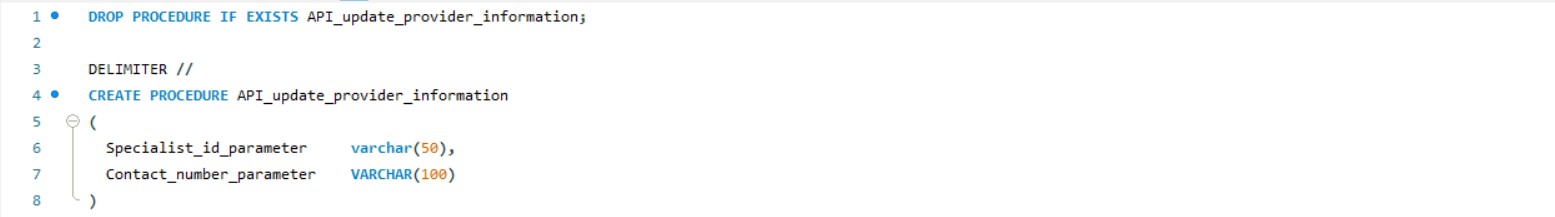
END//

DELIMITER ;

## Scenario

*Update the phone number of the specialist with id A103 to (888) 547-7755 in the provider\_information table.*

call API\_update\_provider\_information ('A103','(888) 547-7755'); select \* from provider\_information where specialist\_id='A103';



## Delete statement using procedure

DROP PROCEDURE IF EXISTS API\_delete\_medicine; DELIMITER //

CREATE PROCEDURE API\_delete\_medicine (

medicine\_name\_para varchar(100)

)

BEGIN

DECLARE sql\_error TINYINT DEFAULT FALSE;

DECLARE CONTINUE HANDLER FOR SQLEXCEPTION

SET sql\_error = TRUE; START TRANSACTION;

delete from medicine where medicine\_name = medicine\_name\_para; IF sql\_error = FALSE THEN

COMMIT;

ELSE

ROLLBACK;

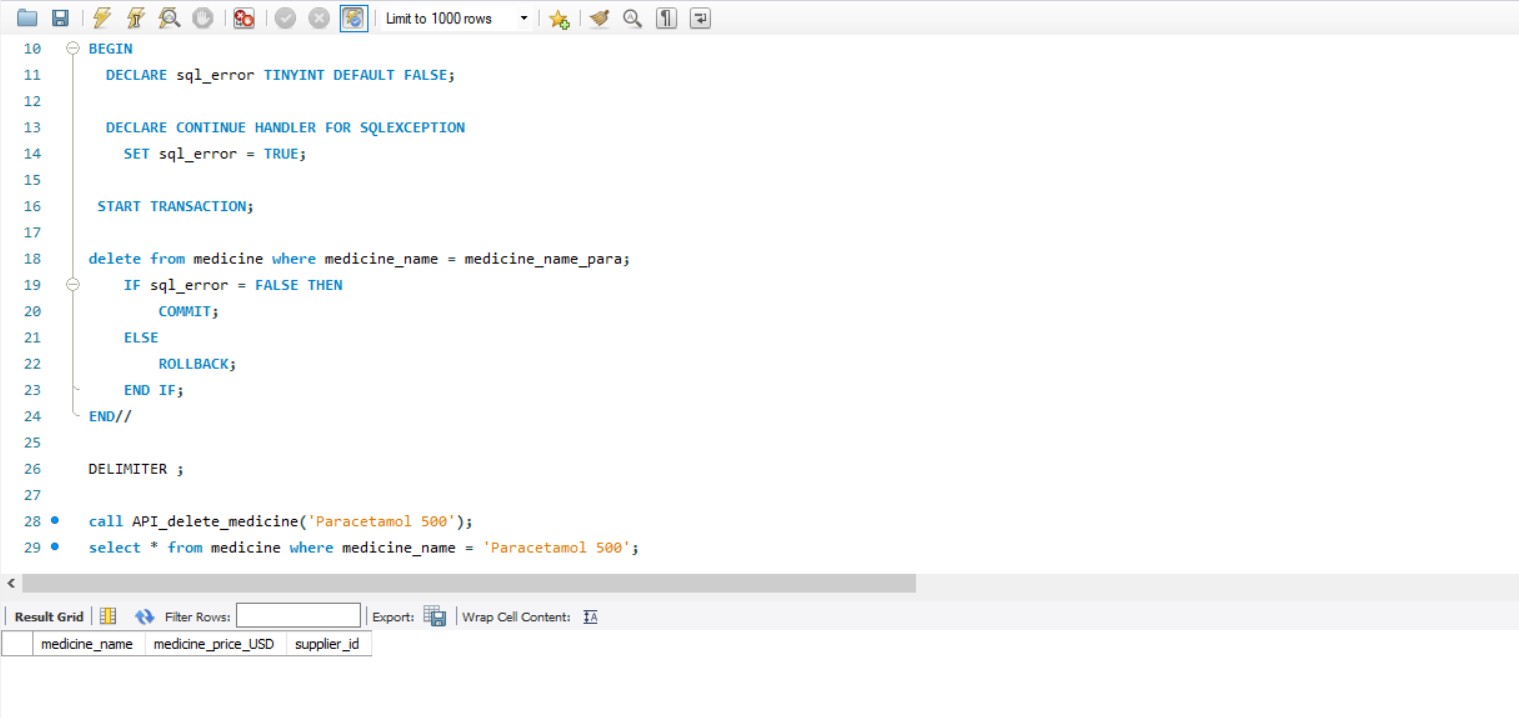
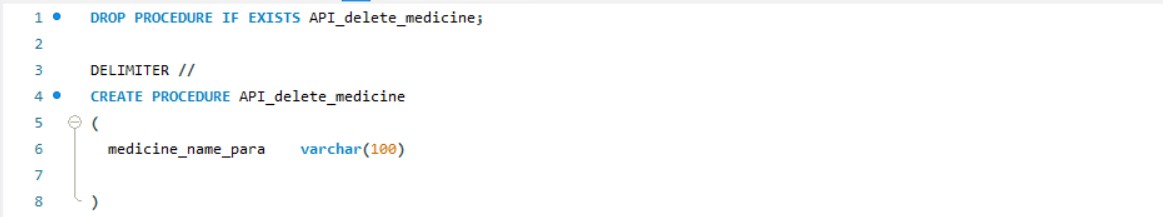
END IF;

END//

DELIMITER ;

## Scenario

*Delete Paracetamol 500 from the medicine table.*



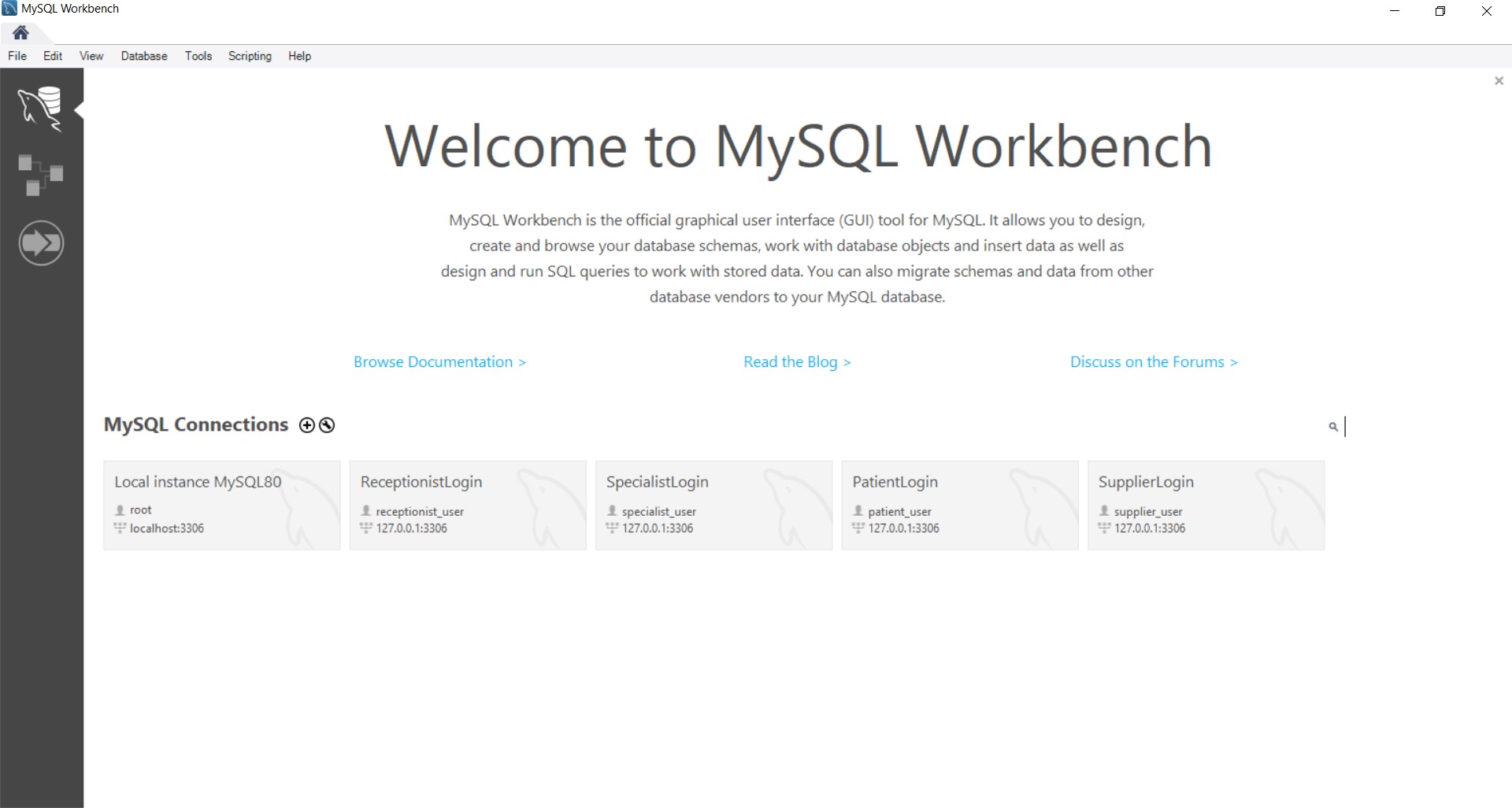
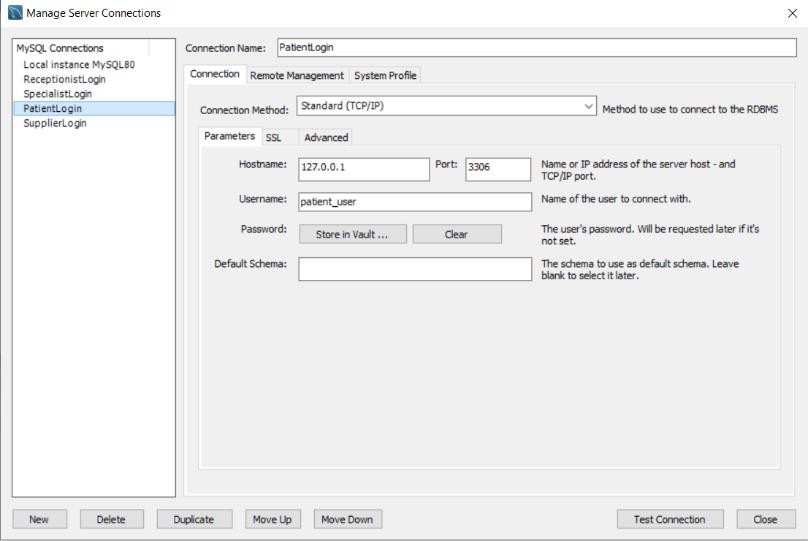
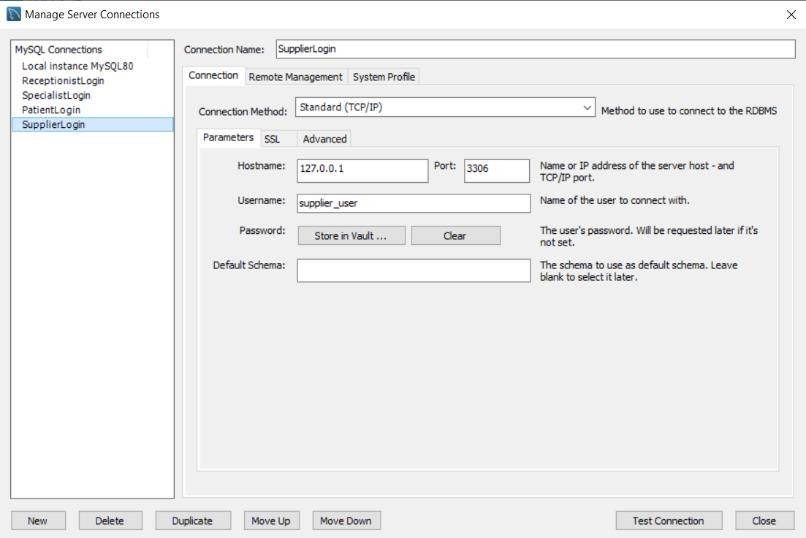
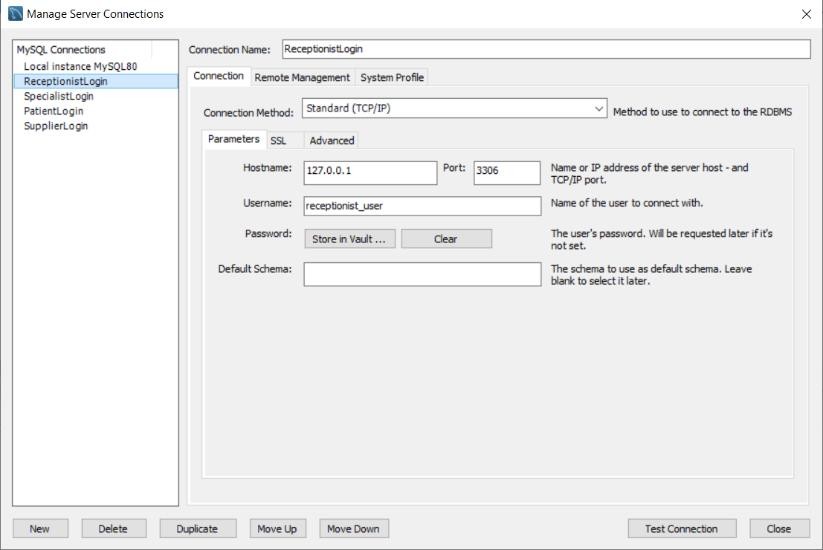
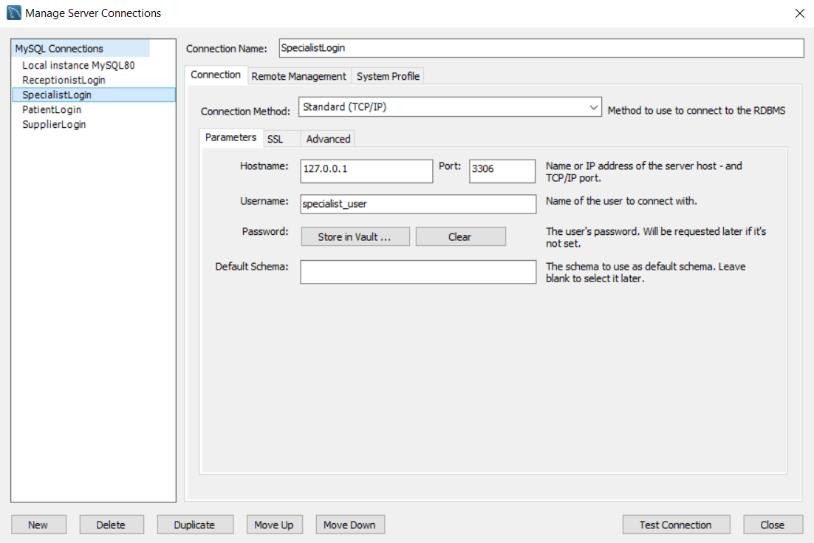
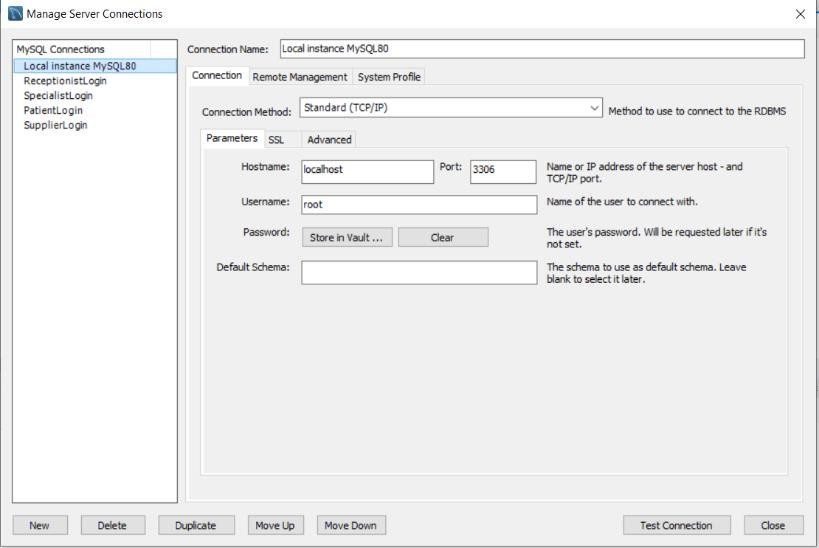
# User Authentication

Following are the five users of emr\_db5 database:

* 1. root (Administrator)
  2. specialist\_user
  3. receptionist\_user
  4. supplier\_user
  5. patient\_user

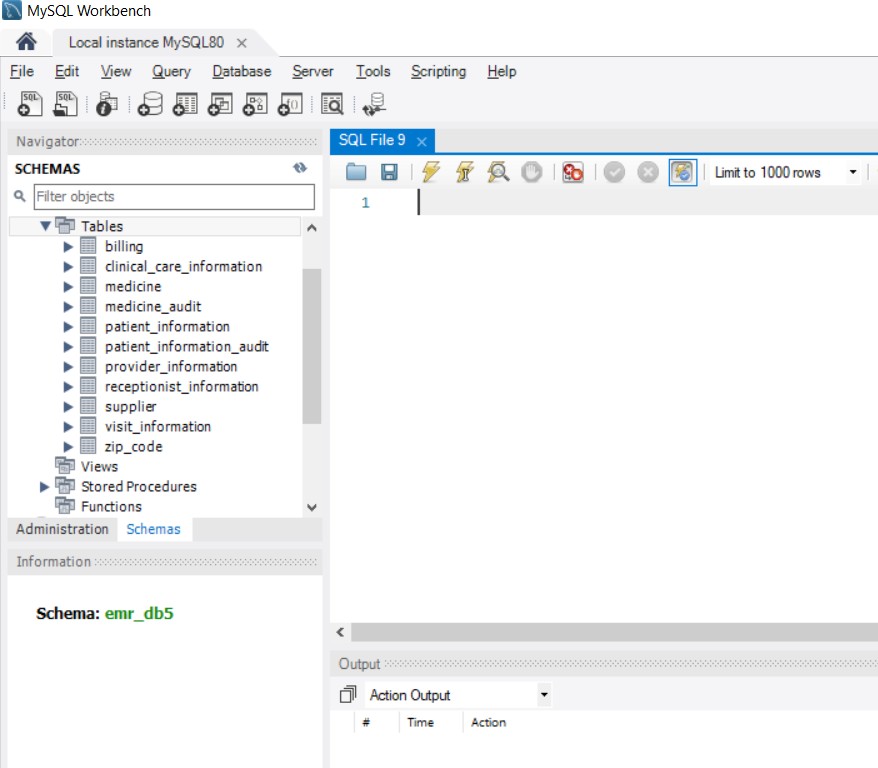
Except the root user, the other four are newly created with the given password. User Logins are also created after creating the users.

CREATE USER 'specialist\_user'@'localhost' IDENTIFIED BY 'xyz123'; CREATE USER 'receptionist\_user'@'localhost' IDENTIFIED BY '12345'; CREATE USER 'supplier\_user'@'localhost' IDENTIFIED BY 'abc123'; CREATE USER 'patient\_user'@'localhost' IDENTIFIED BY 'home123';



# Role-Based Access Control

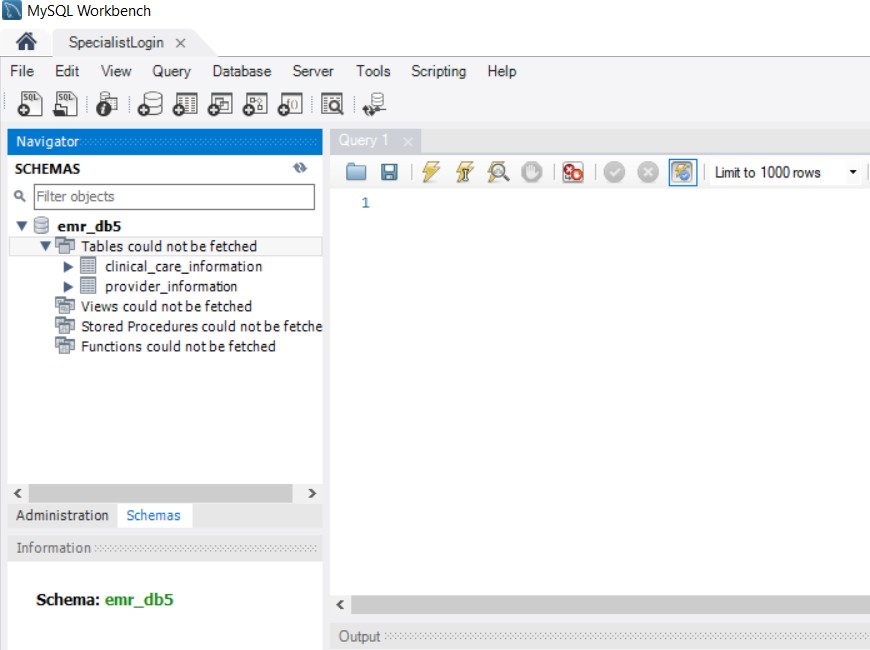
1. ‘root’ user is the admin and he/she can access all the tables of the database system. Also, the root user can do all the operations and can make changes in any of the table.



1. ‘specialist\_user’ is given read only access to provider\_information table to view all the doctors’ information, whereas it has been given read and write access to clinical\_care\_information table to view, insert, update and delete patient’s clinical details. Hence, specialist\_user cannot access any other table of the database.

GRANT SELECT ON emr\_db5.provider\_information TO 'specialist\_user'@'localhost';

GRANT SELECT, INSERT, UPDATE, DELETE ON emr\_db5.clinical\_care\_information TO 'specialist\_user'@'localhost';

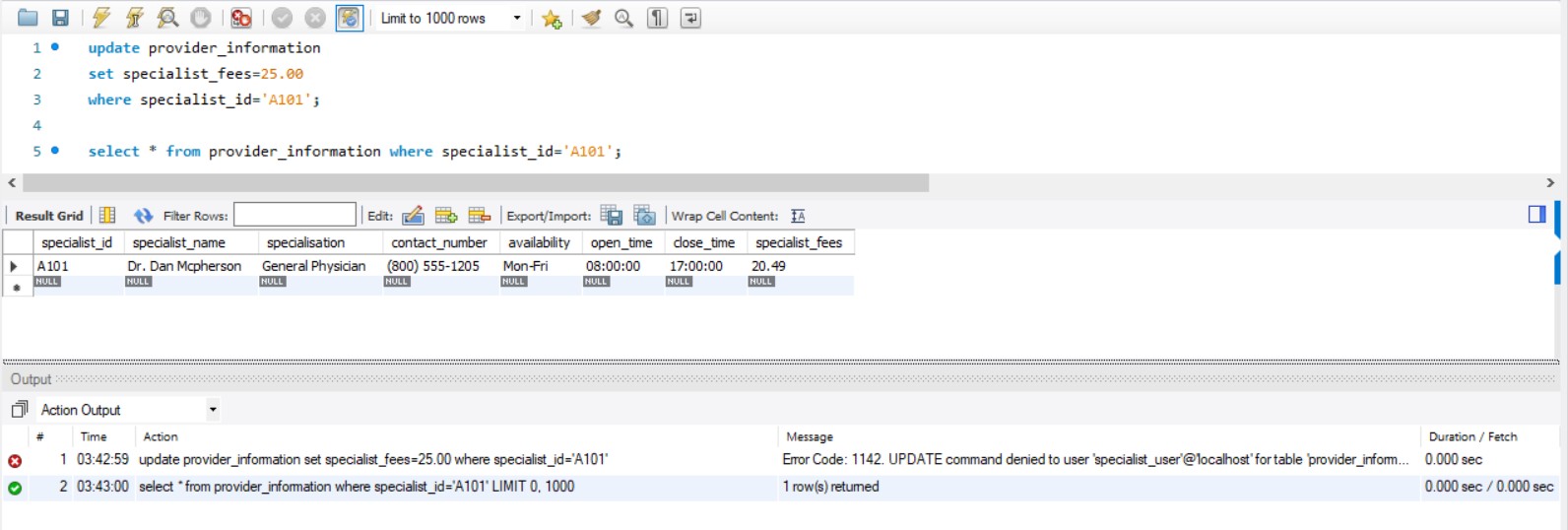


## Scenario I

*Update the specialist\_fees to 25.00 of the specialist with id A101 in the provider\_information table.*

update provider\_information set specialist\_fees=25.00 where specialist\_id='A101';

select \* from provider\_information where specialist\_id='A101';



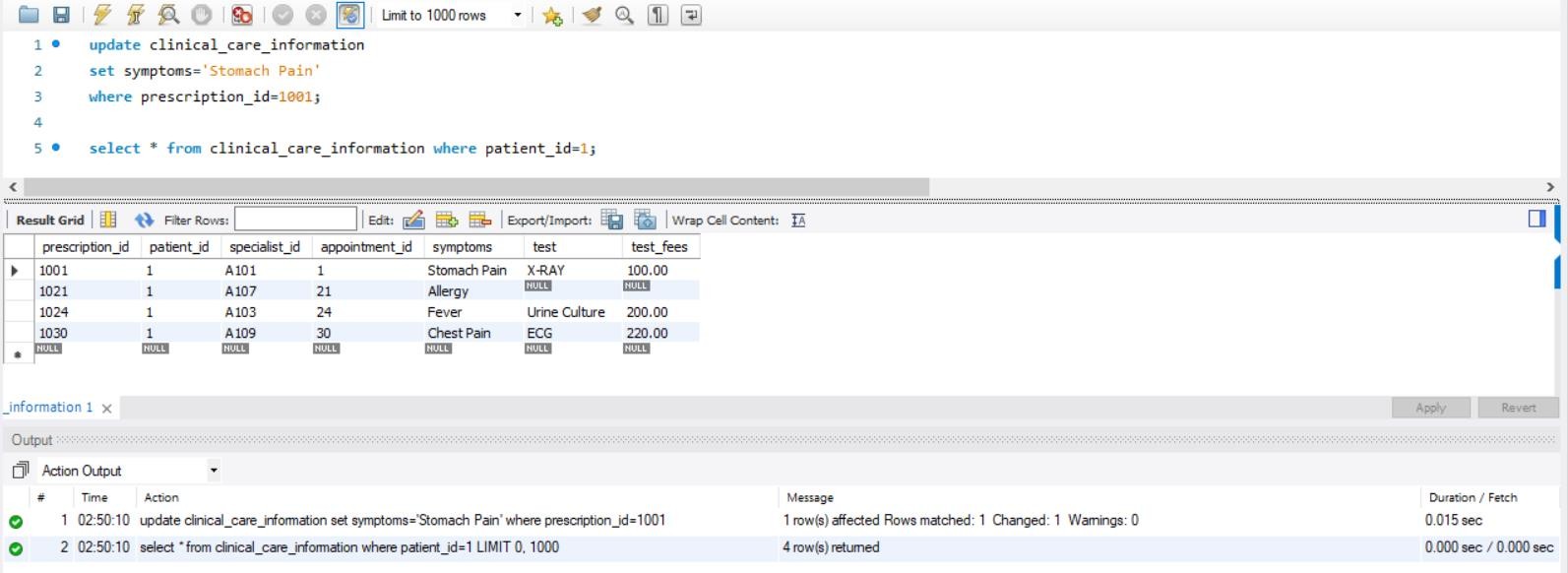
Here, the specialist\_user cannot update the provider\_information table, but he/she can view the table data.

## Scenario II

*Update symptoms to Stomach Pain of the patient with prescription id 1001 in the clinical\_care\_information table.*

update clinical\_care\_information set symptoms='Stomach Pain' where prescription\_id=1001;

select \* from clinical\_care\_information where patient\_id=1;



Here, the specialist\_user can update and view the clinical\_care\_information table.

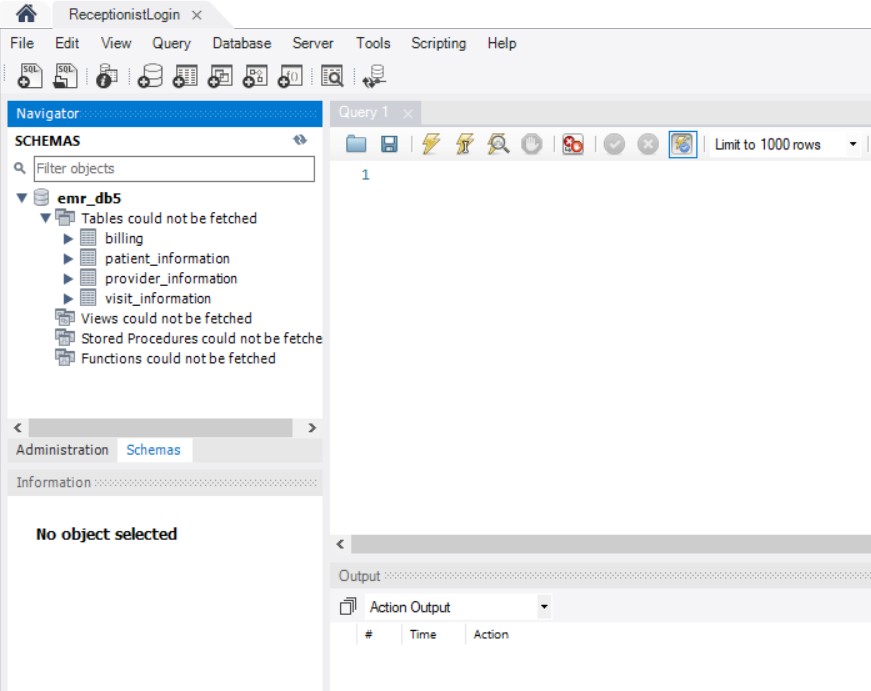
1. ‘receptionist\_user’ is given read only access to provider\_information table to view all the doctors’ information, whereas it has been given read and write access to billing, visit\_information and patient\_information table to view, insert, update and delete patient’s profile, visit and billing details. Hence, receptionist\_user cannot access any other table of the database.

GRANT SELECT ON emr\_db5.provider\_information TO 'receptionist\_user'@'localhost';

GRANT SELECT, INSERT, UPDATE, DELETE ON emr\_db5.billing TO 'receptionist\_user'@'localhost';

GRANT SELECT, INSERT, UPDATE, DELETE ON emr\_db5.visit\_information TO 'receptionist\_user'@'localhost';

GRANT SELECT, INSERT, UPDATE ON emr\_db5.patient\_information TO 'receptionist\_user'@'localhost';

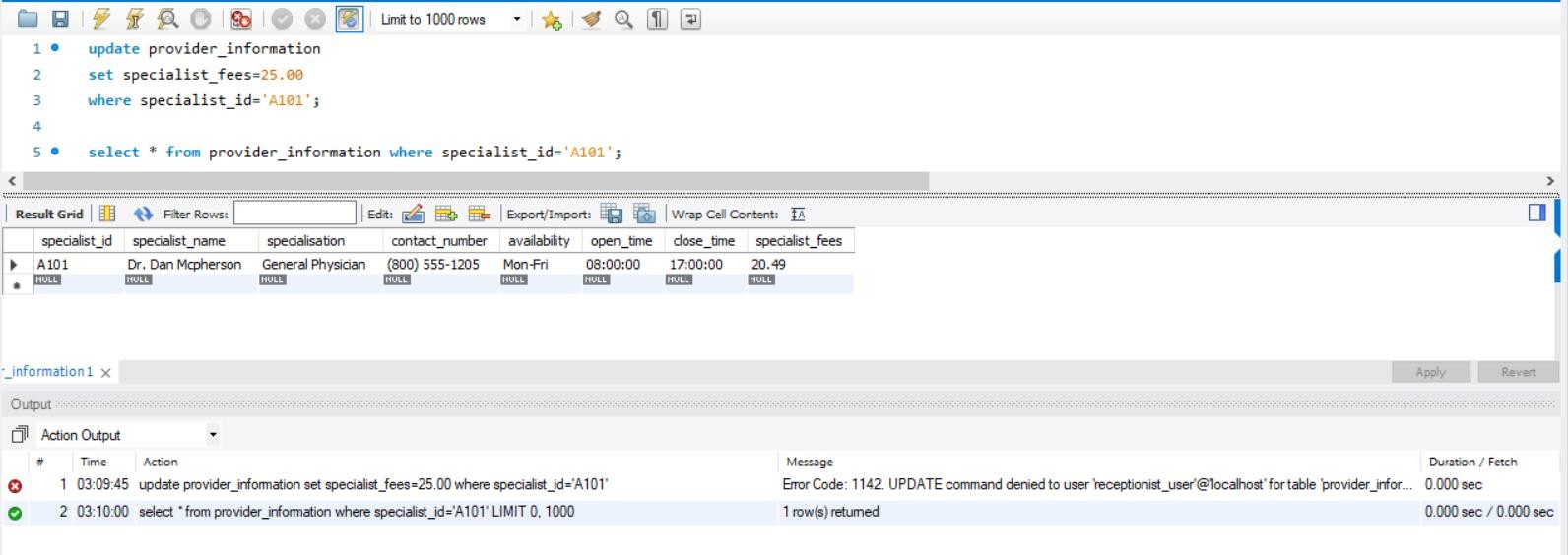


## Scenario I

*Update the specialist\_fees to 25.00 of the specialist with id A101 in the provider\_information table.*

update provider\_information set specialist\_fees=25.00 where specialist\_id='A101';

select \* from provider\_information where specialist\_id=’A101’;

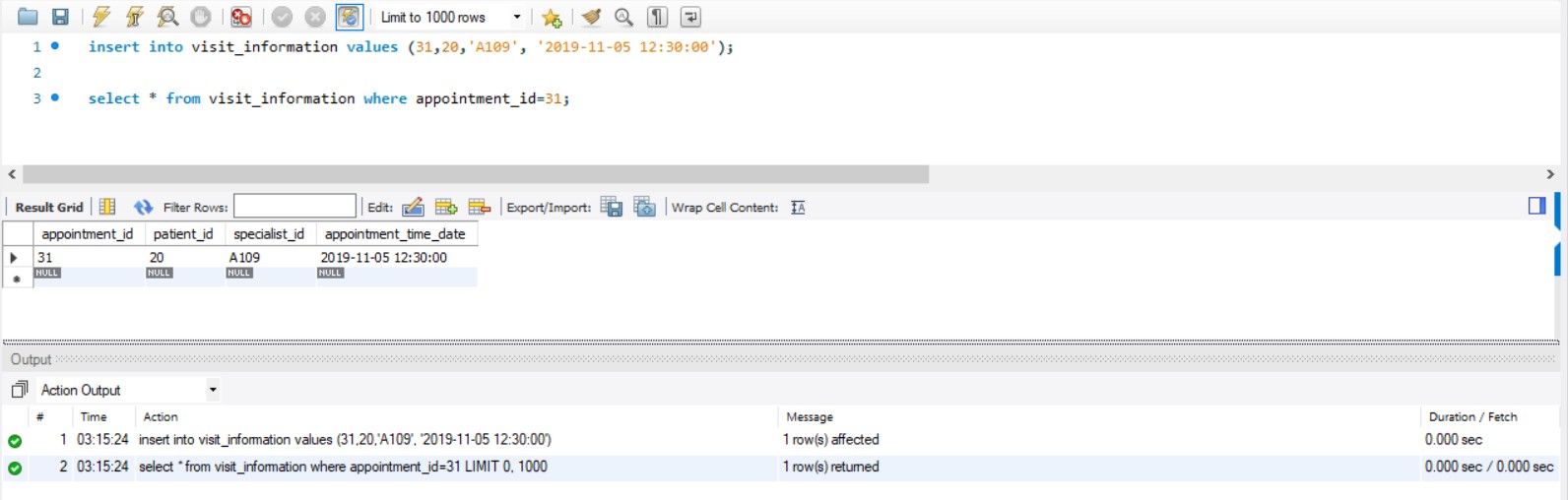


Here, the receptionist\_user cannot update the provider\_information table, but he/she can view the table data.

## Scenario II

*Insert a new entry in the visit\_information table with appointment id 31 for patient id 20 and specialist id A109 on 2019-11-05 at 12:30:00.*

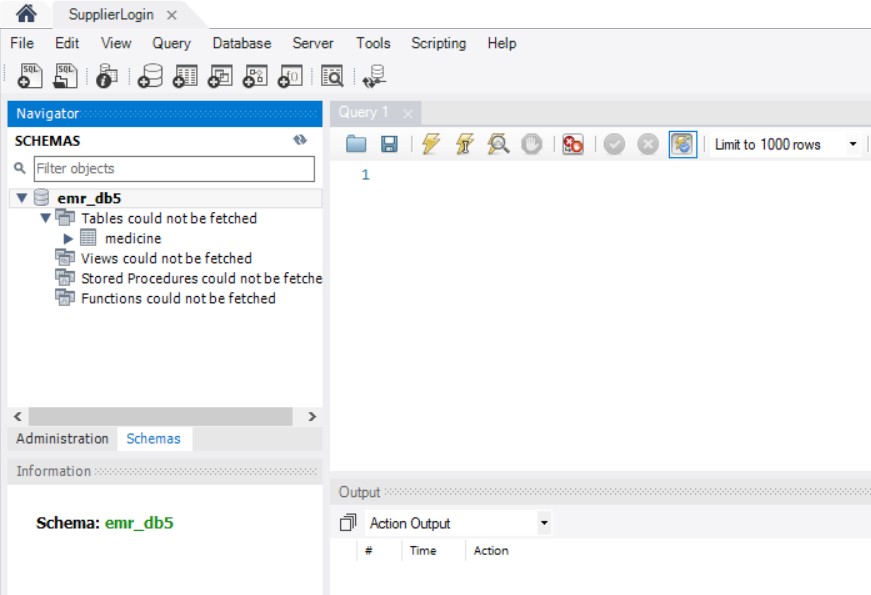
insert into visit\_information values (31,20,'A109', '2019-11-05 12:30:00'); select \* from visit\_information where appointment\_id=31;



Here, the receptionist\_user can insert and view the visit\_information table.

1. ‘supplier\_user’ has been given read and write access to medicine table to view, insert, update and delete medicine details. Hence, supplier\_user cannot access any other table of the database.

GRANT SELECT, INSERT, UPDATE, DELETE ON emr\_db5.medicine TO 'supplier\_user'@'localhost';



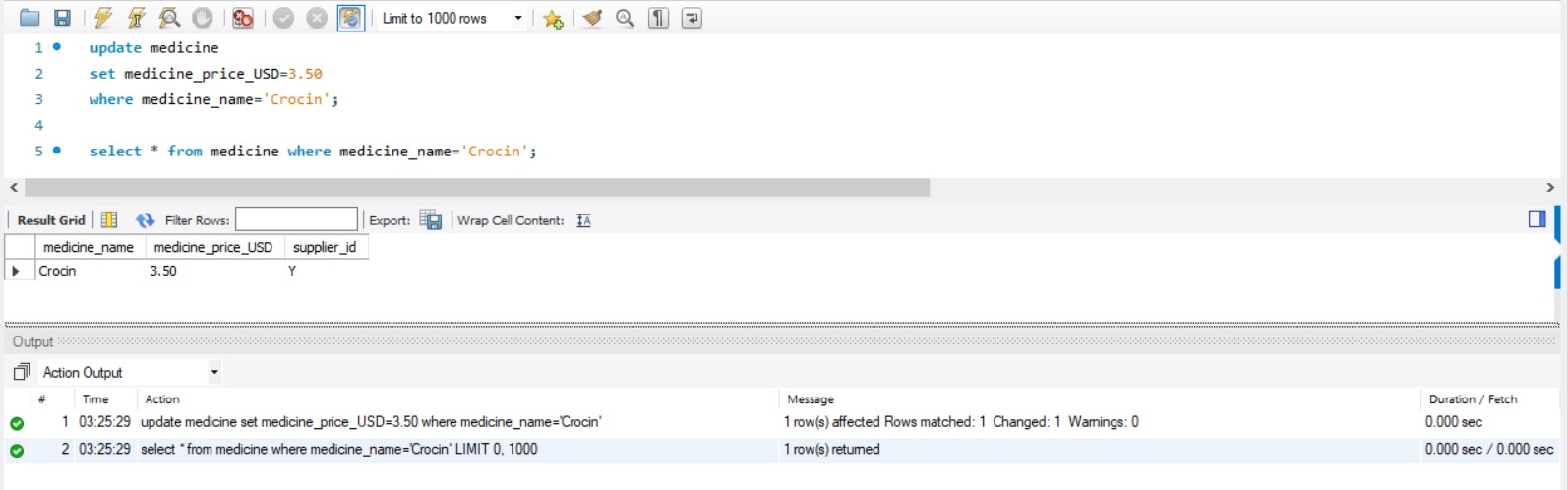
## Scenario

*Update the medicine price to 3.5 USD of Crocin in the medicine table.*

update medicine

set medicine\_price\_USD=3.50 where medicine\_name='Crocin';

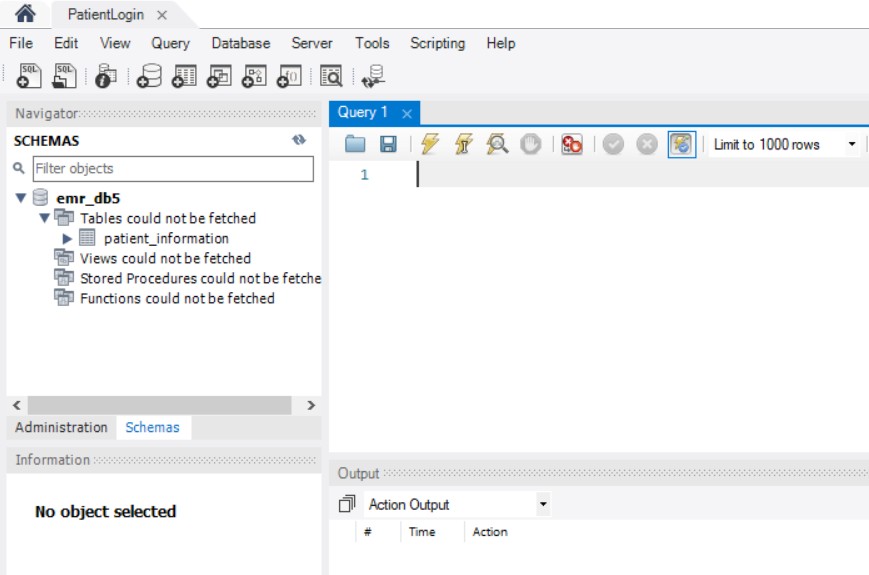
select \* from medicine where medicine\_name='Crocin';



Here, the suplier\_user can update and view the medicine table.

1. ‘patient\_user’ is given read only access to patient\_information table to view patient’s information. Hence, patient\_user cannot access any other table of the database.

GRANT SELECT ON emr\_db5.patient\_information TO 'patient\_user'@'localhost';



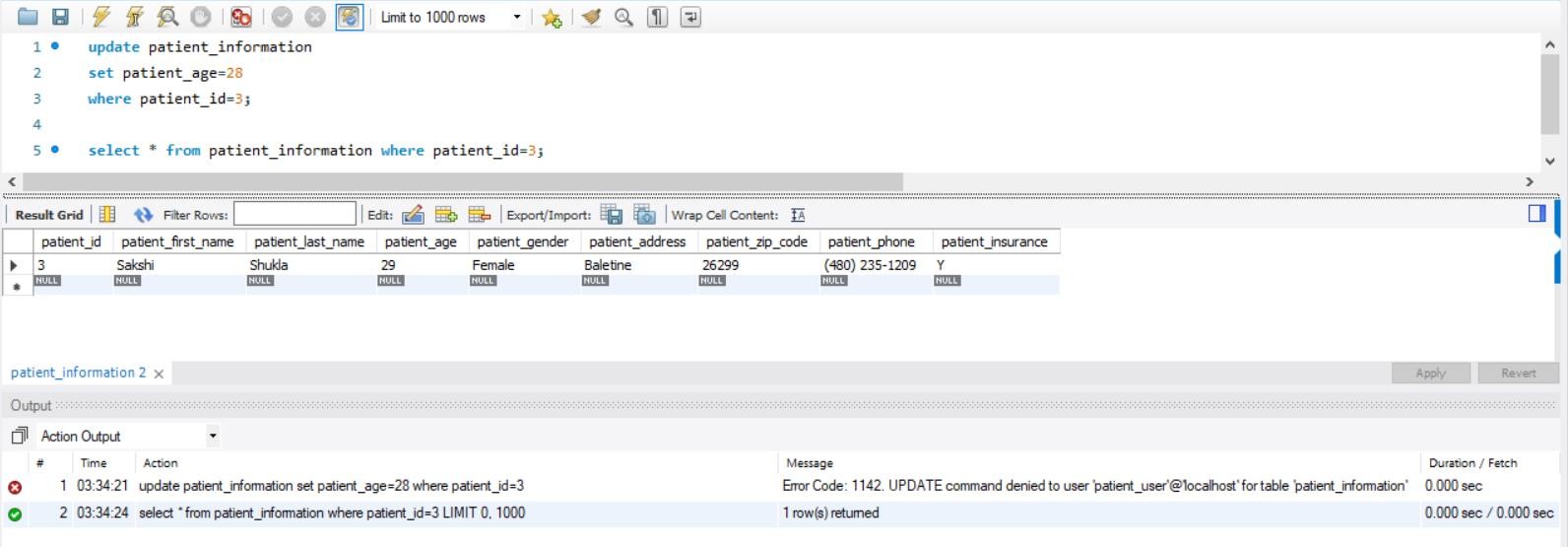
## Scenario

*Update the patient’s age to 28 of the patient with id 3 in the patient\_information table.*

update patient\_information set patient\_age=28

where patient\_id=3;

select \* from patient\_information where patient\_id=3;



Here, the patient\_user cannot update the patient\_information table, but he/she can view the table data.

# Audit Trail: Triggers

A trigger is a block of code that is executed or fired automatically when a particular type of SQL statement is executed. Triggers can be fired before or after when an insert, update and delete statement is executed on a table.

## Audit trail while data is Updated

DROP TRIGGER IF EXISTS patient\_information\_after\_update; DELIMITER //

CREATE TRIGGER patient\_information\_after\_update AFTER UPDATE ON patient\_information

FOR EACH ROW BEGIN

INSERT INTO patient\_information\_audit VALUES

(OLD.patient\_id, OLD.patient\_first\_name, OLD.patient\_last\_name, OLD.patient\_address, OLD.patient\_zip\_code, OLD.patient\_phone, 'UPDATED', session\_user(), NOW());

END// DELIMITER ;

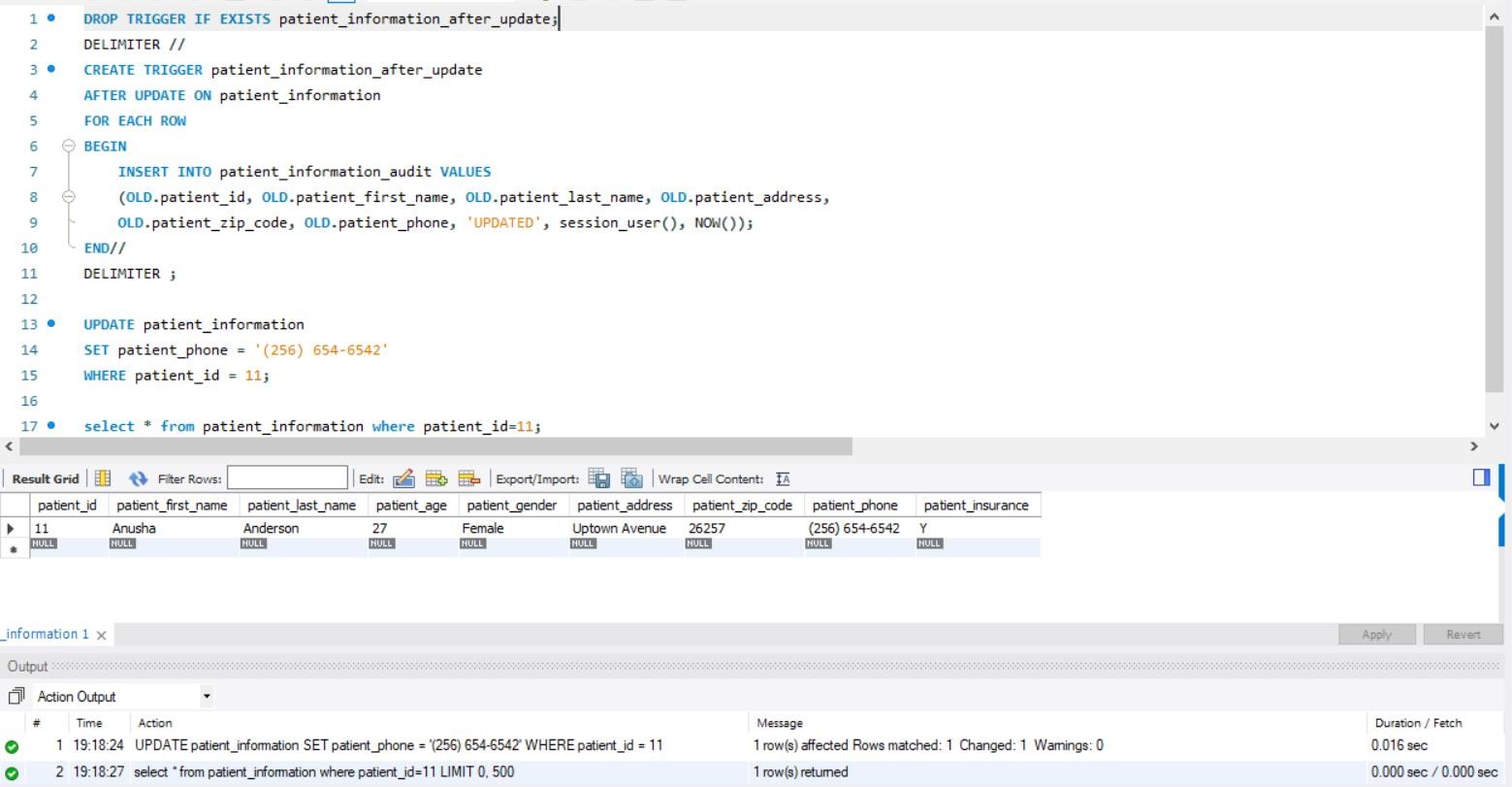
## Scenario

*Update the patient phone number to (256) 654-6542 of the patient with id 11 in the patient\_information table and maintain the old data in the audit table.*

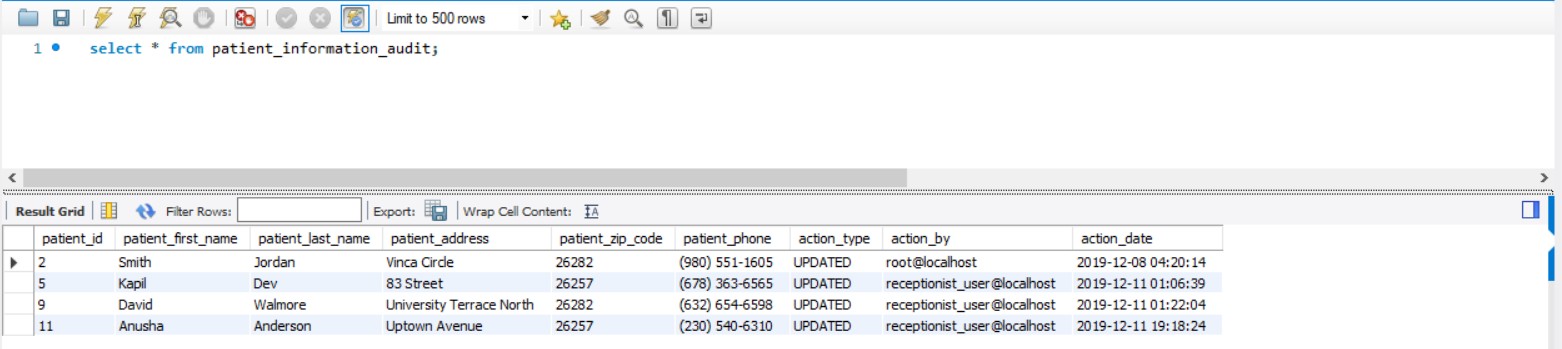
UPDATE patient\_information

SET patient\_phone = '(256) 654-6542' WHERE patient\_id = 11;

select \* from patient\_information where patient\_id=11;



select \* from patient\_information\_audit;



New data is updated in the original table and old data history is maintained in the patient\_information\_audit table with the action type, time and the user details who has done the action.

## Audit trail while data is Deleted

DROP TRIGGER IF EXISTS medicine\_after\_delete; DELIMITER //

CREATE TRIGGER medicine\_after\_delete

AFTER DELETE ON medicine FOR EACH ROW

BEGIN

INSERT INTO medicine\_audit VALUES

(OLD.medicine\_name, OLD.medicine\_price\_USD, OLD.supplier\_id, 'DELETED', session\_user(), NOW()); END; //

DELIMITER ;

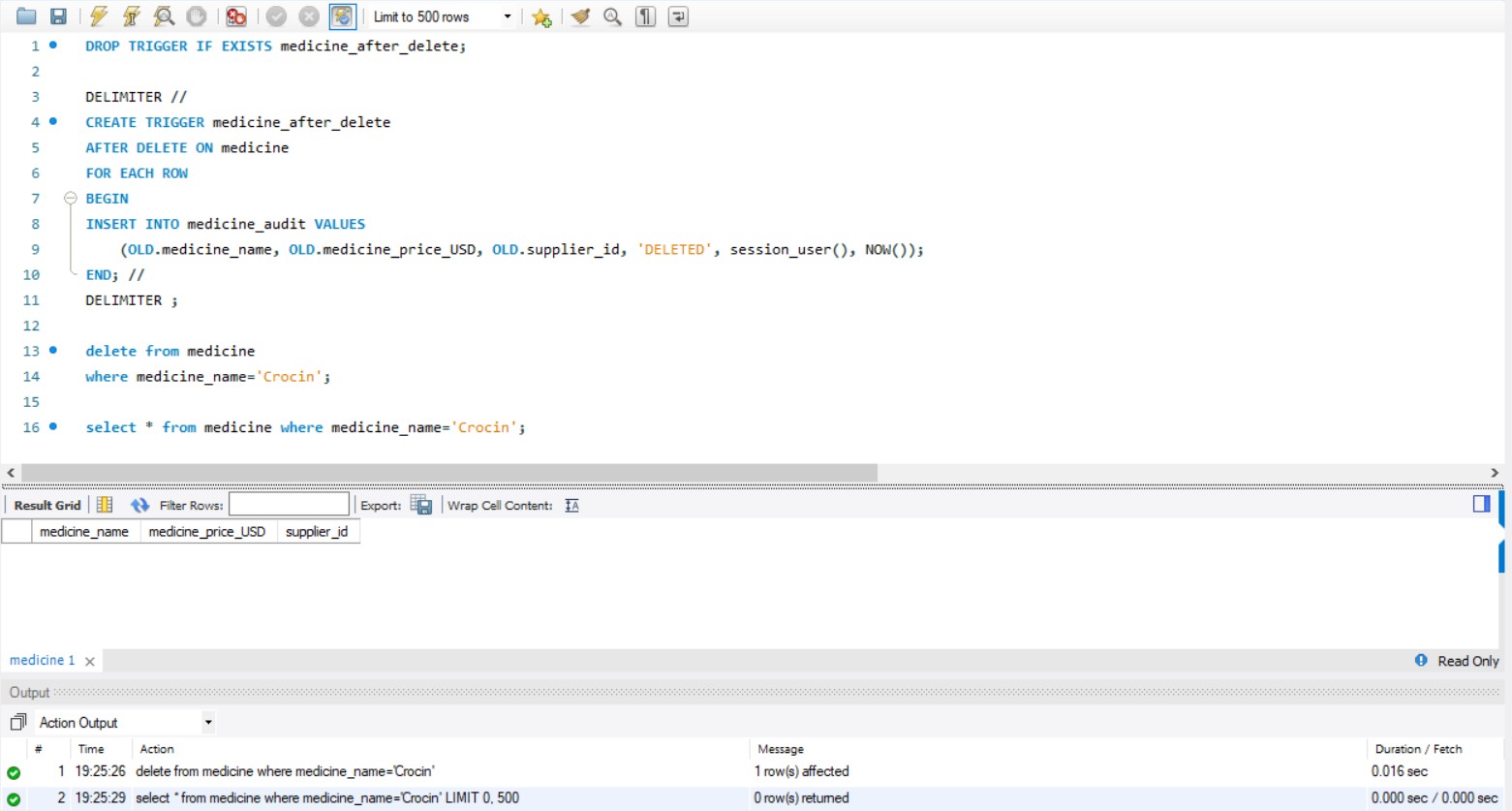
## Scenario

*Delete the medicine Crocin from the medicine table and maintain the old data in the audit table.*

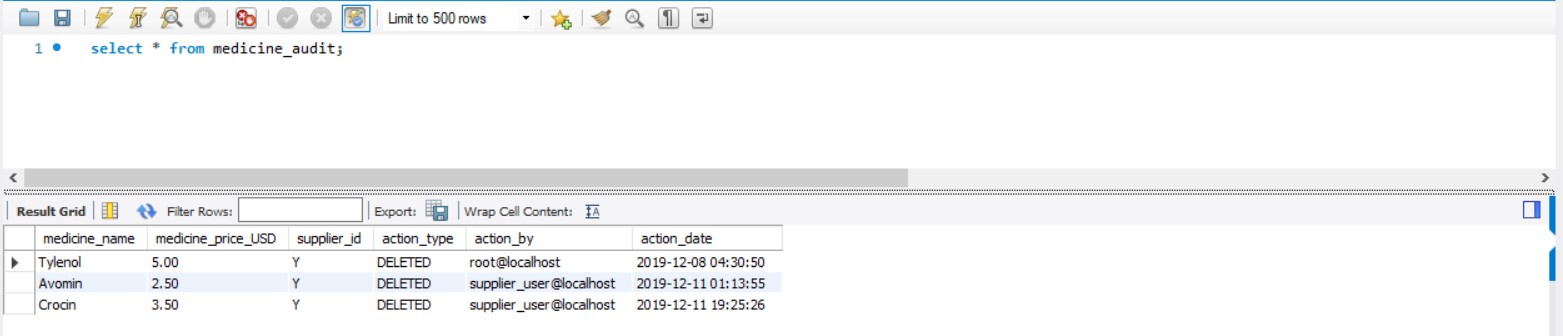
delete from medicine

where medicine\_name='Crocin';

select \* from medicine where medicine\_name='Crocin';



select \* from medicine\_audit;



Data is deleted from the original table and old data history is maintained in the medicine\_audit table with the action type, time and the user details who has done the action.

# Index

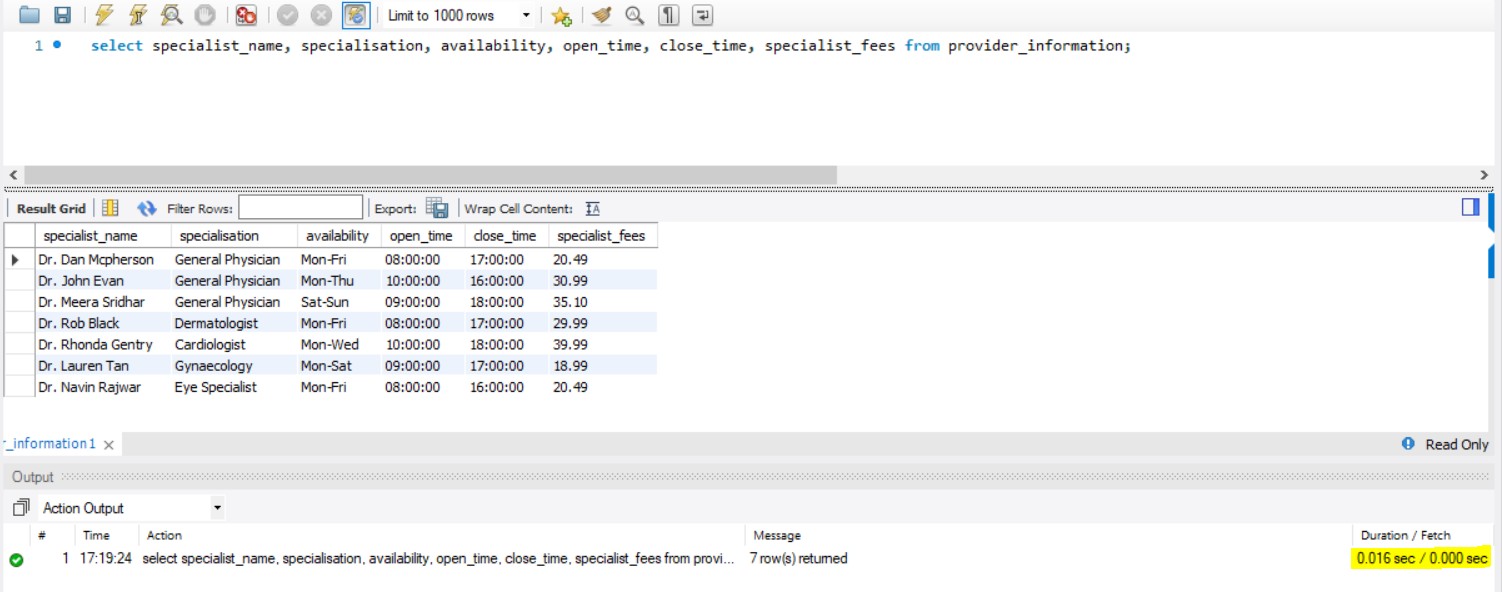
## An index speeds up joins and searches by providing a way for a database management system to go directly to a row rather than having to search through all the rows until it finds the one the user wants. By default, MySQL creates indexes for primary keys, foreign keys, and unique keys of a table. Here, we have created indexes for columns that are frequently used. However, we avoid creating Indexes on columns which are frequently updated as it will slow down Insert, Update and Delete operations.

* Scenario

*Check the execution time with and without using the index when a receptionist tries to view specialist name, specialisation, availability, open and close time, specialist fees from the provider\_information table.*

### Without index:

select specialist\_name, specialisation, availability, open\_time, close\_time, specialist\_fees from provider\_information;

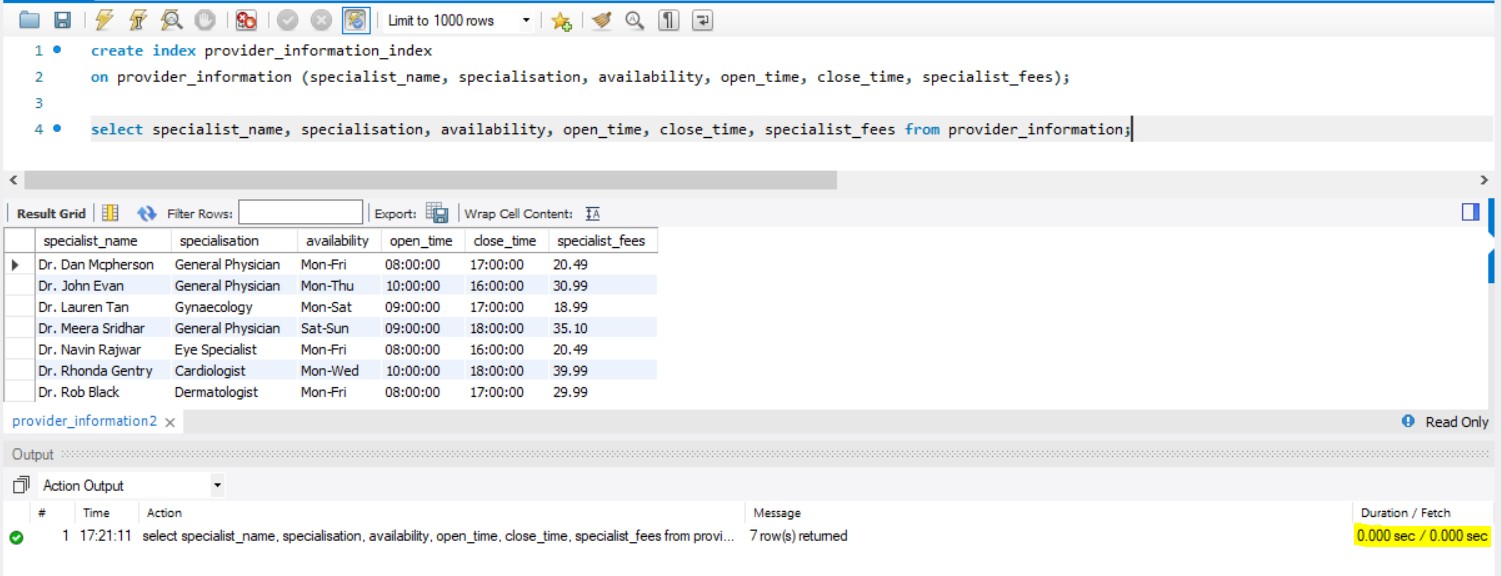


### With Index:

create index provider\_information\_index

on provider\_information (specialist\_name, specialisation, availability, open\_time, close\_time, specialist\_fees);

select specialist\_name, specialisation, availability, open\_time, close\_time, specialist\_fees from provider\_information;



The execution time is less when we use index on some of the columns of the provider\_information table.

# Views

A view is a select statement that is stored in the database as a database object. A view can be thought of a virtual table that consist of rows and columns specified in CREATE VIEW statement. To use a view, we can refer to it from another SQL statement :

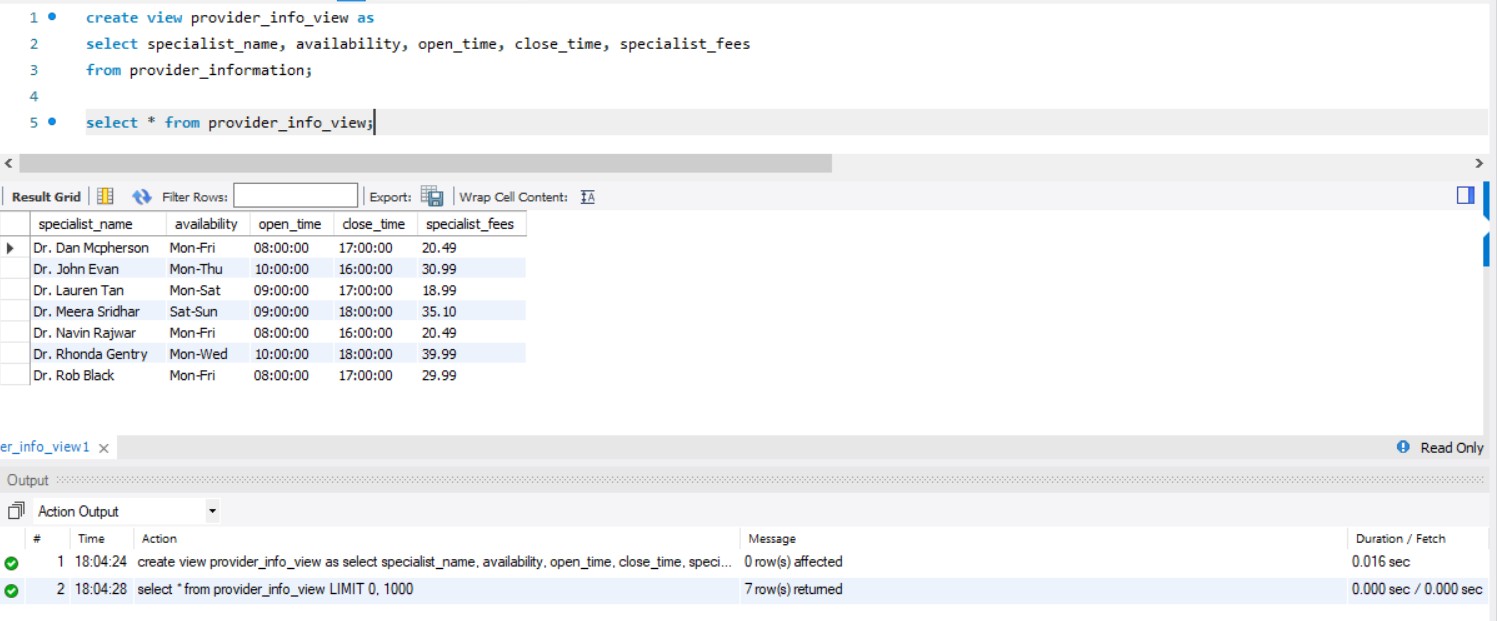
## Scenario I

*Create a view on provider\_information table and show a select statement that uses the view.*

create view provider\_info\_view as

select specialist\_name, availability, open\_time, close\_time, specialist\_fees from provider\_information;

select \* from provider\_info\_view;



## Scenario II

*Create a view on clinical\_care\_information table and show an update statement that uses the view.*

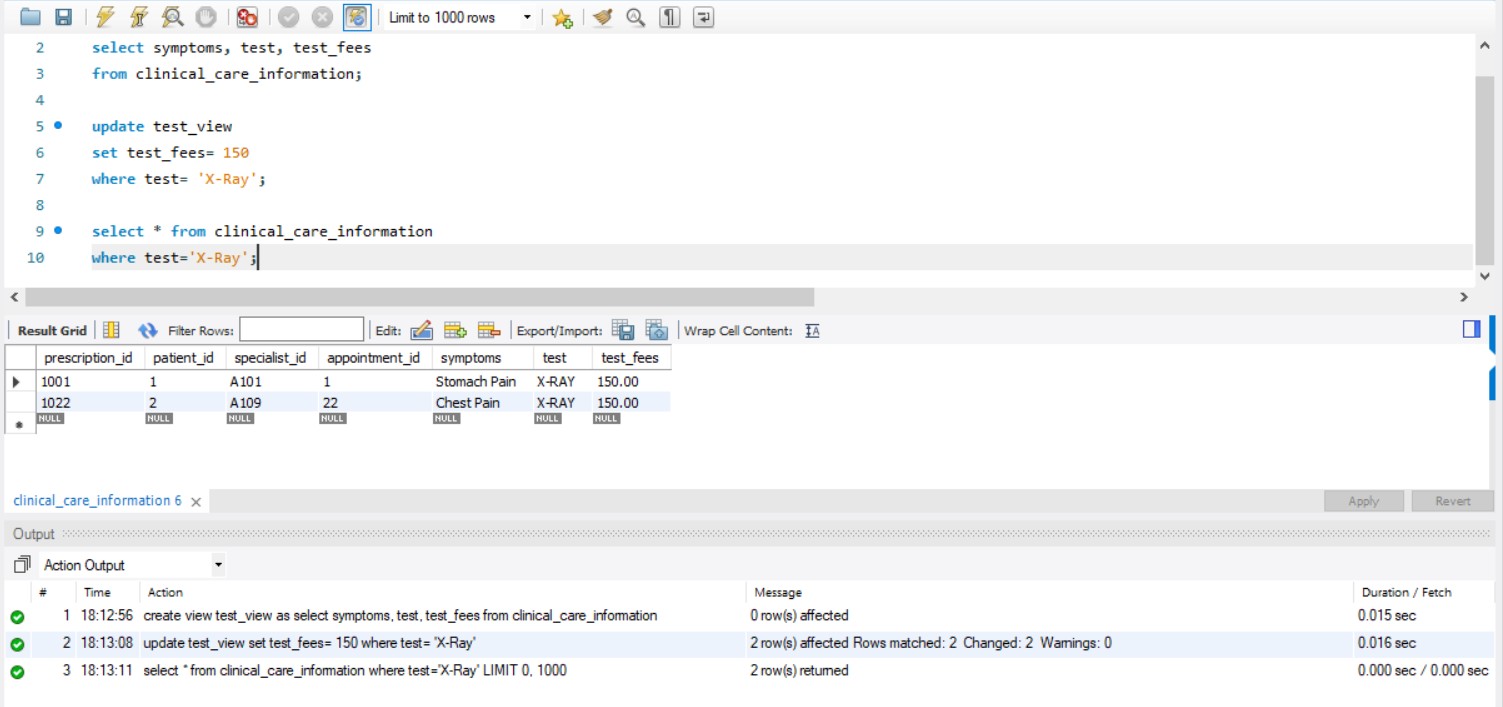
create view test\_view as

select symptoms, test, test\_fees from clinical\_care\_information;

update test\_view set test\_fees= 150

where test= 'X-Ray';

select \* from clinical\_care\_information where test='X-Ray';



# Future Enhancements

In the coming future the scope of the project can be expanded, and this project can be updated. Using the same scenarios and the same requirement arises, the project is flexible enough in terms of expansion. More tables can be added to capture other relevant data like the Insurance availed by students, test rooms availability, diagnosis exam etc. for the future enhancement. Also, index and view can used in so many places as per requirements. Audit tables can be made for different original tables. Stored procedure and triggers can be used in different ways in different areas.

# SQL Scripts

Create\_EMR\_DB5 V2.sql

InsertValues\_EMR\_DB 5 V2.sql

User Authentication.sql

Role-Based Access.sql

Show Procedure.sql Insert Procedure.sql Update

Procedure.sql

Delete Procedure.sql Update Trigger.sql Delete Trigger.sql Index.sql Provider\_Info\_View.sq

l

Test\_View.sql