

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:

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

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.

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.

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)

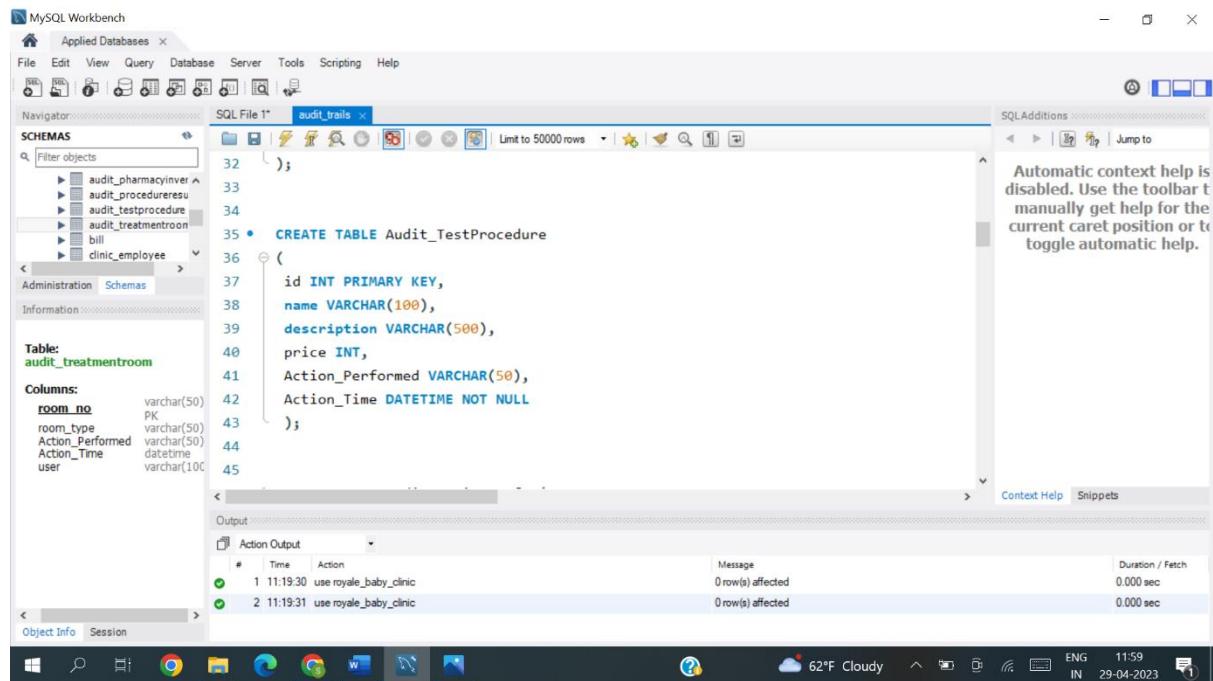
Action_Time	Datetime
user	Varchar(100)

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:



```

MySQL Workbench - Applied Databases
File Edit View Query Database Server Tools Scripting Help
Applied Databases
File Edit View Query Database Server Tools Scripting Help
Navigator
SCHEMAS
Filter objects
audit_pharmacyinver
audit_procedureresu
audit_testprocedure
audit_treatmentroo
bill
clinic_employee
Administration Schemas
Information
Table: audit_treatmentroom
Columns:
room_no varchar(50)
room_type PK
Action_Performed varchar(50)
Action_Time datetime
user varchar(100)

SQL File 1* audit_trails.x
32
33
34
35 • CREATE TABLE Audit_TestProcedure
36 (
37     id INT PRIMARY KEY,
38     name VARCHAR(100),
39     description VARCHAR(500),
40     price INT,
41     Action_Performed VARCHAR(50),
42     Action_Time DATETIME NOT NULL
43 );
44
45
Output
Action Output
# Time Action Message Duration / Fetch
1 11:19:30 use royale_baby_clinic 0 row(s) affected 0.000 sec
2 11:19:31 use royale_baby_clinic 0 row(s) affected 0.000 sec
Context Help Snippets

```

Updating an entry in test_procedure table:

The screenshot shows the MySQL Workbench interface. In the top-left, the Navigator pane displays the schema 'royale_baby_clinic' with objects like Tables, Views, Stored Procedures, and Functions. The main area contains a SQL editor titled 'SQL File 1' with the following code:

```

1 • use royale_baby_clinic;
2
3 • select * from test_procedure;
4 • update test_procedure set price = 750 where id = 5;

```

Below the SQL editor is the Output pane, which shows the execution results:

#	Time	Action	Message	Duration / Fetch
3	12:00:47	update test_procedure set price = 95 where id = 10	Error Code: 1062. Duplicate entry '10' for key 'audit_testprocedure PRIMARY'	0.032 sec
4	12:01:02	select * from test_procedure LIMIT 0, 50000	9 row(s) returned	0.000 sec / 0.000 sec
5	12:01:17	update test_procedure set price = 750 where id = 5	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0	0.094 sec

The status bar at the bottom right shows the date and time as 29-04-2023 12:01.

Table showing who have done changes to the test_procedure table.

The screenshot shows the MySQL Workbench interface. In the top-left, the Navigator pane displays the schema 'royale_baby_clinic' with objects like audit_pharmacyinver, audit_proceduresres, audit_testprocedure, audit_treatmentroom, bill, and clinic_employee. The main area contains a SQL editor titled 'SQL File 1' with the following code:

```

1 • use royale_baby_clinic;
2
3 • select * from audit_testprocedure;

```

Below the SQL editor is the Result Grid pane, which displays the data from the 'audit_trails' table:

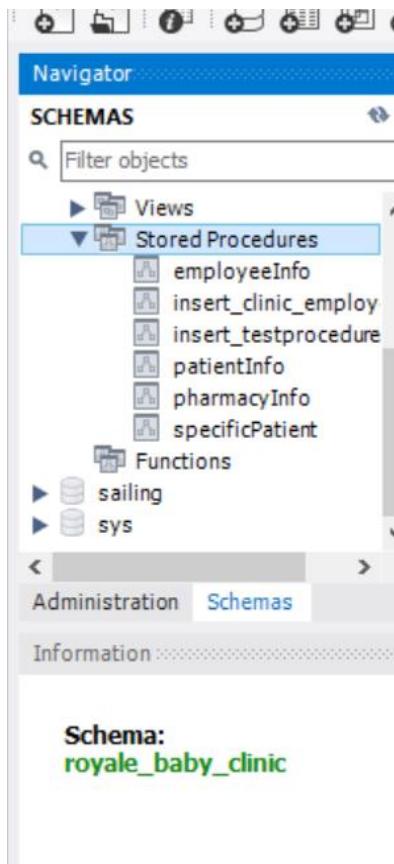
id	name	description	price	Action_Performed	Action_Time	user
1	Blood Tests	general blood test	150	updated	2023-04-26 13:43:58	ITAdmin2@localhost
5	Cavity filling	Repairing teeth	700	updated	2023-04-29 12:01:17	ITAdmin2@localhost
8	Tooth Repair	repairing the tooth	650	updated	2023-04-26 13:38:32	root@localhost
9	Tooth Removal	Removing the tooth	700	deleted	2023-04-26 13:36:14	root@localhost
10	Teeth cleaning	cleaning	90	inserted	2023-04-26 14:50:47	ITAdmin1@localhost
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Below the Result Grid is the Output pane, which shows the execution results:

#	Time	Action	Message	Duration / Fetch
1	11:19:30	use royale_baby_clinic	0 row(s) affected	0.000 sec
2	11:19:31	use royale_baby_clinic	0 row(s) affected	0.000 sec
3	12:02:27	select * from audit_testprocedure LIMIT 0, 50000	5 row(s) returned	0.000 sec / 0.000 sec

The status bar at the bottom right shows the date and time as 29-04-2023 12:02.

Stored Procedures:



Schema:
royale_baby_clinic

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 ;
```

The screenshot shows the MySQL Workbench interface. In the top navigation bar, 'Applied Databases' and 'ITAdmin2' are selected. The left sidebar shows the 'SCHEMAS' tree, with 'royale_baby_clinic' selected. The main query editor window contains the following SQL code:

```
1 • DROP PROCEDURE IF EXISTS employeeInfo;
2 DELIMITER $$;
3 • USE `royale_baby_clinic`$$;
4 • CREATE PROCEDURE employeeInfo()
5 BEGIN
6 select * from clinic_employee;
7 end
8 $$;
9 DELIMITER ;
```

The code is numbered from 1 to 10. The 'Output' pane at the bottom shows the execution results:

#	Time	Action	Message	Duration / Fetch
3	12:02:27	select * from audit_testprocedure LIMIT 0, 50000	5 row(s) returned	0.000 sec / 0.000 sec
4	12:25:18	drop procedure employeeInfo	0 row(s) affected	0.078 sec
5	12:25:22	CREATE PROCEDURE employeeInfo() BEGIN select * from clinic_employee; end	0 row(s) affected	0.031 sec

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 ;

The screenshot shows the MySQL Workbench interface. The left sidebar displays the schema 'royale_baby_clinic' with its tables, views, and stored procedures. The central pane contains the SQL code for creating a stored procedure:

```
37 DROP PROCEDURE IF EXISTS specificPatient;
38 DELIMITER $$;
39 USE `royale_baby_clinic`$$;
40 CREATE PROCEDURE specificPatient(IN pid int)
41 BEGIN
42 select * from patient_info where patient_id = pid;
43 end
44 $$;
45 DELIMITER ;
46
```

The right pane shows the 'Output' tab with the execution results:

#	Time	Action	Message	Duration / Fetch
6	12:26:02	CREATE PROCEDURE pharmacyInfo() BEGIN select * from pharmacy_inventory; e...	Error Code: 1304. PROCEDURE pharmacyInfo already exists	0.000 sec
7	12:26:30	drop procedure specificPatient	0 row(s) affected	0.063 sec
8	12:26:32	CREATE PROCEDURE specificPatient(IN pid int) BEGIN select * from patient_info ...	0 row(s) affected	0.032 sec

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 ;
```

The screenshot shows the MySQL Workbench interface. In the central SQL editor window, the code for creating a stored procedure is being typed:

```
24 • DROP PROCEDURE IF EXISTS pharmacyInfo;
25 DELIMITER $$
26 • USE `royale_baby_clinic`$$
27 • CREATE PROCEDURE pharmacyInfo()
28 BEGIN
29 select * from pharmacy_inventory;
30 end
31 $$
32 DELIMITER ;
33
```

The code is being run against the schema 'royale_baby_clinic'. The output pane shows the results of the execution:

#	Time	Action	Message	Duration / Fetch
11	12:29:16	CREATE PROCEDURE pharmacyInfo() BEGIN select * from pharmacy_inventory; e...	Error Code: 1304. PROCEDURE pharmacyInfo already exists	0.000 sec
12	12:29:32	DROP PROCEDURE IF EXISTS pharmacyInfo	0 row(s) affected	0.125 sec
13	12:29:35	CREATE PROCEDURE pharmacyInfo() BEGIN select * from pharmacy_inventory; e...	0 row(s) affected	0.094 sec

The status bar at the bottom indicates the script was saved to 'C:\Users\Shanmukhi\Desktop\Applied Databases\Project2\stored_procedure.sql'.

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.

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

My db server 8.0.32
connected to database: ('royale_baby_clinic',)
Select :
1. employeeInfo
2. patientinfo
3. pharmacyInfo
1
1201
Stephen
Joseph
Pediatrician
15000.00
-----
1202
Jeffrey
Mary
Neonatologist
12000.00
-----
1203
```

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

Select :
1. employeeInfo
2. patientinfo
3. pharmacyInfo
2
Enter patient id:
1301
1301
Emma

Watson
Female
12
8900786756
12F Arshford
121212001
1201
[]
```

Code used for creating text-based interface:

```
import mysql.connector

connection =
mysql.connector.connect(host='127.0.0.1',database='royale_baby_clinic',user='root',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():

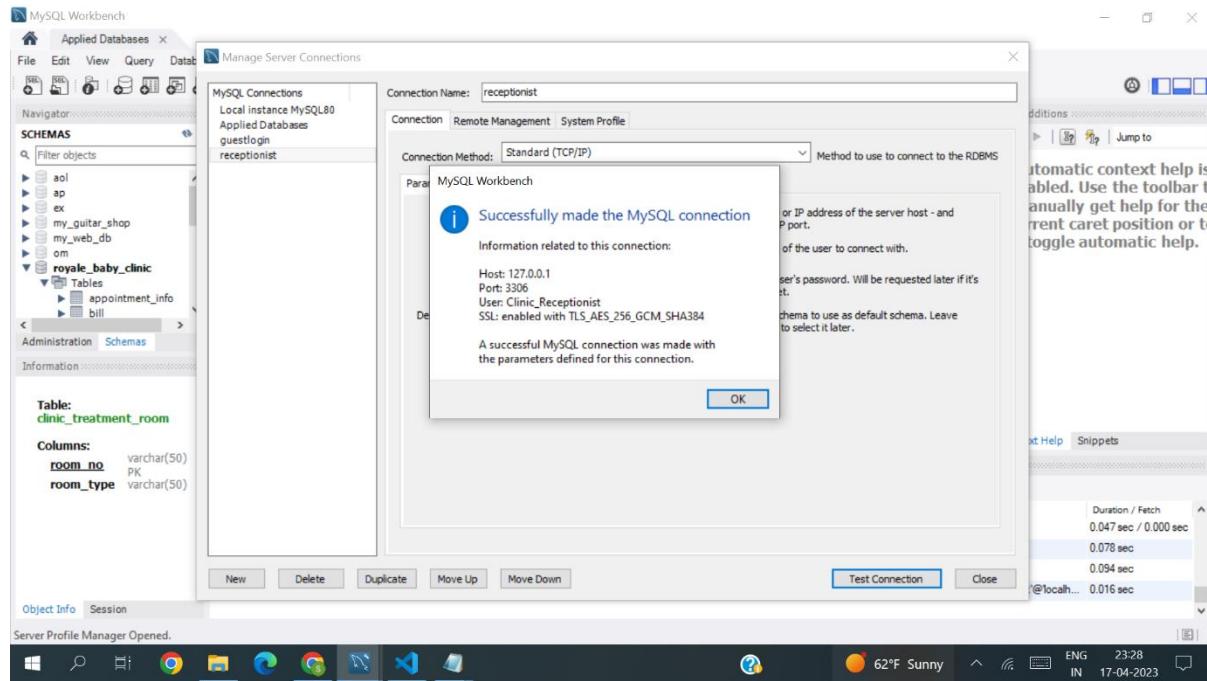
    list = result.fetchall()
    for row in list:
        col = row
        for c in col:
            print(c)
        print('-----')
    print(result.fetchall())
```

USERS & ROLES:

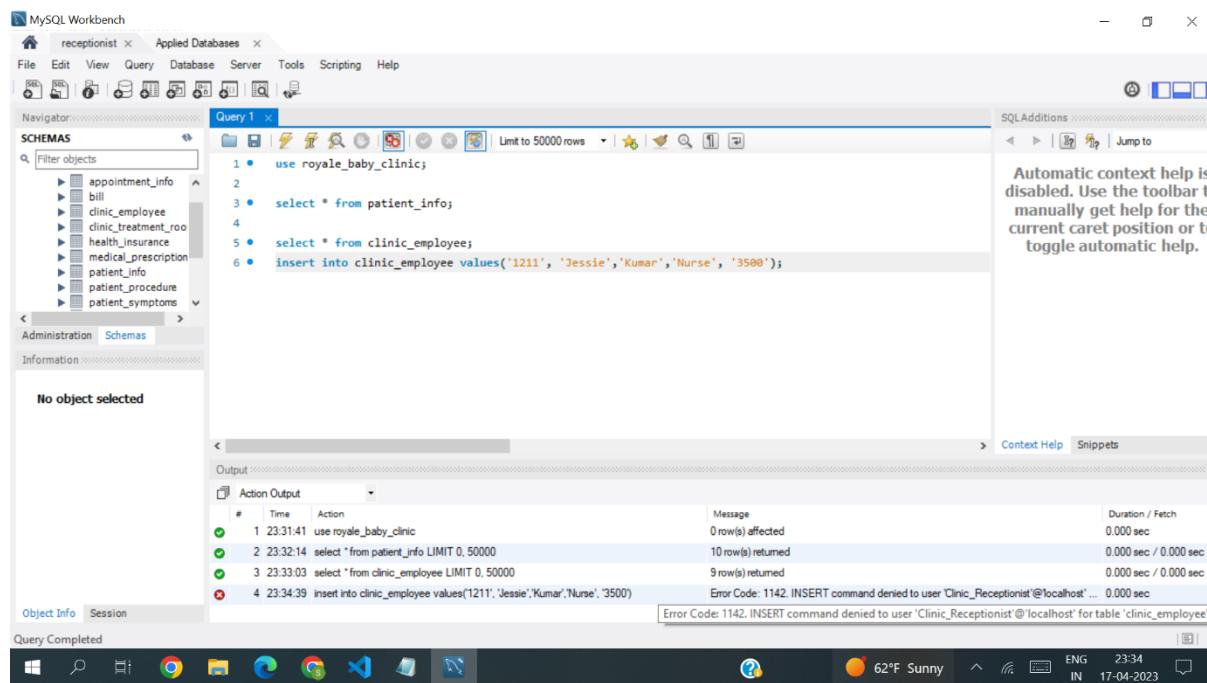
-- creating receptionist

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;

The screenshot shows two instances of MySQL Workbench. The top instance is titled 'Manage Server Connections' and displays a connection named 'ITAdmin' using the 'Standard (TCP/IP)' method. A modal window titled 'MySQL Workbench' shows a success message: 'Successfully made the MySQL connection'. The bottom instance is titled 'ITAdmin' and shows a query editor with the following SQL code:

```

1 • use royale_baby_clinic;
2
3 • select * from clinic_employee;
4 • insert into clinic_employee values('1212','Mary','Kona', 'Nurse', '4500');

```

The 'Output' tab shows the results of the query execution:

#	Time	Action	Message	Duration / Fetch
1	15:24:50	use royale_baby_clinic	0 row(s) affected	0.000 sec
2	15:25:06	select * from clinic_employee LIMIT 0, 50000	9 row(s) returned	0.000 sec / 0.000 sec
3	15:25:53	insert into clinic_employee values('1212','Mary','Kona', 'Nurse', '4500')	1 row(s) affected	0.047 sec

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

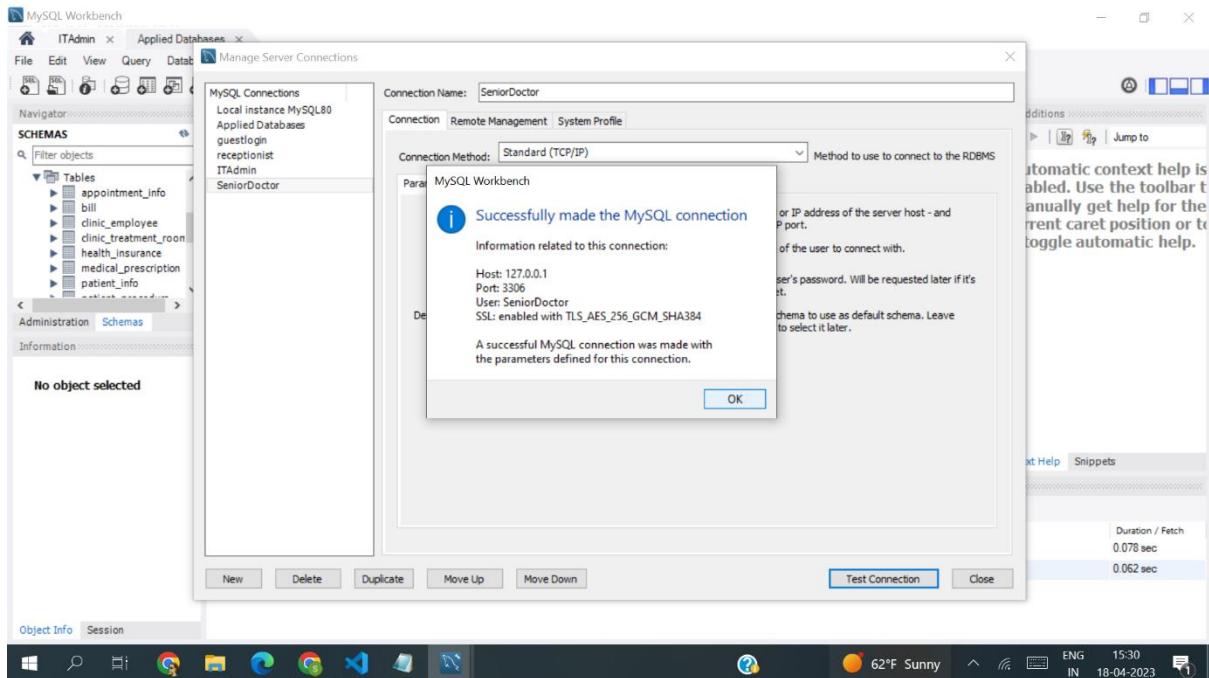
Created user seniordoctor:

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

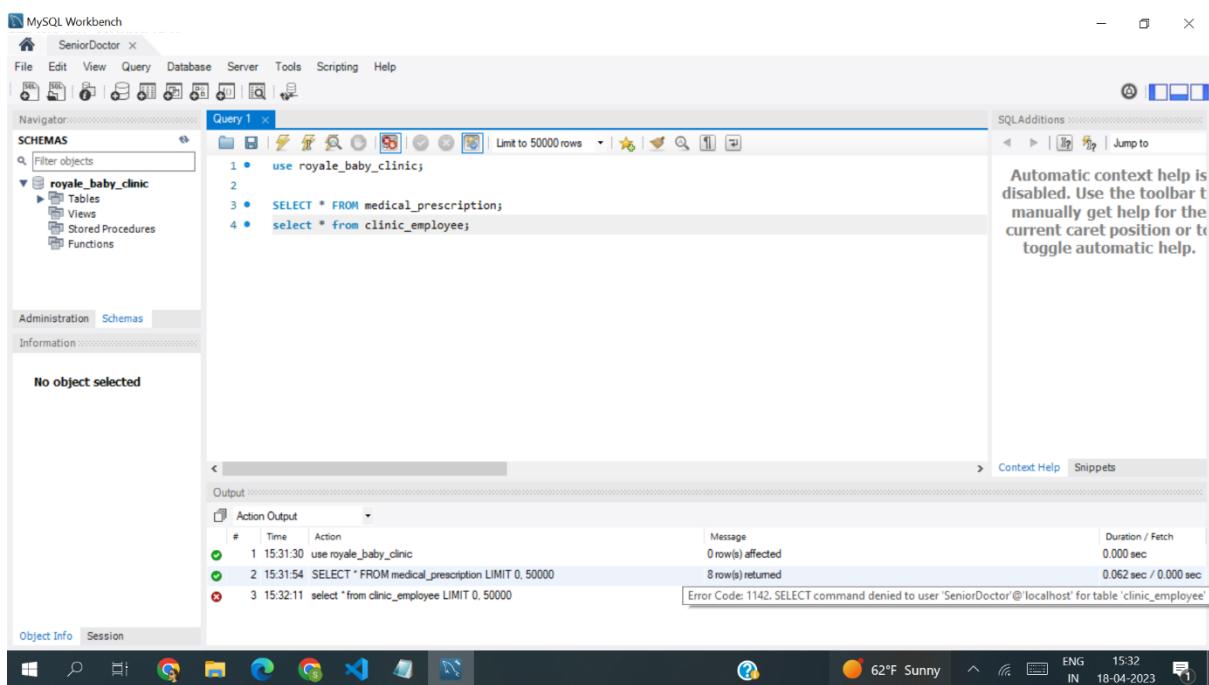
Granted the doctor all privileges only on medical_prescription and patient_info table.

```
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;
```



Doctor trying to access clinic_employee table:

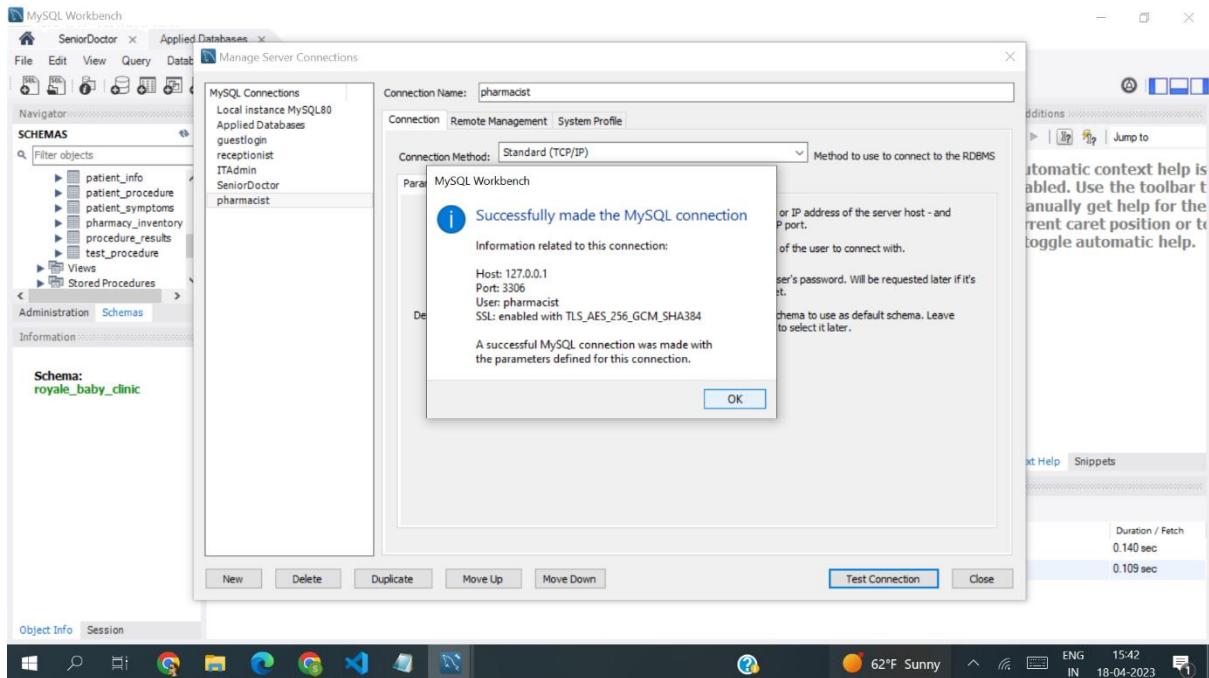


Created user pharmacist:

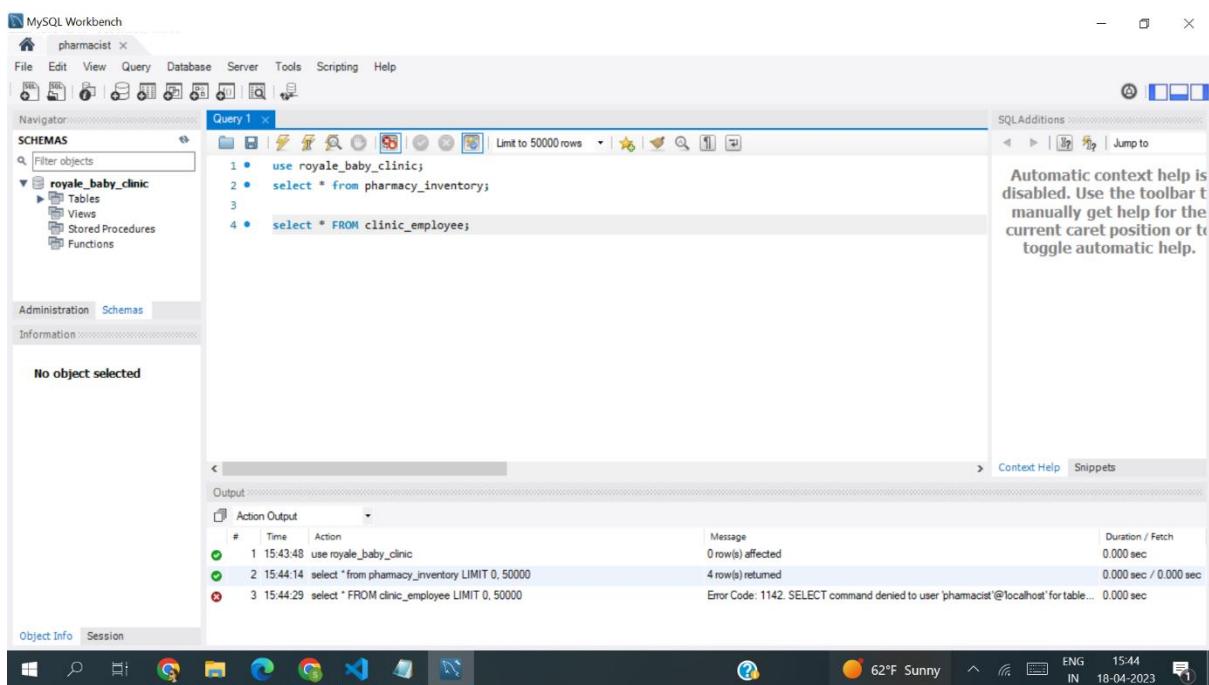
```
CREATE USER pharmacist@localhost IDENTIFIED BY 'pharma@123';
```

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;
```



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);
```

MySQL Workbench Screenshot:

```

project2* SQL File 2*
54 • #INDEXES
55 • CREATE INDEX employee_name_info
56 ON clinic_employee(first_name DESC);
57
58 • show index from clinic_employee;
59

```

Result Grid (Table: health_insurance):

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type
clinic_employee	0	PRIMARY	1	Employee_Id	A	8	HUll	HUll	BTF	BTf
clinic_employee	1	employee_name_info	1	First_Name	D	9	HUll	HUll	BTF	BTf

Action Output (Log):

#	Time	Action	Message	Duration / Fetch
44	16:43:36	CREATE VIEW appointmentInfo_v AS SELECT patient_id, nurse_id, room_number ...	0 row(s) affected	0.094 sec
45	16:43:55	SELECT * FROM appointmentInfo_v LIMIT 0.50000	9 row(s) returned	0.000 sec / 0.000 sec
46	16:54:38	CREATE INDEX employee_name_info ON clinic_employee(first_name DESC)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.625 sec
47	16:55:22	show index from clinic_employee	2 row(s) returned	0.015 sec / 0.000 sec

Index to recognize treatment room by room type:

CREATE INDEX treatmentRoom ON clinic_treatment_room(room_type);

MySQL Workbench Screenshot:

```

project2* SQL File 2*
60
61 • CREATE INDEX treatmentRoom
62 ON clinic_treatment_room(room_type);
63
64 • show index from clinic_treatment_room;
65

```

Result Grid (Table: clinic_treatment_room):

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type
clinic_treatment_room	0	PRIMARY	1	room_no	A	8	HUll	HUll	BTF	BTf
clinic_treatment_room	1	treatmentRoom	1	room_type	A	8	HUll	HUll	BTF	BTf

Action Output (Log):

#	Time	Action	Message	Duration / Fetch
46	16:54:38	CREATE INDEX employee_name_info ON clinic_employee(first_name DESC)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.625 sec
47	16:55:22	show index from clinic_employee	2 row(s) returned	0.015 sec / 0.000 sec
48	16:57:38	CREATE INDEX treatmentRoom ON clinic_treatment_room(room_type)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.297 sec
49	16:58:00	show index from clinic_treatment_room	2 row(s) returned	0.047 sec / 0.000 sec

Index to recognize treatment room by room number:

CREATE INDEX treatmentRoom ON clinic_treatment_room(room_no);

The screenshot shows the MySQL Workbench interface with the following details:

- Schemas:** The current schema is "project2".
- SQL File 2:**

```

CREATE INDEX treatmentRoom1
ON clinic_treatment_room(room_no);

show index from clinic_treatment_room;
    
```
- Result Grid:** Shows the results of the SHOW INDEX query for the "clinic_treatment_room" table.
- Action Output:** Displays the log of actions taken, including the creation of the index and the execution of the SHOW INDEX command.
- System Tray:** Shows the date and time as 17-04-2023, 23:11, and the weather as 62°F Sunny.

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 );
```

The screenshot shows the MySQL Workbench interface with the following details:

- Schemas:** The current schema is "project2".
- SQL File 2:**

```

CREATE INDEX patient_insurance
ON patient_info(Insurance_id DESC);

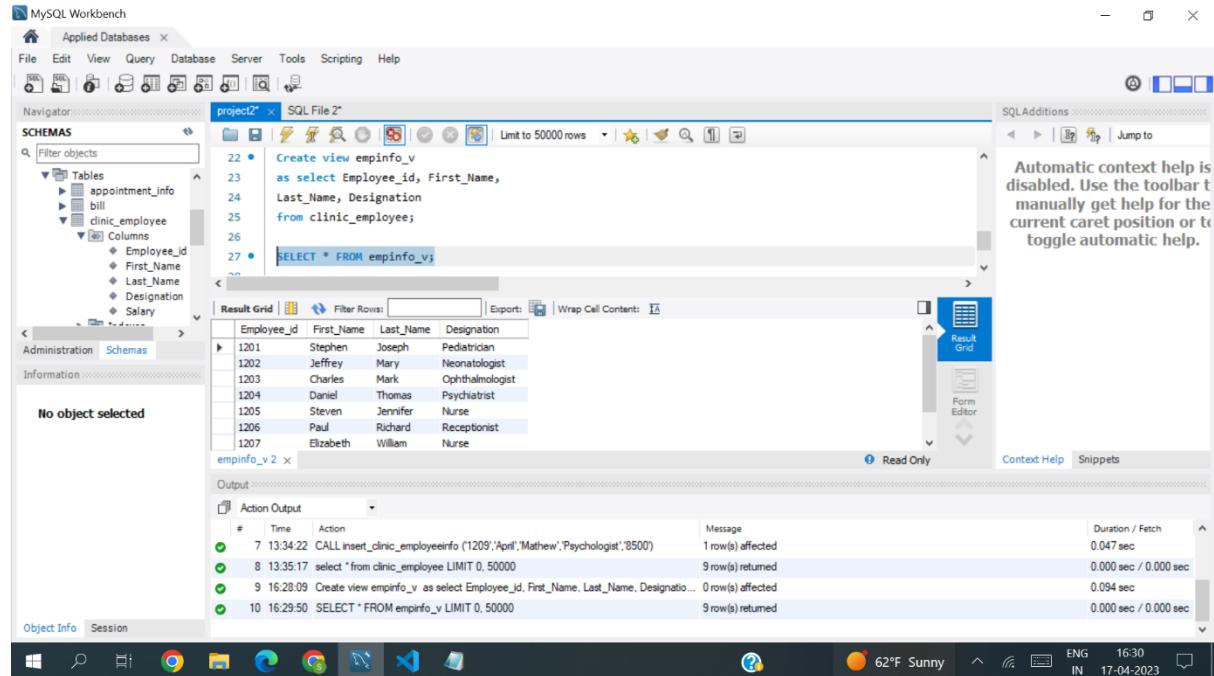
CREATE INDEX patient_id
ON patient_info(patient_id);

SHOW INDEX FROM patient_info;
    
```
- Result Grid:** Shows the results of the SHOW INDEX query for the "patient_info" table.
- Action Output:** Displays the log of actions taken, including the creation of two indexes and the execution of the SHOW INDEX command.
- System Tray:** Shows the date and time as 17-04-2023, 23:13, and the weather as 62°F Sunny.

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;



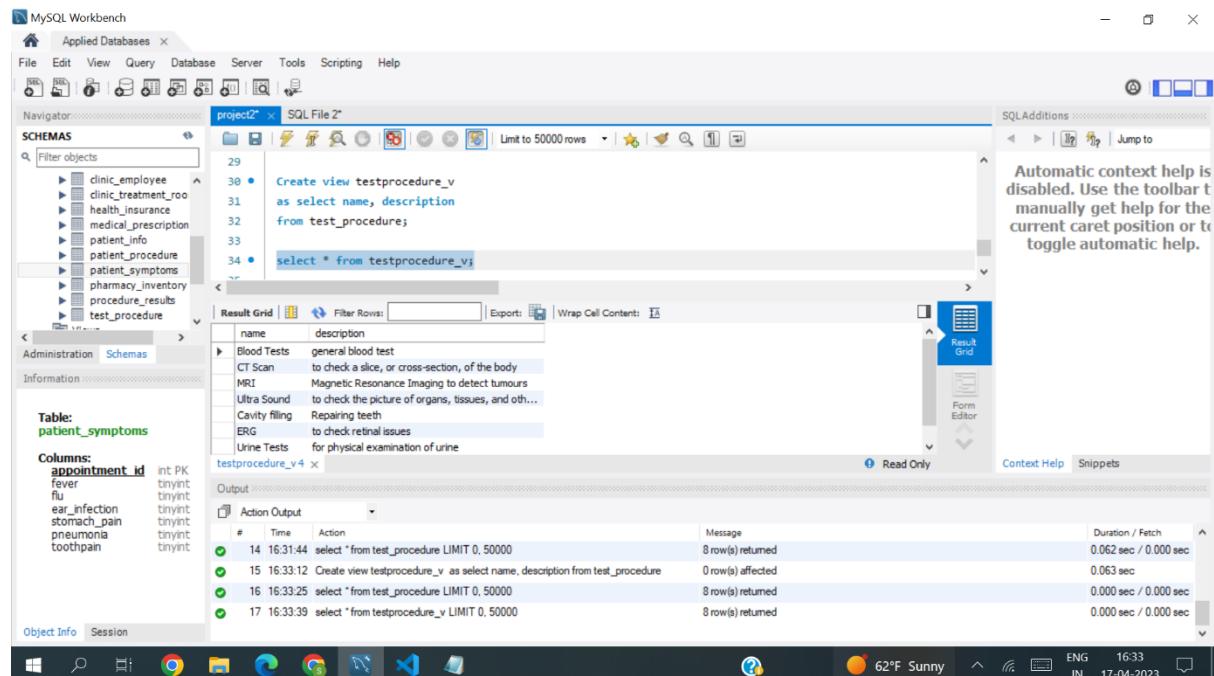
```
CREATE VIEW empinfo_v AS SELECT * FROM clinic_employee;
```

Employee_id	First_Name	Last_Name	Designation
1201	Stephen	Joseph	Pediatrician
1202	Jeffrey	Mary	Neonatologist
1203	Charles	Mark	Ophthalmologist
1204	Daniel	Thomas	Psychiatrist
1205	Steven	Jennifer	Nurse
1206	Paul	Richard	Receptionist
1207	Elizabeth	William	Nurse

```
CALL insert_clinic_employeeinfo ('1209','April','Mathew','Psychologist',8500)
select * from clinic_employee LIMIT 0, 50000
Create view empinfo_v as select Employee_id, First_Name, Last_Name, Designation from clinic_employee;
SELECT * FROM empinfo_v LIMIT 0, 50000
```

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

Create view testprocedure_v as select name, description from test_procedure;



```
CREATE VIEW testprocedure_v AS SELECT * FROM test_procedure;
```

name	description
Blood Tests	general blood test
CT Scan	to check a slice, or cross-section, of the body
MRI	Magnetic Resonance Imaging to detect tumours
Ultra Sound	to check the picture of organs, tissues, and other structures
Cavity filling	Repairing teeth
ERG	to check retinal issues
Urine Tests	for physical examination of urine

```
select * from testprocedure_v
Create view testprocedure_v as select name, description from test_procedure
SELECT * FROM testprocedure_v
```

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;
```

patient_id	First_Name	Gender
1303	Benjamin	James
1304	Henry	Watson
1305	Jack	Levi
1306	Isabella	Mia
1307	Mary	Queen
1308	Evelyn	Patrick
1309	Charlotte	Amelia

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

Create view insuredcadedetails_v as

```
select insurance_id, insurance_provider, policy_no, end_date
from health_insurance;
```

insurance_id	insurance_provider	policy_no	end_date
121212001	Humana	Humana Silver	2023-08-22 08:00:00
121212002	Humana	Humana Gold	2023-05-01 09:00:00
121212003	Cigna	Cigna Silver	2024-01-01 07:00:00
121212004	Cigna	Cigna Gold	2023-05-01 09:00:00
121212005	Coventry	Coventry Silver	2024-10-01 20:00:00
121212006	Coventry	Coventry Gold	2024-09-01 19:00:00
121212007	BlueCross	BlueCross Silver	2024-05-01 10:00:00

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;
```

```
CREATE VIEW appointmentInfo_v AS
SELECT patient_id, nurse_id, room_number from appointment_info;
SELECT * FROM appointmentInfo_v;
```

patient_id	nurse_id	room_number
1301	1205	I101
1302	1205	G101
1303	1207	G102
1304	1205	G103
1303	1205	G101
1303	1207	IC101
1302	1207	X101

Action Output

#	Time	Action	Message	Duration / Fetch
42	16:42:33	CREATE VIEW treatmentroom_v AS SELECT patient_id, nurse_id, room_number fr...	Error Code: 1054 Unknown column 'patient_id' in field list'	0.000 sec
43	16:43:06	select * from appointment_info LIMIT 0, 50000	9 row(s) returned	0.015 sec / 0.000 sec
44	16:43:36	CREATE VIEW appointmentInfo_v AS SELECT patient_id, nurse_id, room_number ...	0 row(s) affected	0.094 sec
45	16:43:55	SELECT * FROM appointmentInfo_v LIMIT 0, 50000	9 row(s) returned	0.000 sec / 0.000 sec

TRIGGERS:

Created triggers for all tables.

Audit_Appointment_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

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

The screenshot shows the MySQL Workbench interface with the following details:

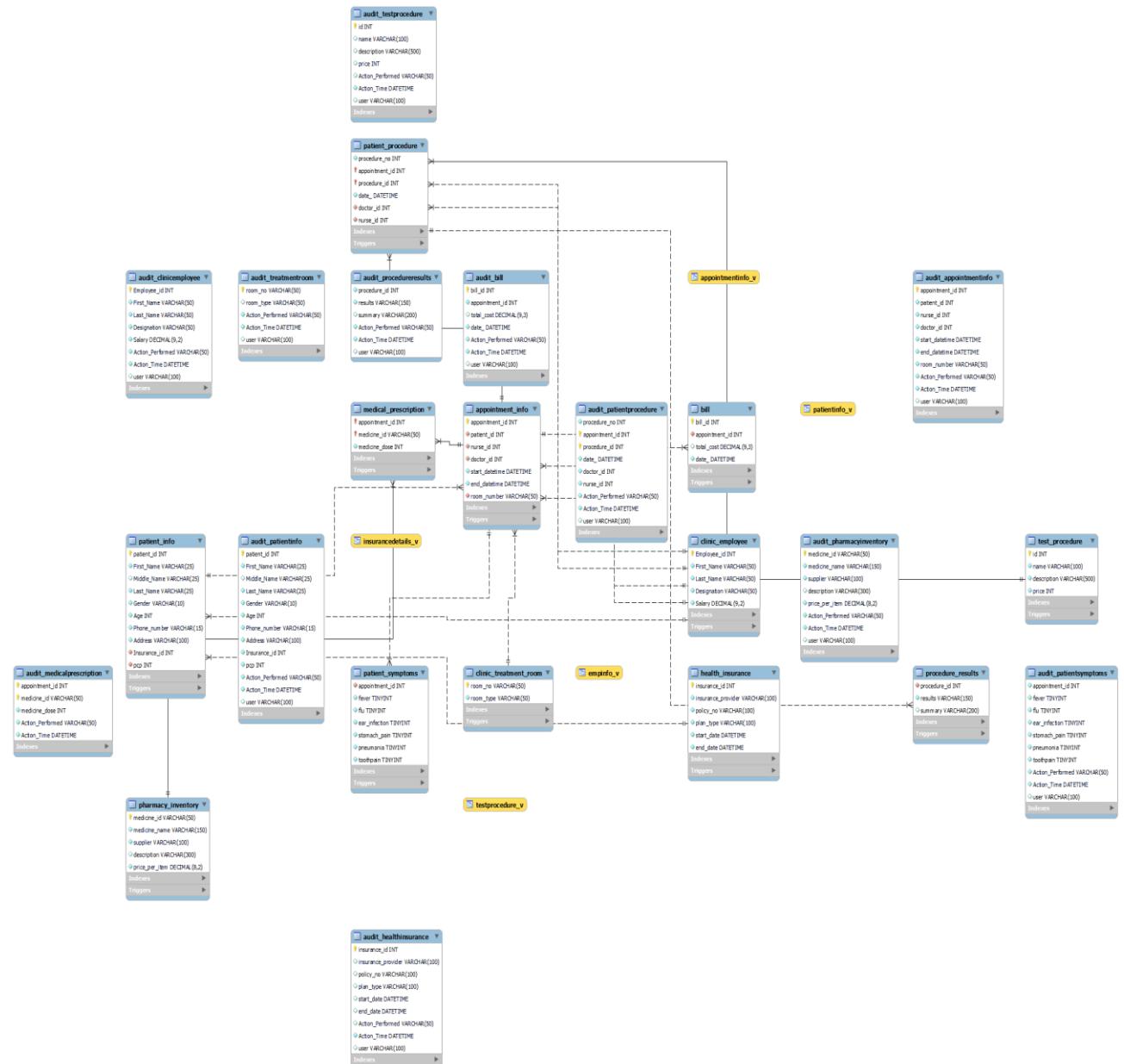
- File Bar:** File, Edit, View, Query, Database, Server, Tools, Scripting, Help.
- Toolbar:** Standard MySQL icons for connection, database selection, and query execution.
- Navigator:** Shows the **SCHEMAS** section with the **royale_baby_clinic** schema selected. Inside, it lists **Tables** (appointmentinfo_v, empinfo_v, insurancedetails_v, patientinfo_v) and **Views** (patient_info).
- SQL Editor:** Contains the following SQL statements:

```
45 patient_info(patient_id );
46
47 • SHOW INDEX FROM patient_info;
48
49 • SHOW TRIGGERS;
```
- Result Grid:** Displays a table of triggers with columns: Trigger, Event, Table, Statement, Timing, and Created. The data includes:

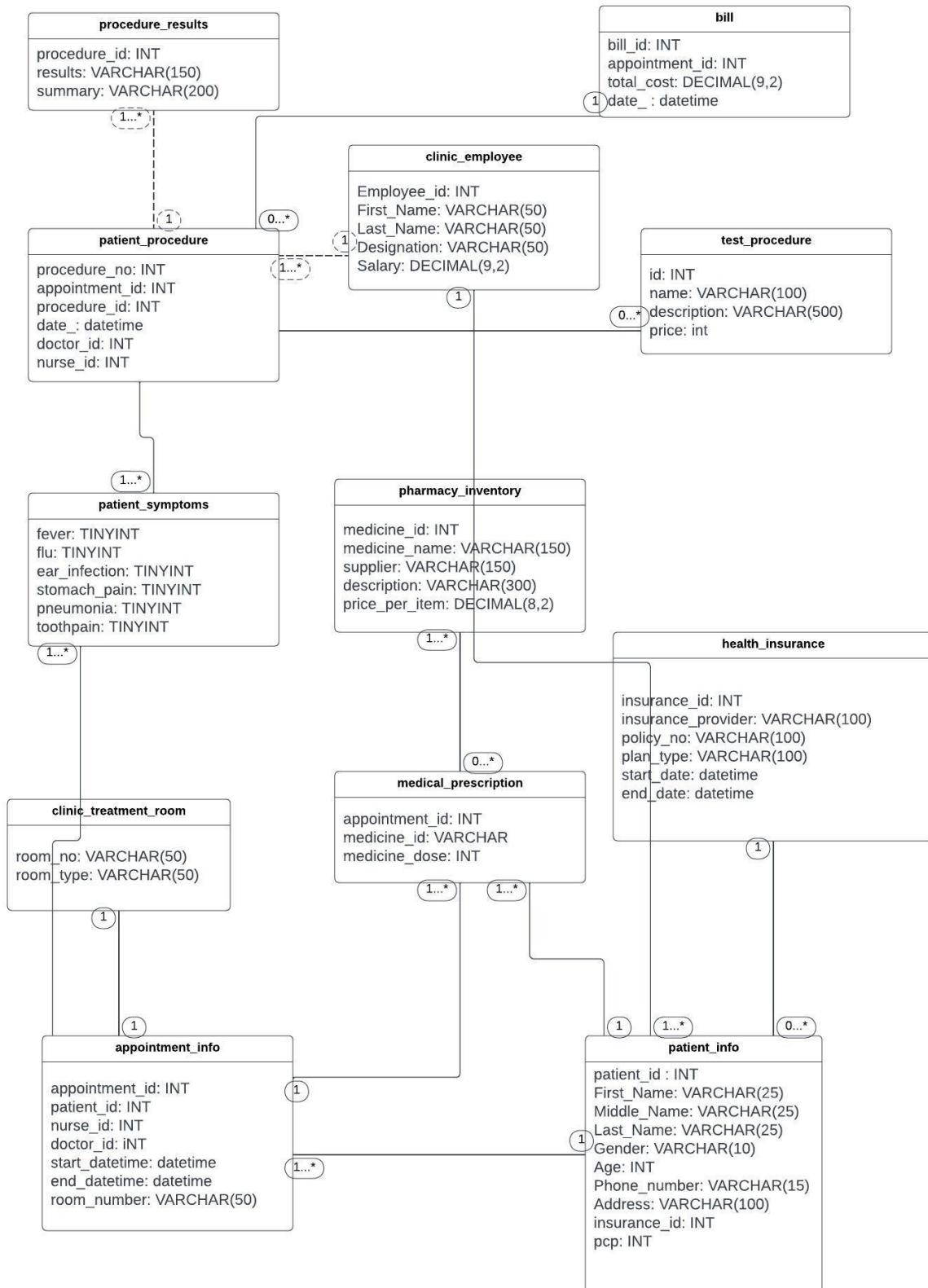
Trigger	Event	Table	Statement	Timing	Created
Audit_bill_AFTER_UPDATE	UPDATE	bill	BEGIN Insert into audit_bill values (OLD..., AFTER	2023-04-26 13:48:08	
Audit_bill_AFTER_DELETE	DELETE	bill	BEGIN Insert into audit_bill values (OLD..., AFTER	2023-04-26 13:48:35	
Audit_ClinicEmployee_AFTER_INSERT	INSERT	clinic_employee	BEGIN Insert into audit_clinicemployee va... AFTER	2023-04-19 10:41:19	
Audit_employee_AFTER_INSERT	INSERT	clinic_employee	BEGIN Insert into audit_clinicemployee va... AFTER	2023-04-26 14:31:11	
Audit_employee_AFTER_UPDATE	UPDATE	clinic_employee	BEGIN Insert into audit_clinicemployee va... AFTER	2023-04-26 14:31:36	
Audit_employee_AFTER_DELETE	DELETE	clinic_employee	BEGIN Insert into audit_clinicemployee va... AFTER	2023-04-26 14:32:02	
Audit_TreatmentRoom_AFTER_INSERT	INSERT	clinic_treatment...	BEGIN Insert into audit_treatmentroom v... AFTER	2023-04-26 14:32:26	
Audit_TreatmentRoom_AFTER_UPDATE	UPDATE	clinic_treatment...	BEGIN Insert into audit_treatmentroom v... AFTER	2023-04-26 14:32:48	
- Output Window:** Shows the results of the executed queries:

#	Time	Action	Message	Duration / Fetch
20	11:23:04	SHOW INDEX FROM clinic_treatment_room	3 row(s) returned	0.062 sec / 0.000 sec
21	11:27:27	SHOW INDEX FROM patient_info	5 row(s) returned	0.016 sec / 0.000 sec
22	11:33:56	SHOW TRIGGERS	38 row(s) returned	0.000 sec / 0.000 sec
- System Tray:** Shows standard Windows icons for network, battery, and system status.

ER diagram:



UML Diagram:



- 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.