

Name : Darshan Puttanna
ID: 801169012

ITIS 6120 8120

Applied Databases-Project 2

*Advanced Features for University Student Clinic Database
System*

Introduction

EMR (Electronic Medical Records) system is a common and easy platform to expedite the functioning of a hospital on a large scale. Due to a large number of health issues in today's scenario, a platform that provides multilateral communication can save a lot of time. Our system gives considerable flexibility for doctors, patients and admin to access this system with ease. This platform provides fast end-to-end process completion from disease diagnosis to patient cure. It aims to improve visibility of all the procedures made in the case of a disease cure. This application will provide business intelligence data for better decision making of staffing, medicines, equipment's etc.

Our project is a database for an EMR system systematized collection of patient and population electronically stored health information in a digital format. These records can be shared across different health care settings. Generally, the EMR system is very complex, often consisting of a huge number of tables. In this project, we developed a database system that serves as the backend of a simple EMR system that captures a small portion of relevant information around a focused care scenario.

Scope

- EMR system is an integrated application for medical practices. Currently, the system plans to cover the general requirements of small unit. We limited our scope based on our domain knowledge and decided to focus on a dental clinic.
- Currently we have 3 user roles: Admin, Doctor, Pharmacist.
- Mentioned user roles are having access to specific tables as required by their role.
- As per role, users are restricted to perform action of other roles.
- System can be used to store appointment details.
- This system can be used to calculate billing details for patients.
- System can handle doctor schedules.
- This system can be used for patient profile creation.

Tools

We have used below tools in our project:

1. MySQL server on Workbench

Requirement's specification

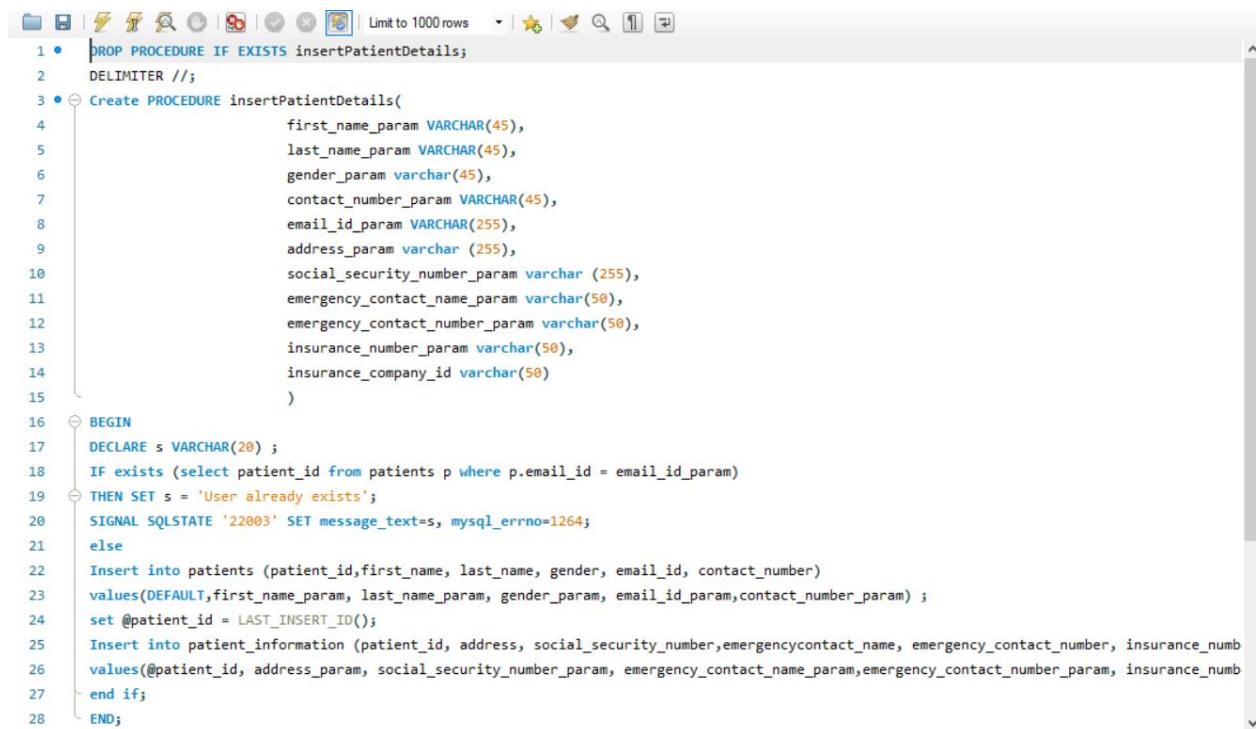
In Project 1, we have created the EMR database for University Student Clinic. We covered almost all requirements in the last project. In this phase, to improve database performance we created stored procedures, indexes, triggers, views and audit trail functionalities.

Stored Procedure:

A stored procedure is a prepared SQL queries that can be save and then just call it to execute It. Stored procedure can be reuse and hence those are efficient. If it contains parameters, it can act based on the parameter value(s) that is passed.

Requirement set 1:

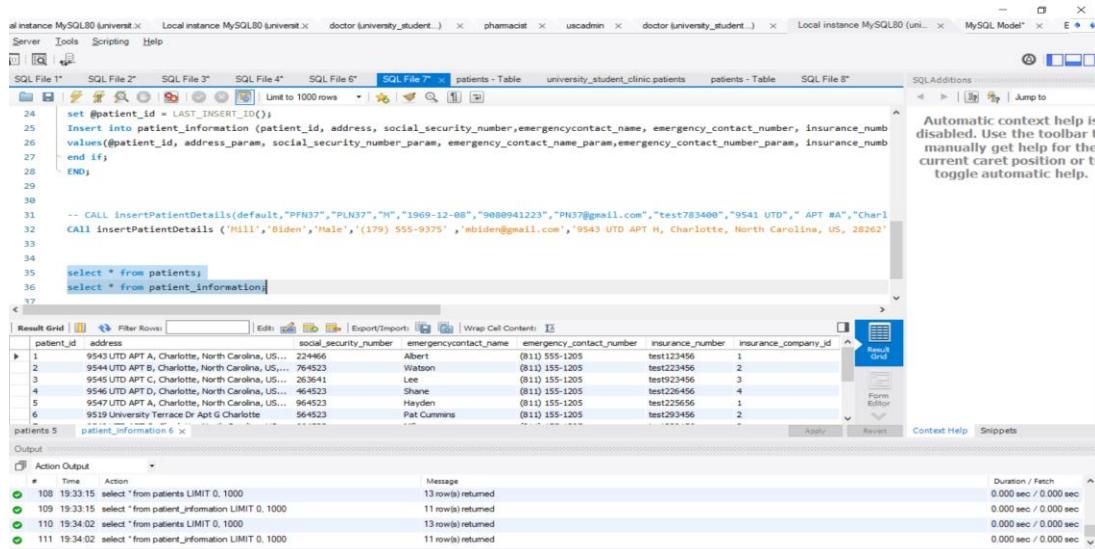
- Create stored procedure to insert patient details in patient table.



The screenshot shows the MySQL Workbench interface with a SQL editor window. The code in the editor is as follows:

```
1 • DROP PROCEDURE IF EXISTS insertPatientDetails;
2 DELIMITER //;
3 • Create PROCEDURE insertPatientDetails(
4     first_name_param VARCHAR(45),
5     last_name_param VARCHAR(45),
6     gender_param varchar(45),
7     contact_number_param VARCHAR(45),
8     email_id_param VARCHAR(255),
9     address_param varchar (255),
10    social_security_number_param varchar (255),
11    emergency_contact_name_param varchar(50),
12    emergency_contact_number_param varchar(50),
13    insurance_number_param varchar(50),
14    insurance_company_id varchar(50)
15 )
16 BEGIN
17     DECLARE s VARCHAR(20);
18     IF exists (select patient_id from patients p where p.email_id = email_id_param)
19     THEN SET s = 'User already exists';
20     SIGNAL SQLSTATE '22003' SET message_text=s, mysql_errno=1264;
21     else
22     Insert into patients (patient_id,first_name, last_name, gender, email_id, contact_number)
23     values(DEFAULT,first_name_param, last_name_param, gender_param, email_id_param,contact_number_param) ;
24     set @patient_id = LAST_INSERT_ID();
25     Insert into patient_information (patient_id, address, social_security_number,emergencycontact_name, emergency_contact_number, insurance_num
26     values(@patient_id, address_param, social_security_number_param, emergency_contact_name_param,emergency_contact_number_param, insurance numb
27     end if;
28 END;
```

Result: To show after calling the procedure patient details is inserted.



The screenshot shows the MySQL Workbench interface with multiple tabs open. The main query editor contains the following SQL code:

```

24 set @patient_id = LAST_INSERT_ID();
25 Insert into patient_information (patient_id, address, social_security_number,emergencycontact_name, emergency_contact_number, insurance_number
26 values(@patient_id, address_param, social_security_number_param, emergency_contact_name_param,emergency_contact_number_param, insurance_number
27 end if;
28 END;
29
30
31 -- CALL insertPatientDetails(default,'PFN37','PLN37','N','1969-12-08','9088941223','PN37@gmail.com','test783400','9541 UTD','APT #A','Charl
CALL insertPatientDetails ('Mill', 'Biden', 'Male', '(179) 555-9375', 'mbiden@gmail.com', '9543 UTD APT H, Charlotte, North Carolina, US, 28262'
33
34
35 select * from patients;
36 select * from patient_information;
37

```

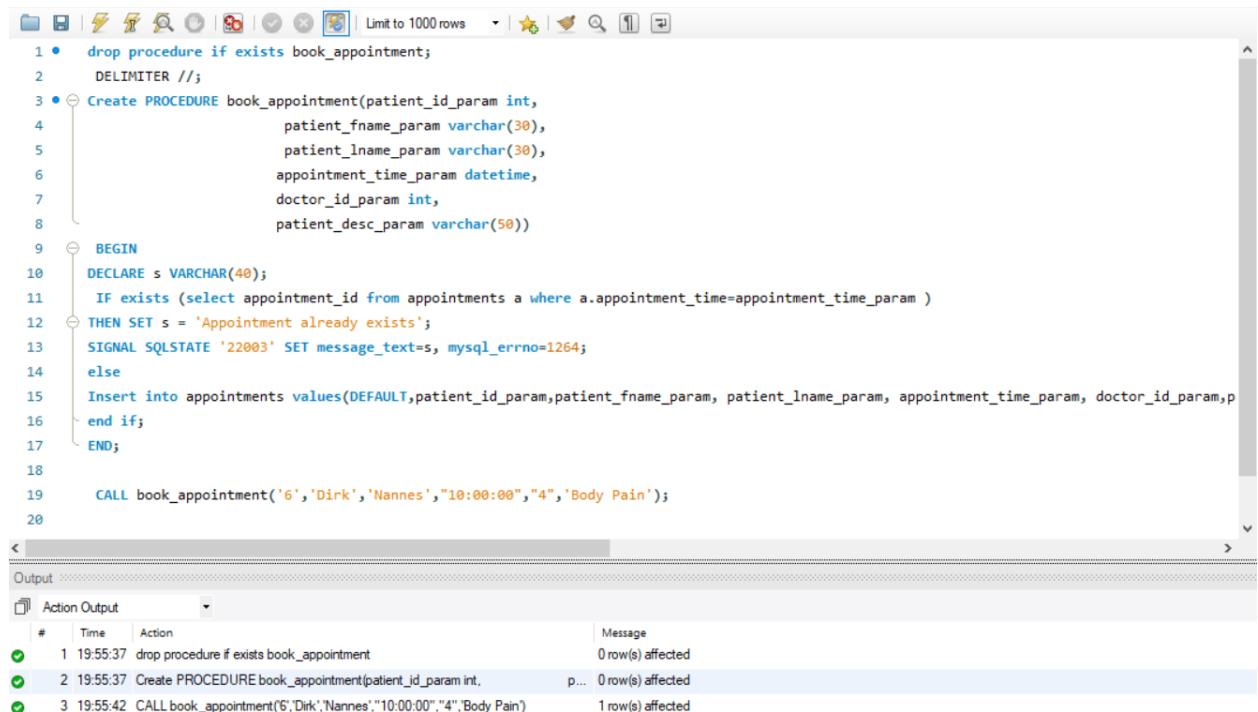
The results grid displays the following data:

patient_id	address	social_security_number	emergencycontact_name	emergency_contact_number	insurance_number	insurance_company_id
1	9543 UTD APT A, Charlotte, North Carolina, US...	224466	Albert	(811) 155-1205	test123456	1
2	9544 UTD APT B, Charlotte, North Carolina, US...	764523	Watson	(811) 155-1205	test223456	2
3	9545 UTD APT C, Charlotte, North Carolina, US...	263641	Lee	(811) 155-1205	test923456	3
4	9546 UTD APT D, Charlotte, North Carolina, US...	464523	Shane	(811) 155-1205	test226456	4
5	9547 UTD APT A, Charlotte, North Carolina, US...	964523	Hayden	(811) 155-1205	test225656	1
6	9519 University Terrace Dr Apt G Charlotte	564523	Pat Cummins	(811) 155-1205	test293456	2

The output pane shows the execution history:

#	Time	Action	Message	Duration / Fetch
108	19:33:15	selected * from patients LIMIT 0, 1000	13 row(s) returned	0.000 sec / 0.000 sec
109	19:33:15	selected * from patient_information LIMIT 0, 1000	11 row(s) returned	0.000 sec / 0.000 sec
110	19:34:02	selected * from patients LIMIT 0, 1000	13 row(s) returned	0.000 sec / 0.000 sec
111	19:34:02	selected * from patient_information LIMIT 0, 1000	11 row(s) returned	0.000 sec / 0.000 sec

- Create stored procedure to insert appointment details in appointmentdetails table.



The screenshot shows the MySQL Workbench interface with the following SQL code in the main editor:

```

1 • drop procedure if exists book_appointment;
2 DELIMITER //;
3 • Create PROCEDURE book_appointment(patient_id_param int,
4                                     patient_fname_param varchar(30),
5                                     patient_lname_param varchar(30),
6                                     appointment_time_param datetime,
7                                     doctor_id_param int,
8                                     patient_desc_param varchar(50))
9 BEGIN
10     DECLARE s VARCHAR(40);
11     IF exists (select appointment_id from appointments a where a.appointment_time=appointment_time_param )
12     THEN SET s = 'Appointment already exists';
13     SIGNAL SQLSTATE '20003' SET message_text=s, mysql_errno=1264;
14     else
15     Insert into appointments values(DEFAULT,patient_id_param,patient_fname_param, patient_lname_param, appointment_time_param, doctor_id_param,
16     end if;
17 END;
18
19     CALL book_appointment('6','Dirk','Nannes','10:00:00','4','Body Pain');
20

```

The output pane shows the execution history:

#	Time	Action	Message
1	19:55:37	drop procedure if exists book_appointment	0 row(s) affected
2	19:55:37	Create PROCEDURE book_appointment(patient_id_param int, p...	0 row(s) affected
3	19:55:42	CALL book_appointment('6','Dirk','Nannes','10:00:00','4','Body Pain')	1 row(s) affected

Name : Darshan Puttanna
ID: 801169012

We can see that patient successfully booked the appointment.

```

10  DECLARE s VARCHAR(40);
11  IF exists (select appointment_id from appointments a where a.appointment_time=appointment_time_param )
12  THEN SET s = 'Appointment already exists';
13  SIGNAL SQLSTATE '22003' SET message_text=s, mysql_errno=1264;
14  else
15  Insert into appointments values(DEFAULT,patient_id_param,patient_fname_param, patient_lname_param, appointment_time_param, doctor_id_param,p
16  end if;
17  END;
18
19  CALL book_appointment('6','Dirk','Nannes','10:00:00','4','Body Pain');
20
21
22  select * from appointments;

```

Result Grid

appointment_id	patient_id	patient_fname	patient_lname	appointment_time	doctor_id	patient_description
8	8	Kaleigh	Eric	2021-04-08 12:30:00	4	Head ache and vomiting sensation
9	9	Virat	Kohli	2021-04-02 12:30:00	6	Pain in Arm
10	10	Joe	Bill	2021-04-04 12:30:00	2	Teeth filling
13	6	Dirk	Nannes	2010-00-00 00:00:00	4	Body Pain
•	HULL	HULL	HULL	HULL	HULL	HULL

Action Output

#	Time	Action	Message	Duration / Fetch
1	19:55:37	drop procedure if exists book_appointment	0 row(s) affected	0.062 sec
2	19:55:37	Create PROCEDURE book_appointment(patient_id_param int,	... 0 row(s) affected	0.000 sec
3	19:55:42	CALL book_appointment('6','Dirk','Nannes','10:00:00','4','Body Pain')	1 row(s) affected	0.016 sec
4	19:56:58	select * from appointments LIMIT 0, 1000	11 row(s) returned	0.000 sec / 0.000 sec

- Create stored procedure to display billing details for patients using visitor_id.

```

1 • drop procedure if exists calculatebillingdetails;
2 DELIMITER //
3 • create procedure calculatebillingdetails( visitor_id_param int )
4 begin
5 select total_bill, concat (a.patient_fname, " ", a.patient_lname ) as patient_name
6   from billing_info b
7 inner join visitors_info v on b.visitor_id=v.visitor_id
8 inner join appointments a on v.appointment_id = a.appointment_id
9 where v.visitor_id=visitor_id_param;
10 end;
11
12 call calculatebillingdetails('2');

```

Result Grid

total_bill	patient_name
100	Alex Hales

Action Output

#	Time	Action	Message	Duration / Fetch
21	20:51:56	call procedure_name_change(7,Tooth Filings)	1 row(s) returned	0.000 sec / 0.000 sec
22	21:00:22	create procedure calculatebillingdetails(visitor_id_param int) begin select total_b...	0 row(s) affected	0.016 sec
23	21:01:08	call calculatebillingdetails('2')	1 row(s) returned	0.016 sec / 0.000 sec

- Create a stored procedure to update procedure list name.

```
DROP PROCEDURE IF EXISTS procedure_name_change;
DELIMITER //
CREATE PROCEDURE procedure_name_change (
proc_id_var int,
proc_status_var varchar(50)
)
BEGIN
DECLARE sql_error tinytext DEFAULT FALSE;
declare proc_name_var varchar(50);
DECLARE CONTINUE HANDLER FOR SQLEXCEPTION

START TRANSACTION;
select procedure_name into proc_name_var from procedure_list where procedure_id=proc_id_var;
UPDATE procedure_details
SET procedure_name = proc_status_var
WHERE procedure_id = proc_id_var;

IF sql_error = FALSE THEN
COMMIT;
SELECT CONCAT('Procedure ',proc_name_var, ' name changed to ',proc_status_var );
ELSE
ROLLBACK;
select ('Procedure name is not updated as error occurred');
END IF;
END//
DELIMITER ;
```

Result: Procedure name is updated.

The screenshot shows the MySQL Workbench interface with several tabs at the top: SQL File 1*, SQL File 2*, SQL File 3*, SQL File 4*, SQL File 6*, SQL File 7*, SQL File 9*, SQL File 10*, SQL File 11*, and SQL File 12*. The SQL File 12* tab is active, displaying the stored procedure code. Below the code, the Result Grid shows the output of the SELECT statement: "CONCAT('Procedure ',proc_name_var, ' name changed to ',proc_status_var)". The result is: "Procedure Repairs name changed to Tooth Fillings". At the bottom, the Action Output table provides detailed information about the execution:

#	Time	Action	Message	Duration / Fetch
17	20:48:42	call procedure_name_change(7,'Tooth Fillings')	1 row(s) returned	0.016 sec / 0.000 sec
18	20:50:25	call procedure_name_change(7,'Tooth Fillings')	1 row(s) returned	0.000 sec / 0.000 sec
19	20:51:50	DROP PROCEDURE IF EXISTS procedure_name_change	0 row(s) affected	0.015 sec
20	20:51:50	CREATE PROCEDURE procedure_name_change (proc_id_var int, proc_stat...	0 row(s) affected	0.000 sec
21	20:51:56	call procedure_name_change(7,'Tooth Fillings')	1 row(s) returned	0.000 sec / 0.000 sec

Name : Darshan Puttanna
ID: 801169012

- Create stored procedure to update employee(doctor) details.

```

SQL File 1* SQL File 2* SQL File 3* SQL File 4* SQL File 6* SQL File 7* SQL File 9* SQL File 10* SQL File 11* SQL File 12* SQL File 13* SQL Additions ..... SQL
Limit to 1000 rows | Jump to
1 • drop procedure if exists updateEmployeeDetails;
2 DELIMITER //
3 • create procedure updateEmployeeDetails(doctor_id_param int, contact_number_param varchar(45), email_param varchar(255), specialization_param
4 begin
5 update doctors set contact_number = contact_number_param, email_id = email_param , specialization = specialization_param
6 where doctor_id= doctor_id_param;
7 end;
8
9 CALL updateEmployeeDetails(6, "(998) 988-7655", "jtrott@gmail.com", "Neurologist");
10
11 select * from doctors

```

Result Grid

doctor_id	first_name	last_name	gender	age	email_id	address	contact_number	specialization
1	Edward	Jenner	Male	36	ejenner@gmail.com	9551 UTD APT A, Charlotte, North Carolina, US...	(311) 222-1234	General Physician
2	Elizabeth	Blackwell	Female	45	elizabeth@gmail.com	9552 UTD APT B, Charlotte, North Carolina, US...	(714) 455-8834	Dentist
3	Daniel	Williams	Male	46	dwilliams@gmail.com	9553 UTD APT C, Charlotte, North Carolina, US...	(814) 156-9834	Neurologist
4	Alexander	Fleming	Male	55	alexander@gmail.com	9554 UTD APT D, Charlotte, North Carolina, US...	(914) 455-7834	General Physician
5	Helen	Brooke	Female	41	helen@gmail.com	9555 UTD APT E, Charlotte, North Carolina, US...	(114) 455-2834	Cardiologist
6	Jonathan	Trott	Male	47	jtrott@gmail.com	9556 UTD APT F, Charlotte, North Carolina, US...	(998) 988-7655	Neurologist
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

doctors 2 x

Output:

#	Time	Action	Message	Duration / Fetch
31	21:29:05	create procedure updateEmployeeDetails(doctor_id_param int, contact_num...	0 row(s) affected	0.000 sec
32	21:29:11	CALL updateEmployeeDetails(6, "(998) 988-7655", "jtrott@gmail.com", "Ne...	1 row(s) affected	0.015 sec
33	21:29:16	select * from doctors LIMIT 0, 1000	6 row(s) returned	0.000 sec / 0.000 sec

- Create stored procedure to display employee details for particular employee ID.

```

SQL File 1* SQL File 2* SQL File 3* SQL File 4* SQL File 6* SQL File 7* SQL File 9* SQL File 10* SQL Additions ..... SQL
Limit to 1000 rows | Jump to
1 • drop procedure if exists get_appointmentDetailsByPatientId;
2 DELIMITER //
3 • create procedure get_appointmentDetailsByPatientId ( patient_id_param int)
4 begin
5 select * from appointmentdetails_view where patient_id = patient_id_param ;
6 end;
7
8 select * from appointmentdetails_view;

```

Result Grid

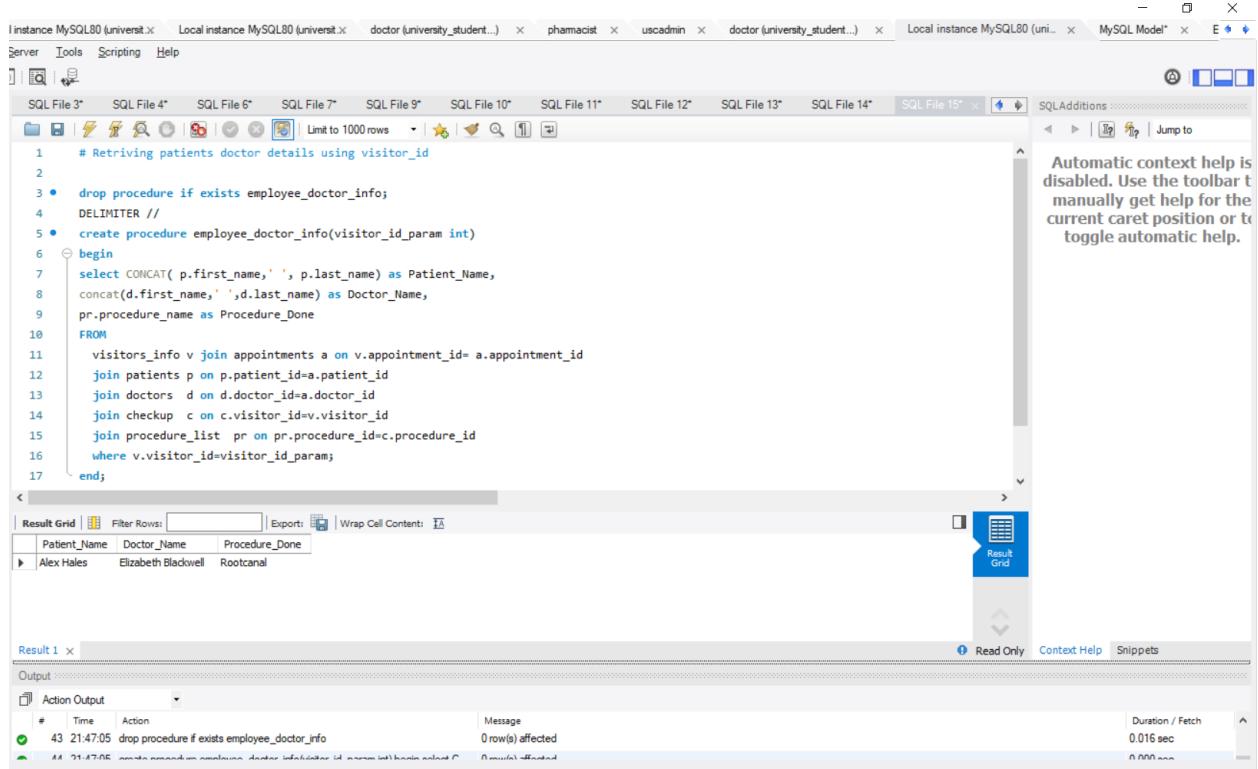
appointment_id	patient_id	appointment_time	Patient_name	Doctor_name
1	1	2021-04-03 12:30:00	Dwane Bravo	Edward Jenner
2	2	2021-04-04 12:30:00	Alex Hales	Elizabeth Blackwell
3	3	2021-12-04 16:30:00	Liana Lukas	Daniel Williams
4	4	2021-04-05 14:10:00	Quinton Boucher	Alexander Fleming
5	5	2021-04-06 11:40:00	Mark Knight	Helen Brooke

tdetails view1 x

Output:

#	Time	Action	Message	Duration / Fetch
3	19:55:42	CALL book_appointment('6','Dirk','Nannes','10:00:00','4','Body Pain')	1 row(s) affected	0.016 sec
4	19:56:58	select * from appointments LIMIT 0, 1000	11 row(s) returned	0.000 sec / 0.000 sec
5	20:10:13	drop procedure if exists get_appointmentDetailsByPatientId	0 row(s) affected, 1 warning(s): 1305 PROCEDURE university_student_clinic_get_appointmentDetailsByPatientId does not exist	0.015 sec
6	20:10:13	create procedure get_appointmentDetailsByPatientId (patient_id_param int) b...	0 row(s) affected	0.000 sec
7	20:11:20	select * from appointmentdetails_view LIMIT 0, 1000	11 row(s) returned	0.000 sec / 0.000 sec

- Create stored procedure to retrieve patient and consulted doctor details using visitor_id.



```

1 # Retrieving patients doctor details using visitor_id
2
3 • drop procedure if exists employee_doctor_info;
4 DELIMITER //
5 • create procedure employee_doctor_info(visitor_id_param int)
6 begin
7 select CONCAT(p.first_name, ' ', p.last_name) as Patient_Name,
8 concat(d.first_name, ' ', d.last_name) as Doctor_Name,
9 pr.procedure_name as Procedure_Done
10 FROM
11 visitors_info v join appointments a on v.appointment_id= a.appointment_id
12 join patients p on p.patient_id=a.patient_id
13 join doctors d on d.doctor_id=a.doctor_id
14 join checkup c on c.visitor_id=v.visitor_id
15 join procedure_list pr on pr.procedure_id=c.procedure_id
16 where v.visitor_id=visitor_id_param;
17 end;

```

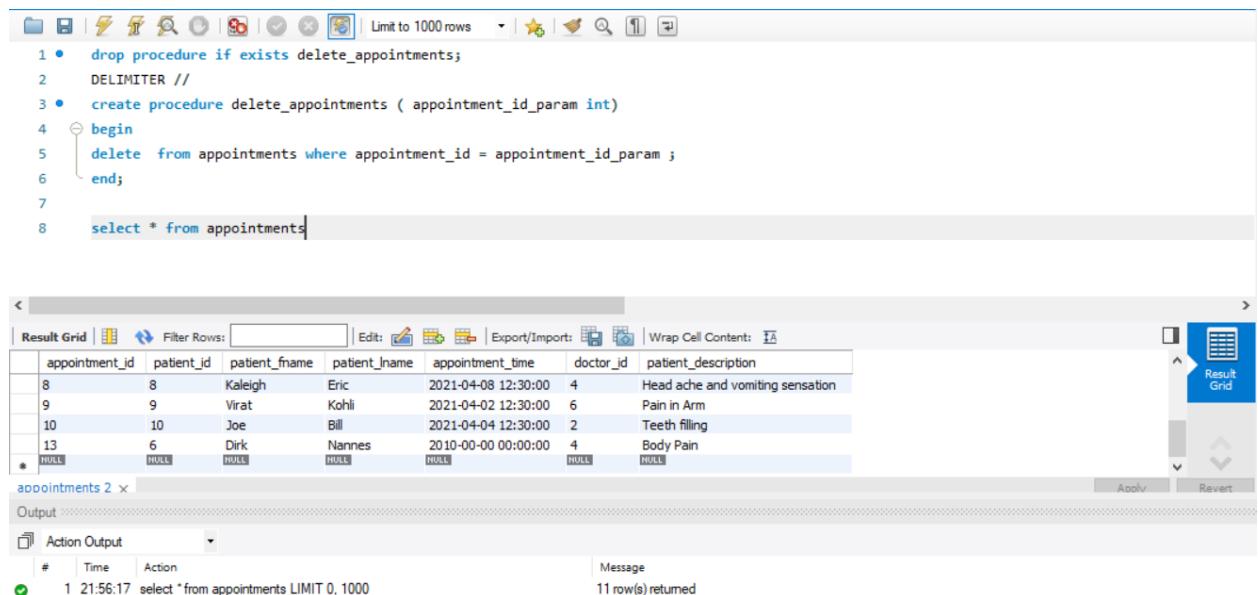
Patient_Name	Doctor_Name	Procedure_Done
Alex Hales	Elizabeth Blackwell	Rootcanal

Result 1 ×

Action Output

#	Time	Action	Message
43	21:47:05	drop procedure if exists employee_doctor_info	0 row(s) affected
44	21:47:05	create procedure employee_doctor_info(visitor_id_param int)	0 rows affected

- Create stored procedure to delete the appointment.



```

1 • drop procedure if exists delete_appointments;
2
3 • create procedure delete_appointments ( appointment_id_param int)
4 begin
5 delete from appointments where appointment_id = appointment_id_param ;
6 end;
7
8 select * from appointments;

```

appointment_id	patient_id	patient_fname	patient_lname	appointment_time	doctor_id	patient_description
8	8	Kaleigh	Eric	2021-04-08 12:30:00	4	Head ache and vomiting sensation
9	9	Virat	Kohli	2021-04-02 12:30:00	6	Pain in Arm
10	10	Joe	Bill	2021-04-04 12:30:00	2	Teeth filling
13	6	Dirk	Nannes	2010-04-00 00:00:00	4	Body Pain
*	HULL	HULL	HULL	HULL	HULL	HULL

appointments 2 ×

Action Output

#	Time	Action	Message
1	21:56:17	select * from appointments LIMIT 0, 1000	11 row(s) returned

Name : Darshan Puttanna
ID: 801169012

From the below screenshot we can see that appointments with appointment_id = 13;

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

SQL Editor:

```
SQL File 4* SQL File 5* SQL File 7* SQL File 9* SQL File 10* SQL File 11* SQL File 12* SQL File 13* SQL File 14* SQL File 15* SQL File 16* SQL Additions
1 • drop procedure if exists delete_appointments;
2 DELIMITER //
3 • create procedure delete_appointments ( appointment_id_param int)
4 begin
5     delete from appointments where appointment_id = appointment_id_param ;
6 end;
7
8 call delete_appointments('13');
9 select * from appointments;
10
```

Result Grid:

appointment_id	patient_id	patient_fname	patient_lname	appointment_time	doctor_id	patient_description
7	7	Meg		2021-04-07 12:30:00	1	Cold and Cough
8	8	Kaleigh	Eric	2021-04-08 12:30:00	4	Head ache and vomiting sensation
9	9	Virat	Kohli	2021-04-02 12:30:00	6	Pain in Arm
10	10	Joe	Bill	2021-04-04 12:30:00	2	Teeth filling
HULL	HULL	HULL	HULL	HULL	HULL	HULL

Action Output:

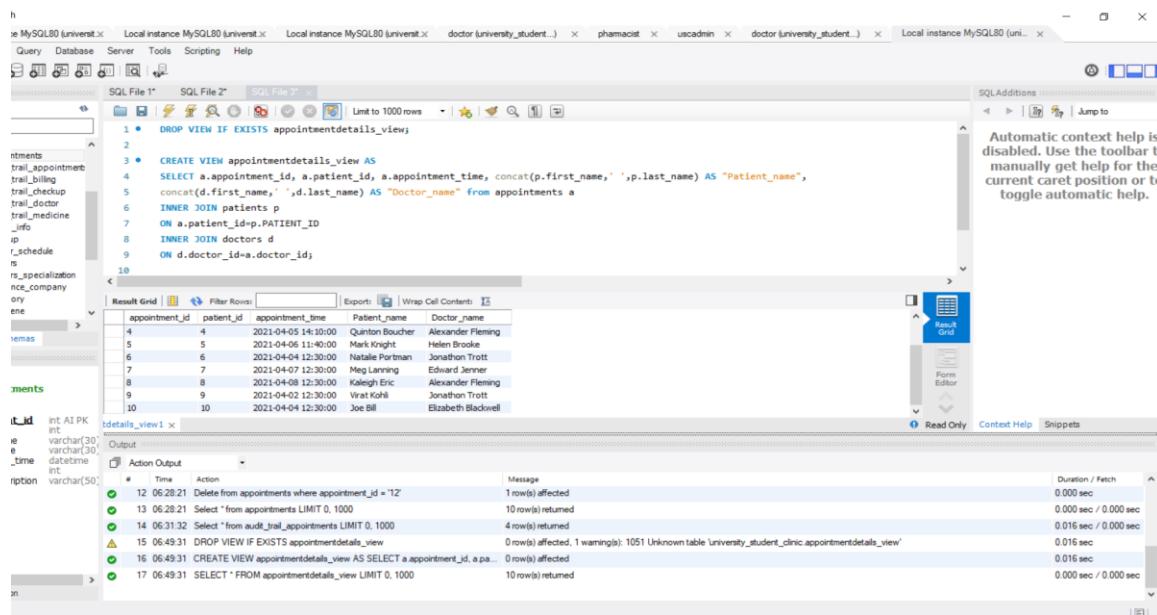
#	Time	Action	Message	Duration / Fetch
1	21:59:13	drop procedure if exists delete_appointments	0 row(s) affected, 1 warning(s): 1305 PROCEDURE university_student_clinic.delete_appointments does not exist	0.016 sec
2	21:59:13	create procedure delete_appointments (appointment_id_param int) begin delet...	0 row(s) affected	0.016 sec
3	21:59:18	call delete_appointments('13')	1 row(s) affected	0.031 sec
4	21:59:33	select * from appointments LIMIT 0, 1000	10 row(s) returned	0.000 sec / 0.000 sec

Views:

A view is a virtual table based on the result-set of an SQL queries. It contains rows and columns. Columns in view are the columns from one or more real tables in the database. We can add WHERE and JOIN statements to a view and present the data as if the data were coming from one single table.

Requirement set 2:

- Create a view to display appointment details.



The screenshot shows the MySQL Workbench interface with multiple tabs open. The main tab displays the following SQL code for creating a view:

```

1 • DROP VIEW IF EXISTS appointmentdetails_view;
2
3 • CREATE VIEW appointmentdetails_view AS
4   SELECT a.appointment_id, a.patient_id, a.appointment_time, concat(p.first_name, ' ', p.last_name) AS "Patient_name",
5   concat(d.first_name, ' ', d.last_name) AS "Doctor_name" FROM appointments a
6   INNER JOIN patients p
7   ON a.patient_id=p.PATIENT_ID
8   INNER JOIN doctors d
9   ON d.doctor_id=a.doctor_id
10

```

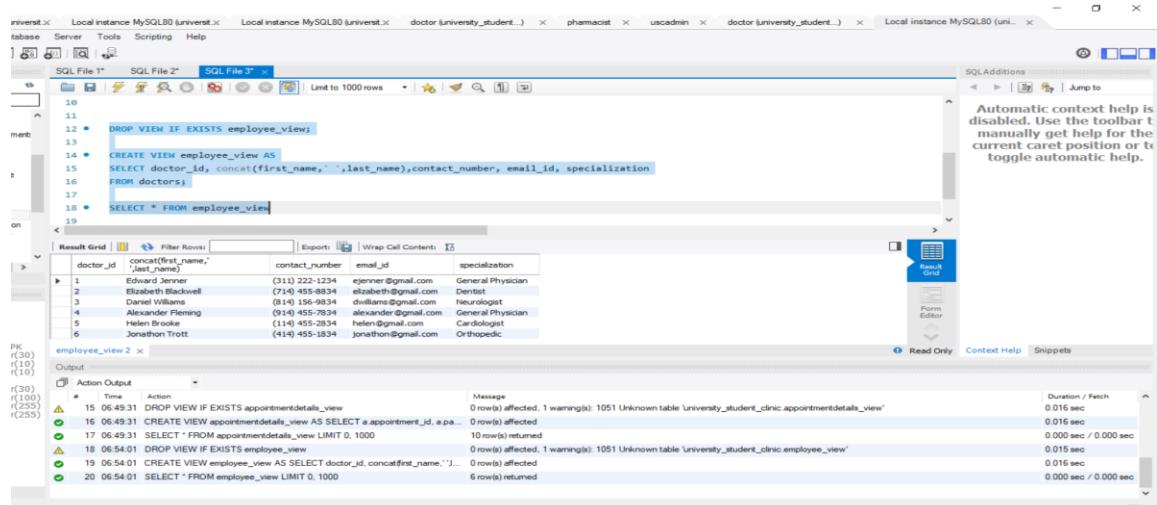
The results grid shows 10 rows of appointment details:

appointment_id	patient_id	appointment_time	Patient_name	Doctor_name
4	4	2021-04-05 14:10:00	Quinton Boucher	Alexander Fleming
5	5	2021-04-05 14:10:00	Mark Knight	Helen Brooke
6	6	2021-04-05 12:30:00	John Chapman	Joseph Everett
7	7	2021-04-07 12:30:00	Meg Lanning	Edward Jenner
8	8	2021-04-08 12:30:00	Kaleigh Eric	Alexander Fleming
9	9	2021-04-02 12:30:00	Wrat Kohi	Jonathon Trott
10	10	2021-04-04 12:30:00	Joe Bill	Elizabeth Blackwell

The action output pane shows the execution history:

- 12. 06:28:21 Delete from appointments where appointment_id = '12' 1 row(s) affected 0.000 sec
- 13. 06:28:21 Select * from appointments LIMIT 0, 1000 10 row(s) returned 0.000 sec / 0.000 sec
- 14. 06:31:32 Select * from aud_it_appointments LIMIT 0, 1000 4 row(s) returned 0.016 sec
- 15. 06:49:31 DROP VIEW IF EXISTS appointmentdetails_view 0 row(s) affected, 1 warning(s): 1051 Unknown table 'university_student_clinic.appointmentdetails_view'
- 16. 06:49:31 CREATE VIEW appointmentdetails_view AS SELECT a.appointment_id, a.p... 0 row(s) affected 0.016 sec
- 17. 06:49:31 SELECT * FROM appointmentdetails_view LIMIT 0, 1000 10 row(s) returned 0.000 sec / 0.000 sec

- Create a view to display doctor details.



The screenshot shows the MySQL Workbench interface with multiple tabs open. The main tab displays the following SQL code for creating a view:

```

10
11
12 • DROP VIEW IF EXISTS employee_view;
13
14 • CREATE VIEW employee_view AS
15   SELECT doctor_id, concat(first_name, ' ', last_name), contact_number, email_id, specialization
16   FROM doctors;
17
18 • SELECT * FROM employee_view;
19

```

The results grid shows 6 rows of doctor details:

doctor_id	concat(first_name, ' ', last_name)	contact_number	email_id	specialization
1	Edward Jenner	(311) 222-1234	edward@gmail.com	General Physician
2	Elizabeth Blackwell	(714) 555-9994	elizabeth@gmail.com	Dentist
3	Daniel Williams	(814) 156-9834	dwilliams@gmail.com	Neurologist
4	Alexander Fleming	(914) 455-7834	alexander@gmail.com	General Physician
5	Helen Brooke	(114) 455-2834	helen@gmail.com	Cardiologist
6	Jonathon Trott	(414) 455-1834	jonathon@gmail.com	Orthopedic

The action output pane shows the execution history:

- 15. 06:49:31 DROP VIEW IF EXISTS appointmentdetails_view 0 row(s) affected, 1 warning(s): 1051 Unknown table 'university_student_clinic.appointmentdetails_view'
- 16. 06:49:31 CREATE VIEW appointmentdetails_view AS SELECT a.appointment_id, a.p... 0 row(s) affected 0.016 sec
- 17. 06:49:31 SELECT * FROM appointmentdetails_view LIMIT 0, 1000 10 row(s) returned 0.000 sec / 0.000 sec
- 18. 06:54:01 DROP VIEW IF EXISTS employee_view 0 row(s) affected, 1 warning(s): 1051 Unknown table 'university_student_clinic.employee_view'
- 19. 06:54:01 CREATE VIEW employee_view AS SELECT doctor_id, concat(first_name, ' ', last_name)
- 20. 06:54:01 SELECT * FROM employee_view LIMIT 0, 1000 6 row(s) returned 0.016 sec

- Create a view to display medicine stock in inventory.

The screenshot shows the MySQL Workbench interface with multiple tabs open. The SQL Editor tab contains the following SQL code:

```
13
14 • DROP VIEW IF EXISTS employee_view;
15 CREATE VIEW employee_view AS
16 SELECT doctor_id, concat(first_name, ' ', last_name), contact_number, email_id, specialization
17 FROM doctors;
18
19 • SELECT * FROM employee_view;
20
21 • DROP VIEW IF EXISTS stock_remaining_view;
```

The Result Grid shows the data from the stock_remaining_view:

medicine_name	stock_remaining
Medicine1	10
Medicine2	154
Medicine3	150
Medicine4	225
Medicine5	9
Medicine6	118
Medicine1	100

The Action Output pane shows the execution log:

#	Time	Action	Message	Duration / Fetch
20	06:54:01	SELECT * FROM employee_view LIMIT 0, 1000	6 row(s) returned	0.000 sec / 0.000 sec
21	06:56:03	select medicene.medicine_name, inventory.stock_remaining from medicene i...	12 row(s) returned	0.016 sec / 0.000 sec
22	06:56:03	select * from inventory LIMIT 0, 1000	12 row(s) returned	0.000 sec / 0.000 sec
23	07:00:28	DROP VIEW IF EXISTS stock_remaining_view	0 row(s) affected, 1 warning(s): 1051 Unknown table 'university_student_clinic.stock_remaining_view'	0.016 sec
24	07:00:28	CREATE VIEW stock_remaining_view AS select medicene.medicine_name, ...	0 row(s) affected	0.015 sec
25	07:00:28	select * from stock_remaining_view LIMIT 0, 1000	12 row(s) returned	0.000 sec / 0.000 sec

- Create a view to display patient's and consulted doctor as patient_checkup_view.

The screenshot shows the MySQL Workbench interface with multiple tabs open. The SQL Editor tab contains the following SQL code:

```
32 • CREATE VIEW patient_checkup_view AS
33 select concat(p.first_name, ' ', p.last_name) as Patient_Name,
34 concat(d.first_name, ' ', d.last_name) as Doctor_Name,
35 pr.procedure_name as Procedure_Done
36
37     visitors_info v join appointments a on v.appointment_id= a.appointment_id
38     join patients p on p.patient_id=a.patient_id
39     join doctors d on d.doctor_id=a.doctor_id
40     join checkup c on c.visitor_id=v.visitor_id
41     join procedure_list pr on pr.procedure_id=c.procedure_id;
42
43 • Select * from patient_checkup_view;
```

The Result Grid shows the data from the patient_checkup_view:

Patient_Name	Doctor_Name	Procedure_Done
Dwane Bravio	Edward Jenner	Body Checkup
Alex Hales	Elizabeth Blackwell	Rootcanal
Liana Lukas	Daniel Williams	EEG
Quinton Boucher	Alexander Fleming	Body Checkup
Mark Knight	Helen Brooke	ECG
Natalie Portman	Jonathon Trott	XRAY
Meg Lanning	Edward Jenner	Body Checkup
Kaleigh Eric	Alexander Fleming	Body Checkup
Virat Kohli	Jonathon Trott	Plaster of paris
Joe Bill	Elizabeth Blackwell	Repars

The Action Output pane shows the execution log:

#	Time	Action	Message	Duration / Fetch
23	07:00:28	DROP VIEW IF EXISTS stock_remaining_view	0 row(s) affected, 1 warning(s): 1051 Unknown table 'university_student_clinic.stock_remaining_view'	0.016 sec
24	07:00:28	CREATE VIEW stock_remaining_view AS select medicene.medicine_name, ...	0 row(s) affected	0.015 sec

Name : Darshan Puttanna
ID: 801169012

- Create a view to display billing details.

The screenshot shows the MySQL Workbench interface with multiple tabs open. The SQL Editor tab contains the following code:

```
46
47
48 • DROP VIEW IF EXISTS billing_details_view;
49
50 • CREATE VIEW billing_details_view AS
51 select total_bill, concat(a.patient_fname, " ", a.patient_lname) as patient_name
52 from billing_info b
53 inner join visitors_info v on b.visitor_id=v.visitor_id
54 inner join appointments a on v.appointment_id = a.appointment_id;
55
56 • select * from billing_details_view;
```

The Result Grid shows the output of the query:

total_bill	patient_name
50	Dwane Bravo
100	Alex Hales
1500	Liam Lukas
50	Quinton
260	Mark
750	Natalie
50	Meg
50	Kaleigh Eric
750	Virat Kohli

The Output pane shows the execution log:

Action	Time	Action	Message	Duration / Fetch
28	07:06:10	Select * from patient_checkup_view LIMIT 0, 1000	10 row(s) returned	0.016 sec / 0.000 sec
29	07:12:10	DROP VIEW IF EXISTS billing_details_view	0 row(s) affected, 1 warning(s): 1051 Unknown table 'university_student_clinic.billing_details_view'	0.016 sec
30	07:12:10	CREATE VIEW billing_details_view AS select total_bill, concat (a.patient_fna...	0 row(s) affected	0.015 sec
31	07:12:10	select * from billing_details_view LIMIT 0, 1000	10 row(s) returned	0.000 sec / 0.000 sec

Name : Darshan Puttanna
ID: 801169012

Index:

These are used to retrieve data from the database more quickly than usual. The users cannot see the indexes but indexes are just used to speed up searches/queries.

Requirement set 3:

We created the index for columns which we are frequently using to fetch the data necessary for the requirements of our project.

- Create index for medicine name column of medicine table.

```
CREATE INDEX idx_medicine_name
ON medicine (medicine_name);
select medicine_name from medicine;
```

Execution Plan (Medicine)

```
Query cost: 1.45
query_block #1
  1.45  12 rows
    Full Index Scan
      medicine
      idx_medicine_name
```

Action Output (Medicine)

#	Time	Action	Message
38	07:25:24	EXPLAIN FORMAT=JSON select medicine_name from medicine	OK
39	07:25:38	CREATE INDEX idx_medicine_name ON medicine (medicine_name)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
40	07:25:44	select medicine_name from medicine LIMIT 0, 1000	12 row(s) returned
41	07:25:49	EXPLAIN select medicine_name from medicine	OK

- Create index for stock remaining column of inventory table.

```
CREATE INDEX idx_medicine_name
ON medicine (medicine_name);
select medicine_name from medicine;
CREATE INDEX idx_stock_remaining
ON inventory (stock_remaining);
select stock_remaining from inventory;
```

Execution Plan (Inventory)

```
Query cost: 2.20
query_block #1
  2.2  12 rows
    Full Index Scan
      inventory
      idx_stock_remaining
```

Action Output (Inventory)

#	Time	Action	Message
42	07:25:49	EXPLAIN FORMAT=JSON select medicine_name from medicine	OK
43	07:27:36	CREATE INDEX idx_stock_remaining ON inventory (stock_remaining)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
44	07:27:40	select stock_remaining from inventory LIMIT 0, 1000	12 row(s) returned
45	07:27:43	EXPLAIN select stock_remaining from inventory	OK

Name : Darshan Puttanna
ID: 801169012

- Create index for insurance company column of insurance company table.

```

CREATE INDEX idx_insurance_plan
ON insurance_company (insurance_plan);

select insurance_plan from insurance_company;
  
```

Index: idx_insurance_plan

Definition:

Type: BTREE
Unique: No
Visible: Yes
Columns: insurance_plan

Object Info Session

- Create index for symptoms description column of checkup table.

```

CREATE INDEX idx_symptoms_desc
ON checkup (symptoms_desc);

select symptoms_desc from checkup;
  
```

Index: idx_symptoms_desc

Definition:

Type: BTREE
Unique: No
Visible: Yes
Columns: symptoms_desc

Object Info Session

- Create index for visit date column of visit_information table.

```

CREATE INDEX idx_visited_time
ON visitors_info (visited_time);

select visited_time from visitors_info;
  
```

Table: visitors_info

Columns:

visitor_id int AI PK
appointment_id varchar(100)
visitor_description varchar(100)
visited_time datetime

Object Info Session

Authentication for different roles:

We have 3 roles in our project:

Requirement set 4:

Create 3 users and grant them access for different operation as required.

```
DROP USER IF EXISTS 'uscadmin'@'localhost';
DROP USER IF EXISTS 'doctor'@'localhost';
DROP USER IF EXISTS 'pharmacist'@'localhost';

CREATE USER 'uscadmin'@'localhost' IDENTIFIED BY 'uscadmin123456';
CREATE USER 'doctor'@'localhost' IDENTIFIED BY 'doctor123456';
CREATE USER 'pharmacist'@'localhost' IDENTIFIED BY 'pharmacist123456';

set password for 'uscadmin'@'localhost' = '123456';
set password for 'doctor'@'localhost' = '123456';
set password for 'pharmacist'@'localhost' = '123456';
```

1. Admin:

Access: Admin has access for all the tables

```
GRANT ALL ON university_student_clinic.* TO 'uscadmin'@'localhost';
```

Views: appointmentdetails_view, employee_view

2. Doctor

Read access:

Tables: medicine, appointments, billing_info, medicine, patients, procedure_list, symptoms

Views: appointmentdetails_view, employee_view,

Name : Darshan Puttanna
ID: 801169012

Read and write access:

```
GRANT SELECT ON university_student_clinic.appointments TO 'doctor'@'localhost';
GRANT SELECT ON university_student_clinic.billing_info TO 'doctor'@'localhost';
GRANT SELECT,INSERT, UPDATE ON university_student_clinic.checkup TO 'doctor'@'localhost';
GRANT SELECT,INSERT, UPDATE ON university_student_clinic.doctors TO 'doctor'@'localhost';
GRANT SELECT,INSERT, UPDATE ON university_student_clinic.doctor_schedule TO 'doctor'@'localhost';
GRANT SELECT ON university_student_clinic.medicene TO 'doctor'@'localhost';
GRANT SELECT ON university_student_clinic.patients TO 'doctor'@'localhost';
GRANT SELECT, INSERT, UPDATE, DELETE ON university_student_clinic.prescription TO 'doctor'@'localhost';
GRANT SELECT ON university_student_clinic.procedure_list TO 'doctor'@'localhost';
GRANT SELECT ON university_student_clinic.symptoms TO 'doctor'@'localhost';
```

3. Pharmacist

Read access:

Tables: checkup

Views: stock_remaining_view, patient_checkup_view

Read and write access:

Tables: inventory, medicine

```
GRANT SELECT,INSERT, UPDATE, DELETE ON university_student_clinic.inventory TO 'pharmacist'@'localhost';
GRANT SELECT,INSERT, UPDATE, DELETE ON university_student_clinic.medicene TO 'pharmacist'@'localhost';
GRANT SELECT ON university_student_clinic.checkup TO 'pharmacist'@'localhost';
```

Role based scenarios:

Requirement set 5:

1. Doctor can update the checkup details but pharmacist can only view those details. They are not allowed to update it.

Updated for doctor role.

The screenshot shows the MySQL Workbench interface with the 'doctor' role selected. In the 'Query Editor' tab, two queries are run:

```
1 • update checkup
2   set result = 'Not detected'
3   where visitor_id = '1'
4
5 • select * from checkup;
```

The 'Result Grid' shows the updated data in the 'checkup' table:

checkup_id	visitor_id	symptoms_desc	procedure_id	doctor_id	result	doctor_comment
1	1	Body Pain	1	1	Not detected	need some medicines
2	2	Tooth Pain	2	2	tooth infection	need for other tooth
3	3	Acute Spinal Cord Injury	5	3	spinal cord injury	x-ray result needed
4	4	Body Pain	1	4	Fever	take mentioned medicines
5	5	Chest Pain	4	5	Mild heart attack	ECG needed
6	6	Knee Pain	6	6	knee injury	Need of x-ray

The 'Output' pane shows the execution log:

- Time Action Message Duration / Fetch
- 04:14:00 update checkup set result = 'Not detected' where visitor_id = '1' 0 rows affected Rows matched: 1 Changed: 0 Warnings: 0 0.000 sec / 0.000 sec
- 04:14:00 select * from checkup LIMIT 0,1000 10 rows returned 0.000 sec / 0.000 sec

Only pharmacist can view.

The screenshot shows the MySQL Workbench interface with the 'pharmacist' role selected. In the 'Query Editor' tab, a single query is run:

```
1
2
3 • select * from checkup;
```

The 'Result Grid' shows the same data as the previous screenshot:

checkup_id	visitor_id	symptoms_desc	procedure_id	doctor_id	result	doctor_comment
1	1	Body Pain	1	1	Not detected	need some medicines
2	2	Tooth Pain	2	2	tooth infection	need for other tooth
3	3	Acute Spinal Cord Injury	5	3	spinal cord injury	x-ray result needed
4	4	Body Pain	1	4	Fever	take mentioned medicines
5	5	Chest Pain	4	5	Mild heart attack	ECG needed
6	6	Knee Pain	6	6	knee injury	Need of x-ray

The 'Output' pane shows the execution log:

- Time Action Message Duration / Fetch
- 04:16:49 select * from checkup LIMIT 0,1000 10 rows returned 0.000 sec / 0.000 sec

Name : Darshan Puttanna

ID: 801169012

But cannot update.

The screenshot shows the MySQL Workbench interface. In the top navigation bar, the database is set to 'doctor'. The left sidebar shows the schema 'university_student_clinic' with its tables, views, stored procedures, and functions. The main window has a query editor titled 'Query 1' containing the following SQL code:

```
1
2 • update checkup
3   set result = 'detected'
4   where visitor_id = '1';
```

The output pane shows the results of the query execution:

#	Time	Action	Message	Duration / Fetch
1	04:16:43	selected from checkup LIMIT 0, 1000	10 row(s) returned	0.000 sec / 0.000 sec
2	04:18:48	update checkup set result = 'detected' where visitor_id = '1'	Error Code: 1142. UPDATE command denied to user 'pharmacist'@'localhost' for table 'checkup'	0.000 sec

A tooltip on the right side of the interface states: "Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help."

2. Admin can update procedure_list but doctor can only view the procedure_list.

The screenshot shows the MySQL Workbench interface with the database set to 'uscadmin'. The left sidebar shows the schema 'university_student_clinic' with its tables, views, stored procedures, and functions. The main window has a query editor titled 'Query 1' containing the following SQL code:

```
1
2
3 • update procedure_list
4   set procedure_name= 'Repairs'
5   where procedure_id='7';
6
7 • select * from procedure_list;
```

The output pane shows the results of the query execution. It includes a result grid showing the updated data in the 'procedure_list' table:

procedure_id	procedure_name
4	ECG
5	EEG
6	XRAY
7	Repairs
*	NULL

The output pane also displays the log of actions taken:

#	Time	Action	Message
1	04:30:43	update procedure_list set procedure_name= 'Repairs' where procedure_id='7'	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0
2	04:30:43	select * from procedure_list LIMIT 0, 1000	7 row(s) returned

A tooltip on the right side of the interface states: "Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help."

Name : Darshan Puttanna

ID: 801169012

The screenshot shows the MySQL Workbench interface with a query editor window titled 'Query 1'. The code entered is:

```
1 update procedure_list
2 set procedure_name='Teeth Fillings and Repairs'
3 where procedure_id='2';
4
5 select * from procedure_list;
```

The results grid shows the following data:

procedure_id	procedure_name
1	Body Checkup
2	Rootcanal
3	Plaster of paris
4	ECG
5	EKG
6	XRAY

The output pane shows the execution log:

#	Time	Action	Message	Duration / Fetch
1	04:32:42	update procedure_list set procedure_name='Teeth Fillings and Repairs' where procedure_id='2';	Error Code: 1142. UPDATE command denied to user 'doctor'@'localhost' for table 'procedure_list'	0.000 sec / 0.000 sec
2	04:32:55	select * from procedure_list LIMIT 0, 1000	7 row(s) returned	0.016 sec / 0.000 sec

3. Pharmacist can update the medicine details but doctor will have only view access to it.

The screenshot shows the MySQL Workbench interface with a query editor window titled 'Query 1'. The code entered is:

```
1 update medicine
2 set amount = '100'
3 where medicine_id = '1001';
4
5 select * from medicines;
```

The results grid shows the following data:

medicine_id	medicine_name	manufacture	amount
1001	Medicine1	Manufacture1	100
1002	Medicine2	Manufacture1	4
1003	Medicine3	Manufacture1	7
1004	Medicine4	Manufacture1	10
1005	Medicine5	Manufacture1	4
1006	Medicine6	Manufacture1	6

The output pane shows the execution log:

#	Time	Action	Message	Duration / Fetch
1	04:16:49	select * from checkup LIMIT 0, 1000	10 row(s) returned	0.000 sec / 0.000 sec
2	04:18:48	update checkup set result = 'detected' where visitor_id = '1'	Error Code: 1142. UPDATE command denied to user 'pharmacist'@'localhost' for table 'checkup'	0.000 sec
3	04:42:36	update medicine set amount = '100' where medicine_id = '1001'	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0	0.016 sec
4	04:42:36	select * from medicine LIMIT 0, 1000	12 row(s) returned	0.000 sec / 0.000 sec

Name : Darshan Puttanna

ID: 801169012

The screenshot shows the MySQL Workbench interface. In the top navigation bar, there are tabs for Local instance MySQL80 (universt...), Local instance MySQL80 (universt...), doctor (university_student...), pharmacist, uscadmin, and doctor (university_student...). Below the tabs is a toolbar with icons for file operations, search, and help.

The main area contains a "Query 1" tab with the following SQL code:

```
1 • update medicine
2   set amount = '15'
3   where medicine_id = '1001';
4
5 • select * from medicine;
```

Below the code is a "Result Grid" showing the contents of the "medicine" table:

medicine_id	medicine_name	manufacture	amount
1001	Medicine1	Manufacture1	100
1002	Medicine2	Manufacture1	4
1003	Medicine3	Manufacture1	7
1004	Medicine4	Manufacture1	10
1005	Medicine5	Manufacture1	4
1006	Medicine6	Manufacture1	6

On the right side of the interface, there is a "SQLAdditions" panel with the message: "Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help." Below the result grid, there is an "Output" section titled "Action Output" which lists the following activity:

#	Time	Action	Message	Duration / Fetch
1	04:32:42	update procedure_list set procedure_name='Teeth Filings and Repairs' where procedure_id>7	Error Code: 1142: UPDATE command denied to user 'doctor'@'localhost' for table 'procedure_list'	0.000 sec
2	04:32:55	select * from procedure_list LIMIT 0, 1000	7 row(s) returned	0.016 sec / 0.000 sec
3	04:43:50	update medicine set amount = '15' where medicine_id = '1001'	Error Code: 1142: UPDATE command denied to user 'doctor'@'localhost' for table 'medicine'	0.000 sec
4	04:44:05	select * from medicine LIMIT 0, 1000	12 row(s) returned	0.000 sec / 0.000 sec

Audit trail:

We have added some audit tables to track the history of the data when it is accessed by any of the users. The tables will store the user who have accessed the data and the timing at which he accessed the data.

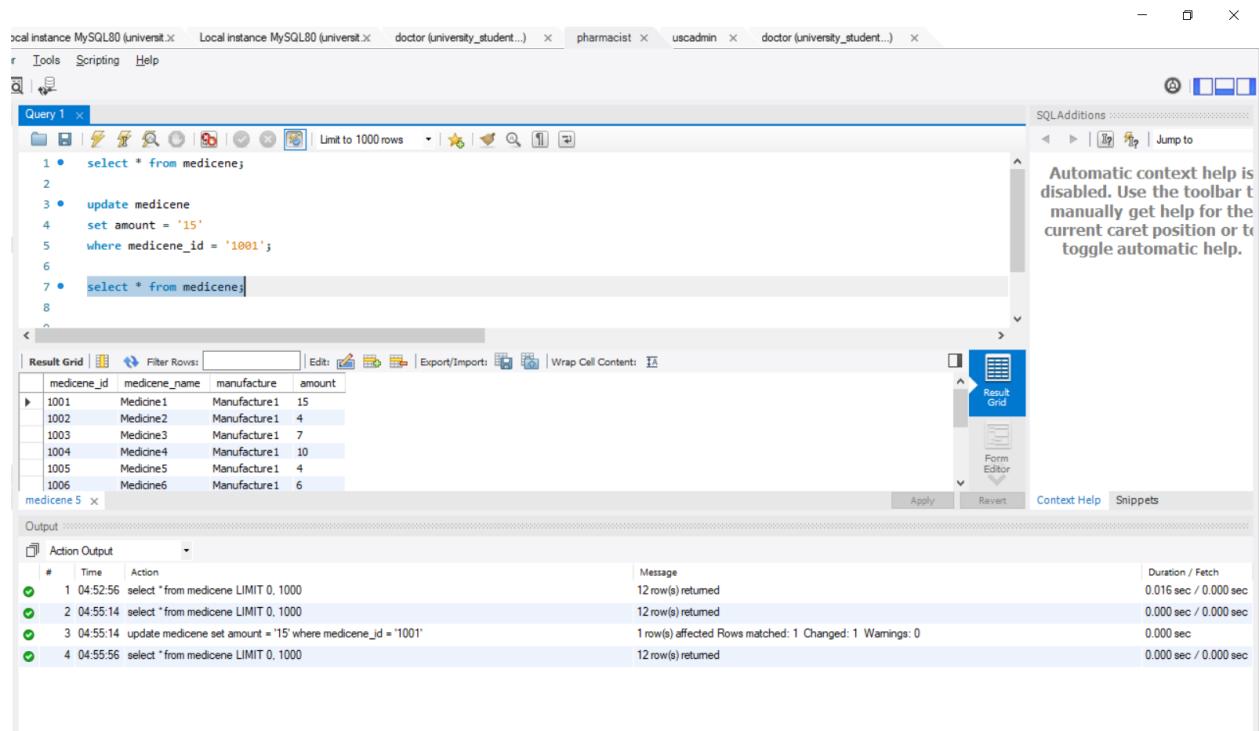
Name : Darshan Puttanna
ID: 801169012

Requirement set 6:

Scenarios:

Insert data in audit table when user perform update action on the data.

1. When pharmacist update medicine amount in medicine table, the old value of medicine should get updated in audit history with timestamp and this can be performed by pharmacist.



The screenshot shows the MySQL Workbench interface. In the Query Editor, the following SQL code is run:

```
1 • select * from medicene;
2
3 • update medicene
4   set amount = '15'
5   where medicene_id = '1001';
6
7 • select * from medicene;
```

The Result Grid displays the following data:

medicene_id	medicene_name	manufacture	amount
1001	Medicine1	Manufacture1	15
1002	Medicine2	Manufacture1	4
1003	Medicine3	Manufacture1	7
1004	Medicine4	Manufacture1	10
1005	Medicine5	Manufacture1	4
1006	Medicine6	Manufacture1	6

The Output pane shows the execution log:

#	Time	Action	Message	Duration / Fetch
1	04:52:56	select *from medicene LIMIT 0, 1000	12 row(s) returned	0.016 sec / 0.000 sec
2	04:55:14	select *from medicene LIMIT 0, 1000	12 row(s) returned	0.000 sec / 0.000 sec
3	04:55:14	update medicene set amount = '15' where medicene_id = '1001'	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0	0.000 sec
4	04:55:56	select *from medicene LIMIT 0, 1000	12 row(s) returned	0.000 sec / 0.000 sec

After the update by pharmacist, if we check the audit_trail_medicene by admin we can see the changes. This shows that trigger statement is working.

Name : Darshan Puttanna

ID: 801169012

The screenshot shows the MySQL Workbench interface with multiple tabs open. The 'Query 1' tab contains the SQL command: `select * from audit_trail_medicine;`. The results grid shows two rows of data:

ID	medicine_name	amount	timestamp
1	Medicine1	100	2021-05-12 04:42:36
2	Medicine1	15	2021-05-12 04:55:14

The 'Output' pane below the results grid displays the execution log:

#	Time	Action	Message	Duration / Fetch
1	04:30:43	update procedure_list set procedure_name='Repairs' where procedure_id=7	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0	0.016 sec
2	04:30:43	select * from procedure_list LIMIT 0, 1000	7 row(s) returned	0.000 sec / 0.000 sec
3	04:57:58	select * from audit_trail_medicine LIMIT 0, 1000	2 row(s) returned	0.000 sec / 0.000 sec

2. When doctor(employee) update their phone number in doctor table, the old value should get updated in audit history with timestamp.

The screenshot shows the MySQL Workbench interface with multiple tabs open. The 'Query 1' tab contains the SQL command: `select * from doctors where doctor_id = '1';`. The results grid shows one row of data:

doctor_id	first_name	last_name	gender	age	email_id	address	contact_number	specialization
1	Edward	Jenner	Male	36	ejenner@gmail.com	9551 UTD APT A, Charlotte, North Carolina, US...	(314) 555-8834	General Physician

The 'Output' pane below the results grid displays the execution log:

#	Time	Action	Message	Duration / Fetch
1	04:32:42	update procedure_list set procedure_name='Teeth Fillings and Repairs' where procedure_id=7	Error Code: 1142. UPDATE command denied to user 'doctor'@'localhost' for table 'procedure_list'	0.000 sec
2	04:32:55	select * from procedure_list LIMIT 0, 1000	7 row(s) returned	0.016 sec / 0.000 sec
3	04:43:50	update medicine set amount = '15' where medicine_id = '1001'	Error Code: 1142. UPDATE command denied to user 'doctor'@'localhost' for table 'medicine'	0.000 sec
4	04:44:05	select * from medicine LIMIT 0, 1000	12 row(s) returned	0.000 sec / 0.000 sec
5	05:06:04	select * from doctors where doctor_id = "1" LIMIT 0, 1000	1 row(s) returned	0.015 sec / 0.000 sec

Name : Darshan Puttanna

ID: 801169012

The screenshot shows the MySQL Workbench interface. In the top tab bar, multiple databases are listed: Local instance MySQL80 (universit...), doctor (university_student...), pharmacist, uscadmin, doctor (university_student...), and Local instance MySQL80 (universit...). The main area contains a query editor with the following SQL script:

```
2 where doctor_id = '1';
3
4 update doctors
5 set contact_number = '(311) 222-1234'
6 where doctor_id = '1';
7
8 select * from doctors
9 where doctor_id = '1';
```

Below the script is a Result Grid showing one row of data:

doctor_id	first_name	last_name	gender	age	email_id	address	contact_number	specialization
1	Edward	Jenner	Male	36	ejenner@gmail.com	9551 LTD APT A, Charlotte, North Carolina, US...	(311) 222-1234	General Physician

On the right side of the interface, there is a SQLAdditions panel with the message: "Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help." Below the result grid is an Action History titled "Action Output" showing the execution of the previous SQL statements:

#	Time	Action	Message	Duration / Fetch
1	04:32:42	update procedure_list set procedure_name='Teeth Filings and Repairs' where procedure_id=7	Error Code: 1142. UPDATE command denied to user 'doctor'@localhost for table 'procedure_list'	0.000 sec
2	04:32:55	select * from procedure_list LIMIT 0, 1000	7 row(s) returned	0.016 sec / 0.000 sec
3	04:43:50	update medicine set amount = 15 where medicine_id = '1001'	Error Code: 1142. UPDATE command denied to user 'doctor'@localhost for table 'medicine'	0.000 sec
4	04:44:05	select * from medicine LIMIT 0, 1000	12 row(s) returned	0.000 sec / 0.000 sec
5	05:06:04	select * from doctors where doctor_id = '1' LIMIT 0, 1000	1 row(s) returned	0.015 sec / 0.000 sec
6	05:09:06	update doctors set contact_number = '(311) 222-1234' where doctor_id = '1'	Error Code: 1146. Table 'university_student_clinic.audit_trail_employee' doesn't exist	0.000 sec
7	05:11:15	update doctors set contact_number = '(311) 222-1234' where doctor_id = '1'	Error Code: 1146. Table 'university_student_clinic.audit_trail_employee' doesn't exist	0.000 sec
8	05:28:26	update doctors set contact_number = '(311) 222-1234' where doctor_id = '1'	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0	0.016 sec
9	05:28:26	select * from doctors where doctor_id = '1' LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec

After doctor update table

The screenshot shows the MySQL Workbench interface. In the top tab bar, multiple databases are listed: Local instance MySQL80 (universit...), doctor (university_student...), pharmacist, uscadmin, doctor (university_student...), and Local instance MySQL80 (universit...). The main area contains a query editor with the following SQL script:

```
1 select * from audit_trail_doctor;
```

Below the script is a Result Grid showing one row of data:

id	old_data	new_data	timestamp	user
1	(314) 555-8834	(311) 222-1234	2021-05-12 05:28:26	doctor@localhost

On the right side of the interface, there is a SQLAdditions panel with the message: "Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help." Below the result grid is an Action History titled "Action Output" showing the execution of the previous SQL statements:

#	Time	Action	Message	Duration / Fetch
1	05:21:48	DROP TABLE IF EXISTS audit_trail_doctor	0 row(s) affected	0.032 sec
2	05:21:55	CREATE TABLE audit_trail_doctor (id int primary key AUTO_INCREMENT, old_data varchar(255), new_data varchar(255), timestamp datetime, user varchar(255))	Error Code: 1064. You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'audit_trail_doctor (id int primary key AUTO_INCREMENT, old_data varchar(255), new_d' at line 1	0.000 sec
3	05:23:11	CREATE TABLE audit_trail_doctor (id int primary key AUTO_INCREMENT, old_data varchar(255), new_data varchar(255), timestamp datetime, user varchar(255))	0 row(s) affected	0.032 sec
4	05:23:46	CREATE TRIGGER UPDATE_doctor BEFORE UPDATE ON doctors FOR EACH ROW SET new_data = (SELECT GROUP_CONCAT(old_data SEPARATOR ',') FROM audit_trail_doctor WHERE id < (SELECT id - 1 FROM audit_trail_doctor WHERE id = NEW.id))	Error Code: 1064. You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'UPDATE_doctor BEFORE UPDATE ON doctors FOR EACH ROW SET new_data = (SELECT GROU' at line 1	0.000 sec
5	05:25:30	CREATE TRIGGER UPDATE_doctor AFTER UPDATE ON doctors FOR EACH ROW SET new_data = (SELECT GROUP_CONCAT(old_data SEPARATOR ',') FROM audit_trail_doctor WHERE id < (SELECT id - 1 FROM audit_trail_doctor WHERE id = NEW.id))	Error Code: 1064. You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'UPDATE_doctor AFTER UPDATE ON doctors FOR EACH ROW SET new_data = (SELECT GROU' at line 1	0.000 sec
6	05:25:51	CREATE TRIGGER UPDATE_doctor AFTER UPDATE ON doctors FOR EACH ROW SET new_data = (SELECT GROUP_CONCAT(old_data SEPARATOR ',') FROM audit_trail_doctor WHERE id < (SELECT id - 1 FROM audit_trail_doctor WHERE id = NEW.id))	Error Code: 1419. You do not have the SUPER privilege and binary logging is enabled (you 'might' want to use the less safe log_bin_trust_function_creators variable)	0.000 sec
7	05:26:07	DROP TRIGGER IF EXISTS UPDATE_doctor	0 row(s) affected, 1 warning(s); 1360 Trigger does not exist	0.016 sec
8	05:26:17	CREATE TRIGGER UPDATE_doctor AFTER UPDATE ON doctors FOR EACH ROW SET new_data = (SELECT GROUP_CONCAT(old_data SEPARATOR ',') FROM audit_trail_doctor WHERE id < (SELECT id - 1 FROM audit_trail_doctor WHERE id = NEW.id))	Error Code: 1419. You do not have the SUPER privilege and binary logging is enabled (you 'might' want to use the less safe log_bin_trust_function_creators variable)	0.000 sec
9	05:29:55	select * from audit_trail_doctor LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec

Name : Darshan Puttanna
ID: 801169012

Insert data in audit table when user perform insert action on data.

1. When patient book an appointment, audit table should get updated with their appointment_id and timestamp at which they are booking the appointment.

The screenshot shows the MySQL Workbench interface with two tabs: 'SQL File 1' and 'SQL File 2'. In 'SQL File 1', the following SQL code is run:

```
1
2
3 • insert into appointments values (12,10,'Joe','Bill','2021-04-05 12:30:00',2,'Root Canal');
4 • select * from appointments;
```

The 'Result Grid' shows the data inserted into the 'appointments' table:

appointment_id	patient_id	patient_name	patient_lname	appointment_time	doctor_id	patient_description
9	9	Virat	Kohli	2021-04-04 12:30:00	6	Pain in Arm
10	10	Joe	Bill	2021-04-04 12:30:00	2	Teeth filling
11	10	Joe	Bill	2021-04-05 12:30:00	2	Teeth filling
12	10	Joe	Bill	2021-04-05 12:30:00	2	Root Canal

The 'Output' pane shows the execution log:

#	Time	Action	Message	Duration / Fetch
1	06:17:24	insert into appointments values (12,10,'Joe','Bill','2021-04-05 12:30:00',2,'Root ...	1 row(s) affected	0.015 sec
2	06:17:43	select * from appointments LIMIT 0, 1000	12 row(s) returned	0.000 sec / 0.000 sec

After inserting the data when we ran audit_trial_appointments the value is getting updated.

The screenshot shows the MySQL Workbench interface with two tabs: 'SQL File 1' and 'SQL File 2'. In 'SQL File 1', the same SQL code as before is run:

```
1
2
3 • insert into appointments values (12,10,'Joe','Bill','2021-04-05 12:30:00',2,'Root Canal');
4 • select * from appointments;
```

The 'Result Grid' shows the data inserted into the 'audit_trail_appointments' table:

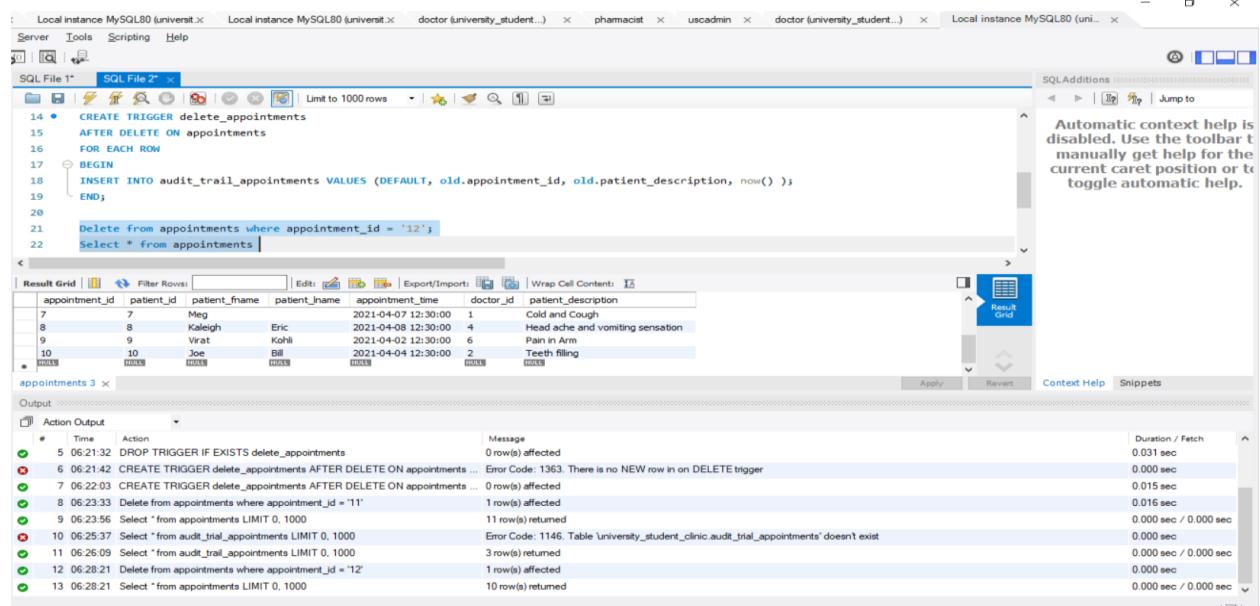
id	appointment_id	patient_description	timestamp
1	11	Teeth filling	2021-05-12 06:15:58
2	12	Root Canal	2021-05-12 06:17:24

The 'Output' pane shows the execution log:

#	Time	Action	Message	Duration / Fetch
1	06:17:24	insert into appointments values (12,10,'Joe','Bill','2021-04-05 12:30:00',2,'Root ...	1 row(s) affected	0.015 sec
2	06:17:43	select * from appointments LIMIT 0, 1000	12 row(s) returned	0.000 sec / 0.000 sec
3	06:19:02	select * from audit_trail_appointments LIMIT 0, 1000	2 row(s) returned	0.000 sec / 0.000 sec

Insert data in audit table when user perform delete action on data.

- When particular appointments is deleted from the appointments table, that data should get updated in the audit table with timestamp.



The screenshot shows the MySQL Workbench interface with multiple tabs open. In the SQL tab, a trigger named 'delete_appointments' is created:

```

14 • CREATE TRIGGER delete_appointments
15   AFTER DELETE ON appointments
16   FOR EACH ROW
17   BEGIN
18     INSERT INTO audit_trail_appointments VALUES (DEFAULT, old.appointment_id, old.patient_description, now());
19   END;
20
21 Delete from appointments where appointment_id = '12';
22 Select * from appointments;

```

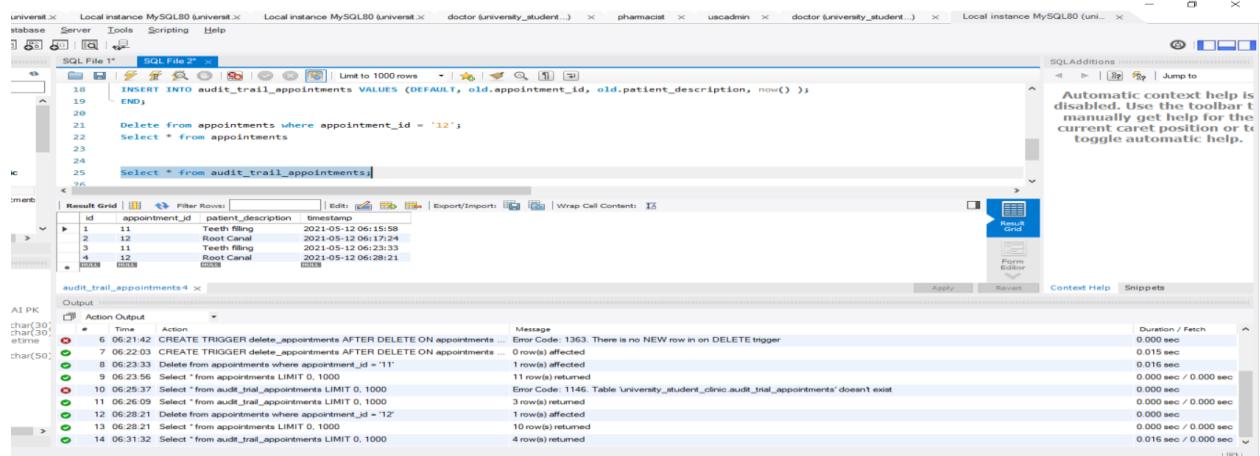
In the Result Grid tab, the 'appointments' table is shown:

appointment_id	patient_id	patient_fname	patient_lname	appointment_time	doctor_id	patient_description
7	7	Meg		2021-04-07 12:30:00	1	Cold and Cough
8	8	Kaleigh	Eric	2021-04-08 12:30:00	4	Head ache and vomiting sensation
9	9	Virat	Kohli	2021-04-02 12:30:00	6	Pain in Arm
10	10	Joe	Bill	2021-04-04 12:30:00	2	Teeth filling

In the Action Output tab, the history of operations is listed:

- 5 06:21:32 DROP TRIGGER IF EXISTS delete_appointments
- 6 06:21:42 CREATE TRIGGER delete_appointments AFTER DELETE ON appointments ... Error Code: 1363. There is no NEW row in on DELETE trigger
- 7 06:22:03 CREATE TRIGGER delete_appointments AFTER DELETE ON appointments ... 0 row(s) affected
- 8 06:23:33 Delete from appointments where appointment_id = '11' 1 row(s) affected
- 9 06:23:56 Select * from appointments LIMIT 0, 1000 11 row(s) returned
- 10 06:25:37 Select * from audit_trail_appointments LIMIT 0, 1000 Error Code: 1146. Table 'university_student_clinic.audit_trail_appointments' doesn't exist
- 11 06:26:09 Select * from audit_trail_appointments LIMIT 0, 1000 3 row(s) returned
- 12 06:28:21 Delete from appointments where appointment_id = '12' 1 row(s) affected
- 13 06:28:21 Select * from appointments LIMIT 0, 1000 10 row(s) returned

When we ran audit_trail_appointments after deleting a row from appointments table. The deleted row is getting updated in audit_trail_appointments table, because of the delete trigger we have written.



The screenshot shows the MySQL Workbench interface with multiple tabs open. In the SQL tab, the same trigger 'delete_appointments' is created again, but this time it works correctly:

```

18 • CREATE TRIGGER delete_appointments
19   AFTER DELETE ON appointments
20   FOR EACH ROW
21   BEGIN
22     INSERT INTO audit_trail_appointments VALUES (DEFAULT, old.appointment_id, old.patient_description, now());
23   END;
24
25 Delete from appointments where appointment_id = '12';
26 Select * from audit_trail_appointments;

```

In the Result Grid tab, the 'audit_trail_appointments' table is shown:

id	appointment_id	patient_description	timestamp
1	11	Teeth filling	2021-05-12 06:15:58
2	12	Root Canal	2021-05-12 06:23:33
3	11	Teeth filling	2021-05-12 06:23:33
4	12	Root Canal	2021-05-12 06:28:21

In the Action Output tab, the history of operations is listed:

- 6 06:21:42 CREATE TRIGGER delete_appointments AFTER DELETE ON appointments ... Error Code: 1363. There is no NEW row in on DELETE trigger
- 7 06:22:03 CREATE TRIGGER delete_appointments AFTER DELETE ON appointments ... 0 row(s) affected
- 8 06:23:33 Delete from appointments where appointment_id = '11' 1 row(s) affected
- 9 06:23:56 Select * from appointments LIMIT 0, 1000 11 row(s) returned
- 10 06:25:37 Select * from audit_trail_appointments LIMIT 0, 1000 Error Code: 1146. Table 'university_student_clinic.audit_trail_appointments' doesn't exist
- 11 06:26:09 Select * from audit_trail_appointments LIMIT 0, 1000 3 row(s) returned
- 12 06:28:21 Delete from appointments where appointment_id = '12' 1 row(s) affected
- 13 06:28:21 Select * from appointments LIMIT 0, 1000 10 row(s) returned
- 14 06:31:32 Select * from audit_trail_appointments LIMIT 0, 1000 4 row(s) returned

Triggers:

A trigger is a set of actions that are run automatically when a specified change operation is performed on a specified table. Triggers are useful for audit trail.

Types of trigger used are: After insert, before insert, after update, before update, before delete.

Requirement set 7:

1. Write trigger to add record in an audit table after medicine amount is getting updated in medicine table.

```
DELIMITER //
DROP TRIGGER IF EXISTS UPDATE_MEDICINE_AMOUNT;
DELIMITER //
CREATE TRIGGER UPDATE_MEDICINE_AMOUNT
AFTER UPDATE ON MEDICENE
FOR EACH ROW
BEGIN
INSERT INTO audit_trail_medicine VALUES (DEFAULT, NEW.medicene_name, NEW.amount, now());
END;
```

2. Write trigger to add record in an audit table before employee phone number is updated in employee table.

```
DELIMITER //
DROP TRIGGER IF EXISTS UPDATE_doctor;
DELIMITER //
CREATE TRIGGER UPDATE_doctor
BEFORE UPDATE ON doctors
FOR EACH ROW
BEGIN
INSERT INTO audit_trail_doctor VALUES (DEFAULT, OLD.contact_number, NEW.contact_number, "doctor", now());
END;
```

3. Write trigger to insert data in audit table after patient had booked an appointment.

Name : Darshan Puttanna
ID: 801169012

```
-- Trigger 3 and audit-insert

DELIMITER //
DROP TRIGGER IF EXISTS insert_appointments;
DELIMITER //
CREATE TRIGGER insert_appointments
AFTER INSERT ON appointments
FOR EACH ROW
BEGIN
INSERT INTO audit_trail_appointments VALUES (DEFAULT, NEW.appointment_id, NEW.patient_description, now() );
END;
```

4. Write trigger to insert data in visit_information table after checkup table is updated.

```
DELIMITER //
DROP TRIGGER IF EXISTS insert_visitinformation;
DELIMITER //
CREATE TRIGGER insert_visitinformation
AFTER INSERT ON checkup
FOR EACH ROW
BEGIN
INSERT INTO visit_information VALUES (DEFAULT, NEW.visitor_id, New.symptoms_desc, NEW.result, now() );
END;
```

5. Write trigger to insert data in audit_trail_nilling table after billing_info table is updated.

```
--Trigger 5: Billing details

DELIMITER //
DROP TRIGGER IF EXISTS insert_billing_details;
DELIMITER //
CREATE TRIGGER insert_billing_details
AFTER INSERT ON billing_info
FOR EACH ROW
BEGIN
INSERT INTO audit_trail_billing VALUES (DEFAULT, NEW.visitor_id, New.total_bill);
END;
```

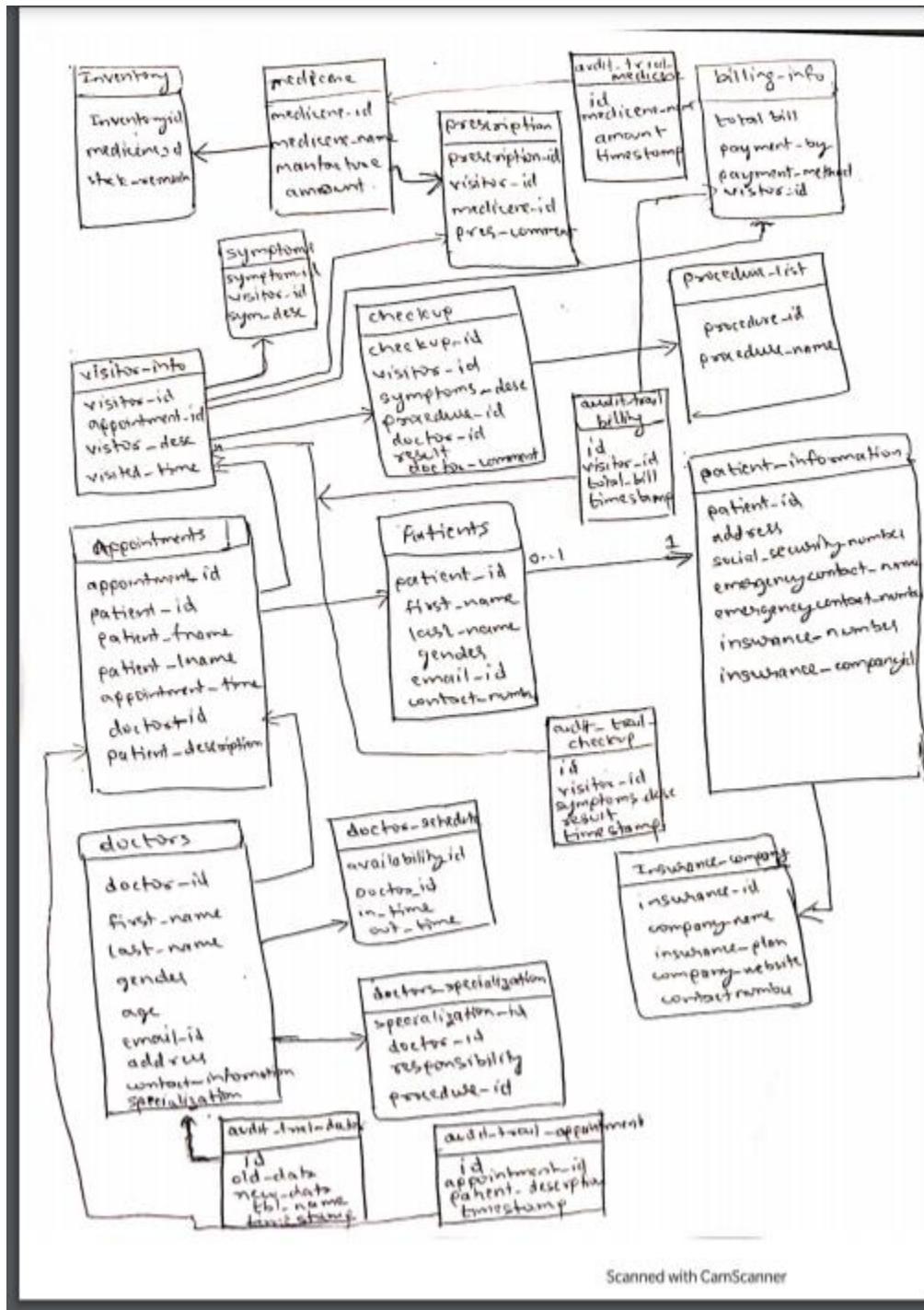
Name : Darshan Puttanna
ID: 801169012

6. Write trigger to insert data in audit table after patient had deleted an appointment.

```
-- Trigger 6 and audit-delete

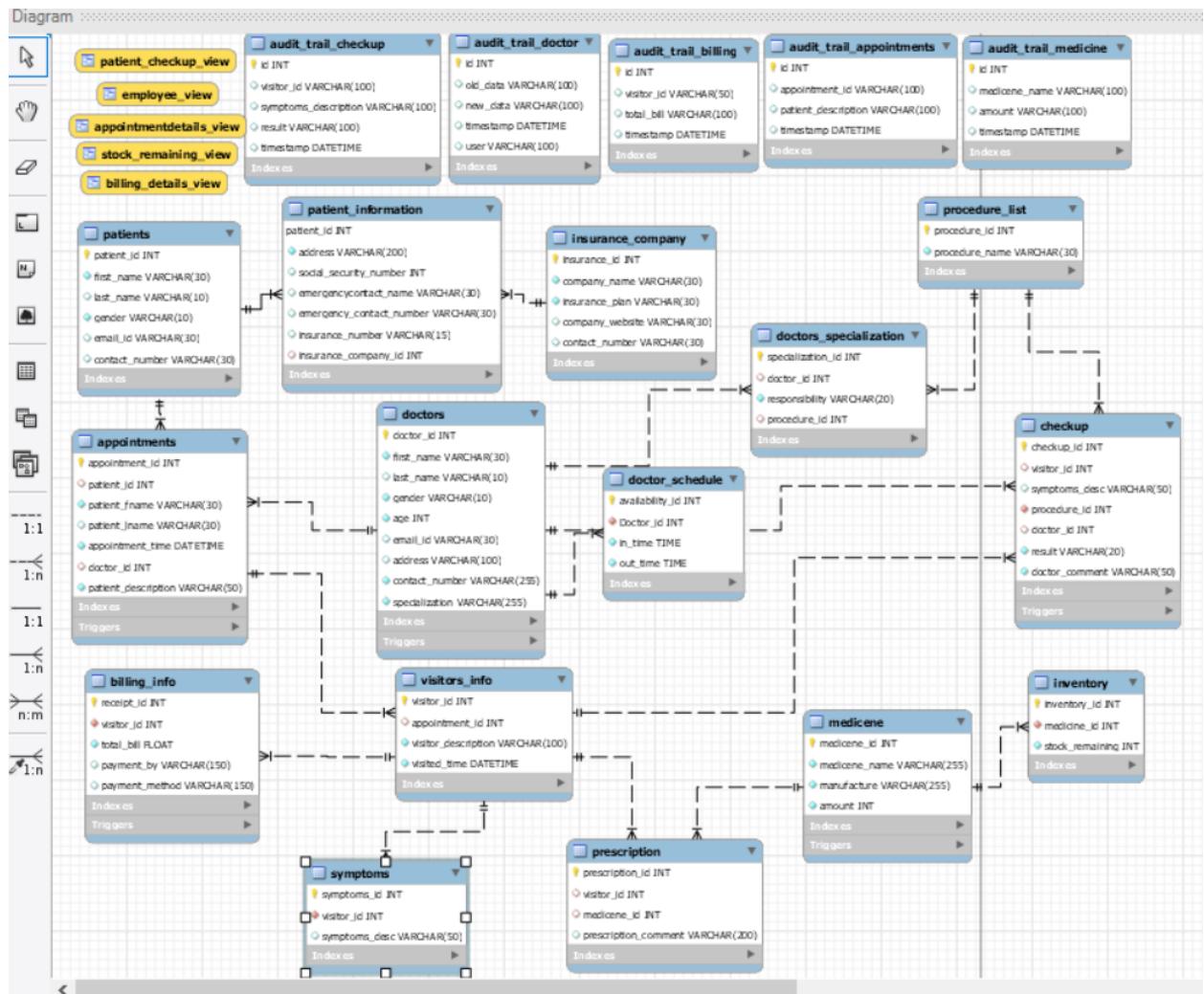
DELIMITER //
DROP TRIGGER IF EXISTS delete_appointments;
DELIMITER //
CREATE TRIGGER delete_appointments
AFTER DELETE ON appointments
FOR EACH ROW
BEGIN
INSERT INTO audit_trail_appointments VALUES (DEFAULT, NEW.appointment_id, NEW.patient_description, now() );
END;
```

UML Data Model



Name : Darshan Puttanna
ID: 801169012

ER Diagram



Name : Darshan Puttanna
ID: 801169012

Thank You