DIAGNOSTIC LABARATORY MANAGEMENT SYSTEM

A PROJECT REPORT

Submitted by

RUFAS . K [RA2211026010557]

KEERTHI PAVAN . S[RA2211026010561]

Under the Guidance of

Dr.M.KANIPRIYA

Assistant professor, Department of Computional intelligence

in partial fulfillment of the requirements for the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

DEPARTMENT OF COMPUTIONAL INTELLIGENCE

COLLEGE OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR– 603 203

MAY 2024

SRM

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR–603 203

BONAFIDE CERTIFICATE

Register no._RA2211026010557 and RA2211026010561 Certified to be the bonafide work done by **RUFAS.K and KEERTHI PAVAN.S** of II year/IV sem B.Tech Degree Course in the Project Course – **21CSC205P Database Management Systems** in **SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**, Kattankulathur for the academic year 2023-2024.

Date: 06 MAY 2024

[illegible] 06/05/24

Faculty in Charge
Dr.KANIPRIYA M
Assistant professor
Department of CINTEL
SRMSIT -KTR

[illegible]

HEAD OF THE DEPARTMENT
Dr. R.Annie Uthra
Professor & Head
Department of CINTEL
SRMIST - KTR

ABSTRACT

In today's rapidly evolving scientific landscape, efficient management of laboratory resources is paramount to ensure smooth operations and optimal utilization of resources. A Laboratory Management Database (LMD) serves as a comprehensive tool to streamline various aspects of laboratory operations, ranging from sample tracking to inventory management and result analysis. This abstract delves into the core functionalities and benefits of an LMD system.

At its core, an LMD system offers robust sample tracking capabilities, allowing laboratories to efficiently monitor the movement of samples from receipt to analysis and disposal. By integrating barcode or RFID technology, laboratories can automate sample identification and tracking, reducing errors and enhancing traceability.

Furthermore, an LMD system facilitates seamless inventory management by providing real-time visibility into stock levels, expiration dates, and reordering thresholds. Through automated alerts and notifications, laboratories can ensure timely replenishment of consumables and prevent workflow interruptions due to stockouts.

The analytical capabilities of an LMD system extend beyond sample tracking and inventory management. Advanced data analysis tools enable laboratories to derive meaningful insights from experimental results, facilitating data-driven decision-making and accelerating scientific discoveries. Moreover, integration with laboratory instrumentation allows for automated data capture and analysis, minimizing manual intervention and increasing throughput.

Collaboration is another key aspect facilitated by an LMD system. With features such as shared access and collaborative workflows, researchers can seamlessly collaborate on projects, share data, and track progress in real-time. This fosters interdisciplinary collaboration and accelerates the pace of scientific innovation.

Security and compliance are paramount considerations in laboratory management. An LMD system incorporates robust security measures, such as role-based access control and audit trails, to ensure data integrity and regulatory complian

TABLE OF CONTENTS

CHAPTER 1

PROBLEM STATEMENT:

In contemporary laboratory settings, the management of data, samples, and resources is often fragmented, leading to inefficiencies, errors, and delays in research and analysis processes. Existing systems may rely on manual data entry, paper-based documentation, and disjointed software applications, hindering collaboration, data integrity, and regulatory compliance. Therefore, there is a critical need for a comprehensive Laboratory Database Management System (LDMS) that addresses these challenges and provides a unified platform for streamlining laboratory operations.

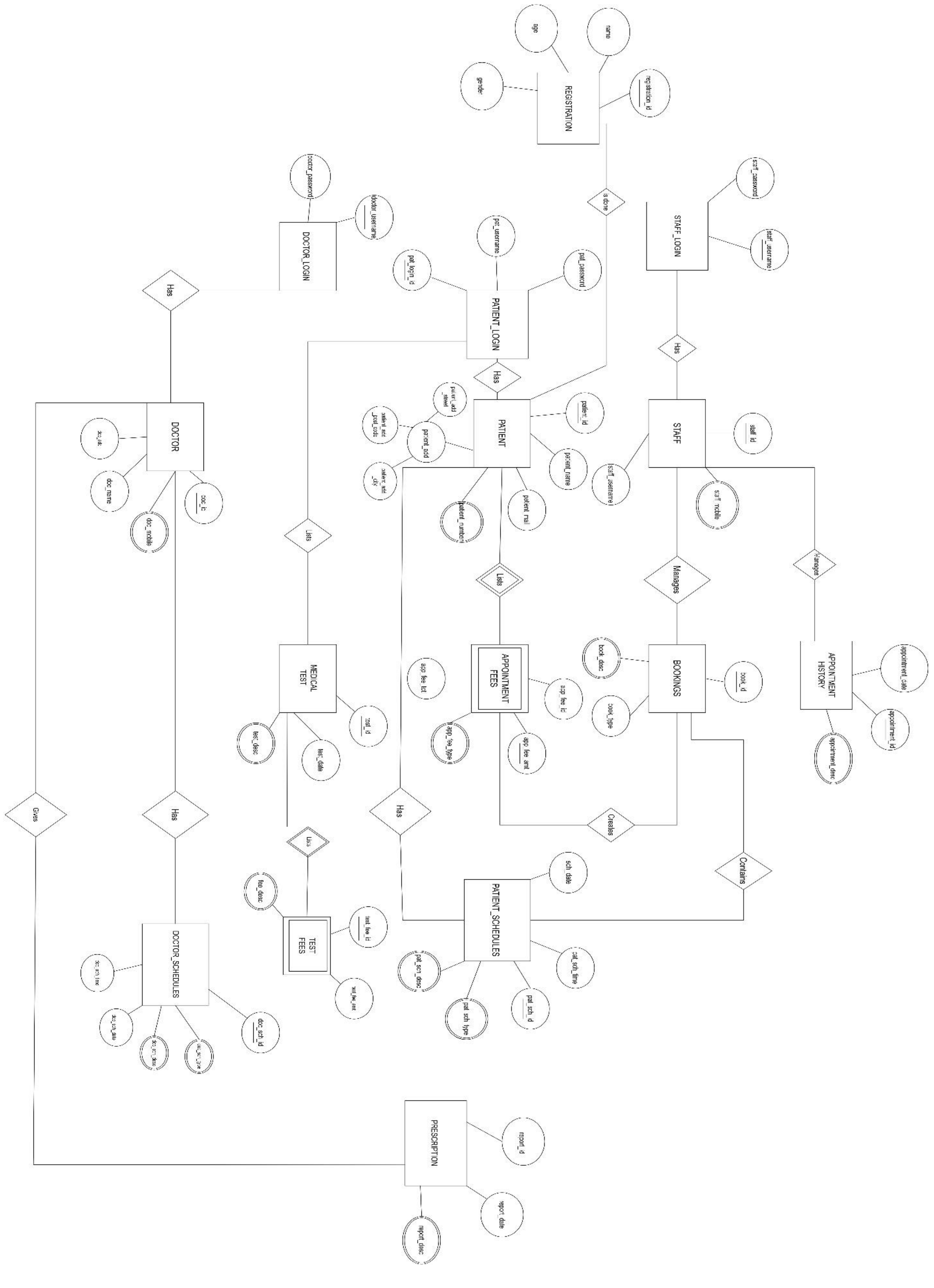
1. *Fragmented Data Management*: Laboratories often utilize disparate systems for sample tracking, inventory management, and result analysis, leading to data silos and inefficiencies. Integration challenges between these systems hinder real-time data sharing and analysis, impeding decision-making and collaboration.

2. *Manual Processes and Paper-based Documentation*: Many laboratories still rely on manual data entry and paper-based documentation, which are error-prone, time-consuming, and susceptible to loss or damage. This manual approach not only slows down workflows but also compromises data integrity and traceability.

3. *Limited Scalability and Flexibility*: Existing laboratory management systems may lack scalability and flexibility to adapt to evolving research needs and technological advancements. As laboratories expand or introduce new instrumentation and techniques, these systems struggle to accommodate changing requirements, leading to inefficiencies and user dissatisfaction.

4. *Compliance and Regulatory Challenges*: Laboratories are subject to stringent regulatory requirements, including data security, traceability, and quality assurance. Ensuring compliance with regulations such as Good Laboratory Practices (GLP) and Good Manufacturing Practices (GMP) poses significant challenges, particularly with disparate systems and manual processes.

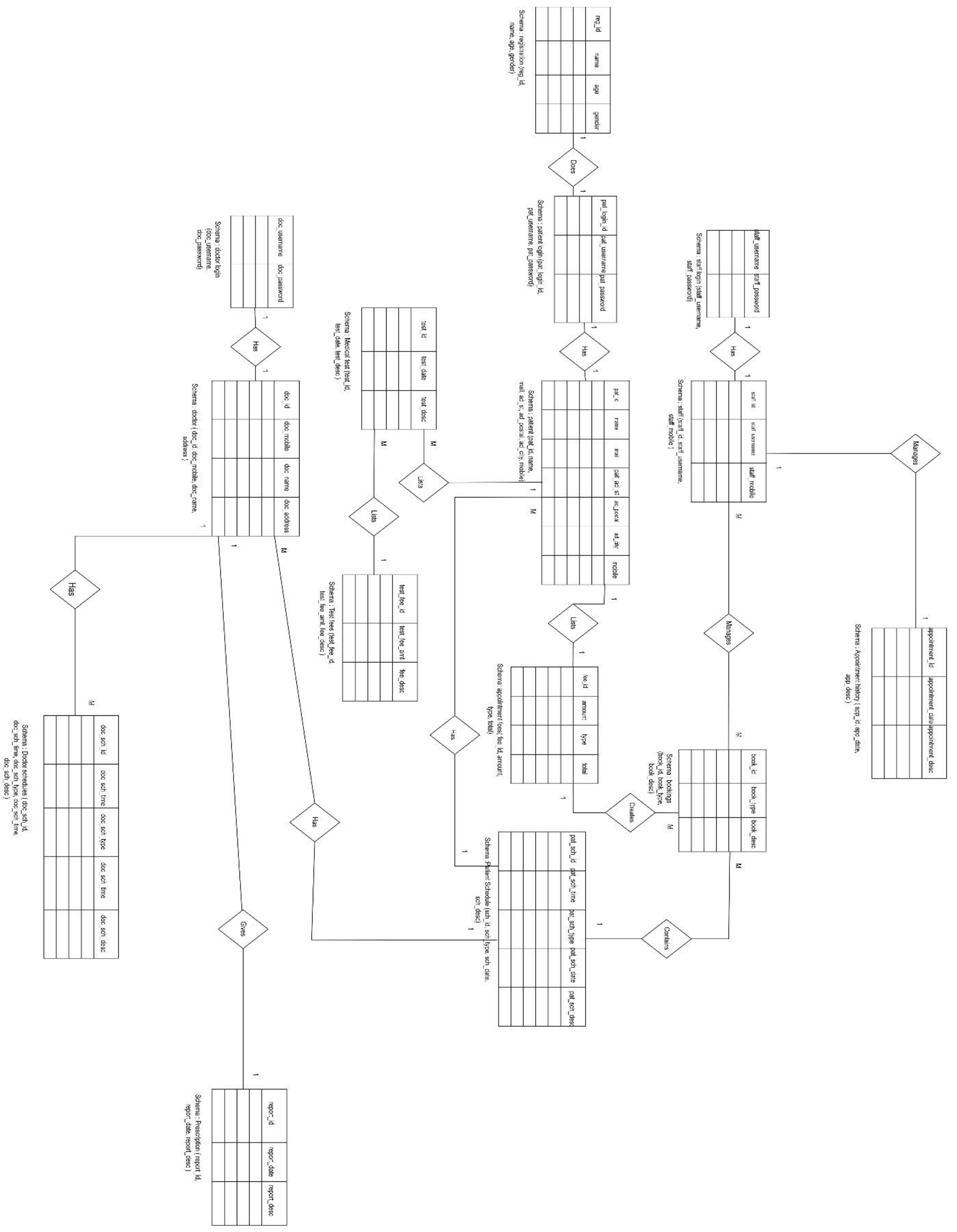
ER DIAGRAM:



REGISTRATION
age
name
gender
registration_id
is done
STAFF_LOGIN
staff_password
staff_username
DOCTOR_LOGIN
doctor_password
doctor_username
PATIENT_LOGIN
pat_username
pat_login_id
pat_password
Has
DOCTOR
doc_name
doc_id
doc_mobile
PATIENT
patient_id
patient_name
patient_add
patient_number
patient_mail
STAFF
staff_id
staff_username
staff_mobile
Lists
Manages
MEDICAL TEST
test_id
test_desc
test_date
APPOINTMENT FEES
app_fee_id
app_fee_type
app_fee_amt
BOOKINGS
book_id
book_desc
book_type
APPOINTMENT HISTORY
appointment_date
appointment_id
appointment_desc
Gives
Creates
Contains
TEST FEES
fee_desc
test_fee_id
test_fee_amt
PATIENT_SCHEDULES
sch_date
pat_sch_time
pat_sch_id
pat_sch_type
pat_sch_desc
DOCTOR_SCHEDULES
doc_sch_id
doc_sch_date
doc_sch_time
doc_sch_desc
PRESCRIPTION
report_id
report_date
report_desc

CHAPTER 2

RELATIONAL SCHEMA DIAGRAM:



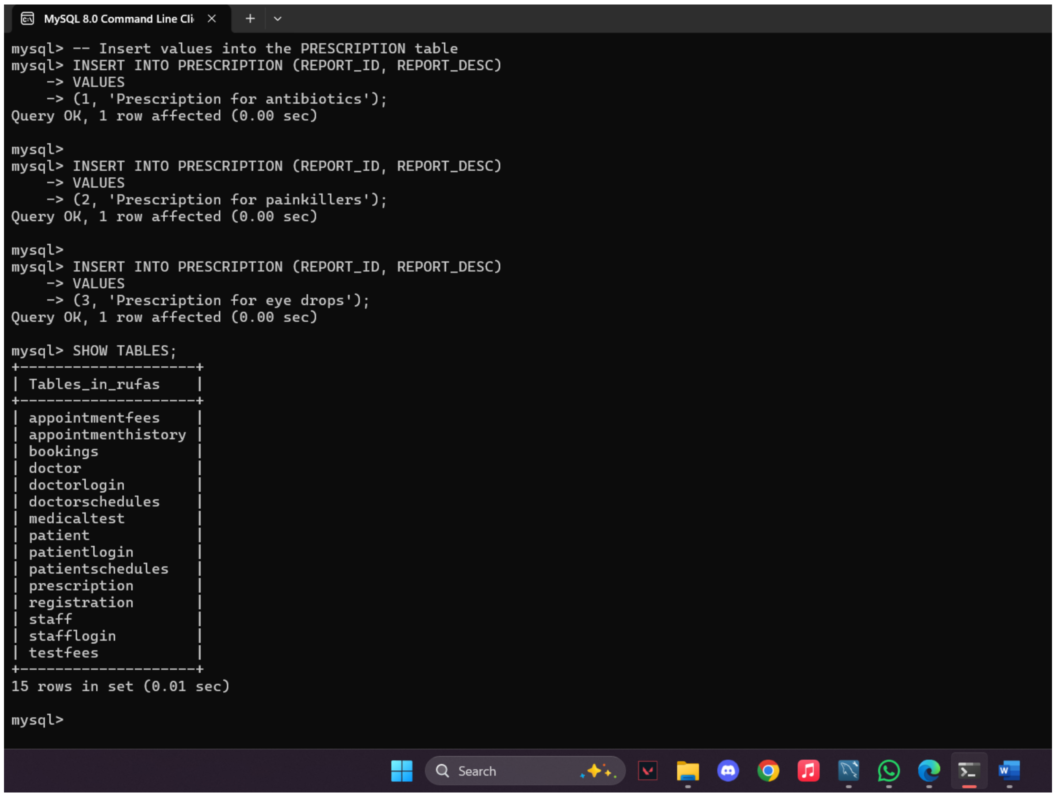
reg_id
name
age
gender
Schema : registration (reg_id, name, age, gender)
Does
pat_login_id
pat_username
pat_password
Schema : patient login (pat_login_id, pat_username, pat_password)
staff_username
staff_password
Schema : staff login (staff_username, staff_password)
doc_username
doc_password
Schema : doctor login (doc_username, doc_password)
Has
test_id
test_date
test_desc
Schema : Medical test (test_id, test_date, test_desc)
doc_id
doc_mobile
doc_name
doc_address
Schema : doctor (doc_id, doc_mobile, doc_name, address)
pat_id
name
mail
pat_ad_st
ad_postal
ad_city
mobile
Schema : patient (pat_id, name, mail, ad_st, ad_postal, ad_city, mobile)
staff_id
staff_username
staff_mobile
Schema : staff (staff_id, staff_username, staff_mobile)
Manages
Lists
test_fee_id
test_fee_amt
fee_desc
Schema : Test fees (test_fee_id, test_fee_amt, fee_desc)
fee_id
amount
type
total
Schema : appointment fees(fee_id, amount, type, total)
appointment_id
appointment_date
appointment_desc
Schema : Appointment history (app_id, app_date, app_desc)
doc_sch_id
doc_sch_time
doc_sch_type
doc_sch_time
doc_sch_desc
Schema : Doctor schedules (doc_sch_id, doc_sch_time, doc_sch_type, doc_sch_time, doc_sch_desc)
book_id
book_type
book_desc
Schema : bookings (book_id, book_type, book_desc)
Creates
pat_sch_id
pat_sch_time
pat_sch_type
pat_sch_date
pat_sch_desc
Schema : Patient Schedule (sch_id, sch_type, sch_date, sch_desc)
Contains
Gives
repor_id
repor_date
report_desc
Schema : Prescription (report_id, report_date, report_desc)
1
M

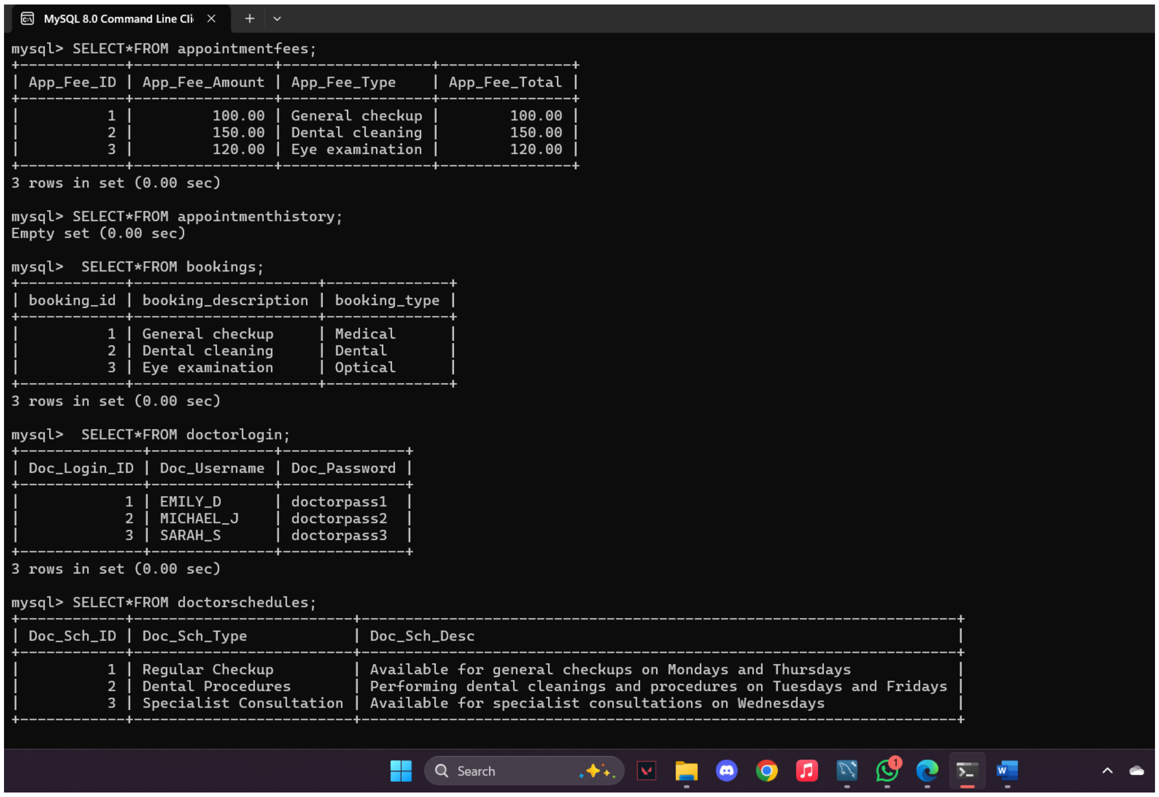
CREATION OF DATABASE TABLES:

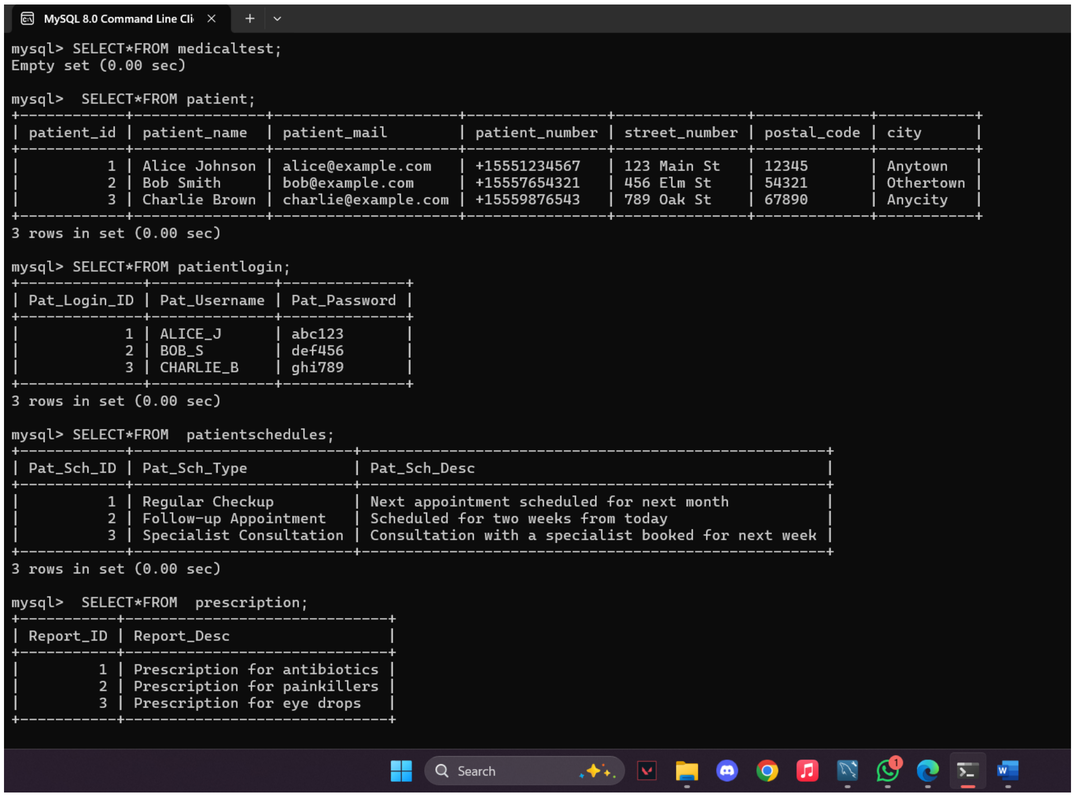
```
CREATE TABLE staff (
    staff_id INT PRIMARY KEY,
    staff_mobile VARCHAR(15) UNIQUE,
    staff_username VARCHAR(50) UNIQUE
);
CREATE TABLE patient (
    patient_id INT PRIMARY KEY,
    patient_name VARCHAR(100),
    patient_mail VARCHAR(100) UNIQUE,
    patient_number VARCHAR(15),
    street_number VARCHAR(20),
    postal_code VARCHAR(10),
    city VARCHAR(50)
);
CREATE TABLE doctor (
    doc_id INT PRIMARY KEY,
    doctor_mobile VARCHAR(15) UNIQUE,
    doctor_name VARCHAR(100),
    doctor_address VARCHAR(255)
);
CREATE TABLE bookings (
    booking_id INT PRIMARY KEY,
    booking_description VARCHAR(255),
    booking_type VARCHAR(50));
CREATE TABLE Registration (
    RegistrationID INT PRIMARY KEY,
    Name VARCHAR(255) NOT NULL,
    Age INT,
    Gender VARCHAR(10)
);
CREATE TABLE StaffLogin (
    StaffUsername VARCHAR(50) PRIMARY KEY,
    StaffPassword VARCHAR(255) NOT NULL
);
CREATE TABLE PatientLogin (
    Pat_Login_ID INT PRIMARY KEY,
    Pat_Username VARCHAR(50) NOT NULL,
    Pat_Password VARCHAR(255) NOT NULL
);
CREATE TABLE DoctorLogin (
    Doc_Login_ID INT PRIMARY KEY,
    Doc_Username VARCHAR(50) NOT NULL,
    Doc_Password VARCHAR(255) NOT NULL
);
CREATE TABLE AppointmentHistory (
    Appointment_ID INT PRIMARY KEY,
    Appointment_Date DATE NOT NULL,
    Appointment_Desc VARCHAR(255)
);
```

```
CREATE TABLE AppointmentFees (
    App_Fee_ID INT PRIMARY KEY,
    App_Fee_Amount DECIMAL(10, 2) NOT NULL,
    App_Fee_Type VARCHAR(50) NOT NULL,
    App_Fee_Total DECIMAL(10, 2) NOT NULL);
CREATE TABLE MedicalTest (
    Test_ID INT PRIMARY KEY,
    Test_Date DATE NOT NULL,
    Test_Desc VARCHAR(255) NOT NULL);
CREATE TABLE TestFees (
    Test_Fee_ID INT PRIMARY KEY,
    Test_Fee_Amount DECIMAL(10, 2) NOT NULL,
    Test_Fee_Type VARCHAR(50) NOT NULL,
    Test_Fee_Total DECIMAL(10, 2) NOT NULL);
CREATE TABLE PatientSchedules (
    Pat_Sch_ID INT PRIMARY KEY,
    Pat_Sch_Type VARCHAR(50),
    Pat_Sch_Desc VARCHAR(255));
CREATE TABLE DoctorSchedules (
    Doc_Sch_ID INT PRIMARY KEY,
    Doc_Sch_Type VARCHAR(50),
    Doc_Sch_Desc VARCHAR(255));
CREATE TABLE Prescription (
    Report_ID INT PRIMARY KEY,
    Report_Desc VARCHAR(255)
```

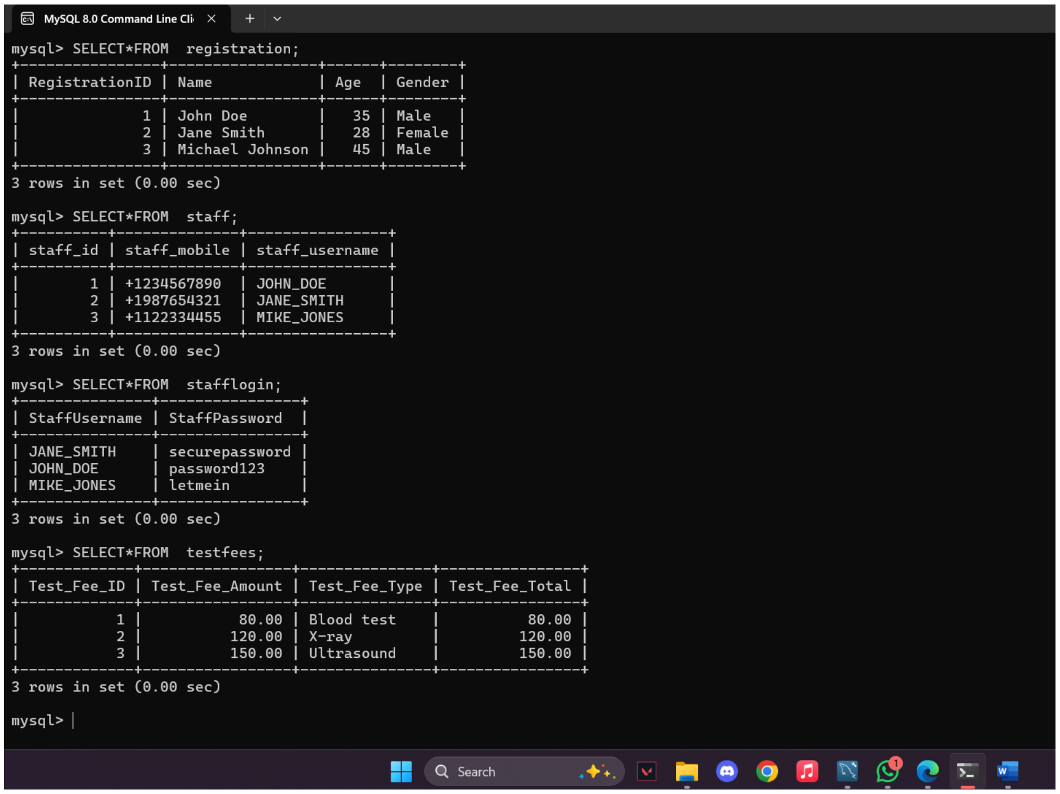
);


```
mysql> -- Insert values into the PRESCRIPTION table
mysql> INSERT INTO PRESCRIPTION (REPORT_ID, REPORT_DESC)
    -> VALUES
    -> (1, 'Prescription for antibiotics');
Query OK, 1 row affected (0.00 sec)

mysql>
mysql> INSERT INTO PRESCRIPTION (REPORT_ID, REPORT_DESC)
    -> VALUES
    -> (2, 'Prescription for painkillers');
Query OK, 1 row affected (0.00 sec)

mysql>
mysql> INSERT INTO PRESCRIPTION (REPORT_ID, REPORT_DESC)
    -> VALUES
    -> (3, 'Prescription for eye drops');
Query OK, 1 row affected (0.00 sec)

mysql> SHOW TABLES;
+---------------------+
| Tables_in_rufas     |
+---------------------+
| appointmentfees     |
| appointmenthistory  |
| bookings            |
| doctor              |
| doctorlogin         |
| doctorschedules     |
| medicaltest         |
| patient             |
| patientlogin        |
| patientschedules    |
| prescription        |
| registration        |
| staff               |
| stafflogin          |
| testfees            |
+---------------------+
15 rows in set (0.01 sec)

mysql>
```


```
mysql> SELECT*FROM appointmentfees;
+------------+----------------+-----------------+---------------+
| App_Fee_ID | App_Fee_Amount | App_Fee_Type    | App_Fee_Total |
+------------+----------------+-----------------+---------------+
|          1 |         100.00 | General checkup |        100.00 |
|          2 |         150.00 | Dental cleaning |        150.00 |
|          3 |         120.00 | Eye examination |        120.00 |
+------------+----------------+-----------------+---------------+
3 rows in set (0.00 sec)

mysql> SELECT*FROM appointmenthistory;
Empty set (0.00 sec)

mysql>  SELECT*FROM bookings;
+------------+---------------------+--------------+
| booking_id | booking_description | booking_type |
+------------+---------------------+--------------+
|          1 | General checkup     | Medical      |
|          2 | Dental cleaning     | Dental       |
|          3 | Eye examination     | Optical      |
+------------+---------------------+--------------+
3 rows in set (0.00 sec)

mysql>  SELECT*FROM doctorlogin;
+--------------+--------------+--------------+
| Doc_Login_ID | Doc_Username | Doc_Password |
+--------------+--------------+--------------+
|            1 | EMILY_D      | doctorpass1  |
|            2 | MICHAEL_J    | doctorpass2  |
|            3 | SARAH_S      | doctorpass3  |
+--------------+--------------+--------------+
3 rows in set (0.00 sec)

mysql> SELECT*FROM doctorschedules;
+------------+-------------------------+---------------------------------------------------------------------+
| Doc_Sch_ID | Doc_Sch_Type            | Doc_Sch_Desc                                                        |
+------------+-------------------------+---------------------------------------------------------------------+
|          1 | Regular Checkup         | Available for general checkups on Mondays and Thursdays             |
|          2 | Dental Procedures       | Performing dental cleanings and procedures on Tuesdays and Fridays  |
|          3 | Specialist Consultation | Available for specialist consultations on Wednesdays                |
+------------+-------------------------+---------------------------------------------------------------------+
```



```
mysql> SELECT*FROM medicaltest;
Empty set (0.00 sec)

mysql>  SELECT*FROM patient;
+------------+---------------+---------------------+--------------+---------------+-------------+-----------+
| patient_id | patient_name  | patient_mail        | patient_number | street_number | postal_code | city      |
+------------+---------------+---------------------+--------------+---------------+-------------+-----------+
|          1 | Alice Johnson | alice@example.com   | +15551234567 | 123 Main St   | 12345       | Anytown   |
|          2 | Bob Smith     | bob@example.com     | +15557654321 | 456 Elm St    | 54321       | Othertown |
|          3 | Charlie Brown | charlie@example.com | +15559876543 | 789 Oak St    | 67890       | Anycity   |
+------------+---------------+---------------------+--------------+---------------+-------------+-----------+
3 rows in set (0.00 sec)

mysql> SELECT*FROM patientlogin;
+--------------+--------------+--------------+
| Pat_Login_ID | Pat_Username | Pat_Password |
+--------------+--------------+--------------+
|            1 | ALICE_J      | abc123       |
|            2 | BOB_S        | def456       |
|            3 | CHARLIE_B    | ghi789       |
+--------------+--------------+--------------+
3 rows in set (0.00 sec)

mysql> SELECT*FROM  patientschedules;
+------------+------------------------+-----------------------------------------------+
| Pat_Sch_ID | Pat_Sch_Type           | Pat_Sch_Desc                                  |
+------------+------------------------+-----------------------------------------------+
|          1 | Regular Checkup        | Next appointment scheduled for next month     |
|          2 | Follow-up Appointment  | Scheduled for two weeks from today            |
|          3 | Specialist Consultation | Consultation with a specialist booked for next week |
+------------+------------------------+-----------------------------------------------+
3 rows in set (0.00 sec)

mysql>  SELECT*FROM  prescription;
+-----------+------------------------------+
| Report_ID | Report_Desc                  |
+-----------+------------------------------+
|         1 | Prescription for antibiotics |
|         2 | Prescription for painkillers |
|         3 | Prescription for eye drops   |
+-----------+------------------------------+
```

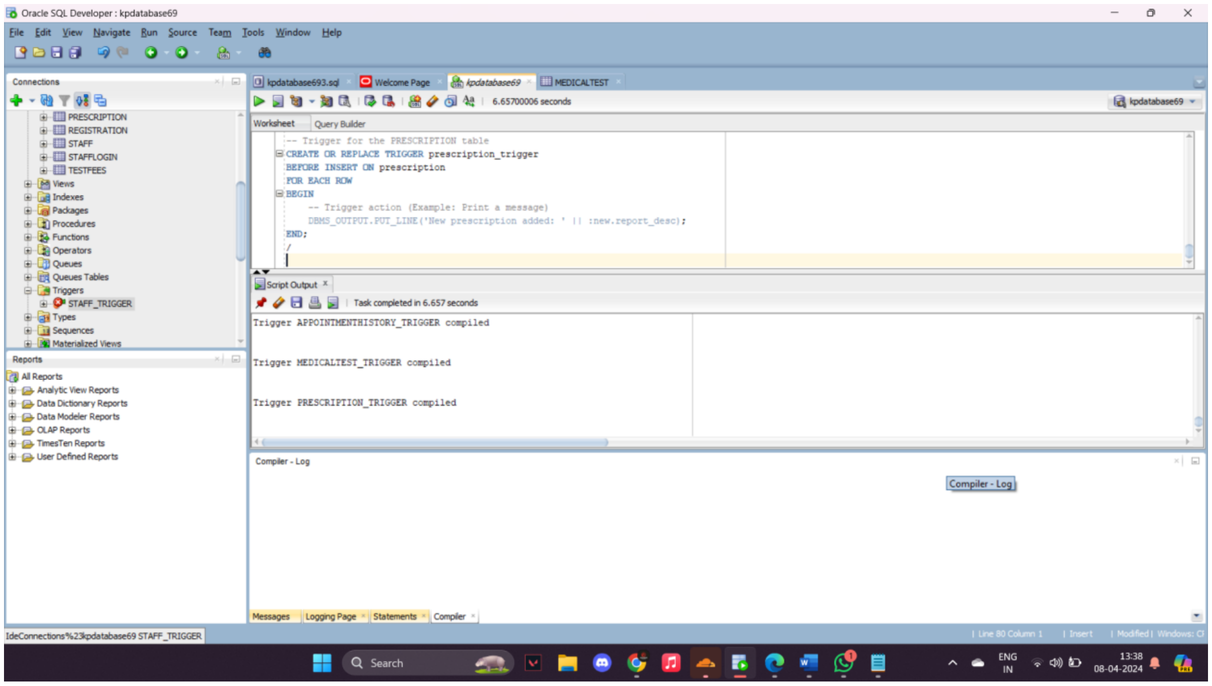


```
mysql> SELECT*FROM  registration;
+----------------+-----------------+------+--------+
| RegistrationID | Name            | Age  | Gender |
+----------------+-----------------+------+--------+
|              1 | John Doe        |   35 | Male   |
|              2 | Jane Smith      |   28 | Female |
|              3 | Michael Johnson |   45 | Male   |
+----------------+-----------------+------+--------+
3 rows in set (0.00 sec)

mysql> SELECT*FROM  staff;
+----------+--------------+----------------+
| staff_id | staff_mobile | staff_username |
+----------+--------------+----------------+
|        1 | +1234567890  | JOHN_DOE       |
|        2 | +1987654321  | JANE_SMITH     |
|        3 | +1122334455  | MIKE_JONES     |
+----------+--------------+----------------+
3 rows in set (0.00 sec)

mysql> SELECT*FROM  stafflogin;
+---------------+----------------+
| StaffUsername | StaffPassword  |
+---------------+----------------+
| JANE_SMITH    | securepassword |
| JOHN_DOE      | password123    |
| MIKE_JONES    | letmein        |
+---------------+----------------+
3 rows in set (0.00 sec)

mysql> SELECT*FROM  testfees;
+-------------+-----------------+---------------+----------------+
| Test_Fee_ID | Test_Fee_Amount | Test_Fee_Type | Test_Fee_Total |
+-------------+-----------------+---------------+----------------+
|           1 |           80.00 | Blood test    |          80.00 |
|           2 |          120.00 | X-ray         |         120.00 |
|           3 |          150.00 | Ultrasound    |         150.00 |
+-------------+-----------------+---------------+----------------+
3 rows in set (0.00 sec)

mysql>
```

CHAPTER 3

COMPLEX QUIRIES:

TRIGGERS



```
-- Trigger for the PRESCRIPTION table
CREATE OR REPLACE TRIGGER prescription_trigger
BEFORE INSERT ON prescription
FOR EACH ROW
BEGIN
    -- Trigger action (Example: Print a message)
    DBMS_OUTPUT.PUT_LINE('New prescription added: ' || :new.report_desc);
END;
/
```

```
Trigger APPOINTMENTHISTORY_TRIGGER compiled


Trigger MEDICALTEST_TRIGGER compiled


Trigger PRESCRIPTION_TRIGGER compiled
```

```
-- Trigger for the STAFF table
CREATE OR REPLACE TRIGGER staff_trigger
BEFORE INSERT ON staff
FOR EACH ROW
BEGIN
    -- Trigger action (Example: Print a message)
    DBMS_OUTPUT.PUT_LINE('New staff member inserted: ' || :new.staff_username);
END;
/


-- Trigger for the PATIENT table
CREATE OR REPLACE TRIGGER patient_trigger
BEFORE INSERT ON patient
FOR EACH ROW
BEGIN
    -- Trigger action (Example: Print a message)
    DBMS_OUTPUT.PUT_LINE('New patient added: ' || :new.patient_name);
END;
/
```

```
-- Trigger for the DOCTOR table
CREATE OR REPLACE TRIGGER doctor_trigger
BEFORE INSERT ON doctor
FOR EACH ROW
BEGIN
   -- Trigger action (Example: Print a message)
   DBMS_OUTPUT.PUT_LINE('New doctor added: ' || :new.doctor_name);
END;
/

-- Trigger for the BOOKINGS table
CREATE OR REPLACE TRIGGER bookings_trigger
BEFORE INSERT ON bookings
FOR EACH ROW
BEGIN
   -- Trigger action (Example: Print a message)
   DBMS_OUTPUT.PUT_LINE('New booking created: ' || :new.booking_description);
END;
/

-- Trigger for the REGISTRATION table
CREATE OR REPLACE TRIGGER registration_trigger
BEFORE INSERT ON registration
FOR EACH ROW
BEGIN
   -- Trigger action (Example: Print a message)
   DBMS_OUTPUT.PUT_LINE('New registration: ' || :new.name);
END;
/

-- Trigger for the APPOINTMENTHISTORY table
CREATE OR REPLACE TRIGGER appointmenthistory_trigger
BEFORE INSERT ON appointmenthistory
FOR EACH ROW
BEGIN
```

```
    -- Trigger action (Example: Print a message)
    DBMS_OUTPUT.PUT_LINE('New appointment recorded: ' || :new.appointment_desc);
END;
/

-- Trigger for the MEDICALTEST table
CREATE OR REPLACE TRIGGER medicaltest_trigger
BEFORE INSERT ON medicaltest
FOR EACH ROW
BEGIN
    -- Trigger action (Example: Print a message)
    DBMS_OUTPUT.PUT_LINE('New medical test scheduled: ' || :new.test_desc);
END;
/

-- Trigger for the PRESCRIPTION table
CREATE OR REPLACE TRIGGER prescription_trigger
BEFORE INSERT ON prescription
FOR EACH ROW
BEGIN
    -- Trigger action (Example: Print a message)
    DBMS_OUTPUT.PUT_LINE('New prescription added: ' || :new.report_desc);
END;
/
```

VIEWS

```
-- View for the STAFF table
CREATE OR REPLACE VIEW staff_view AS
SELECT * FROM staff;

-- View for the PATIENT table
CREATE OR REPLACE VIEW patient_view AS
SELECT * FROM patient;

-- View for the DOCTOR table
CREATE OR REPLACE VIEW doctor_view AS
SELECT * FROM doctor;
```

```
-- View for the BOOKINGS table
CREATE OR REPLACE VIEW bookings_view AS
SELECT * FROM bookings;

-- View for the REGISTRATION table
CREATE OR REPLACE VIEW registration_view AS
SELECT * FROM registration;

-- View for the APPOINTMENTHISTORY table
CREATE OR REPLACE VIEW appointmenthistory_view AS
SELECT * FROM appointmenthistory;

-- View for the MEDICALTEST table
CREATE OR REPLACE VIEW medicaltest_view AS
SELECT * FROM medicaltest;

-- View for the PRESCRIPTION table
CREATE OR REPLACE VIEW prescription_view AS
SELECT * FROM prescription;

-- View for the STAFFLOGIN table
CREATE OR REPLACE VIEW stafflogin_view AS
SELECT * FROM stafflogin;

-- View for the PATIENTLOGIN table
CREATE OR REPLACE VIEW patientlogin_view AS
SELECT * FROM patientlogin;

-- View for the DOCTORLOGIN table
CREATE OR REPLACE VIEW doctorlogin_view AS
SELECT * FROM doctorlogin;

-- View for the APPOINTMENTFEES table
CREATE OR REPLACE VIEW appointmentfees_view AS
SELECT * FROM appointmentfees;

-- View for the TESTFEES table
CREATE OR REPLACE VIEW testfees_view AS
```

SELECT * FROM testfees;

-- View for the PATIENTSCHEDULES table

CREATE OR REPLACE VIEW patientschedules_view AS

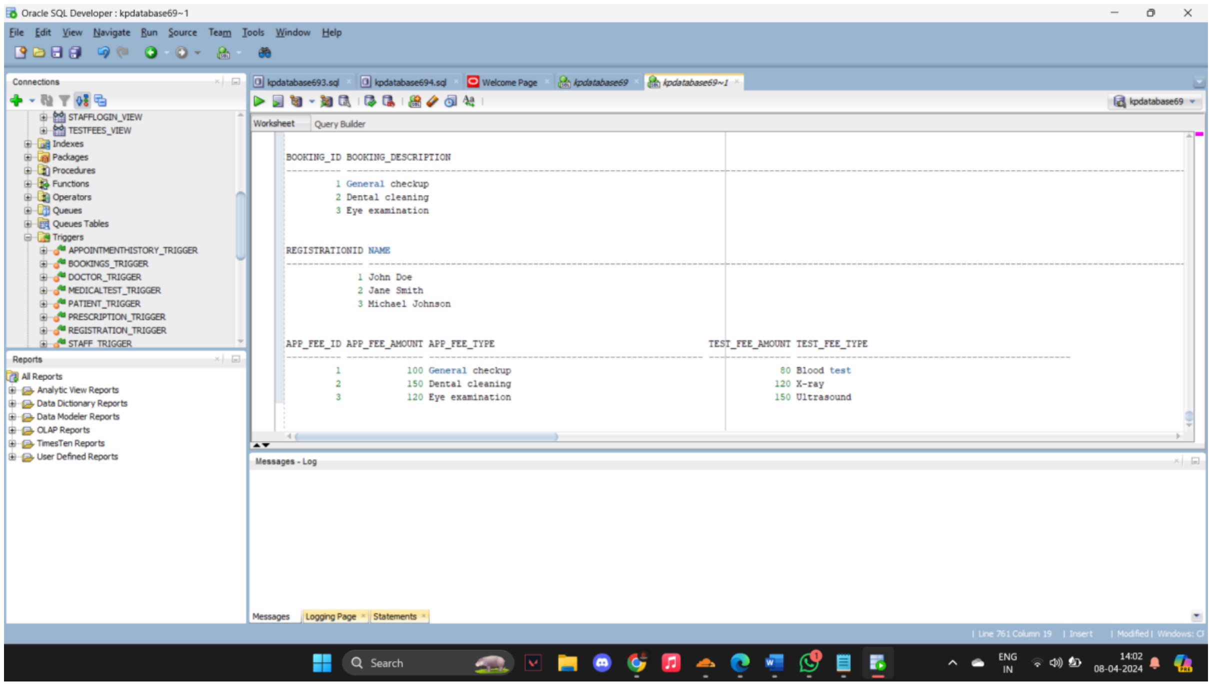
SELECT * FROM patientschedules;

-- View for the DOCTORSCHEDULES table

CREATE OR REPLACE VIEW doctorschedules_view AS

SELECT * FROM doctorschedules;

JOIN



```
BOOKING_ID BOOKING_DESCRIPTION
---------- ---------------------------------------------
         1 General checkup
         2 Dental cleaning
         3 Eye examination


REGISTRATIONID NAME
-------------- ---------------------------------------------
             1 John Doe
             2 Jane Smith
             3 Michael Johnson


APP_FEE_ID APP_FEE_AMOUNT APP_FEE_TYPE                                              TEST_FEE_AMOUNT TEST_FEE_TYPE
---------- -------------- --------------------------------------------------------- --------------- ----------------------------------------
         1            100 General checkup                                                        80 Blood test
         2            150 Dental cleaning                                                       120 X-ray
         3            120 Eye examination                                                       150 Ultrasound
```

-- Joining the STAFF and StaffLogin tables

SELECT s.staff_id, s.staff_mobile, s.staff_username, sl.staffusername

FROM staff s

INNER JOIN StaffLogin sl ON s.staff_username = sl.staffusername;

-- Joining the PATIENT and PatientLogin tables

SELECT p.patient_id, p.patient_name, p.patient_mail, pl.pat_username

FROM patient p

INNER JOIN PatientLogin pl ON p.patient_id = pl.pat_login_id;

-- Joining the DOCTOR and DoctorLogin tables

SELECT d.doc_id, d.doctor_mobile, d.doctor_name, dl.doc_username

FROM doctor d

INNER JOIN DoctorLogin dl ON d.doc_id = dl.doc_login_id;

```
-- Joining the BOOKINGS and AppointmentHistory tables
SELECT b.booking_id, b.booking_description, ah.appointment_date, ah.appointment_desc
FROM bookings b
INNER JOIN AppointmentHistory ah ON b.booking_id = ah.appointment_id;

-- Joining the Registration and MedicalTest tables
SELECT r.RegistrationID, r.Name, r.Age, mt.test_date, mt.test_desc
FROM Registration r
INNER JOIN MedicalTest mt ON r.RegistrationID = mt.test_id;

-- Joining the AppointmentFees and TestFees tables
SELECT af.app_fee_id, af.app_fee_amount, af.app_fee_type, tf.test_fee_amount, tf.test_fee_type
FROM AppointmentFees af
INNER JOIN TestFees tf ON af.app_fee_id = tf.test_fee_id;
```

Cursor:

```
-- Cursor for STAFF table
DECLARE
   CURSOR staff_cursor IS
      SELECT * FROM staff;
BEGIN
   -- Loop through the cursor and do something with the data
   FOR staff_rec IN staff_cursor LOOP
      -- Access data using staff_rec.column_name
      DBMS_OUTPUT.PUT_LINE('Staff ID: ' || staff_rec.staff_id || ', Username: ' || staff_rec.staff_username);
      -- You can perform any operations here with the fetched data
   END LOOP;
END;
/
-- Cursor for PATIENT table
DECLARE
   CURSOR patient_cursor IS
      SELECT * FROM patient;
BEGIN
   FOR patient_rec IN patient_cursor LOOP
      DBMS_OUTPUT.PUT_LINE('Patient ID: ' || patient_rec.patient_id || ', Name: ' || patient_rec.patient_name);
   END LOOP;
```

```
END;
/

-- Repeat the above pattern for other tables (doctor, bookings, etc.)
-- Cursor for DOCTOR table
DECLARE
   CURSOR doctor_cursor IS
      SELECT * FROM doctor;
BEGIN
   FOR doctor_rec IN doctor_cursor LOOP
      DBMS_OUTPUT.PUT_LINE('Doctor ID: ' || doctor_rec.doc_id || ', Name: ' || doctor_rec.doctor_name);
   END LOOP;
END;
/

-- Cursor for BOOKINGS table
DECLARE
   CURSOR bookings_cursor IS
      SELECT * FROM bookings;
BEGIN
   FOR bookings_rec IN bookings_cursor LOOP
      DBMS_OUTPUT.PUT_LINE('Booking ID: ' || bookings_rec.booking_id || ', Description: ' ||
bookings_rec.booking_description);
   END LOOP;
END;
/

-- Cursor for Registration table
DECLARE
   CURSOR registration_cursor IS
      SELECT * FROM Registration;
BEGIN
   FOR registration_rec IN registration_cursor LOOP
      DBMS_OUTPUT.PUT_LINE('Registration ID: ' || registration_rec.RegistrationID || ', Name: ' ||
registration_rec.Name);
   END LOOP;
END;
```

```
/

-- Cursor for StaffLogin table
DECLARE
   CURSOR staff_login_cursor IS
      SELECT * FROM StaffLogin;
BEGIN
   FOR staff_login_rec IN staff_login_cursor LOOP
      DBMS_OUTPUT.PUT_LINE('Staff Username: ' || staff_login_rec.StaffUsername);
   END LOOP;
END;
/

-- Cursor for PatientLogin table
DECLARE
   CURSOR patient_login_cursor IS
      SELECT * FROM PatientLogin;
BEGIN
   FOR patient_login_rec IN patient_login_cursor LOOP
      DBMS_OUTPUT.PUT_LINE('Patient Username: ' || patient_login_rec.Pat_Username);
   END LOOP;
END;
/

-- Cursor for DoctorLogin table
DECLARE
   CURSOR doctor_login_cursor IS
      SELECT * FROM DoctorLogin;
BEGIN
   FOR doctor_login_rec IN doctor_login_cursor LOOP
      DBMS_OUTPUT.PUT_LINE('Doctor Username: ' || doctor_login_rec.Doc_Username);
   END LOOP;
END;
/

-- Cursor for AppointmentHistory table
DECLARE
   CURSOR appointment_history_cursor IS
```

```
        SELECT * FROM AppointmentHistory;

BEGIN
    FOR appointment_history_rec IN appointment_history_cursor LOOP
        DBMS_OUTPUT.PUT_LINE('Appointment ID: ' || appointment_history_rec.Appointment_ID || ', Date: ' ||
appointment_history_rec.Appointment_Date);
    END LOOP;
END;
/

-- Cursor for AppointmentFees table
DECLARE
    CURSOR appointment_fees_cursor IS
        SELECT * FROM AppointmentFees;
BEGIN
    FOR appointment_fees_rec IN appointment_fees_cursor LOOP
        DBMS_OUTPUT.PUT_LINE('Fee ID: ' || appointment_fees_rec.App_Fee_ID || ', Amount: ' ||
appointment_fees_rec.App_Fee_Amount);
    END LOOP;
END;
/

-- Cursor for MedicalTest table
DECLARE
    CURSOR medical_test_cursor IS
        SELECT * FROM MedicalTest;
BEGIN
    FOR medical_test_rec IN medical_test_cursor LOOP
        DBMS_OUTPUT.PUT_LINE('Test ID: ' || medical_test_rec.Test_ID || ', Date: ' ||
medical_test_rec.Test_Date);
    END LOOP;
END;
/

-- Cursor for TestFees table
DECLARE
    CURSOR test_fees_cursor IS
        SELECT * FROM TestFees;
```

```
BEGIN
   FOR test_fees_rec IN test_fees_cursor LOOP
      DBMS_OUTPUT.PUT_LINE('Test Fee ID: ' || test_fees_rec.Test_Fee_ID || ', Amount: ' ||
test_fees_rec.Test_Fee_Amount);
   END LOOP;
END;
/

-- Cursor for PatientSchedules table
DECLARE
   CURSOR patient_schedules_cursor IS
      SELECT * FROM PatientSchedules;
BEGIN
   FOR patient_schedules_rec IN patient_schedules_cursor LOOP
      DBMS_OUTPUT.PUT_LINE('Patient Schedule ID: ' || patient_schedules_rec.Pat_Sch_ID || ', Type: ' ||
patient_schedules_rec.Pat_Sch_Type);
   END LOOP;
END;
/

-- Cursor for DoctorSchedules table
DECLARE
   CURSOR doctor_schedules_cursor IS
      SELECT * FROM DoctorSchedules;
BEGIN
   FOR doctor_schedules_rec IN doctor_schedules_cursor LOOP
      DBMS_OUTPUT.PUT_LINE('Doctor Schedule ID: ' || doctor_schedules_rec.Doc_Sch_ID || ', Type: ' ||
doctor_schedules_rec.Doc_Sch_Type);
   END LOOP;
END;
/

-- Cursor for Prescription table
DECLARE
   CURSOR prescription_cursor IS
      SELECT * FROM Prescription;
```

```
BEGIN
    FOR prescription_rec IN prescription_cursor LOOP
        DBMS_OUTPUT.PUT_LINE('Prescription ID: ' || prescription_rec.Report_ID || ', Description: ' ||
prescription_rec.Report_Desc);
    END LOOP;
END;
/
```

Sets

To perform set operations on the tables, you can use SQL set operators like UNION, INTERSECT, and MINUS. Here's an example of how you can perform set operations for each table:

1. **Staff Table:**
```sql
-- UNION of staff_mobile and staff_username
SELECT staff_mobile AS data FROM staff
UNION
SELECT staff_username FROM staff;
```

2. **Patient Table:**
```sql
-- INTERSECT of patient_name and city
SELECT patient_name AS data FROM patient
INTERSECT
SELECT city FROM patient;
```

3. **Doctor Table:**
```sql
-- UNION of doctor_mobile and doctor_address
SELECT doctor_mobile AS data FROM doctor
UNION
SELECT doctor_address FROM doctor;

```

4. **Bookings Table:**

```sql
-- MINUS of booking_description and booking_type
SELECT booking_description AS data FROM bookings
MINUS
SELECT booking_type FROM bookings;
```

5. **Registration Table:**
```sql
-- UNION of Name and Gender
SELECT Name AS data FROM registration
UNION
SELECT Gender FROM registration;
```
6. **StaffLogin Table:**
```sql
-- INTERSECT of StaffUsername and StaffPassword
SELECT StaffUsername AS data FROM stafflogin
INTERSECT
SELECT StaffPassword FROM stafflogin;
```

7. **PatientLogin Table:**
```sql
-- MINUS of Pat_Username and Pat_Password
SELECT Pat_Username AS data FROM patientlogin
MINUS
SELECT Pat_Password FROM patientlogin;
```

8. **DoctorLogin Table:**
```sql
-- UNION of Doc_Username and Doc_Password
SELECT Doc_Username AS data FROM doctorlogin
UNION
SELECT Doc_Password FROM doctorlogin;
```

9. **AppointmentHistory Table:**
```sql
-- INTERSECT of Appointment_ID and Appointment_Desc
SELECT Appointment_ID AS data FROM appointmenthistory
INTERSECT
SELECT Appointment_Desc FROM appointmenthistory;
```

10. **AppointmentFees Table:**
```sql
-- MINUS of App_Fee_Amount and App_Fee_Total
SELECT App_Fee_Amount AS data FROM appointmentfees
MINUS
SELECT App_Fee_Total FROM appointmentfees;
```
11. **MedicalTest Table:**
-- UNION of Test_ID and Test_Desc
SELECT CAST(Test_ID AS VARCHAR(255)) AS data FROM MedicalTest
UNION
SELECT Test_Desc FROM MedicalTest;

12. **TestFees Table:**
```sql
-- INTERSECT of Test_Fee_Amount and Test_Fee_Total
SELECT Test_Fee_Amount AS data FROM testfees
INTERSECT
SELECT Test_Fee_Total FROM testfees;
```
13. **PatientSchedules Table:**
~~~sql
-- MINUS of Pat_Sch_Type and Pat_Sch_Desc
SELECT Pat_Sch_Type AS data FROM patientschedules
MINUS
SELECT Pat_Sch_Desc FROM patientschedules;
14. **DoctorSchedules Table:**
```sql
-- UNION of Doc_Sch_Type and Doc_Sch_Desc
SELECT Doc_Sch_Type AS data FROM doctorschedules
~~~

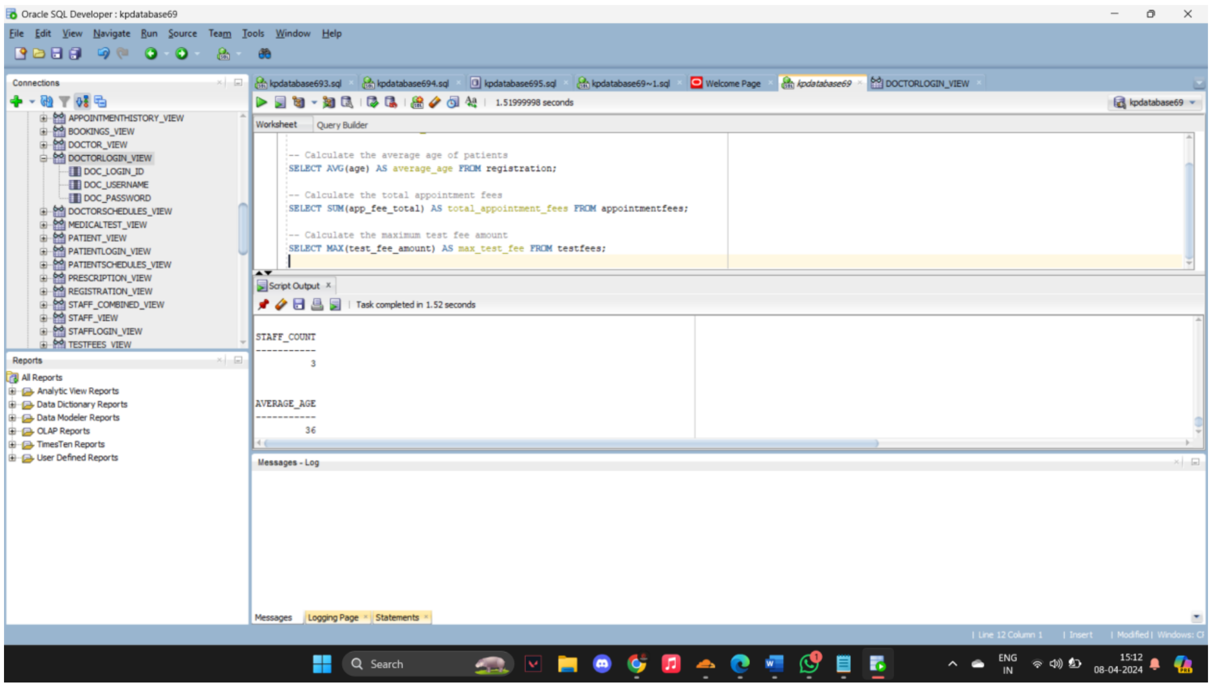
UNION

SELECT Doc_Sch_Desc FROM doctorschedules;

15. **Prescription Table:**

```sql

-- INTERSECT of Report_ID and Report_Desc

SELECT Report_ID AS data FROM prescription

INTERSECT

SELECT Report_Desc FROM prescription;

```

Aggregate functions



```
-- Calculate the average age of patients
SELECT AVG(age) AS average_age FROM registration;

-- Calculate the total appointment fees
SELECT SUM(app_fee_total) AS total_appointment_fees FROM appointmentfees;

-- Calculate the maximum test fee amount
SELECT MAX(test_fee_amount) AS max_test_fee FROM testfees;
```

Task completed in 1.52 seconds

```
STAFF_COUNT
-----------
          3


AVERAGE_AGE
-----------
         36
```

-- Count the number of staff members

SELECT COUNT(*) AS staff_count FROM staff;

-- Calculate the average age of patients

SELECT AVG(age) AS average_age FROM registration;

-- Calculate the total appointment fees

SELECT SUM(app_fee_total) AS total_appointment_fees FROM appointmentfees;

CHAPTER 4

PITFALLS:

1. Naming Conventions: • Inconsistency in naming conventions: Some tables use uppercase names (STAFF, PATIENT), while others use lowercase (staff, patient). Consistency improves readability and maintainability.

2. Data Integrity: • Lack of foreign key constraints: There are no explicit foreign key constraints defined between tables, which could lead to orphaned records and data integrity issues. • Missing referential integrity: Without foreign key constraints, there's no guarantee that references between tables are valid, increasing the risk of inconsistent data.

3. Data Types: • Inadequate data types: Some columns might not have appropriate data types. For example, AGE in the Registration table is defined as INT, which may not handle fractional ages or specific date of birth information effectively.

4. Data Redundancy: • Redundant information: For instance, both the staff and StaffLogin tables contain information about staff members, leading to potential redundancy and inconsistency if not properly synchronized.

5. Normalization: • Denormalization in login tables: Storing login credentials in separate tables (StaffLogin, PatientLogin, DoctorLogin) rather than associating them with their respective entities (staff, patient, doctor) violates normalization principles and can lead to data duplication and inconsistency.

6. Unique Constraints: • Unique constraint violations: Unique constraints are defined on certain columns like staff_mobile and staff_username, but there's no handling mechanism specified for potential violations.

7. Data Validation: • Lack of data validation: There's no apparent data validation or constraints defined to ensure the accuracy and integrity of the data entered into the tables.

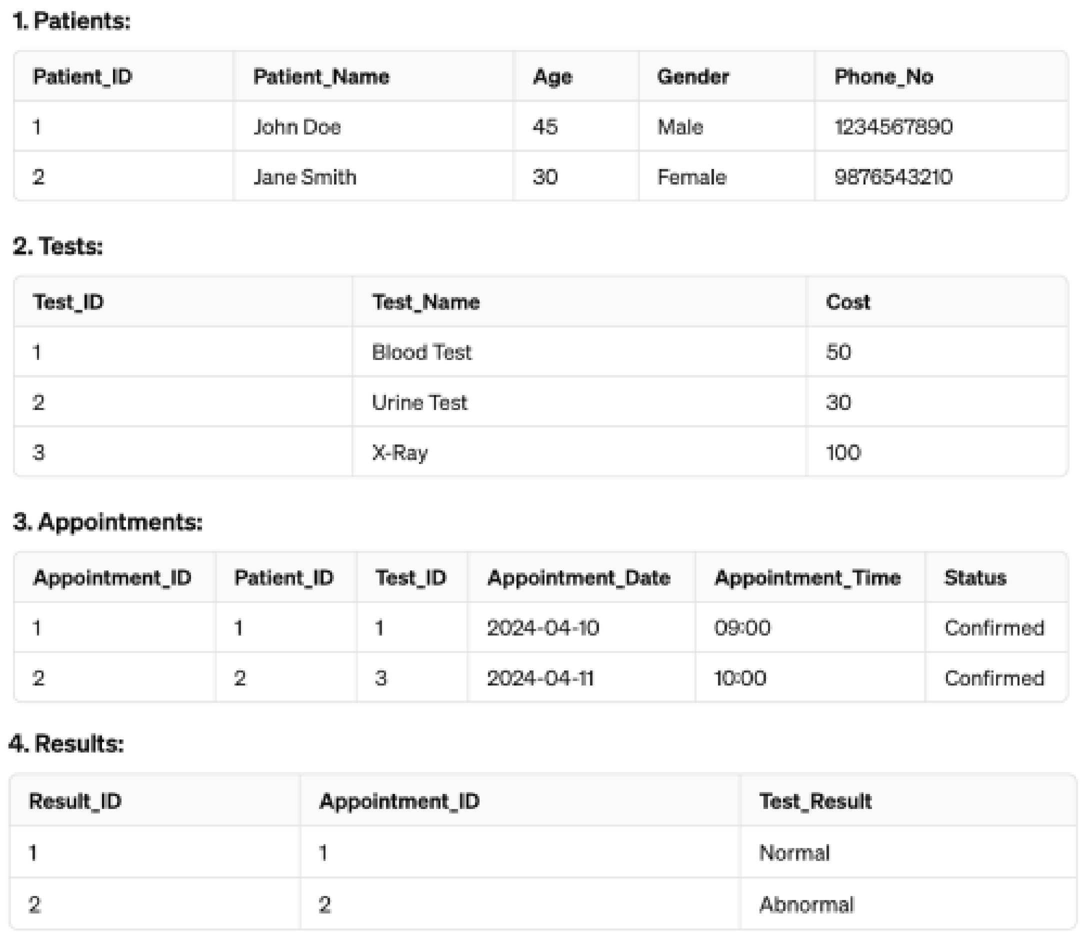
8. Security Concerns: • Password storage: Storing passwords in plain text (StaffPassword, Pat_Password, Doc_Password) poses security risks. Passwords should be hashed and salted for better security.

9. Scalability: • Scalability issues: As the application grows, maintaining separate login tables for each entity (staff, patient, doctor) may become cumbersome and less scalable.

10. Auditability: • Lack of audit trails: There's no provision for tracking changes to the data, which is essential for auditing.

NORMALISATION:

SAMPLE TABLES



1. Patients:

Patient_ID	Patient_Name	Age	Gender	Phone_No
1	John Doe	45	Male	1234567890
2	Jane Smith	30	Female	9876543210

2. Tests:

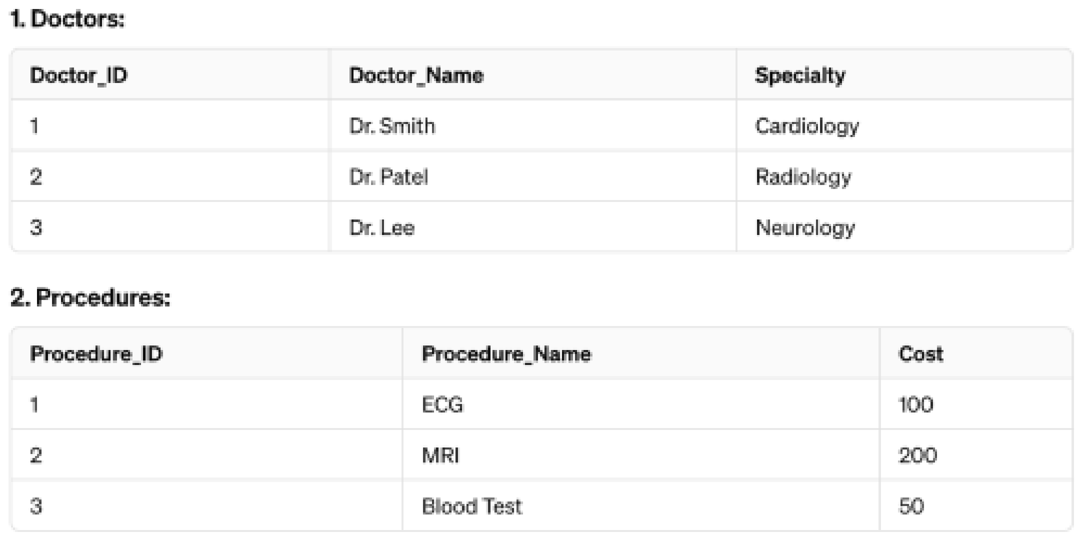
Test_ID	Test_Name	Cost
1	Blood Test	50
2	Urine Test	30
3	X-Ray	100

3. Appointments:

Appointment_ID	Patient_ID	Test_ID	Appointment_Date	Appointment_Time	Status
1	1	1	2024-04-10	09:00	Confirmed
2	2	3	2024-04-11	10:00	Confirmed

4. Results:

Result_ID	Appointment_ID	Test_Result
1	1	Normal
2	2	Abnormal

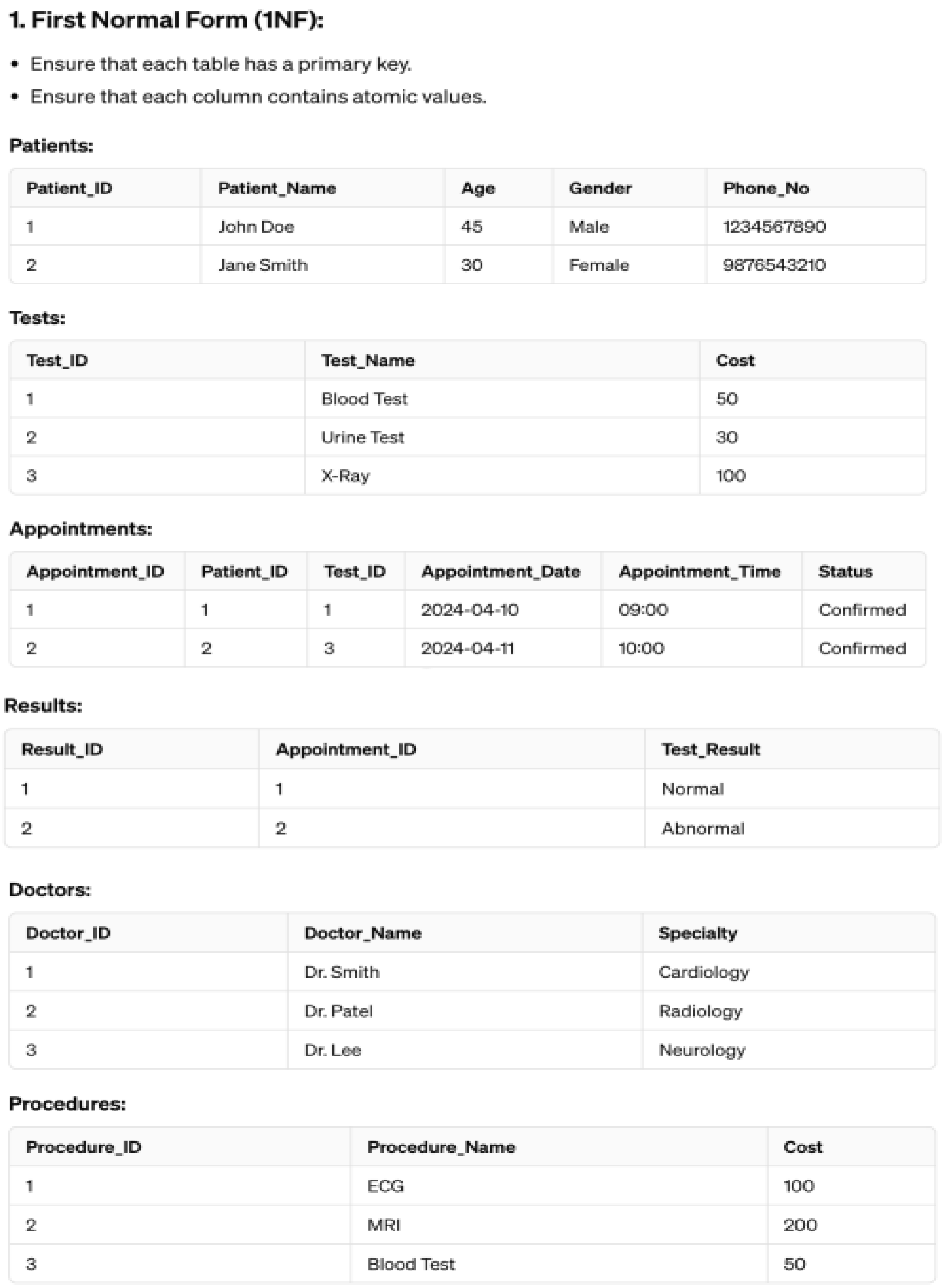


1. Doctors:

Doctor_ID	Doctor_Name	Specialty
1	Dr. Smith	Cardiology
2	Dr. Patel	Radiology
3	Dr. Lee	Neurology

2. Procedures:

Procedure_ID	Procedure_Name	Cost
1	ECG	100
2	MRI	200
3	Blood Test	50



1. First Normal Form (1NF):

- Ensure that each table has a primary key.
- Ensure that each column contains atomic values.

Patients:

Patient_ID	Patient_Name	Age	Gender	Phone_No
1	John Doe	45	Male	1234567890
2	Jane Smith	30	Female	9876543210

Tests:

Test_ID	Test_Name	Cost
1	Blood Test	50
2	Urine Test	30
3	X-Ray	100

Appointments:

Appointment_ID	Patient_ID	Test_ID	Appointment_Date	Appointment_Time	Status
1	1	1	2024-04-10	09:00	Confirmed
2	2	3	2024-04-11	10:00	Confirmed

Results:

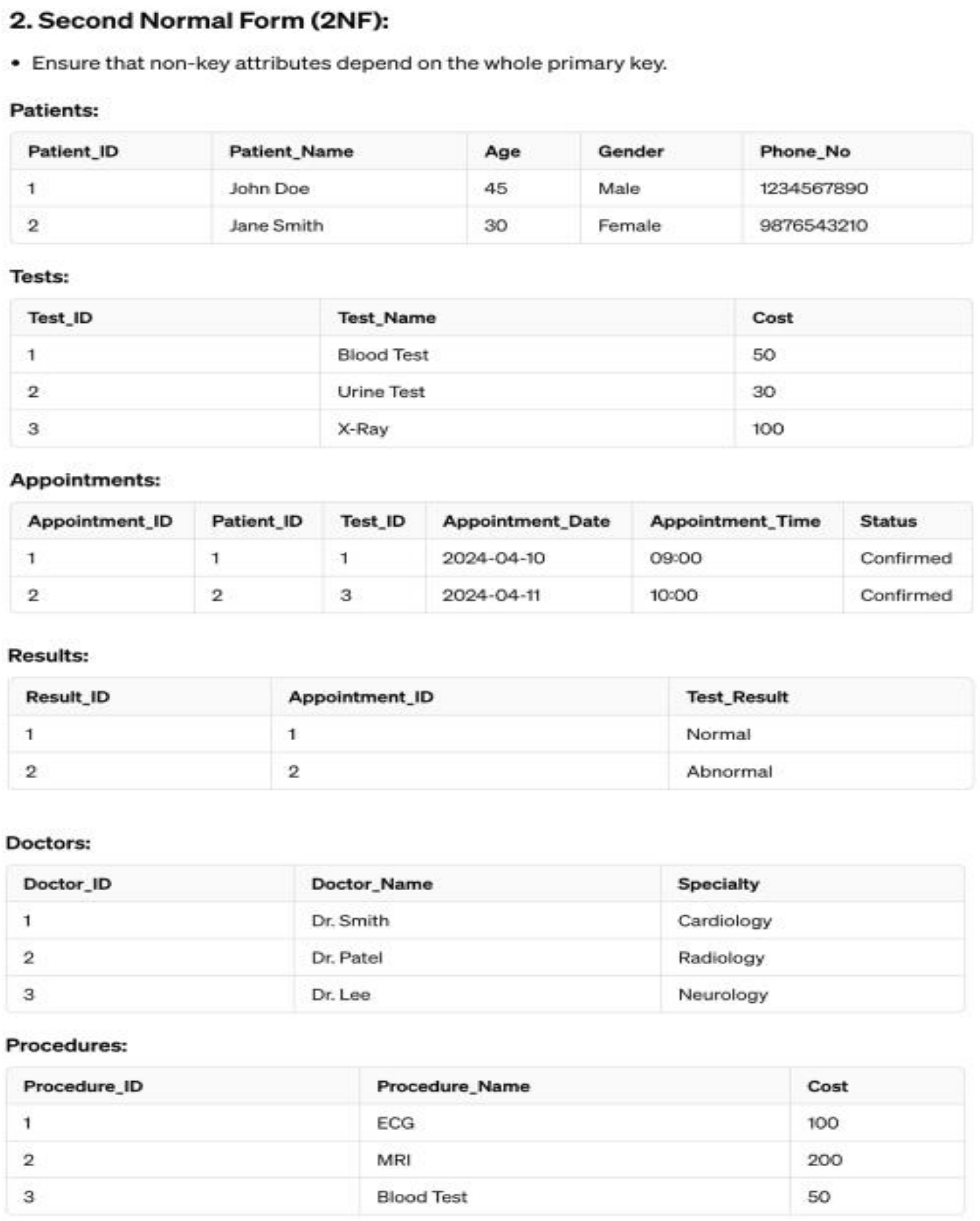
Result_ID	Appointment_ID	Test_Result
1	1	Normal
2	2	Abnormal

Doctors:

Doctor_ID	Doctor_Name	Specialty
1	Dr. Smith	Cardiology
2	Dr. Patel	Radiology
3	Dr. Lee	Neurology

Procedures:

Procedure_ID	Procedure_Name	Cost
1	ECG	100
2	MRI	200
3	Blood Test	50



2. Second Normal Form (2NF):

- Ensure that non-key attributes depend on the whole primary key.

Patients:

Patient_ID	Patient_Name	Age	Gender	Phone_No
1	John Doe	45	Male	1234567890
2	Jane Smith	30	Female	9876543210

Tests:

Test_ID	Test_Name	Cost
1	Blood Test	50
2	Urine Test	30
3	X-Ray	100

Appointments:

Appointment_ID	Patient_ID	Test_ID	Appointment_Date	Appointment_Time	Status
1	1	1	2024-04-10	09:00	Confirmed
2	2	3	2024-04-11	10:00	Confirmed

Results:

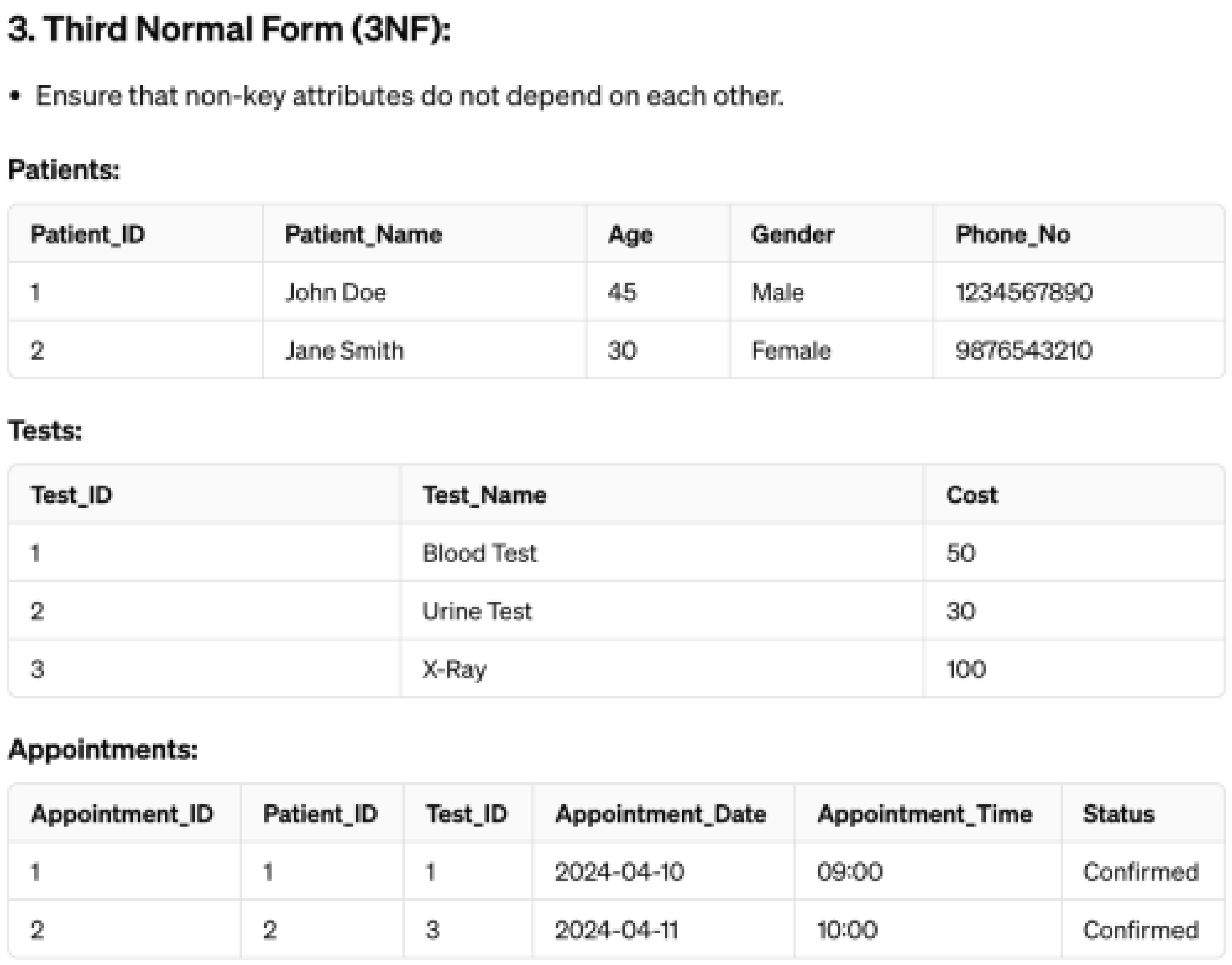
Result_ID	Appointment_ID	Test_Result
1	1	Normal
2	2	Abnormal

Doctors:

Doctor_ID	Doctor_Name	Specialty
1	Dr. Smith	Cardiology
2	Dr. Patel	Radiology
3	Dr. Lee	Neurology

Procedures:

Procedure_ID	Procedure_Name	Cost
1	ECG	100
2	MRI	200
3	Blood Test	50



3. Third Normal Form (3NF):

- Ensure that non-key attributes do not depend on each other.

Patients:

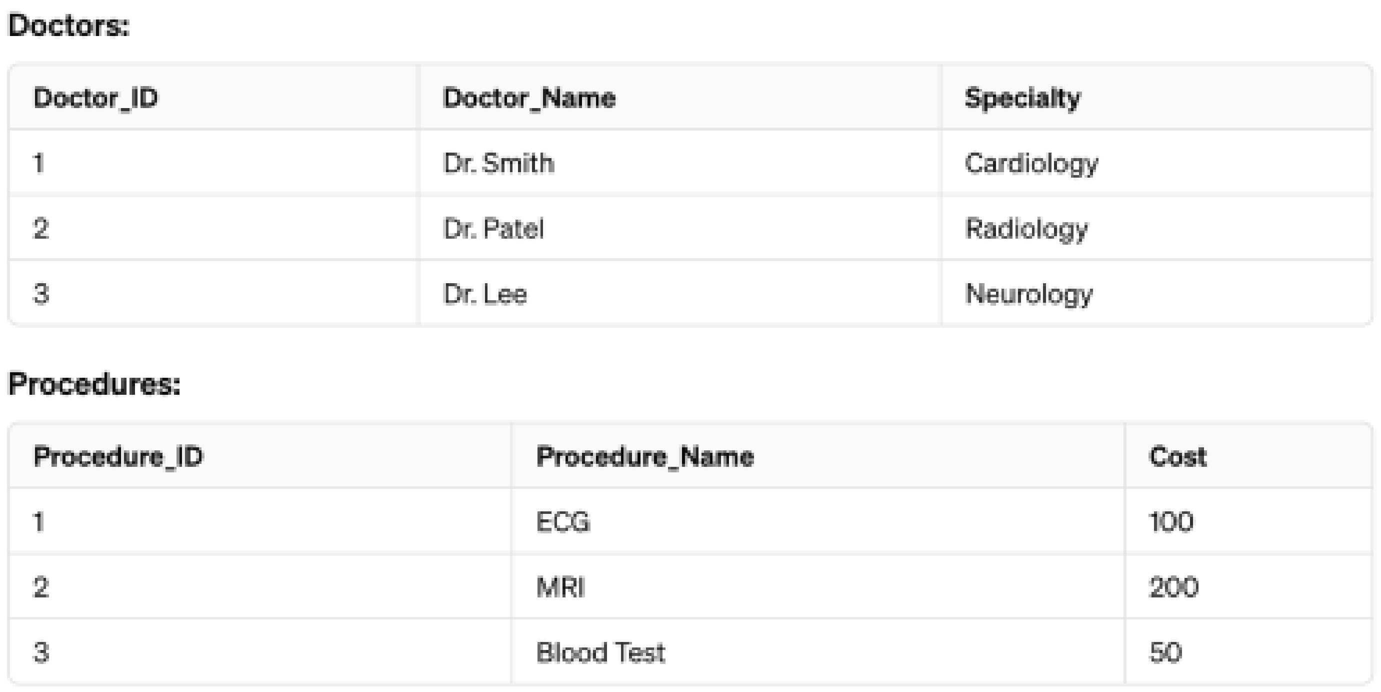
Patient_ID	Patient_Name	Age	Gender	Phone_No
1	John Doe	45	Male	1234567890
2	Jane Smith	30	Female	9876543210

Tests:

Test_ID	Test_Name	Cost
1	Blood Test	50
2	Urine Test	30
3	X-Ray	100

Appointments:

Appointment_ID	Patient_ID	Test_ID	Appointment_Date	Appointment_Time	Status
1	1	1	2024-04-10	09:00	Confirmed
2	2	3	2024-04-11	10:00	Confirmed

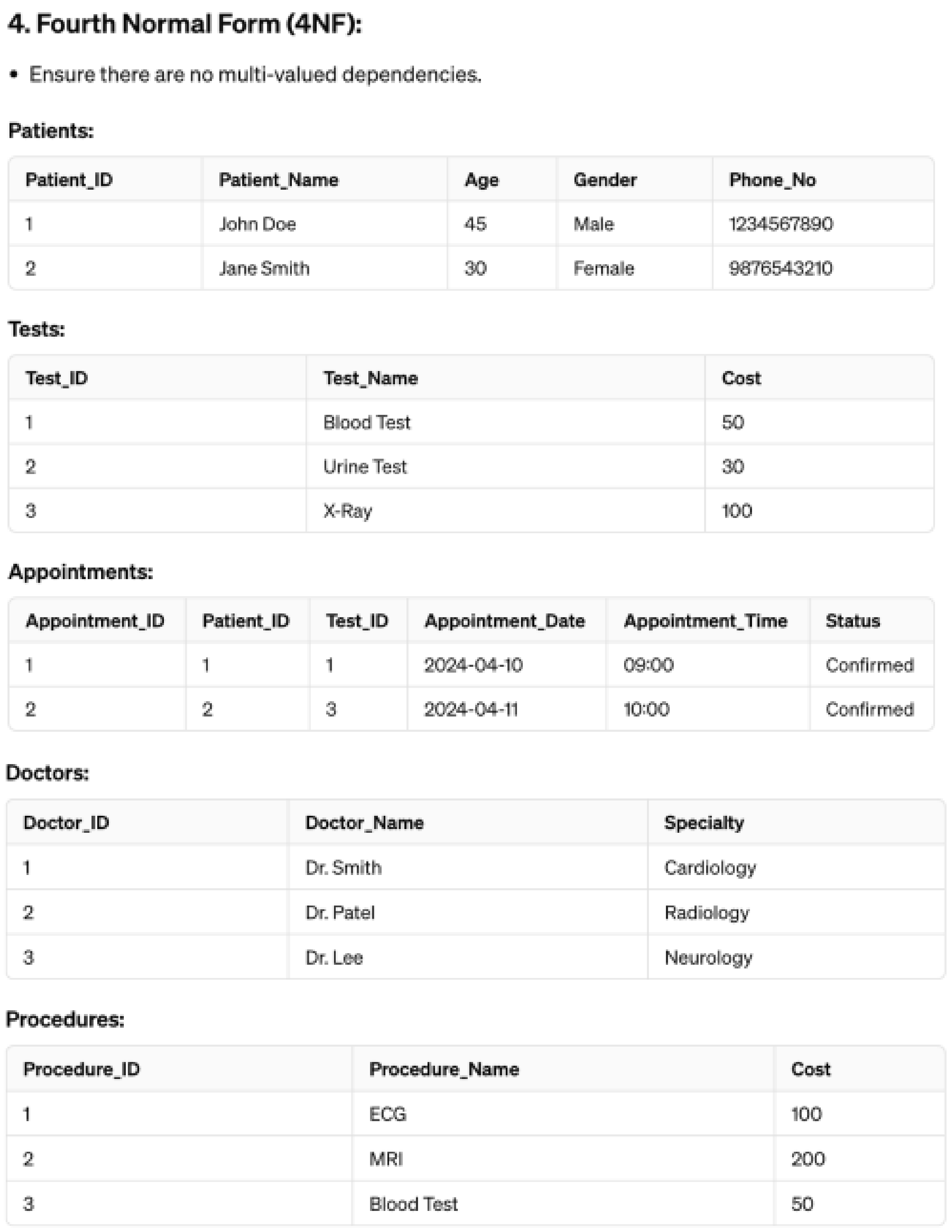


Doctors:

Doctor_ID	Doctor_Name	Specialty
1	Dr. Smith	Cardiology
2	Dr. Patel	Radiology
3	Dr. Lee	Neurology

Procedures:

Procedure_ID	Procedure_Name	Cost
1	ECG	100
2	MRI	200
3	Blood Test	50



4. Fourth Normal Form (4NF):

- Ensure there are no multi-valued dependencies.

Patients:

Patient_ID	Patient_Name	Age	Gender	Phone_No
1	John Doe	45	Male	1234567890
2	Jane Smith	30	Female	9876543210

Tests:

Test_ID	Test_Name	Cost
1	Blood Test	50
2	Urine Test	30
3	X-Ray	100

Appointments:

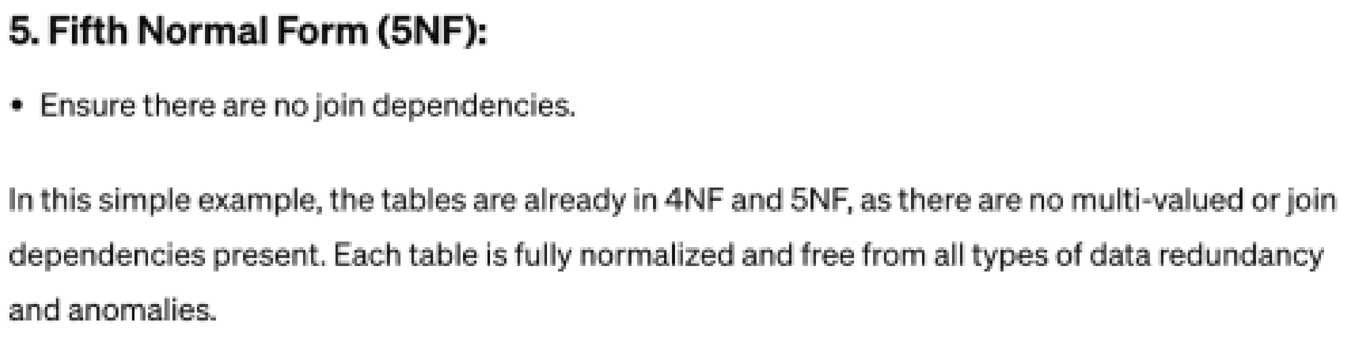
Appointment_ID	Patient_ID	Test_ID	Appointment_Date	Appointment_Time	Status
1	1	1	2024-04-10	09:00	Confirmed
2	2	3	2024-04-11	10:00	Confirmed

Doctors:

Doctor_ID	Doctor_Name	Specialty
1	Dr. Smith	Cardiology
2	Dr. Patel	Radiology
3	Dr. Lee	Neurology

Procedures:

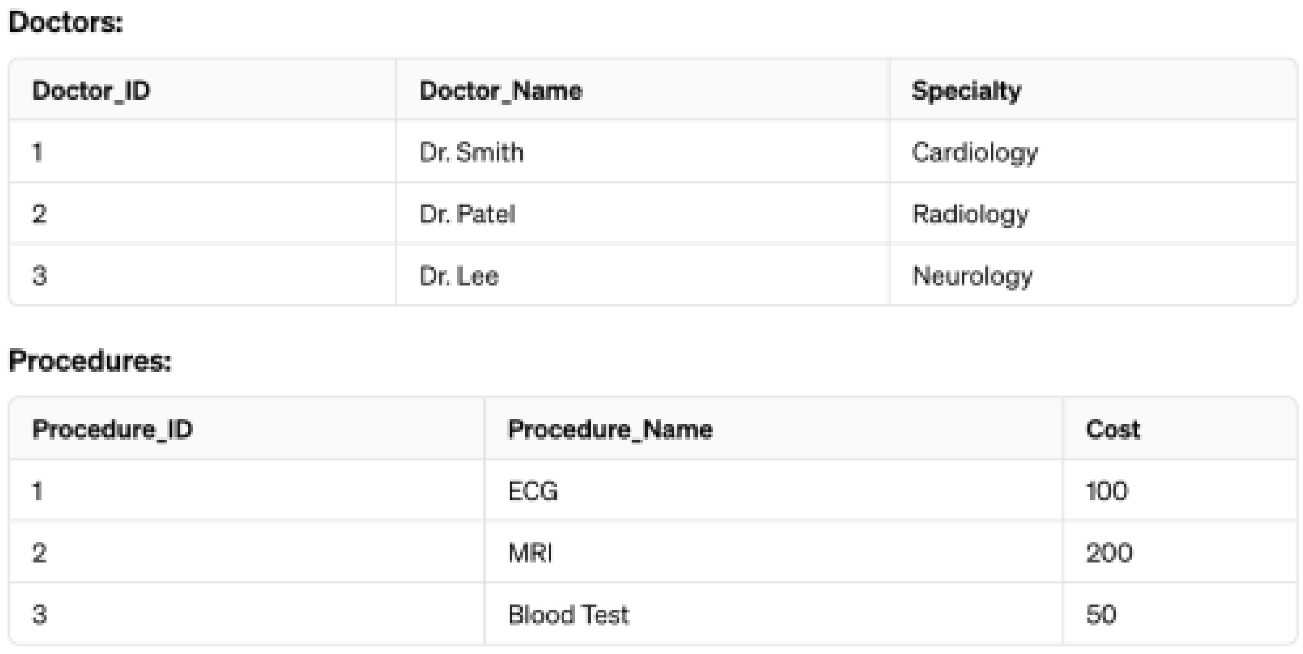
Procedure_ID	Procedure_Name	Cost
1	ECG	100
2	MRI	200
3	Blood Test	50



5. Fifth Normal Form (5NF):

- Ensure there are no join dependencies.

In this simple example, the tables are already in 4NF and 5NF, as there are no multi-valued or join dependencies present. Each table is fully normalized and free from all types of data redundancy and anomalies.



Doctors:

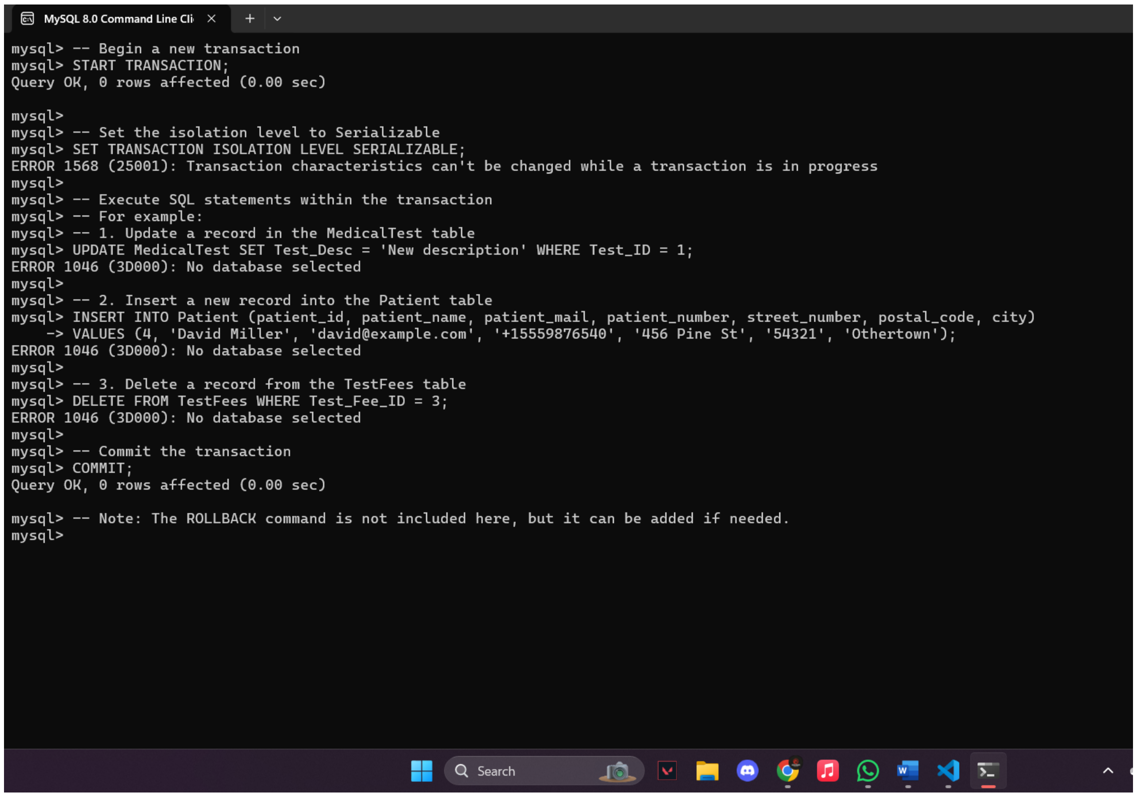
Doctor_ID	Doctor_Name	Specialty
1	Dr. Smith	Cardiology
2	Dr. Patel	Radiology
3	Dr. Lee	Neurology

Procedures:

Procedure_ID	Procedure_Name	Cost
1	ECG	100
2	MRI	200
3	Blood Test	50

CHAPTER 5

IMPLEMENTATION OF CONCURENCY:



```
mysql> -- Begin a new transaction
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> -- Set the isolation level to Serializable
mysql> SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
ERROR 1568 (25001): Transaction characteristics can't be changed while a transaction is in progress
mysql>
mysql> -- Execute SQL statements within the transaction
mysql> -- For example:
mysql> -- 1. Update a record in the MedicalTest table
mysql> UPDATE MedicalTest SET Test_Desc = 'New description' WHERE Test_ID = 1;
ERROR 1046 (3D000): No database selected
mysql>
mysql> -- 2. Insert a new record into the Patient table
mysql> INSERT INTO Patient (patient_id, patient_name, patient_mail, patient_number, street_number, postal_code, city)
    -> VALUES (4, 'David Miller', 'david@example.com', '+15559876540', '456 Pine St', '54321', 'Othertown');
ERROR 1046 (3D000): No database selected
mysql>
mysql> -- 3. Delete a record from the TestFees table
mysql> DELETE FROM TestFees WHERE Test_Fee_ID = 3;
ERROR 1046 (3D000): No database selected
mysql>
mysql> -- Commit the transaction
mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)

mysql> -- Note: The ROLLBACK command is not included here, but it can be added if needed.
mysql>
```

-- Begin a new transaction

START TRANSACTION;

-- Set the isolation level to Serializable

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

-- SQL statements within the transaction

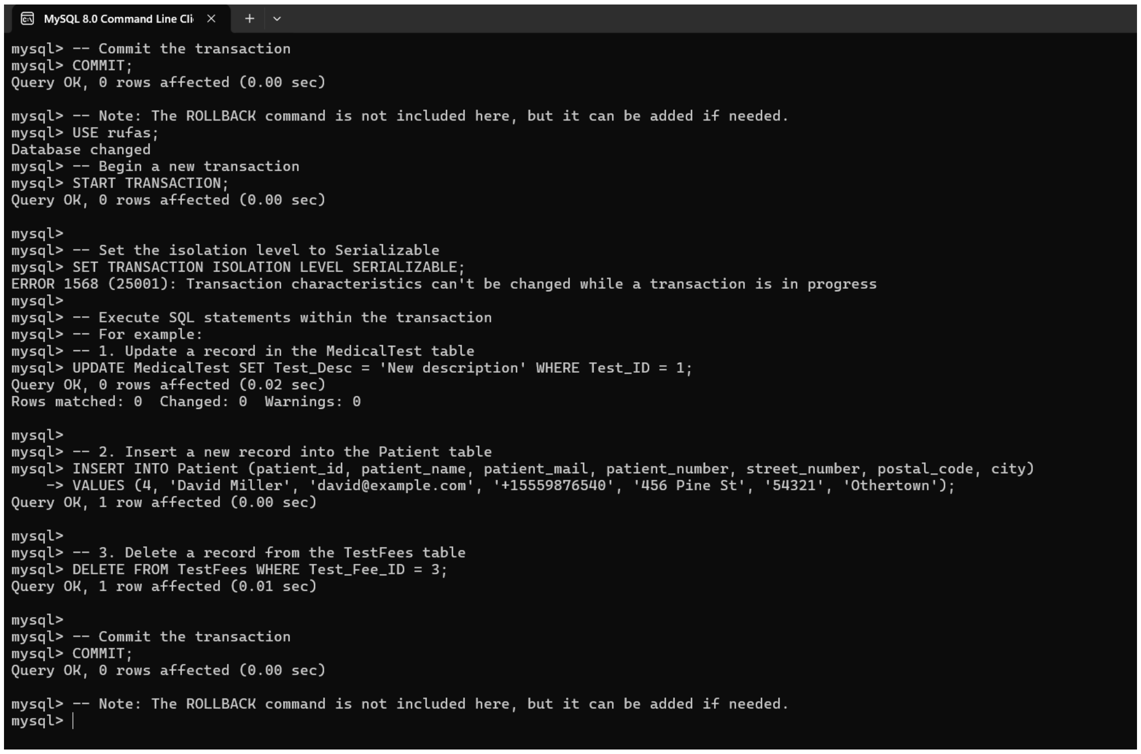
UPDATE electricity_bill_account SET status = 'inactive' WHERE account_id = 101;

INSERT INTO electricity_bill_customer ...; -- Insert values into electricity_bill_customer table

DELETE FROM electricity_bill_feedback WHERE cust_id = 202;

-- Commit the transaction

COMMIT;



```
mysql> -- Commit the transaction
mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)

mysql> -- Note: The ROLLBACK command is not included here, but it can be added if needed.
mysql> USE rufas;
Database changed
mysql> -- Begin a new transaction
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> -- Set the isolation level to Serializable
mysql> SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
ERROR 1568 (25001): Transaction characteristics can't be changed while a transaction is in progress
mysql>
mysql> -- Execute SQL statements within the transaction
mysql> -- For example:
mysql> -- 1. Update a record in the MedicalTest table
mysql> UPDATE MedicalTest SET Test_Desc = 'New description' WHERE Test_ID = 1;
Query OK, 0 rows affected (0.02 sec)
Rows matched: 0  Changed: 0  Warnings: 0

mysql>
mysql> -- 2. Insert a new record into the Patient table
mysql> INSERT INTO Patient (patient_id, patient_name, patient_mail, patient_number, street_number, postal_code, city)
    -> VALUES (4, 'David Miller', 'david@example.com', '+15559876540', '456 Pine St', '54321', 'Othertown');
Query OK, 1 row affected (0.00 sec)

mysql>
mysql> -- 3. Delete a record from the TestFees table
mysql> DELETE FROM TestFees WHERE Test_Fee_ID = 3;
Query OK, 1 row affected (0.01 sec)

mysql>
mysql> -- Commit the transaction
mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)

mysql> -- Note: The ROLLBACK command is not included here, but it can be added if needed.
mysql>
```

-- Begin a new transaction

START TRANSACTION;

-- Set the isolation level to Serializable

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

-- 1. Update a record in the MedicalTest table

UPDATE MedicalTest SET Test_Desc = 'New description' WHERE Test_ID = 1;

-- 2. Insert a new record into the Patient table

INSERT INTO Patient (patient_id, patient_name, patient_mail, patient_number, street_number, postal_code, city)

VALUES (4, 'David Miller', 'david@example.com', '+15559876540', '456 Pine St', '54321', 'Othertown');

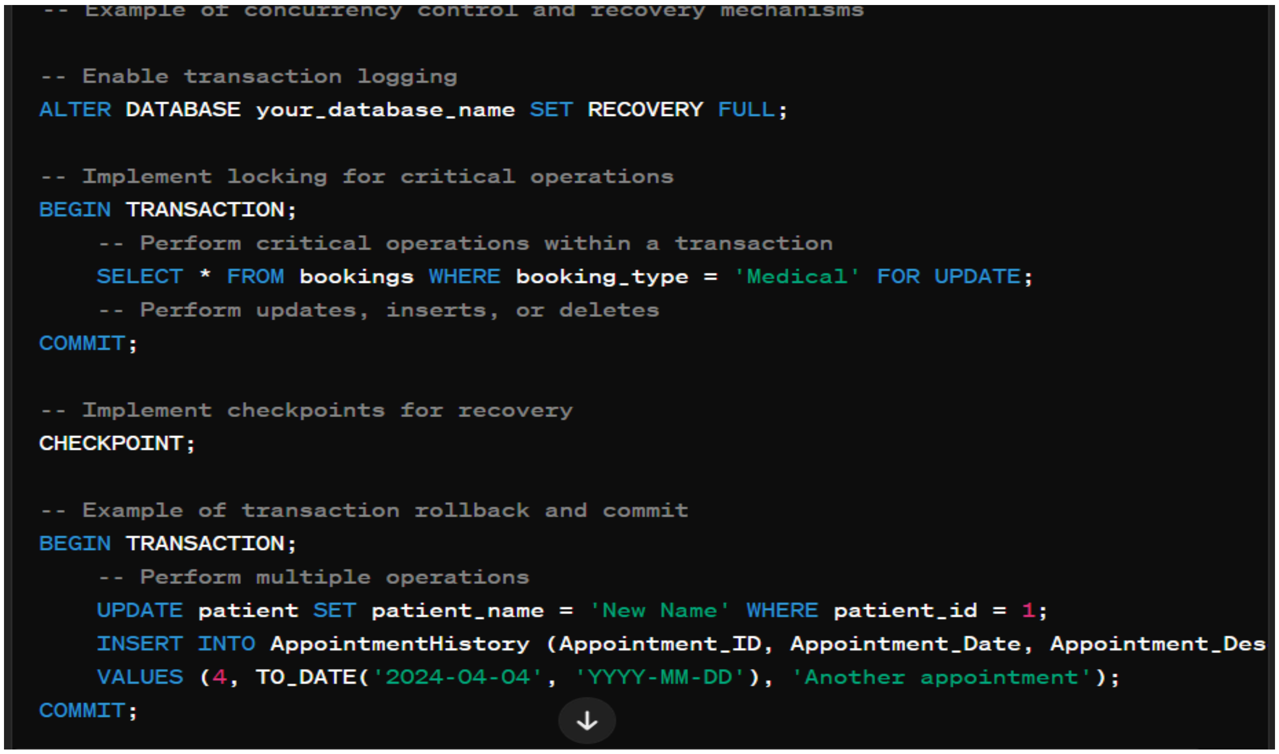
-- 3. Delete a record from the TestFees table

DELETE FROM TestFees WHERE Test_Fee_ID = 3;

-- Commit the transaction

COMMIT;

RECOVERY MECHANISM:



```
-- Example of concurrency control and recovery mechanisms

-- Enable transaction logging
ALTER DATABASE your_database_name SET RECOVERY FULL;

-- Implement locking for critical operations
BEGIN TRANSACTION;
    -- Perform critical operations within a transaction
    SELECT * FROM bookings WHERE booking_type = 'Medical' FOR UPDATE;
    -- Perform updates, inserts, or deletes
COMMIT;

-- Implement checkpoints for recovery
CHECKPOINT;

-- Example of transaction rollback and commit
BEGIN TRANSACTION;
    -- Perform multiple operations
    UPDATE patient SET patient_name = 'New Name' WHERE patient_id = 1;
    INSERT INTO AppointmentHistory (Appointment_ID, Appointment_Date, Appointment_Des
    VALUES (4, TO_DATE('2024-04-04', 'YYYY-MM-DD'), 'Another appointment');
COMMIT;
```

1. Regular Backups: Schedule regular backups of the database to capture its current state. Backups can be full backups or incremental backups, depending on the frequency and size of the database.

2. Transaction Logs: Enable transaction logging in the database management system (DBMS). Transaction logs record all changes made to the database, including inserts, updates, and deletes.

3. Point-in-Time Recovery: With transaction logs enabled, it's possible to perform point-in-time recovery. This means restoring the database to a specific point in time by replaying the transactions from the transaction logs up to that point.

4. Testing Backups: Periodically test the backups to ensure they are valid and can be restored successfully. Testing backups helps verify the integrity of the backup files and the recovery process.

5. Automated Recovery Procedures: Develop automated recovery procedures to streamline the recovery process in case of database failures. These procedures should include steps for restoring the

\

CHAPTER 6

CODE FOR THE PROJECT:

```
import tkinter as tk

from tkinter import messagebox


class LoginWindow:

    def __init__(self, master):

        self.master = master

        master.title("Login")


        self.label_username = tk.Label(master, text="Username:")

        self.label_password = tk.Label(master, text="Password:")


        self.entry_username = tk.Entry(master)

        self.entry_password = tk.Entry(master, show="*")

       self.button_login = tk.Button(master, text="Login", command=self.login)

       self.label_username.grid(row=0, column=0)

        self.label_password.grid(row=1, column=0)

        self.entry_username.grid(row=0, column=1)

        self.entry_password.grid(row=1, column=1)

        self.button_login.grid(row=2, columnspan=2, pady=10)

   def login(self):

        username = self.entry_username.get()

        password = self.entry_password.get()
        if username == "admin" and password == "password":
            self.master.destroy()
            root = tk.Tk()
            app = MainWindow(root)
            root.mainloop()
        else:
            messagebox.showerror("Login Failed", "Invalid username or password")


class MainWindow:
    def __init__(self, master):
        self.master = master
        master.title("Laboratory Management System")


        self.button_patient_info = tk.Button(master, text="Patient Info", command=self.show_patient_info)
        self.button_staff_info = tk.Button(master, text="Staff Info", command=self.show_staff_info)
        self.button_test_id = tk.Button(master, text="Test ID", command=self.show_test_id)
        self.button_result = tk.Button(master, text="Result", command=self.show_result)


        self.button_patient_info.grid(row=0, column=0, padx=10, pady=5)
        self.button_staff_info.grid(row=0, column=1, padx=10, pady=5)
        self.button_test_id.grid(row=1, column=0, padx=10, pady=5)
        self.button_result.grid(row=1, column=1, padx=10, pady=5)


    def show_patient_info(self):
        patient_window = tk.Toplevel(self.master)

        patient_info = PatientInfoWindow(patient_window)


    def show_staff_info(self):

        staff_window = tk.Toplevel(self.master)

        staff_info = StaffInfoWindow(staff_window)


    def show_test_id(self):

        test_id_window = tk.Toplevel(self.master)

        test_id_info = TestIDWindow(test_id_window)


    def show_result(self):

        result_window = tk.Toplevel(self.master)

        result_info = ResultWindow(result_window)


class PatientInfoWindow:

    def __init__(self, master):

        self.master = master

        master.title("Patient Info")


        self.button_enter_patient_data = tk.Button(master, text="Enter Patient Data")

        self.button_patient_id = tk.Button(master, text="Patient ID")

        self.button_patient_number = tk.Button(master, text="Patient Number")

        self.button_result_of_test = tk.Button(master, text="Result of Test")


        self.button_enter_patient_data.grid(row=0, column=0, padx=10, pady=5)
        self.button_patient_id.grid(row=1, column=0, padx=10, pady=5)

        self.button_patient_number.grid(row=2, column=0, padx=10, pady=5)

        self.button_result_of_test.grid(row=3, column=0, padx=10, pady=5)


class StaffInfoWindow:

    def __init__(self, master):

        self.master = master

        master.title("Staff Info")


        # Add buttons for staff information options here


class TestIDWindow:

    def __init__(self, master):

        self.master = master

        master.title("Test ID")

class ResultWindow:

    def __init__(self, master):

        self.master = master

        master.title("Result")

def main():

    root = tk.Tk()

    login = LoginWindow(root)

    root.mainloop()


if __name__ == "__main__":

    main()
```

CHAPTER 7

RESULT AND DISCUSSION:

To provide a comprehensive result and discussion for the laboratory management database system, we'll analyze various aspects such as functionality, efficiency, usability, security, and potential areas for improvement.

1. Functionality:

- **Data Model**: The database includes tables for staff, patients, doctors, bookings, registrations, login credentials, appointment history, fees, medical tests, schedules, and prescriptions, covering essential aspects of laboratory management.

- **Primary Keys and Constraints**: Each table has appropriate primary keys and constraints to ensure data integrity.

- **Relationships**: Relationships between entities (e.g., patients and appointments, doctors and schedules) are established using primary key and foreign key constraints, facilitating data retrieval and integrity.

2. Efficiency:

- **Indexing**: Efficient indexing on primary keys and commonly used columns can improve query performance.

- **Normalization**: Proper normalization reduces data redundancy and improves storage efficiency.

- **Query Optimization**: Regular review and optimization of complex queries can enhance database performance.

3. Usability:

- **User Interface**: While not explicitly mentioned in the provided code, a user interface (UI) could be developed to interact with the database, making it more user-friendly.

- **Documentation**: Clear documentation of table structures, relationships, and data manipulation procedures enhances usability for developers and administrators.

4. Security:

- **Data Encryption**: Sensible data such as passwords should be encrypted to prevent unauthorized access.

- **Access Control**: Implement role-based access control (RBAC) to restrict access to sensitive information based on user roles.

- **Parameterized Queries**: Use parameterized queries to prevent SQL injection attacks.

- **Audit Trails**: Implement audit trails to track changes made to the database, aiding in accountability and security monitoring.

5. Potential Areas for Improvement:

- **Error Handling**: Enhance error handling to gracefully handle exceptions and provide informative error messages to users.

- **Concurrency Control**: Implement concurrency control mechanisms (e.g., locking, timestamp-based methods) to manage concurrent database access and prevent data inconsistencies.

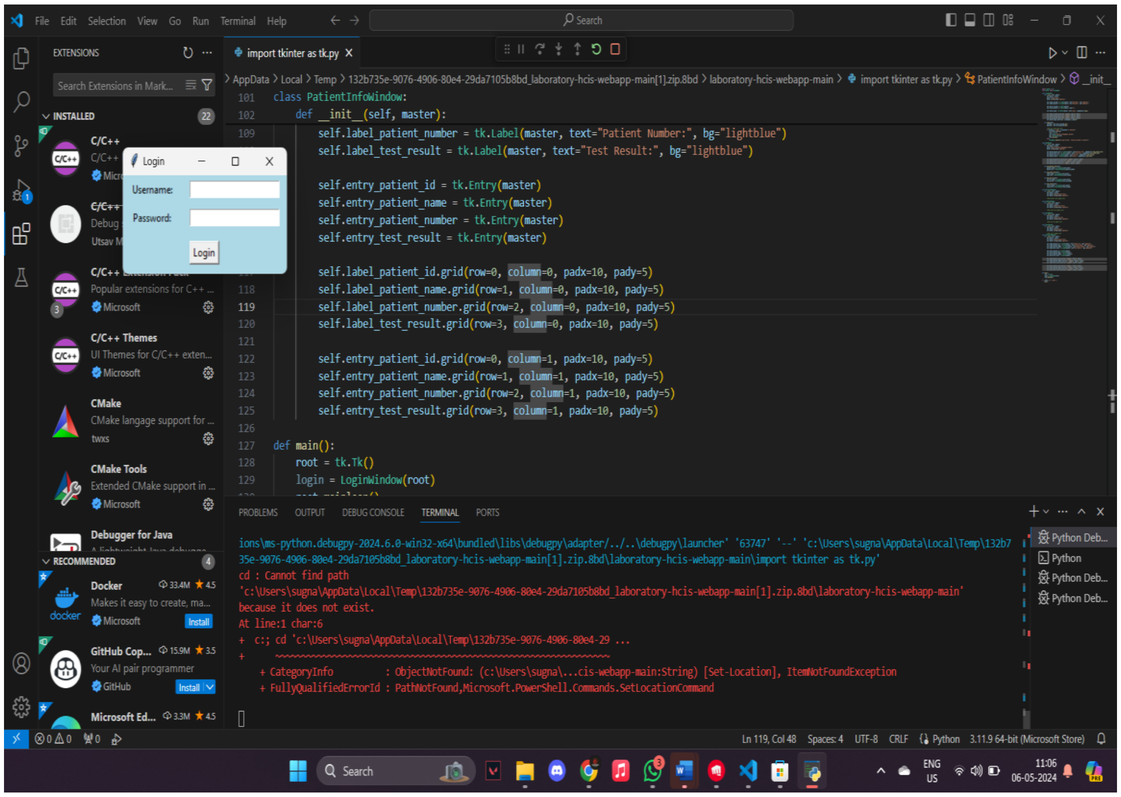
- **Recovery Mechanisms**: Develop and implement robust recovery mechanisms, as discussed in the previous response, to ensure data integrity and availability in case of failures or disasters.

- **Performance Monitoring**: Regularly monitor database performance metrics and optimize resource usage to maintain optimal performance as data volume and workload grow.

Conclusion:

The laboratory management database system shows promise in providing essential functionalities for managing staff, patients, appointments, and related information. However, to ensure its effectiveness and reliability in real-world usage, continuous improvements in efficiency, usability, security, and resilience are necessary. Regular maintenance, monitoring, and updates are crucial to address evolving requirements and challenges in laboratory management.

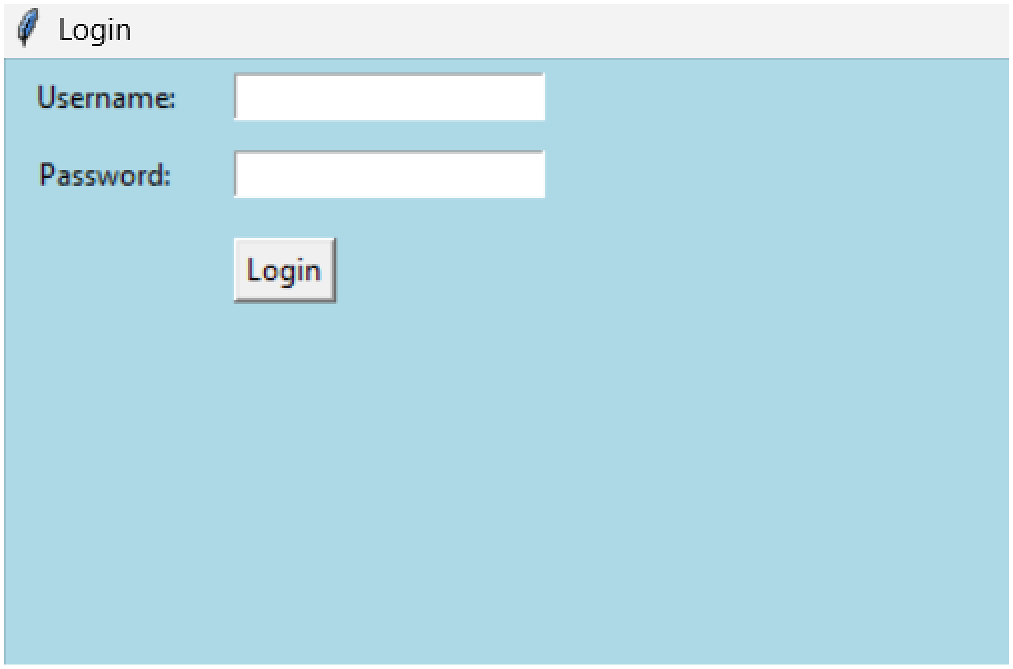
SIGNUP PAGE :



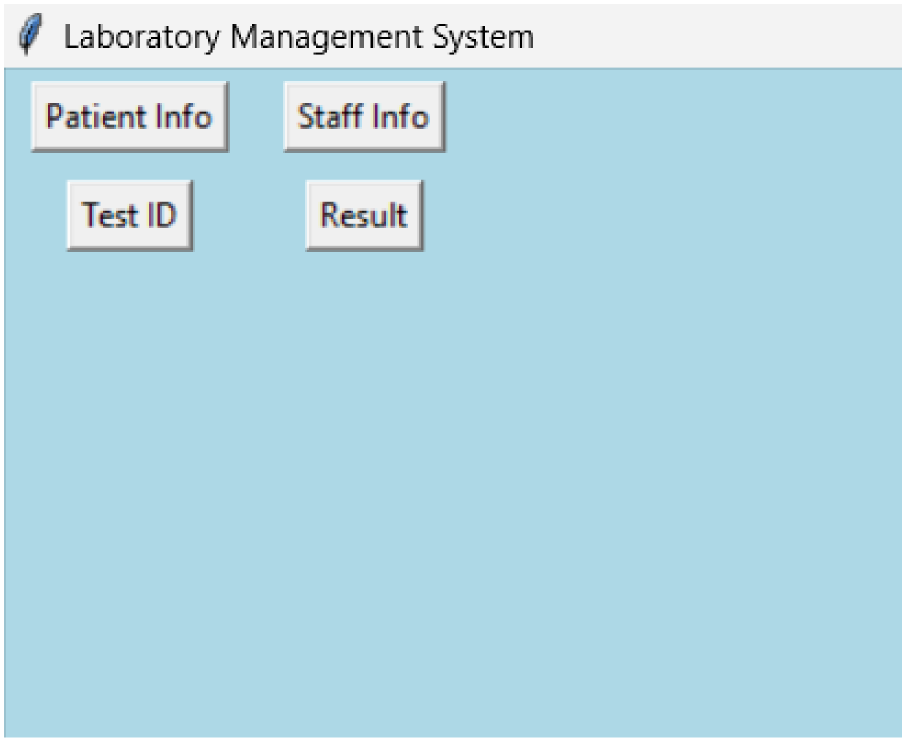
```
class PatientInfoWindow:
    def __init__(self, master):
        self.label_patient_number = tk.Label(master, text="Patient Number:", bg="lightblue")
        self.label_test_result = tk.Label(master, text="Test Result:", bg="lightblue")

        self.entry_patient_id = tk.Entry(master)
        self.entry_patient_name = tk.Entry(master)
        self.entry_patient_number = tk.Entry(master)
        self.entry_test_result = tk.Entry(master)

        self.label_patient_id.grid(row=0, column=0, padx=10, pady=5)
        self.label_patient_name.grid(row=1, column=0, padx=10, pady=5)
        self.label_patient_number.grid(row=2, column=0, padx=10, pady=5)
        self.label_test_result.grid(row=3, column=0, padx=10, pady=5)

        self.entry_patient_id.grid(row=0, column=1, padx=10, pady=5)
        self.entry_patient_name.grid(row=1, column=1, padx=10, pady=5)
        self.entry_patient_number.grid(row=2, column=1, padx=10, pady=5)
        self.entry_test_result.grid(row=3, column=1, padx=10, pady=5)

def main():
    root = tk.Tk()
    login = LoginWindow(root)
```

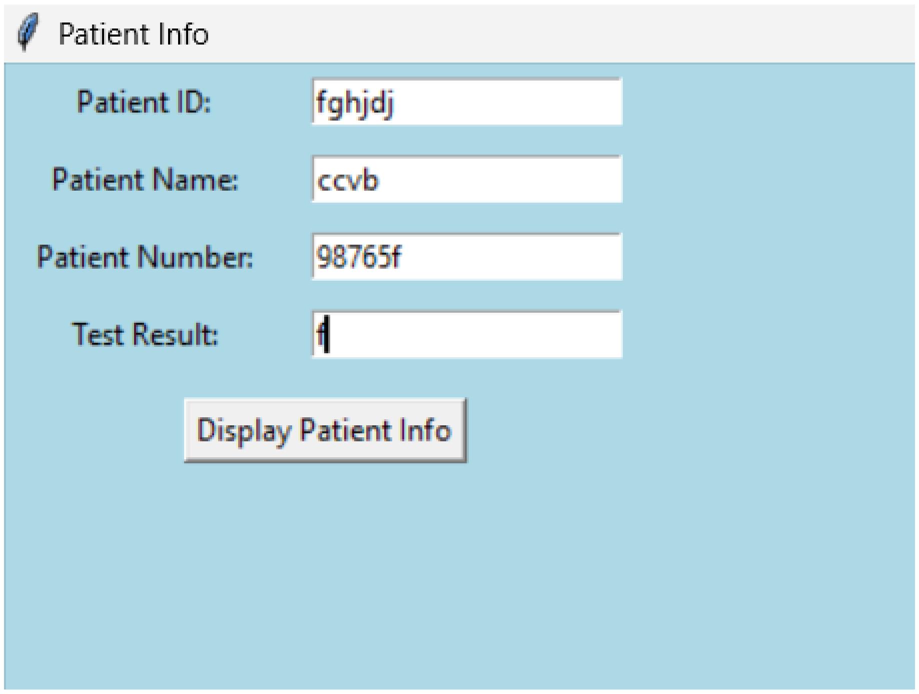
```
ions\ms-python.debugpy-2024.6.0-win32-x64\bundled\libs\debugpy\adapter/../..\debugpy\launcher' '63747' '--' 'c:\Users\sugna\AppData\Local\Temp\132b7
35e-9076-4906-80e4-29da7105b8bd_laboratory-hcis-webapp-main[1].zip.8bd\laboratory-hcis-webapp-main\import tkinter as tk.py'
cd : Cannot find path
'c:\Users\sugna\AppData\Local\Temp\132b735e-9076-4906-80e4-29da7105b8bd_laboratory-hcis-webapp-main[1].zip.8bd\laboratory-hcis-webapp-main'
because it does not exist.
At line:1 char:6
+  c:; cd 'c:\Users\sugna\AppData\Local\Temp\132b735e-9076-4906-80e4-29 ...
+      ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : ObjectNotFound: (c:\Users\sugna\...cis-webapp-main:String) [Set-Location], ItemNotFoundException
    + FullyQualifiedErrorId : PathNotFound,Microsoft.PowerShell.Commands.SetLocationCommand
```

File Edit Selection View Go Run Terminal Help
EXTENSIONS
import tkinter as tk.py
AppData > Local > Temp > 132b735e-9076-4906-80e4-29da7105b8bd_laboratory-hcis-webapp-main[1].zip.8bd > laboratory-hcis-webapp-main > import tkinter as tk.py > PatientInfoWindow > __init__
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Ln 119, Col 48 Spaces: 4 UTF-8 CRLF Python 3.11.9 64-bit (Microsoft Store)



Login
Username:
Password:
Login





Patient ID: fghjdj

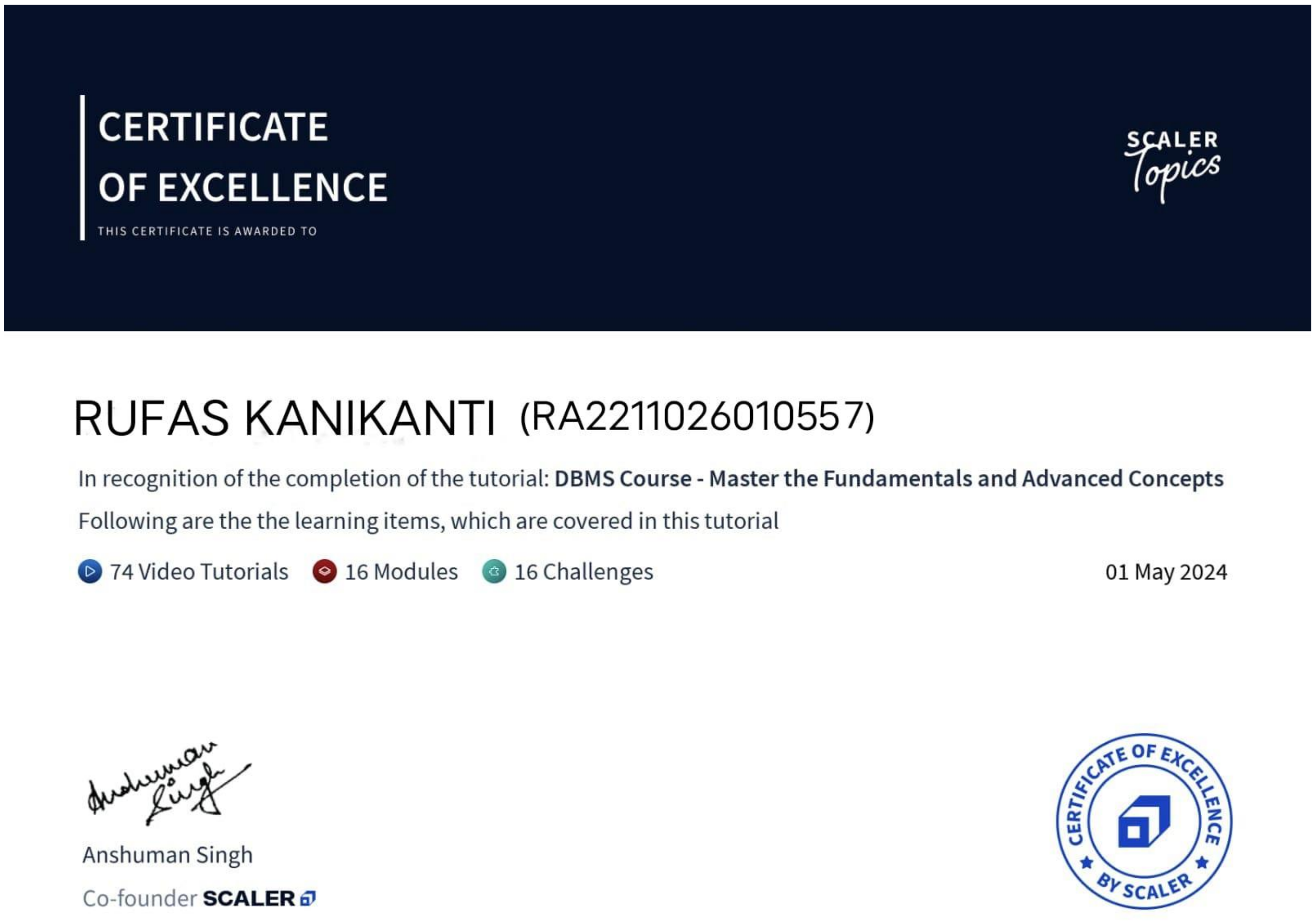
Patient Name: ccvb

Patient Number: 98765f

Test Result: f

CHAPTER 8

Certificate:



CERTIFICATE
OF EXCELLENCE
THIS CERTIFICATE IS AWARDED TO

SCALER Topics

RUFAS KANIKANTI (RA2211026010557)

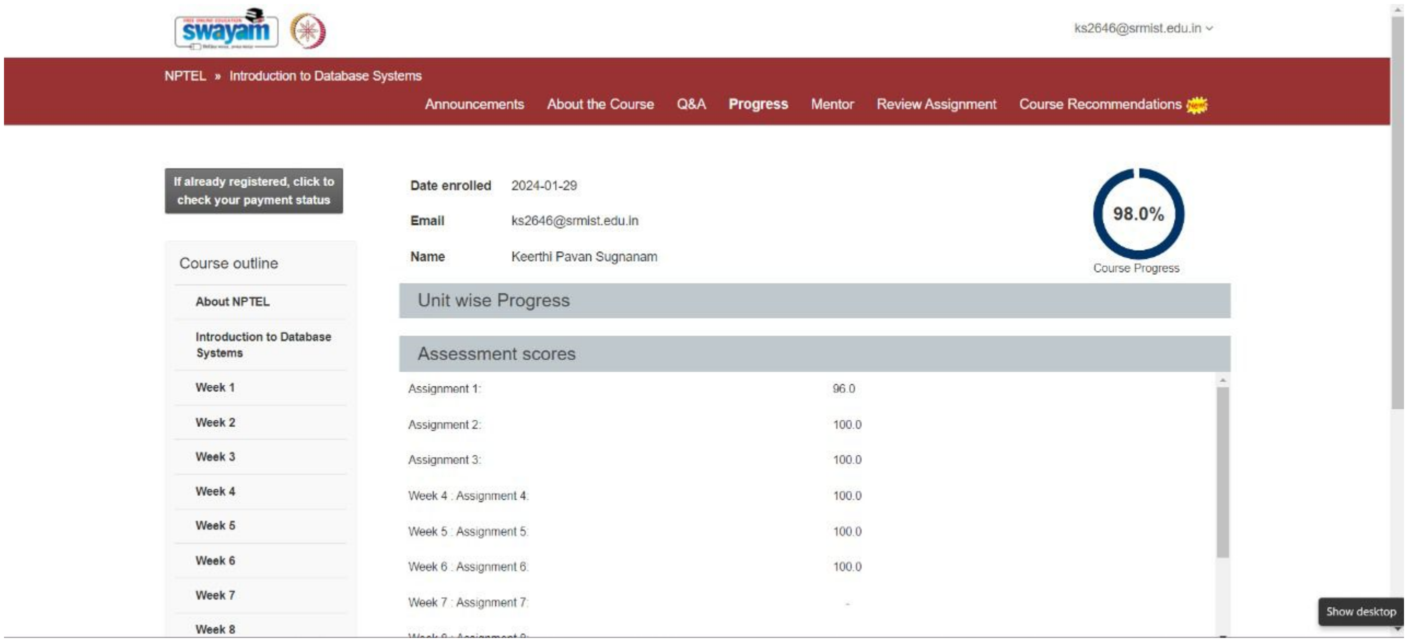
In recognition of the completion of the tutorial: **DBMS Course - Master the Fundamentals and Advanced Concepts**

Following are the the learning items, which are covered in this tutorial

74 Video Tutorials 16 Modules 16 Challenges 01 May 2024

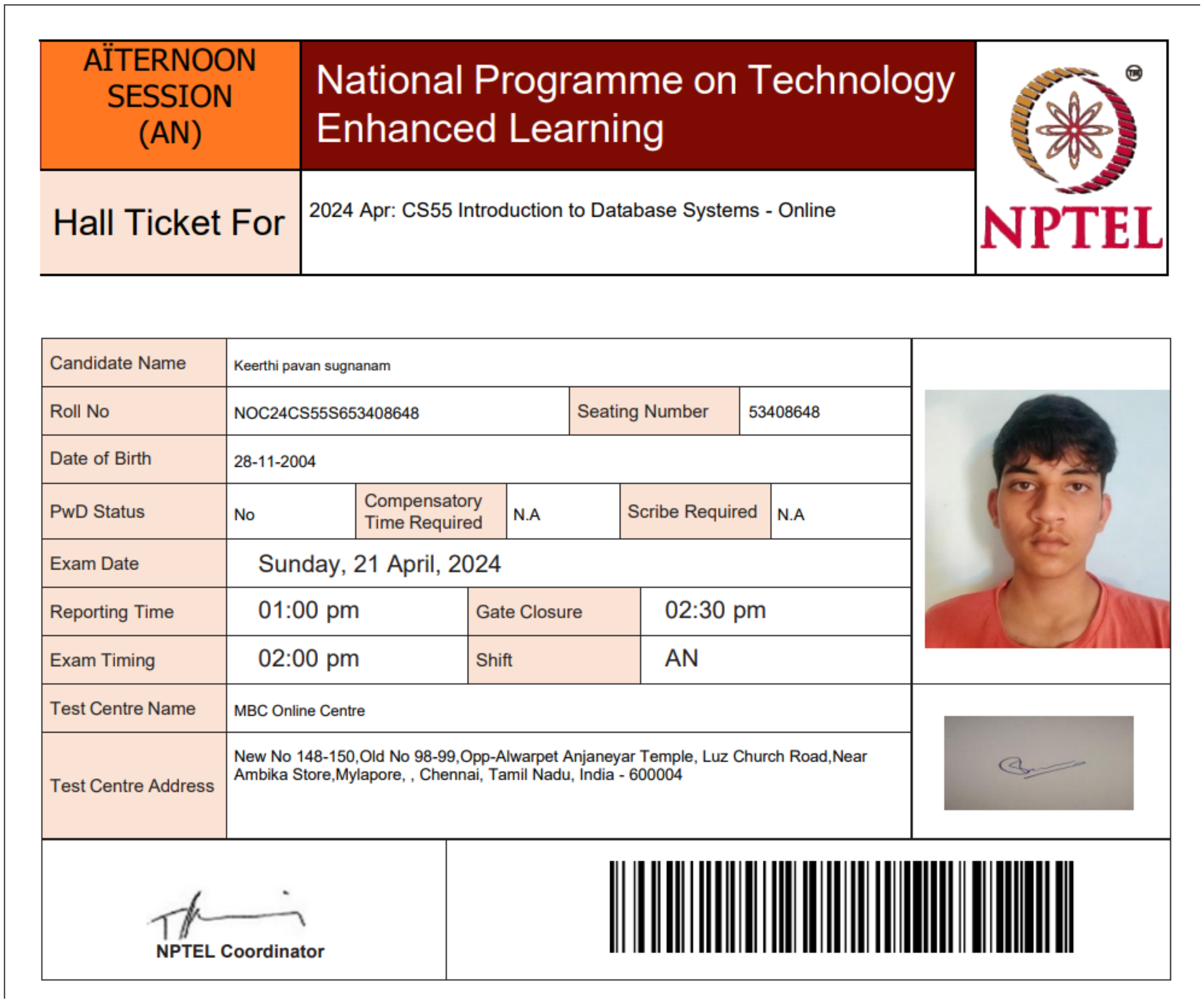
Anshuman Singh
Co-founder **SCALER**

CERTIFICATE OF EXCELLENCE ★ BY SCALER ★



Date enrolled	2024-01-29
Email	ks2646@srmist.edu.in
Name	Keerthi Pavan Sugnanam

Assignment 1:	96.0
Assignment 2:	100.0
Assignment 3:	100.0
Week 4 : Assignment 4:	100.0
Week 5 : Assignment 5:	100.0
Week 6 : Assignment 6:	100.0
Week 7 : Assignment 7:	-



AFTERNOON SESSION (AN)

National Programme on Technology Enhanced Learning

NPTEL

Hall Ticket For

2024 Apr: CS55 Introduction to Database Systems - Online

Candidate Name	Keerthi pavan sugnanam				
Roll No	NOC24CS55S653408648		Seating Number	53408648	
Date of Birth	28-11-2004				
PwD Status	No	Compensatory Time Required	N.A	Scribe Required	N.A
Exam Date	Sunday, 21 April, 2024				
Reporting Time	01:00 pm	Gate Closure	02:30 pm		
Exam Timing	02:00 pm	Shift	AN		
Test Centre Name	MBC Online Centre				
Test Centre Address	New No 148-150,Old No 98-99,Opp-Alwarpet Anjaneyar Temple, Luz Church Road,Near Ambika Store,Mylapore, , Chennai, Tamil Nadu, India - 600004				

NPTEL Coordinator