National Institute Of Business Management

Higher National Diploma In Software Engineering

Advanced Database Management System

Assessment 2

SUBMITTED BY

MAHNDSE232F -025 G.A.DILMI

MAHNDSE232F-035 A.P.S.I.KUMARA

MAHNDSE232F-028 H.A.C.PRABOD

Date of Submission : 05-03-2024

Acknowledgement

We would like to express our sincere appreciation to everyone who contributed to the successful completion of this assignment. First and foremost, we extend our gratitude to our lecturer, MS Kaushalya Dissanayake, for providing us with the opportunity to explore and deepen our knowledge of Oracle database management and PL/SQL programming. We would also like to thank our fellow group members for their collaboration and shared commitment to the project. Their insights and hard work greatly enriched the final deliveries. Additionally, we are thankful for the support and resources provided by our institution, which made this assignment possible. Last but not least, we appreciate the patience and understanding of our friends and family during the intense work on this project. Their encouragement and support were invaluable throughout this journey.

Table Of Contents

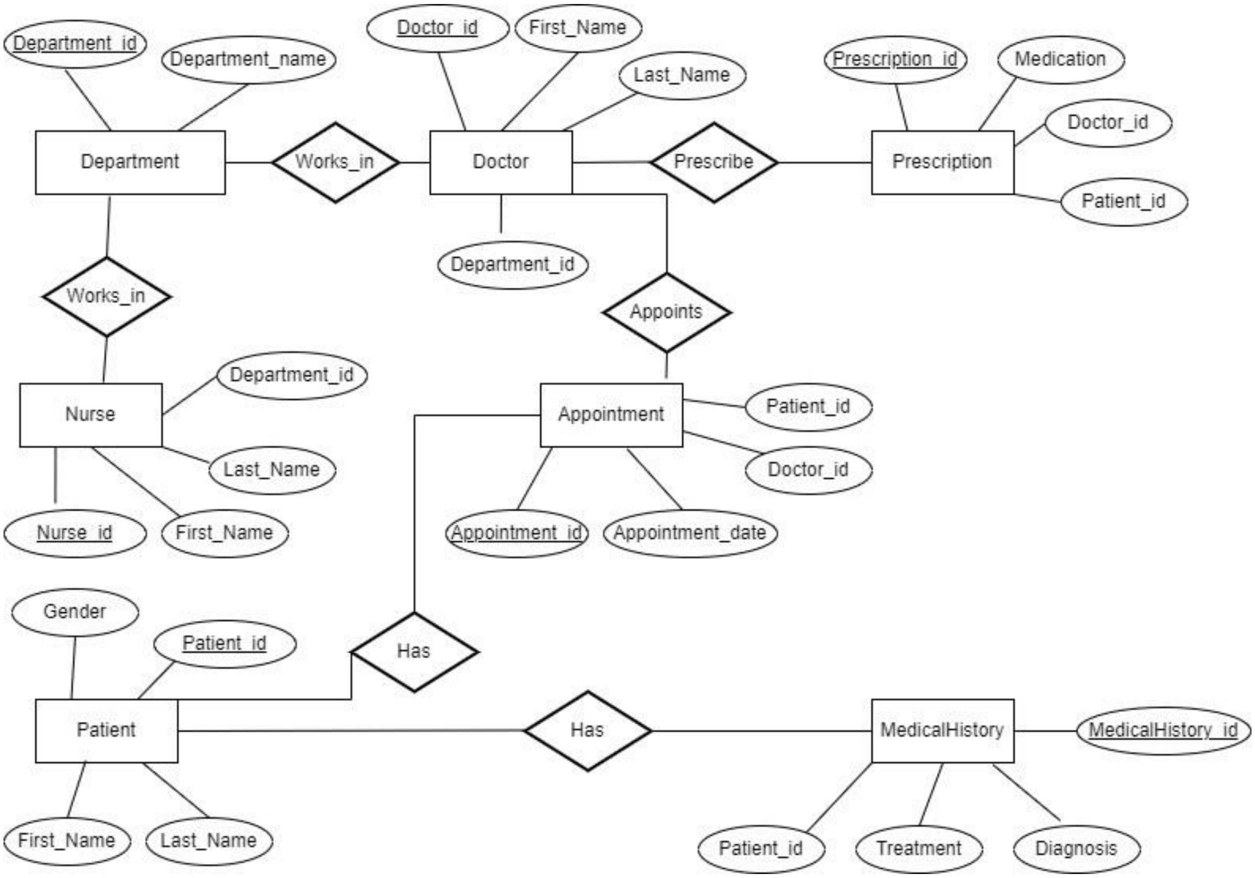
Table Of FIGURES

Chapter 1 : Introduction

1.1. Background

The Hospital Management System project aims to streamline healthcare processes by efficiently managing patient records, appointments, medical histories, prescriptions, and staff information. Through a meticulously designed database schema, tables for departments, doctors, nurses, patients, medical histories, prescriptions, and appointments are created, ensuring data integrity and efficient data retrieval. The implementation is based on Oracle Server, facilitating various database operations such as querying data, joining tables, and utilizing stored procedures and views for optimized task execution and data presentation. To ensure data security, user authentication and permissions are implemented, while indexing and backup strategies are employed for database optimization and recovery. With this robust system in place, healthcare professionals can effectively manage patient care and administrative tasks, ultimately enhancing the overall efficiency and quality of healthcare delivery.

1.2.ER Diagram



1.3 Database Design and Normalization

Our hospital management system database design adheres to the principles of data normalization to ensure data integrity, minimize redundancy, and optimize performance. Data normalization involves organizing data into tables and defining relationships between them to eliminate data anomalies and inconsistencies.

1. First Normal Form (1NF)

Each table in our database is in the First Normal Form, ensuring atomic attributes and absence of repeating groups. For instance, the Patient table contains attributes such as PatientID, FirstName, LastName, DateOfBirth, and Gender, all of which are atomic and indivisible.

2. Second Normal Form (2NF) and Third Normal Form (3NF)

Our database schema complies with the principles of the Second and Third Normal Forms. Non-key attributes are functionally dependent on the entire primary key, and there are no transitive dependencies between attributes. For example, in the MedicalHistory table, attributes such as MedicalHistoryID, PatientID, Diagnosis, and Treatment are stored without redundancy, ensuring data consistency and integrity.

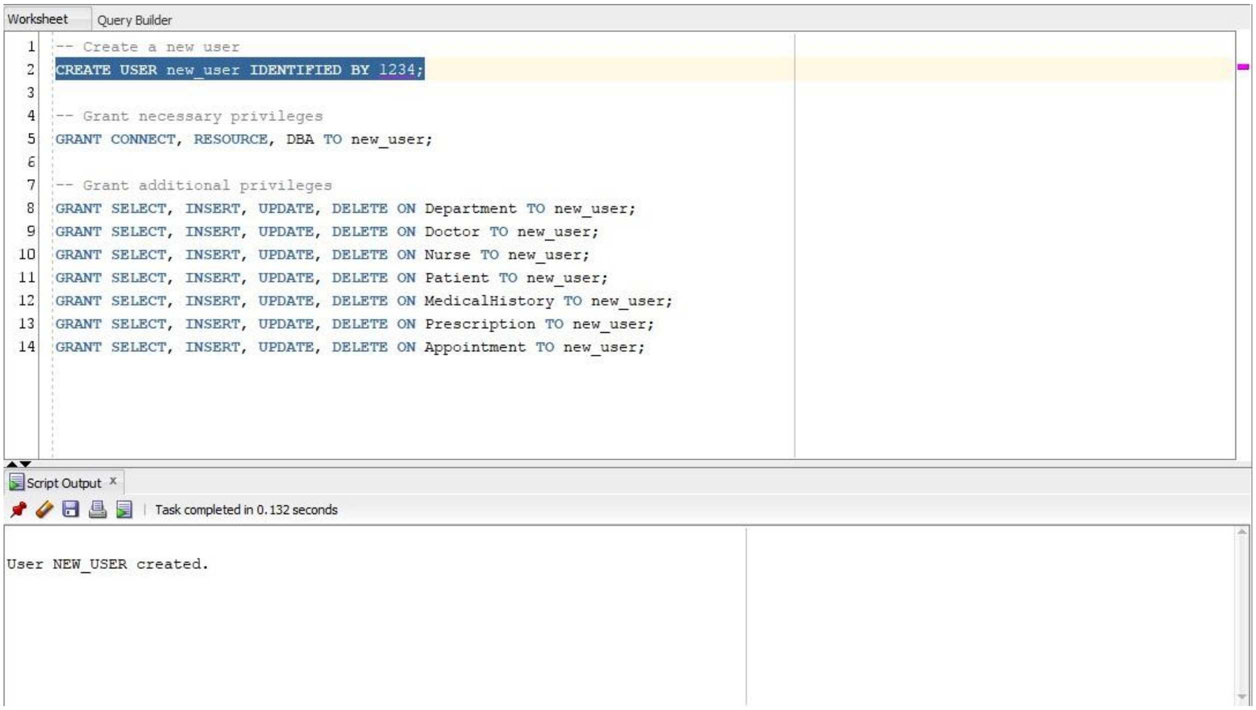
3. Additional Considerations

- Proper relationships between tables are established using foreign key constraints to maintain referential integrity.
- Data validation checks and constraints are implemented at the database level to prevent insertion of invalid or inconsistent data.
- Indexes are created on frequently queried columns to enhance query performance and optimize data retrieval.
- The following SQL code demonstrates the implementation of the aforementioned principles in our database design:
- Tables creation and data insertion SQL statements are provided according to our code.
- Additional SQL statements, such as stored procedures, views, indexes, and user management, are also implemented in alignment with the principles of data normalization and omitted for brevity.

Chapter 2 : Questions And Answers

1) Create a new user with assigning necessary privileges [Should provide explanation of user].

The SQL code creates a new user named new_user with the password 1234. It grants this user the CONNECT privilege, allowing database connections. Additionally, it provides the RESOURCE privilege for basic schema operations. Furthermore, it grants the DBA privilege, granting administrative control over the database.



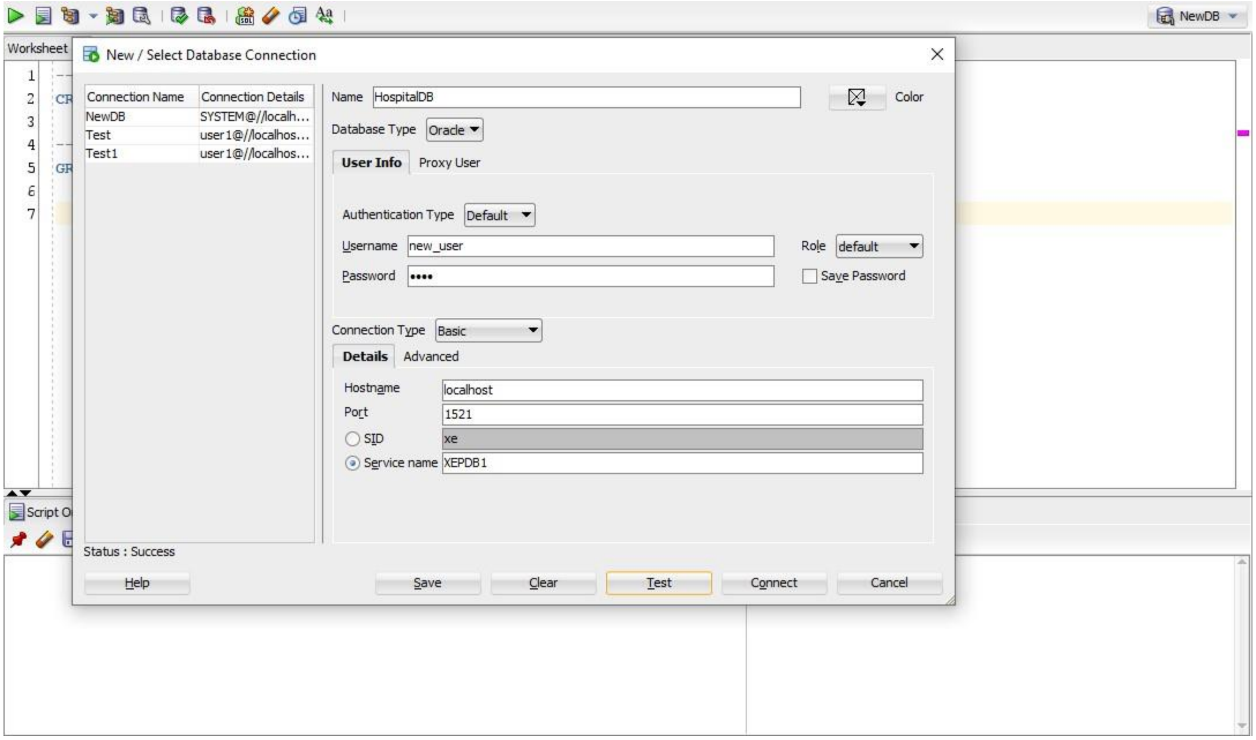
```
-- Create a new user
CREATE USER new_user IDENTIFIED BY 1234;

-- Grant necessary privileges
GRANT CONNECT, RESOURCE, DBA TO new_user;

-- Grant additional privileges
GRANT SELECT, INSERT, UPDATE, DELETE ON Department TO new_user;
GRANT SELECT, INSERT, UPDATE, DELETE ON Doctor TO new_user;
GRANT SELECT, INSERT, UPDATE, DELETE ON Nurse TO new_user;
GRANT SELECT, INSERT, UPDATE, DELETE ON Patient TO new_user;
GRANT SELECT, INSERT, UPDATE, DELETE ON MedicalHistory TO new_user;
GRANT SELECT, INSERT, UPDATE, DELETE ON Prescription TO new_user;
GRANT SELECT, INSERT, UPDATE, DELETE ON Appointment TO new_user;
```

Task completed in 0.132 seconds

```
User NEW_USER created.
```

Figure 1: Create a new user with assigning necessary privileges

2) Create a new database connection for above created user.



Connection Name	Connection Details
NewDB	SYSTEM@//localh...
Test	user1@//localhos...
Test1	user1@//localhos...

Name HospitalDB

Database Type Oracle

User Info | Proxy User

Authentication Type Default

Username new_user Role default

Password •••• ☐ Save Password

Connection Type Basic

Details | Advanced

Hostname localhost

Port 1521

○ SID xe

● Service name XEPDB1

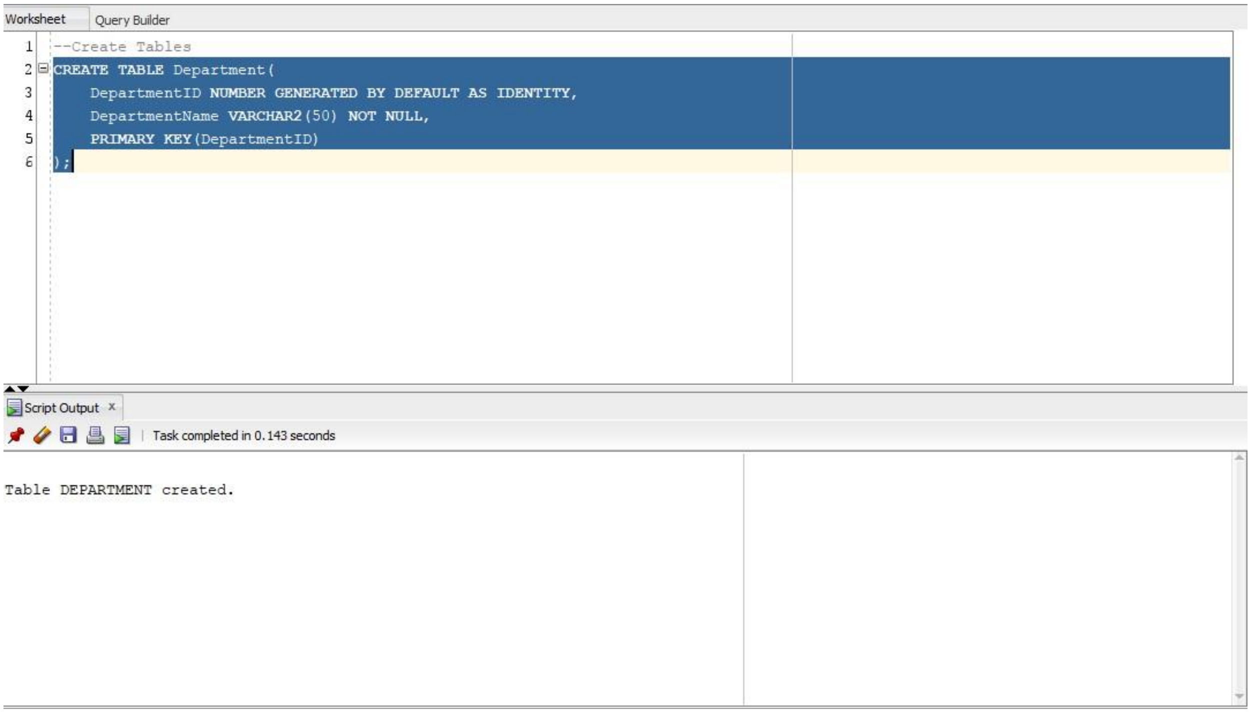
Status : Success

Figure 2: Create a new database connection for above created user

3) The tables should create with considering following features.

- Primary key and foreign key constraints
- Check Constraint
- Not Null Constraint

The Department table is defined to store department-related information. It includes attributes such as DepartmentID as a unique identifier, DepartmentName for naming departments, and a primary key constraint on DepartmentID to ensure data integrity.



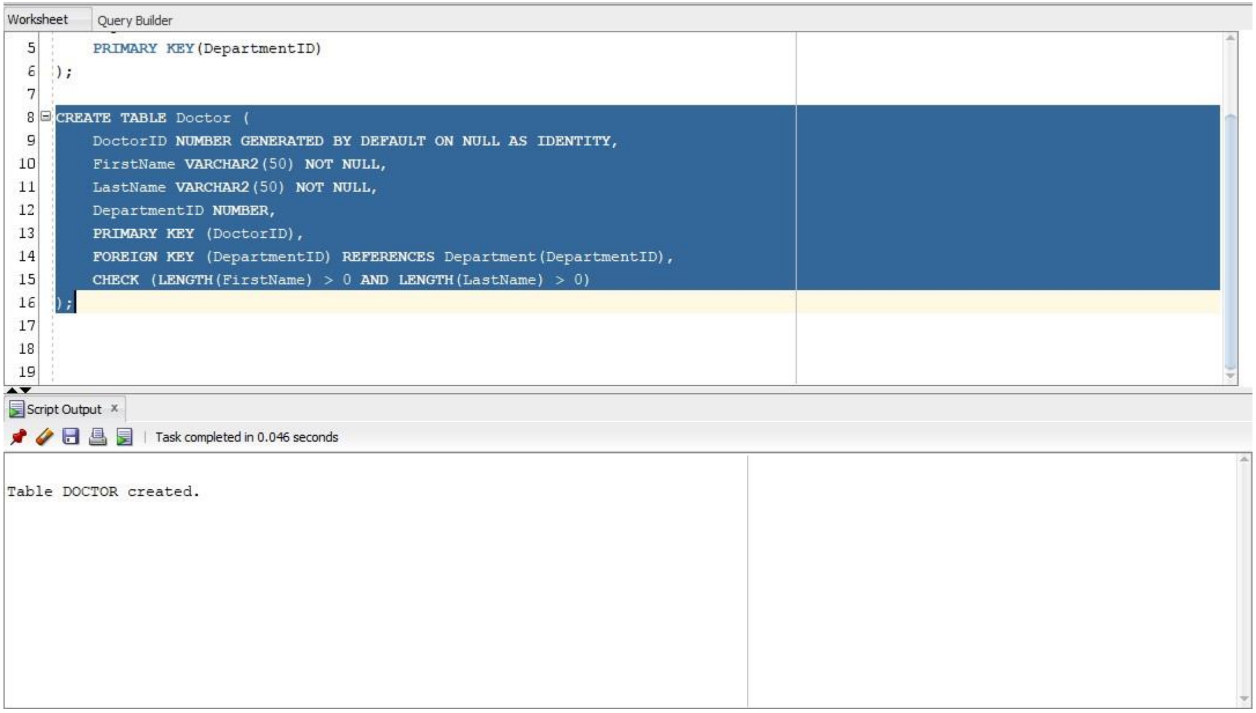
```
--Create Tables
CREATE TABLE Department(
    DepartmentID NUMBER GENERATED BY DEFAULT AS IDENTITY,
    DepartmentName VARCHAR2(50) NOT NULL,
    PRIMARY KEY(DepartmentID)
);
```

Task completed in 0.143 seconds

```
Table DEPARTMENT created.
```

Figure 3: Create a Department Table

The Doctor table is created to manage doctor details. It includes fields like DoctorID for unique identification, FirstName, and LastName for doctor names. Additionally, it contains a DepartmentID field to establish relationships with departments, enforced by a foreign key constraint referencing the Department table. Furthermore, the table has a check constraint to ensure that both first and last names have non-zero lengths.



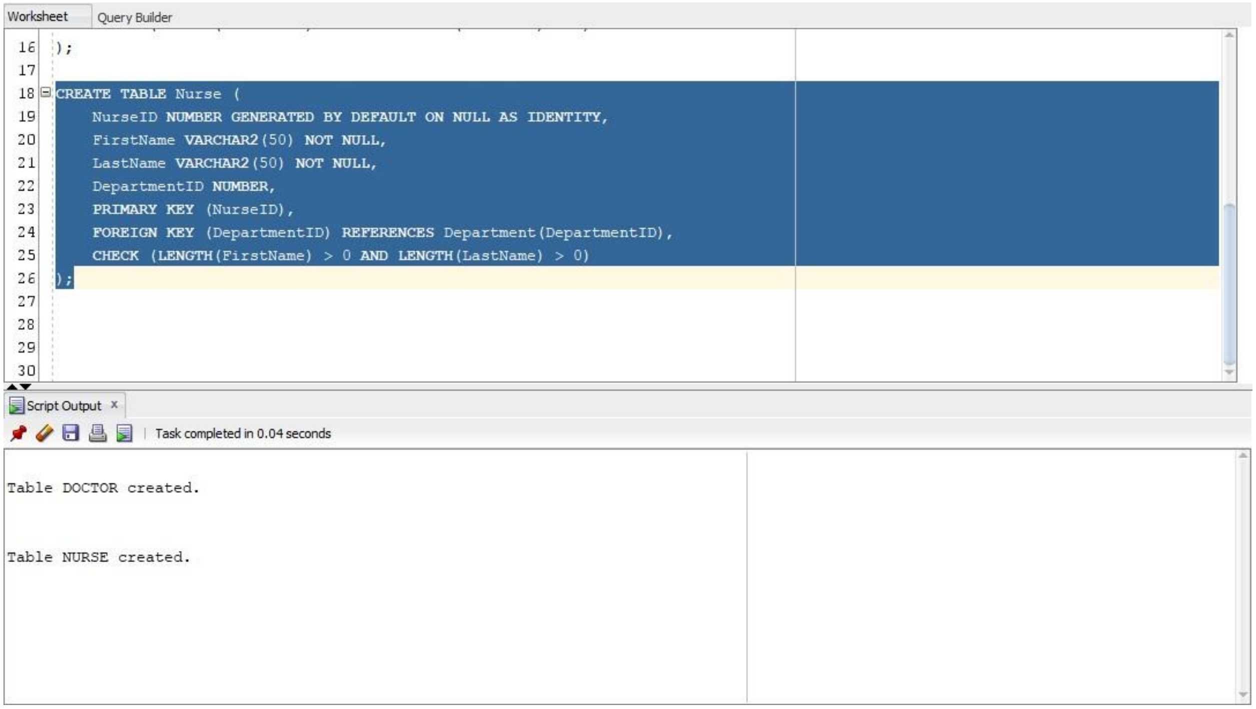
```
    PRIMARY KEY(DepartmentID)
);

CREATE TABLE Doctor (
    DoctorID NUMBER GENERATED BY DEFAULT ON NULL AS IDENTITY,
    FirstName VARCHAR2(50) NOT NULL,
    LastName VARCHAR2(50) NOT NULL,
    DepartmentID NUMBER,
    PRIMARY KEY (DoctorID),
    FOREIGN KEY (DepartmentID) REFERENCES Department(DepartmentID),
    CHECK (LENGTH(FirstName) > 0 AND LENGTH(LastName) > 0)
);
```

Task completed in 0.046 seconds

```
Table DOCTOR created.
```

Figure 4: Create a Doctor Table

The Nurse table follows a similar structure to the Doctor table, storing nurse information such as NurseID, FirstName, LastName, and DepartmentID. Similar to the doctor table, it includes a foreign key constraint to link nurses with departments and a check constraint for name length validation.



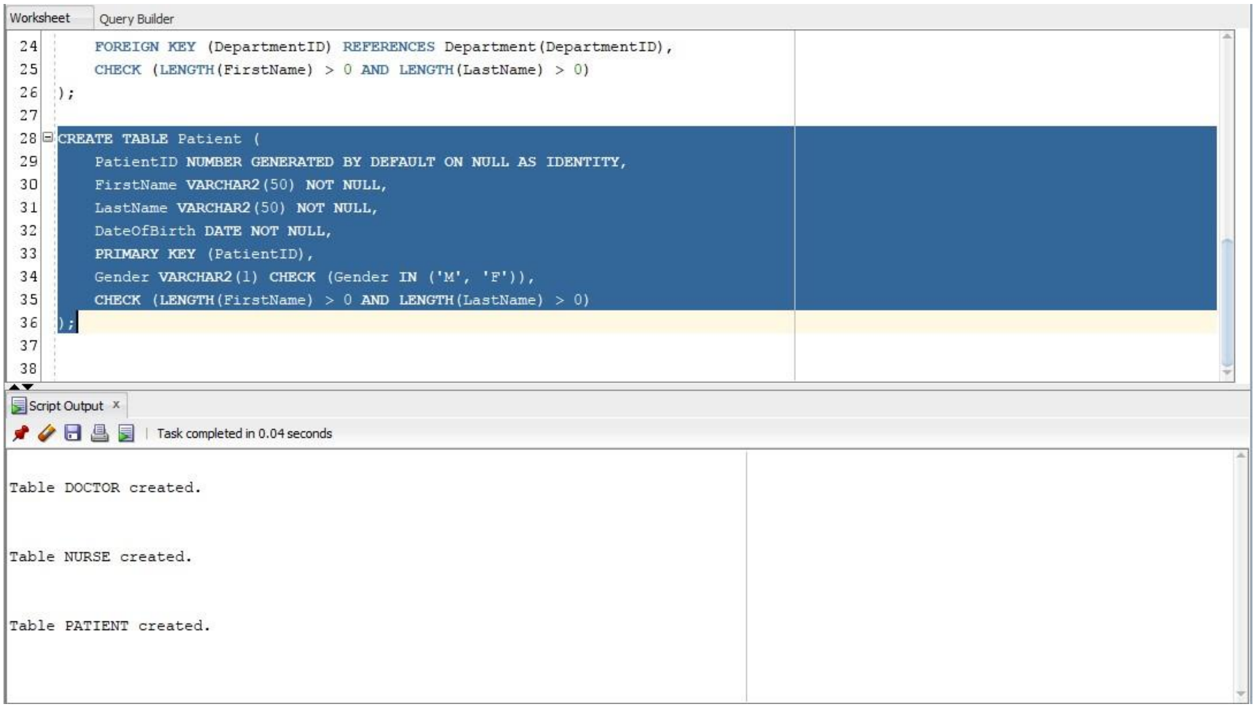
```
);

CREATE TABLE Nurse (
    NurseID NUMBER GENERATED BY DEFAULT ON NULL AS IDENTITY,
    FirstName VARCHAR2(50) NOT NULL,
    LastName VARCHAR2(50) NOT NULL,
    DepartmentID NUMBER,
    PRIMARY KEY (NurseID),
    FOREIGN KEY (DepartmentID) REFERENCES Department(DepartmentID),
    CHECK (LENGTH(FirstName) > 0 AND LENGTH(LastName) > 0)
);
```

Task completed in 0.04 seconds

```
Table DOCTOR created.


Table NURSE created.
```

Figure 5: Create a Nurse Table

The Patient table is designed to hold patient records. It includes fields like PatientID, FirstName, LastName, DateOfBirth, and Gender, with the latter constrained to accept only 'M' or 'F' values. This table also enforces constraints on name length and defines PatientID as the primary key.



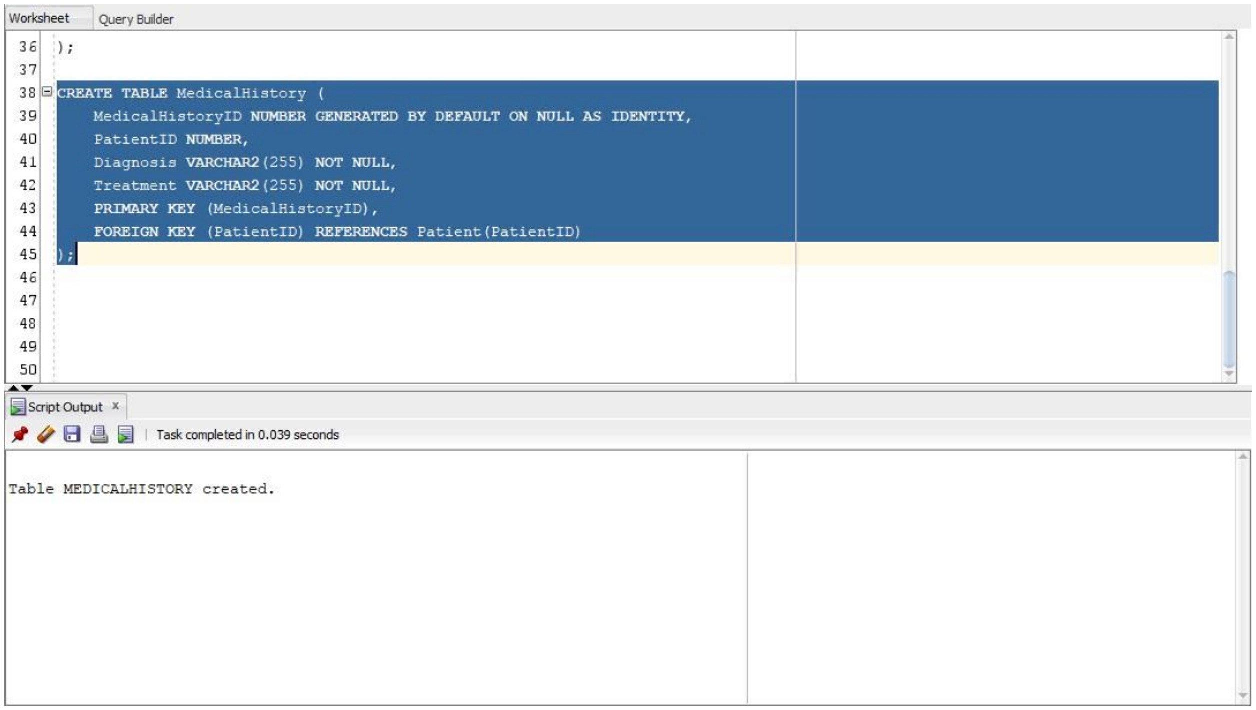
```
    FOREIGN KEY (DepartmentID) REFERENCES Department(DepartmentID),
    CHECK (LENGTH(FirstName) > 0 AND LENGTH(LastName) > 0)
);

CREATE TABLE Patient (
    PatientID NUMBER GENERATED BY DEFAULT ON NULL AS IDENTITY,
    FirstName VARCHAR2(50) NOT NULL,
    LastName VARCHAR2(50) NOT NULL,
    DateOfBirth DATE NOT NULL,
    PRIMARY KEY (PatientID),
    Gender VARCHAR2(1) CHECK (Gender IN ('M', 'F')),
    CHECK (LENGTH(FirstName) > 0 AND LENGTH(LastName) > 0)
);
```

Task completed in 0.04 seconds

```
Table DOCTOR created.


Table NURSE created.


Table PATIENT created.
```

Figure 6: Create a Patient Table

Moving on, the MedicalHistory table is created to track patient medical records. It consists of MedicalHistoryID as the primary key, PatientID as a foreign key referencing patients, Diagnosis, and Treatment fields for diagnosis and treatment details.



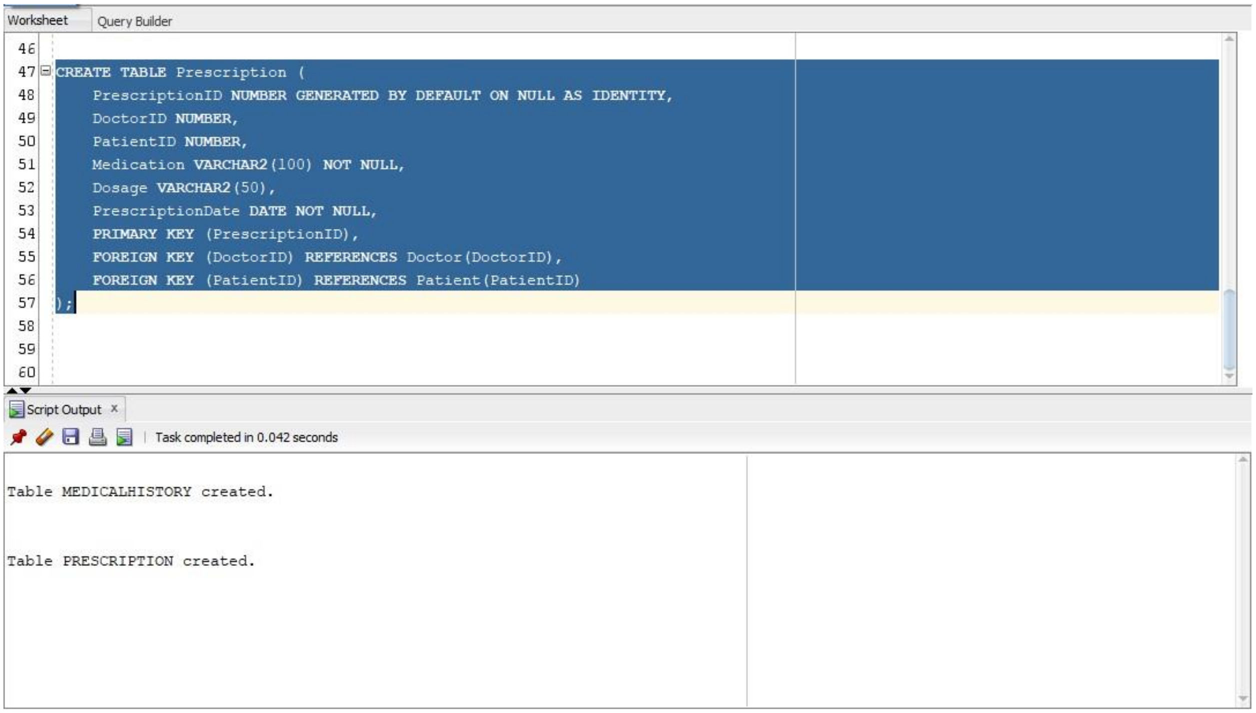
```
);

CREATE TABLE MedicalHistory (
    MedicalHistoryID NUMBER GENERATED BY DEFAULT ON NULL AS IDENTITY,
    PatientID NUMBER,
    Diagnosis VARCHAR2(255) NOT NULL,
    Treatment VARCHAR2(255) NOT NULL,
    PRIMARY KEY (MedicalHistoryID),
    FOREIGN KEY (PatientID) REFERENCES Patient(PatientID)
);
```

Task completed in 0.039 seconds

```
Table MEDICALHISTORY created.
```

Figure 7: Create a MedicalHistory Table

The Prescription table manages prescription details issued by doctors to patients. It contains fields like PrescriptionID, DoctorID, PatientID, Medication, Dosage, and PrescriptionDate. Foreign key constraints are applied to DoctorID and PatientID, referencing the respective tables.



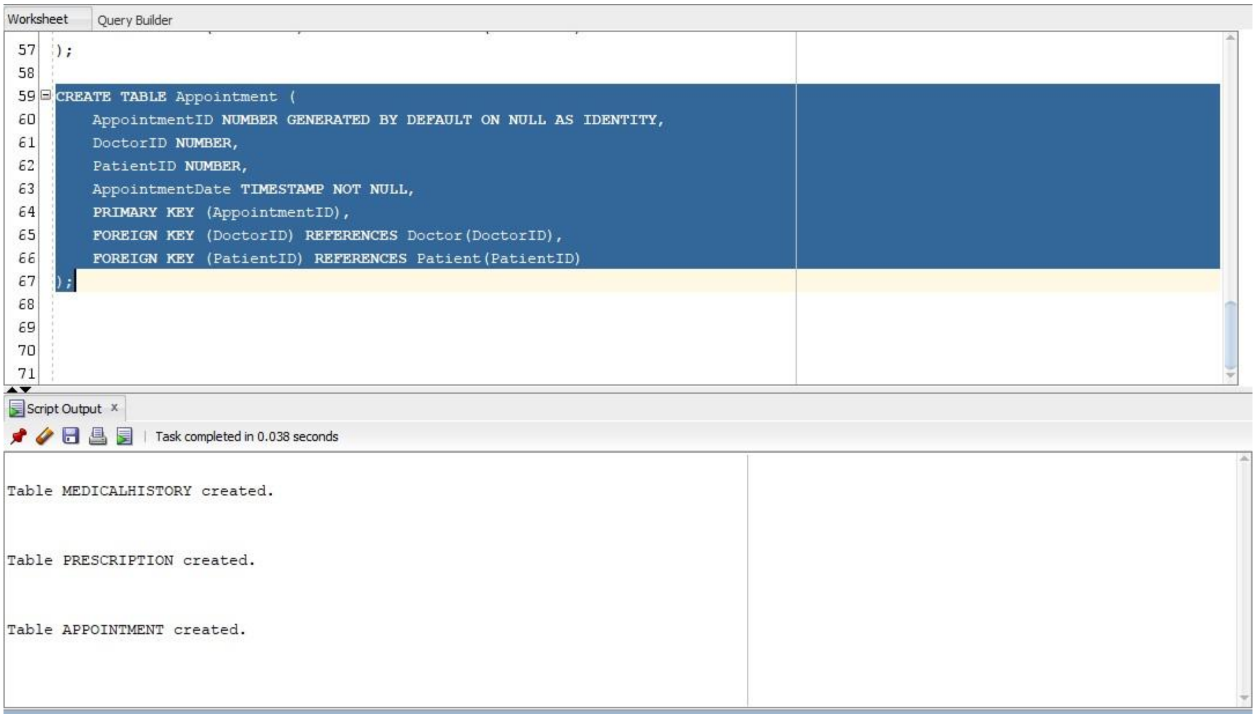
```
CREATE TABLE Prescription (
    PrescriptionID NUMBER GENERATED BY DEFAULT ON NULL AS IDENTITY,
    DoctorID NUMBER,
    PatientID NUMBER,
    Medication VARCHAR2(100) NOT NULL,
    Dosage VARCHAR2(50),
    PrescriptionDate DATE NOT NULL,
    PRIMARY KEY (PrescriptionID),
    FOREIGN KEY (DoctorID) REFERENCES Doctor(DoctorID),
    FOREIGN KEY (PatientID) REFERENCES Patient(PatientID)
);
```

Task completed in 0.042 seconds

```
Table MEDICALHISTORY created.


Table PRESCRIPTION created.
```

Figure 8: Create a Prescription Table

the Appointment table is established to record appointments between doctors and patients. It includes fields such as AppointmentID, DoctorID, PatientID, and AppointmentDate, with foreign key constraints referencing the Doctor and Patient tables



```
);

CREATE TABLE Appointment (
    AppointmentID NUMBER GENERATED BY DEFAULT ON NULL AS IDENTITY,
    DoctorID NUMBER,
    PatientID NUMBER,
    AppointmentDate TIMESTAMP NOT NULL,
    PRIMARY KEY (AppointmentID),
    FOREIGN KEY (DoctorID) REFERENCES Doctor(DoctorID),
    FOREIGN KEY (PatientID) REFERENCES Patient(PatientID)
);
```

Task completed in 0.038 seconds

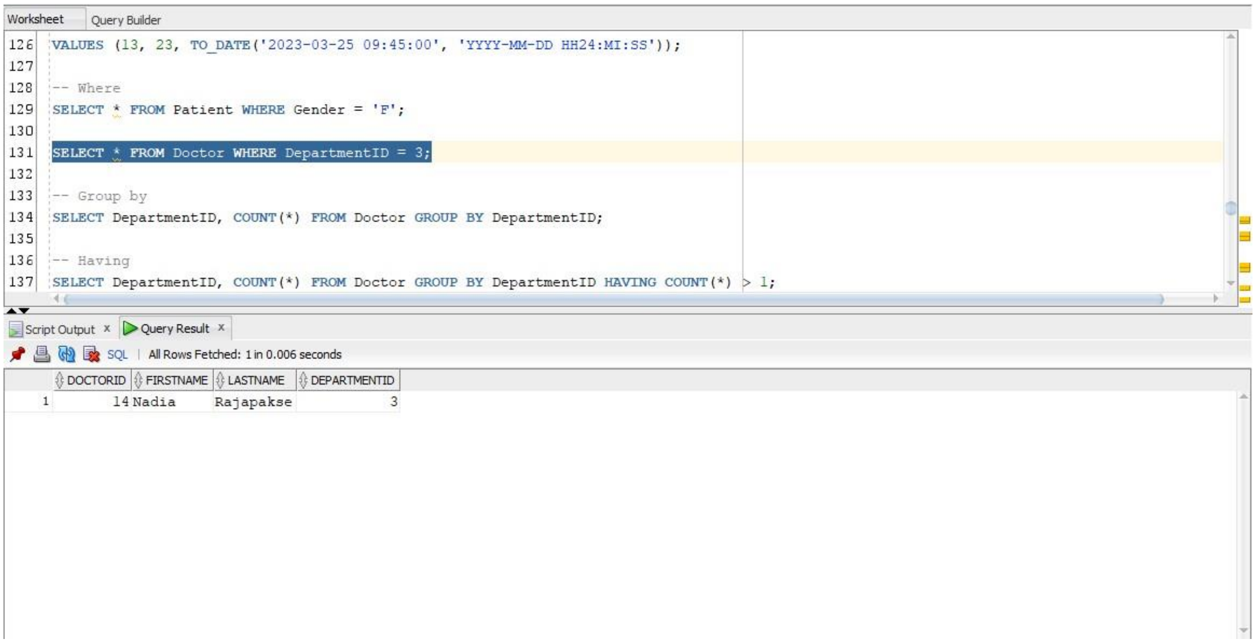
```
Table MEDICALHISTORY created.


Table PRESCRIPTION created.


Table APPOINTMENT created.
```

Figure 9: Create a Appointment Table

4) Write 20 DQL [Data Query Language] Command with covering following criteria.

- Where

The first query retrieves doctor records with a department ID of 3, specifying the department for which doctor information is required. The second query selects all patient records where the gender is 'F', filtering the Patient table based on gender. Both queries employ the WHERE clause to refine the results based on specific conditions.

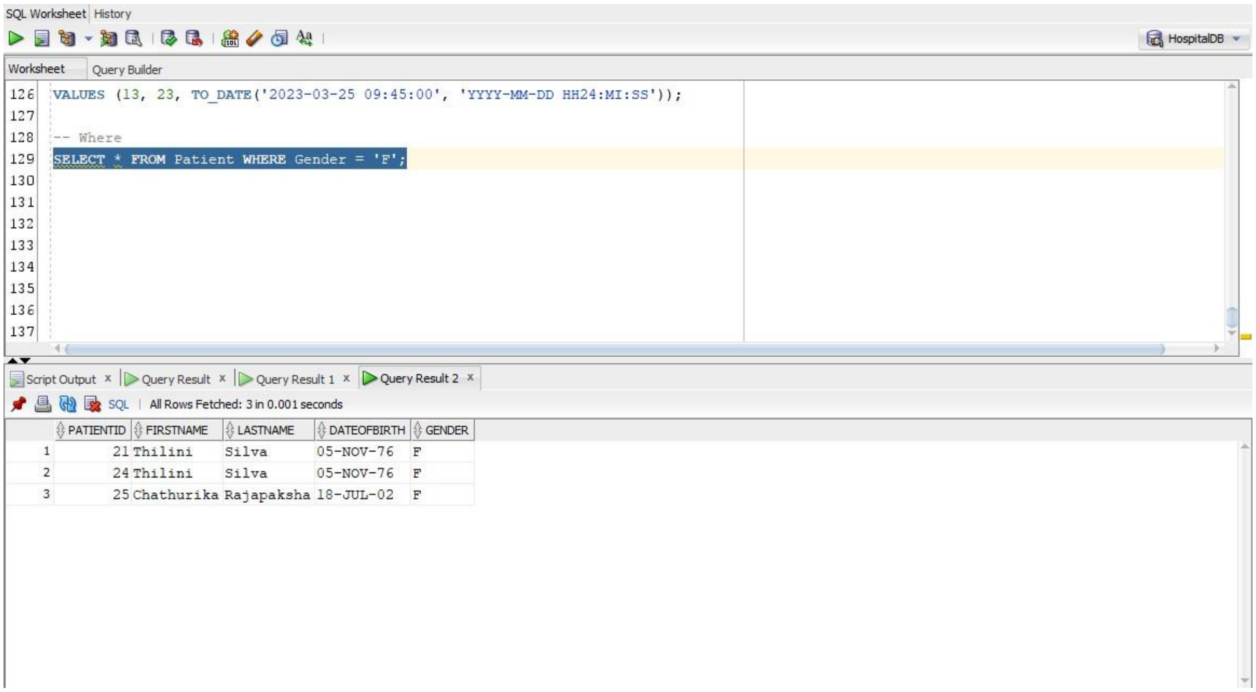


```
VALUES (13, 23, TO_DATE('2023-03-25 09:45:00', 'YYYY-MM-DD HH24:MI:SS'));

-- Where
SELECT * FROM Patient WHERE Gender = 'F';

SELECT * FROM Doctor WHERE DepartmentID = 3;

-- Group by
SELECT DepartmentID, COUNT(*) FROM Doctor GROUP BY DepartmentID;

-- Having
SELECT DepartmentID, COUNT(*) FROM Doctor GROUP BY DepartmentID HAVING COUNT(*) > 1;
```

All Rows Fetched: 1 in 0.006 seconds

	DOCTORID	FIRSTNAME	LASTNAME	DEPARTMENTID
1	14	Nadia	Rajapakse	3

Figure 10: DQL Command WHERE - 01



```
VALUES (13, 23, TO_DATE('2023-03-25 09:45:00', 'YYYY-MM-DD HH24:MI:SS'));

-- Where
SELECT * FROM Patient WHERE Gender = 'F';
```

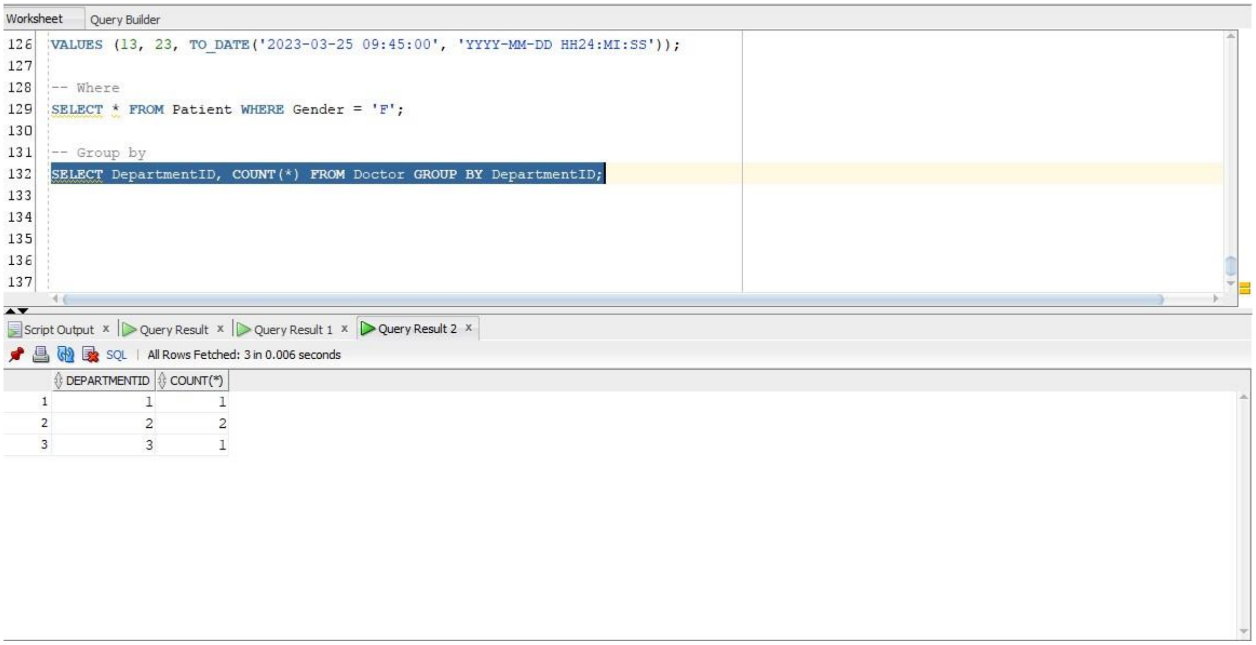
All Rows Fetched: 3 in 0.001 seconds

	PATIENTID	FIRSTNAME	LASTNAME	DATEOFBIRTH	GENDER
1	21	Thilini	Silva	05-NOV-76	F
2	24	Thilini	Silva	05-NOV-76	F
3	25	Chathurika	Rajapaksha	18-JUL-02	F

Figure 11: DQL Command WHERE - 02

- Group by

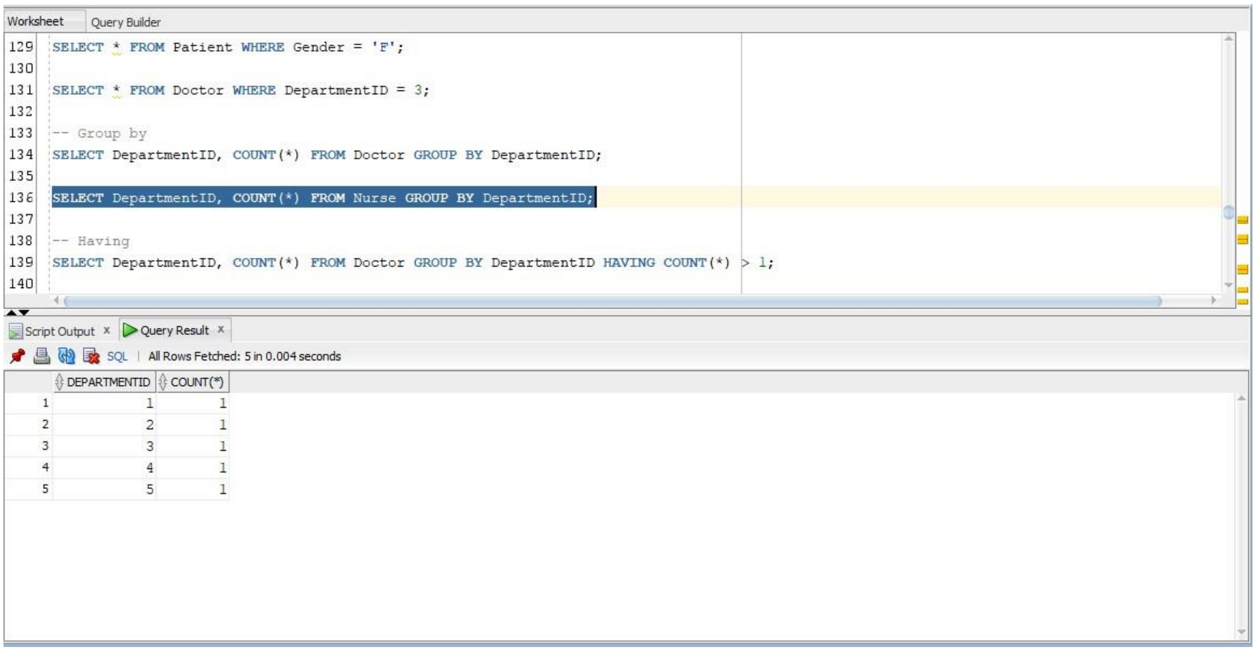
The first query calculates the count of doctors grouped by department, providing a tally of doctors in each department. The second query performs a similar operation for nurses, counting the number of nurses in each department. Both queries utilize the GROUP BY clause to aggregate data based on department, enabling the analysis of staff distribution across different departments.



```
VALUES (13, 23, TO_DATE('2023-03-25 09:45:00', 'YYYY-MM-DD HH24:MI:SS'));

-- Where
SELECT * FROM Patient WHERE Gender = 'F';

-- Group by
SELECT DepartmentID, COUNT(*) FROM Doctor GROUP BY DepartmentID;
```

```
All Rows Fetched: 3 in 0.006 seconds

    DEPARTMENTID  COUNT(*)
1              1         1
2              2         2
3              3         1
```

Figure 12: DQL Command GROUP BY - 01



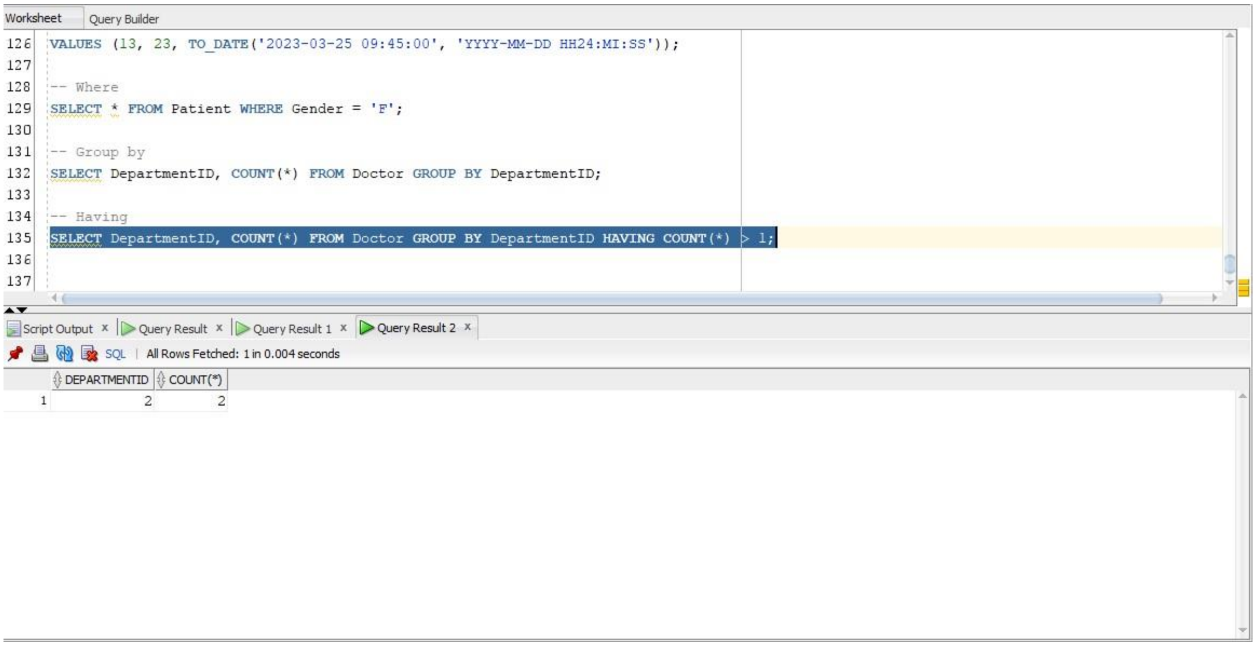
```
SELECT * FROM Patient WHERE Gender = 'F';

SELECT * FROM Doctor WHERE DepartmentID = 3;

-- Group by
SELECT DepartmentID, COUNT(*) FROM Doctor GROUP BY DepartmentID;

SELECT DepartmentID, COUNT(*) FROM Nurse GROUP BY DepartmentID;

-- Having
SELECT DepartmentID, COUNT(*) FROM Doctor GROUP BY DepartmentID HAVING COUNT(*) > 1;
```

```
All Rows Fetched: 5 in 0.004 seconds

    DEPARTMENTID  COUNT(*)
1              1         1
2              2         1
3              3         1
4              4         1
5              5         1
```

Figure 13: Figure 12: DQL Command GROUP BY - 02

- Having

This query retrieves department IDs along with the count of doctors in each department, filtering results to include only those departments with more than one doctor. It employs the GROUP BY clause to group doctors by department, and the HAVING clause to further refine results based on the count condition.



```
VALUES (13, 23, TO_DATE('2023-03-25 09:45:00', 'YYYY-MM-DD HH24:MI:SS'));

-- Where
SELECT * FROM Patient WHERE Gender = 'F';

-- Group by
SELECT DepartmentID, COUNT(*) FROM Doctor GROUP BY DepartmentID;

-- Having
SELECT DepartmentID, COUNT(*) FROM Doctor GROUP BY DepartmentID HAVING COUNT(*) > 1;
```

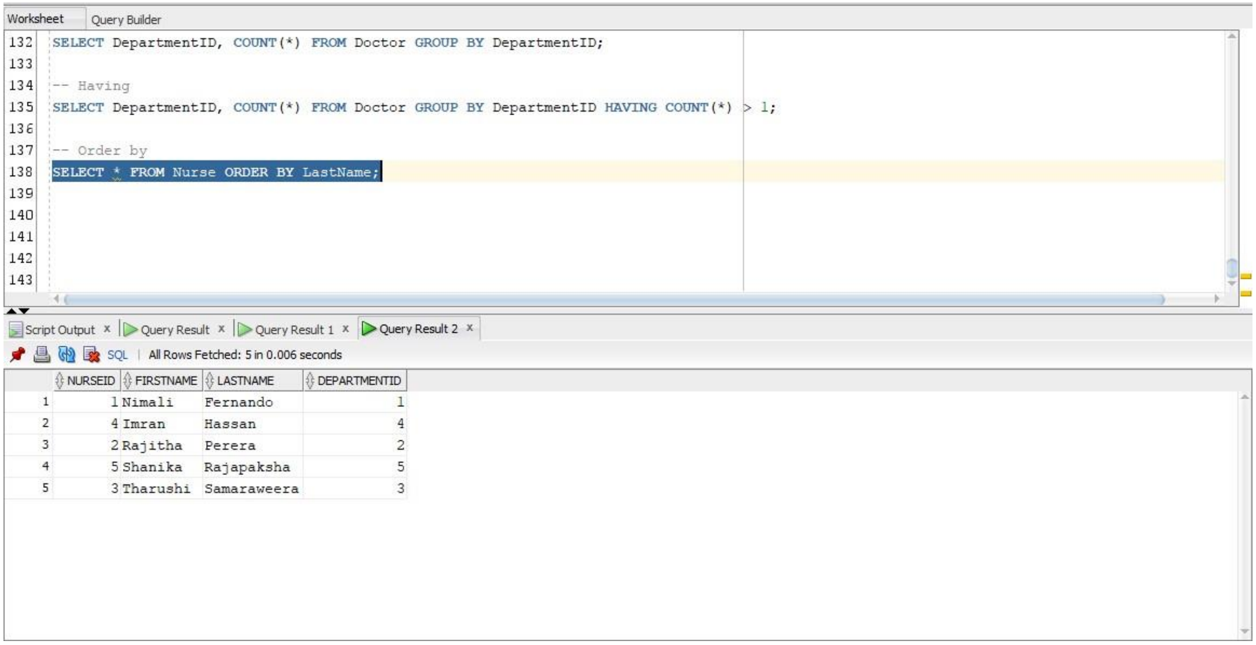
All Rows Fetched: 1 in 0.004 seconds

```
     DEPARTMENTID  COUNT(*)
1               2         2
```

Figure 14: Figure 12: DQL Command HAVING

- Order by

These queries retrieve all nurse and patient records, respectively, sorted alphabetically by last name for nurses and by last name then first name for patients. They utilize the ORDER BY clause to organize the results in ascending order based on specified attributes.

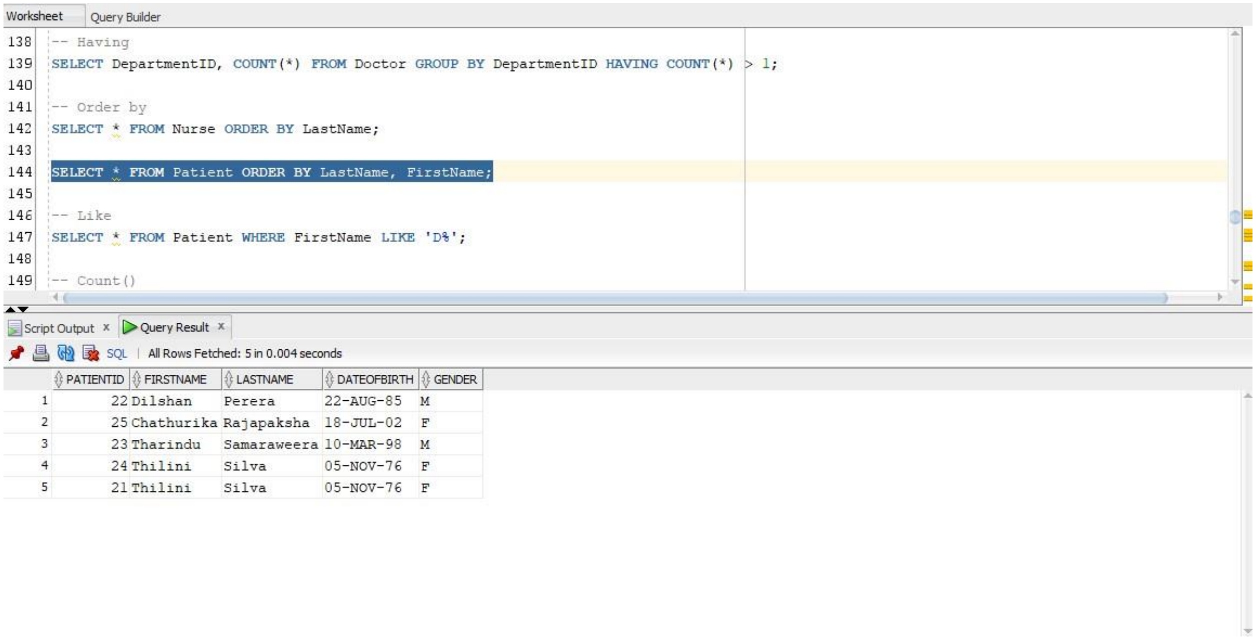


```
SELECT DepartmentID, COUNT(*) FROM Doctor GROUP BY DepartmentID;

-- Having
SELECT DepartmentID, COUNT(*) FROM Doctor GROUP BY DepartmentID HAVING COUNT(*) > 1;

-- Order by
SELECT * FROM Nurse ORDER BY LastName;
```

All Rows Fetched: 5 in 0.006 seconds

	NURSEID	FIRSTNAME	LASTNAME	DEPARTMENTID
1	1	Nimali	Fernando	1
2	4	Imran	Hassan	4
3	2	Rajitha	Perera	2
4	5	Shanika	Rajapaksha	5
5	3	Tharushi	Samaraweera	3

Figure 15: DQL Command ORDER BY - 01



```
-- Having
SELECT DepartmentID, COUNT(*) FROM Doctor GROUP BY DepartmentID HAVING COUNT(*) > 1;

-- Order by
SELECT * FROM Nurse ORDER BY LastName;

SELECT * FROM Patient ORDER BY LastName, FirstName;

-- Like
SELECT * FROM Patient WHERE FirstName LIKE 'D%';

-- Count()
```

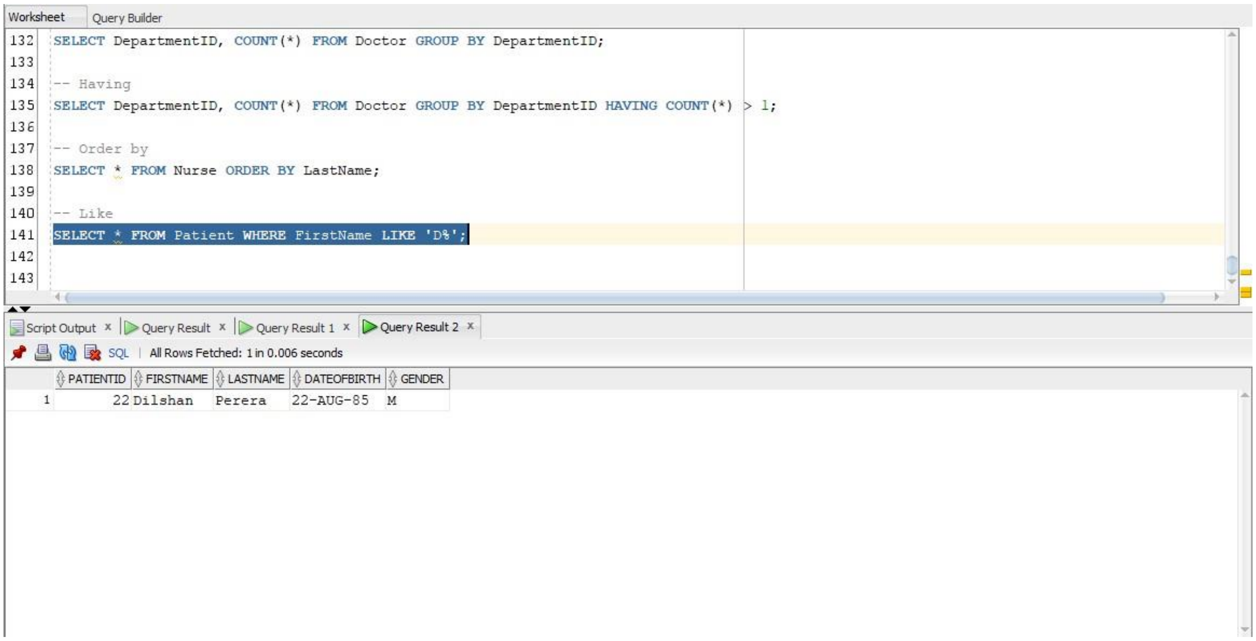
All Rows Fetched: 5 in 0.004 seconds

	PATIENTID	FIRSTNAME	LASTNAME	DATEOFBIRTH	GENDER
1	22	Dilshan	Perera	22-AUG-85	M
2	25	Chathurika	Rajapaksha	18-JUL-02	F
3	23	Tharindu	Samaraweera	10-MAR-98	M
4	24	Thilini	Silva	05-NOV-76	F
5	21	Thilini	Silva	05-NOV-76	F

Figure 16: DQL Command ORDER BY - 02

- Like

These queries retrieve patient and nurse records, respectively, filtering based on first names that start with the letter 'D' for patients and 'N' for nurses using the LIKE operator and a wildcard (%). They aim to find records where the first name matches the specified pattern.

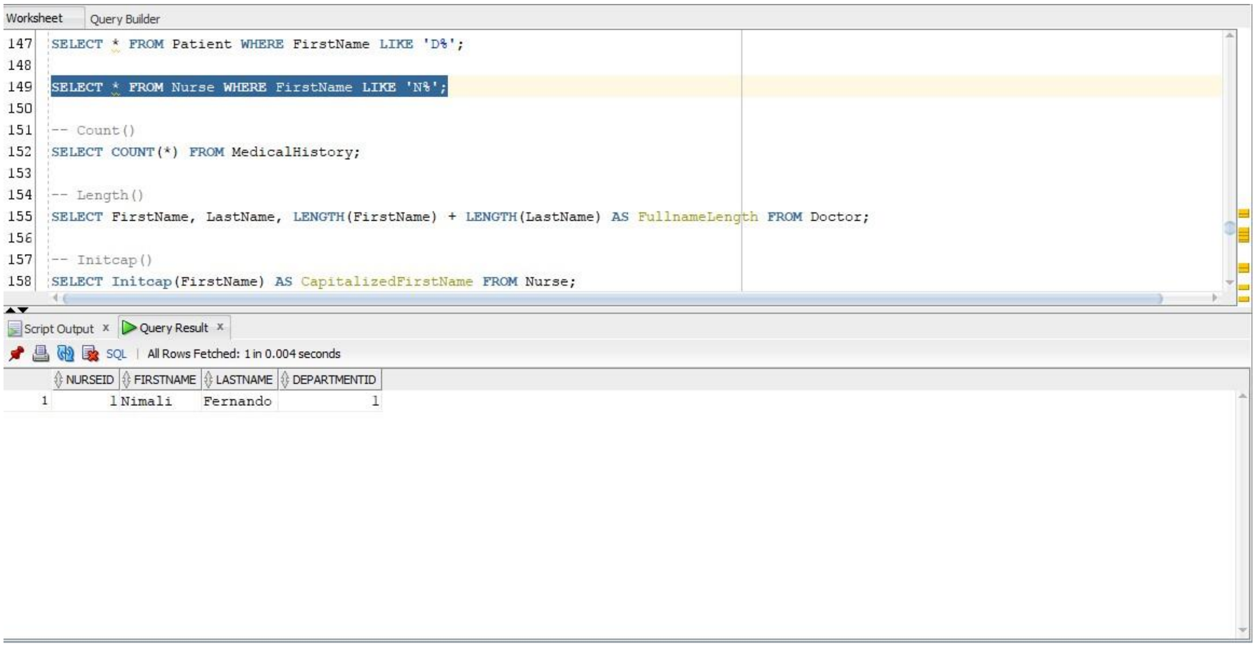


```
SELECT DepartmentID, COUNT(*) FROM Doctor GROUP BY DepartmentID;

-- Having
SELECT DepartmentID, COUNT(*) FROM Doctor GROUP BY DepartmentID HAVING COUNT(*) > 1;

-- Order by
SELECT * FROM Nurse ORDER BY LastName;

-- Like
SELECT * FROM Patient WHERE FirstName LIKE 'D%';
```

All Rows Fetched: 1 in 0.006 seconds

```
   PATIENTID FIRSTNAME LASTNAME DATEOFBIRTH GENDER
1         22 Dilshan   Perera   22-AUG-85   M
```

Figure 17: DQL Command LIKE - 01



```
SELECT * FROM Patient WHERE FirstName LIKE 'D%';

SELECT * FROM Nurse WHERE FirstName LIKE 'N%';

-- Count()
SELECT COUNT(*) FROM MedicalHistory;

-- Length()
SELECT FirstName, LastName, LENGTH(FirstName) + LENGTH(LastName) AS FullnameLength FROM Doctor;

-- Initcap()
SELECT Initcap(FirstName) AS CapitalizedFirstName FROM Nurse;
```

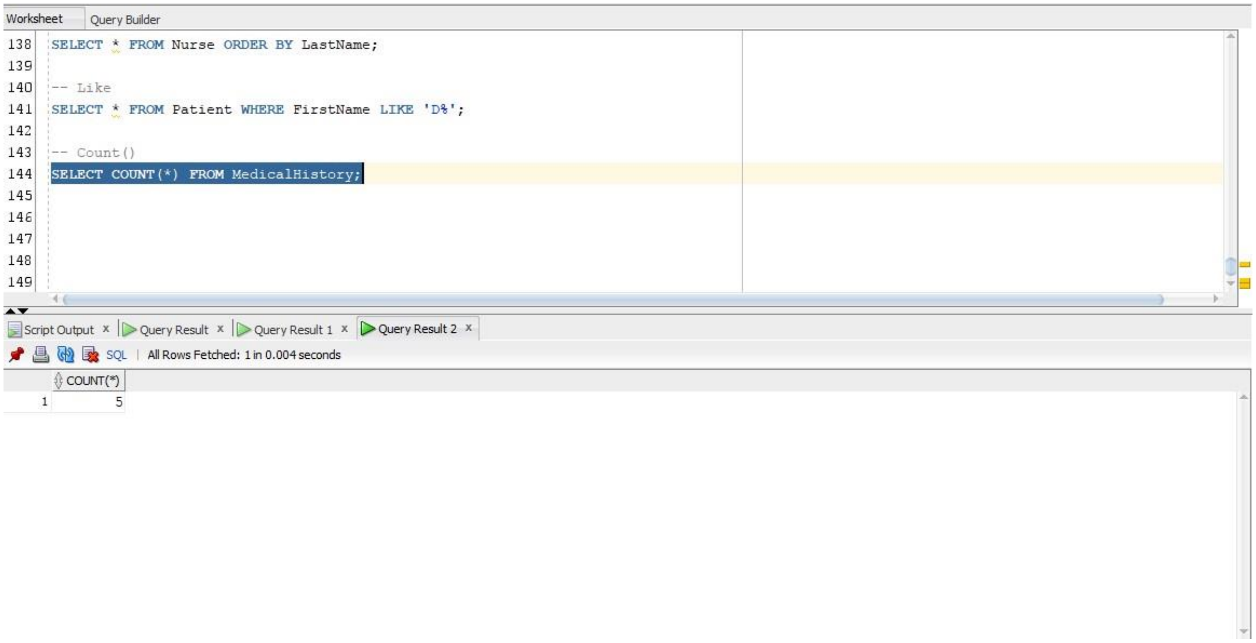
All Rows Fetched: 1 in 0.004 seconds

```
   NURSEID FIRSTNAME LASTNAME DEPARTMENTID
1        1 Nimali    Fernando            1
```

Figure 18: DQL Command LIKE - 02

- Count()

These queries count the total number of records in the MedicalHistory and Prescription tables, respectively, providing the total count of medical history records and prescription records stored in the database.

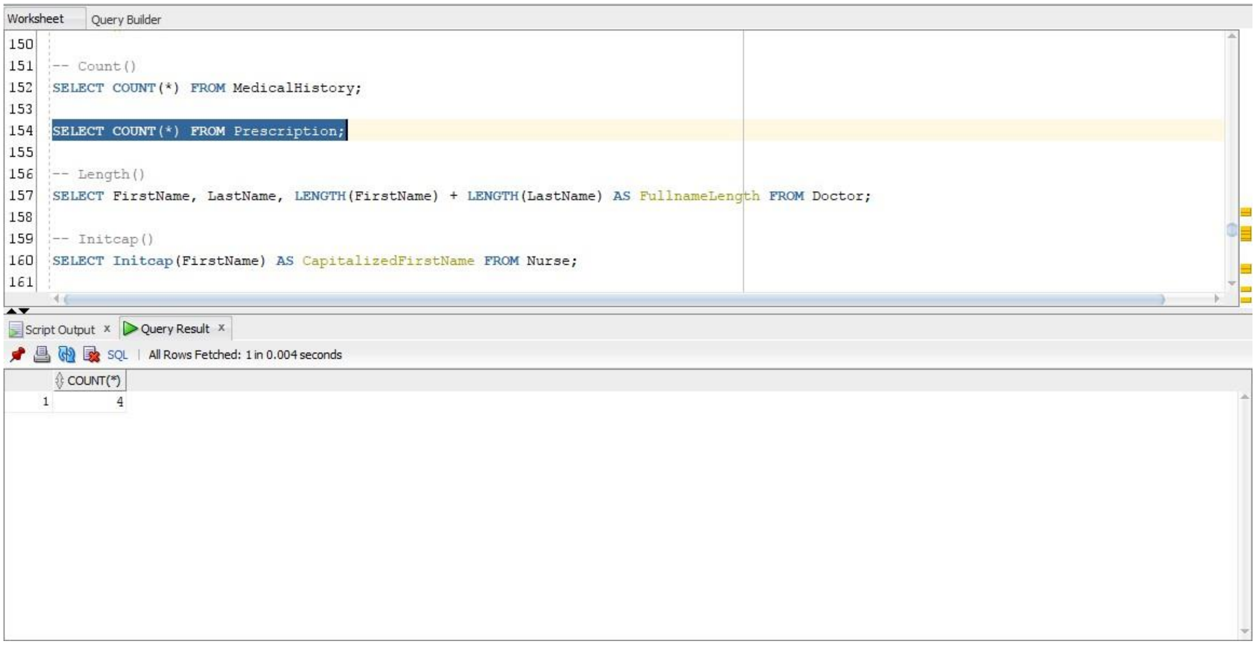


```
SELECT * FROM Nurse ORDER BY LastName;

-- Like
SELECT * FROM Patient WHERE FirstName LIKE 'D%';

-- Count()
SELECT COUNT(*) FROM MedicalHistory;
```

All Rows Fetched: 1 in 0.004 seconds

```
     COUNT(*)
1           5
```

Figure 19: DQL Command COUNT - 01



```

-- Count()
SELECT COUNT(*) FROM MedicalHistory;

SELECT COUNT(*) FROM Prescription;

-- Length()
SELECT FirstName, LastName, LENGTH(FirstName) + LENGTH(LastName) AS FullnameLength FROM Doctor;

-- Initcap()
SELECT Initcap(FirstName) AS CapitalizedFirstName FROM Nurse;
```

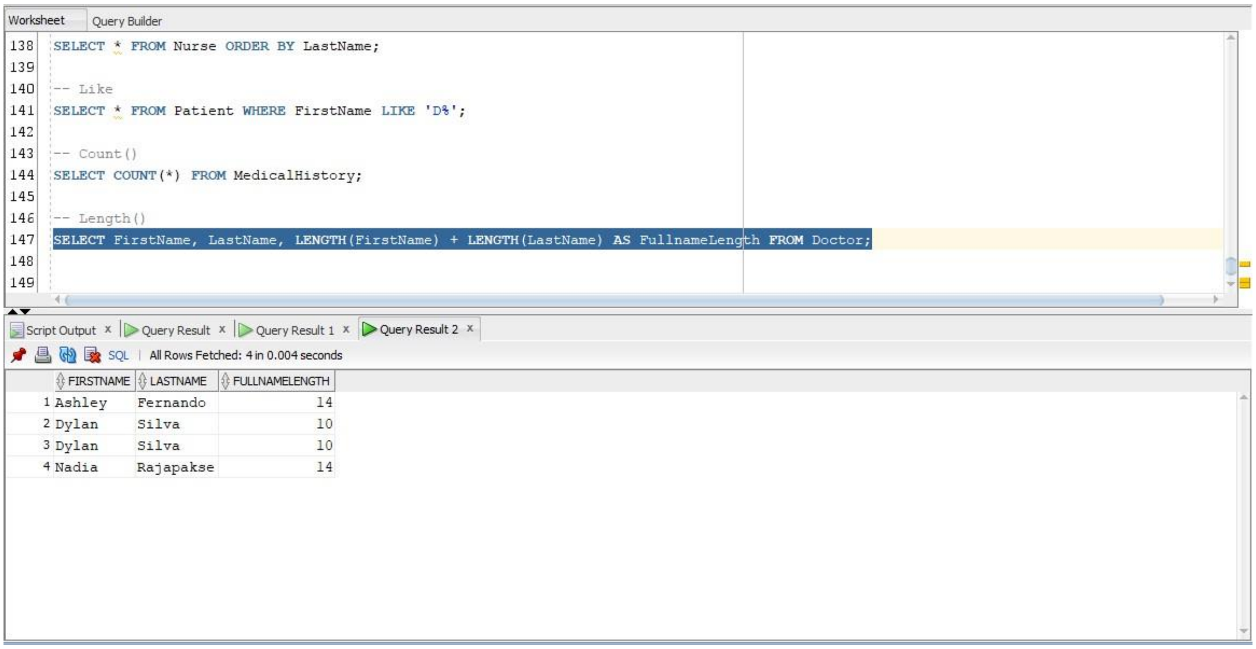
All Rows Fetched: 1 in 0.004 seconds

```
     COUNT(*)
1           4
```

Figure 20: DQL Command COUNT - 02

- Length()

These queries retrieve the first and last names of doctors and nurses along with the sum of the lengths of their first and last names, resulting in the total length of their full names. They utilize the LENGTH function to calculate the length of each name component and + operator to sum them up, providing the total length of the full name.

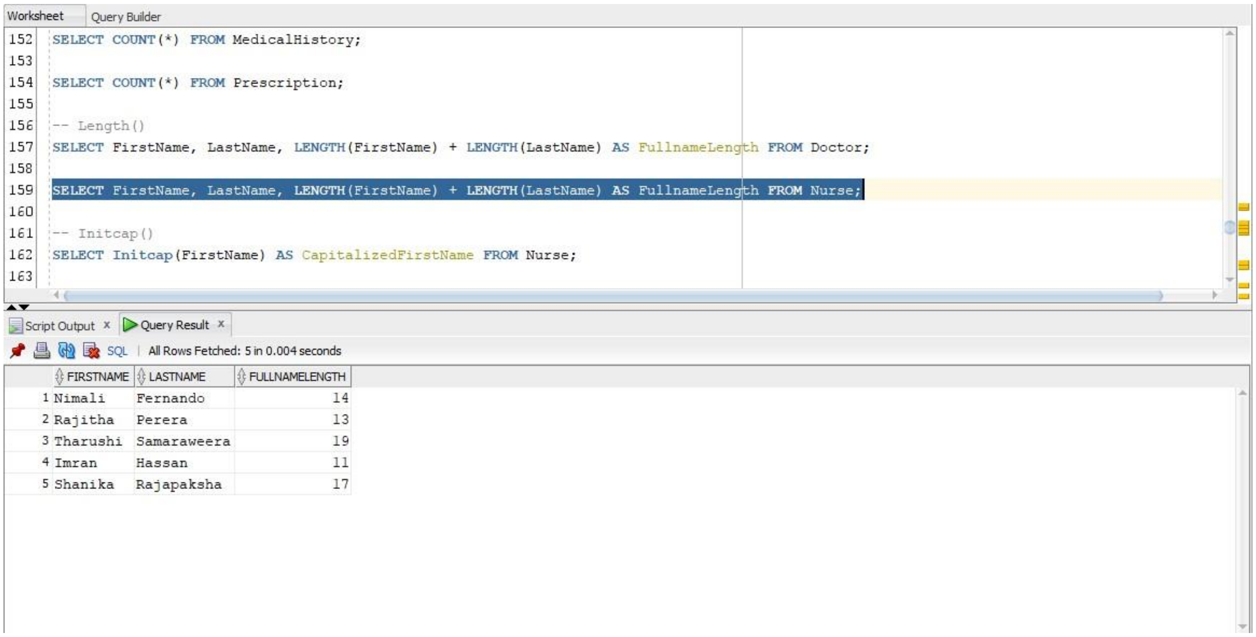


```
SELECT * FROM Nurse ORDER BY LastName;

-- Like
SELECT * FROM Patient WHERE FirstName LIKE 'D%';

-- Count()
SELECT COUNT(*) FROM MedicalHistory;

-- Length()
SELECT FirstName, LastName, LENGTH(FirstName) + LENGTH(LastName) AS FullnameLength FROM Doctor;
```

All Rows Fetched: 4 in 0.004 seconds

```
   FIRSTNAME  LASTNAME   FULLNAMELENGTH
 1 Ashley     Fernando               14
 2 Dylan      Silva                  10
 3 Dylan      Silva                  10
 4 Nadia      Rajapakse              14
```

Figure 21: DQL Command LENGTH - 01



```
SELECT COUNT(*) FROM MedicalHistory;

SELECT COUNT(*) FROM Prescription;

-- Length()
SELECT FirstName, LastName, LENGTH(FirstName) + LENGTH(LastName) AS FullnameLength FROM Doctor;

SELECT FirstName, LastName, LENGTH(FirstName) + LENGTH(LastName) AS FullnameLength FROM Nurse;

-- Initcap()
SELECT Initcap(FirstName) AS CapitalizedFirstName FROM Nurse;
```

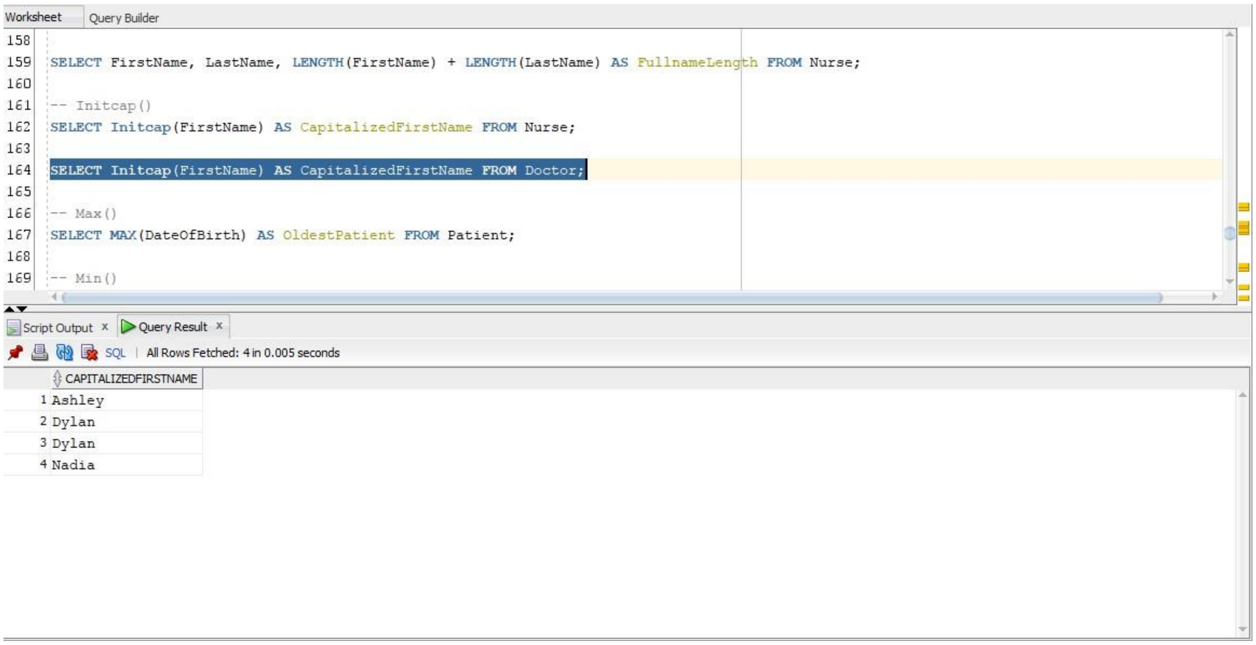
All Rows Fetched: 5 in 0.004 seconds

```
   FIRSTNAME  LASTNAME     FULLNAMELENGTH
 1 Nimali     Fernando                 14
 2 Rajitha    Perera                   13
 3 Tharushi   Samaraweera              19
 4 Imran      Hassan                   11
 5 Shanika    Rajapaksha               17
```

Figure 22: DQL Command LENGTH - 02

- initcap()

These queries retrieve the first names of nurses and doctors, respectively, with each first letter capitalized using the Initcap function. The Initcap function capitalizes the first letter of each word in a string. This operation results in the first names being presented with consistent capitalization, regardless of their original format.

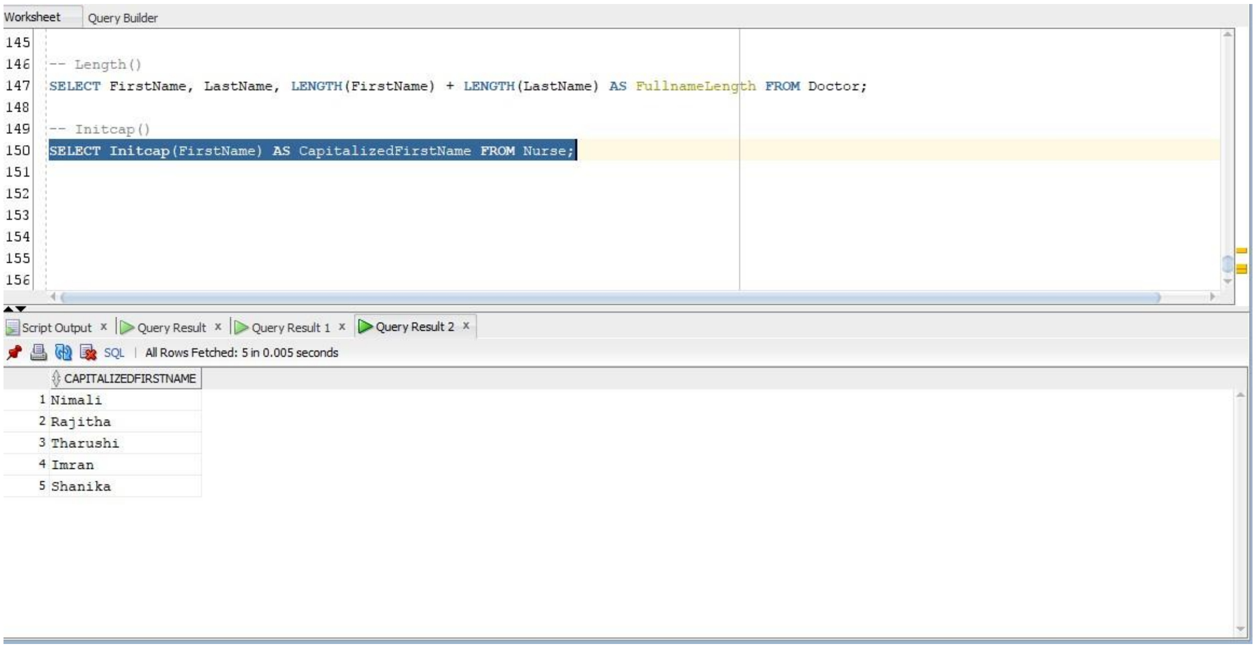


```
SELECT FirstName, LastName, LENGTH(FirstName) + LENGTH(LastName) AS FullnameLength FROM Nurse;

-- Initcap()
SELECT Initcap(FirstName) AS CapitalizedFirstName FROM Nurse;

SELECT Initcap(FirstName) AS CapitalizedFirstName FROM Doctor;

-- Max()
SELECT MAX(DateOfBirth) AS OldestPatient FROM Patient;

-- Min()
```

All Rows Fetched: 4 in 0.005 seconds

	CAPITALIZEDFIRSTNAME
1	Ashley
2	Dylan
3	Dylan
4	Nadia

Figure 23: DQL Command Initcap - 01



```
-- Length()
SELECT FirstName, LastName, LENGTH(FirstName) + LENGTH(LastName) AS FullnameLength FROM Doctor;

-- Initcap()
SELECT Initcap(FirstName) AS CapitalizedFirstName FROM Nurse;
```

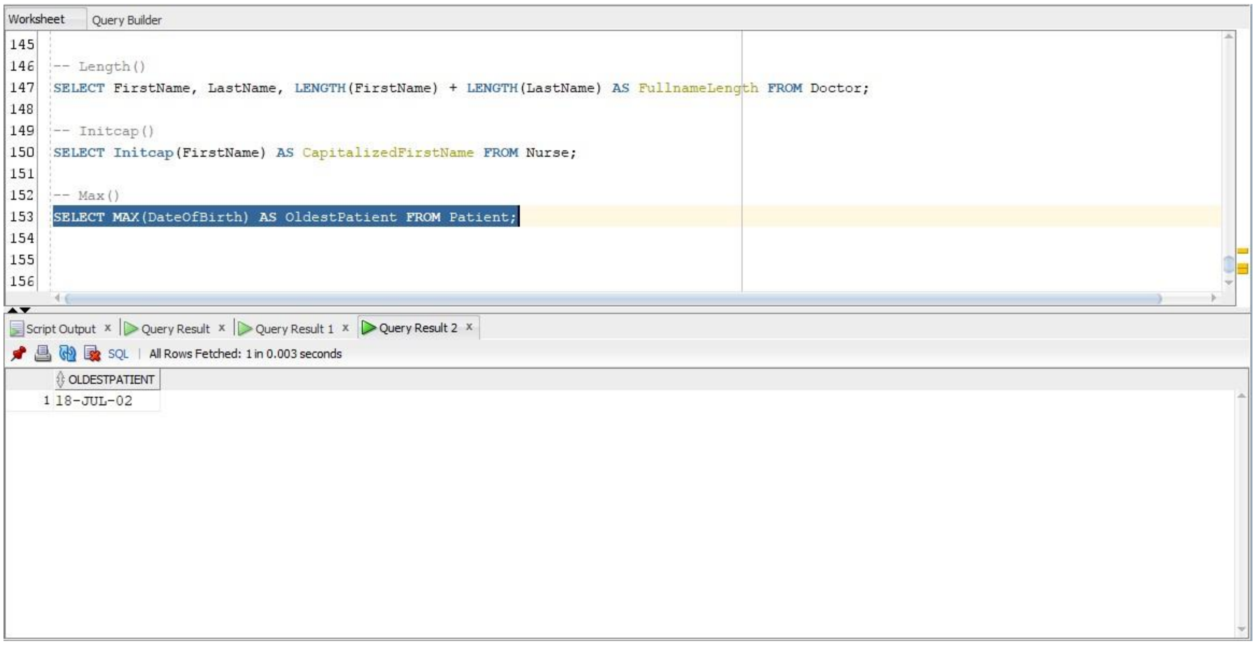
All Rows Fetched: 5 in 0.005 seconds

	CAPITALIZEDFIRSTNAME
1	Nimali
2	Rajitha
3	Tharushi
4	Imran
5	Shanika

Figure 24: DQL Command Initcap - 02

- Max()

This query retrieves the maximum (oldest) date of birth among all patients stored in the Patient table. The MAX() function is used to find the maximum value of the DateOfBirth attribute. It aliases the result as OldestPatient for clarity.



```
-- Length()
SELECT FirstName, LastName, LENGTH(FirstName) + LENGTH(LastName) AS FullnameLength FROM Doctor;

-- Initcap()
SELECT Initcap(FirstName) AS CapitalizedFirstName FROM Nurse;

-- Max()
SELECT MAX(DateOfBirth) AS OldestPatient FROM Patient;
```

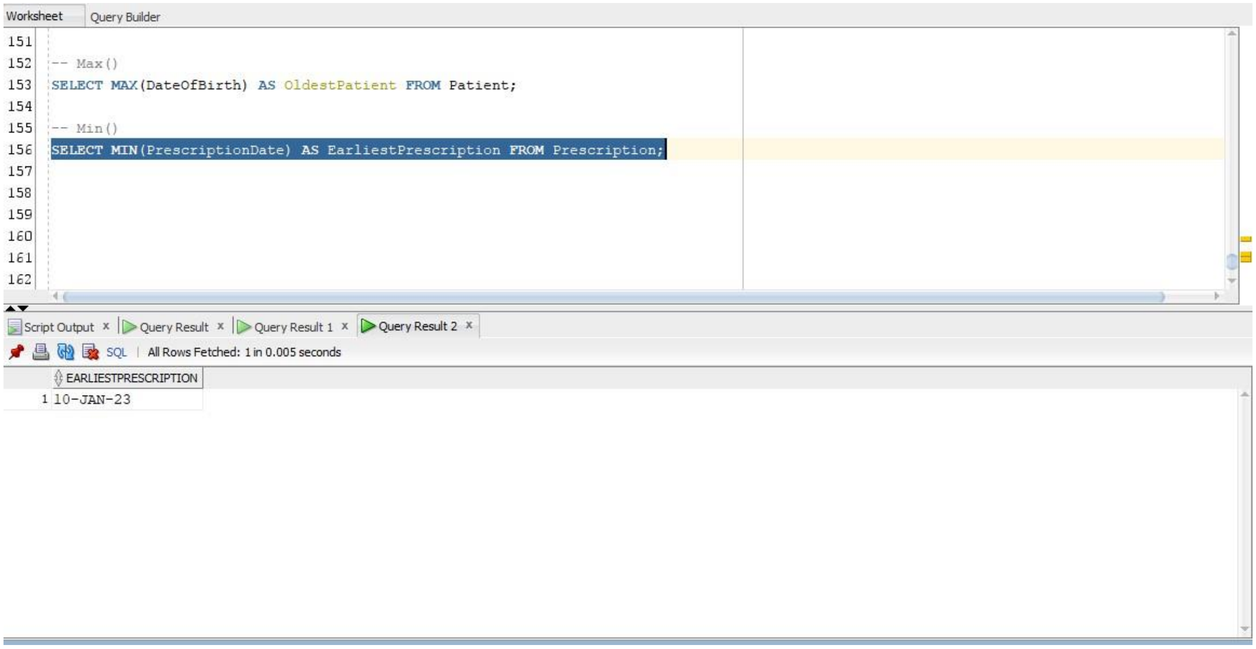
All Rows Fetched: 1 in 0.003 seconds

```
    OLDESTPATIENT
  1 18-JUL-02
```

Figure 25: DQL Command MAX

- Min()

This query retrieves the minimum (earliest) prescription date among all prescriptions stored in the Prescription table. The MIN() function is utilized to find the minimum value of the PrescriptionDate attribute. It aliases the result as EarliestPrescription for clarity.



```
-- Max()
SELECT MAX(DateOfBirth) AS OldestPatient FROM Patient;

-- Min()
SELECT MIN(PrescriptionDate) AS EarliestPrescription FROM Prescription;
```

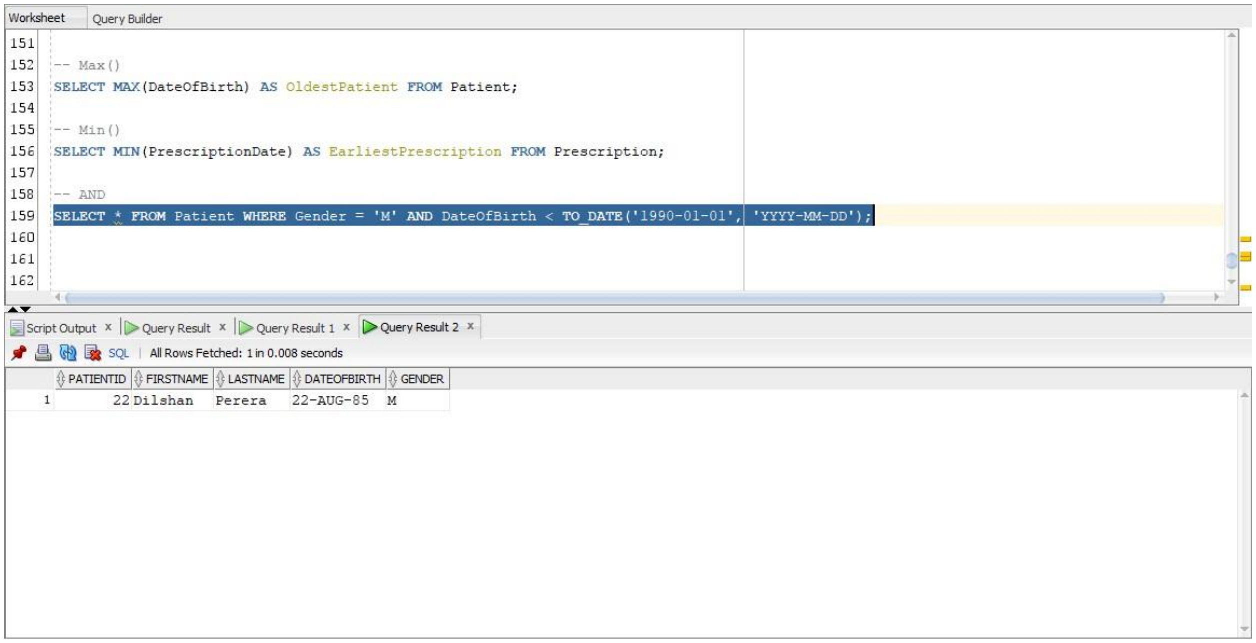
All Rows Fetched: 1 in 0.005 seconds

```
    EARLIESTPRESCRIPTION
  1 10-JAN-23
```

Figure 26: DQL Command MIN

- AND

This query selects all male patients (Gender = 'M') whose date of birth is before January 1, 1990. It combines two conditions using the AND operator to narrow down the result set based on both gender and date of birth.



```

-- Max()
SELECT MAX(DateOfBirth) AS OldestPatient FROM Patient;

-- Min()
SELECT MIN(PrescriptionDate) AS EarliestPrescription FROM Prescription;

-- AND
SELECT * FROM Patient WHERE Gender = 'M' AND DateOfBirth < TO_DATE('1990-01-01', 'YYYY-MM-DD');
```

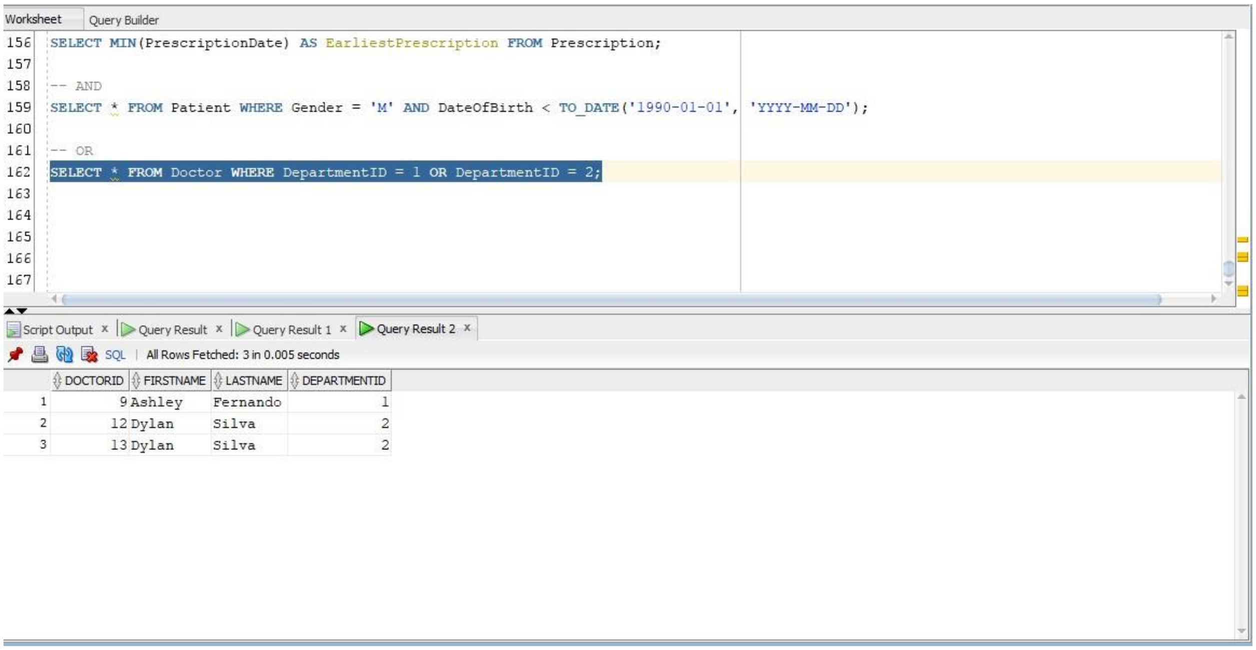
All Rows Fetched: 1 in 0.008 seconds

	PATIENTID	FIRSTNAME	LASTNAME	DATEOFBIRTH	GENDER
1	22	Dilshan	Perera	22-AUG-85	M

Figure 27: DQL Command AND

- OR

This query retrieves all doctors who belong to either department 1 or department 2. It uses the OR logical operator to include doctors from either department, broadening the scope of the result set.



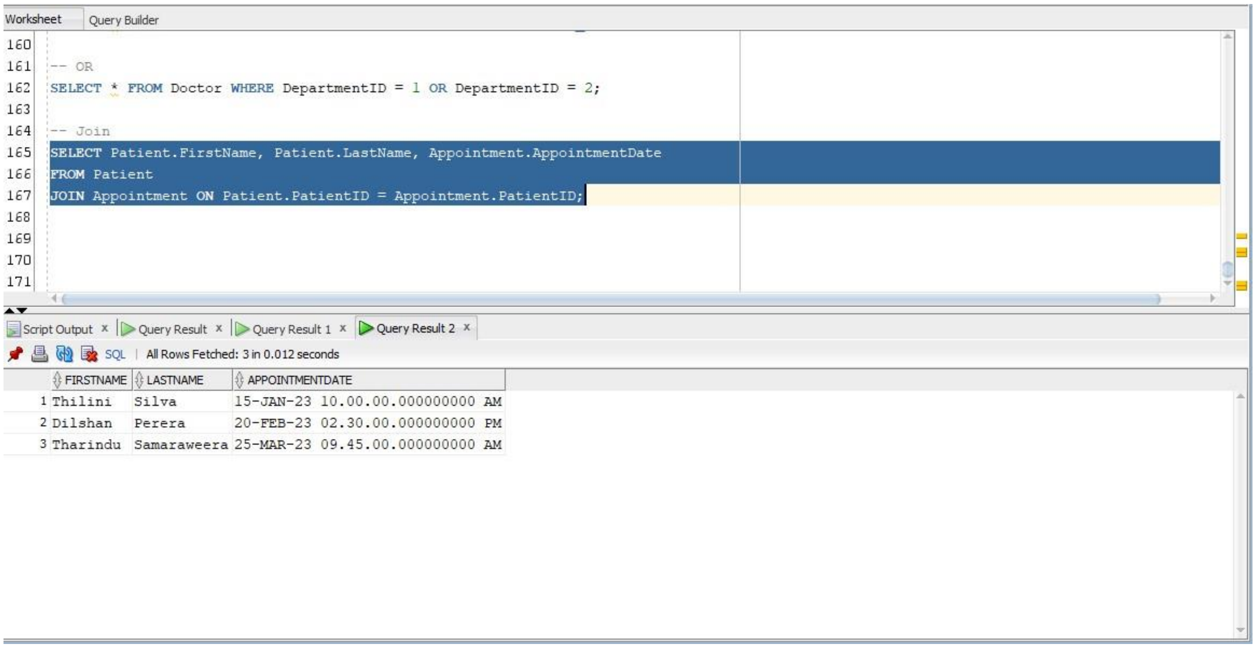
```sql
SELECT MIN(PrescriptionDate) AS EarliestPrescription FROM Prescription;

-- AND
SELECT * FROM Patient WHERE Gender = 'M' AND DateOfBirth < TO_DATE('1990-01-01', 'YYYY-MM-DD');

-- OR
SELECT * FROM Doctor WHERE DepartmentID = 1 OR DepartmentID = 2;
```

	DOCTORID	FIRSTNAME	LASTNAME	DEPARTMENTID
1	9	Ashley	Fernando	1
2	12	Dylan	Silva	2
3	13	Dylan	Silva	2

Figure 28: DQL Command OR

- Join

This query combines data from the Patient and Appointment tables based on the common PatientID attribute. It retrieves patient first names, last names, and appointment dates by joining the tables, allowing for the retrieval of patient information alongside their appointment details.



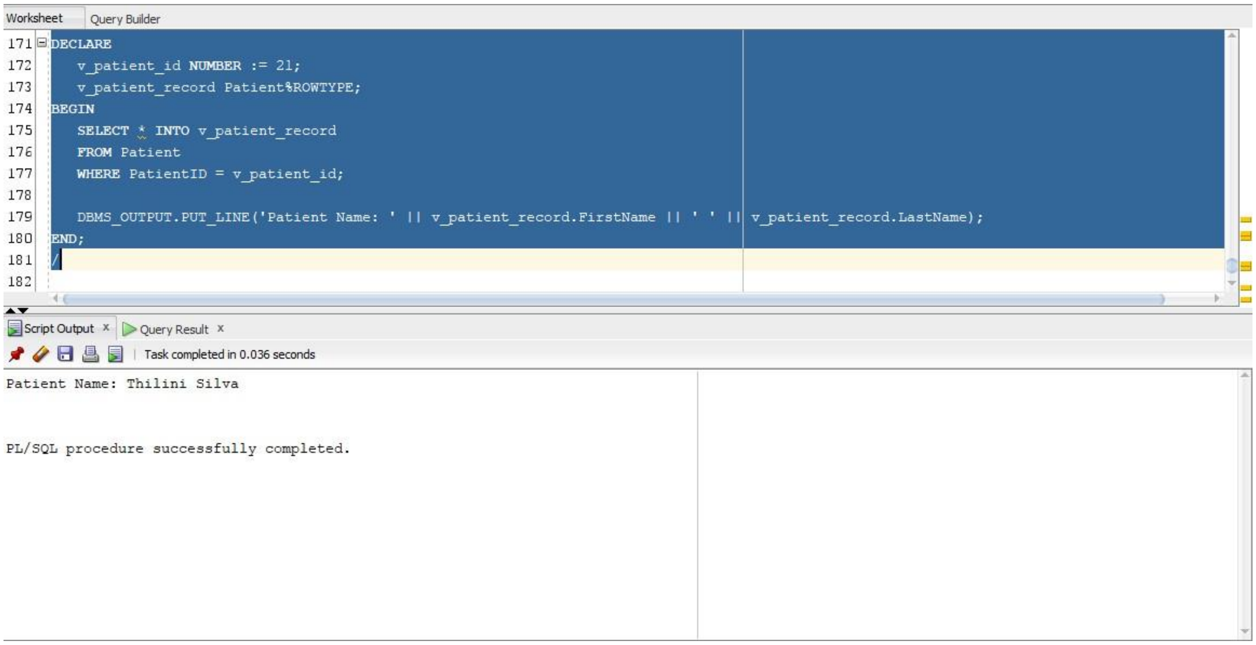
```sql
-- OR
SELECT * FROM Doctor WHERE DepartmentID = 1 OR DepartmentID = 2;

-- Join
SELECT Patient.FirstName, Patient.LastName, Appointment.AppointmentDate
FROM Patient
JOIN Appointment ON Patient.PatientID = Appointment.PatientID;
```

	FIRSTNAME	LASTNAME	APPOINTMENTDATE
1	Thilini	Silva	15-JAN-23 10.00.00.000000000 AM
2	Dilshan	Perera	20-FEB-23 02.30.00.000000000 PM
3	Tharindu	Samaraweera	25-MAR-23 09.45.00.000000000 AM

Figure 29: DQL Command JOIN

5)Write five PL/SQL programs to retrieve data based on the user given inputs.

This PL/SQL block retrieves a patient record based on the provided v_patient_id. It uses %ROWTYPE to define a record variable matching the structure of the Patient table. Then, it selects the patient record into the record variable v_patient_record and prints the patient's full name using DBMS_OUTPUT.PUT_LINE.



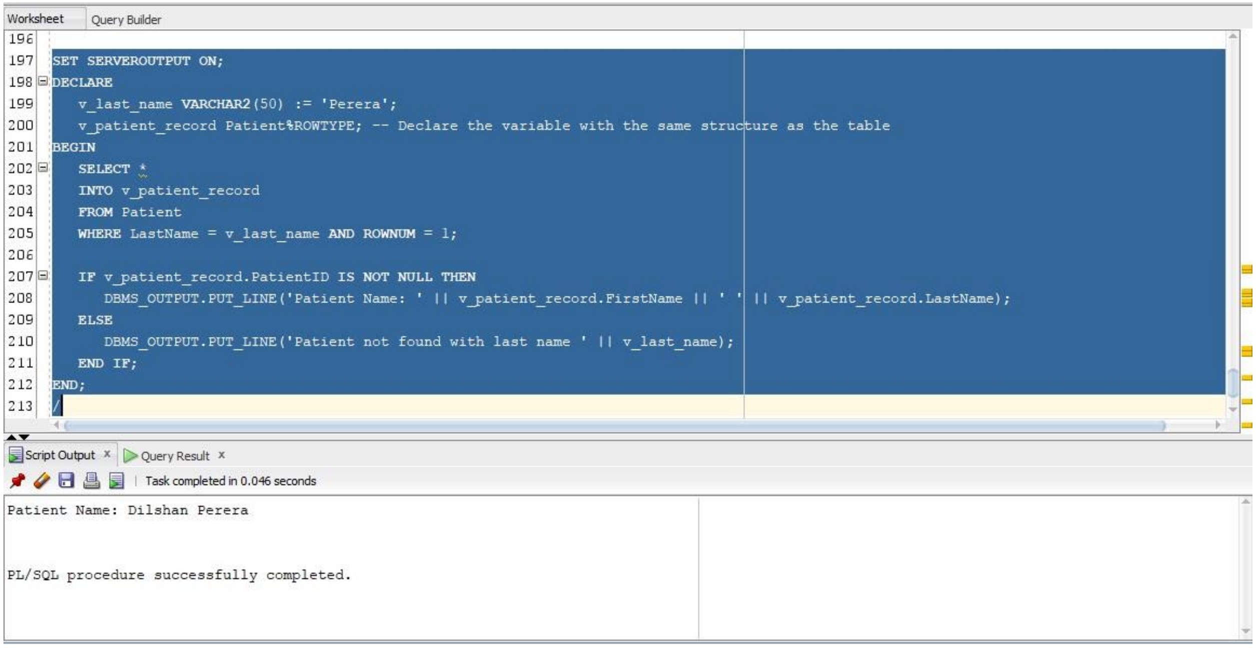
```
DECLARE
   v_patient_id NUMBER := 21;
   v_patient_record Patient%ROWTYPE;
BEGIN
   SELECT * INTO v_patient_record
   FROM Patient
   WHERE PatientID = v_patient_id;

   DBMS_OUTPUT.PUT_LINE('Patient Name: ' || v_patient_record.FirstName || ' ' || v_patient_record.LastName);
END;
/
```

Task completed in 0.036 seconds

```
Patient Name: Thilini Silva


PL/SQL procedure successfully completed.
```

Figure 30: PL/SQL program 1: Retrieve patient details based on PatientID

This PL/SQL block searches for a patient record with the last name 'Perera' in the Patient table and retrieves it. If found, it prints the patient's full name using DBMS_OUTPUT.PUT_LINE. If no patient is found with the provided last name, it outputs a message indicating the absence of such a patient.



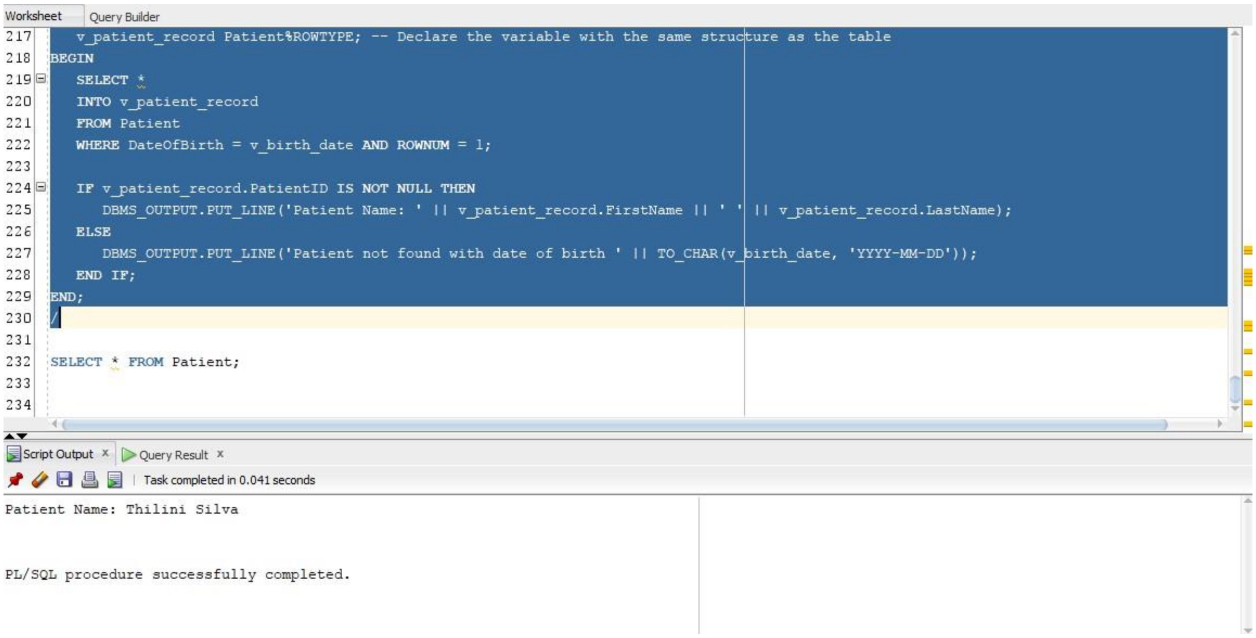
```
SET SERVEROUTPUT ON;
DECLARE
   v_last_name VARCHAR2(50) := 'Perera';
   v_patient_record Patient%ROWTYPE; -- Declare the variable with the same structure as the table
BEGIN
   SELECT *
   INTO v_patient_record
   FROM Patient
   WHERE LastName = v_last_name AND ROWNUM = 1;

   IF v_patient_record.PatientID IS NOT NULL THEN
      DBMS_OUTPUT.PUT_LINE('Patient Name: ' || v_patient_record.FirstName || ' ' || v_patient_record.LastName);
   ELSE
      DBMS_OUTPUT.PUT_LINE('Patient not found with last name ' || v_last_name);
   END IF;
END;
/
```

Task completed in 0.046 seconds

```
Patient Name: Dilshan Perera


PL/SQL procedure successfully completed.
```

Figure 31: PL/SQL Program 2: Retrieve Patient Details by Last Name

This PL/SQL block searches for a patient record with the date of birth '1976-11-05' in the Patient table and retrieves it. If found, it prints the patient's full name using DBMS_OUTPUT.PUT_LINE. If no patient is found with the provided date of birth, it outputs a message indicating the absence of such a patient.



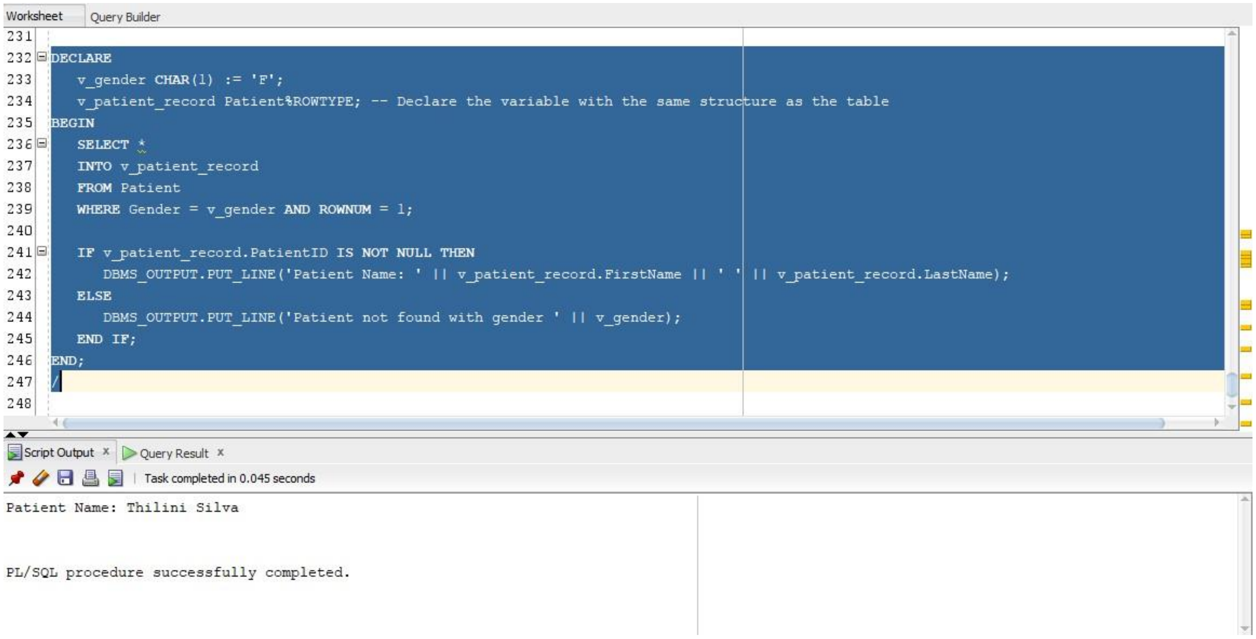
```
   v_patient_record Patient%ROWTYPE; -- Declare the variable with the same structure as the table
BEGIN
   SELECT *
   INTO v_patient_record
   FROM Patient
   WHERE DateOfBirth = v_birth_date AND ROWNUM = 1;

   IF v_patient_record.PatientID IS NOT NULL THEN
      DBMS_OUTPUT.PUT_LINE('Patient Name: ' || v_patient_record.FirstName || ' ' || v_patient_record.LastName);
   ELSE
      DBMS_OUTPUT.PUT_LINE('Patient not found with date of birth ' || TO_CHAR(v_birth_date, 'YYYY-MM-DD'));
   END IF;
END;
/

SELECT * FROM Patient;
```

Task completed in 0.041 seconds

```
Patient Name: Thilini Silva


PL/SQL procedure successfully completed.
```

Figure 32: PL/SQL Program 3: Retrieve Patient Details by Date of Birth

This PL/SQL block searches for a female patient record in the Patient table and retrieves it. If found, it prints the patient's full name using DBMS_OUTPUT.PUT_LINE. If no patient is found with the provided gender, it outputs a message indicating the absence of such a patient.



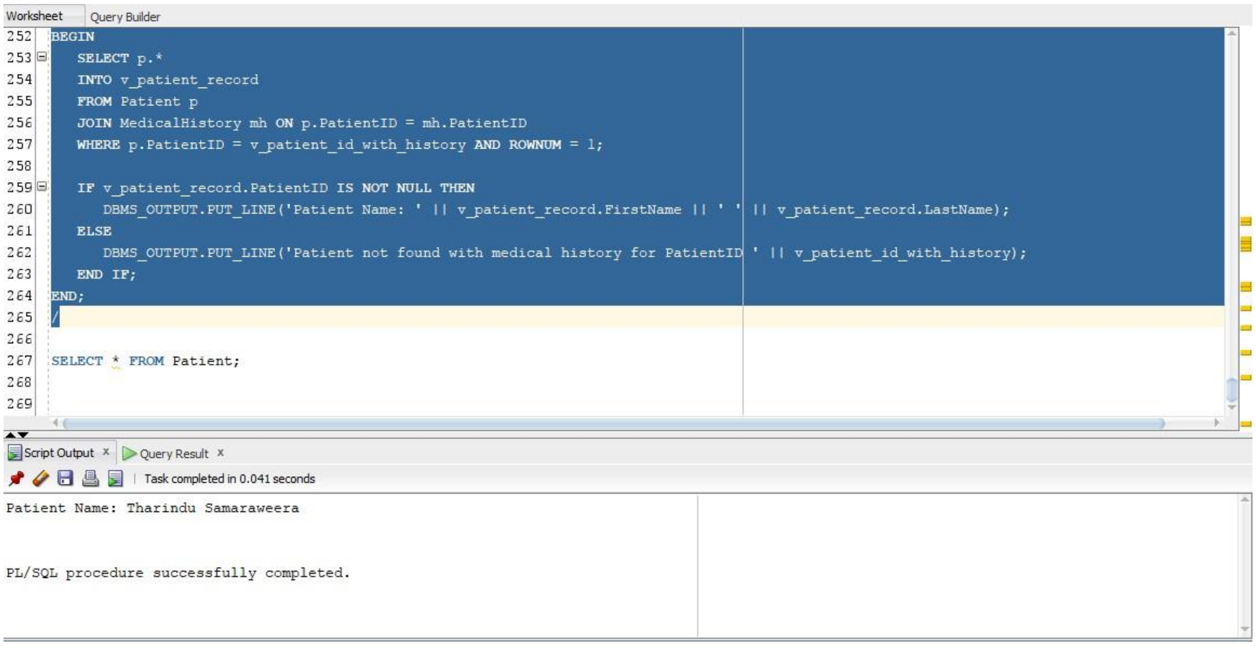
```
DECLARE
   v_gender CHAR(1) := 'F';
   v_patient_record Patient%ROWTYPE; -- Declare the variable with the same structure as the table
BEGIN
   SELECT *
   INTO v_patient_record
   FROM Patient
   WHERE Gender = v_gender AND ROWNUM = 1;

   IF v_patient_record.PatientID IS NOT NULL THEN
      DBMS_OUTPUT.PUT_LINE('Patient Name: ' || v_patient_record.FirstName || ' ' || v_patient_record.LastName);
   ELSE
      DBMS_OUTPUT.PUT_LINE('Patient not found with gender ' || v_gender);
   END IF;
END;
/
```

Task completed in 0.045 seconds

```
Patient Name: Thilini Silva


PL/SQL procedure successfully completed.
```

Figure 33: PL/SQL Program 4: Retrieve Patient Details by Gender

This PL/SQL block searches for a patient with medical history associated with the provided v_patient_id_with_history in the Patient and MedicalHistory tables. If found, it prints the patient's full name using DBMS_OUTPUT.PUT_LINE. If no patient with medical history is found for the provided patient ID, it outputs a message indicating the absence of such a patient.



```
BEGIN
   SELECT p.*
   INTO v_patient_record
   FROM Patient p
   JOIN MedicalHistory mh ON p.PatientID = mh.PatientID
   WHERE p.PatientID = v_patient_id_with_history AND ROWNUM = 1;

   IF v_patient_record.PatientID IS NOT NULL THEN
      DBMS_OUTPUT.PUT_LINE('Patient Name: ' || v_patient_record.FirstName || ' ' || v_patient_record.LastName);
   ELSE
      DBMS_OUTPUT.PUT_LINE('Patient not found with medical history for PatientID ' || v_patient_id_with_history);
   END IF;
END;
/

SELECT * FROM Patient;
```

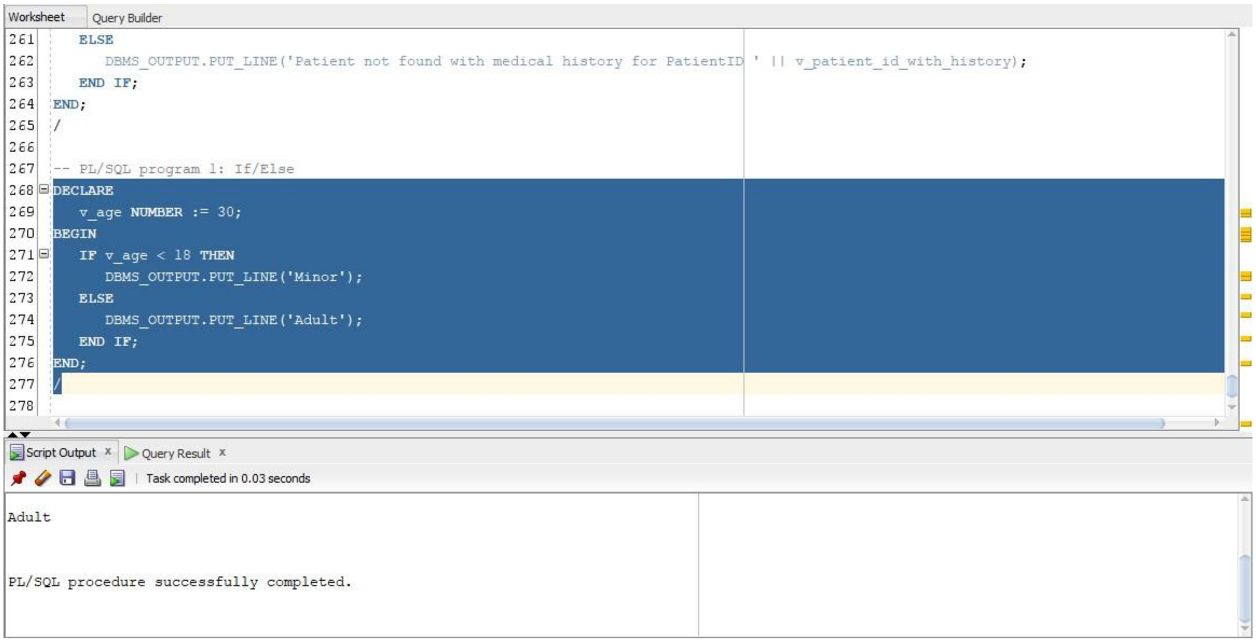
```
Patient Name: Tharindu Samaraweera


PL/SQL procedure successfully completed.
```

Figure 34: PL/SQL Program 5: Retrieve Patient Details with Medical History

6) Write PL/SQL programs to cover following control structures.

- If/Else

This PL/SQL block evaluates the value of the variable v_age to determine whether it represents a minor or an adult. If v_age is less than 18, it prints 'Minor' using DBMS_OUTPUT.PUT_LINE. Otherwise, it prints 'Adult'.



```
    ELSE
        DBMS_OUTPUT.PUT_LINE('Patient not found with medical history for PatientID ' || v_patient_id_with_history);
    END IF;
END;
/

-- PL/SQL program 1: If/Else
DECLARE
    v_age NUMBER := 30;
BEGIN
    IF v_age < 18 THEN
        DBMS_OUTPUT.PUT_LINE('Minor');
    ELSE
        DBMS_OUTPUT.PUT_LINE('Adult');
    END IF;
END;
/
```

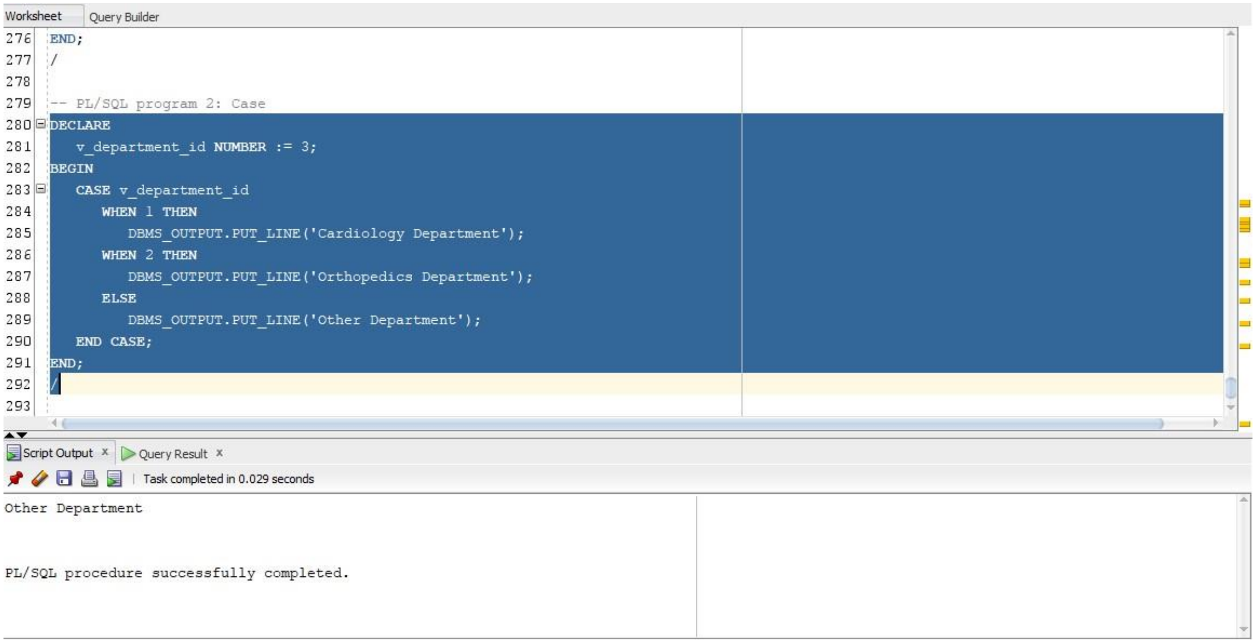
Task completed in 0.03 seconds

```
Adult


PL/SQL procedure successfully completed.
```

Figure 35: PL/SQL program 1: If/Else

- Case

This PL/SQL block utilizes a CASE statement to evaluate the value of the variable v_department_id. Depending on its value, it prints the corresponding department name. If v_department_id is 1, it prints 'Cardiology Department'; if it's 2, it prints 'Orthopedics Department'. For any other value, it prints 'Other Department'.



```
END;
/

-- PL/SQL program 2: Case
DECLARE
    v_department_id NUMBER := 3;
BEGIN
    CASE v_department_id
        WHEN 1 THEN
            DBMS_OUTPUT.PUT_LINE('Cardiology Department');
        WHEN 2 THEN
            DBMS_OUTPUT.PUT_LINE('Orthopedics Department');
        ELSE
            DBMS_OUTPUT.PUT_LINE('Other Department');
    END CASE;
END;
/
```

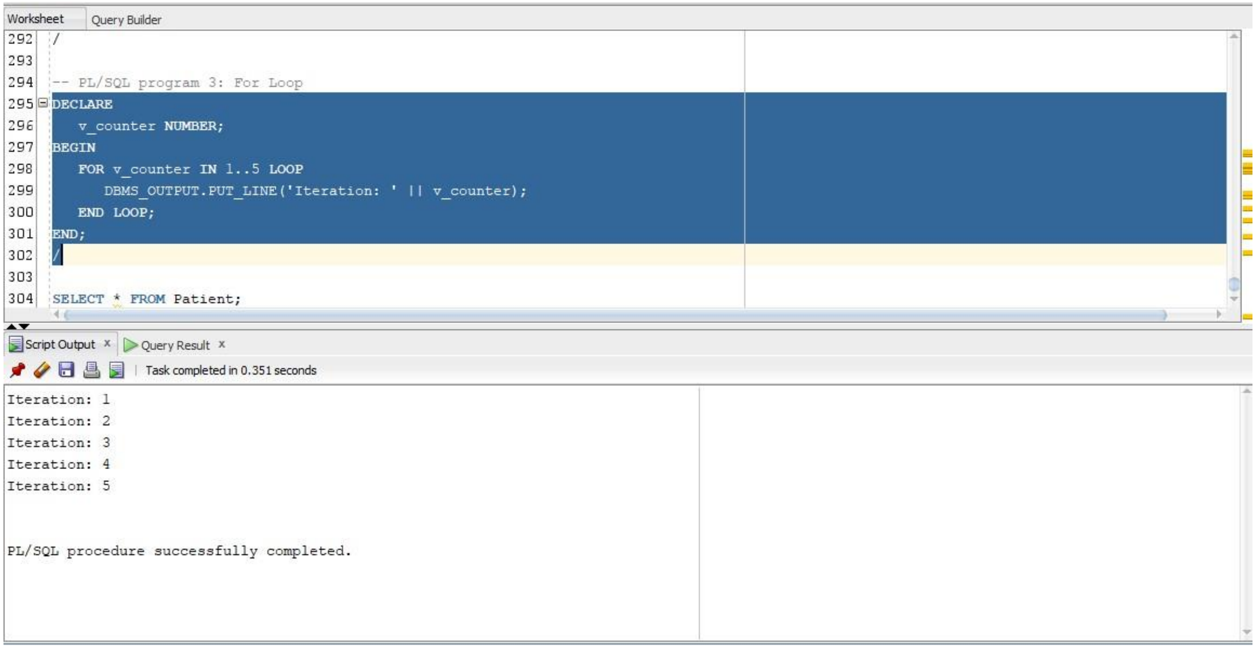
Task completed in 0.029 seconds

```
Other Department


PL/SQL procedure successfully completed.
```

Figure 36: PL/SQL program 2: Case

- For Loop

This PL/SQL block initializes a counter variable v_counter and iterates through a loop from 1 to 5 using a FOR loop construct. Within each iteration, it prints the current iteration number using DBMS_OUTPUT.PUT_LINE. After completing five iterations, the loop terminates.



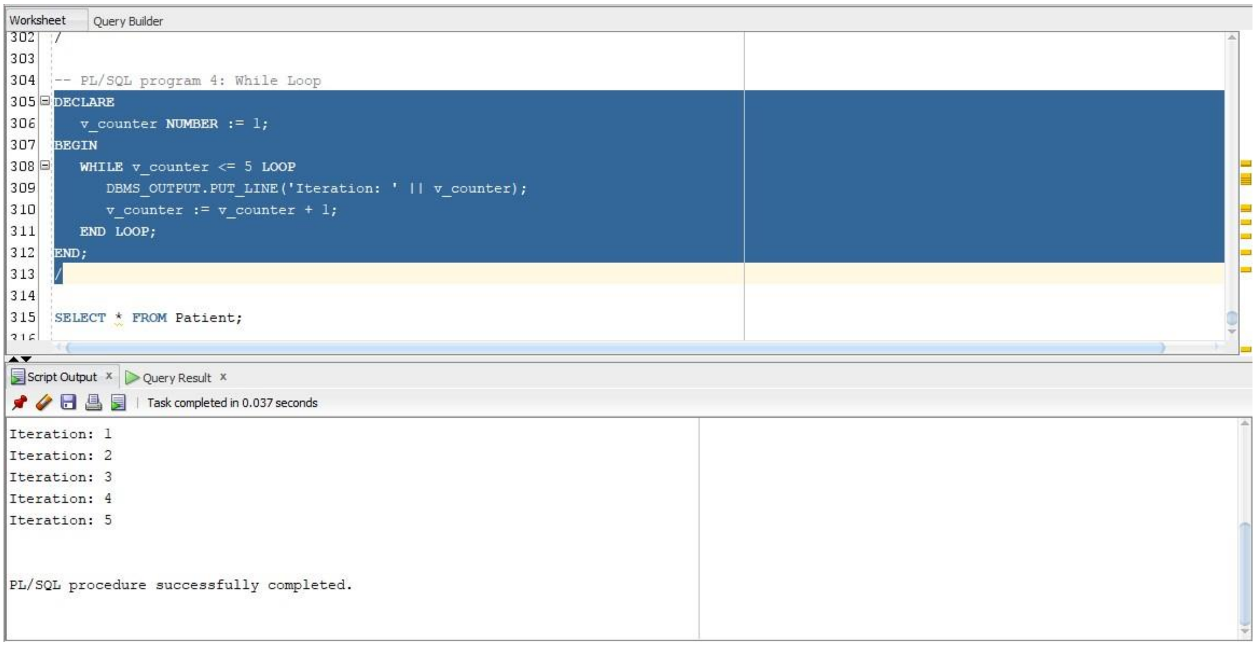
```
/

-- PL/SQL program 3: For Loop
DECLARE
   v_counter NUMBER;
BEGIN
   FOR v_counter IN 1..5 LOOP
      DBMS_OUTPUT.PUT_LINE('Iteration: ' || v_counter);
   END LOOP;
END;
/

SELECT * FROM Patient;
```

```
Iteration: 1
Iteration: 2
Iteration: 3
Iteration: 4
Iteration: 5


PL/SQL procedure successfully completed.
```

Figure 37: PL/SQL program 3: For Loop

- While Loop

This PL/SQL block initializes a counter variable v_counter with an initial value of 1. It enters a WHILE loop that continues iterating as long as v_counter is less than or equal to 5. Within each iteration, it prints the current iteration number using DBMS_OUTPUT.PUT_LINE and increments v_counter by 1. The loop continues until v_counter exceeds 5, after which it terminates.



```
/

-- PL/SQL program 4: While Loop
DECLARE
   v_counter NUMBER := 1;
BEGIN
   WHILE v_counter <= 5 LOOP
      DBMS_OUTPUT.PUT_LINE('Iteration: ' || v_counter);
      v_counter := v_counter + 1;
   END LOOP;
END;
/

SELECT * FROM Patient;
```

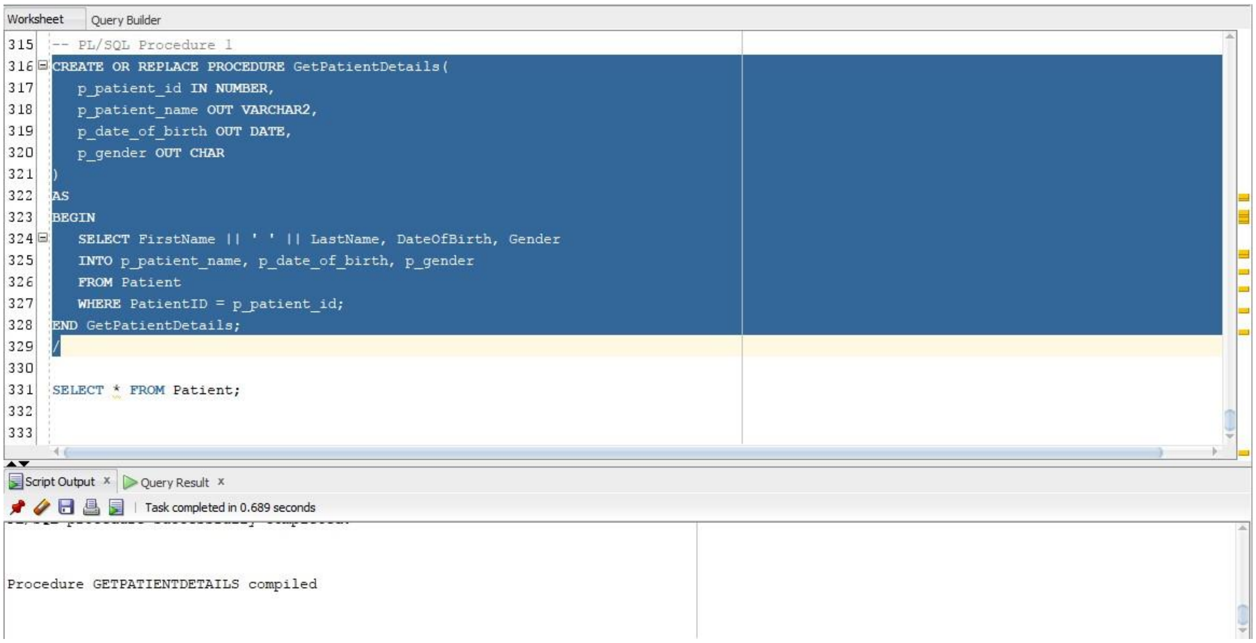
Task completed in 0.037 seconds

```
Iteration: 1
Iteration: 2
Iteration: 3
Iteration: 4
Iteration: 5


PL/SQL procedure successfully completed.
```

Figure 38: PL/SQL program 4: While Loop

7) Create three PL/SQL procedure with in and out parameters.

This PL/SQL block retrieves patient details by calling a stored procedure GetPatientDetails with parameters v_patient_id, v_patient_name, v_date_of_birth, and v_gender. It then displays the retrieved patient details, including name, date of birth, and gender, using DBMS_OUTPUT.PUT_LINE.



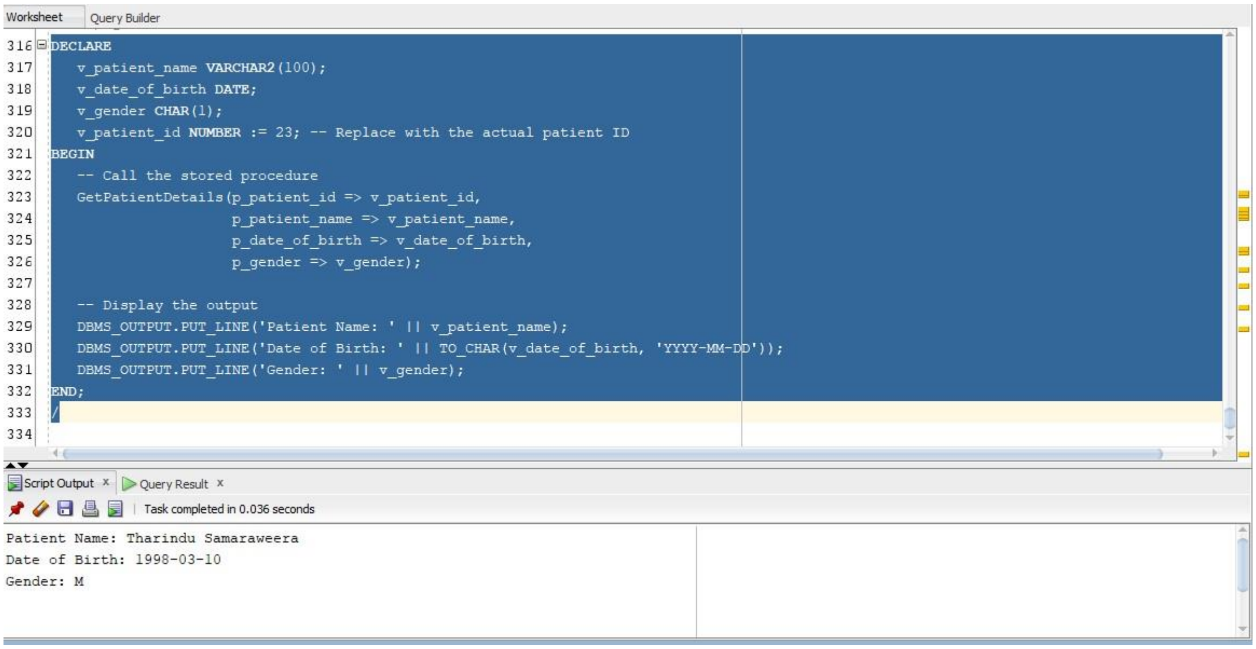
```
-- PL/SQL Procedure 1
CREATE OR REPLACE PROCEDURE GetPatientDetails(
   p_patient_id IN NUMBER,
   p_patient_name OUT VARCHAR2,
   p_date_of_birth OUT DATE,
   p_gender OUT CHAR
)
AS
BEGIN
   SELECT FirstName || ' ' || LastName, DateOfBirth, Gender
   INTO p_patient_name, p_date_of_birth, p_gender
   FROM Patient
   WHERE PatientID = p_patient_id;
END GetPatientDetails;
/

SELECT * FROM Patient;
```

Task completed in 0.689 seconds

```
PL/SQL procedure successfully completed.


Procedure GETPATIENTDETAILS compiled
```

Figure 39: PL/SQL Procedure 1

This PL/SQL block calls a stored procedure named GetPatientDetails, passing in the patient ID v_patient_id. The procedure retrieves and assigns the patient's name to v_patient_name, date of birth to v_date_of_birth, and gender to v_gender. Subsequently, it displays the retrieved patient details using DBMS_OUTPUT.PUT_LINE.

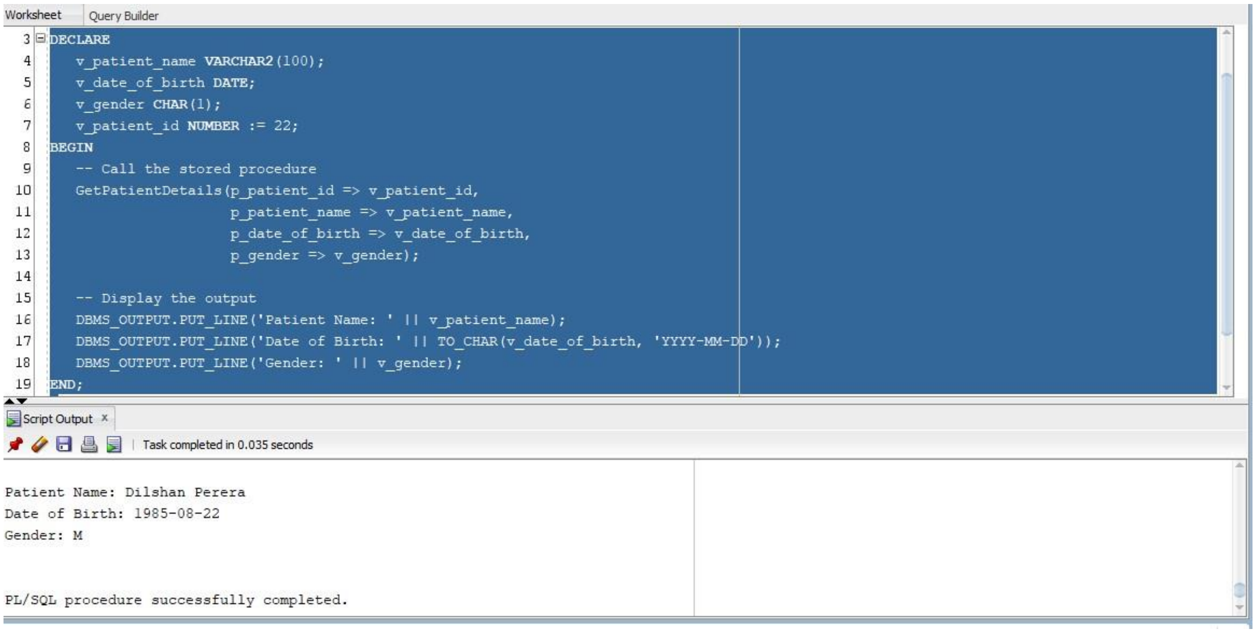


```
DECLARE
   v_patient_name VARCHAR2(100);
   v_date_of_birth DATE;
   v_gender CHAR(1);
   v_patient_id NUMBER := 23; -- Replace with the actual patient ID
BEGIN
   -- Call the stored procedure
   GetPatientDetails(p_patient_id => v_patient_id,
                     p_patient_name => v_patient_name,
                     p_date_of_birth => v_date_of_birth,
                     p_gender => v_gender);

   -- Display the output
   DBMS_OUTPUT.PUT_LINE('Patient Name: ' || v_patient_name);
   DBMS_OUTPUT.PUT_LINE('Date of Birth: ' || TO_CHAR(v_date_of_birth, 'YYYY-MM-DD'));
   DBMS_OUTPUT.PUT_LINE('Gender: ' || v_gender);
END;
/
```

```
Patient Name: Tharindu Samaraweera
Date of Birth: 1998-03-10
Gender: M
```

Figure 40: PL/SQL Procedure 2

This PL/SQL block invokes the stored procedure GetPatientDetails, passing the patient ID v_patient_id as an argument. The procedure retrieves the patient's name, date of birth, and gender, storing them in variables v_patient_name, v_date_of_birth, and v_gender, respectively. Finally, it outputs these details using DBMS_OUTPUT.PUT_LINE.



```
DECLARE
   v_patient_name VARCHAR2(100);
   v_date_of_birth DATE;
   v_gender CHAR(1);
   v_patient_id NUMBER := 22;
BEGIN
   -- Call the stored procedure
   GetPatientDetails(p_patient_id => v_patient_id,
                     p_patient_name => v_patient_name,
                     p_date_of_birth => v_date_of_birth,
                     p_gender => v_gender);

   -- Display the output
   DBMS_OUTPUT.PUT_LINE('Patient Name: ' || v_patient_name);
   DBMS_OUTPUT.PUT_LINE('Date of Birth: ' || TO_CHAR(v_date_of_birth, 'YYYY-MM-DD'));
   DBMS_OUTPUT.PUT_LINE('Gender: ' || v_gender);
END;
```

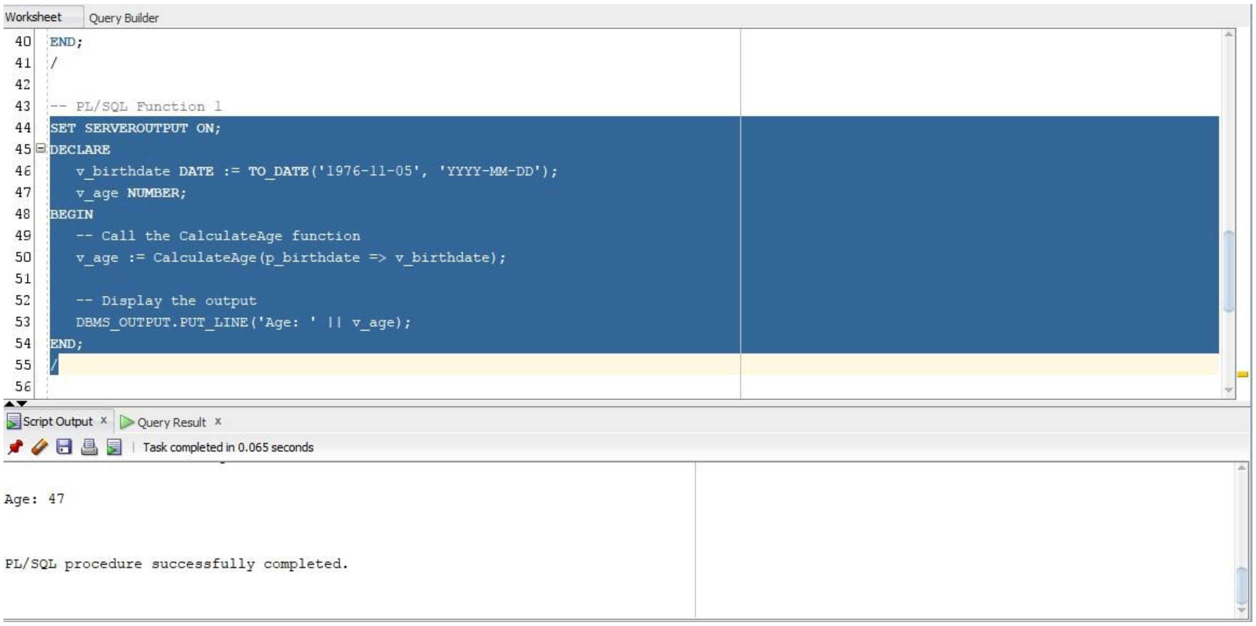
```
Patient Name: Dilshan Perera
Date of Birth: 1985-08-22
Gender: M


PL/SQL procedure successfully completed.
```

Figure 41: PL/SQL Procedure

3

8) Create three PL/SQL Function.

This PL/SQL block demonstrates the use of a PL/SQL function named CalculateAge. It calculates the age based on the provided birthdate v_birthdate. The calculated age is then assigned to the variable v_age, which is subsequently outputted using DBMS_OUTPUT.PUT_LINE.

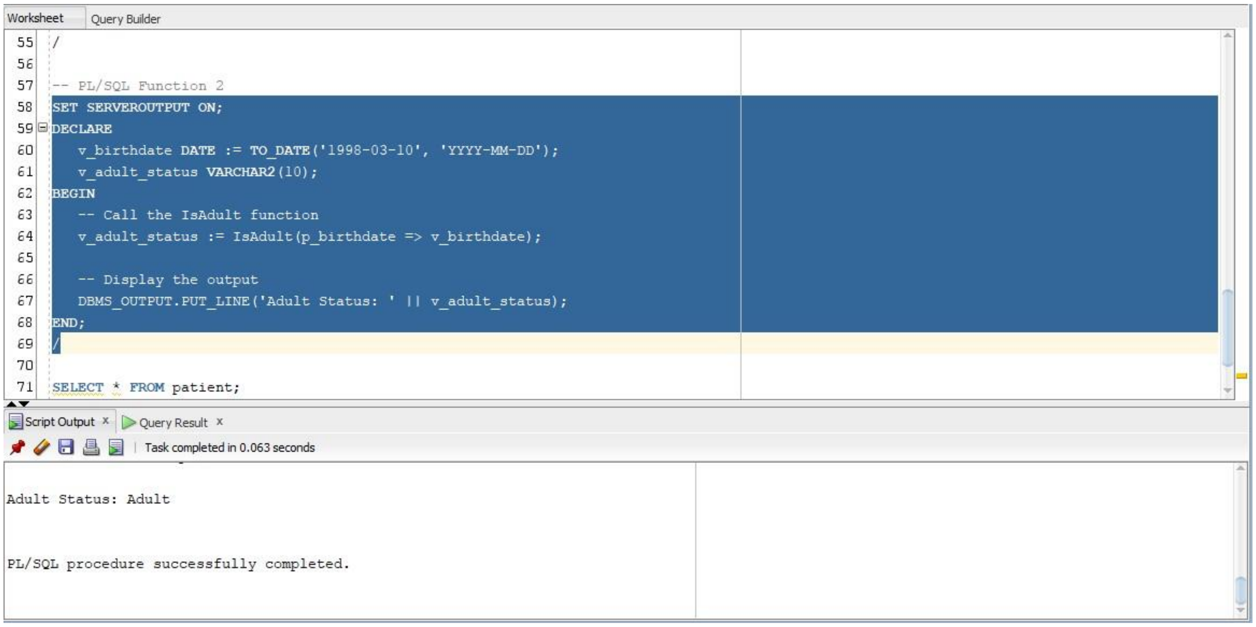


```
END;
/

-- PL/SQL Function 1
SET SERVEROUTPUT ON;
DECLARE
   v_birthdate DATE := TO_DATE('1976-11-05', 'YYYY-MM-DD');
   v_age NUMBER;
BEGIN
   -- Call the CalculateAge function
   v_age := CalculateAge(p_birthdate => v_birthdate);

   -- Display the output
   DBMS_OUTPUT.PUT_LINE('Age: ' || v_age);
END;
/
```

```
Age: 47


PL/SQL procedure successfully completed.
```

Figure 42: PL/SQL Function 1

This PL/SQL block utilizes a PL/SQL function named IsAdult to determine the adult status based on the provided birthdate v_birthdate. The function returns a VARCHAR2 value indicating whether the person is an adult or not. The result is assigned to the variable v_adult_status and then printed using DBMS_OUTPUT.PUT_LINE.

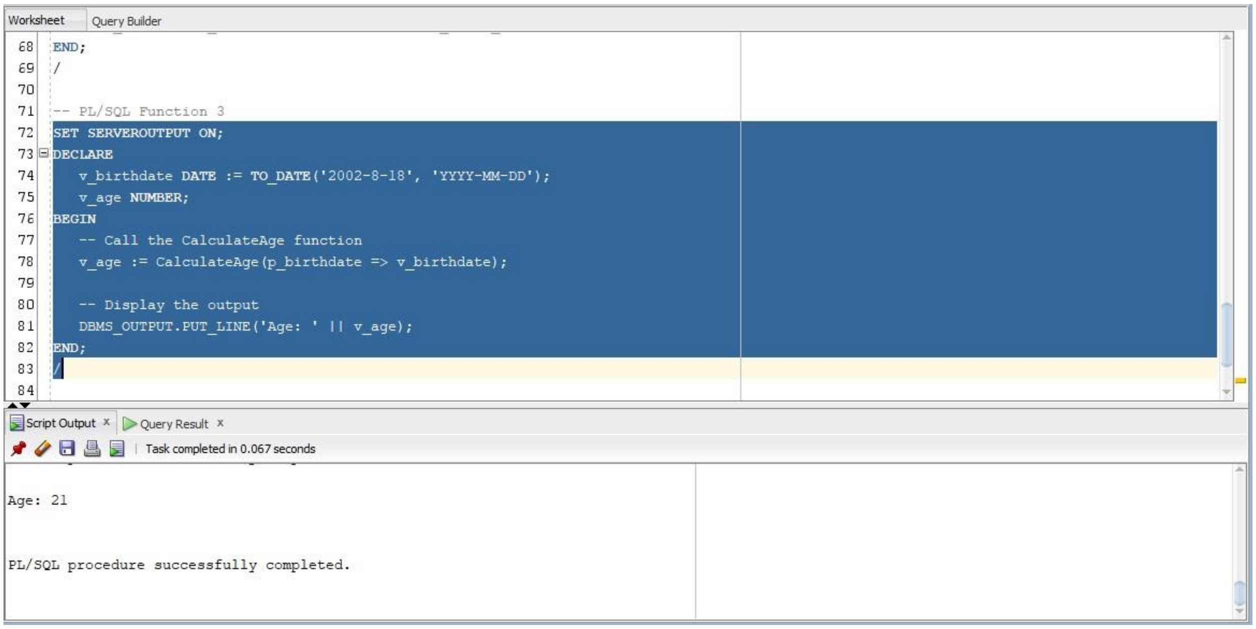


```
/

-- PL/SQL Function 2
SET SERVEROUTPUT ON;
DECLARE
   v_birthdate DATE := TO_DATE('1998-03-10', 'YYYY-MM-DD');
   v_adult_status VARCHAR2(10);
BEGIN
   -- Call the IsAdult function
   v_adult_status := IsAdult(p_birthdate => v_birthdate);

   -- Display the output
   DBMS_OUTPUT.PUT_LINE('Adult Status: ' || v_adult_status);
END;
/

SELECT * FROM patient;
```

```
Adult Status: Adult


PL/SQL procedure successfully completed.
```

Figure 43: PL/SQL Function 2

This PL/SQL block demonstrates the utilization of the CalculateAge function, which calculates the age based on the provided birthdate v_birthdate. The calculated age is then assigned to the variable v_age, and it's subsequently displayed using DBMS_OUTPUT.PUT_LINE.



```
END;
/

-- PL/SQL Function 3
SET SERVEROUTPUT ON;
DECLARE
   v_birthdate DATE := TO_DATE('2002-8-18', 'YYYY-MM-DD');
   v_age NUMBER;
BEGIN
   -- Call the CalculateAge function
   v_age := CalculateAge(p_birthdate => v_birthdate);

   -- Display the output
   DBMS_OUTPUT.PUT_LINE('Age: ' || v_age);
END;
/

```

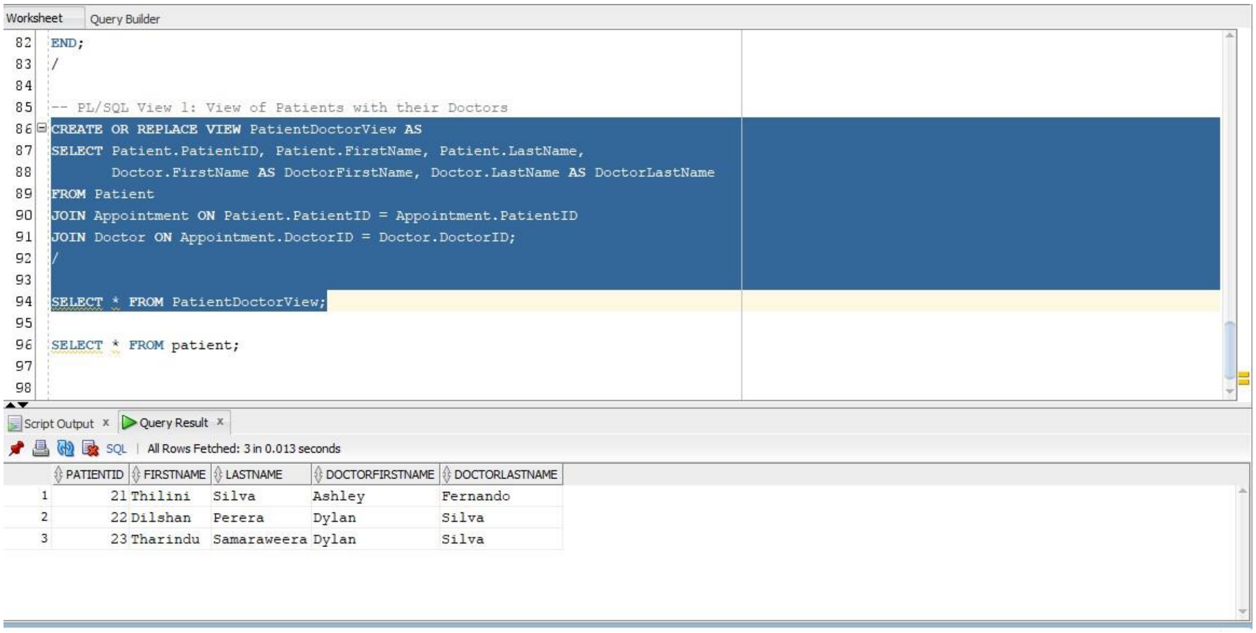
```
Age: 21


PL/SQL procedure successfully completed.
```

Figure 44: PL/SQL Function

9) Create three PL/SQL view.

This SQL script defines a view named PatientDoctorView, which combines data from the Patient, Appointment, and Doctor tables. It selects patient details along with the first and last names of their corresponding doctors by joining these tables based on the PatientID and DoctorID attributes.

Finally, it executes a query to retrieve and display the data from the PatientDoctorView.



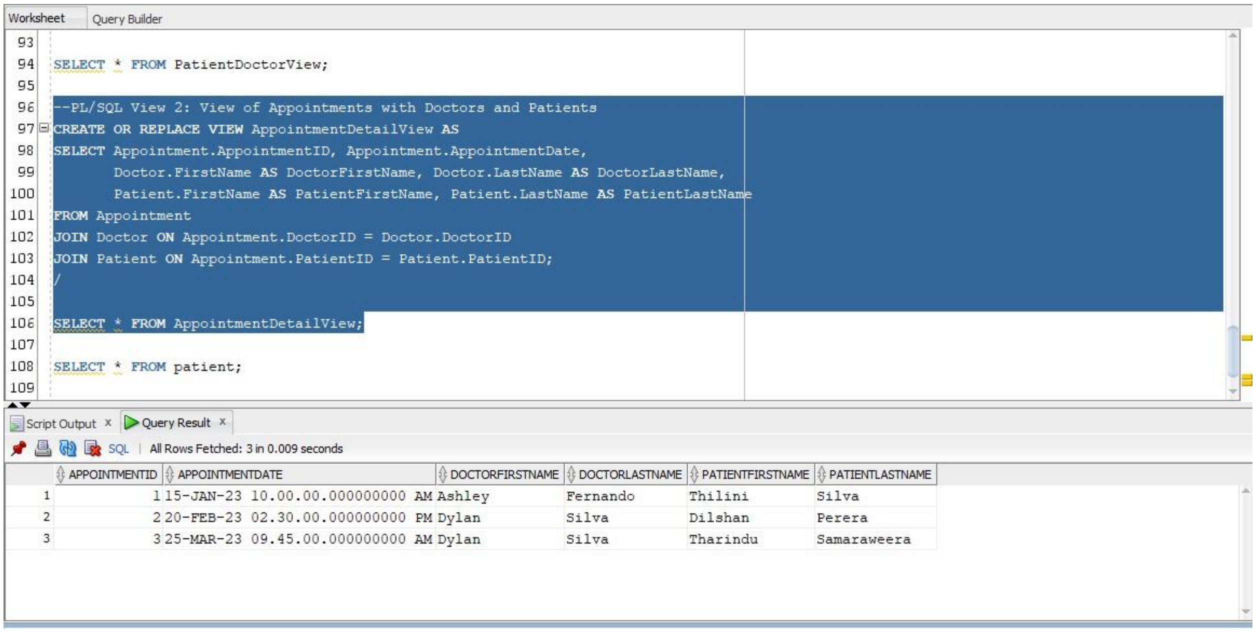
```
END;
/

-- PL/SQL View 1: View of Patients with their Doctors
CREATE OR REPLACE VIEW PatientDoctorView AS
SELECT Patient.PatientID, Patient.FirstName, Patient.LastName,
       Doctor.FirstName AS DoctorFirstName, Doctor.LastName AS DoctorLastName
FROM Patient
JOIN Appointment ON Patient.PatientID = Appointment.PatientID
JOIN Doctor ON Appointment.DoctorID = Doctor.DoctorID;
/

SELECT * FROM PatientDoctorView;

SELECT * FROM patient;
```

All Rows Fetched: 3 in 0.013 seconds

	PATIENTID	FIRSTNAME	LASTNAME	DOCTORFIRSTNAME	DOCTORLASTNAME
1	21	Thilini	Silva	Ashley	Fernando
2	22	Dilshan	Perera	Dylan	Silva
3	23	Tharindu	Samaraweera	Dylan	Silva

Figure 45: PL/SQL View 1: View of Patients with their Doctors

The SQL script defines a view called AppointmentDetailView, amalgamating data from the Appointment, Doctor, and Patient tables. It selects appointment details such as ID and date, alongside the first and last names of the associated doctor and patient by joining these tables. Finally, a query is executed to retrieve and display the data from the AppointmentDetailView view.

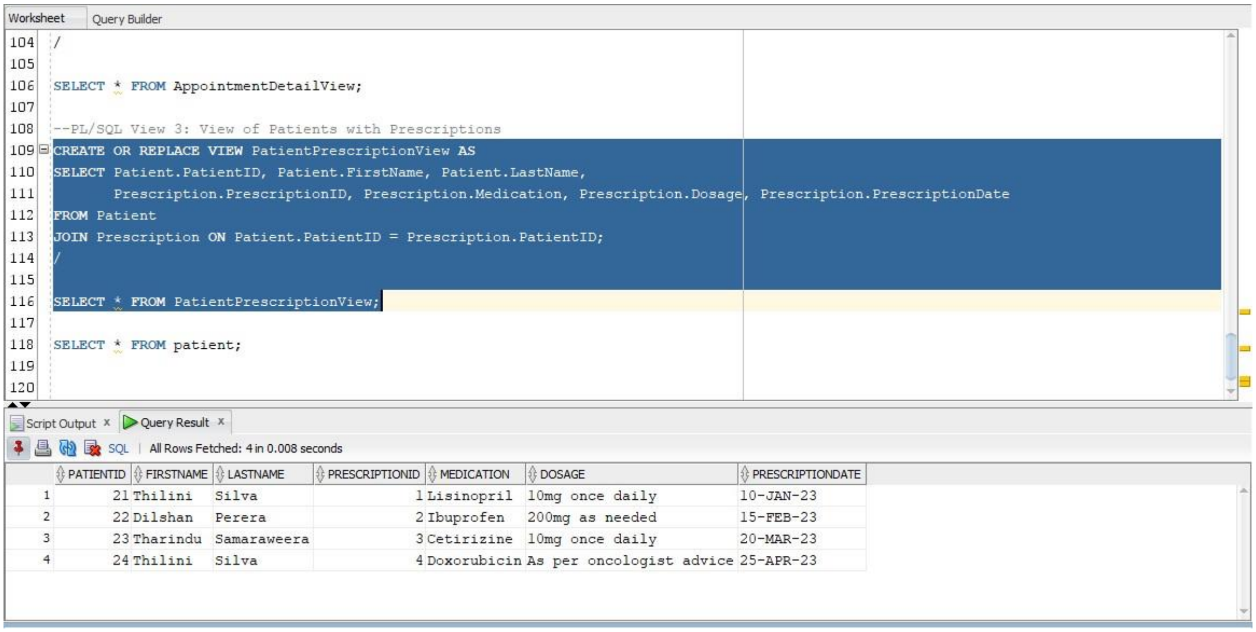


```sql

SELECT * FROM PatientDoctorView;

--PL/SQL View 2: View of Appointments with Doctors and Patients
CREATE OR REPLACE VIEW AppointmentDetailView AS
SELECT Appointment.AppointmentID, Appointment.AppointmentDate,
       Doctor.FirstName AS DoctorFirstName, Doctor.LastName AS DoctorLastName,
       Patient.FirstName AS PatientFirstName, Patient.LastName AS PatientLastName
FROM Appointment
JOIN Doctor ON Appointment.DoctorID = Doctor.DoctorID
JOIN Patient ON Appointment.PatientID = Patient.PatientID;
/

SELECT * FROM AppointmentDetailView;

SELECT * FROM patient;

```

```
    APPOINTMENTID APPOINTMENTDATE                  DOCTORFIRSTNAME DOCTORLASTNAME PATIENTFIRSTNAME PATIENTLASTNAME
1               1 15-JAN-23 10.00.00.000000000 AM Ashley          Fernando       Thilini          Silva
2               2 20-FEB-23 02.30.00.000000000 PM Dylan           Silva          Dilshan          Perera
3               3 25-MAR-23 09.45.00.000000000 AM Dylan           Silva          Tharindu         Samaraweera
```

Figure 46: PL/SQL View 2: View of Appointments with Doctors and Patients

The SQL script creates or replaces a view named PatientPrescriptionView, consolidating data from the Patient and Prescription tables. It selects patient details such as ID, first name, and last name, along with prescription details like ID, medication, dosage, and prescription date. The view is then queried to retrieve and display the combined data.



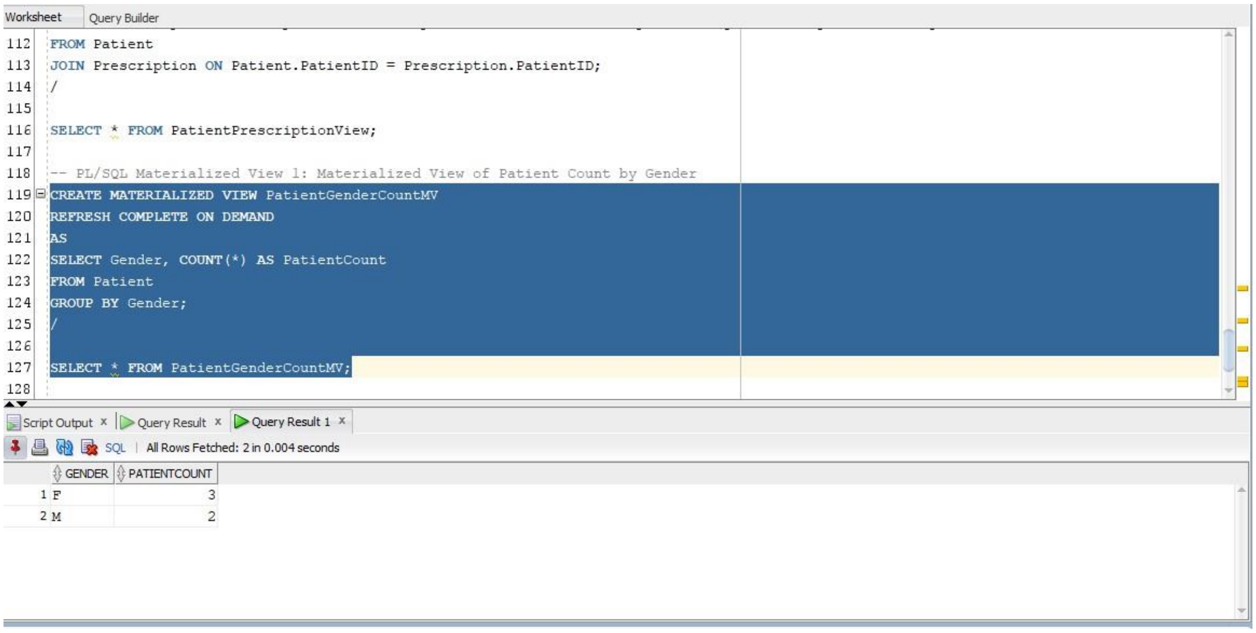
```sql
/

SELECT * FROM AppointmentDetailView;

--PL/SQL View 3: View of Patients with Prescriptions
CREATE OR REPLACE VIEW PatientPrescriptionView AS
SELECT Patient.PatientID, Patient.FirstName, Patient.LastName,
       Prescription.PrescriptionID, Prescription.Medication, Prescription.Dosage, Prescription.PrescriptionDate
FROM Patient
JOIN Prescription ON Patient.PatientID = Prescription.PatientID;
/

SELECT * FROM PatientPrescriptionView;

SELECT * FROM patient;


```

```
    PATIENTID FIRSTNAME LASTNAME    PRESCRIPTIONID MEDICATION  DOSAGE                     PRESCRIPTIONDATE
1          21 Thilini   Silva                    1 Lisinopril  10mg once daily            10-JAN-23
2          22 Dilshan   Perera                   2 Ibuprofen   200mg as needed            15-FEB-23
3          23 Tharindu  Samaraweera              3 Cetirizine  10mg once daily            20-MAR-23
4          24 Thilini   Silva                    4 Doxorubicin As per oncologist advice 25-APR-23
```

Figure 47: PL/SQL View 3: View of Patients with Prescriptions

10) Create three PL/SQL materialized view.

The SQL script creates a materialized view named PatientGenderCountMV, refreshing it completely upon demand. It calculates and stores the count of patients based on their gender, aggregating data from the Patient table. Finally, it queries the materialized view to display the gender-wise patient count.

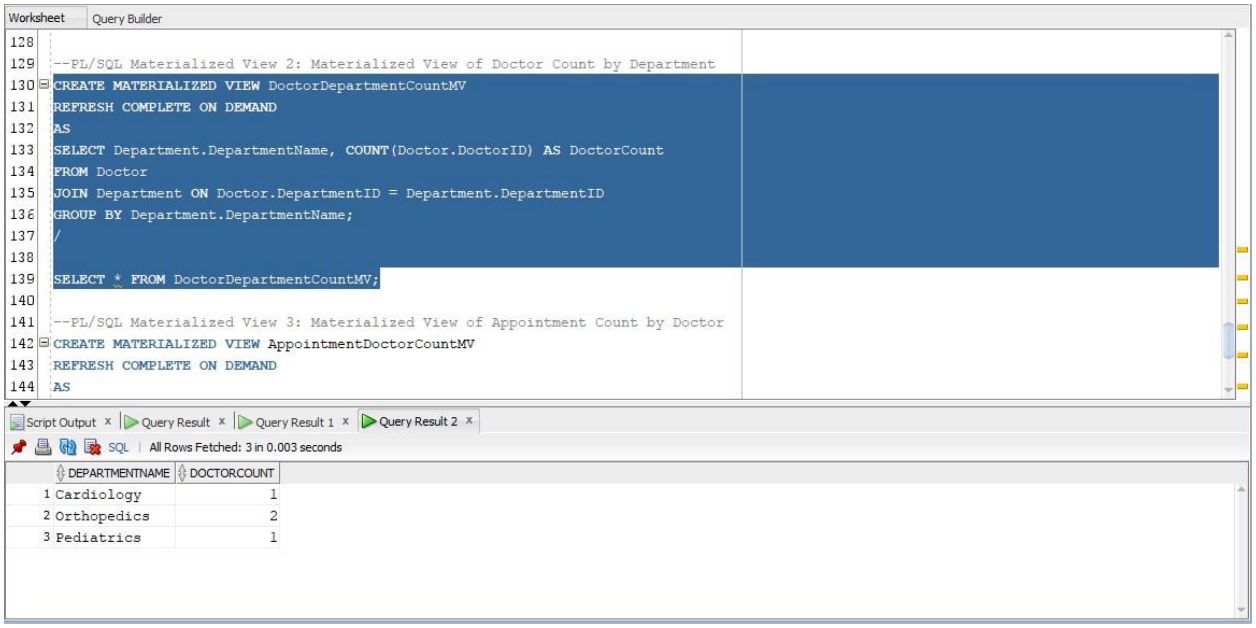


```sql
FROM Patient
JOIN Prescription ON Patient.PatientID = Prescription.PatientID;
/

SELECT * FROM PatientPrescriptionView;

-- PL/SQL Materialized View 1: Materialized View of Patient Count by Gender
CREATE MATERIALIZED VIEW PatientGenderCountMV
REFRESH COMPLETE ON DEMAND
AS
SELECT Gender, COUNT(*) AS PatientCount
FROM Patient
GROUP BY Gender;
/

SELECT * FROM PatientGenderCountMV;
```

```
All Rows Fetched: 2 in 0.004 seconds

    GENDER  PATIENTCOUNT
  1 F                  3
  2 M                  2
```

Figure 48: PL/SQL Materialized View 1: Materialized View of Patient Count by Gender

The SQL script creates a materialized view named DoctorDepartmentCountMV, which refreshes completely upon demand. It calculates and stores the count of doctors per department, aggregating data from the Doctor and Department tables. Finally, it queries the materialized view to display the departmentwise doctor count.



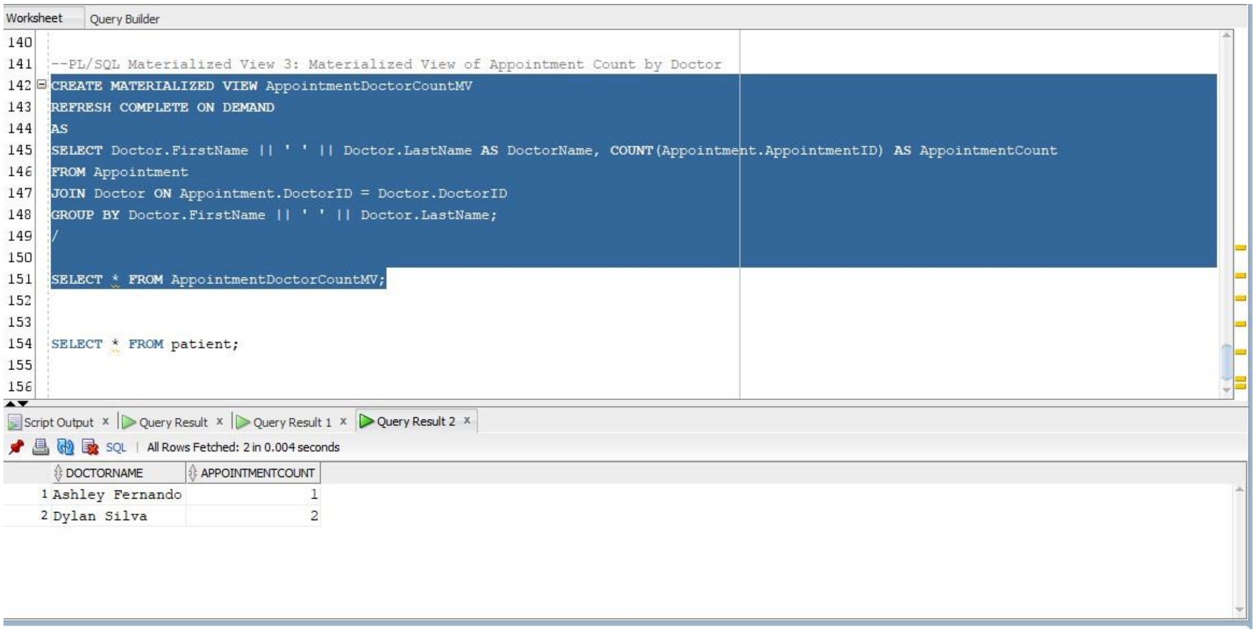
```
--PL/SQL Materialized View 2: Materialized View of Doctor Count by Department
CREATE MATERIALIZED VIEW DoctorDepartmentCountMV
REFRESH COMPLETE ON DEMAND
AS
SELECT Department.DepartmentName, COUNT(Doctor.DoctorID) AS DoctorCount
FROM Doctor
JOIN Department ON Doctor.DepartmentID = Department.DepartmentID
GROUP BY Department.DepartmentName;
/

SELECT * FROM DoctorDepartmentCountMV;

--PL/SQL Materialized View 3: Materialized View of Appointment Count by Doctor
CREATE MATERIALIZED VIEW AppointmentDoctorCountMV
REFRESH COMPLETE ON DEMAND
AS
```

All Rows Fetched: 3 in 0.003 seconds

	DEPARTMENTNAME	DOCTORCOUNT
1	Cardiology	1
2	Orthopedics	2
3	Pediatrics	1

Figure 49: PL/SQL Materialized View 2: Materialized View of Doctor Count by Department

The SQL script creates a materialized view named AppointmentDoctorCountMV, which refreshes completely upon demand. It calculates and stores the count of appointments for each doctor by aggregating data from the Appointment and Doctor tables. Finally, it queries the materialized view to display the count of appointments per doctor.



```

--PL/SQL Materialized View 3: Materialized View of Appointment Count by Doctor
CREATE MATERIALIZED VIEW AppointmentDoctorCountMV
REFRESH COMPLETE ON DEMAND
AS
SELECT Doctor.FirstName || ' ' || Doctor.LastName AS DoctorName, COUNT(Appointment.AppointmentID) AS AppointmentCount
FROM Appointment
JOIN Doctor ON Appointment.DoctorID = Doctor.DoctorID
GROUP BY Doctor.FirstName || ' ' || Doctor.LastName;
/

SELECT * FROM AppointmentDoctorCountMV;


SELECT * FROM patient;

```

All Rows Fetched: 2 in 0.004 seconds

	DOCTORNAME	APPOINTMENTCOUNT
1	Ashley Fernando	1
2	Dylan Silva	2

Figure 50: PL/SQL Materialized View 3: Materialized View of Appointment Count by Doctor