Topic: Week 16 Final Project Report

Student's Full Name: - Shashank Satish Kohade
Course Title: - INFO 579: SQL/NoSQL Databases for Data and Information Sciences
Term name and year: - Fall 2024
Submission Week: - Week 16 Final Project Report
Instructor's Name: - Dr. Nayem Rahman
Date of Submission: - December 04, 2024

Clinic Management System

Project Overview:

The purpose of this project is to create a data management system tailored for a small clinic that provides various health services, including diagnostic tests, treatments, and medication management. The current method of managing patient information, treatment records, and diagnostic results is manual, leading to inefficiencies and a higher likelihood of errors, which can compromise patient care. By automating these processes, the small clinic aims to enhance operational efficiency, improve data accuracy, and ultimately deliver better patient care. This initiative will focus on effectively collecting, storing, and organizing data pertaining to patients, healthcare providers, treatments, diagnostic tests, and medical equipment.

Data and Source:

The project will leverage detailed records concerning patients, healthcare providers, treatments administered, diagnostic tests conducted, and the medical equipment utilized within the clinic. These records are vital for constructing a robust data management system that can elevate operational efficiency and enhance patient care services. The primary source for this data will be Kaggle, a well-known repository for diverse datasets. A selection of datasets from Kaggle has been carefully chosen to encompass all relevant aspects of the clinic's operations.

The collected data will include:

- **Patient Records:** Comprehensive information such as patient names, contact details, addresses, visit dates, treatment requirements, assigned healthcare providers, and prescribed medications.

- **Diagnostic Procedures:** An overview of the types of tests performed at the clinic, including blood tests, blood pressure monitoring, cholesterol assessments, cardiovascular screenings, respiratory evaluations, weight management consultations, and pain management assessments.

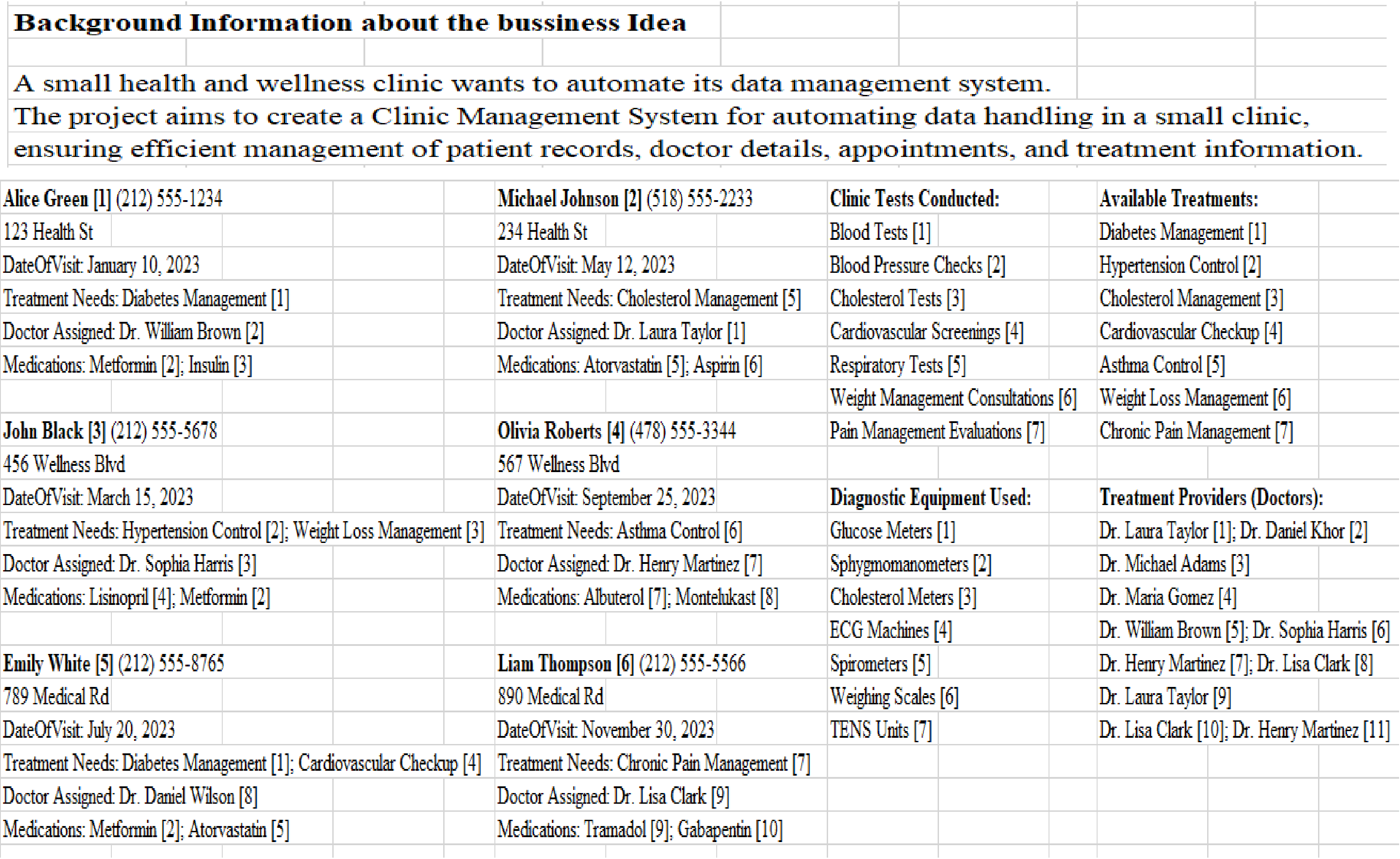
- **Diagnostic Equipment:** Information regarding the tools and equipment employed in the clinic, such as glucose meters, blood pressure cuffs, cholesterol testing devices, ECG machines, spirometers, weighing scales, and TENS units.

- **Available Treatments:** Descriptions of the treatments offered by the clinic, including diabetes management, hypertension control, cholesterol management, cardiovascular evaluations, asthma management, weight loss programs, and chronic pain management.

- **Healthcare Providers (Doctors):** Data about the clinic's doctors, detailing their specializations and the treatments they provide.

By implementing this data management system, the small clinic aims to significantly improve its operational efficiency and the quality of care provided to patients. Enhanced management of patient information will lead to greater accuracy and efficiency, fostering better overall performance for the clinic.

Data:

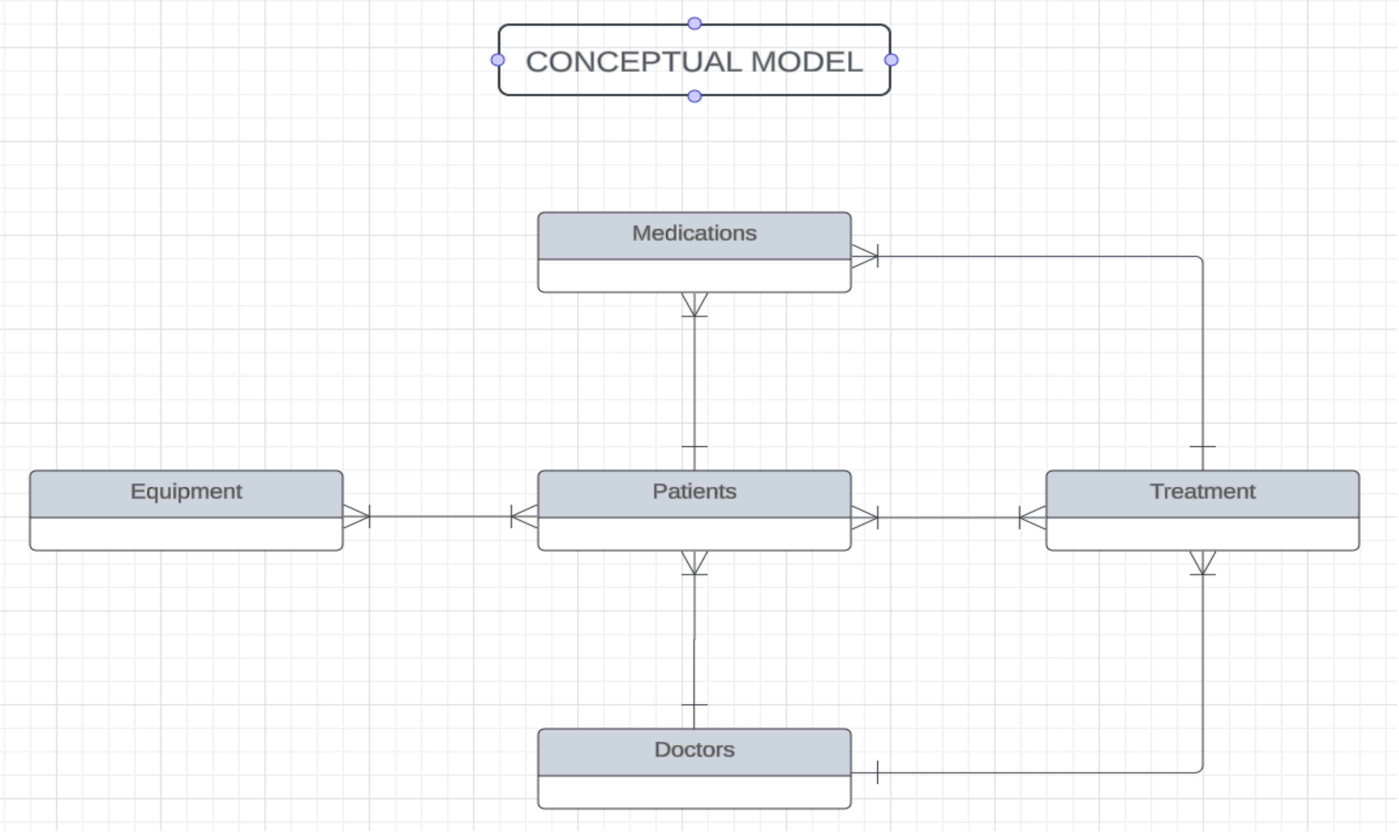


Background Information about the bussiness Idea

A small health and wellness clinic wants to automate its data management system.
The project aims to create a Clinic Management System for automating data handling in a small clinic, ensuring efficient management of patient records, doctor details, appointments, and treatment information.

Alice Green [1] (212) 555-1234	**Michael Johnson [2]** (518) 555-2233	**Clinic Tests Conducted:**	**Available Treatments:**
123 Health St	234 Health St	Blood Tests [1]	Diabetes Management [1]
DateOfVisit: January 10, 2023	DateOfVisit: May 12, 2023	Blood Pressure Checks [2]	Hypertension Control [2]
Treatment Needs: Diabetes Management [1]	Treatment Needs: Cholesterol Management [5]	Cholesterol Tests [3]	Cholesterol Management [3]
Doctor Assigned: Dr. William Brown [2]	Doctor Assigned: Dr. Laura Taylor [1]	Cardiovascular Screenings [4]	Cardiovascular Checkup [4]
Medications: Metformin [2]; Insulin [3]	Medications: Atorvastatin [5]; Aspirin [6]	Respiratory Tests [5]	Asthma Control [5]
		Weight Management Consultations [6]	Weight Loss Management [6]
John Black [3] (212) 555-5678	**Olivia Roberts [4]** (478) 555-3344	Pain Management Evaluations [7]	Chronic Pain Management [7]
456 Wellness Blvd	567 Wellness Blvd		
DateOfVisit: March 15, 2023	DateOfVisit: September 25, 2023	**Diagnostic Equipment Used:**	**Treatment Providers (Doctors):**
Treatment Needs: Hypertension Control [2]; Weight Loss Management [3]	Treatment Needs: Asthma Control [6]	Glucose Meters [1]	Dr. Laura Taylor [1]; Dr. Daniel Khor [2]
Doctor Assigned: Dr. Sophia Harris [3]	Doctor Assigned: Dr. Henry Martinez [7]	Sphygmomanometers [2]	Dr. Michael Adams [3]
Medications: Lisinopril [4]; Metformin [2]	Medications: Albuterol [7]; Montelukast [8]	Cholesterol Meters [3]	Dr. Maria Gomez [4]
		ECG Machines [4]	Dr. William Brown [5]; Dr. Sophia Harris [6]
Emily White [5] (212) 555-8765	**Liam Thompson [6]** (212) 555-5566	Spirometers [5]	Dr. Henry Martinez [7]; Dr. Lisa Clark [8]
789 Medical Rd	890 Medical Rd	Weighing Scales [6]	Dr. Laura Taylor [9]
DateOfVisit: July 20, 2023	DateOfVisit: November 30, 2023	TENS Units [7]	Dr. Lisa Clark [10]; Dr. Henry Martinez [11]
Treatment Needs: Diabetes Management [1]; Cardiovascular Checkup [4]	Treatment Needs: Chronic Pain Management [7]		
Doctor Assigned: Dr. Daniel Wilson [8]	Doctor Assigned: Dr. Lisa Clark [9]		
Medications: Metformin [2]; Atorvastatin [5]	Medications: Tramadol [9]; Gabapentin [10]		

3. **Develop a Conceptual Model with 5 or 6 entities in it. Make sure you have at least one many-to-many relationship that exists in your conceptual model. Explain with data why it's a many-to-many relationship.**



Based on the data, I'll define five entities: Patients, Doctors, Treatments, Medications, and Equipment. Each entity represents a key part of the clinic management system.

Entities and Relationships:

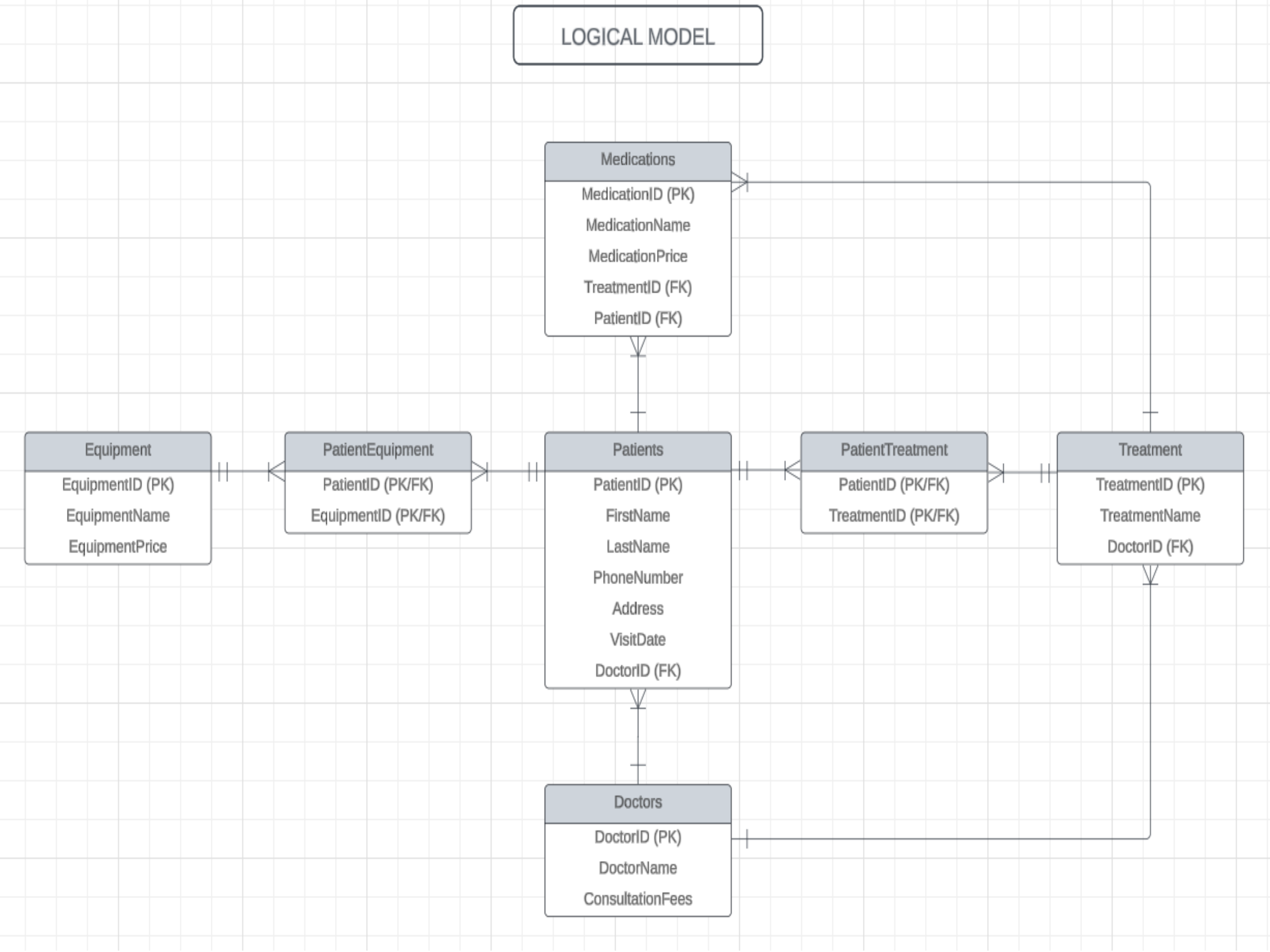
- **Patients:** Represents individuals visiting the clinic, with details like contact information and visit date.
- **Doctors:** Represents medical professionals in the clinic who are assigned to patients and specialize in specific treatments.
- **Treatments:** Represents medical treatments or specialties that address patient needs, such as Diabetes Management, Hypertension Control, and Cardiovascular Checkups.
- **Medications:** Represents the medications prescribed to patients based on their treatments.
- **Equipment:** Represents the diagnostic or therapeutic equipment used in the clinic for various treatments or patient evaluations.

Relationships:

- **Doctor-to-Patient (One-to-Many):**
  - Each doctor is responsible for treating multiple patients, but each patient is assigned to one primary doctor for their consultation. This means that a single doctor can manage the treatment needs of multiple patients, while each patient typically has one doctor overseeing their care.
- **Patient-to-Treatment (Many-to-Many):**
  - Patients can require multiple treatments (e.g., a patient may need both Diabetes Management and Cardiovascular Checkups). At the same time, each treatment can be applied to multiple patients, as several patients might need the same treatment type.
  - Data Explanation: For example, "Alice Green" has a treatment need for Diabetes Management, while "Emily White" has treatment needs for both Diabetes Management and Cardiovascular Checkup. Additionally, Diabetes Management is required by multiple patients. This demonstrates the many-to-many nature of this relationship, as each patient can have multiple treatments, and each treatment can serve multiple patients.
- **Doctor-to-Treatment (One-to-Many):**
  - Each doctor can provide multiple treatments, but each treatment is uniquely associated with a single doctor. This means that one doctor may handle multiple types of treatments, but each treatment is exclusively managed by one doctor.
- **Patient-to-Medication (One-to-Many):**
  - Each patient may be prescribed multiple medications, but each medication is typically associated with a single patient at a given time. This is a one-to-many relationship, where one patient can have many medications, but each medication is linked to one specific patient.
- **Patient-to-Equipment (Many-to-Many):**
  - Multiple patients may use the same piece of equipment, and each patient may require various equipment types for their treatments.

- Data Explanation: For example, if "Alice Green" requires a Glucose Meter and a Sphygmomanometer for monitoring her diabetes and blood pressure, and "John Black" also needs the Sphygmomanometer to monitor his hypertension, this indicates that: Alice Green uses multiple pieces of equipment (Glucose Meter, Sphygmomanometer). John Black also uses the Sphygmomanometer, showing that a single piece of equipment can be used by multiple patients.

4. Develop a Logical Model using the Conceptual Model. Make sure you come up with a junction entity to resolve the many-to-many relationship.



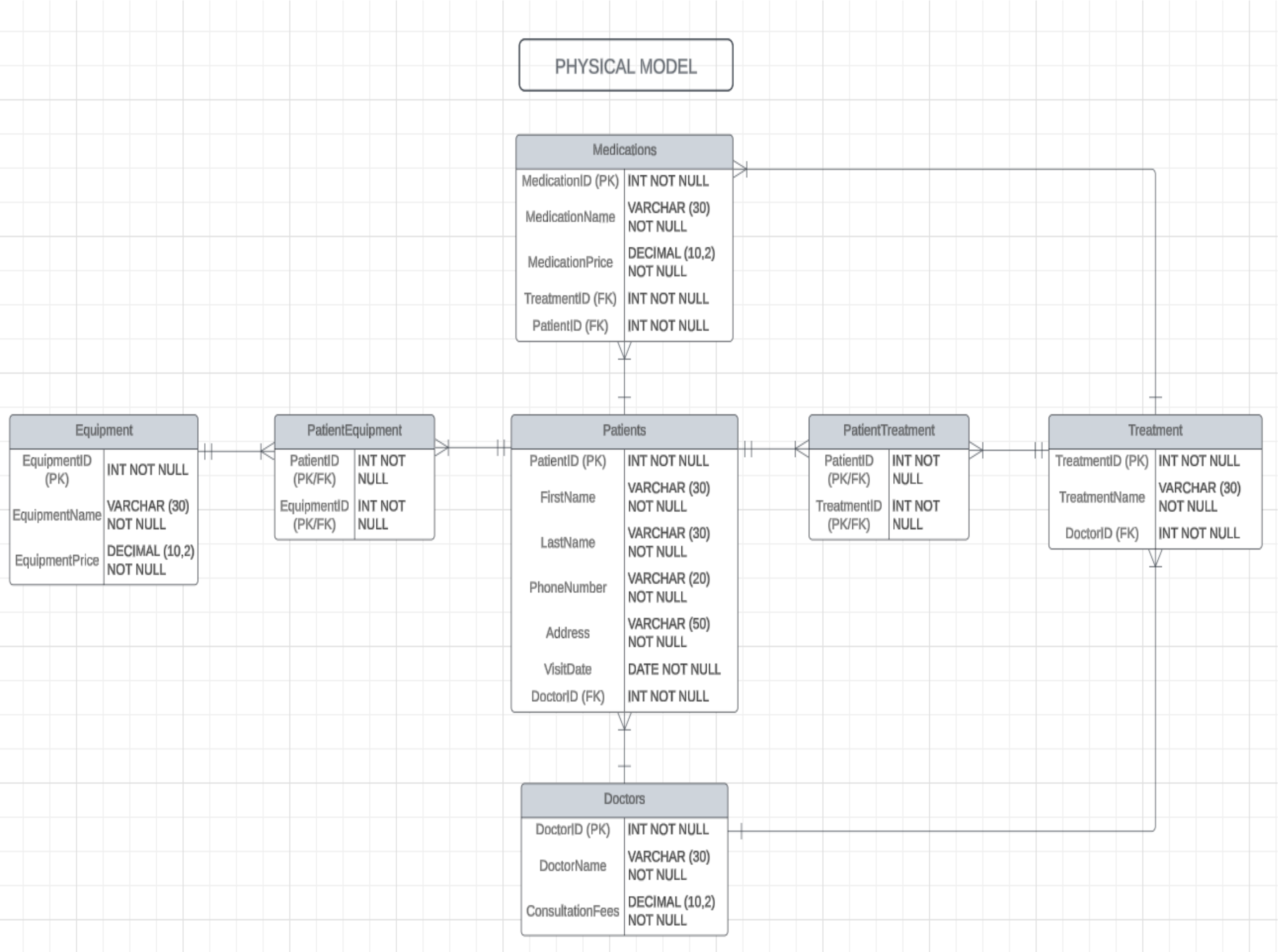
In the above Logical Model, I have included PatientEquipment and PatientTreatment as Junction Entities to resolve the many to many relationships.

Junction Entities:

1. **PatientTreatment Table** (resolves the many-to-many relationship between Patients and Treatments):
   - Attributes: PatientID (FK), TreatmentID (FK)
   - Primary Key: Composite key of PatientID and TreatmentID
   - Purpose: Allows each patient to be linked to multiple treatments, and each treatment to be linked to multiple patients.

2. **PatientEquipment Table** (resolves the many-to-many relationship between Patients and Equipment):
   - Attributes: PatientID (FK), EquipmentID (FK)
   - Primary Key: Composite key of PatientID and EquipmentID
   - Purpose: Allows patients to use multiple pieces of equipment and for each equipment to be used by multiple patients.

5. **Develop the physical model based on the Logical Model.**



Medications	
MedicationID (PK)	INT NOT NULL
MedicationName	VARCHAR (30) NOT NULL
MedicationPrice	DECIMAL (10,2) NOT NULL
TreatmentID (FK)	INT NOT NULL
PatientID (FK)	INT NOT NULL

Equipment	
EquipmentID (PK)	INT NOT NULL
EquipmentName	VARCHAR (30) NOT NULL
EquipmentPrice	DECIMAL (10,2) NOT NULL

PatientEquipment	
PatientID (PK/FK)	INT NOT NULL
EquipmentID (PK/FK)	INT NOT NULL

Patients	
PatientID (PK)	INT NOT NULL
FirstName	VARCHAR (30) NOT NULL
LastName	VARCHAR (30) NOT NULL
PhoneNumber	VARCHAR (20) NOT NULL
Address	VARCHAR (50) NOT NULL
VisitDate	DATE NOT NULL
DoctorID (FK)	INT NOT NULL

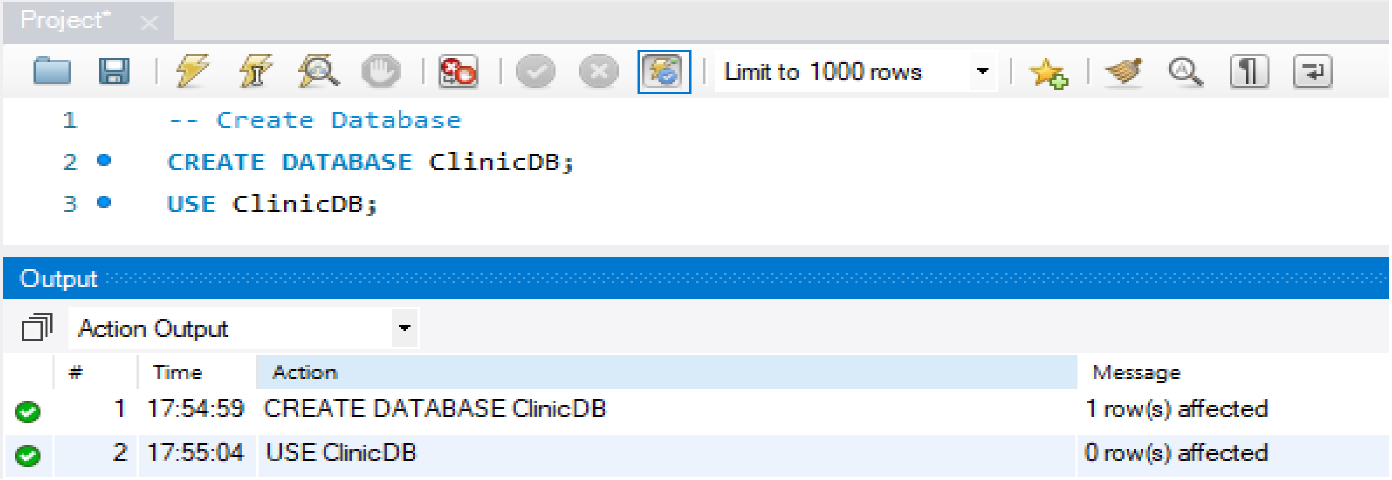
PatientTreatment	
PatientID (PK/FK)	INT NOT NULL
TreatmentID (PK/FK)	INT NOT NULL

Treatment	
TreatmentID (PK)	INT NOT NULL
TreatmentName	VARCHAR (30) NOT NULL
DoctorID (FK)	INT NOT NULL

Doctors	
DoctorID (PK)	INT NOT NULL
DoctorName	VARCHAR (30) NOT NULL
ConsultationFees	DECIMAL (10,2) NOT NULL

6. **Create tables using a database system. Insert data into the database tables. You must provide the DDL (CREATE TABLE statements), INSERT statements, and SELECT statements.**
Details: Create the tables that you have come up with (the table must be based on the Physical Model).

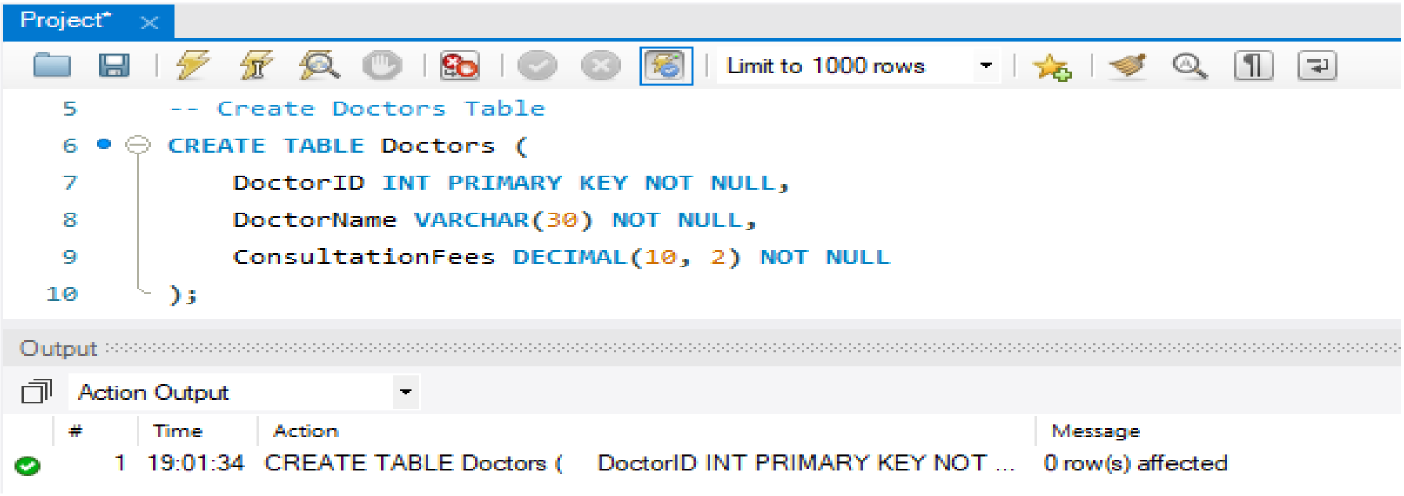
CREATE –



```
-- Create Database
CREATE DATABASE ClinicDB;
USE ClinicDB;
```

	#	Time	Action	Message
✓	1	17:54:59	CREATE DATABASE ClinicDB	1 row(s) affected
✓	2	17:55:04	USE ClinicDB	0 row(s) affected

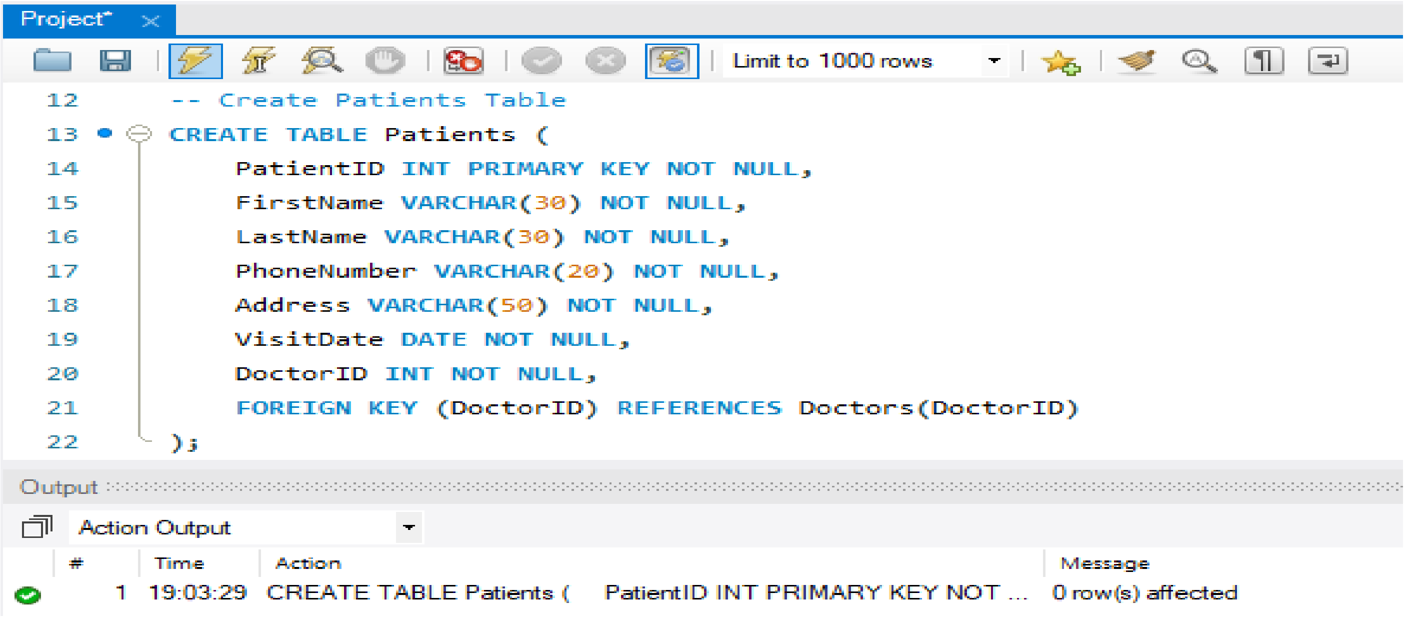
In this screenshot, the **ClinicDB** database has been successfully created, and the **USE** command has been executed to select this database for further operations.



```
-- Create Doctors Table
CREATE TABLE Doctors (
    DoctorID INT PRIMARY KEY NOT NULL,
    DoctorName VARCHAR(30) NOT NULL,
    ConsultationFees DECIMAL(10, 2) NOT NULL
);
```

```
  #   Time      Action                                                      Message
  1   19:01:34  CREATE TABLE Doctors (   DoctorID INT PRIMARY KEY NOT ...  0 row(s) affected
```

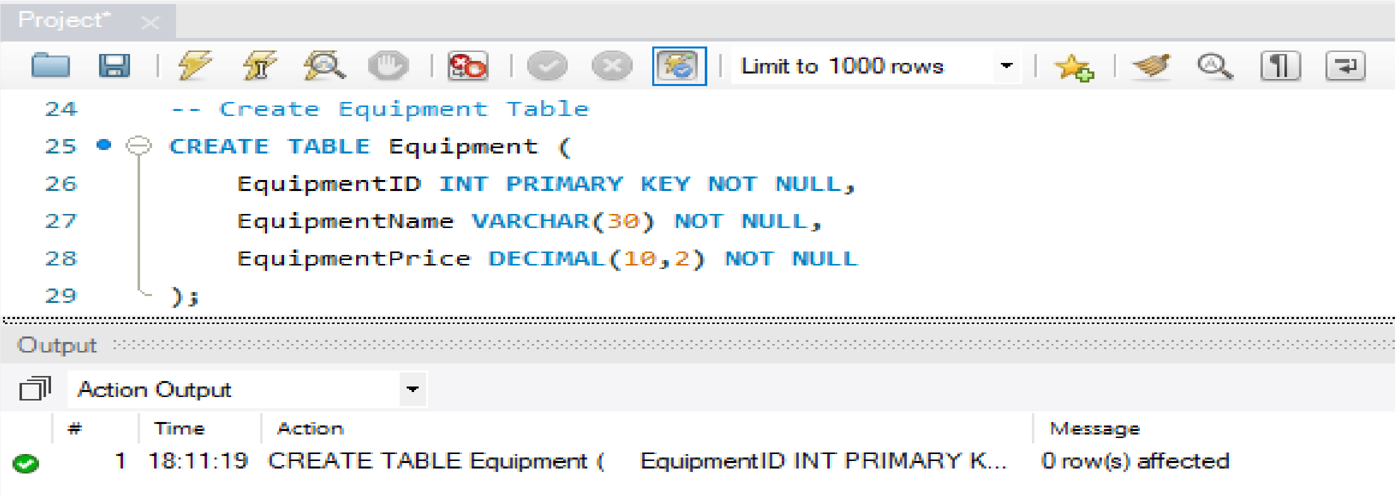
In this screenshot, the **Doctors** table has been created in the **ClinicDB** database using the **CREATE** command.



```
-- Create Patients Table
CREATE TABLE Patients (
    PatientID INT PRIMARY KEY NOT NULL,
    FirstName VARCHAR(30) NOT NULL,
    LastName VARCHAR(30) NOT NULL,
    PhoneNumber VARCHAR(20) NOT NULL,
    Address VARCHAR(50) NOT NULL,
    VisitDate DATE NOT NULL,
    DoctorID INT NOT NULL,
    FOREIGN KEY (DoctorID) REFERENCES Doctors(DoctorID)
);
```

```
  #   Time      Action                                                        Message
  1   19:03:29  CREATE TABLE Patients (   PatientID INT PRIMARY KEY NOT ...  0 row(s) affected
```

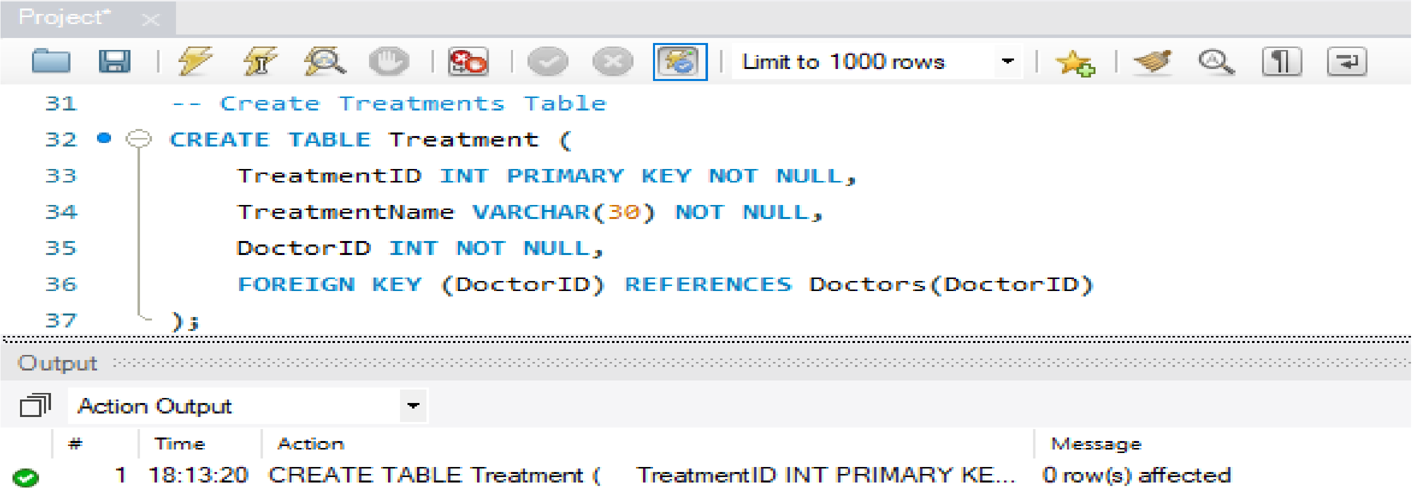
In this screenshot, the **Patients** table has been created in the **ClinicDB** database using the **CREATE** command.



```
-- Create Equipment Table
CREATE TABLE Equipment (
    EquipmentID INT PRIMARY KEY NOT NULL,
    EquipmentName VARCHAR(30) NOT NULL,
    EquipmentPrice DECIMAL(10,2) NOT NULL
);
```

```
  #   Time      Action                                                     Message
  1   18:11:19  CREATE TABLE Equipment (   EquipmentID INT PRIMARY K...   0 row(s) affected
```

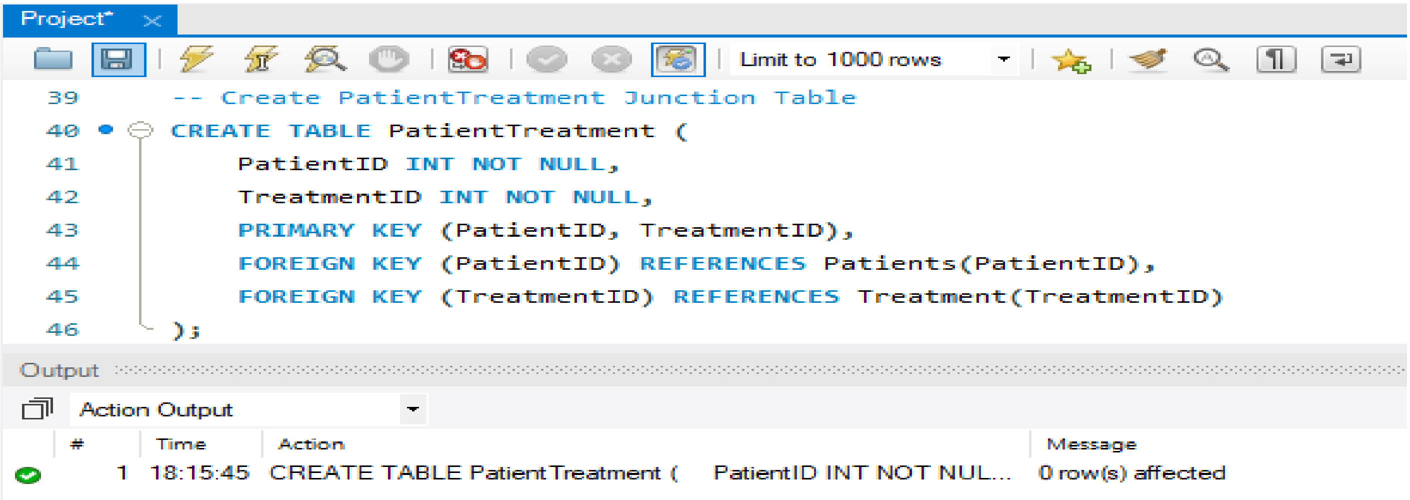
In this screenshot, the **Equipment** table has been created in the **ClinicDB** database using the **CREATE** command.



```sql
    -- Create Treatments Table
    CREATE TABLE Treatment (
        TreatmentID INT PRIMARY KEY NOT NULL,
        TreatmentName VARCHAR(30) NOT NULL,
        DoctorID INT NOT NULL,
        FOREIGN KEY (DoctorID) REFERENCES Doctors(DoctorID)
    );
```

```
  #  Time      Action                        Message
  1  18:13:20  CREATE TABLE Treatment (     TreatmentID INT PRIMARY KE...  0 row(s) affected
```

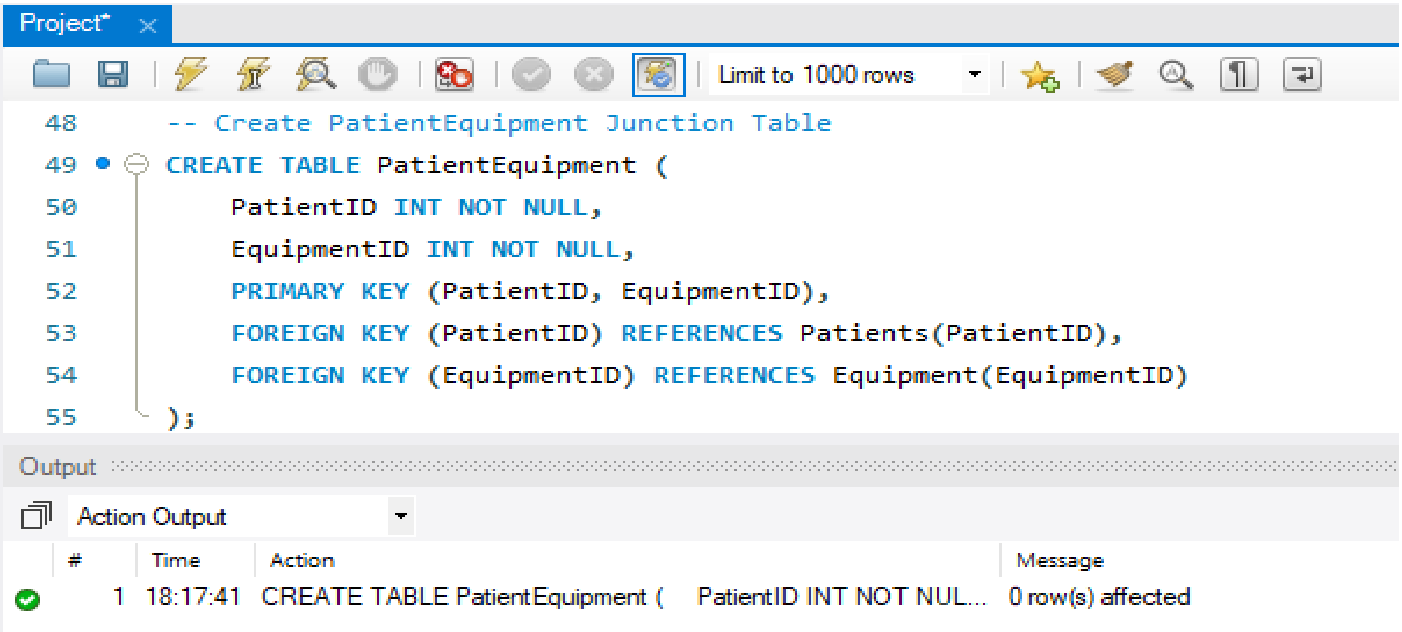
In this screenshot, the **Treatment** table has been created in the **ClinicDB** database using the **CREATE** command.



```sql
    -- Create PatientTreatment Junction Table
    CREATE TABLE PatientTreatment (
        PatientID INT NOT NULL,
        TreatmentID INT NOT NULL,
        PRIMARY KEY (PatientID, TreatmentID),
        FOREIGN KEY (PatientID) REFERENCES Patients(PatientID),
        FOREIGN KEY (TreatmentID) REFERENCES Treatment(TreatmentID)
    );
```

```
  #  Time      Action                                Message
  1  18:15:45  CREATE TABLE PatientTreatment (     PatientID INT NOT NUL...  0 row(s) affected
```

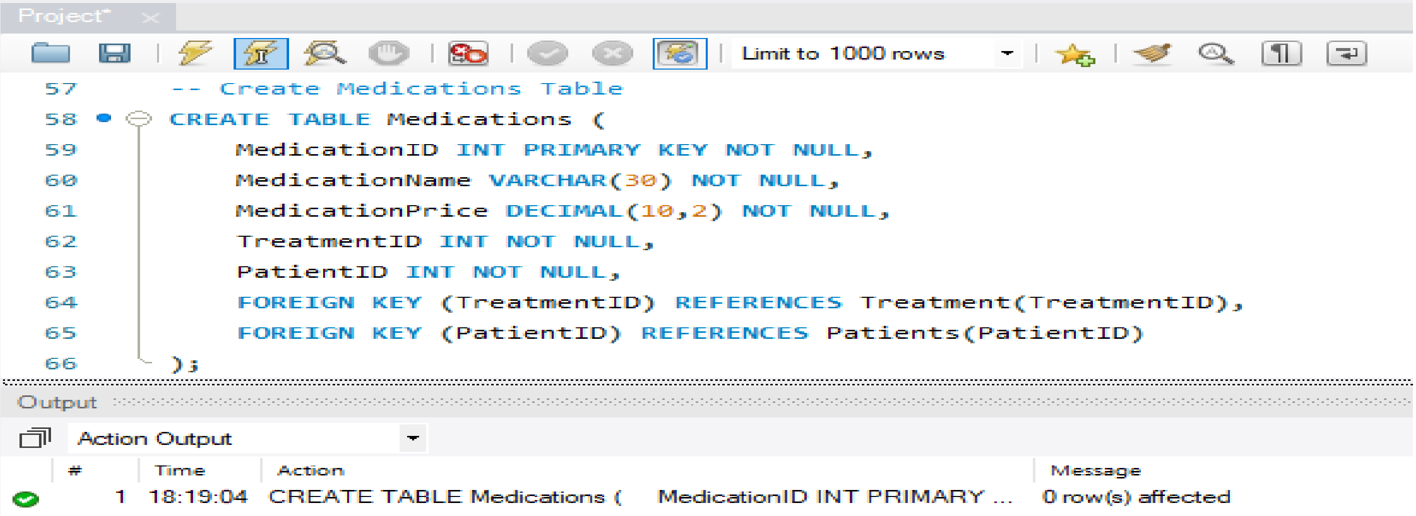
In this screenshot, the **PatientTreatment** table has been created in the **ClinicDB** database using the **CREATE** command.



```sql
    -- Create PatientEquipment Junction Table
    CREATE TABLE PatientEquipment (
        PatientID INT NOT NULL,
        EquipmentID INT NOT NULL,
        PRIMARY KEY (PatientID, EquipmentID),
        FOREIGN KEY (PatientID) REFERENCES Patients(PatientID),
        FOREIGN KEY (EquipmentID) REFERENCES Equipment(EquipmentID)
    );
```

```
  #  Time      Action                                Message
  1  18:17:41  CREATE TABLE PatientEquipment (     PatientID INT NOT NUL...  0 row(s) affected
```

In this screenshot, the **PatientTreatment** table has been created in the **ClinicDB** database using the **CREATE** command.

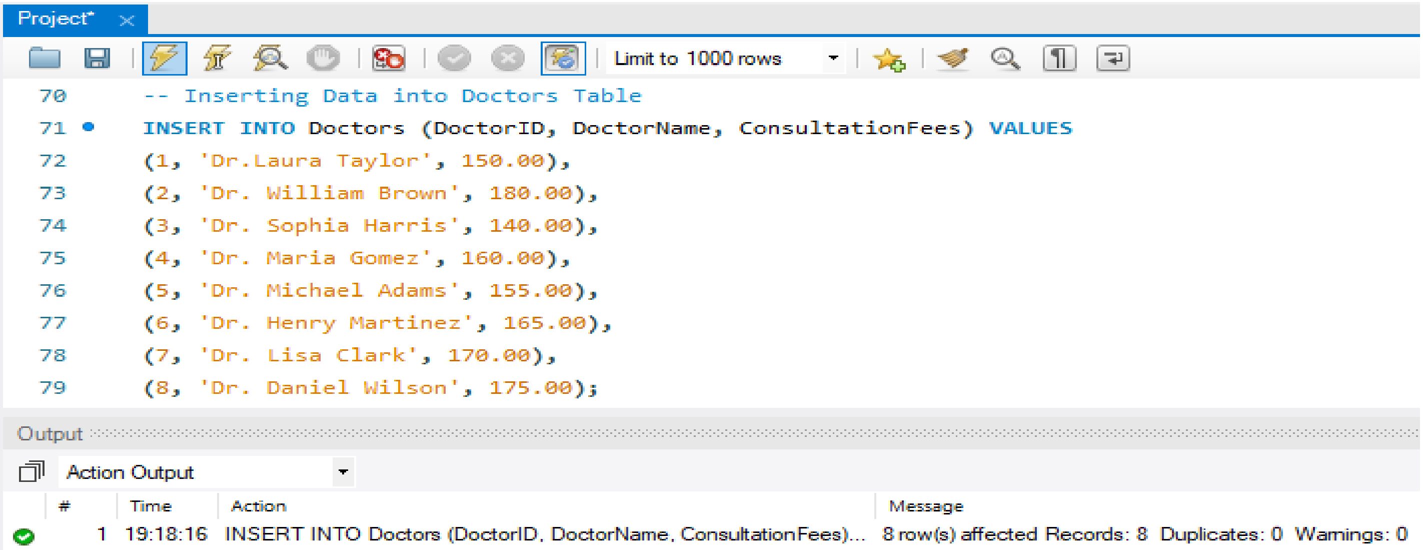


```sql
-- Create Medications Table
CREATE TABLE Medications (
    MedicationID INT PRIMARY KEY NOT NULL,
    MedicationName VARCHAR(30) NOT NULL,
    MedicationPrice DECIMAL(10,2) NOT NULL,
    TreatmentID INT NOT NULL,
    PatientID INT NOT NULL,
    FOREIGN KEY (TreatmentID) REFERENCES Treatment(TreatmentID),
    FOREIGN KEY (PatientID) REFERENCES Patients(PatientID)
);
```

```
   #   Time       Action                                                      Message
✓  1   18:19:04   CREATE TABLE Medications (    MedicationID INT PRIMARY ...   0 row(s) affected
```

In this screenshot, the **Medications** table has been created in the **ClinicDB** database using the **CREATE** command.

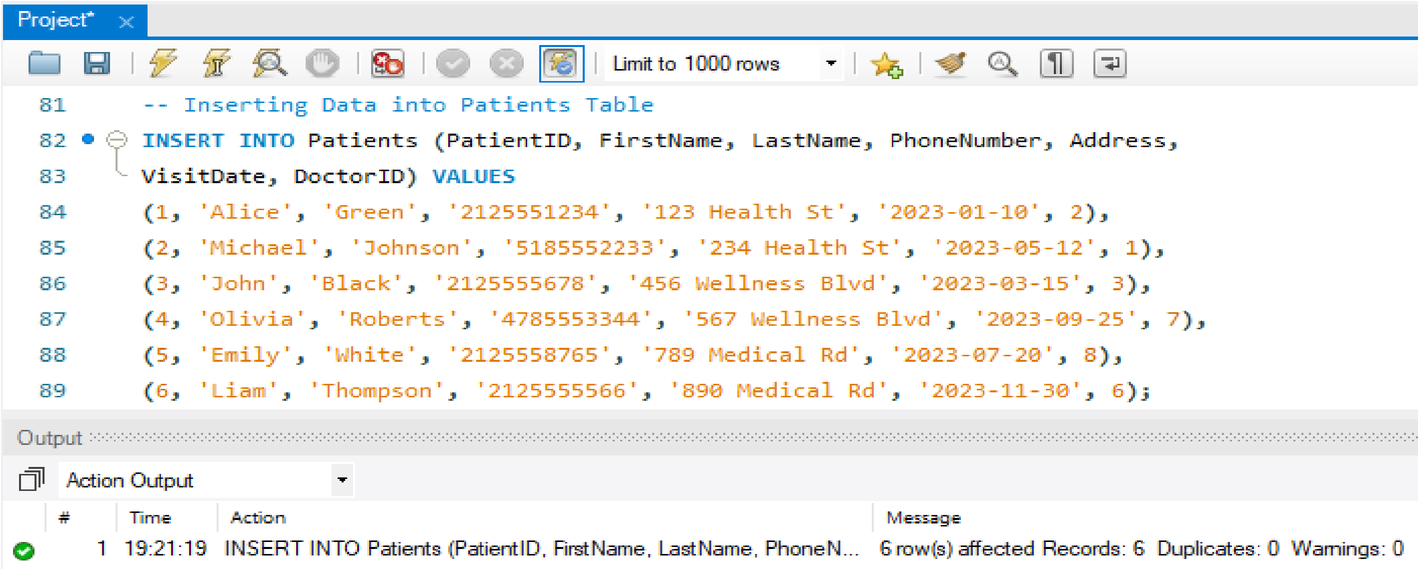
INSERT –



```sql
-- Inserting Data into Doctors Table
INSERT INTO Doctors (DoctorID, DoctorName, ConsultationFees) VALUES
(1, 'Dr.Laura Taylor', 150.00),
(2, 'Dr. William Brown', 180.00),
(3, 'Dr. Sophia Harris', 140.00),
(4, 'Dr. Maria Gomez', 160.00),
(5, 'Dr. Michael Adams', 155.00),
(6, 'Dr. Henry Martinez', 165.00),
(7, 'Dr. Lisa Clark', 170.00),
(8, 'Dr. Daniel Wilson', 175.00);
```

```
   #   Time       Action                                                         Message
✓  1   19:18:16   INSERT INTO Doctors (DoctorID, DoctorName, ConsultationFees)...   8 row(s) affected Records: 8  Duplicates: 0  Warnings: 0
```

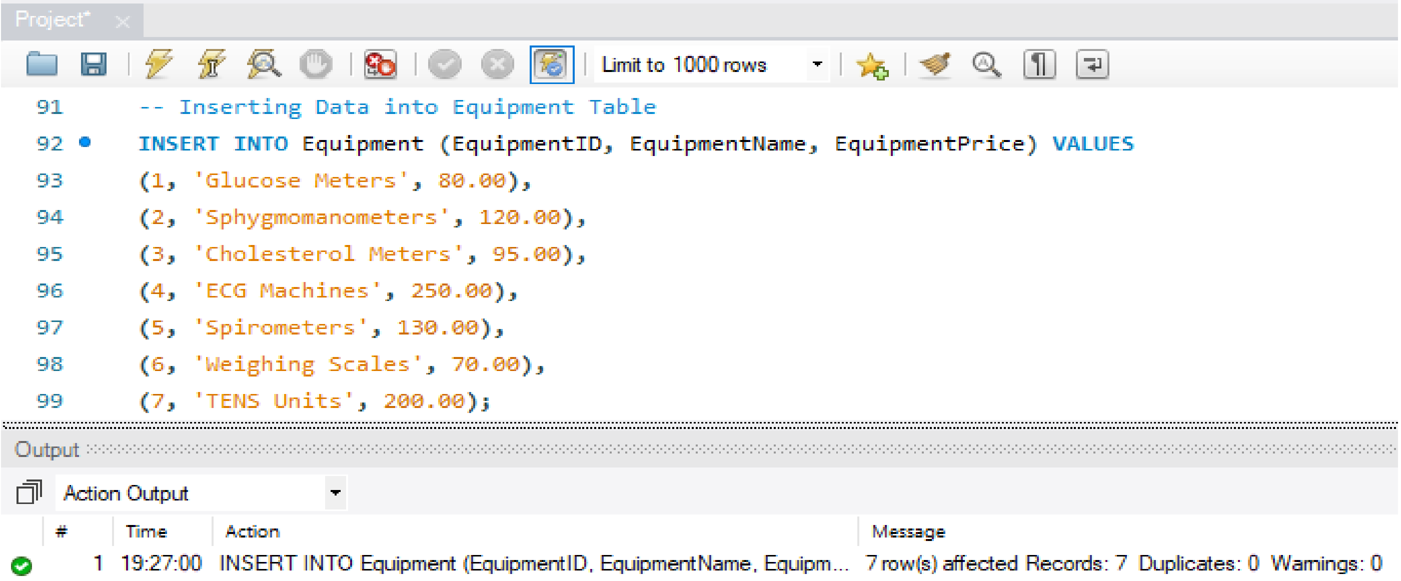
In this screenshot, data has been inserted into the **Doctors** table in the **ClinicDB** database using the **INSERT** command.



```sql
-- Inserting Data into Patients Table
INSERT INTO Patients (PatientID, FirstName, LastName, PhoneNumber, Address,
VisitDate, DoctorID) VALUES
(1, 'Alice', 'Green', '2125551234', '123 Health St', '2023-01-10', 2),
(2, 'Michael', 'Johnson', '5185552233', '234 Health St', '2023-05-12', 1),
(3, 'John', 'Black', '2125555678', '456 Wellness Blvd', '2023-03-15', 3),
(4, 'Olivia', 'Roberts', '4785553344', '567 Wellness Blvd', '2023-09-25', 7),
(5, 'Emily', 'White', '2125558765', '789 Medical Rd', '2023-07-20', 8),
(6, 'Liam', 'Thompson', '2125555566', '890 Medical Rd', '2023-11-30', 6);
```

```
   #   Time       Action                                                      Message
✓  1   19:21:19   INSERT INTO Patients (PatientID, FirstName, LastName, PhoneN...   6 row(s) affected Records: 6  Duplicates: 0  Warnings: 0
```

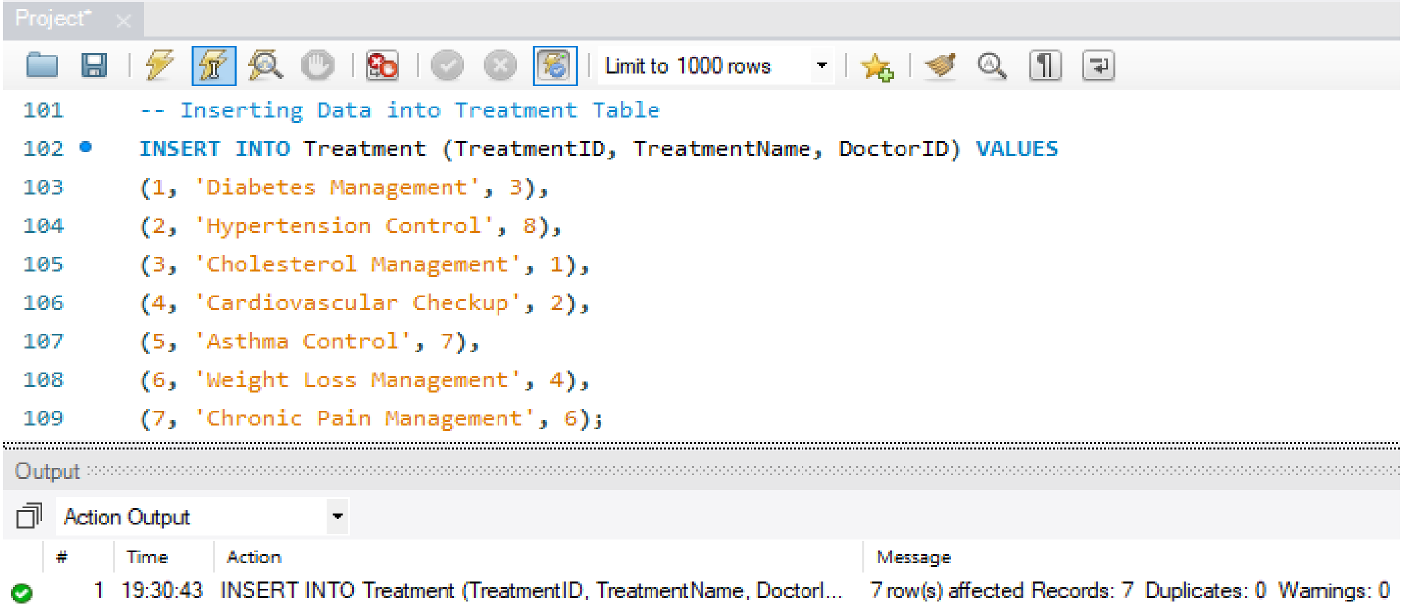
In this screenshot, data has been inserted into the **Patients** table in the **ClinicDB** database using the **INSERT** command.



```sql
-- Inserting Data into Equipment Table
INSERT INTO Equipment (EquipmentID, EquipmentName, EquipmentPrice) VALUES
(1, 'Glucose Meters', 80.00),
(2, 'Sphygmomanometers', 120.00),
(3, 'Cholesterol Meters', 95.00),
(4, 'ECG Machines', 250.00),
(5, 'Spirometers', 130.00),
(6, 'Weighing Scales', 70.00),
(7, 'TENS Units', 200.00);
```

#	Time	Action	Message
1	19:27:00	INSERT INTO Equipment (EquipmentID, EquipmentName, Equipm...	7 row(s) affected Records: 7 Duplicates: 0 Warnings: 0

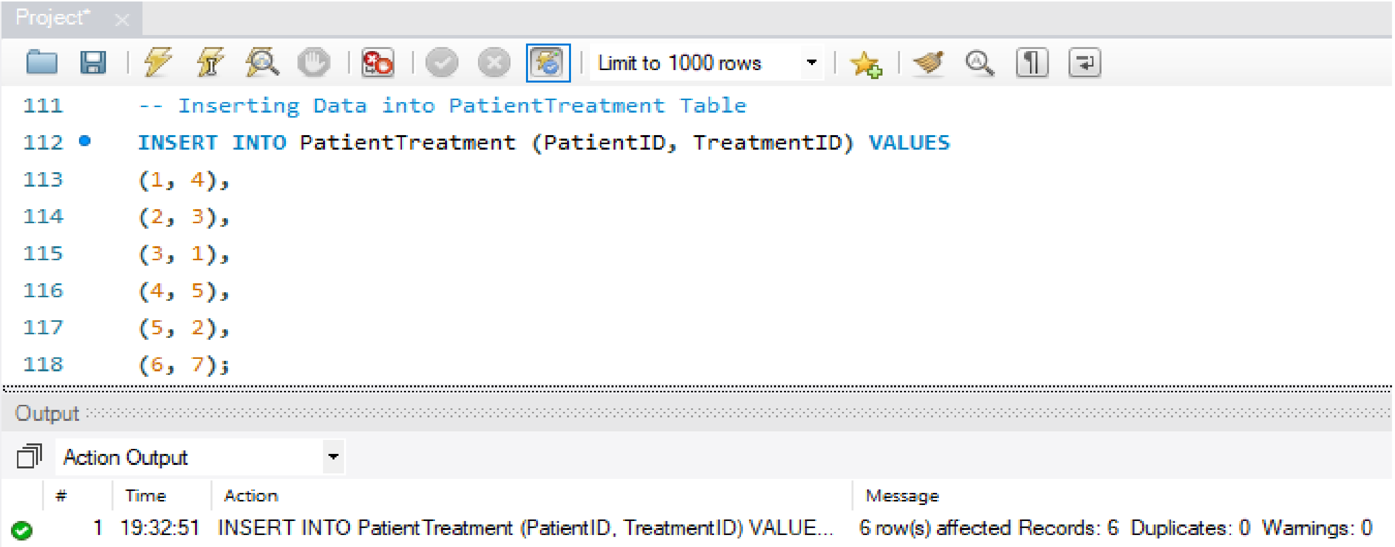
In this screenshot, data has been inserted into the **Equipment** table in the **ClinicDB** database using the **INSERT** command.



```sql
-- Inserting Data into Treatment Table
INSERT INTO Treatment (TreatmentID, TreatmentName, DoctorID) VALUES
(1, 'Diabetes Management', 3),
(2, 'Hypertension Control', 8),
(3, 'Cholesterol Management', 1),
(4, 'Cardiovascular Checkup', 2),
(5, 'Asthma Control', 7),
(6, 'Weight Loss Management', 4),
(7, 'Chronic Pain Management', 6);
```

#	Time	Action	Message
1	19:30:43	INSERT INTO Treatment (TreatmentID, TreatmentName, DoctorI...	7 row(s) affected Records: 7 Duplicates: 0 Warnings: 0

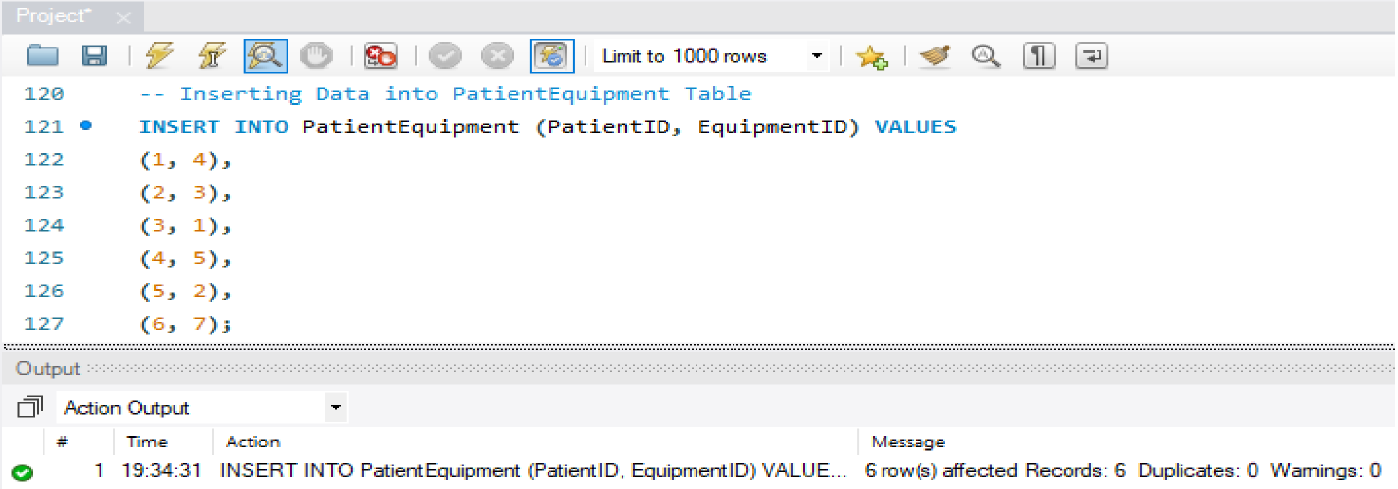
In this screenshot, data has been inserted into the **Treatment** table in the **ClinicDB** database using the **INSERT** command.



```sql
-- Inserting Data into PatientTreatment Table
INSERT INTO PatientTreatment (PatientID, TreatmentID) VALUES
(1, 4),
(2, 3),
(3, 1),
(4, 5),
(5, 2),
(6, 7);
```

#	Time	Action	Message
1	19:32:51	INSERT INTO PatientTreatment (PatientID, TreatmentID) VALUE...	6 row(s) affected Records: 6 Duplicates: 0 Warnings: 0

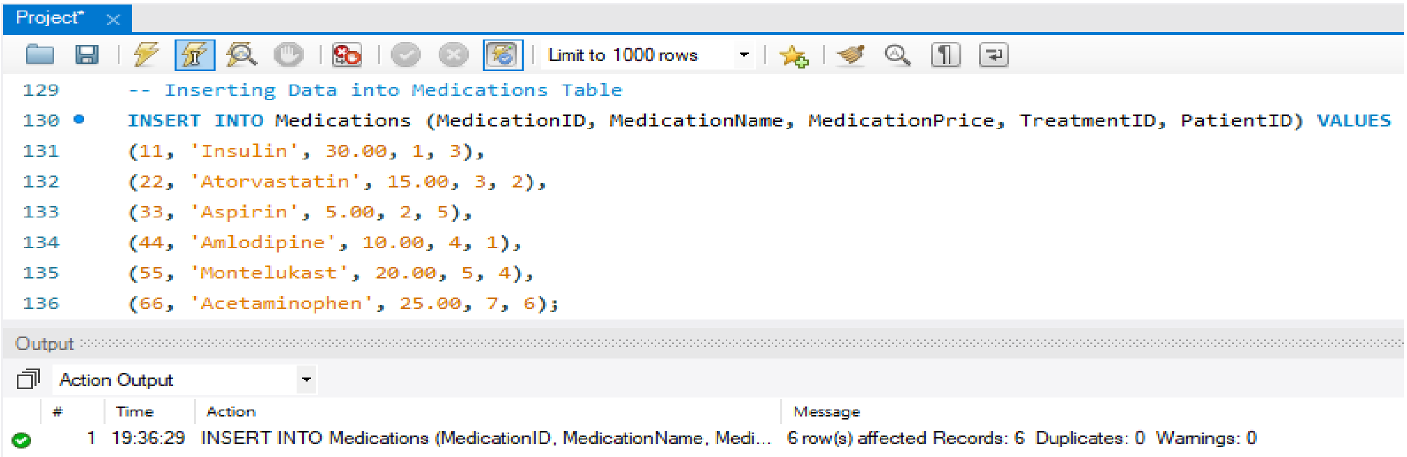
In this screenshot, data has been inserted into the **PatientTreatment** table in the **ClinicDB** database using the **INSERT** command.



```sql
-- Inserting Data into PatientEquipment Table
INSERT INTO PatientEquipment (PatientID, EquipmentID) VALUES
(1, 4),
(2, 3),
(3, 1),
(4, 5),
(5, 2),
(6, 7);
```

```
#   Time       Action                                                     Message
1   19:34:31   INSERT INTO PatientEquipment (PatientID, EquipmentID) VALUE...   6 row(s) affected Records: 6  Duplicates: 0  Warnings: 0
```

In this screenshot, data has been inserted into the **PatientEquipment** table in the **ClinicDB** database using the **INSERT** command.



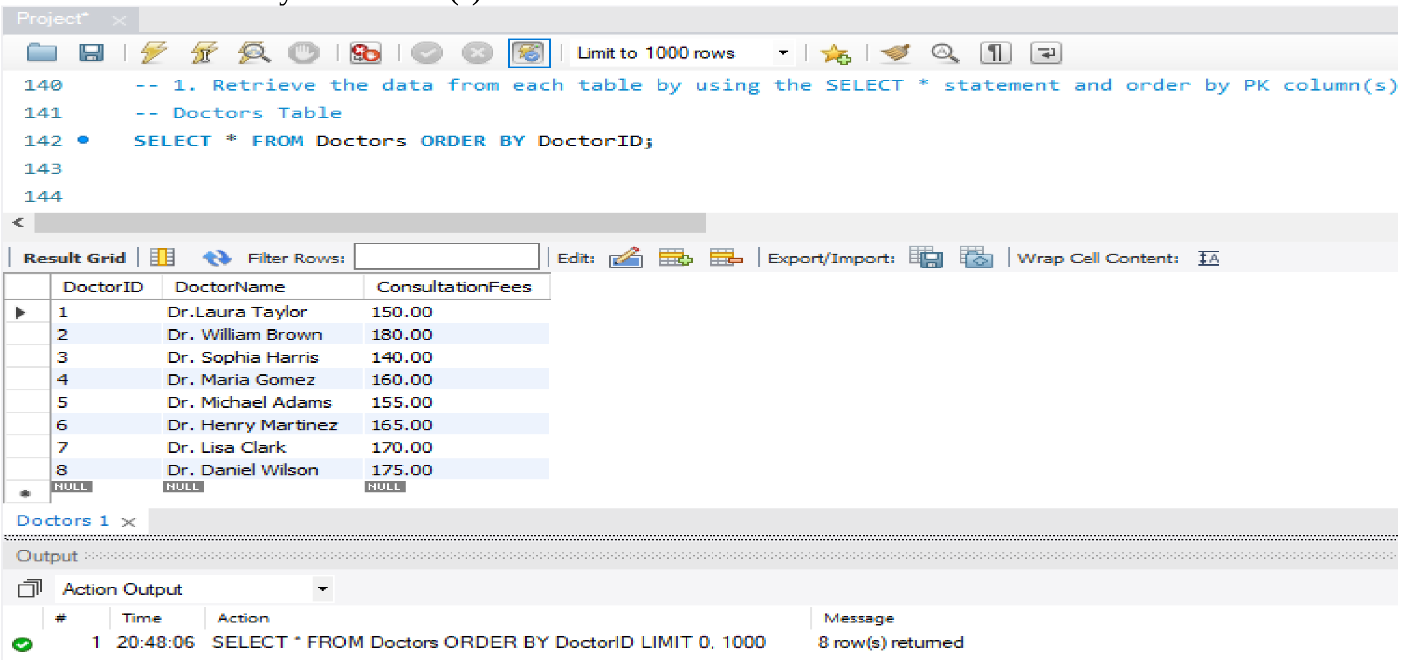
```sql
-- Inserting Data into Medications Table
INSERT INTO Medications (MedicationID, MedicationName, MedicationPrice, TreatmentID, PatientID) VALUES
(11, 'Insulin', 30.00, 1, 3),
(22, 'Atorvastatin', 15.00, 3, 2),
(33, 'Aspirin', 5.00, 2, 5),
(44, 'Amlodipine', 10.00, 4, 1),
(55, 'Montelukast', 20.00, 5, 4),
(66, 'Acetaminophen', 25.00, 7, 6);
```

```
#   Time       Action                                                        Message
1   19:36:29   INSERT INTO Medications (MedicationID, MedicationName, Medi...   6 row(s) affected Records: 6  Duplicates: 0  Warnings: 0
```

In this screenshot, data has been inserted into the **Medications** table in the **ClinicDB** database using the **INSERT** command.

7. Create a variety of SQL queries to retrieve data from one or many tables:

I. Retrieve the data from each table by using the SELECT * statement and order by PK column(s). Show the output. Make sure you show the print screen of the complete set of rows and columns. The rows must be ordered by PK column(s).

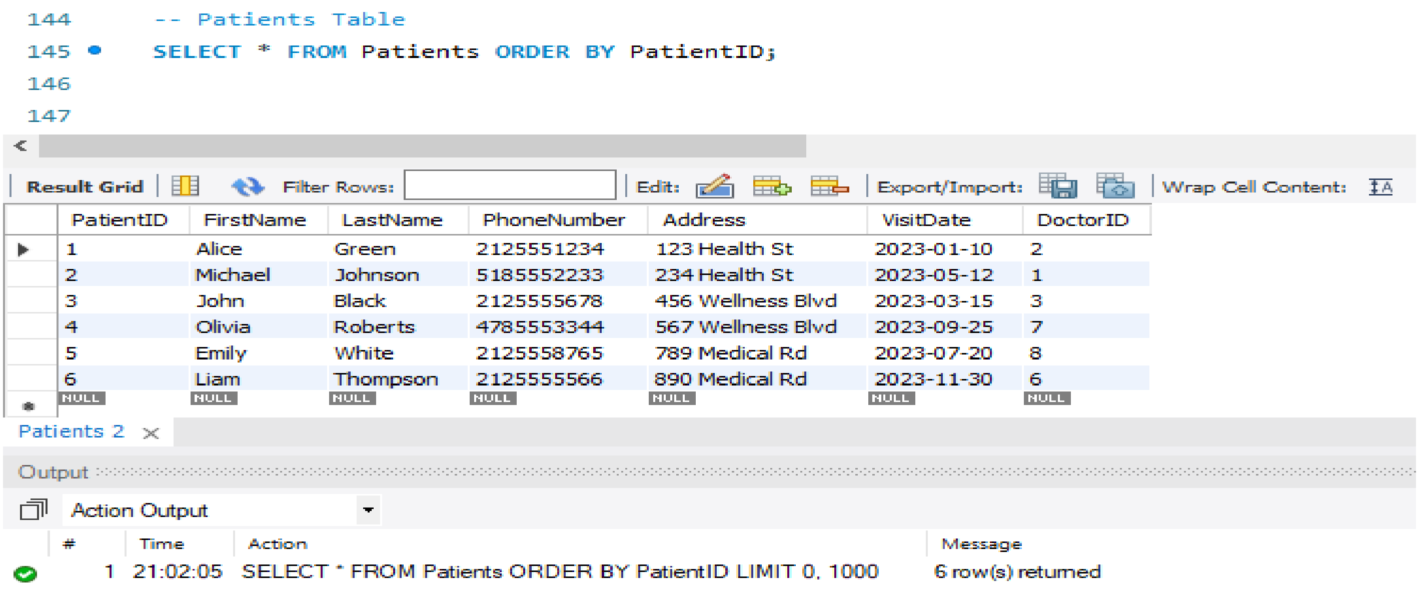


```sql
-- 1. Retrieve the data from each table by using the SELECT * statement and order by PK column(s)
-- Doctors Table
SELECT * FROM Doctors ORDER BY DoctorID;
```

DoctorID	DoctorName	ConsultationFees
1	Dr.Laura Taylor	150.00
2	Dr. William Brown	180.00
3	Dr. Sophia Harris	140.00
4	Dr. Maria Gomez	160.00
5	Dr. Michael Adams	155.00
6	Dr. Henry Martinez	165.00
7	Dr. Lisa Clark	170.00
8	Dr. Daniel Wilson	175.00
NULL	NULL	NULL

```
#   Time       Action                                               Message
1   20:48:06   SELECT * FROM Doctors ORDER BY DoctorID LIMIT 0, 1000   8 row(s) returned
```

In this screenshot, data is retrieved from the **Doctors** table using the **SELECT** statement.

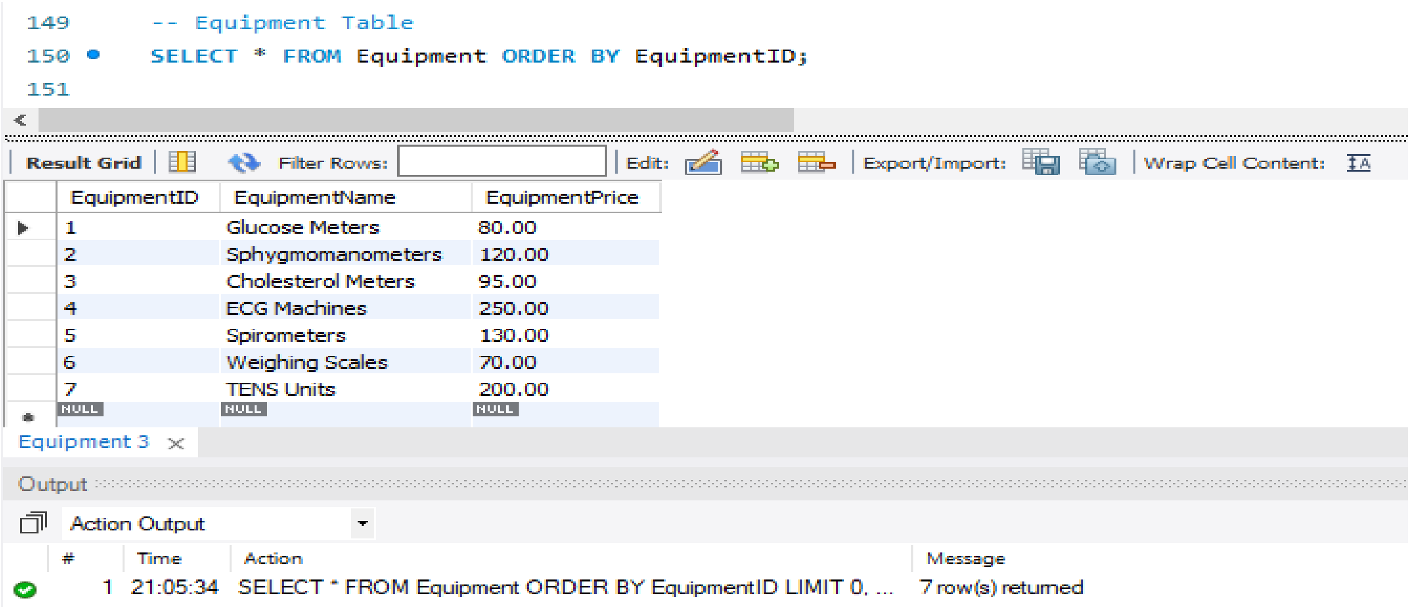


```
-- Patients Table
SELECT * FROM Patients ORDER BY PatientID;
```

	PatientID	FirstName	LastName	PhoneNumber	Address	VisitDate	DoctorID
▶	1	Alice	Green	2125551234	123 Health St	2023-01-10	2
	2	Michael	Johnson	5185552233	234 Health St	2023-05-12	1
	3	John	Black	2125555678	456 Wellness Blvd	2023-03-15	3
	4	Olivia	Roberts	4785553344	567 Wellness Blvd	2023-09-25	7
	5	Emily	White	2125558765	789 Medical Rd	2023-07-20	8
	6	Liam	Thompson	2125555566	890 Medical Rd	2023-11-30	6
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL

	#	Time	Action	Message
✓	1	21:02:05	SELECT * FROM Patients ORDER BY PatientID LIMIT 0, 1000	6 row(s) returned

In this screenshot, data is retrieved from the **Patients** table using the **SELECT** statement.

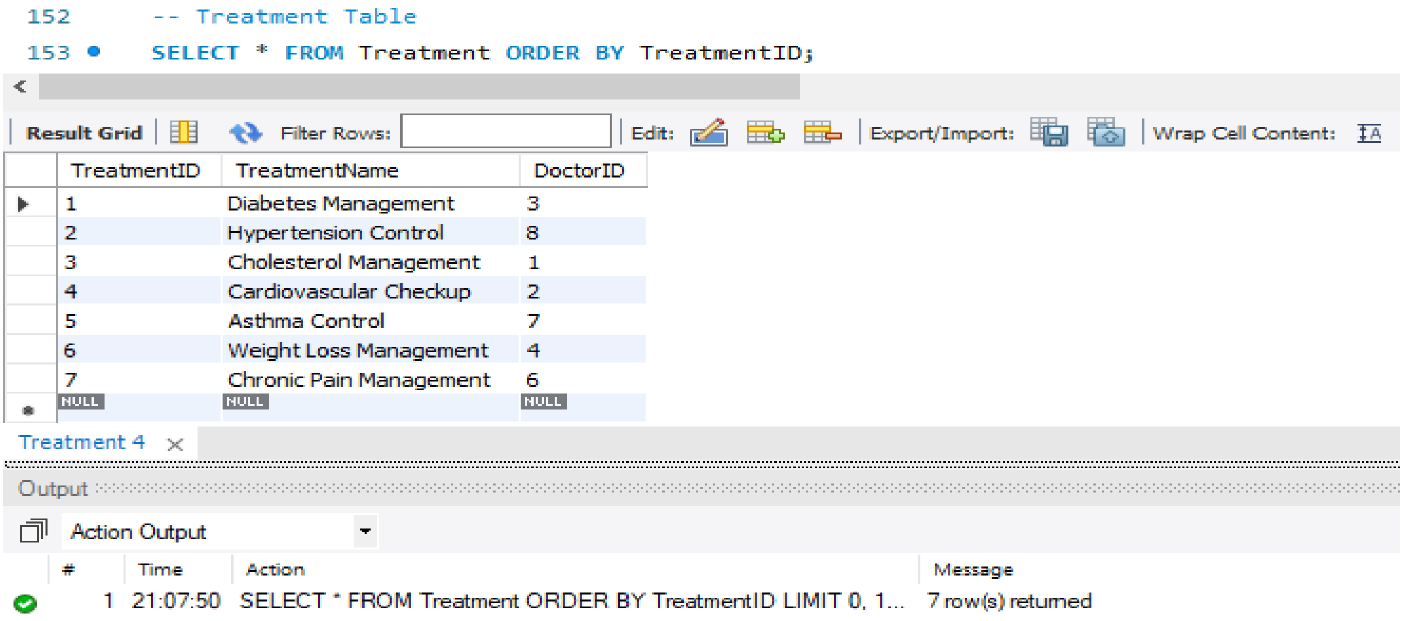


```
-- Equipment Table
SELECT * FROM Equipment ORDER BY EquipmentID;
```

	EquipmentID	EquipmentName	EquipmentPrice
▶	1	Glucose Meters	80.00
	2	Sphygmomanometers	120.00
	3	Cholesterol Meters	95.00
	4	ECG Machines	250.00
	5	Spirometers	130.00
	6	Weighing Scales	70.00
	7	TENS Units	200.00
*	NULL	NULL	NULL

	#	Time	Action	Message
✓	1	21:05:34	SELECT * FROM Equipment ORDER BY EquipmentID LIMIT 0, ...	7 row(s) returned

In this screenshot, data is retrieved from the **Equipment** table using the **SELECT** statement.

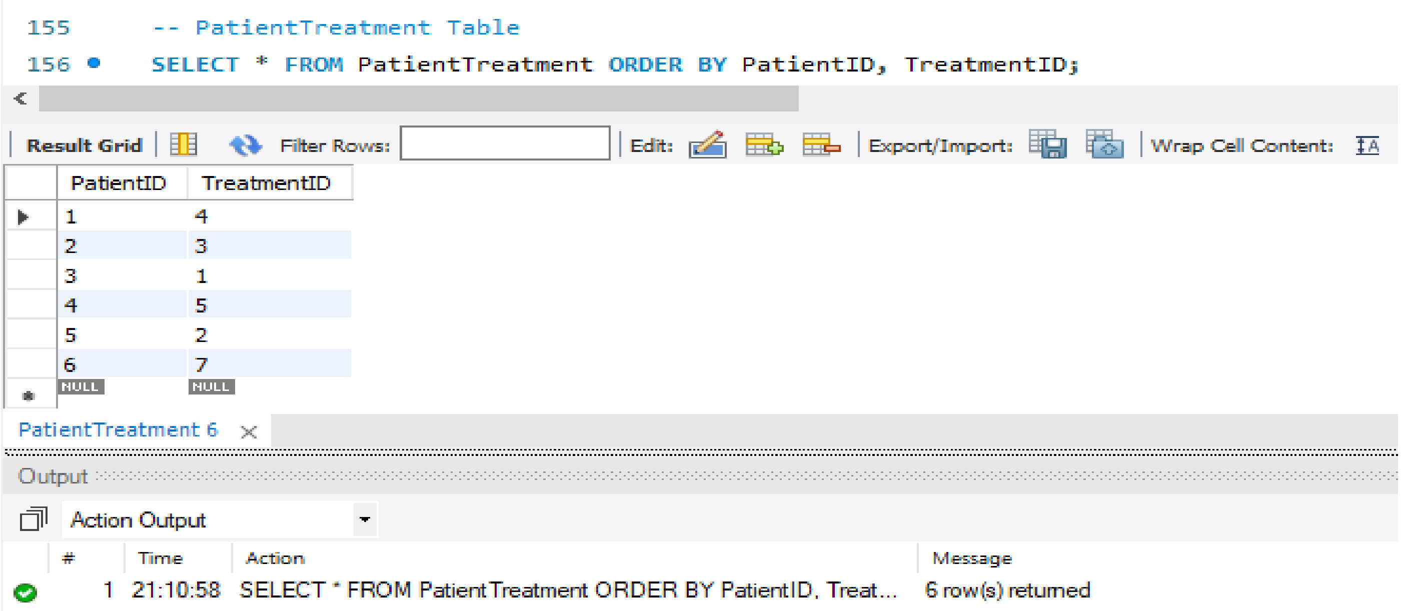


```
-- Treatment Table
SELECT * FROM Treatment ORDER BY TreatmentID;
```

	TreatmentID	TreatmentName	DoctorID
▶	1	Diabetes Management	3
	2	Hypertension Control	8
	3	Cholesterol Management	1
	4	Cardiovascular Checkup	2
	5	Asthma Control	7
	6	Weight Loss Management	4
	7	Chronic Pain Management	6
*	NULL	NULL	NULL

	#	Time	Action	Message
✓	1	21:07:50	SELECT * FROM Treatment ORDER BY TreatmentID LIMIT 0, 1...	7 row(s) returned

In this screenshot, data is retrieved from the **Treatment** table using the **SELECT** statement.

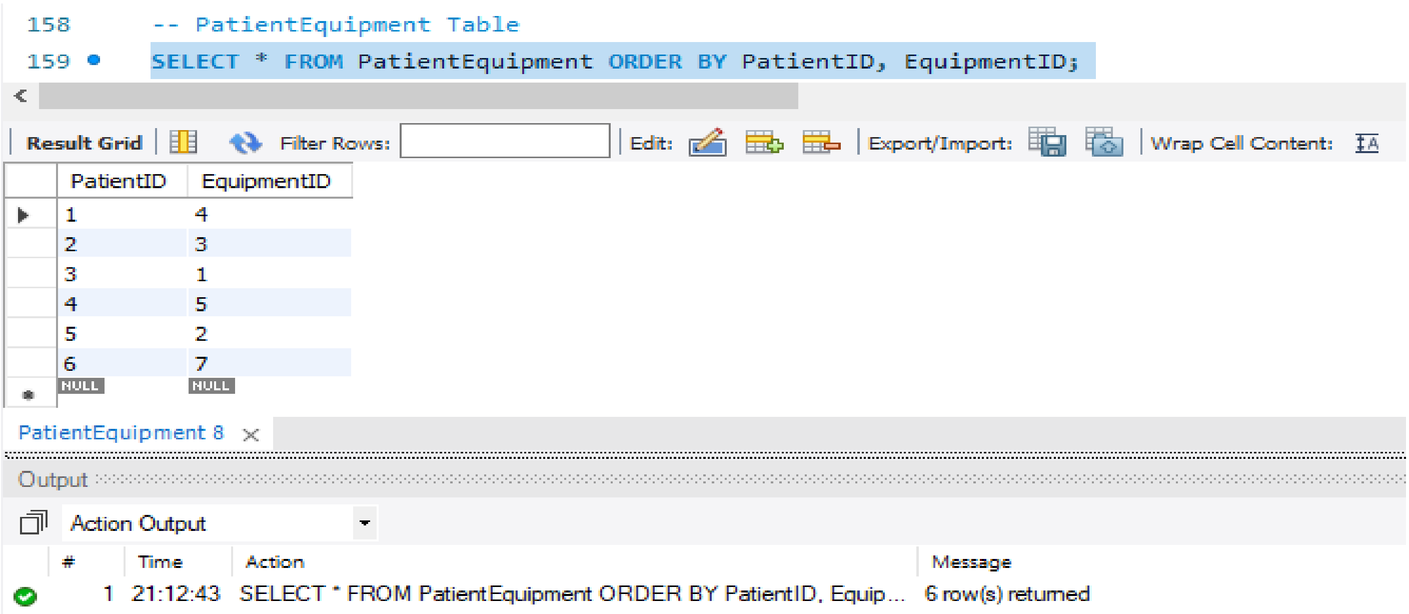


```sql
-- PatientTreatment Table
SELECT * FROM PatientTreatment ORDER BY PatientID, TreatmentID;
```

PatientID	TreatmentID
1	4
2	3
3	1
4	5
5	2
6	7
NULL	NULL

```
#   Time      Action                                                      Message
1   21:10:58  SELECT * FROM PatientTreatment ORDER BY PatientID, Treat...  6 row(s) returned
```

In this screenshot, data is retrieved from the **PatientTreatment** table using the **SELECT** statement.

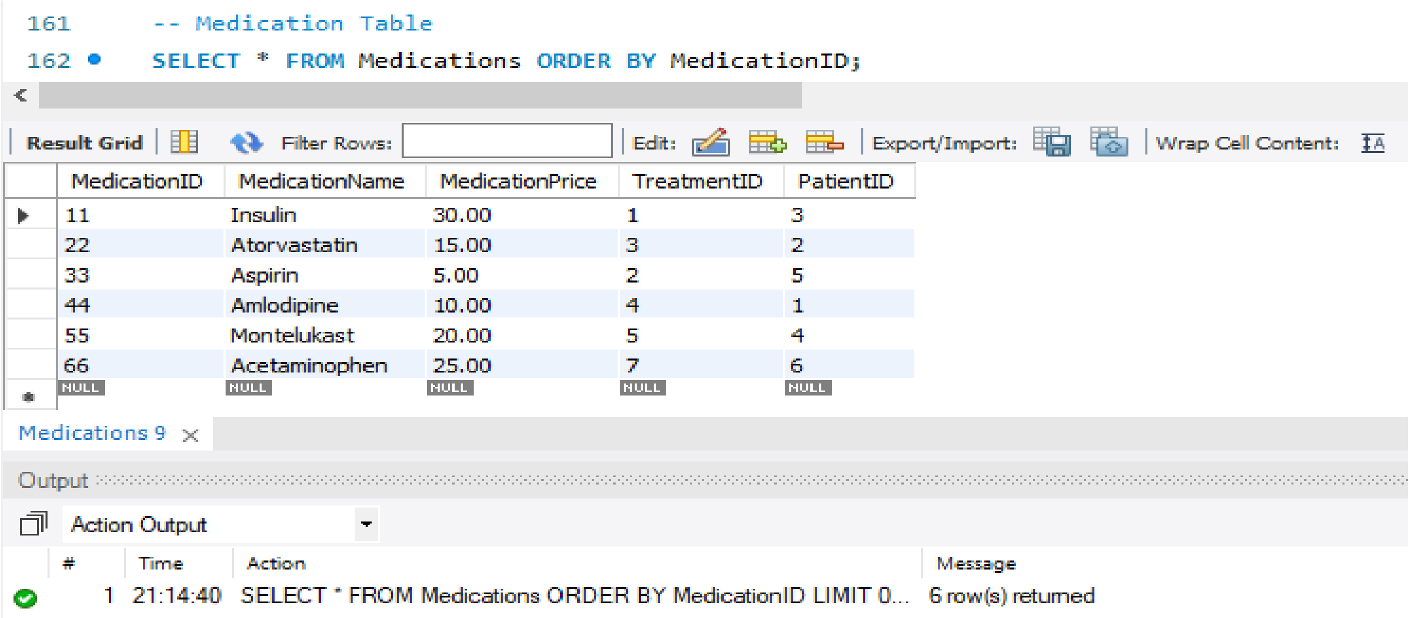


```sql
-- PatientEquipment Table
SELECT * FROM PatientEquipment ORDER BY PatientID, EquipmentID;
```

PatientID	EquipmentID
1	4
2	3
3	1
4	5
5	2
6	7
NULL	NULL

```
#   Time      Action                                                      Message
1   21:12:43  SELECT * FROM PatientEquipment ORDER BY PatientID, Equip...  6 row(s) returned
```

In this screenshot, data is retrieved from the **PatientTreatment** table using the **SELECT** statement.



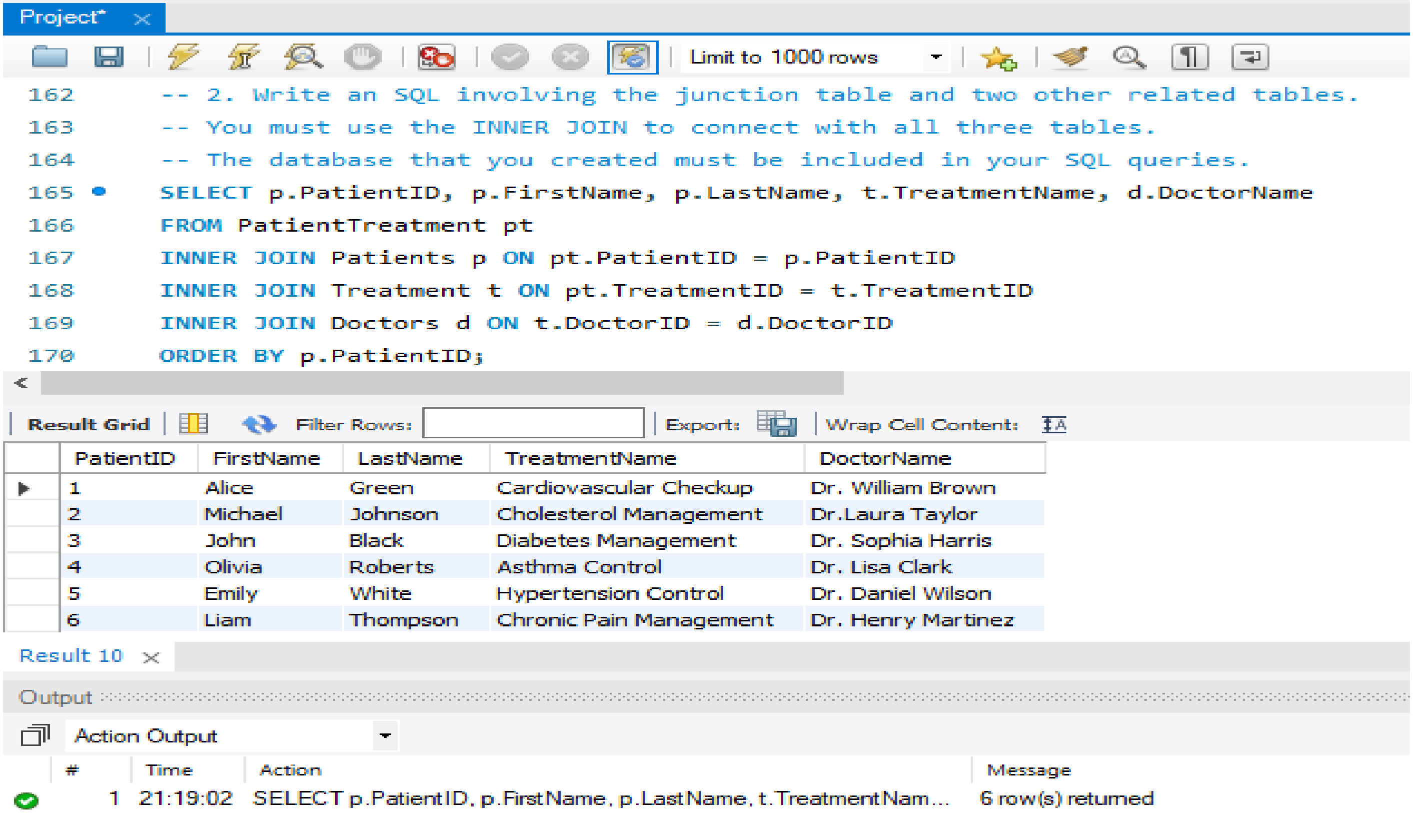
```sql
-- Medication Table
SELECT * FROM Medications ORDER BY MedicationID;
```

MedicationID	MedicationName	MedicationPrice	TreatmentID	PatientID
11	Insulin	30.00	1	3
22	Atorvastatin	15.00	3	2
33	Aspirin	5.00	2	5
44	Amlodipine	10.00	4	1
55	Montelukast	20.00	5	4
66	Acetaminophen	25.00	7	6
NULL	NULL	NULL	NULL	NULL

```
#   Time      Action                                                      Message
1   21:14:40  SELECT * FROM Medications ORDER BY MedicationID LIMIT 0...  6 row(s) returned
```

In this screenshot, data is retrieved from the **Medications** table using the **SELECT** statement.

II. Write an SQL involving the junction table and two other related tables. You must use the INNER JOIN to connect with all three tables. The database that you created must be included in your SQL queries.



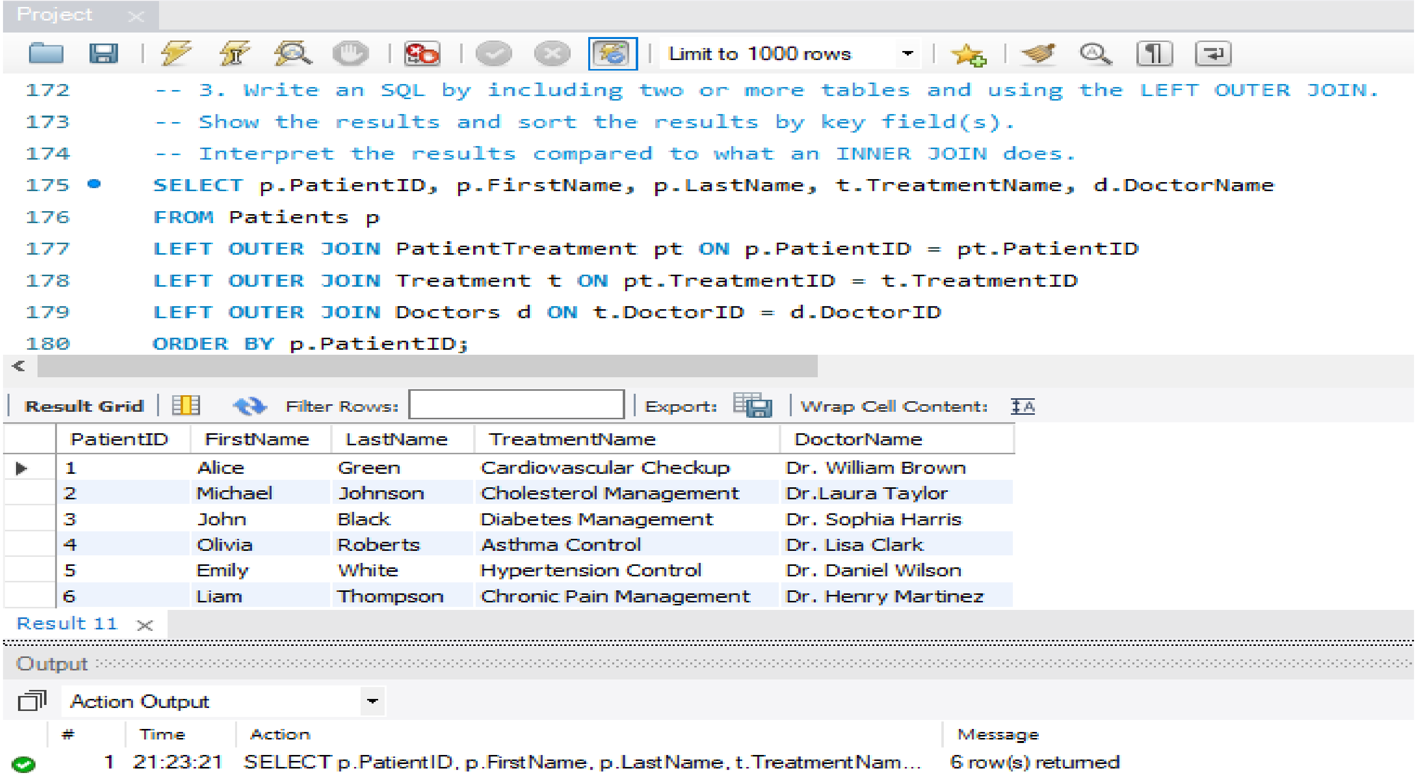
```
-- 2. Write an SQL involving the junction table and two other related tables.
-- You must use the INNER JOIN to connect with all three tables.
-- The database that you created must be included in your SQL queries.
SELECT p.PatientID, p.FirstName, p.LastName, t.TreatmentName, d.DoctorName
FROM PatientTreatment pt
INNER JOIN Patients p ON pt.PatientID = p.PatientID
INNER JOIN Treatment t ON pt.TreatmentID = t.TreatmentID
INNER JOIN Doctors d ON t.DoctorID = d.DoctorID
ORDER BY p.PatientID;
```

PatientID	FirstName	LastName	TreatmentName	DoctorName
1	Alice	Green	Cardiovascular Checkup	Dr. William Brown
2	Michael	Johnson	Cholesterol Management	Dr.Laura Taylor
3	John	Black	Diabetes Management	Dr. Sophia Harris
4	Olivia	Roberts	Asthma Control	Dr. Lisa Clark
5	Emily	White	Hypertension Control	Dr. Daniel Wilson
6	Liam	Thompson	Chronic Pain Management	Dr. Henry Martinez

```
#   Time       Action                                                          Message
1   21:19:02   SELECT p.PatientID, p.FirstName, p.LastName, t.TreatmentNam...   6 row(s) returned
```

- **Interpretation:** This SQL query uses **INNER JOINs** to combine data from three related tables (PatientTreatment, Patients, Treatment, and Doctors) to retrieve details about patients, treatments, and doctors.

III. Write an SQL by including two or more tables and using the LEFT OUTER JOIN. Show the results and sort the results by key field(s). Interpret the results compared to what an INNER JOIN does.



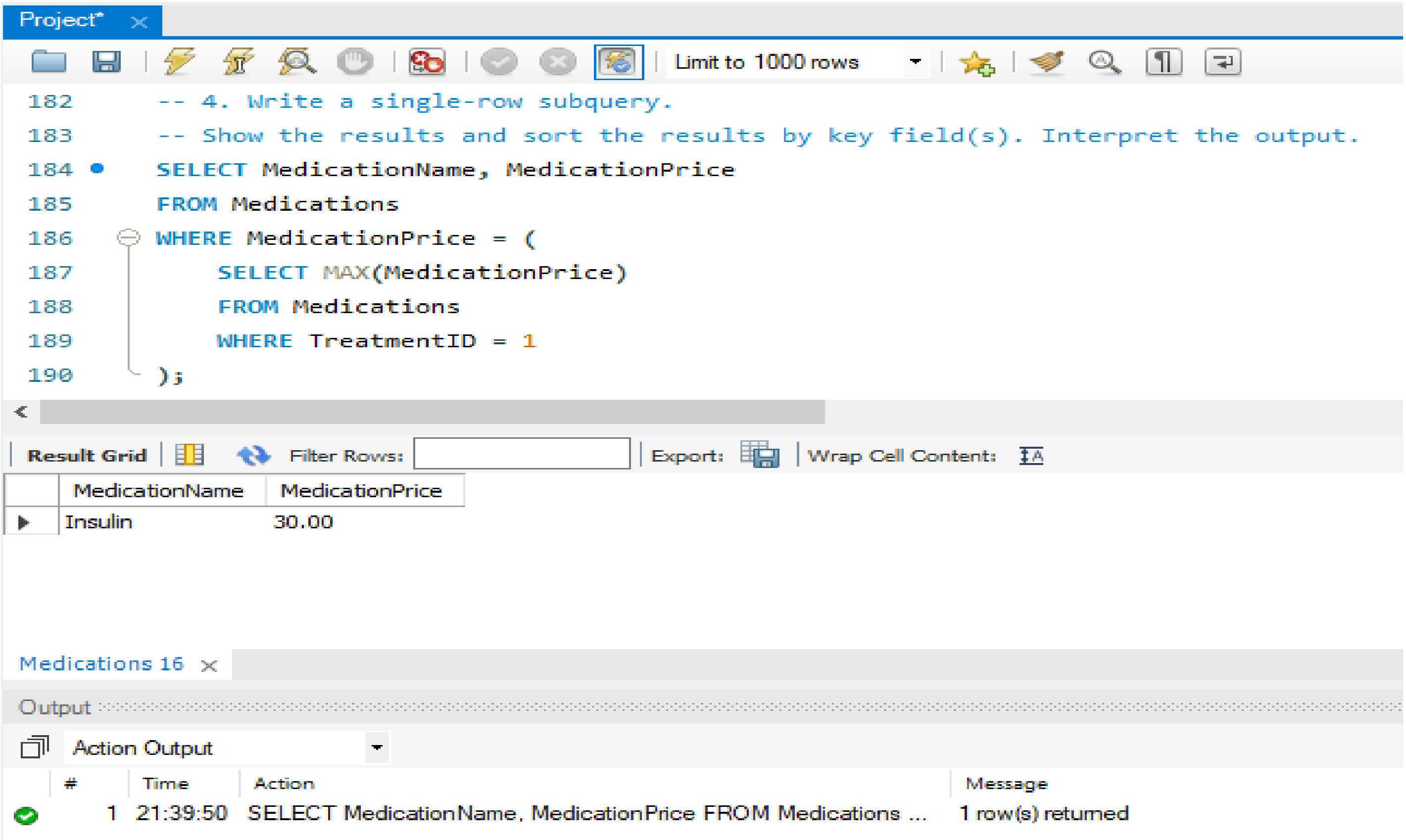
```
-- 3. Write an SQL by including two or more tables and using the LEFT OUTER JOIN.
-- Show the results and sort the results by key field(s).
-- Interpret the results compared to what an INNER JOIN does.
SELECT p.PatientID, p.FirstName, p.LastName, t.TreatmentName, d.DoctorName
FROM Patients p
LEFT OUTER JOIN PatientTreatment pt ON p.PatientID = pt.PatientID
LEFT OUTER JOIN Treatment t ON pt.TreatmentID = t.TreatmentID
LEFT OUTER JOIN Doctors d ON t.DoctorID = d.DoctorID
ORDER BY p.PatientID;
```

PatientID	FirstName	LastName	TreatmentName	DoctorName
1	Alice	Green	Cardiovascular Checkup	Dr. William Brown
2	Michael	Johnson	Cholesterol Management	Dr.Laura Taylor
3	John	Black	Diabetes Management	Dr. Sophia Harris
4	Olivia	Roberts	Asthma Control	Dr. Lisa Clark
5	Emily	White	Hypertension Control	Dr. Daniel Wilson
6	Liam	Thompson	Chronic Pain Management	Dr. Henry Martinez

```
#   Time       Action                                                          Message
1   21:23:21   SELECT p.PatientID, p.FirstName, p.LastName, t.TreatmentNam...   6 row(s) returned
```

- **Interpretation:** The LEFT JOIN returns all patients, including those who may not have a treatment assigned. INNER JOIN only returns patients with treatments.

IV. Write a single-row subquery. Show the results and sort the results by key field(s). Interpret the output.



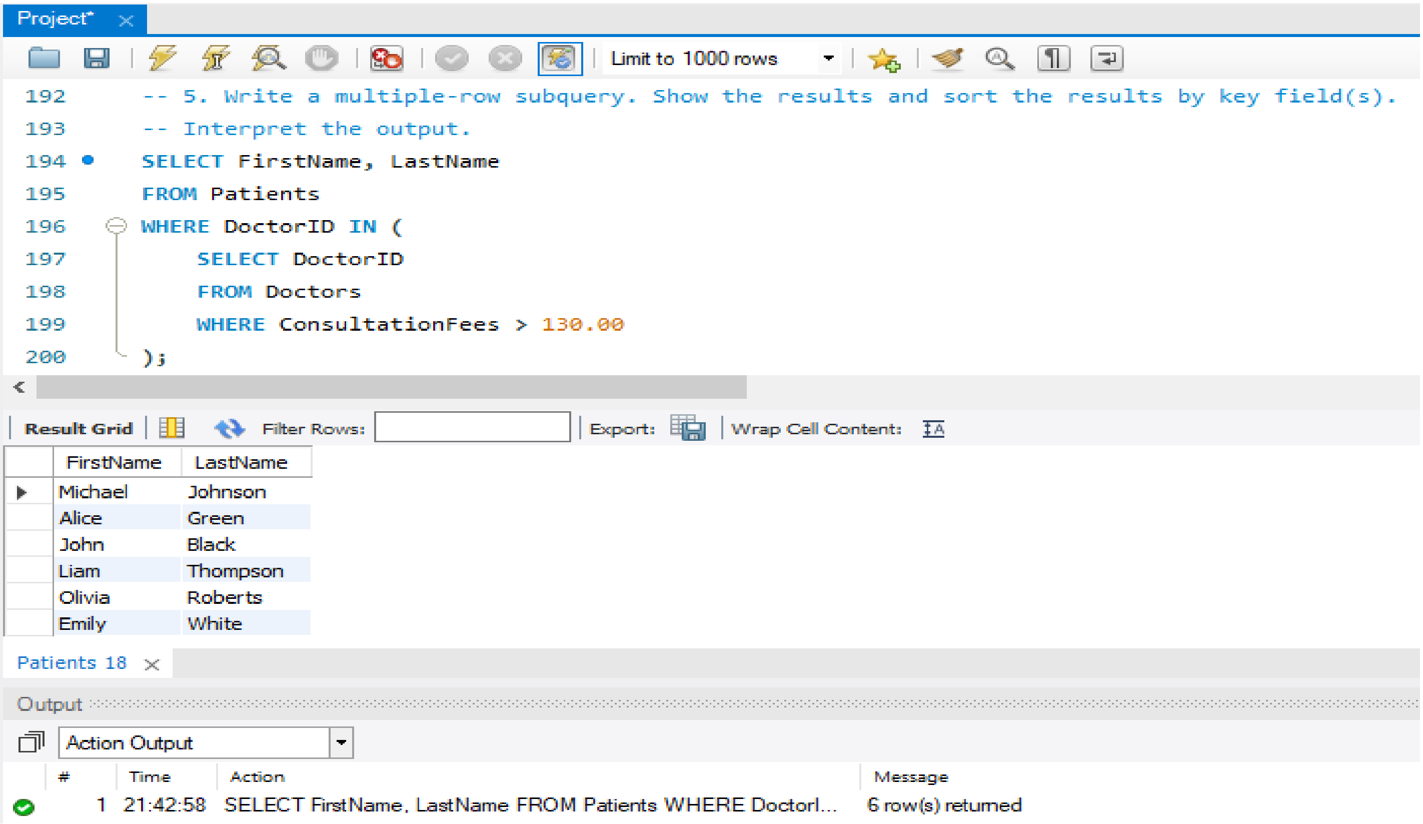
```
-- 4. Write a single-row subquery.
-- Show the results and sort the results by key field(s). Interpret the output.
SELECT MedicationName, MedicationPrice
FROM Medications
WHERE MedicationPrice = (
    SELECT MAX(MedicationPrice)
    FROM Medications
    WHERE TreatmentID = 1
);
```

MedicationName	MedicationPrice
Insulin	30.00

```
#   Time      Action                                                  Message
1   21:39:50  SELECT MedicationName, MedicationPrice FROM Medications ...   1 row(s) returned
```

Interpretation: This query finds the most expensive medication for TreatmentID = 1. The subquery calculates the maximum price for that treatment, and the outer query returns the name and price of the medication.

V. Write a multiple-row subquery. Show the results and sort the results by key field(s). Interpret the output.



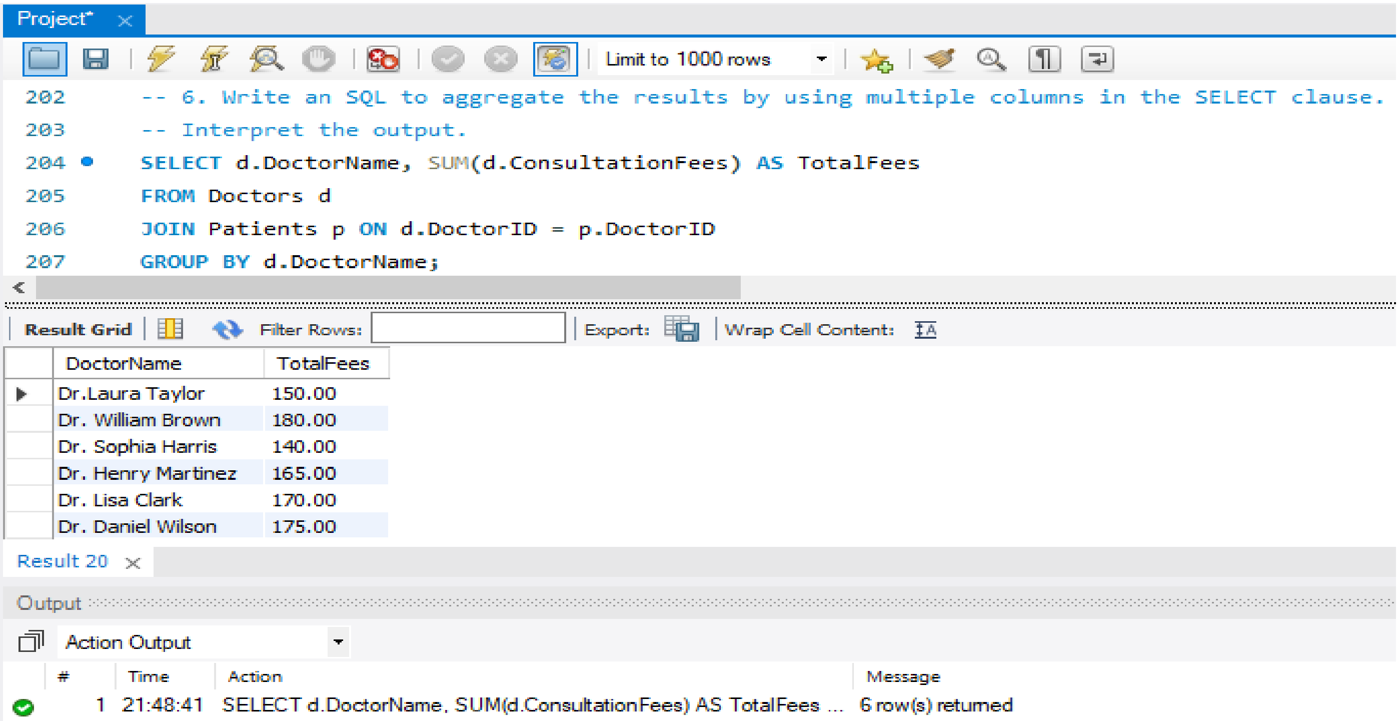
```
-- 5. Write a multiple-row subquery. Show the results and sort the results by key field(s).
-- Interpret the output.
SELECT FirstName, LastName
FROM Patients
WHERE DoctorID IN (
    SELECT DoctorID
    FROM Doctors
    WHERE ConsultationFees > 130.00
);
```

FirstName	LastName
Michael	Johnson
Alice	Green
John	Black
Liam	Thompson
Olivia	Roberts
Emily	White

```
#   Time      Action                                                Message
1   21:42:58  SELECT FirstName, LastName FROM Patients WHERE DoctorI...   6 row(s) returned
```

Interpretation: The subquery retrieves DoctorID values where the consultation fee is greater than $130. The outer query returns the patients treated by those doctors.

VI. Write an SQL to aggregate the results by using multiple columns in the SELECT clause. Interpret the output.



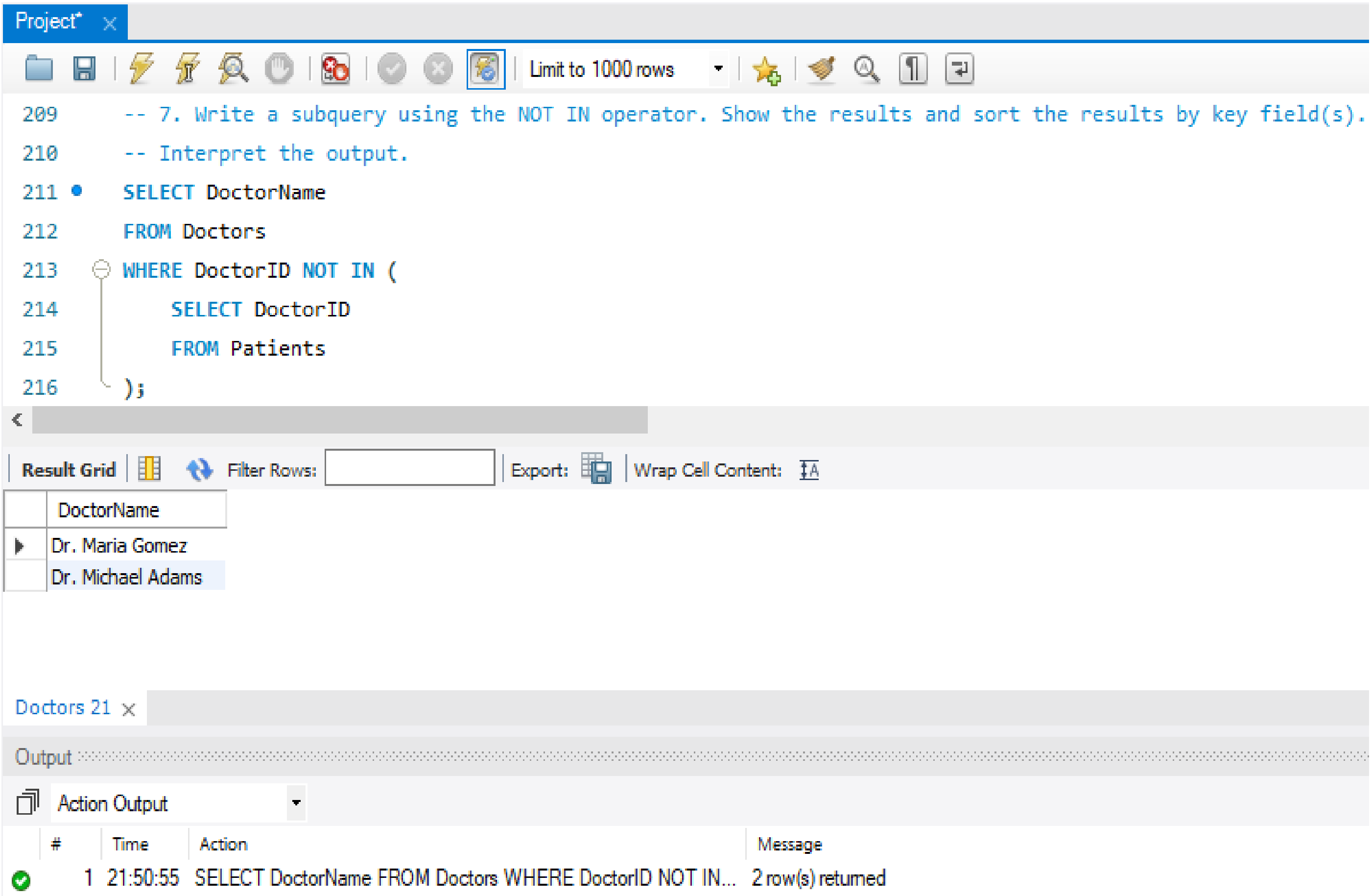
```sql
-- 6. Write an SQL to aggregate the results by using multiple columns in the SELECT clause.
-- Interpret the output.
SELECT d.DoctorName, SUM(d.ConsultationFees) AS TotalFees
FROM Doctors d
JOIN Patients p ON d.DoctorID = p.DoctorID
GROUP BY d.DoctorName;
```

DoctorName	TotalFees
Dr.Laura Taylor	150.00
Dr. William Brown	180.00
Dr. Sophia Harris	140.00
Dr. Henry Martinez	165.00
Dr. Lisa Clark	170.00
Dr. Daniel Wilson	175.00

```
#   Time      Action                                                              Message
1   21:48:41  SELECT d.DoctorName, SUM(d.ConsultationFees) AS TotalFees ...      6 row(s) returned
```

Interpretation: This query aggregates the total consultation fees collected by each doctor, using the SUM function, and groups the results by doctor.

VII. Write a subquery using the NOT IN operator. Show the results and sort the results by key field(s). Interpret the output.



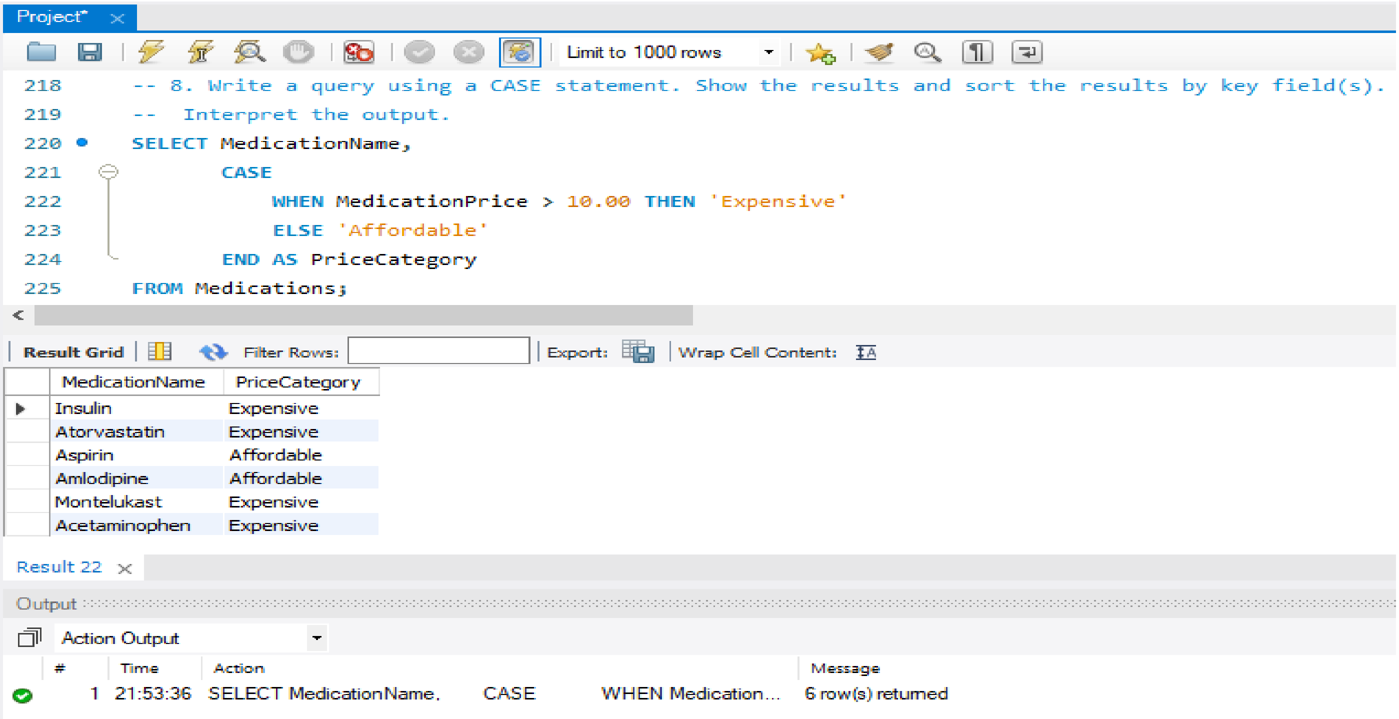
```sql
-- 7. Write a subquery using the NOT IN operator. Show the results and sort the results by key field(s).
-- Interpret the output.
SELECT DoctorName
FROM Doctors
WHERE DoctorID NOT IN (
    SELECT DoctorID
    FROM Patients
);
```

DoctorName
Dr. Maria Gomez
Dr. Michael Adams

```
#   Time      Action                                               Message
1   21:50:55  SELECT DoctorName FROM Doctors WHERE DoctorID NOT IN...   2 row(s) returned
```

Interpretation: The subquery selects all DoctorID values from the Patients table, and the outer query returns doctors whose DoctorID is not in that list.

VIII. Write a query using a CASE statement. Show the results and sort the results by key field(s). Interpret the output.



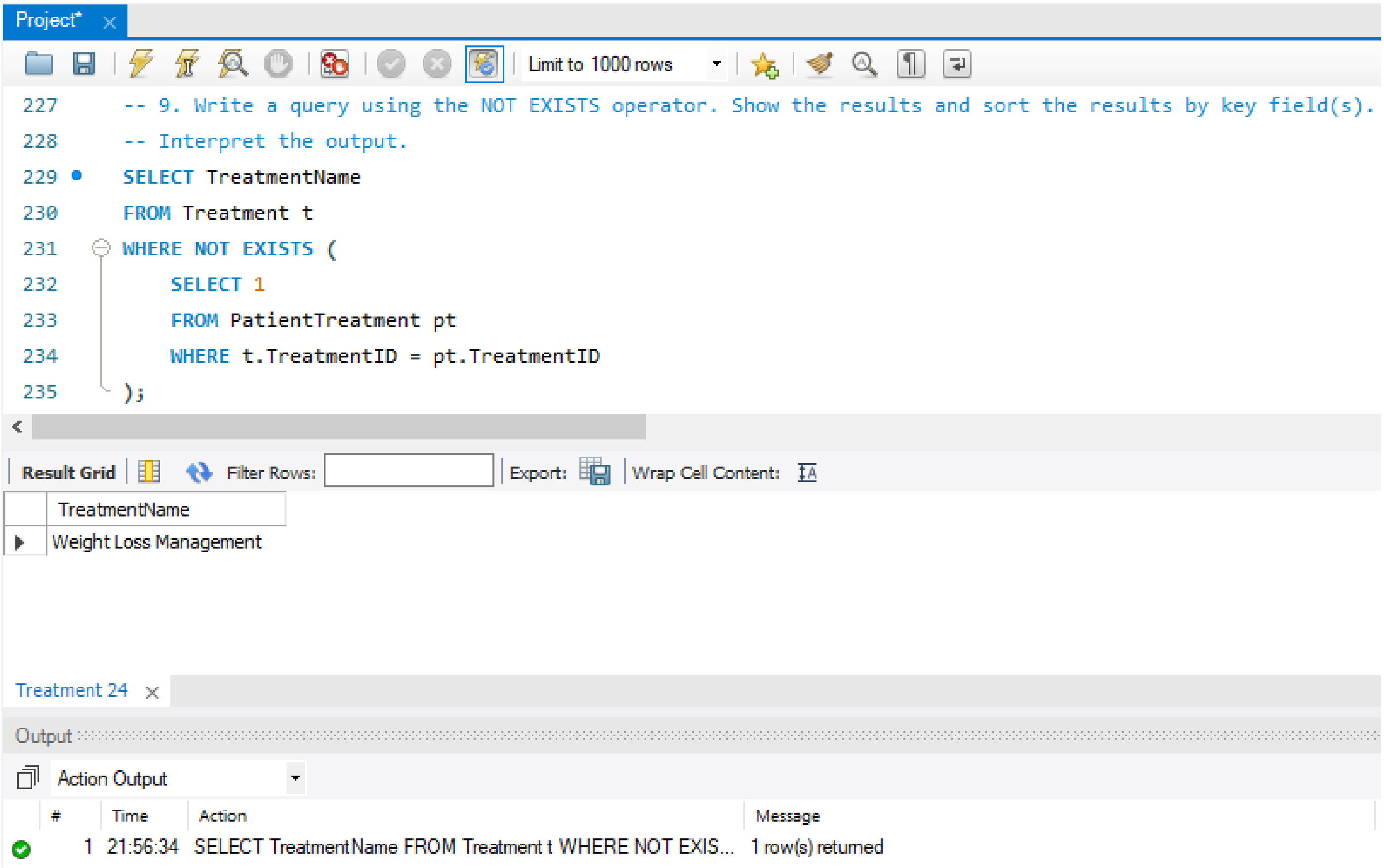
```
-- 8. Write a query using a CASE statement. Show the results and sort the results by key field(s).
--  Interpret the output.
SELECT MedicationName,
       CASE
           WHEN MedicationPrice > 10.00 THEN 'Expensive'
           ELSE 'Affordable'
       END AS PriceCategory
FROM Medications;
```

MedicationName	PriceCategory
Insulin	Expensive
Atorvastatin	Expensive
Aspirin	Affordable
Amlodipine	Affordable
Montelukast	Expensive
Acetaminophen	Expensive

```
#  Time      Action                                              Message
1  21:53:36  SELECT MedicationName,   CASE   WHEN Medication...  6 row(s) returned
```

Interpretation: The CASE statement categorizes medications based on their price into 'Expensive' or 'Affordable'.

IX. Write a query using the NOT EXISTS operator. Show the results and sort the results by key field(s). Interpret the output.



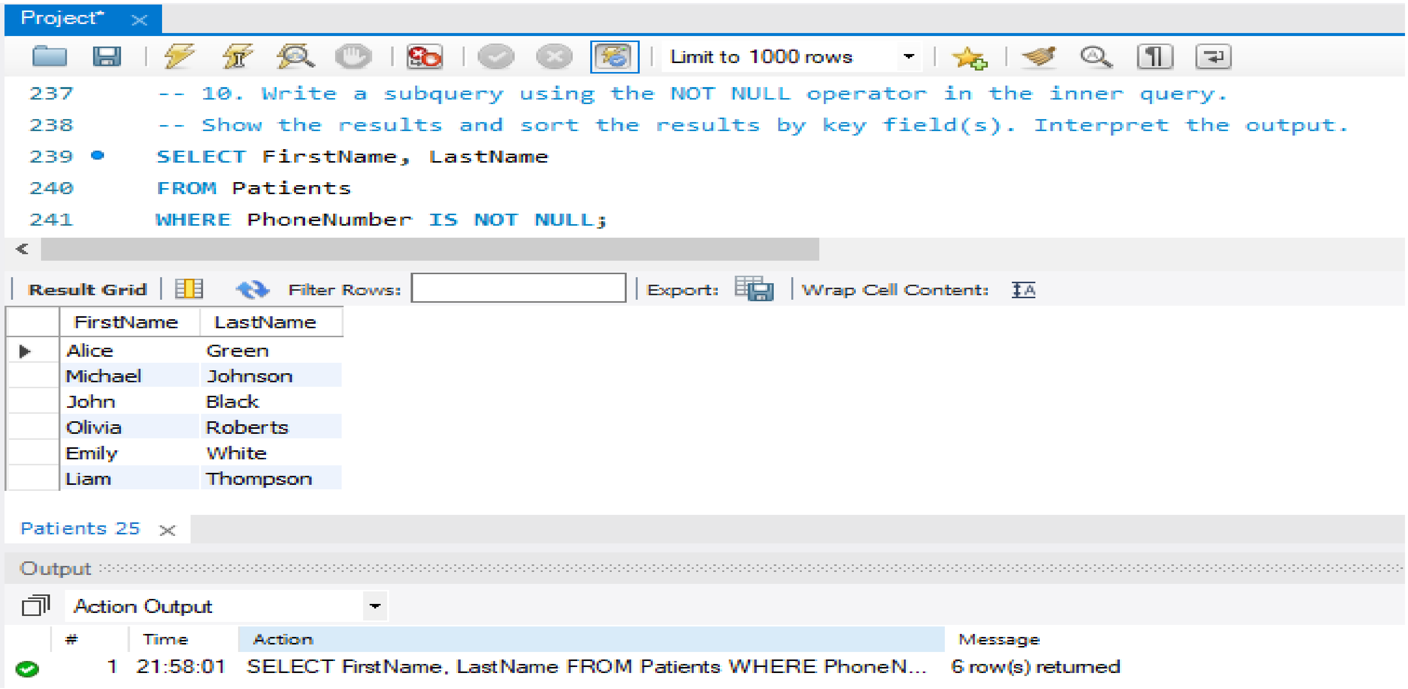
```
-- 9. Write a query using the NOT EXISTS operator. Show the results and sort the results by key field(s).
-- Interpret the output.
SELECT TreatmentName
FROM Treatment t
WHERE NOT EXISTS (
    SELECT 1
    FROM PatientTreatment pt
    WHERE t.TreatmentID = pt.TreatmentID
);
```

TreatmentName
Weight Loss Management

```
#  Time      Action                                                   Message
1  21:56:34  SELECT TreatmentName FROM Treatment t WHERE NOT EXIS...  1 row(s) returned
```

Interpretation: The subquery checks for the existence of any patient assignment for each treatment. The outer query returns treatments with no assigned patients.

X. Write a subquery using the NOT NULL operator in the inner query. Show the results and sort the results by key field(s). Interpret the output.



```sql
-- 10. Write a subquery using the NOT NULL operator in the inner query.
-- Show the results and sort the results by key field(s). Interpret the output.
SELECT FirstName, LastName
FROM Patients
WHERE PhoneNumber IS NOT NULL;
```

FirstName	LastName
Alice	Green
Michael	Johnson
John	Black
Olivia	Roberts
Emily	White
Liam	Thompson

```
#   Time       Action                                              Message
1   21:58:01   SELECT FirstName, LastName FROM Patients WHERE PhoneN...   6 row(s) returned
```

Interpretation: This query selects the names of patients whose PhoneNumber field is not NULL, ensuring only patients with recorded phone numbers are retrieved.

Summary:

This project focused on developing a data management system for a small clinic to enhance operational efficiency and patient care by addressing the limitations of manual record handling. The system was designed using a comprehensive database modelling approach, which included conceptual, logical, and physical data models. Key entities such as Patients, Doctors, Treatments, Medications, and Equipment were defined, with many-to-many relationships resolved through junction tables like PatientTreatment and PatientEquipment. The relational database design ensured that clinic data ranging from patient information to diagnostic procedures and equipment usage was organized into well-defined structures to enable accurate storage, retrieval, and analysis.

The technical implementation showcased advanced SQL proficiency through database creation (DDL), data insertion (DML), and a range of complex queries. These queries included SELECT statements with sorting and filtering, INNER and LEFT OUTER JOINS, aggregate functions, CASE statements, and both single-row and multi-row subqueries. For instance, a subquery was used to identify the most expensive medication for a specific treatment, while aggregate functions calculated total consultation fees by doctor. By combining robust theoretical database principles with practical SQL implementation, the project delivered a functional prototype of a clinic management system, demonstrating its ability to improve resource utilization and support effective decision-making.