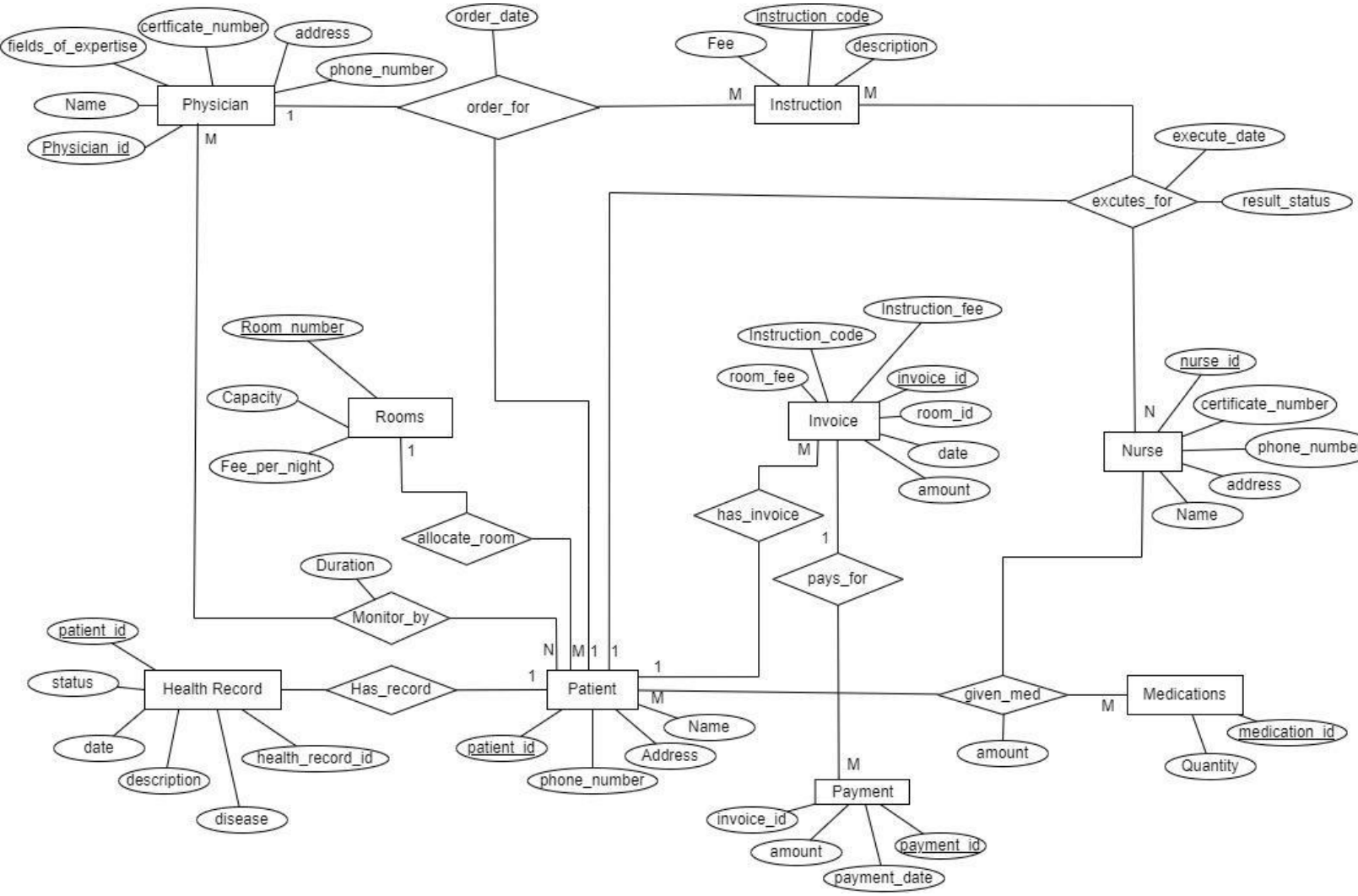
Term Project Report

Group Members: Irisha Banal, Abdullah Irfan, MJ Javadinasr
CS 480: Database Systems

1. **Assumptions**

- The relationships "Patient - Room," "Physician - Patient," "Physician - Instruction," "Nurse - Instruction," and "Patient - Medication" are represented as foreign key constraints in the respective tables.
- Each patient can have multiple health records, medications, and instructions.
- Each physician and nurse can have multiple patients under their care.
- The "Payment" table records the payments made by patients for the hospital services, including rooms and instructions.
- The "Medication" table records the medications provided by nurses to patients.

2. **(E)ERD**



3. Relationships

1. Patient - Room (Many-to-One):
   A patient is hospitalized in a specific room for some nights.
   Patient has Rooms(room_id) (Foreign Key referencing Room).
2. Physician - Patient (Many-to-Many):
   Every patient has some physicians who monitor the patient for a specific duration.
   Monitor_By with Physician(physician_id) (Foreign Key referencing Physician)
   and Patient(patient_id) (Foreign Key referencing Patient).
3. Nurse - Patient (Many-to-Many):
   A nurse can provide services to many patients, and each patient can receive care
   from multiple nurses.

Executes_For has Nurse(nurse_id) (Foreign Key referencing Nurse)
and Patient(patient_id) (Foreign Key referencing Patient).

4. Physician - Instruction (One-to-Many):
   A physician orders some instructions for a patient on a specific date.
   Orders_For has Physician(physician_id) (Foreign Key referencing Physician) and Instruction(instruction_code) (Foreign Key referencing Instruction).
5. Nurse - Instruction (One-to-Many):
   A number of nurses execute the physician's order for a specific patient on a date, and the execution results in a status.
   Executes_For has Nurse(nurse_id) (Foreign Key referencing Nurse)
   And Instruction(instruction_code) (Foreign Key referencing Instruction).
6. Patient - Medication (One-to-Many):
   Each patient has specific medications, and there is a specific amount of medication that is given to the patient daily by nurses.
   Gives_Med has Patient(patient_id) (Foreign Key referencing Patient)
7. Nurse - Medication (Many-to-Many):
   A nurse can provide multiple medications to a patient, and a patient can receive medications from multiple nurses.
   Gives_Med has Nurse(nurse_id) (Foreign Key referencing Nurse)
8. Patient - Health Record (One-to-One):
   Each patient may have a health record, and each health record belongs to only one patient.
   Patient has Health_Record(health_record_id) (Foreign Key referencing Health_Record)
9. Patient - Invoice (One-to-Many):
   Each patient can have multiple invoices for payable items, and each invoice belongs to one patient.
   Patient has Invoice(invoice_id) (Foreign Key referencing Invoice)
10. Patient - Payment (One-to-Many):
    Each patient can make multiple payments to the hospital, and each payment is associated with only one patient.
    Patient has Payment(payment_id) (Foreign Key referencing Payment).

4. Entities and Attributes

1. Physician
   - Physician(physician_id, name, certificate_number, field_of_expertise, address, phone_number)
   - Primary Key: {physician_id}

2. Nurse

- Nurse(nurse_id, name, certificate_number, phone_number, address)
- Primary Key: {nurse_id}

3. Patient
    - Patient(patient_id, room_no, health_record_id, phone_number, address, name)
    - Primary Key: {patient_id}
    - Foreign Key: {room_no references Rooms(room_no), health_record_id references Health_Record(health_record_id)}

4. Rooms
    - Rooms(room_no, capacity, fee_per_night)
    - Primary Key: {room_no}

5. Health_Record
    - Health_Record(health_record_id, status, date, description, disease)
    - Primary Key: {health_record_id}

6. Instruction
    - Instruction(instruction_code, fee, description)
    - Primary Key: {instruction_code}

7. Orders_For
    - Orders_For(order_date, instruction_code, physician_id, patient_id)
    - Foreign Key: {instruction_code references Instruction(instruction_code), physician_id references Physician(physician_id), patient_id references Patient(patient_id)}

8. Executes_For
    - Executes_For(execute_date, result_status, instruction_code, nurse_id, patient_id)
    - Foreign Key: {instruction_code references Instruction(instruction_code), nurse_id references Nurse(nurse_id), patient_id references Patient(patient_id)}

9. Invoice
    - Invoice(invoice_id, patient_id, room_no, room_fee, instruction_fee)
    - Primary Key: {invoice_id}
    - Foreign Key: {patient_id references Patient(patient_id), room_no references Rooms(room_no)}

10. Payment
    - Payment(payment_id, patient_id, invoice_id, payment_date, amount)
    - Primary Key: {payment_id}
    - Foreign Key: {patient_id references Patient(patient_id), invoice_id references Invoice(invoice_id)}

11. Medications
    - Medications(medication_id, name)
    - Primary Key: {medication_id}

12. Gives_Med
    - Gives_Med(medication_id, nurse_id, patient_id, amount)
    - Foreign Key: {medication_id references Medication(medication_id), nurse_id references Nurse(nurse_id), patient_id references Patient(patient_id)}

13. Monitor_By
    - Monitor_By(physician_id, patient_id, duration)
    - Foreign Key: {physician_id references Physician(physician_id), patient_id references Patient(patient_id)}

5. Views and Descriptions

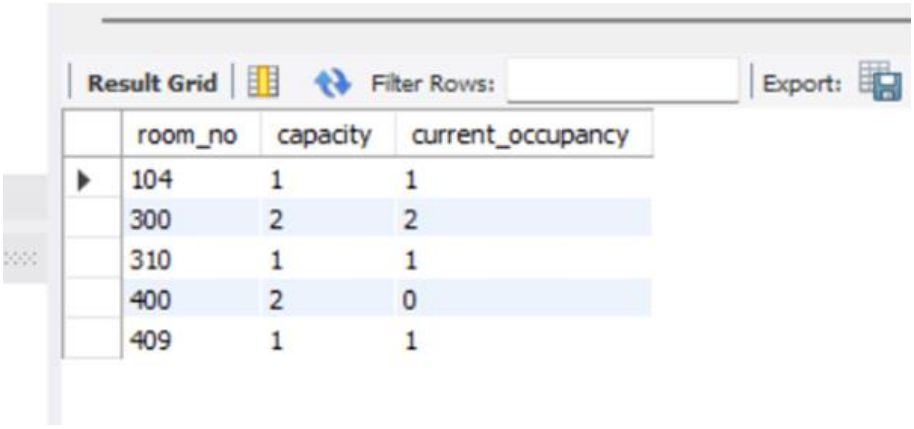
View 1.

The "room_status_view" is a predefined SQL query that combines data from the "rooms" table and the "patient" table to provide a snapshot of the current state of each room. It shows the room number, its capacity, and the current occupancy of each room. This SQL statement joins the "rooms" and "patient" tables based on the "room_no" field. It then groups the data by "room_no" to calculate the number of patients (current occupancy) in each room.

The "room_status_view" is useful to our hospital database system, not only for its operational benefits but also as a security measure. It helps maintain data privacy and integrity by restricting access to critical information. This view only exposes essential details, such as room occupancy and capacity, while hiding sensitive data, such as personal details of patients or staff.

```
CREATE VIEW room_status_view AS
```

```
SELECT r.room_no, r.capacity, COUNT(p.patient_id) AS current_occupancy
FROM rooms r
LEFT JOIN patient p ON r.room_no = p.room_no
GROUP BY r.room_no;
```



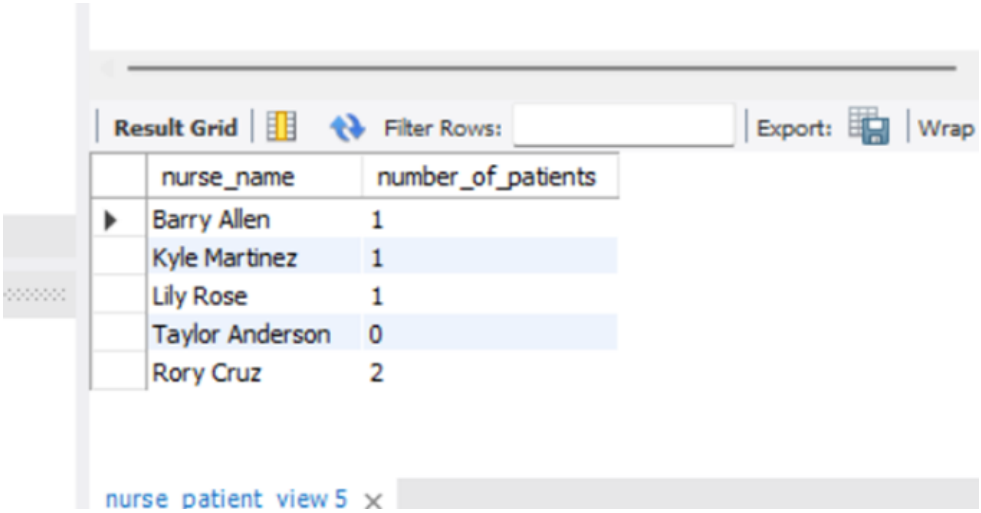
room_no	capacity	current_occupancy
104	1	1
300	2	2
310	1	1
400	2	0
409	1	1

View 2.

The nurse_patient_view is designed to provide an easy-to-understand summary of the patient load for each nurse. This view aggregates data from the nurse and gives_med tables to calculate the number of patients that each nurse is currently attending to. This view links the nurse and gives_med tables via a LEFT JOIN, based on matching nurse_id fields. The results are then grouped by nurse_id, ensuring the patient count is separate for each nurse. The nurse_patient_view offers valuable insights into the distribution of workload among nursing staff by presenting the total number of patients each nurse is assigned. Regarding data sensitivity, this view only exposes necessary information (nurse names and the number of patients they're caring for) while hiding other potentially sensitive data in the underlying tables. For instance, details about the medication being given to patients by nurses are not revealed in this view. This way, even if users have access to this view, they will not have access to certain sensitive data, thereby adhering to principles of information security and privacy.

```
CREATE VIEW nurse_patient_view AS
SELECT n.name AS nurse_name, COUNT(gm.patient_id) AS number_of_patients
FROM nurse n
```

```
LEFT JOIN gives_med gm ON n.nurse_id = gm.nurse_id
GROUP BY n.nurse_id;
```

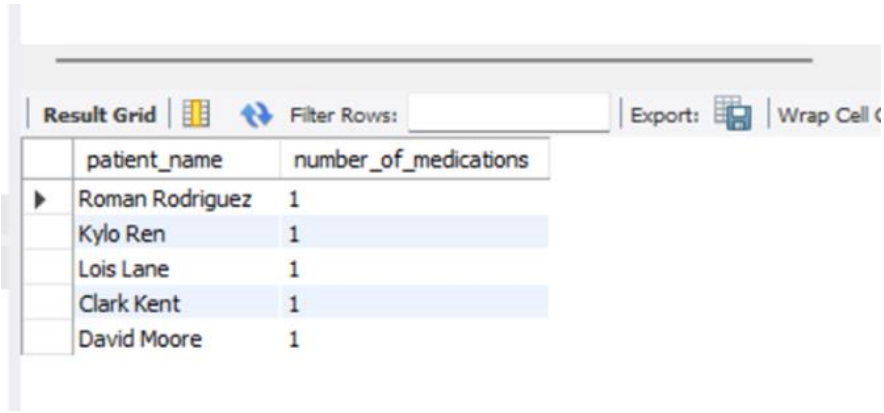


View 3.

This view shows the number of medications each patient is currently taking. This is important to monitor each patient's medication intake and ensure they are getting the correct treatments. The patient_medication_view is a SQL view that presents a summarized perspective of each patient's medication usage within the hospital. It uses the JOIN operation to combine the patient and gives_med tables, correlating the patient's name with the total number of unique medications they're currently taking. This count is achieved by grouping the records by patient_id and applying a count operation on medication_id. By only displaying the patient's name and the total count of their medications, this view not only provides essential information but also ensures data security and confidentiality. No sensitive information such as specific medication details, address, or phone number of the patient is exposed, adhering to principles of minimum necessary access. This view is particularly beneficial for the dataset as it enables healthcare professionals to quickly identify patients with high medication counts, which can be indicative of complex treatment

```
CREATE VIEW patient_medication_view AS
```

```
SELECT pt.name AS patient_name, COUNT(gm.medication_id) AS number_of_medications

FROM patient pt

JOIN gives_med gm ON pt.patient_id = gm.patient_id

GROUP BY pt.patient_id;
```



Result Grid | Filter Rows: | Export: | Wrap Cell

patient_name	number_of_medications
Roman Rodriguez	1
Kylo Ren	1
Lois Lane	1
Clark Kent	1
David Moore	1

6. Triggers and descriptions

Trigger 1.

This trigger decreases the capacity of the room by one each time a new patient is assigned to a room. This trigger is particularly useful for maintaining data consistency and integrity within our database. In a hospital management system, it is important to keep track of the available capacity in each room in order to prevent overbooking. By automating the process of reducing room capacity when a new patient is added, we minimize the risk of human error and ensure that the room capacity value is always accurate. This trigger ensures that whenever a new patient is assigned to a room, the capacity of that room is automatically reduced, accurately reflecting the number of free spaces remaining in that room.

```
DELIMITER //
```

```
CREATE TRIGGER new_patient_room_assignment

AFTER INSERT ON patient

FOR EACH ROW

BEGIN

        UPDATE rooms SET capacity = capacity - 1 WHERE room_no = NEW.room_no;

end //

DELIMITER ;
```

Trigger 2.

The new_payment_trigger is a trigger that is activated after a new entry is made into the payment table, which records the payments made by patients. The trigger updates the invoice table, reducing the remaining balance by the amount paid. It uses the UPDATE SQL statement to modify the total_amount field of the invoice table, decreasing it by the amount of the new payment (NEW.amount). This operation takes place where the invoice_id in the invoice table matches the invoice_id of the newly inserted payment (NEW.invoice_id).

This trigger is crucial for the database as it automates the process of updating the balance in the invoice table whenever a payment is made. Without it, the invoice table wouldn't be up to date, which could lead to inconsistencies and errors in financial management. Furthermore, from a data security perspective, the trigger enforces integrity constraints at the database level, which reduces the likelihood of human error.

```
DELIMITER //

CREATE TRIGGER new_payment_trigger

AFTER INSERT ON payment

FOR EACH ROW

BEGIN
```

```
    UPDATE invoice SET total_amount = total_amount - NEW.amount WHERE invoice_id
= NEW.invoice_id;

END //

DELIMITER ;
```

Trigger 3.

The room_fee_update trigger is designed to automatically adjust the room_fee field in the invoice table whenever there's an update to the fee_per_night field in the rooms table. The body of the trigger is composed of an SQL UPDATE statement, which targets the invoice table. It sets the room_fee equal to the new fee_per_night of the room that was just updated (NEW.room_no). This trigger is useful for our database as it simplifies the management of room fees in relation to invoices. Without it, any time a room's nightly fee is updated, one would have to manually track down all relevant invoices and adjust their room_fee values accordingly, a process that could cause errors and is inefficient.

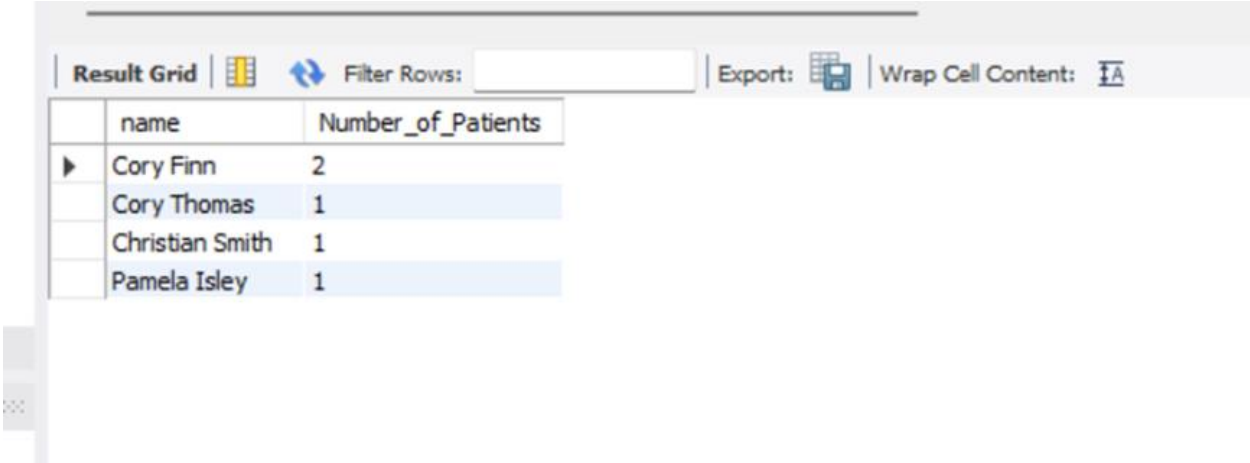
```
DELIMITER //

CREATE TRIGGER room_fee_update

AFTER UPDATE ON rooms

FOR EACH ROW

BEGIN

        UPDATE invoice SET room_fee = NEW.fee_per_night WHERE room_no =
NEW.room_no;

END //

DELIMITER ;
```

7. Queries, Descriptions, and Results

Query 1 (join1).

This query retrieves the names of all physicians and counts how many patients each physician has provided medical instructions for. It operates by joining the 'physician' and 'orders_for' tables, grouping the data by physician name, and counting the number of unique patient IDs associated with each physician via the medical instructions. The result is then sorted in descending order of patient count, so physicians who have provided medical instructions to the most patients are listed first.

```
SELECT physician.name, COUNT(*) AS Number_of_Patients
FROM physician
INNER JOIN orders_for
ON physician.physician_id = orders_for.physician_id
GROUP BY physician.name
ORDER BY Number_of_Patients DESC;
```



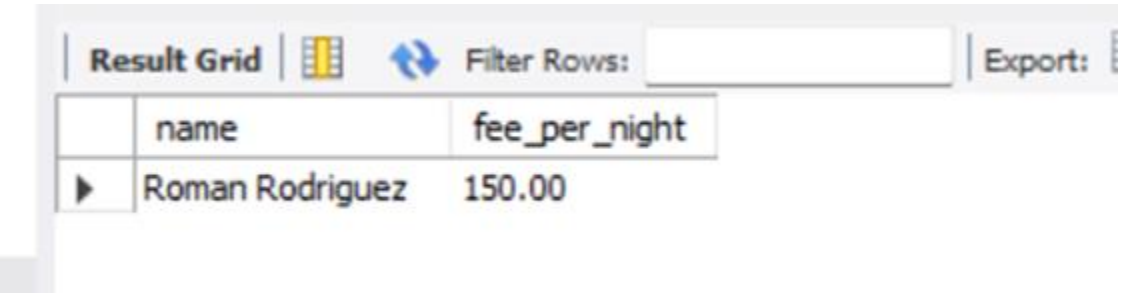
name	Number_of_Patients
Cory Finn	2
Cory Thomas	1
Christian Smith	1
Pamela Isley	1

Query 2 (join2).

This query retrieves the names of all patients staying in rooms that cost more than $100 per night. It operates by joining the 'patient' and 'rooms' tables based on the room number and then filtering to only include rooms with a 'fee_per_night' value greater than 100.

```
SELECT patient.name, rooms.fee_per_night
```

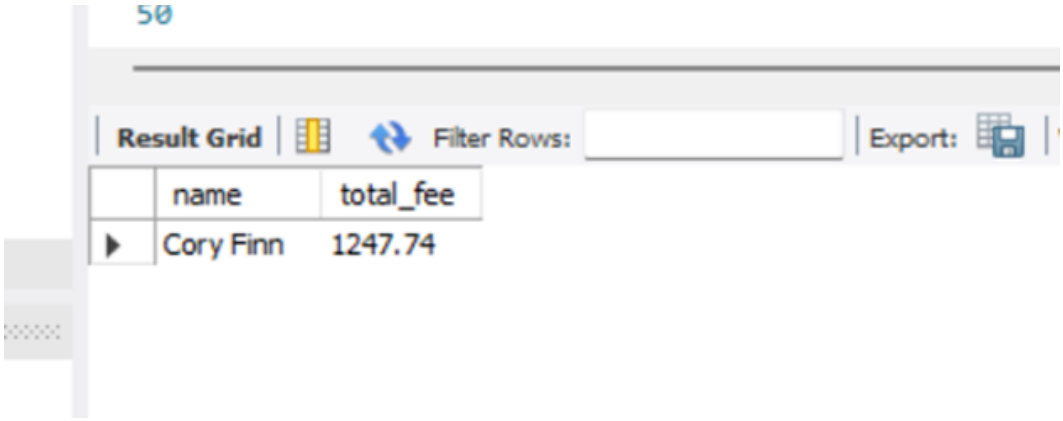
```
FROM patient

INNER JOIN rooms

ON patient.room_no = rooms.room_no

WHERE rooms.fee_per_night > 100;
```



Query 3 (join3).

This query returns the names of physicians and the sum of the fees of their respective orders, but only for those physicians where the total fee is greater than $1000. The join operation is used to link the physician, orders_for, and instruction tables.

```
SELECT physician.name, SUM(instruction.fee) AS total_fee

FROM physician

JOIN orders_for ON physician.physician_id = orders_for.physician_id

JOIN instruction ON orders_for.instruction_code = instruction.instruction_code

GROUP BY physician.name

HAVING total_fee > 1000;
```

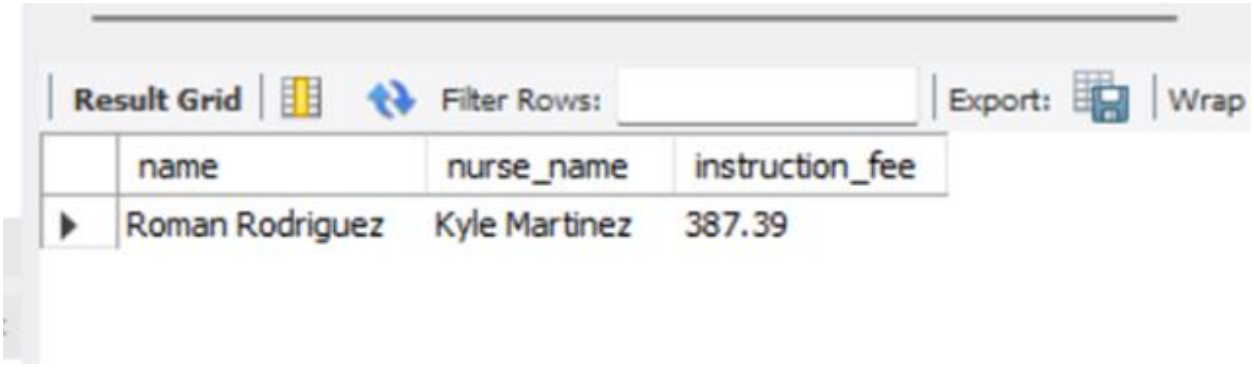


Query 4 (nested 1).

This query is intended to retrieve the names of patients, the nurses who executed instructions for them, and the fees of those instructions, for patients who are staying in rooms where the fee per night is greater than the average room fee per night in the hospital. The innermost query calculates the average room fee per night. The next level of the nested query identifies the room numbers where the fee per night is greater than this average. Finally, the outer query retrieves the relevant information for the patients who are staying in these rooms.

```
SELECT p.name, n.name AS "nurse_name", i.fee AS "instruction_fee"
FROM patient p
INNER JOIN executes_for ef ON p.patient_id = ef.patient_id
INNER JOIN nurse n ON ef.nurse_id = n.nurse_id
INNER JOIN instruction i ON ef.instruction_code = i.instruction_code
WHERE p.room_no IN (
        SELECT r.room_no
        FROM rooms r
        WHERE r.fee_per_night > (
        SELECT AVG(r2.fee_per_night)
```

```
        FROM rooms r2
        )
);
```



Result Grid | Filter Rows: | Export: | Wrap

	name	nurse_name	instruction_fee
▶	Roman Rodriguez	Kyle Martinez	387.39

Query 5 (nested 2).

This query retrieves the names of the physicians who have ordered an instruction with a fee higher than the average fee of all instructions. This query works in the following way:

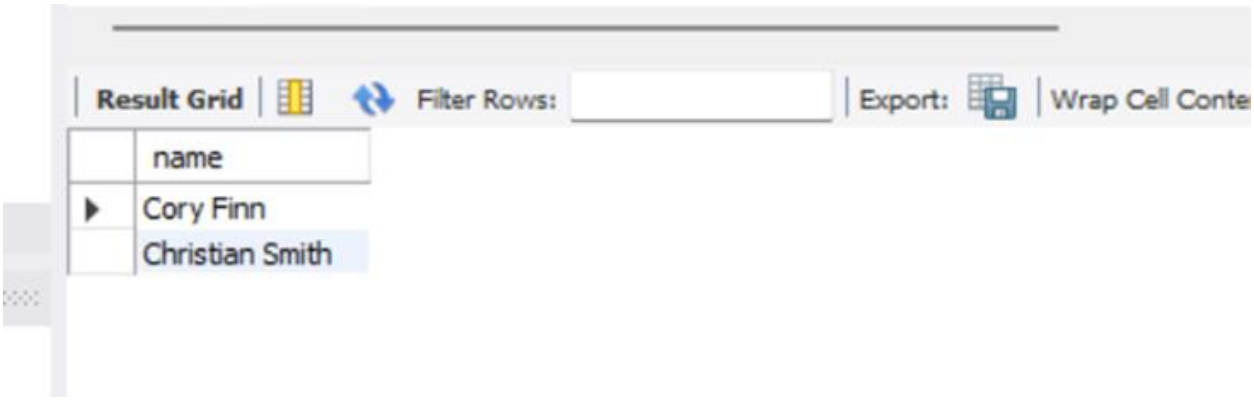
· The innermost query calculates the average fee of all instructions.

· The next layer of the subquery selects instruction codes from the instruction table that have a fee greater than the average calculated in the previous step.

· The outermost query then selects physicians who have ordered these high-fee instructions.

```
SELECT name
FROM physician
WHERE physician_id IN (
        SELECT physician_id
        FROM orders_for
        WHERE instruction_code IN (
```

```
        SELECT instruction_code
        FROM instruction
        WHERE fee > (SELECT AVG(fee) FROM instruction)
        )
);
```

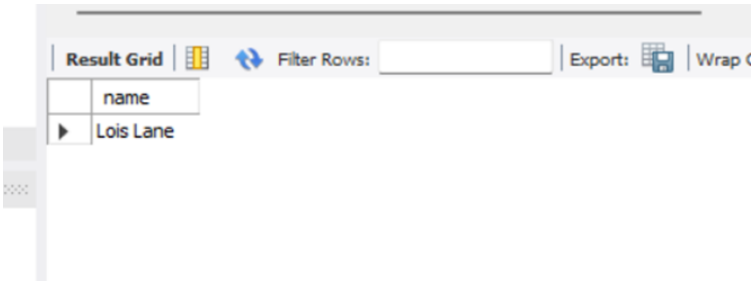


Query 6 (nested 3).

This query retrieves the names of patients who have received instructions from physicians who specialize in Cardiology. This query works by first selecting the ids of physicians in the field of Cardiology. It then uses those ids to select the ids of patients who have been given orders by those physicians. Finally, it uses those patient ids to select the names of the patients from the “patient” table.

```
SELECT p.name
FROM patient p
WHERE p.patient_id IN (
   SELECT o.patient_id
   FROM orders_for o
   WHERE o.physician_id IN (
        SELECT ph.physician_id
```

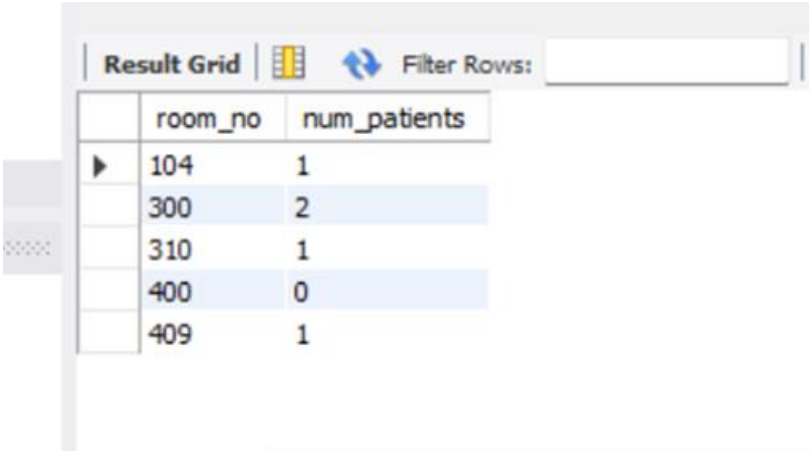
```
        FROM physician ph
        WHERE ph.field_of_expertise = 'Cardiology'
    )
);
```



Query 7 (aggregation 1).

This query retrieves the count of patients in each room. It does this by joining the “rooms” table and “patient” table on the room_no field, then grouping by room_no. The COUNT function is used to count the number of patients in each room.

```
SELECT rooms.room_no, COUNT(patient.patient_id) as num_patients
FROM rooms
LEFT JOIN patient ON rooms.room_no = patient.room_no
GROUP BY rooms.room_no;
```

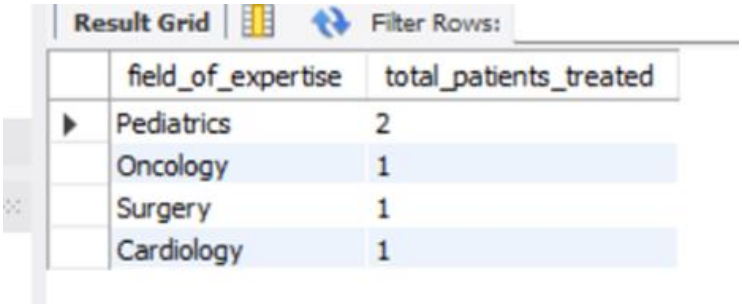


room_no	num_patients
104	1
300	2
310	1
400	0
409	1

Query 8 (aggregation 2).

This query retrieves the count of patients treated by physicians based on their field of expertise. It does this by joining the physician table and orders_for table on the physician_id field, then grouping by field_of_expertise. The COUNT function is used to count the number of patients treated in each field of expertise.

```
SELECT physician.field_of_expertise, COUNT(*) AS total_patients_treated
FROM physician
JOIN orders_for ON physician.physician_id = orders_for.physician_id
GROUP BY physician.field_of_expertise;
```

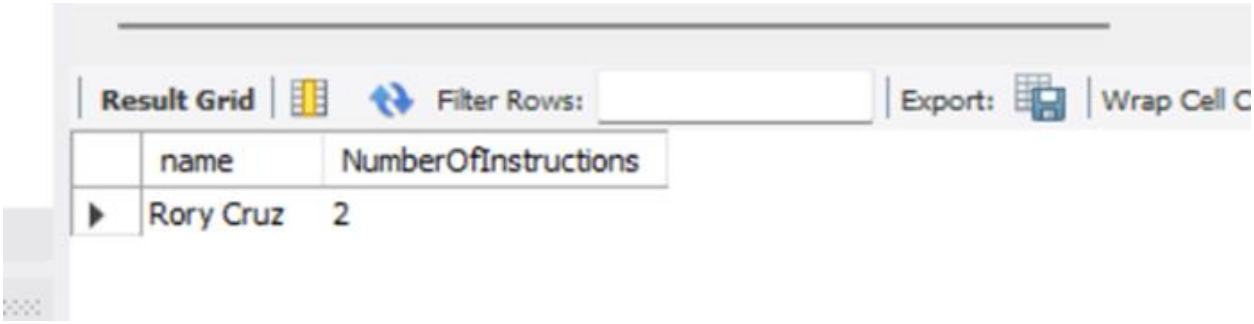


field_of_expertise	total_patients_treated
Pediatrics	2
Oncology	1
Surgery	1
Cardiology	1

Query 9 (aggregation 3).

This query is to obtain the list of nurses who have executed more than one instruction, along with the number of instructions they've executed. It joins the nurse and executes_for tables on nurse_id, and groups the result by nurse.name. The COUNT(*) function is used to count the numb er of instructions each nurse has executed. The HAVING clause then filters out the nurses who have executed only one instruction.

```
SELECT nurse.name, COUNT(*) as NumberOfInstructions

FROM nurse

JOIN executes_for ON nurse.nurse_id = executes_for.nurse_id

GROUP BY nurse.name

HAVING COUNT(*) > 1;
```

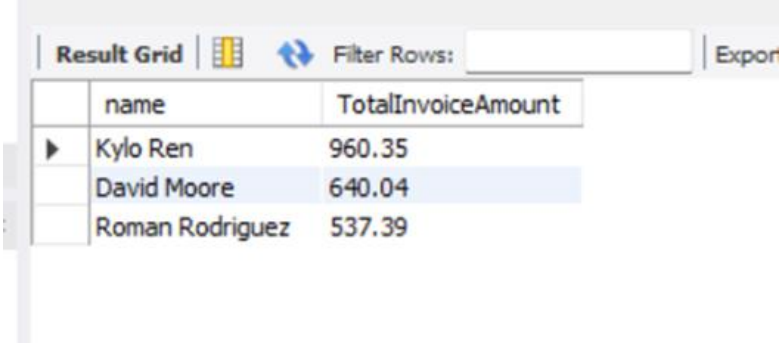


Query 10.

This SQL query is designed to find the names of the top three patients who have the highest total invoice amount, by combining both the instruction fee and the room fee. The query works by first joining the 'patient' and 'invoice' tables based on the patient's ID. It then sums up the instruction fee and the room fee for each patient and groups the results by the patient's name. The 'ORDER BY' clause sorts these results in descending order, and the 'LIMIT' clause restricts the output to the top 3 patients.

```
SELECT patient.name, SUM(invoice.instruction_fee + invoice.room_fee) as TotalInvoiceAmount

FROM patient

JOIN invoice ON patient.patient_id = invoice.patient_id
```

```
GROUP BY patient.name

ORDER BY TotalInvoiceAmount DESC

LIMIT 3;
```



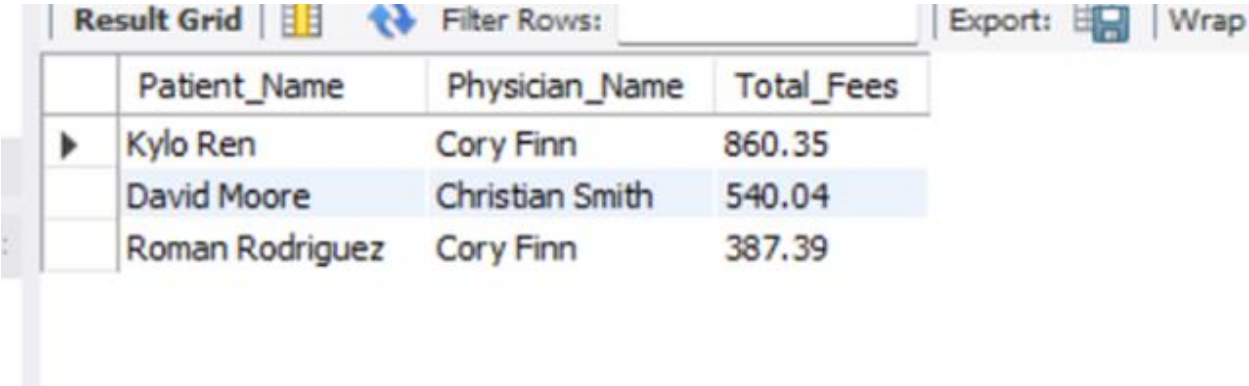
name	TotalInvoiceAmount
Kylo Ren	960.35
David Moore	640.04
Roman Rodriguez	537.39

Query 11.

This query provides information on the top 3 patients who have accrued the highest total instruction fees, along with the corresponding physicians who provided the instructions. The query joins the patient, orders_for, physician, and instruction tables together to form a comprehensive view of the patient's interactions with their physicians and the instructions given. The SUM function is used in conjunction with GROUP BY to calculate the total instruction fees per patient-physician pairing. The results are then ordered in descending order of total fees, and the LIMIT clause is used to restrict the output to the top 3 pairings with the highest total instruction fees.

```
SELECT p.name AS Patient_Name, phy.name AS Physician_Name, SUM(ins.fee) AS Total_Fees

FROM patient p

JOIN orders_for o ON p.patient_id = o.patient_id

JOIN physician phy ON o.physician_id = phy.physician_id

JOIN instruction ins ON o.instruction_code = ins.instruction_code

GROUP BY p.patient_id, phy.physician_id

ORDER BY Total_Fees DESC
```

```
LIMIT 3;
```

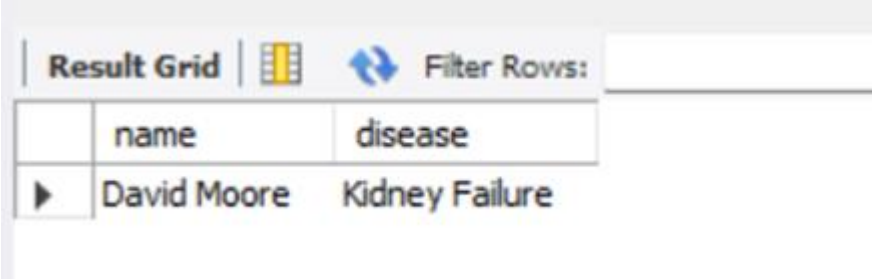


	Patient_Name	Physician_Name	Total_Fees
▶	Kylo Ren	Cory Finn	860.35
	David Moore	Christian Smith	540.04
	Roman Rodriguez	Cory Finn	387.39

Query 12.

This query retrieves the names of all patients who are diagnosed with the disease of kidney Failure '. The 'JOIN' statement is used to combine rows from two tables, which are 'patient' and 'health_record_id'.

```
SELECT p.name, h.disease

FROM patient p

JOIN health_record h ON p.health_record_id = h.health_record_id

WHERE h.disease = 'Kidney Failure';
```

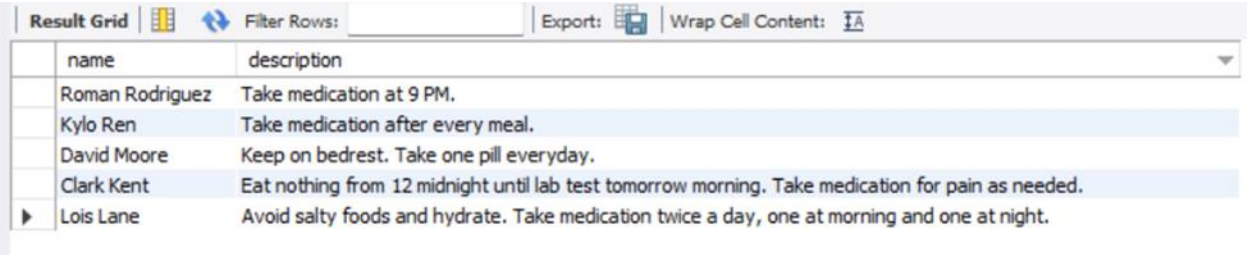


	name	disease
▶	David Moore	Kidney Failure

Query 13.

This query will provide the name of each patient and the description of the latest instruction that the patient received. It's achieved by joining the patient, orders_for, and instruction tables. The subquery is used to find the latest date (MAX(order_date)) when an instruction was ordered for each patient. The result will be a list of patient names and descriptions of the latest instructions they received. If there are patients who have not received any instructions, they will not appear in the result.

```
SELECT p.name, i.description

FROM patient p

JOIN orders_for o ON p.patient_id = o.patient_id

JOIN instruction i ON o.instruction_code = i.instruction_code

WHERE o.order_date = (SELECT MAX(order_date) FROM orders_for WHERE
patient_id = p.patient_id);
```

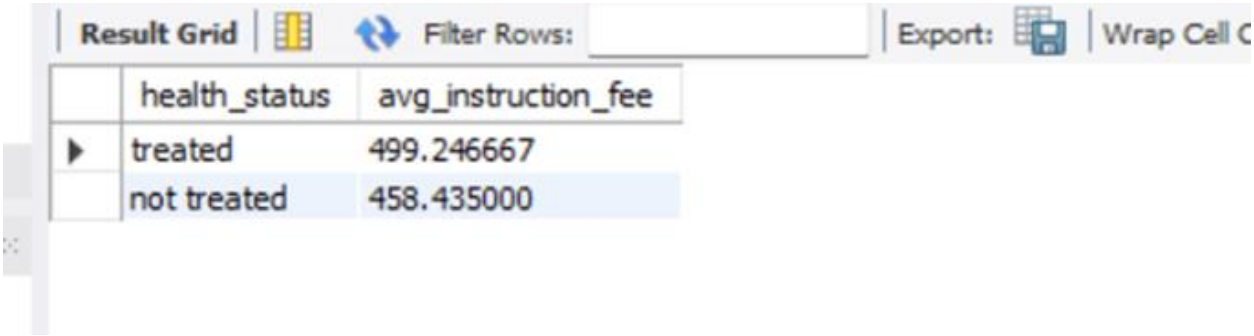

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

name	description
Roman Rodriguez	Take medication at 9 PM.
Kylo Ren	Take medication after every meal.
David Moore	Keep on bedrest. Take one pill everyday.
Clark Kent	Eat nothing from 12 midnight until lab test tomorrow morning. Take medication for pain as needed.
Lois Lane	Avoid salty foods and hydrate. Take medication twice a day, one at morning and one at night.

Query 14.

This query returns the average instruction fee for each type of health status. We're using INNER JOIN clauses to join health_record, patient, orders_for, and instruction tables based on their common fields (health_record_id, patient_id, and instruction_code). This allows us to link instructions to health records through the patients. The AVG(instruction.fee) calculates the average instruction fee for each type of health status. The GROUP BY clause groups the results by the health status, so the average of the instruction fees is calculated separately for each status.

```
SELECT
        health_record.status AS health_status,
        AVG(instruction.fee) AS avg_instruction_fee
FROM health_record
INNER JOIN patient ON health_record.health_record_id = patient.health_record_id
INNER JOIN orders_for ON patient.patient_id = orders_for.patient_id
INNER JOIN instruction ON orders_for.instruction_code = instruction.instruction_code
GROUP BY health_record.status;
```



health_status	avg_instruction_fee
treated	499.246667
not treated	458.435000

Query 15.

This query returns the physician_id and name of physicians who have not given any orders in the year 2023. The inner subquery selects distinct physician_id from the orders_for table for orders made between '2023-01-01' and '2023-12-31'. The outer query selects the physician_id and name from the physician table where the physician_id is not in the list retrieved from the inner subquery.

```
SELECT physician_id, name
FROM physician
WHERE physician_id NOT IN (
        SELECT DISTINCT physician_id
        FROM orders_for
```

```
    WHERE order_date BETWEEN '2023-01-01' AND '2023-12-31'
);
```

8. Transactions and Descriptions

Transaction 1.
This transaction updates the room_no where a patient is staying within the Patient table and updates the invoice for the corresponding room_fee of the new room assignment in the Invoice table. This transaction ensures that the records of where the patient is staying is updated when they are assigned a new room as well as reflecting the correct amount on their invoice that they would be charged with after their stay at the hospital.

```
BEGIN;

UPDATE patient
SET room_no = 400
WHERE patient_id = 7283495;

UPDATE invoice
SET room_no = 400, room_fee = 80.00
where patient_id = 7283495;

COMMIT;
```

Transaction 2.

This transaction retrieves a patient's health record by searching for their name in the Patient table then stores the health_record_id into a user-defined variable @hr_id. In doing so, we can search and update the Health_Record table for the specific patient using the @hr_id variable and update the status to "treated". This transaction lets the physician and nurse update a patient's health record once the patient has recovered and completed their treatment at the hospital. If the patient's health record is ever referenced by other medical professionals in the future, they will know that the patient finished their treatment for their disease and completed their medication as prescribed by their previous doctor.

```
BEGIN;

SELECT health_record_id INTO @hr_id
FROM patient
WHERE patient_name = "Clark Kent";

UPDATE health_record
SET status = "treated"
WHERE health_record_id = @hr_id;

COMMIT;
```