Group Members:
Raudel Vargas(In-Person) rvarga32
Laxmishivaprasad Sandhireddy(In-Person) lsandh3
Ricardo Gonzalez Diaz(In-Person) rgonza82

Part 1: Assumptions

1. Not every patient has a Health Record. (Note that Patient does not totally participate in the relationship). This decision was made based on the following instruction: 'A patient "may" have a health record'. Implying what we just mentioned.

2. A patient may be monitored by more than one physician. This is implied by the following phrase: 'Every patient has "some" physicians who monitor the patient'

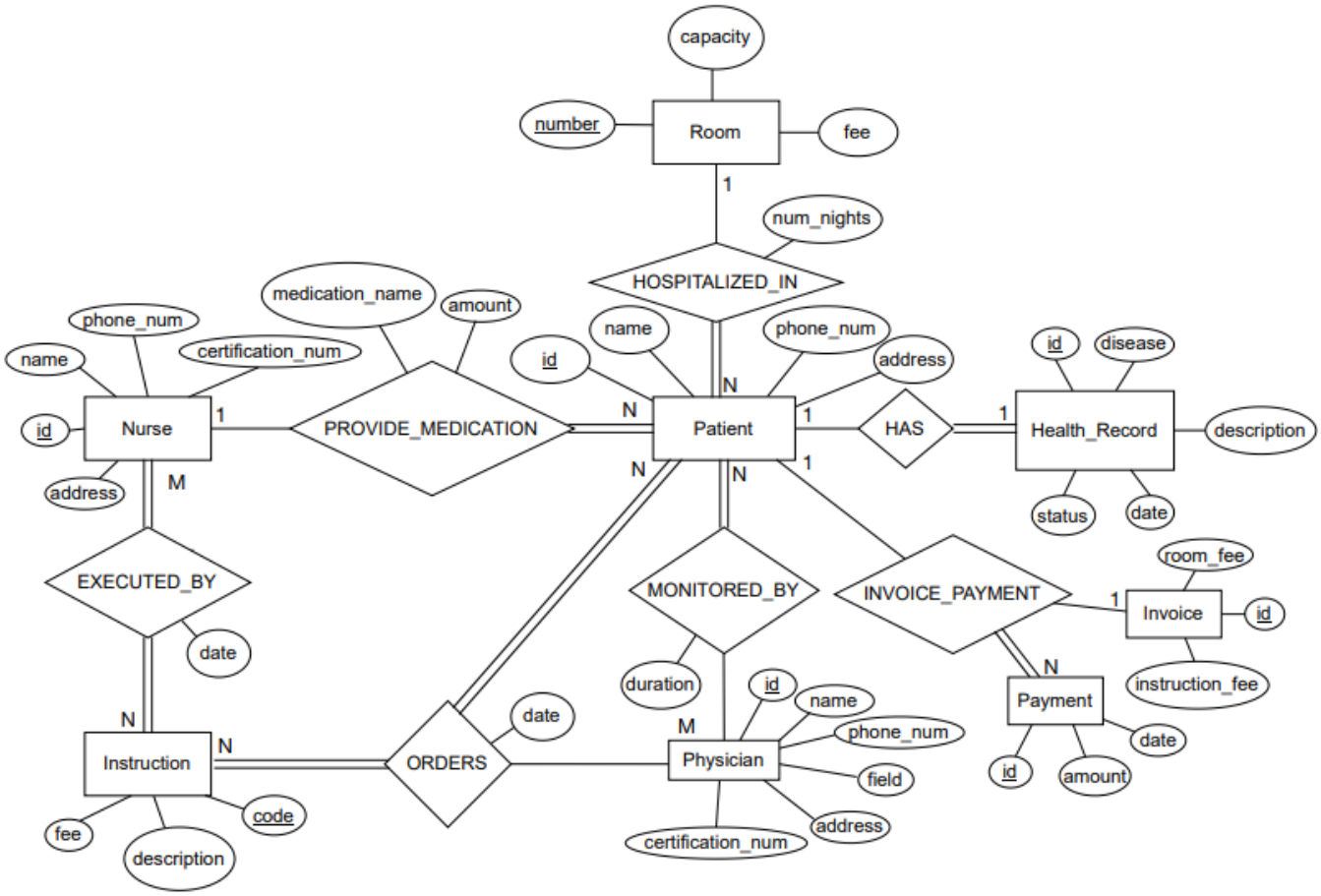
3. A Physician orders an instruction, and the instruction is then executed by a nurse.

4. Every nurse executes instructions. Therefore, Nurse totally participates in the EXECUTED_BY relationship.

5. There are nurses that may not provide medication for patients. This is why they do not totally participate in the relationship PROVIDE_MEDICATION.

6. Every Patient is assigned a single room. However, a single room may hold more than one patient.

2. (E)ERD


capacity
number
Room
fee
1
num_nights
HOSPITALIZED_IN
medication_name
amount
phone_num
name
certification_num
id
Nurse
1
PROVIDE_MEDICATION
N
N
Patient
1
HAS
1
Health_Record
description
id
disease
status
date
address
M
N
N
1
EXECUTED_BY
date
MONITORED_BY
INVOICE_PAYMENT
room_fee
1
Invoice
id
instruction_fee
N
Payment
date
id
amount
duration
id
name
phone_num
field
address
certification_num
M
Physician
N
N
Instruction
ORDERS
date
fee
description
code

3. Relations and Keys

Patients (Patient_ID, name, phone_num, address, nurse_id, medication_name, medication_amount, room_number, num_nights)

Primary key : {Patient_ID}

Foreign key : { nurse_id REFERENCES NURSES(nurse_id) , room_number REFERENCES ROOMS(room_number) }

—----------

Health_records(health_id, disease,status, date, description, patient_id)

Primary key: {health_id}

Foreign key : { patient_id REFERENCES PATIENTS(patient_id)}

—----------

Invoices(invoice_id, instruction_fee, room_fee, patient_id)

Primary key: {invoice_id}

Foreign key :{ patient_id REFERENCES PATIENTS(patient_id) }

—----------

Payments(payment_id, amount, date, patient_id)

Primary key: {payment_id}

Foreign key : { patient_id REFERENCES PATIENTS(patient_id) }

—----------

Physicians(physician_id, name, phone_num, field, address, certification_num)

Primary key: { physician_id }

—----------

Instructions(Instruction_code, description, fee)

Primary key: { Instruction_code }

—----------

Nurses(nurse_id, address, name, phone_num, certification_num)

Primary key: { nurse_id }

—----------

Rooms(room_number, capacity, fee)

Primary key:{ room_number }

—----------

Invoice_payments(patient_id, invoice_id, payment_id)

Primary key: {patient_id, invoice_id, payment_id }

Foreign key : { patient_id REFERENCES PATIENTS(patient_id), invoice_id REFERENCES INVOICES(invoice_id), payment_id REFERENCES PAYMENTS(payment_id) }

—----------

Monitored(patient_id, physician_id, duration)

Primary key : { patient_id, physician_id}

Foreign key : { patient_id REFERENCES PATIENTS(patient_id), physician_id REFERENCES PHYSICIANS(physician_id) }

—----------

Orders(patient_id, instruction_code, physician_id, order_date)

Primary key: { patient_id, instruction_code, physician_id }

Foreign key : { patient_id REFERENCES PATIENTS(patient_id), instruction_code REFERENCES INSTRUCTIONS(instruction_code), physician_id REFERENCES PHYSICIANS(physician_id) }

—----------

Executed(instruction_code, nurse_id, date)

Primary key: { instruction_code,nurse_id }

Foreign key : { instruction_code REFERENCES INSTRUCTIONS(instruction_code), nurse_id REFERENCES NURSES(nurse_id) }

4. Views and Descriptions

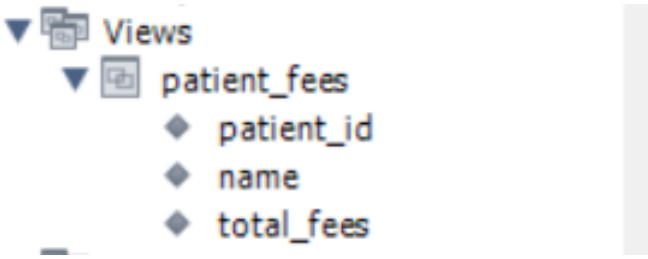
Description/Query

I is helpful as provides an easy and efficient way to access patient fee information, which can be helpful for various use cases such as financial reporting, monitoring revenue streams, or identifying high-cost patients. Instead of writing complex SQL queries each time to calculate patient fees, users can simply query the patient_fees view to retrieve all relevant information.

```sql
-- This view query shows the total fees paid by each patient.

CREATE VIEW patient_fees AS
SELECT p.patient_id, p.name, SUM(i.instruction_fee + i.room_fee) AS total_fees
FROM PATIENTS p
JOIN INVOICES i ON p.patient_id = i.patient_id
GROUP BY p.patient_id, p.name;
```

Output



Description/Query

By analyzing changes in disease status and other health record information over time, healthcare providers can identify trends and patterns that may inform treatment decisions.

```sql
-- This view that shows all patients with their corresponding health records,
--  including only the most recent record for each patient

CREATE VIEW patient_latest_health_record AS
SELECT h.patient_id, p.name AS patient_name, h.health_id,
       h.disease, h.status, h.date, h.description
FROM PATIENTS p
JOIN HEALTH_RECORDS h ON p.patient_id = h.patient_id
WHERE h.date = (SELECT MAX(date) FROM HEALTH_RECORDS WHERE patient_id = h.patient_id);
```

Output

- patient_latest_health_record
  - patient_id
  - patient_name
  - health_id
  - disease
  - status
  - date
  - description

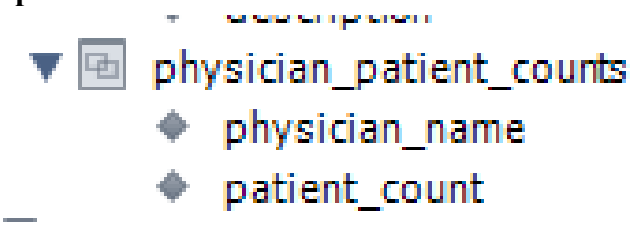
Description/Query

By monitoring patient counts for each physician, healthcare providers can identify physicians who may be overworked or struggling to keep up with patient demand. This can inform staffing decisions and help ensure that patients receive timely and appropriate care.

```
-- This view that all physicians along with the number of patients they are currently monitoring:


CREATE VIEW physician_patient_counts AS
SELECT ph.name AS physician_name, COUNT(m.patient_id) AS patient_count
FROM PHYSICIANS ph
LEFT JOIN MONITORED m ON ph.physician_id = m.physician_id
GROUP BY ph.physician_id;
```

Output



5. Triggers and descriptions

Description/Query

By updating the capacity of a room automatically when a new patient is added, healthcare providers can ensure that room capacity is managed effectively and that patient comfort and safety are maintained.

```
--  This trigger query updates the room's capacity when a patient is added to the room

DELIMITER //

CREATE TRIGGER update_room_capacity
AFTER INSERT ON PATIENTS
FOR EACH ROW
BEGIN
    UPDATE ROOMS
    SET capacity = capacity - 1
    WHERE room_number = NEW.room_number;
END //

DELIMITER ;
```

Description/Query

By updating the capacity of a room automatically when a patient is removed, healthcare providers can ensure that room capacity is managed effectively and that patient comfort and safety are maintained.

```
--  This trigger query updates the room's capacity when a patient is removed from a room


DELIMITER //

CREATE TRIGGER update_room_capacity
AFTER DELETE ON PATIENTS
FOR EACH ROW
BEGIN
    UPDATE ROOMS
    SET capacity = capacity + 1
    WHERE room_number = OLD.room_number;
END; //

DELIMITER ;
```

Description/Query

By updating the patient status automatically when a new health record is inserted, healthcare providers can ensure that patient status is accurately tracked over time, improving patient care and outcomes.

```
-- This trigger fires after each insert operation on the HEALTH_RECORDS table and
-- updates the PATIENTS table by setting the status column to the value of the
-- status column in the most recent HEALTH_RECORDS record for that patient.

DELIMITER //
CREATE TRIGGER update_patient_status
AFTER INSERT ON HEALTH_RECORDS
FOR EACH ROW
BEGIN
  UPDATE PATIENTS p
  SET p.status = NEW.status
  WHERE p.patient_id = NEW.patient_id
  AND NEW.date = (SELECT MAX(date) FROM HEALTH_RECORDS WHERE patient_id = NEW.patient_id);
END //
DELIMITER ;
```

6- Queries, Descriptions, and Results.

Aggregation Queries

Query 1

```
-- This query calculates the average fee of all instructions in the INSTRUCTIONS table:

SELECT AVG(fee) AS avg_fee FROM INSTRUCTIONS;
```

Output 1

avg_fee
330.000000

Query 2

```
-- This query retrieves counts the number of patients assigned to each nurse in the NURSES table

SELECT nurse_id, COUNT(patient_id) AS num_patients FROM PATIENTS
GROUP BY nurse_id;
```

Output 2

nurse_id	num_patients
2001	2
2002	1
2003	1
2004	1

Query 3

```
-- This query calculates the total revenue generated by each physician in the PHYSICIANS table:

SELECT ph.name, SUM(inv.instruction_fee + inv.room_fee) AS total_revenue FROM PHYSICIANS ph
JOIN MONITORED m ON ph.physician_id = m.physician_id
JOIN PATIENTS p ON m.patient_id = p.patient_id
JOIN INVOICES inv ON p.patient_id = inv.patient_id
GROUP BY ph.name;
```

Output 3

name	total_revenue
John Smith	1800.00
Jane Doe	1800.00
Mark Johnson	900.00

Join Queries

Query 1

```sql
-- This query retrieves the details of patients along with their corresponding nurse
-- and room information from the Patients, Nurses, and Rooms tables using JOIN operations.

SELECT p.patient_id, p.name AS patient_name, p.phone_num AS patient_phone_num,
       n.name AS nurse_name, r.room_number, r.capacity
FROM PATIENTS p
JOIN NURSES n ON p.nurse_id = n.nurse_id
JOIN ROOMS r ON p.room_number = r.room_number;
```

Output 1

patient_id	patient_name	patient_phone_num	nurse_name	room_number	capacity
1001	John Doe	123-456-7890	Emily Lee	101	1
1002	Jane Doe	555-555-5555	Jessica Chen	102	2
1003	Bob Smith	111-111-1111	William Wong	103	2
1004	Samantha Johnson	222-222-2222	Michael Smith	104	1
1005	James Williams	333-333-3333	Emily Lee	105	1

Query 2

```sql
-- This query retrieve the total amount of payments made by each patient:
-- from the Patients and Payments tables using JOIN operations.

SELECT p.patient_id, p.name, SUM(pm.amount) AS total_payments
FROM PATIENTS p
JOIN PAYMENTS pm ON p.patient_id = pm.patient_id
GROUP BY p.patient_id;
```

Output 2

patient_id	name	total_payments
1001	John Doe	200.00
1002	Jane Doe	300.00
1003	Bob Smith	400.00
1004	Samantha Johnson	500.00
1005	James Williams	600.00

Query 3

```
-- This query retrieves the details of invoices along with their corresponding patient and
-- payment information from the Invoices, Patients, and Payments tables using JOIN operations.

SELECT i.invoice_id, i.instruction_fee, i.room_fee, i.patient_id,
       p.name AS patient_name, py.amount AS payment_amount, py.date AS payment_date
FROM INVOICES i
JOIN PATIENTS p ON i.patient_id = p.patient_id
JOIN PAYMENTS py ON i.patient_id = py.patient_id;
```

Output 3

invoice_id	instruction_fee	room_fee	patient_id	patient_name	payment_amount	payment_date
1	100.00	500.00	1001	John Doe	200.00	2022-02-01
2	150.00	600.00	1002	Jane Doe	300.00	2022-03-05
3	200.00	700.00	1003	Bob Smith	400.00	2022-04-10
4	250.00	800.00	1004	Samantha Johnson	500.00	2022-05-15
5	300.00	900.00	1005	James Williams	600.00	2022-06-20

Nested Queries

Query 1

```
-- This query selects the names of all physicians who are not currently monitoring any patients.

SELECT name FROM PHYSICIANS
WHERE physician_id NOT IN (
  SELECT physician_id
  FROM MONITORED
)
```

Output 1

name
Sarah Lee
David Kim

Query 2

```sql
-- This query selects the names of all patients who are assigned to nurses
-- with a certain certification number OP12345

SELECT name FROM PATIENTS
WHERE nurse_id IN (
  SELECT nurse_id
  FROM NURSES
  WHERE certification_num = 'OP12345'
)
```

Output 2

	name
▶	Bob Smith

Query 3

```sql
-- This query uses a nested subquery to count the number of patients treated by each nurse, and
-- then joins the results with the NURSES table to retrieve the names of the corresponding nurses

SELECT name, patient_count FROM NURSES
JOIN
  (SELECT nurse_id, COUNT(*) AS patient_count
   FROM PATIENTS
   GROUP BY nurse_id) AS patient_counts
ON NURSES.nurse_id = patient_counts.nurse_id
ORDER BY patient_count DESC;
```

Output 3

	name	patient_count
▶	Emily Lee	2
	Jessica Chen	1
	William Wong	1
	Michael Smith	1

General Queries

Query 1

```
-- This query finds the total fee of all invoices generated for a specific patient

SELECT SUM(instruction_fee + room_fee) AS total_fee
FROM INVOICES
WHERE patient_id = 1005;
```

Output 1

total_fee
1200.00

Query 2

```
-- This query finds the names and phone numbers of all physicians who have patients that are currently staying in the ho

SELECT DISTINCT p.name, p.phone_num FROM PHYSICIANS p
JOIN MONITORED m ON p.physician_id = m.physician_id
JOIN PATIENTS pat ON m.patient_id = pat.patient_id;
```

Output 2

name	phone_num
John Smith	123-456-7890
Jane Doe	555-555-5555
Mark Johnson	111-222-3333

Query 3

```
-- This query finds the average number of nights that all patients stays in the hospital.

SELECT AVG(num_nights) AS avg_nights
FROM PATIENTS;
```

Output 3

	avg_nights
▶	3.0000

Query 4

```
-- This query finds the names of the nurses who are assigned to patients who
-- have been staying in the hospital for more than 4 nights.


SELECT DISTINCT p.name FROM PATIENTS p
WHERE p.num_nights > 4;
```

Output 4

	name
▶	Bob Smith

Query 5

```
SELECT i.instruction_code, i.description, COUNT(*) AS num_times_ordered FROM INSTRUCTIONS i
JOIN ORDERS o ON i.instruction_code = o.instruction_code
GROUP BY i.instruction_code, i.description
ORDER BY COUNT(*) DESC
LIMIT 3;
```

Output 5

	instruction_code	description	num_times_ordered
▶	IC001	MRI Scan	1
	IC002	Blood Test	1
	IC003	X-Ray	1

Query 6

```sql
-- This query retrieves the patient name, ID, and instruction fee from the invoices table,
-- the output to the top 3 records.

SELECT p.name AS patient_name, i.patient_id, i.instruction_fee FROM INVOICES i
JOIN PATIENTS p ON i.patient_id = p.patient_id
ORDER BY i.instruction_fee DESC
LIMIT 3;
```

Output 6

patient_name	patient_id	instruction_fee
James Williams	1005	300.00
Samantha Johnson	1004	250.00
Bob Smith	1003	200.00

7- Transactions and Description

Description/Query

The provided SQL transaction ensures data integrity and consistency when updating a patient's room assignment in a healthcare facility. By grouping multiple SQL statements into a single transaction, the database ensures that the patient is assigned to the correct room and that the room capacities are updated correctly, preventing any data inconsistencies that may occur if the statements were executed separately. The use of transactions also provides a way to rollback changes in case of errors or failures, reducing

the likelihood of data corruption and minimizing the impact of data errors on the system.

```
-- This transaction query is used to transfer a patient to another room

START TRANSACTION;

-- Update the current room capacity
UPDATE rooms
SET capacity = capacity + 1
WHERE room_number = '100';

-- Update the new room capacity
UPDATE rooms
SET capacity = capacity - 1
WHERE room_number = '101';

-- Update the patient's room number
UPDATE patients
SET room_number = '101'
WHERE patient_id = '1001';

COMMIT;
```

Description/Query

This SQL transaction updates a patient's room number and decrements the capacity of the room they were previously in. This ensures data consistency in the database and prevents overbooking of rooms. By using a transaction, the changes are executed as a single atomic unit, ensuring that they are either all committed or all rolled back if an error occurs, which helps maintain data integrity.

```sql
-- This transaction query is moving the patient with ID 1001 to room number 105
START TRANSACTION;

UPDATE PATIENTS
SET room_number = 105
WHERE patient_id = 1001;

UPDATE ROOMS
SET capacity = capacity - 1
WHERE room_number = 105;

COMMIT;
```