Group Members: Quang Le (qle21)

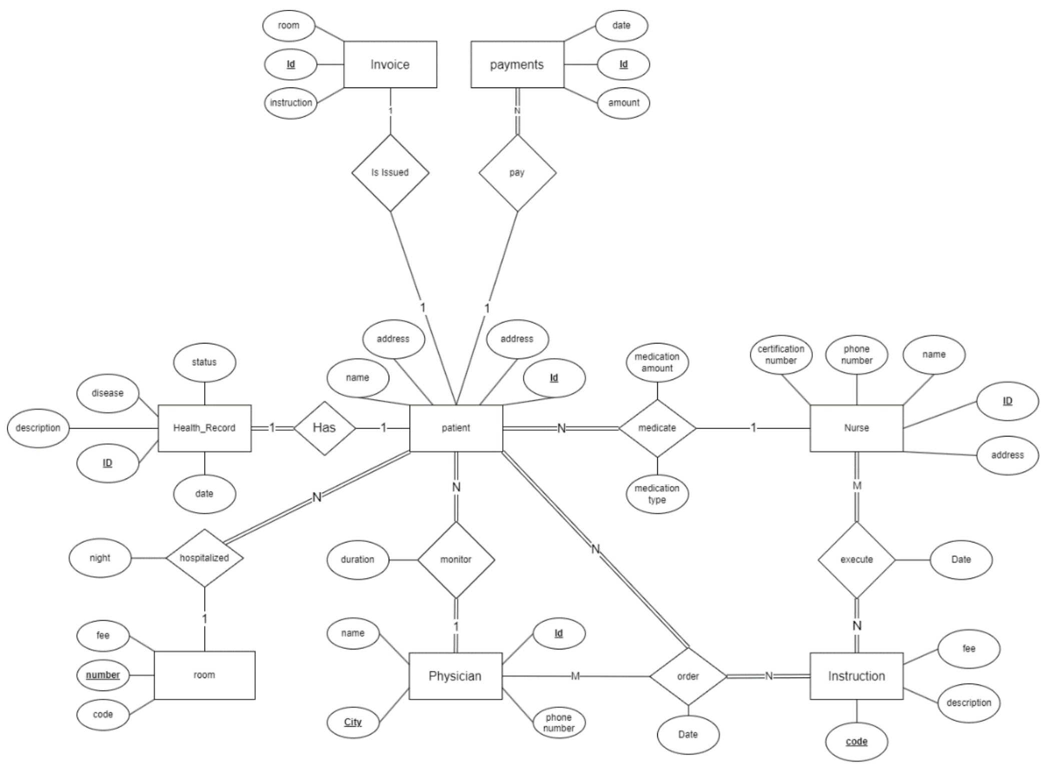
Group member: Miguel Rodriguez, Quang Le, Danyal Warraich

Part1: (E)ER design

1- Assumption

- The hospital database is designed to manage information about physicians, nurses, patients, rooms, health records, invoices, payments, and other relevant entities in a hospital setting.
- Each physician, nurse, and patient has a unique identification number (physicianID, nurseID, patientID).
- Each patient is assigned to one nurse and one room.
- Health records are associated with patients through their patientID.
- Invoices and payments are linked to patients based on their patientID.
- All monetary values are represented using the decimal data type for precision.
- Dates are represented using the date data type for accuracy.

2- (E)ERD



Part 2: Relational Mapping, Creating Database Schema, and Populating Database

3- Relations and keys

Physician(physicianID, physician_name, phone_number, field, address, certification_num)

primary key: {physicianID}

Instruction(code, description, InstructionFee)

primary key: {code}

Nurse(nurseID, address, nurse_name, phone_number, certification_num)

primary key: {nurseID}

Room(room_num, capacity, roomFee)

primary key: {room_num}

Patient(patientID, patient_name, phone_number, address, nurseID, type_of_med, amount_of_med, room_num, night_stay)

primary key: {patientID}

foreign key: {nurseID references Nurse(nurseID), room_num references Room(room_num)}

Health_Record(health_record_ID, disease, status, date, description, patientID)

primary key: {health_record_ID}

foreign key: {patientID references Patient(patientID)}

Invoice(invoiceID, InstructionFee, roomFee, patientID)

primary key: {invoiceID}

foreign key: {patientID references Patient(patientID)}

Payment(payID, amount, date, patientID)

primary key: {payID}

foreign key: {patientID references Patient(patientID)}

Issue_Pay(patientID, invoiceID, payID)

primary key: {patientID, invoiceID, payID}

foreign key: {patientID references Patient(patientID), invoiceID references Invoice(invoiceID), payID references Payment(payID)}

Monitored(patientID, physicianID, duration)

primary key: {patientID, physicianID}

foreign key: {patientID references Patient(patientID), physicianID references Physician(physicianID)}

Order(patientID, code, physicianID, order_date)

primary key: {patientID, code, physicianID}

foreign key: {patientID references Patient(patientID), code references Instruction(code), physicianID references Physician(physicianID)}

Executed(code, nurseID, date)

primary key: {code, nurseID}

foreign key: {code references Instruction(code), nurseID references Nurse(nurseID)}

Part3: Query, View, Trigger, Transaction and Final Report

4- Views and descriptions

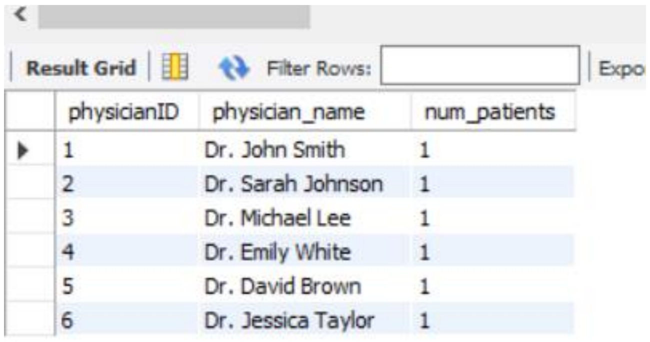
+ View 1: Physician_Patient_Count

Description: This view provides a count of the number of patients each physician is currently monitoring. The Physician_Patient_Count view allows the hospital to keep track of how many patients each physician is currently responsible for, assisting in workload management and assignment.

Query

```
CREATE VIEW Physician_Patient_Count AS

SELECT

  p.physicianID,

  p.physician_name,

  COUNT(m.patientID) AS num_patients

FROM

  Physician p

LEFT JOIN Monitor m ON p.physicianID = m.physicianID

GROUP BY p.physicianID, p.physician_name;  Result
```

Result:



physicianID	physician_name	num_patients
1	Dr. John Smith	1
2	Dr. Sarah Johnson	1
3	Dr. Michael Lee	1
4	Dr. Emily White	1
5	Dr. David Brown	1
6	Dr. Jessica Taylor	1

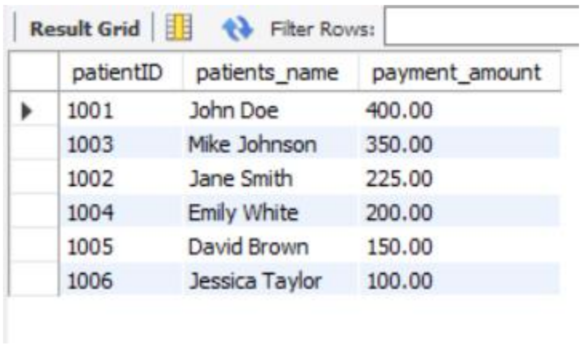
+View 2: High_Payment_Patients

Description: This view lists patients who made high payments for their treatments, ordered by the amount of payment in descending order.The High_Payment_Patients view allows the hospital to identify patients who have made substantial payments, potentially indicating the complexity or duration of their treatment.

Query:

```
CREATE VIEW High_Payment_Patients AS

SELECT

  p.patientID,

  p.patients_name,

  pa.amount AS payment_amount

FROM

  Patient p

INNER JOIN Payment pa ON p.patientID = pa.patientID

ORDER BY pa.amount DESC;
```

Result:



patientID	patients_name	payment_amount
1001	John Doe	400.00
1003	Mike Johnson	350.00
1002	Jane Smith	225.00
1004	Emily White	200.00
1005	David Brown	150.00
1006	Jessica Taylor	100.00

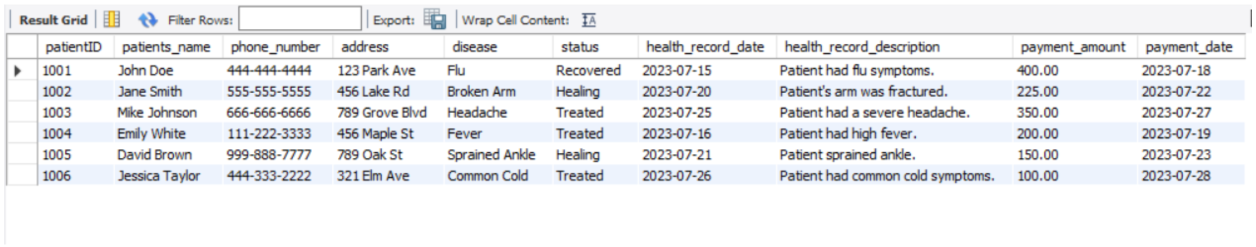
+View 3: View_Patient_Info

Description: This view combines information from the Patient, Health_Record, and Payment tables to provide comprehensive patient information, including health records and payment details.

Query:

```
CREATE VIEW View_Patient_Info AS

SELECT

  p.patientID,

  p.patients_name,

  p.phone_number,

  p.address,

  h.disease,

  h.status,

  h.date AS health_record_date,

  h.description AS health_record_description,

  py.amount AS payment_amount,

  py.date AS payment_date

FROM

  Patient p

LEFT JOIN Health_Record h ON p.patientID = h.patientID

LEFT JOIN Payment py ON p.patientID = py.patientID;
```

Result:



patientID	patients_name	phone_number	address	disease	status	health_record_date	health_record_description	payment_amount	payment_date
1001	John Doe	444-444-4444	123 Park Ave	Flu	Recovered	2023-07-15	Patient had flu symptoms.	400.00	2023-07-18
1002	Jane Smith	555-555-5555	456 Lake Rd	Broken Arm	Healing	2023-07-20	Patient's arm was fractured.	225.00	2023-07-22
1003	Mike Johnson	666-666-6666	789 Grove Blvd	Headache	Treated	2023-07-25	Patient had a severe headache.	350.00	2023-07-27
1004	Emily White	111-222-3333	456 Maple St	Fever	Treated	2023-07-16	Patient had high fever.	200.00	2023-07-19
1005	David Brown	999-888-7777	789 Oak St	Sprained Ankle	Healing	2023-07-21	Patient sprained ankle.	150.00	2023-07-23
1006	Jessica Taylor	444-333-2222	321 Elm Ave	Common Cold	Treated	2023-07-26	Patient had common cold symptoms.	100.00	2023-07-28

5- Triggers and descriptions

+Trigger 1: trig_update_health_status

Description: This trigger updates the status of a patient's health record to "Recovered" when the patient's health record status changes to "Treated" and the current date is past the date of treatment. The trig_update_health_status trigger helps to automatically change the status of a patient's health record from "Treated" to "Recovered" when the patient's treatment is completed and the current date is past the treatment date.

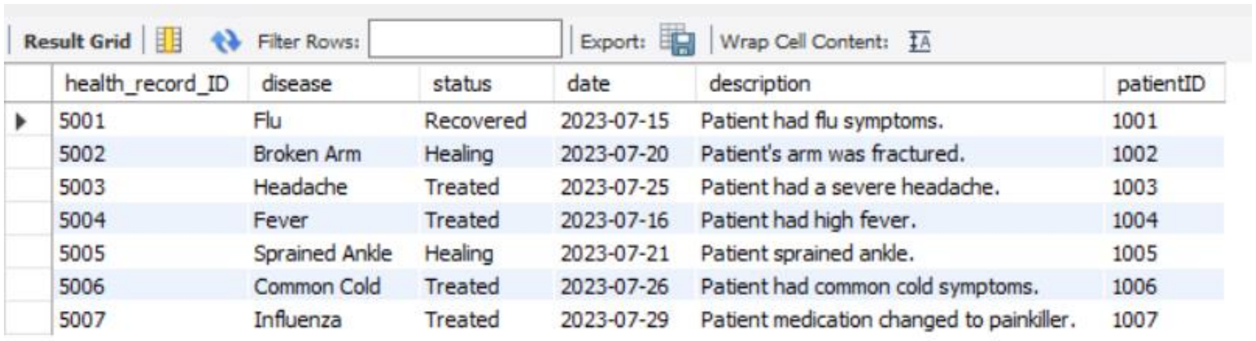
Query:

```
DELIMITER //

CREATE TRIGGER trig_update_health_status

AFTER UPDATE ON Health_Record

FOR EACH ROW

BEGIN

  IF NEW.status = 'Treated' AND NEW.date < CURDATE() THEN

    UPDATE Health_Record

    SET status = 'Recovered'

    WHERE health_record_ID = NEW.health_record_ID;

  END IF;

END;
```

//

DELIMITER ;

Result:



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

	health_record_ID	disease	status	date	description	patientID
▶	5001	Flu	Recovered	2023-07-15	Patient had flu symptoms.	1001
	5002	Broken Arm	Healing	2023-07-20	Patient's arm was fractured.	1002
	5003	Headache	Treated	2023-07-25	Patient had a severe headache.	1003
	5004	Fever	Treated	2023-07-16	Patient had high fever.	1004
	5005	Sprained Ankle	Healing	2023-07-21	Patient sprained ankle.	1005
	5006	Common Cold	Treated	2023-07-26	Patient had common cold symptoms.	1006
	5007	Influenza	Treated	2023-07-29	Patient medication changed to painkiller.	1007

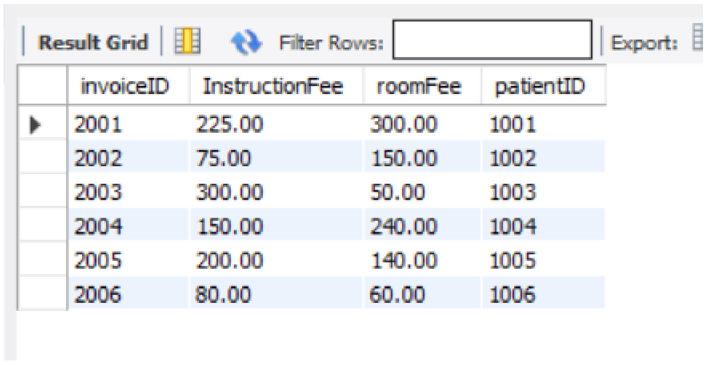
+Trigger 2: trig_calculate_invoice_total

Description: This trigger calculates the total amount for an invoice whenever a new row is inserted into the Invoice table. The trig_calculate_invoice_total trigger automatically calculates the total amount for an invoice by summing the InstructionFee and roomFee whenever a new row is inserted into the Invoice table.

Query:

```
DELIMITER //

CREATE TRIGGER trig_calculate_invoice_total

AFTER INSERT ON Invoice

FOR EACH ROW

BEGIN

  DECLARE total_amount DECIMAL(10, 2);



  SELECT (NEW.InstructionFee + NEW.roomFee) INTO total_amount;
  UPDATE Invoice

  SET total_amount = total_amount

  WHERE invoiceID = NEW.invoiceID;

END;

//

DELIMITER ;
```

Result:



invoiceID	InstructionFee	roomFee	patientID
2001	225.00	300.00	1001
2002	75.00	150.00	1002
2003	300.00	50.00	1003
2004	150.00	240.00	1004
2005	200.00	140.00	1005
2006	80.00	60.00	1006

+Trigger 3: trig_check_room_capacity

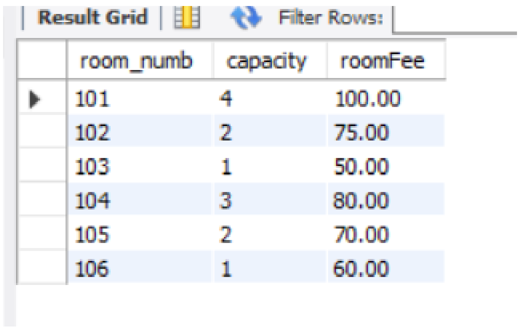
Description: This trigger prevents inserting a new patient into a room if the room has reached its capacity. The trig_check_room_capacity trigger ensures that a new patient is not assigned to a room that has reached its capacity. It prevents overloading a room and helps maintain a balanced distribution of patients in available rooms.

Query:

```
DELIMITER //

CREATE TRIGGER trig_check_room_capacity

BEFORE INSERT ON Patient

FOR EACH ROW
```

```
BEGIN

  DECLARE current_capacity INT;

  SELECT capacity INTO current_capacity

  FROM Room

  WHERE room_numb = NEW.room_numb;

  IF current_capacity >= (SELECT COUNT(*) FROM Patient WHERE room_numb =
NEW.room_numb) THEN

    SIGNAL SQLSTATE '45000'

    SET MESSAGE_TEXT = 'Room is at full capacity';

  END IF;

END;

//

DELIMITER ;
```

Result:



room_numb	capacity	roomFee
101	4	100.00
102	2	75.00
103	1	50.00
104	3	80.00
105	2	70.00
106	1	60.00

6- Queries, descriptions, and results.

Aggregation

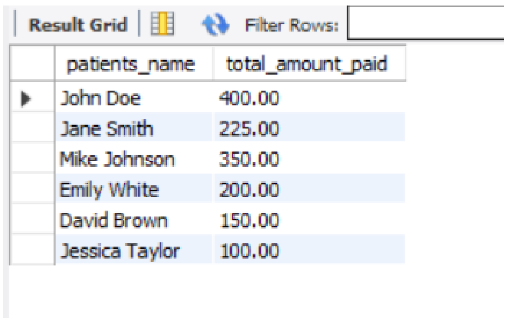
+Aggregation Query 1: Calculate the total amount paid by each patient.

Description: This query calculates the total amount paid by each patient by performing a left join between the Patient and Payment tables and then grouping the results by patientID and patients_name.

Query:

```
SELECT

  p.patients_name,

  SUM(py.amount) AS total_amount_paid

FROM

  Patient p

LEFT JOIN Payment py ON p.patientID = py.patientID

GROUP BY p.patientID, p.patients_name;
```

Result:



patients_name	total_amount_paid
John Doe	400.00
Jane Smith	225.00
Mike Johnson	350.00
Emily White	200.00
David Brown	150.00
Jessica Taylor	100.00

+Aggregation Query 2: Find the average room fee charged for each type of room.

Description: This query calculates the average room fee charged for each type of room based on its capacity by performing an aggregation using the AVG function on the roomFee column and grouping the results by capacity.

Query:

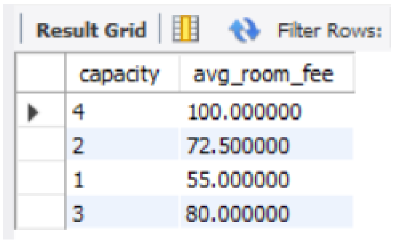
```
SELECT

  r.capacity,

  AVG(r.roomFee) AS avg_room_fee
```

FROM

  Room r

GROUP BY r.capacity;

Result:



capacity	avg_room_fee
4	100.000000
2	72.500000
1	55.000000
3	80.000000

+Aggregation Query 3: Calculate the total number of patients treated in each room.

Description: This query calculates the total number of patients treated in each room by performing left joins between the Room, Patient, and Monitor tables and then counting the distinct patient IDs for each room.

Query

SELECT

  r.room_numb,

  COUNT(DISTINCT m.patientID) AS num_patients_treated

FROM

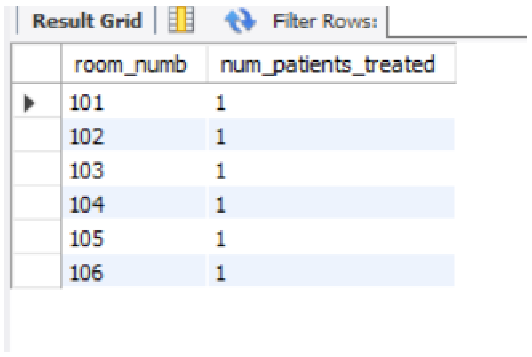
  Room r

LEFT JOIN Patient p ON r.room_numb = p.room_numb

LEFT JOIN Monitor m ON p.patientID = m.patientID

GROUP BY r.room_numb;

Result:



room_numb	num_patients_treated
101	1
102	1
103	1
104	1
105	1
106	1

Nested Queries:

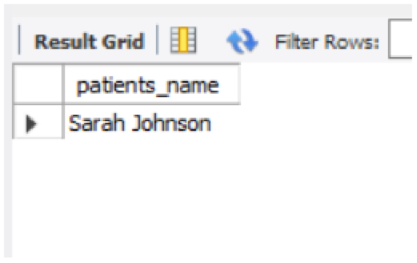
+Nested Query 1: Find the patients who have not made any payments yet.

Description: This nested query finds the names of patients who have not made any payments yet by using a subquery to get the patientIDs from the Payment table and then excluding those patientIDs from the Patient table.

Query:

```
SELECT patients_name

FROM Patient

WHERE patientID NOT IN (

  SELECT patientID

  FROM Payment

);
```

Result:



patients_name
Sarah Johnson

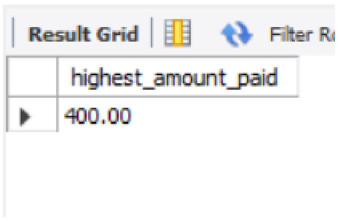
+Nested Query 2: Find the highest total amount paid among patients who have made payments.

Description: This nested query finds the highest total amount paid among patients who have made payments by using a subquery to calculate the total_amount_paid for each patient and then finding the maximum value from the result.

Query:

```
SELECT MAX(total_amount_paid) AS highest_amount_paid

FROM (

  SELECT

    p.patients_name,

    SUM(py.amount) AS total_amount_paid

  FROM

    Patient p

  LEFT JOIN Payment py ON p.patientID = py.patientID

  GROUP BY p.patientID, p.patients_name

) AS subquery;
```

Result:



```
highest_amount_paid
400.00
```

Nested Query 3: Find the patients with more than one health record and their health records.

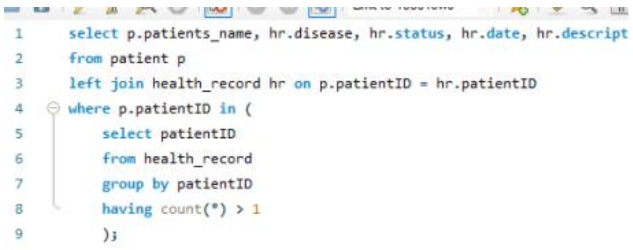
Description: This nested query finds the names of patients with more than one health record and retrieves their health records by using a subquery to get the patientIDs with more than one health record and then including only those patients in the main query.

"for testing purposes we inserted the following row: insert into health_record values(5007, 'flu', 'recovered', curdate(), 'patient has flu.', 1001)"

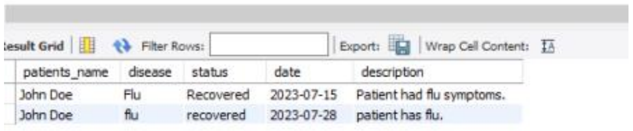
Query:

```
SELECT

  p.patients_name,

  hr.disease,

  hr.status,

  hr.date,

  hr.description

FROM

  Patient p

LEFT JOIN Health_Record hr ON p.patientID = hr.patientID

WHERE p.patientID IN (

  SELECT patientID

  FROM Health_Record

  GROUP BY patientID

  HAVING COUNT(*) > 1

);
```

Result:



```
select p.patients_name, hr.disease, hr.status, hr.date, hr.descript
from patient p
left join health_record hr on p.patientID = hr.patientID
where p.patientID in (
    select patientID
    from health_record
    group by patientID
    having count(*) > 1
    );
```



```
patients_name  disease  status     date        description
John Doe       Flu      Recovered  2023-07-15  Patient had flu symptoms.
John Doe       flu      recovered  2023-07-28  patient has flu.
```

Join Queries:

+Join Query 1: Retrieve patients along with their assigned physicians and nurses.

Description: This query retrieves the names of patients along with their assigned physicians and nurses by performing left joins between the Patient, Monitor, Physician, and Nurse tables.

Query:

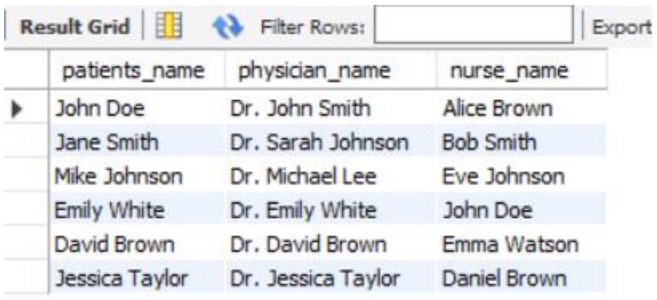
```
SELECT
  p.patients_name,
  ph.physician_name,
  n.nurse_name
FROM
  Patient p
```

LEFT JOIN Monitor m ON p.patientID = m.patientID

LEFT JOIN Physician ph ON m.physicianID = ph.physicianID

LEFT JOIN Nurse n ON p.nurseID = n.nurseID;

Result:



patients_name	physician_name	nurse_name
John Doe	Dr. John Smith	Alice Brown
Jane Smith	Dr. Sarah Johnson	Bob Smith
Mike Johnson	Dr. Michael Lee	Eve Johnson
Emily White	Dr. Emily White	John Doe
David Brown	Dr. David Brown	Emma Watson
Jessica Taylor	Dr. Jessica Taylor	Daniel Brown

+Join Query 2: Retrieve patient names and their corresponding room information.

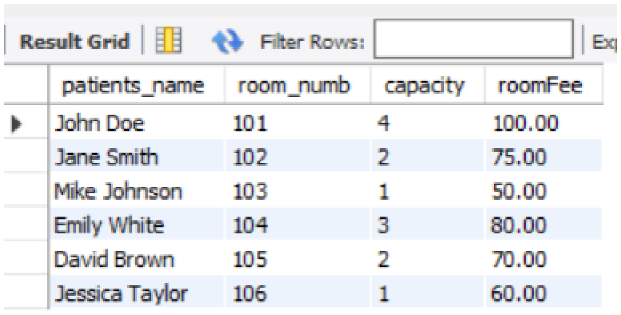
Description: This query retrieves the names of patients along with their corresponding room information (room number, capacity, and room fee) by performing an inner join between the Patient and Room tables.

Query

```
SELECT

  p.patients_name,

  r.room_numb,

  r.capacity,

  r.roomFee

FROM

  Patient p
```

INNER JOIN Room r ON p.room_numb = r.room_numb;

Result:



patients_name	room_numb	capacity	roomFee
John Doe	101	4	100.00
Jane Smith	102	2	75.00
Mike Johnson	103	1	50.00
Emily White	104	3	80.00
David Brown	105	2	70.00
Jessica Taylor	106	1	60.00

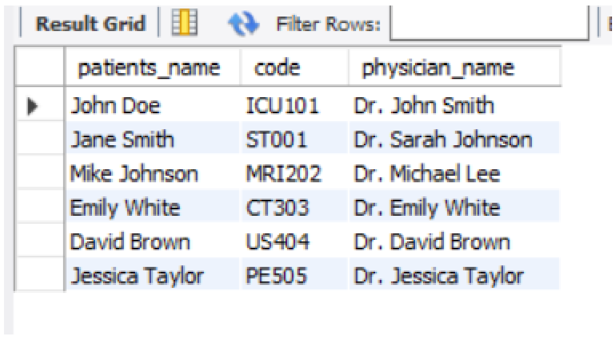
+Join Query 3: Retrieve patient names and the instructions ordered by their assigned physicians.

Description: This query retrieves the names of patients along with the instructions ordered by their assigned physicians by performing left joins between the Patient, Order, and Physician tables.

Query:

```
SELECT

  p.patients_name,

  o.code,

  ph.physician_name

FROM

  Patient p

LEFT JOIN Ordered o ON p.patientID = o.patientID

LEFT JOIN Physician ph ON o.physicianID = ph.physicianID;
```

Result:



patients_name	code	physician_name
John Doe	ICU101	Dr. John Smith
Jane Smith	ST001	Dr. Sarah Johnson
Mike Johnson	MRI202	Dr. Michael Lee
Emily White	CT303	Dr. Emily White
David Brown	US404	Dr. David Brown
Jessica Taylor	PE505	Dr. Jessica Taylor

7- Transactions and description

+Transaction 1: trans_create_patient_with_invoice

Description: This transaction creates a new patient record and generates an invoice for the patient with the specified details. The trans_create_patient_with_invoice transaction ensures that creating a new patient record, health record, and invoice are executed as a single unit of work. If any part of the transaction fails, all changes are rolled back, maintaining data integrity.

Query:

```
START TRANSACTION;

INSERT INTO Patient (patients_name, phone_number, address, nurseID, type_of_med,
amount_of_med, room_numb, night_stay)

VALUES ('John Smith', '777-777-7777', '789 Park Rd', 101, 'Painkiller', 3, 101, 2);



SELECT @new_patientID := LAST_INSERT_ID();



INSERT INTO Health_Record (disease, status, date, description, patientID)

VALUES ('Headache', 'Treated', '2023-07-30', 'Patient had a headache.',
@new_patientID);
```

```
INSERT INTO Invoice (InstructionFee, roomFee, patientID)

VALUES (250.00, 200.00, @new_patientID);

COMMIT;
```

Result:

+Transaction 2: Add New Patient and Related Health Record

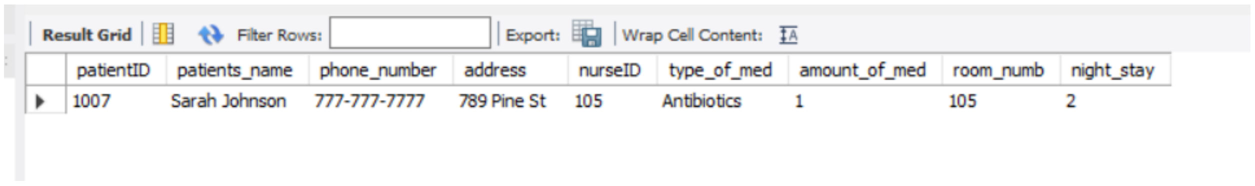
Description: This transaction adds a new patient to the Patient table and simultaneously inserts a new health record for that patient in the Health_Record table.

Query:

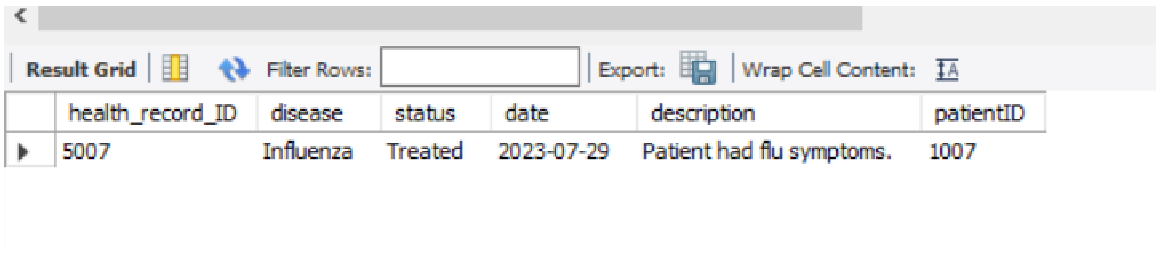
```
START TRANSACTION;

-- Step 1: Insert new patient

INSERT INTO Patient (patientID, patients_name, phone_number, address, nurseID,
type_of_med, amount_of_med, room_numb, night_stay)

VALUES (1007, 'Sarah Johnson', '777-777-7777', '789 Pine St', 105, 'Antibiotics', 1,
105, 2);

-- Step 2: Insert health record for the new patient

INSERT INTO Health_Record (health_record_ID, disease, status, date, description,
patientID)

VALUES (5007, 'Influenza', 'Treated', '2023-07-29', 'Patient had flu symptoms.', 1007);
```

COMMIT;

Result:



	patientID	patients_name	phone_number	address	nurseID	type_of_med	amount_of_med	room_numb	night_stay
▶	1007	Sarah Johnson	777-777-7777	789 Pine St	105	Antibiotics	1	105	2



	health_record_ID	disease	status	date	description	patientID
▶	5007	Influenza	Treated	2023-07-29	Patient had flu symptoms.	1007

+Transaction 3: Assign New Physician to a Patient

Description: This transaction assigns a new physician to an existing patient in the Monitor table.

Query:

START TRANSACTION;

-- Step 1: Check if the physician exists

SELECT COUNT(*) INTO @physician_exists

FROM Physician

WHERE physicianID = 7;

-- Step 2: If the physician exists, assign to the patient

IF @physician_exists > 0 THEN

```
  INSERT INTO Monitor (patientID, physicianID, duration)

  VALUES (1007, 7, '3 days');

  SELECT 'Physician assigned successfully.' AS message;

ELSE

  SELECT 'Physician does not exist. Please check physicianID.' AS message;

END IF;



COMMIT;
```

Result:

Transaction 4: Update Patient's Medication

Description: This transaction updates the type of medication for a patient in the Patient table and updates the corresponding Health_Record for the patient. In these transactions, we use the START TRANSACTION statement to begin a new transaction and COMMIT to commit the changes if all the steps are executed successfully. If any error occurs during the transaction, the ROLLBACK statement can be used to undo all the changes made within the transaction.

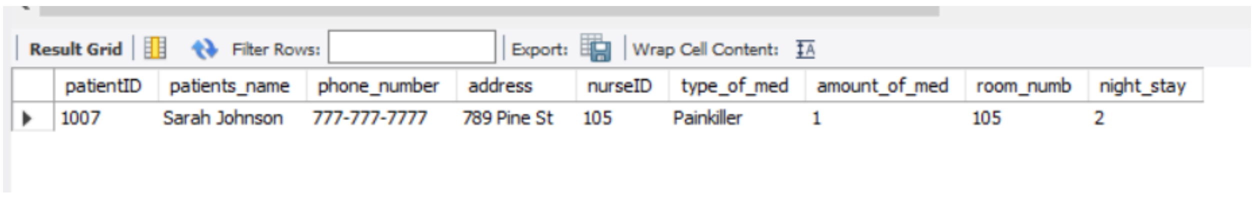
Query:

```
START TRANSACTION;

-- Step 1: Update patient's medication

UPDATE Patient

SET type_of_med = 'Painkiller'

WHERE patientID = 1007;

-- Step 2: Update health record description for the medication change

UPDATE Health_Record

SET description = 'Patient medication changed to painkiller.'
```

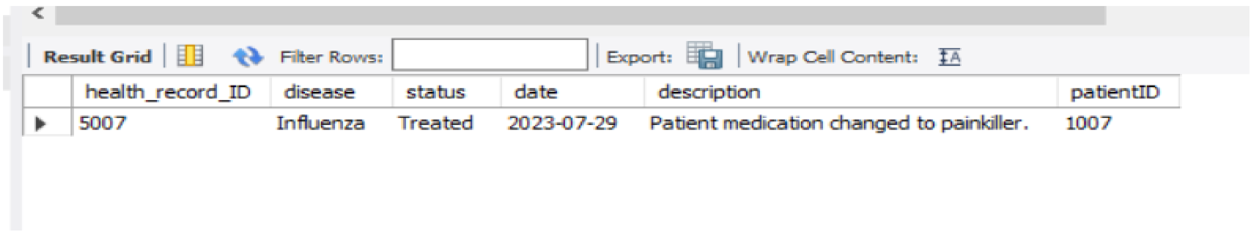
WHERE patientID = 1007 AND disease = 'Influenza';

COMMIT;

Result:



patientID	patients_name	phone_number	address	nurseID	type_of_med	amount_of_med	room_numb	night_stay
1007	Sarah Johnson	777-777-7777	789 Pine St	105	Painkiller	1	105	2



health_record_ID	disease	status	date	description	patientID
5007	Influenza	Treated	2023-07-29	Patient medication changed to painkiller.	1007

+Transaction 5: This transaction will transfer a patient to a different room

Query:

START TRANSACTION;

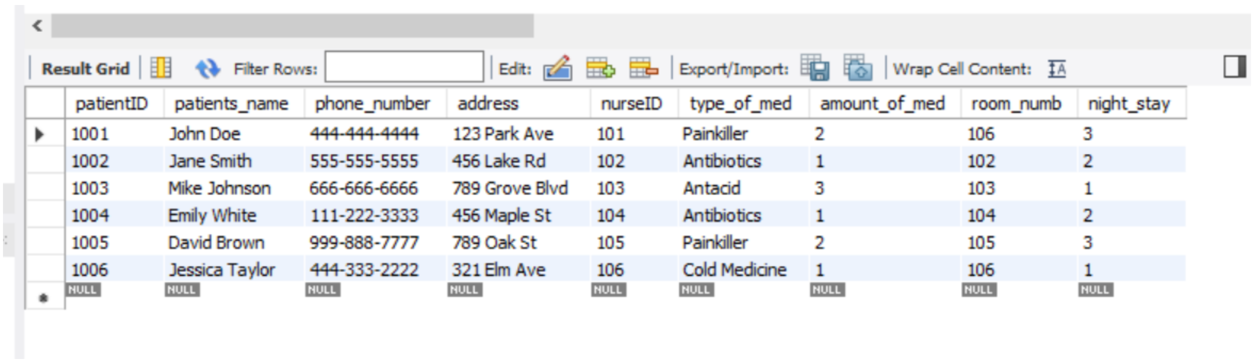
UPDATE Patient

SET room_numb = 106, night_stay = 3

WHERE patientID = 1001;

COMMIT;

Result:



patientID	patients_name	phone_number	address	nurseID	type_of_med	amount_of_med	room_numb	night_stay
1001	John Doe	444-444-4444	123 Park Ave	101	Painkiller	2	106	3
1002	Jane Smith	555-555-5555	456 Lake Rd	102	Antibiotics	1	102	2
1003	Mike Johnson	666-666-6666	789 Grove Blvd	103	Antacid	3	103	1
1004	Emily White	111-222-3333	456 Maple St	104	Antibiotics	1	104	2
1005	David Brown	999-888-7777	789 Oak St	105	Painkiller	2	105	3
1006	Jessica Taylor	444-333-2222	321 Elm Ave	106	Cold Medicine	1	106	1
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

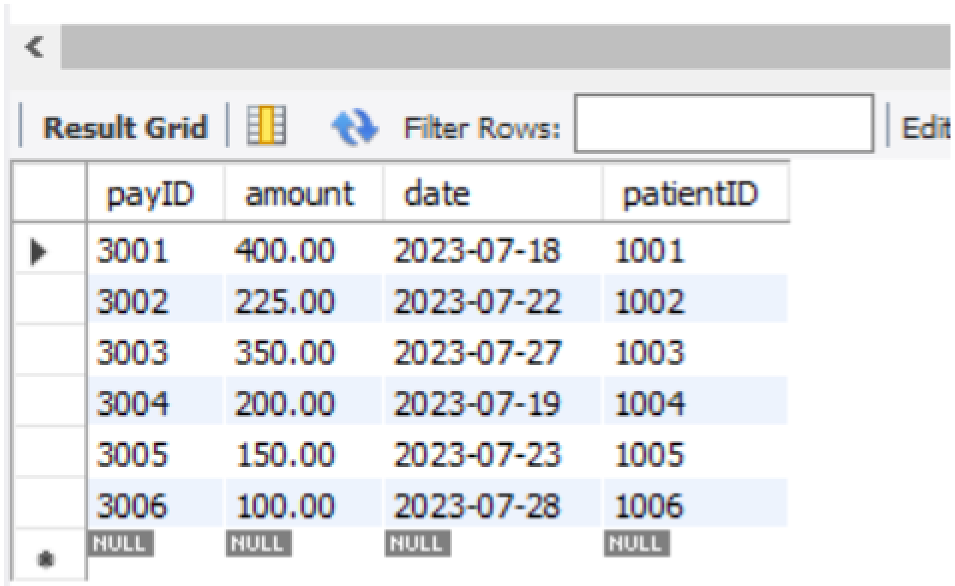
+Transaction 6: This transaction query will update the Payment table to reflect a partial payment made by a patient.

Query:

```
START TRANSACTION;

-- Update the payment amount for patient with patientID 1001

UPDATE Payment

SET amount = amount - 200.00

WHERE patientID = 1001;


-- Insert a new record in the Payment table for the partial payment

INSERT INTO Payment (payID, amount, date, patientID)

VALUES (3007, -200.00, '2023-07-20', 1001);


--Update the latest payment

UPDATE Issue_Pay

SET payID = 3007

WHERE patientID = 1001;


COMMIT;
```

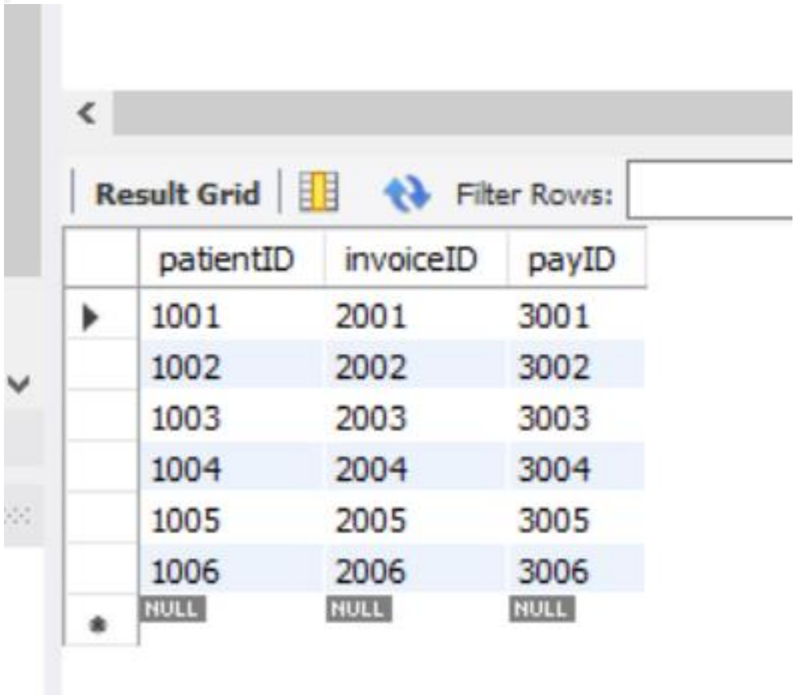
Result:

Before:



Result Grid | Filter Rows: | Edi

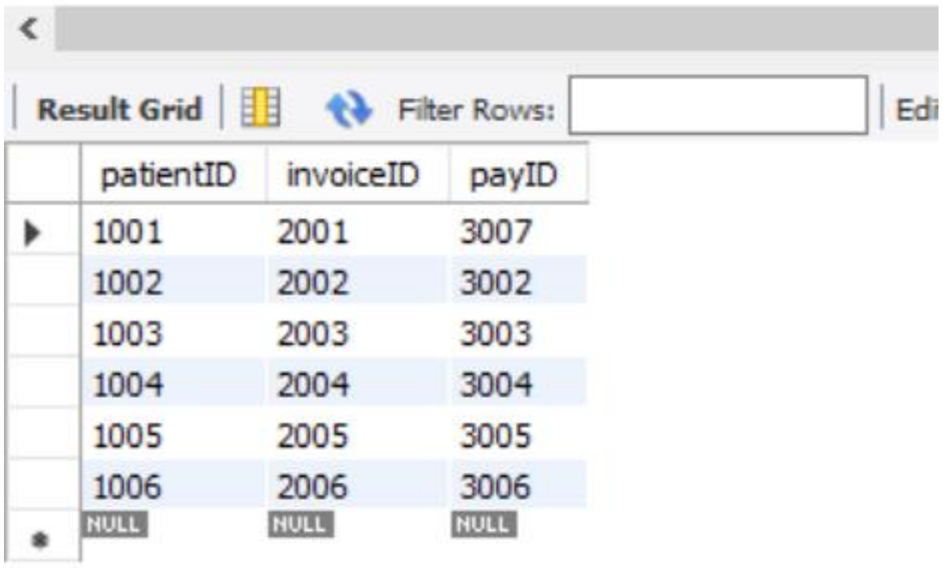
payID	amount	date	patientID
3001	400.00	2023-07-18	1001
3002	225.00	2023-07-22	1002
3003	350.00	2023-07-27	1003
3004	200.00	2023-07-19	1004
3005	150.00	2023-07-23	1005
3006	100.00	2023-07-28	1006
NULL	NULL	NULL	NULL



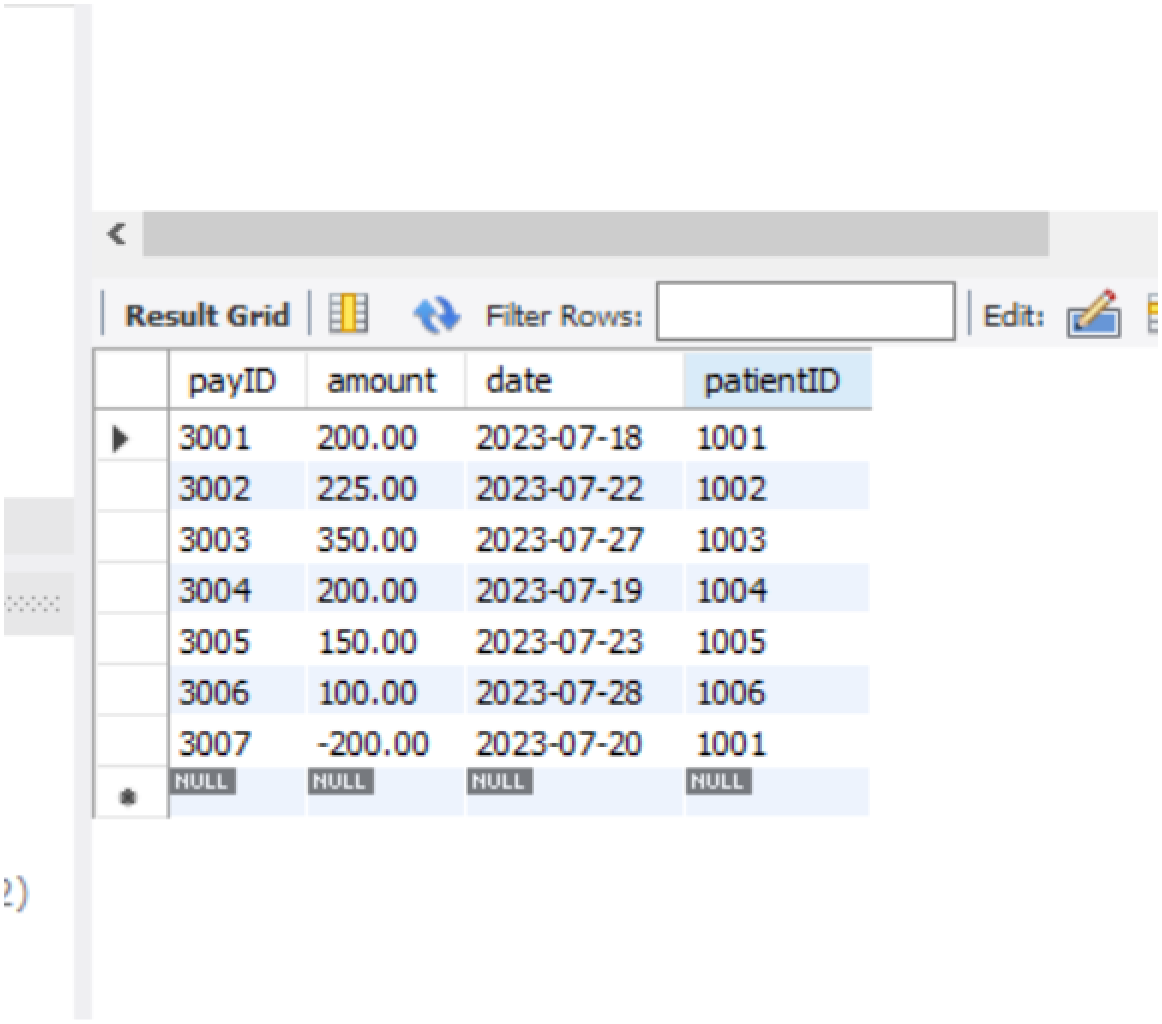
Result Grid | Filter Rows:

patientID	invoiceID	payID
1001	2001	3001
1002	2002	3002
1003	2003	3003
1004	2004	3004
1005	2005	3005
1006	2006	3006
NULL	NULL	NULL

After:



patientID	invoiceID	payID
1001	2001	3007
1002	2002	3002
1003	2003	3003
1004	2004	3004
1005	2005	3005
1006	2006	3006
NULL	NULL	NULL



Result Grid Filter Rows: Edit:

payID	amount	date	patientID
3001	200.00	2023-07-18	1001
3002	225.00	2023-07-22	1002
3003	350.00	2023-07-27	1003
3004	200.00	2023-07-19	1004
3005	150.00	2023-07-23	1005
3006	100.00	2023-07-28	1006
3007	-200.00	2023-07-20	1001
NULL	NULL	NULL	NULL