University
Mohammed VI
Polytechnic

College of Computing

Deliverable #: Title of Lab/Project here (e.g.: Conceptual Design ....)

Data Management Course

UM6P College of Computing

Professor: Karima Echihabi **Program:** Computer Engineering

Session: Fall 2025

Team Information

Team Name	.....
Member 1	Adam Chikri
Member 2	Amine Abdellaoui
Member 3	Douae Bouayadi
Member 4	Aya Abid
Member 5	Nissrine Benallal
Member 6	Yasmin Bouaboud
Repository Link	`https://github.com/Adam-chi/Lab5-DBMS`

---

1 Introduction

This lab focused on designing and implementing a hospital management database. Tasks included creating and normalizing tables, enforcing data integrity with keys and constraints, inserting and manipulating sample data, and writing SQL queries to extract meaningful information. The work provided hands-on experience in building a structured, reliable, and database system that we can run queries on.

2 Requirements

The objective of this project part was to work through several tasks related to refining and operating the MNHS database .We first hade to validate each relation against BCNF and verify lossless joins and dependency preservation after decomposition.After that ,we implemented the corrected shema using SQL DDL,populated it with sample data using DML operations .Finally , we executed the 20 queries on the MNHS schema created .

3 Methodology

This part will provide the logic when working with BCNF: Validating the BCNF, and Testing the lossless joins, and finally checking if it's dependency preserving.

A. BCNF Validation

For each relation (Patient, ContactLocation, Staff, Hospital, Department, Insurance, ClinicalActivity, Expense, Appointment, Emergency, Medication, Stock, Prescription, Includes, have, Patient–Insurance, Work_in). The first step is listing the FDs then Identifying keys and by the end verifying superkey condition.

All relations were found to be already in BCNF because all FDs had a superkey on the left side.

B. Lossless Join Verification

To test lossless join, we used the standard condition:

For a decomposition $R \to R_1$ and R_2, the join is lossless if:

$$R_1 \cap R_2 \to R_1 \quad \text{or} \quad R_1 \cap R_2 \to R_2$$

In other words, the common attributes between the decomposed relations must form a key for at least one of them.

For each relation in our schema that required decomposition, we documented the intersection and verified the lossless condition.

C. Dependency Preservation Verification

After decomposition, we checked whether all functional dependencies of the original relation were still preserved.

Compute the union of FDs in decomposed relations

First step: Compute the closure:

$$(F_1' \cup F_2' \cup \cdots \cup F_n')^+$$

Second step: Compare with the closure of the original FDs.

If we have:

$$(F_1' \cup F_2' \cup \cdots \cup F_n')^+ = F^+$$

then the dependency preservation is satisfied.

Using closure-based reasoning, we confirmed: Patient: all FDs preserved in R_1 and R_2

Conclusion: All decompositions preserve functional dependencies.

This methodology part will be dedicated to explain our logic behind the queries.

Query 1:

we order patients by Fullname (there is no atrribute called last name) using ORDER BY FullName

Query 2:

This query lists all distinct insurance types available in the system. By using SELECT DISTINCT on the Insurance table, it removes duplicates and provides a clear overview of coverage options offered to patients, useful for administrative reference and patient registration.

Query 3:

This query retrieves the ID's and full names of staff members who work in hospitals located in Rabat. To do that, we join the Staff, work_in, Department, and Hospital tables to connect staff with hospitals through their department assignments, we then filter the results to only include hospitals where the city is Rabat. DISTINCT is used to avoid duplicate entries if staff work in multiple departments or hospitals in Rabat.

Query 4:

To get the appointments scheduled for the next 7 days, we'd need to join Appointments with Clinical Activity since it's the table that has the dates and then retrieving the rows where the date is between today and after 7 days.

Query 5:

The Appointment and ClinicalActivity tables were joined to link each appointment to its corresponding department. The results were then grouped by department, and the number of appointments in each group was calculated to analyze their distribution

Query 6:

We calculate the average unit price using AVG(UnitPrice) from the table Stock

Query 7:

This query identifies hospitals with high emergency admission rates. By joining the Hospital, Department, ClinicalActivity, and Emergency tables, it connects each hospital to its departments and emergency clinical activities. The filter on 'Admitted' ensures only serious cases are counted, and the HAVING clause isolates facilities with more than 20 admissions, highlighting those that might need additional resources.

Query 8:

To get the unit price of a medication we'll need to join the medication table with the stock, and then comparing the rows to see where the unit price is below 200 and in the 'Antibiotic' Therapeutic Class

Query 9:

Keep medication if there are <3 medications in the same hospital that cost more.

Query 10:

This query returns the number of scheduled, completed, and cancelled appointments for each department, all in one result table. Here's how it works: DEP_ID and department name are selected from Department, then three separate subqueries are used to count scheduled, cancelled and completed appointments. Each one of the three joins ClinicalActivity and Appointment, filtering by department and status. After that counts are arranged as new columns for each department in the output.

Query 11:

To identify patients with no scheduled appointments in the next 30 days, a subquery was used to find patients with upcoming scheduled appointments, and then only those patients whose IDs were not in this list were selected. This highlighted patients who are not scheduled for any consultations in the near future

Query 12:

This query calculates for each staff member the total number of appointments they are responsible for, and their percentage share of all appointments in their hospital. It joins Staff, ClinicalActivity, Appointment, and Department to relate each staff member to their department and hospital. It then counts appointments per staff (COUNT(A.CAID)), the percentage is calculated as (staff member's appointment count) divided by (total appointments in their hospital), times 100. It uses a subquery in the percentage calculation to get hospital totals. The results are grouped by staff to display statistics for each individual.

Query 13:

To monitor hospital stock and identify medications below the reorder level, the Stock table was queried to find entries where the current quantity was less than the reorder threshold. Joining with the Hospital and Medication tables allowed for more descriptive output, showing which medications in which hospitals needed attention

Query 14:

We used the basic division pattern seen in the lecture by changing the table names to get the appropriate result, a logical way to see this, is if for a hospital the table of the total antibiotic medications that exist minus the antibiotics that the hospital has is empty, then the hospital has every single antibiotic in the catalog.

Query 15:

This query analyzes medication pricing strategies between hospitals in the same city. It first calculates the average price per hospital and therapeutic class through joins between Hospital, Stock, and Medication. Correlated subqueries then compute average prices by city and drug class. Finally, UNION combines two sets: hospitals with prices above the local average and those with prices below or equal to average, providing a complete picture of pricing positioning.

Query 16:

This query determines the next appointment date for each patient. By linking the Patient, ClinicalActivity, and Appointment tables, it identifies all future scheduled appointments. The MIN function finds the nearest date for each patient, and HAVING ensures only patients with confirmed appointments appear in the results.

Query 17:

Counts(*) counts every row in the grouped result, giving the total_emergency_visist for each patient ,then we check if it is >=2 to select patients with at least two emergency visits.MAX(C.Date)finds the most recent date of all emergency visits,which is then compared using DATEDIFF to ensure it is within 14 days of today(using CURDATE())

Query 18:

We need to join the Hospital, Department, Clinical Activity, and Appointment tables to retrieve all the necessary data. Then, we count the number of appointments and rank them by city (so that the results are grouped by city) and by appointment count in descending order.

Query 19:

This query finds medications in each city where the highest and lowest hospital unit prices differ by more than 30%, showing: the city, medication name, minimum/maximum price, and percentage spread between prices. Here's how it works, it joins Hospital, Stock, and Medication to relate city, hospital stock, and medications. For each city and medication, it calculates min/max prices using aggregation. It then Computes price spread percentage: (max-min)/min times 100. After that, it filters results with HAVING to include only those where the spread is above 30%.

Query 20:

For the data quality check on stock entries, all stock rows with negative quantities or non-positive unit prices were selected. This allowed for detecting potential errors in the stock database that could impact inventory management or reporting

4 Implementation & Results

Validating BCNF, Lossless joins, Dependency preservation:

Normalization

• Validate each relation against BCNF.

1. **Patient**
   FDs:
   - IID → Name,Sex,Birth,BloodGroup,Phone,CIN
   - CIN → IID,Name,Sex,Birth,BloodGroup,Phone (CIN is a candidate key)

   Both IID and CIN are candidate keys. (superkeys)
   Conclusion: BCNF is satisfied

2. **ContactLocation**
   FD: CLID → Street,City,Province,PostalCode,Phone
   • CLID is the primary key(superkey)
   Conclusion: BCNF is satisfied

3. **Staff (with subtypes ISA)**
   FDs:
   - STAFF ID → Name,Status
   - STAFF ID → LicenseNumber,Specialty (Practitioner)
   - STAFF ID → Grade,Ward (Caregiving)
   - STAFF ID → Modality,Certifications (Technical)

   STAFF ID is the key in all cases.
   All Attributes are functionally dependent on STAFF ID.
   Conclusion: BCNF is satisfied

4. **Hospital**
   FD: assume HID → Name,City,Region
   All Attributes are functionally dependent on HID
   Conclusion: BCNF is satisfied

5. **Department**
   FDs:
   - DEPT ID → Name,Specialty,HID

   Conclusion: BCNF is satisfied

6. **Insurance**
   FD: InsID → Type
   Conclusion: BCNF is satisfied

7. **ClinicalActivity**
   FD:
   - CAID → Title,Time,Date,IID,STAFF ID,DEPT ID,ExpID

   Conclusion: BCNF is satisfied.

8. **Expense**
   FDs:
   - ExpID → Total,InsID,CAID
   - CAID ↔ ExpID
   - Primary key: ExpID
   - Transitive: ExpID → CAID → (Patient, Staff, Department, Hospital)

   Conclusion: BCNF is satisfied.

9. **Prescription**
   FDs:
   - PID → DateIssued

   Conclusion: BCNF is satisfied.

10. **Appointment (ISA of ClinicalActivity)**
    FD: CAID → Reason,Status
    Conclusion: BCNF is satisfied.

11. **Emergency (ISA of ClinicalActivity)**
    FD: CAID → TriageLevel,Outcome
    Conclusion: BCNF is satisfied.

12. **Medication**
    FD: DrugID → Name,Form,Strength,Class,ActiveIngredient,Manufacturer
    Conclusion: BCNF is satisfied.

13. **Stock (Hospital–Medication)**
    FD: {HID,DrugID,StockTimestamp} → UnitPrice,StockRemaining,Qty,ReorderLevel
    • Key: composite {HID,DrugID,StockTimestamp}
    Conclusion: BCNF is satisfied.

14. **Prescription–Medication (M:N)**
    FD: {PID,DrugID} → Dosage,Duration
    • Composite key {PID,DrugID}
    Conclusion: BCNF is satisfied.

15. **Patient–ContactLocation**
    FD: {IID,CLID} → relationship attributes
    • Composite key {IID,CLID}
    Conclusion: BCNF is satisfied.

16. **Patient–Insurance**
    FDs: {IID,InsID} → relationship attributes
    Conclusion: BCNF is satisfied.

17. **Staff–Department (WorkIn)**
    FD: {STAFF ID,DEPT ID} → (relationship attributes)
    Conclusion: BCNF is satisfied.

• Verify lossless joins and dependency preservation after decomposition.

Patient

FDs:
IID → Name,Sex,Birth,BloodGroup,Phone,CIN
CIN → IID
Decomposition:
R_1(IID,Name,Sex,Birth,BloodGroup,Phone) and R_2(IID,CIN)

• Intersection: IID
• IID → R1
Conclusion: Lossless join is satisfied.

For R1:
FD1:
IID → Name,Sex,Birth,BloodGroup,Phone

For R1:
FD1:
CIN → Name,Sex,Birth,BloodGroup,Phone

(FD1 Union FD2)$^+$ = FDs$^+$
Conclusion: Dependency preserving is satisfied.

NOTE : for the other relations there is no decomposition needed because already their Functional dependencies contain only primarykey→All attributes

We provided alongside this pdf a sql file in the Github Repository named **'Lab5.sql'** that is used to create the database schema.

Inserting 5 samples into each table and then manipulating the data:

```
USE lab3;

INSERT INTO Patient VALUES
(1, 'CIN1001', 'Sara El Amrani', '1995-06-21', 'F', 'A+', '0612345678'),
(2, 'CIN1002', 'Omar Bennani', '1988-09-15', 'M', 'O-', '0623456789'),
```

```
(3, 'CIN1003', 'Youssef Khadir', '2000-01-10', 'M', 'B+', '0634567890'),
(4, 'CIN1004', 'Lina Berrada', '1993-11-05', 'F', 'AB-', '0645678901'),
(5, 'CIN1005', 'Amine Fassi', '1998-04-20', 'M', 'A-', '0656789012');

INSERT INTO Hospital VALUES
(1, 'CHU Rabat', 'Rabat', 'Rabat-Sale'),
(2, 'CHU Casablanca', 'Casablanca', 'Casablanca-Settat'),
(3, 'CHU Marrakech', 'Marrakech', 'Marrakech-Safi'),
(4, 'CHU Fes', 'Fes', 'Fes-Meknes'),
(5, 'CHU Agadir', 'Agadir', 'Souss-Massa');

INSERT INTO Department VALUES
(1, 1, 'Cardiology', 'Heart Diseases'),
(2, 1, 'Neurology', 'Brain and Nerves'),
(3, 2, 'Pediatrics', 'Child Health'),
(4, 3, 'Surgery', 'General Surgery'),
(5, 4, 'Radiology', 'Imaging');

INSERT INTO Staff VALUES
(1, 'Dr. Hicham El Idrissi', 'Active'),
(2, 'Dr. Salma Tazi', 'Active'),
(3, 'Nurse Rania Lahlou', 'Active'),
(4, 'Technician Yassine Kabbaj', 'Retired'),
(5, 'Dr. Mehdi Rahali', 'Active');

INSERT INTO Work_in VALUES
(1, 1),
(2, 2),
(3, 3),
(4, 4),
(4, 5),
(5, 5);

INSERT INTO ClinicalActivity VALUES
(1, 1, 1, 1, '2025-10-10', '10:00:00'),
(2, 2, 2, 2, '2025-10-30', '11:45:00'),
(3, 3, 3, 3, '2025-09-05', '15:00:00'),
(4, 4, 4, 4, '2025-09-22', '12:15:00'),
(5, 5, 4, 5, '2026-01-30', '11:45:00'),
(6, 1, 1, 1,'2025-11-11', '11:45:00'),
(7, 2, 2, 2, '2025-01-11', '09:30:00'),
(8, 5, 5, 5, '2025-11-20', '09:30:00');

INSERT INTO Appointment VALUES
(1, 'Routine Checkup', 'Completed'),
(2, 'Neurology Consultation', 'Completed'),
(3, 'Pediatric Visit', 'Cancelled'),
(4, 'Surgery Follow-up', 'Completed'),
(5, 'X-ray Review', 'Scheduled'),
(6, 'Routine Checkup', 'Completed'),
(7, 'Neurology Consultation', 'Completed'),
```

```
(8, 'Monitoring treatment progress', 'Scheduled');

INSERT INTO Emergency VALUES
(1, 3, 'Discharged'),
(2, 4, 'Admitted'),
(3, 2, 'Transferred'),
(4, 5, 'Deceased'),
(5, 1, 'Admitted');

INSERT INTO Insurance VALUES
(1, 'CNOPS'),
(2, 'CNSS'),
(3, 'RAMED'),
(4, 'Private'),
(5, 'None');

INSERT INTO Expense VALUES
(1, 1, 1, 500.00),
(2, 2, 2, 800.00),
(3, 3, 3, 200.00),
(4, 4, 4, 1200.00),
(5, 5, 5, 50.00);

INSERT INTO Medication VALUES
(1, 'Paracetamol', 'Tablet', '500mg', 'Paracetamol', 'Analgesic', 'Sanofi'),
(2, 'Amoxicillin', 'Capsule', '250mg', 'Amoxicillin', 'Antibiotic', 'Pfizer'),
(3, 'Aspirin', 'Tablet', '100mg', 'Acetylsalicylic Acid', 'Analgesic', 'Bayer'),
(4, 'Cough Syrup', 'Liquid', '5mg/10ml', 'Dextromethorphan', 'Antitussive',
    'GSK'),
(5, 'Insulin', 'Injection', '10IU/ml', 'Insulin', 'Antidiabetic', 'Novo
    Nordisk');

INSERT INTO Stock VALUES
(1, 1, NOW(), 10.00, 100, 10),
(2, 2, NOW(), 25.00, 50, 5),
(2, 3, NOW(), 7.00, 200, 20),
(3, 4, NOW(), 10.00, 200, 20),
(3, 3, NOW(), 5.00, 200, 20),
(4, 4, NOW(), 15.00, 30, 10),
(5, 5, NOW(), 50.00, 10, 5);

INSERT INTO Prescription VALUES
(1, 1, '2025-10-10'),
(2, 2, '2026-01-11'),
(3, 3, '2025-09-05'),
(4, 4, '2025-09-22'),
(5, 5, '2026-01-30');

INSERT INTO Includes VALUES
(1, 1, '1 tablet every 8h', '5 days'),
(2, 2, '1 capsule every 12h', '7 days'),
```

```
(3, 3, '1 tablet daily', '10 days'),
(4, 4, '10ml twice daily', '5 days'),
(5, 5, 'Inject before meals', '30 days');

INSERT INTO ContactLocation VALUES
(1, 'Rabat', 'Rabat-Sale', 'Avenue Hassan II', '12', '10000', '0537001122'),
(2, 'Casablanca', 'Casablanca-Settat', 'Bd Zerktouni', '22', '20000',
    '0522003344'),
(3, 'Marrakech', 'Marrakech-Safi', 'Rue Mohammed V', '5', '40000',
    '0524005566'),
(4, 'Fes', 'Fes-Meknes', 'Rue Hassan', '8', '30000', '0535007788'),
(5, 'Agadir', 'Souss-Massa', 'Bd 20 Aout', '3', '80000', '0528009900');

INSERT INTO have VALUES
(1, 1),
(2, 2),
(3, 3),
(4, 4),
(5, 5);
```

Now to alter the tables:

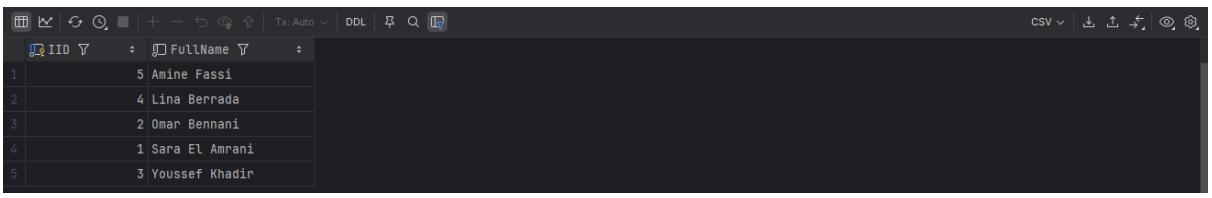
```
ALTER TABLE Patient ADD Email VARCHAR(100);
UPDATE Patient SET Phone = '0611223344' WHERE IID = 3;
UPDATE Hospital SET Region = 'Casablanca-Settat' WHERE HID = 2;

DELETE FROM Appointment WHERE Status = 'Cancelled';
```

We will now provide screenshots of the results given after running the queries on the database schema created using the **"Lab5.sql"** file provided alongside this pdf in the GitHub repository after inserting the samples above and manipulation the data.

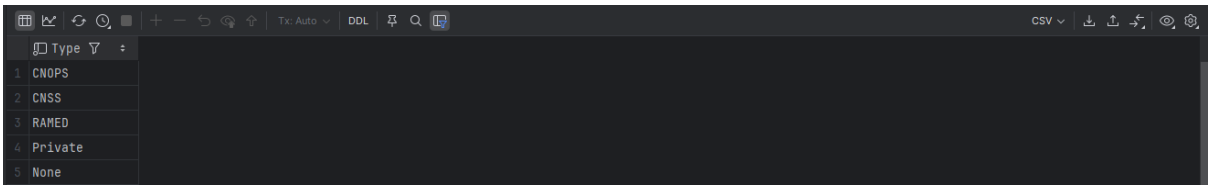
Query 1:

```
SELECT City, COUNT(*) AS total
FROM Hospital
WHERE City = 'Casablanca'
GROUP BY City;
```



Query 2:

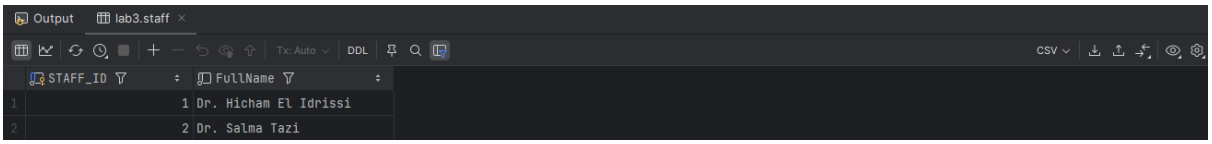
```sql
SELECT DISTINCT Ins.Type
FROM Insurance Ins;
```



	Type
1	CNOPS
2	CNSS
3	RAMED
4	Private
5	None

Query 3:

```sql
SELECT DISTINCT S.STAFF_ID , S.FullName
FROM Staff S
JOIN Work_in W ON S.STAFF_ID = W.STAFF_ID
JOIN Department D ON W.DEP_ID = D.DEP_ID
JOIN Hospital H ON D.HID = H.HID
WHERE H.City='Rabat';
```

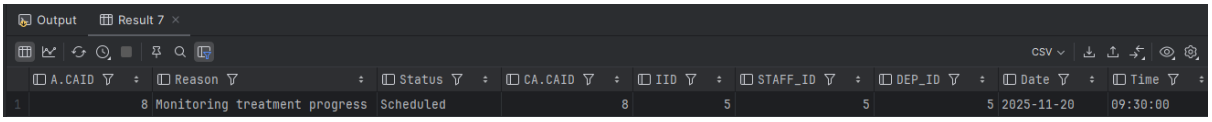


Output | lab3.staff

	STAFF_ID	FullName
1	1	Dr. Hicham El Idrissi
2	2	Dr. Salma Tazi

Query 4:

```sql
SELECT A.CAID
    FROM Appointment A
    JOIN ClinicalActivity CA ON CA.CAID = A.CAID
WHERE CURDATE() <= CA.Date AND DATEDIFF(CA.Date, CURDATE()) <= 7 AND A.Status =
    'Scheduled';
```



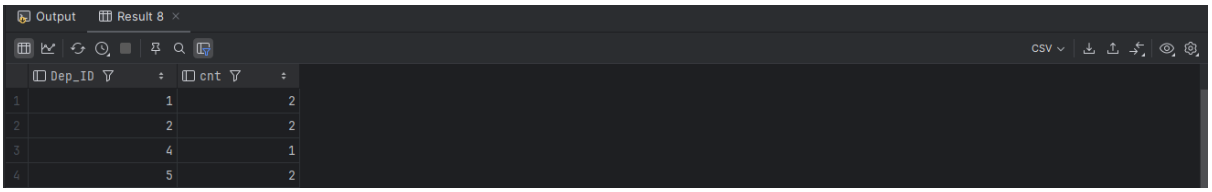
Output | Result 7

	A.CAID	Reason	Status	CA.CAID	IID	STAFF_ID	DEP_ID	Date	Time
1	8	Monitoring treatment progress	Scheduled	8	5	5	5	2025-11-20	09:30:00

Query 5:

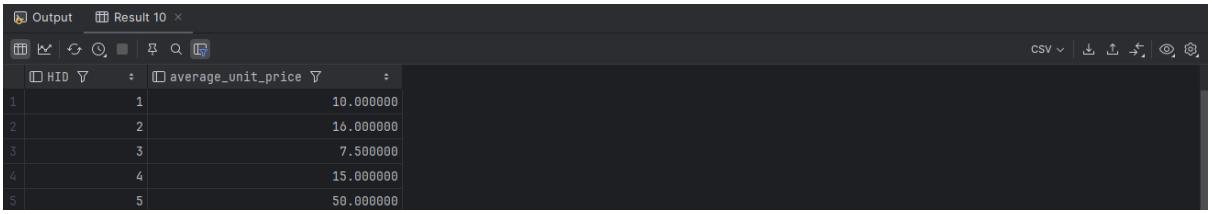
```sql
SELECT CA.Dep_ID, COUNT(*) AS cnt
    FROM Appointment A
    JOIN ClinicalActivity CA on A.CAID=CA.CAID
```

```
4   GROUP BY CA.Dep_ID
```



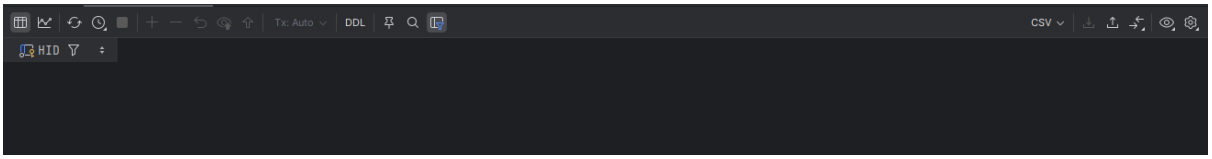
Query 6:

```
1 SELECT S.HID,AVG(UnitPrice) as average_unit_price
2 FROM Stock S
3 GROUP BY S.HID;
```



Query 7:

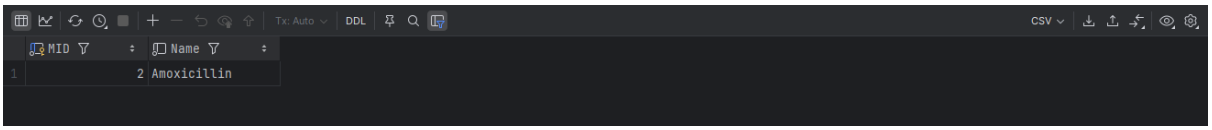
```
1 SELECT HID
2 FROM Hospital H
3 JOIN Department D ON H.HID = D.HID
4 JOIN ClinicalActivity CA ON D.DEP_ID = CA.DEP_ID
5 JOIN Emergency E ON CA.CAID = E.CAID
6 WHERE E.Outcome = 'Admitted'
7 GROUP BY H.HID
8 HAVING COUNT(E.CAID) > 20;
```



The result gave an empty table, since we only implemented around 5 samples per table, so no hospital exceed 20 emergency admissions.

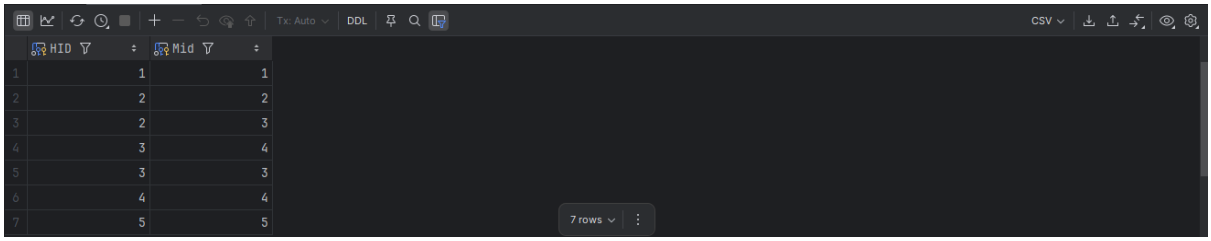
Query 8:

```sql
SELECT M.MID, M.Name FROM Medication M
    JOIN Stock S ON M.MID = S.MID
WHERE M.TherapeuticClass = 'Antibiotic' AND S.UnitPrice < 200;
```



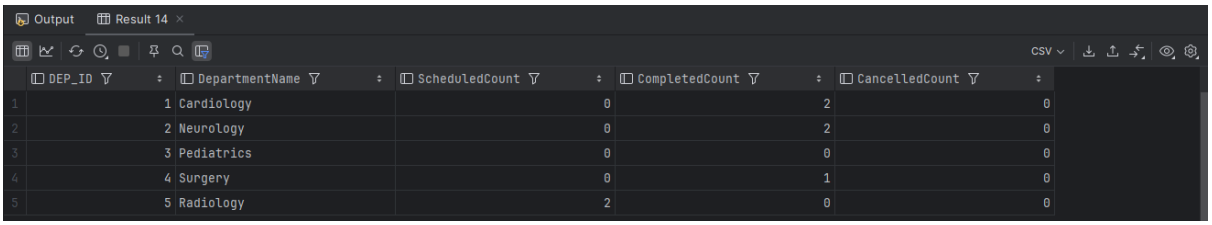
Query 9:

```sql
SELECT S1.HID,S1.Mid
FROM Stock S1
    WHERE (SELECT count(*)
    FROM Stock S2
    WHERE S1.HID = S2.HID
    AND S2.UnitPrice>S1.UnitPrice)<3
ORDER BY S1.HID ,S1.UnitPrice DESC;
```



Query 10:

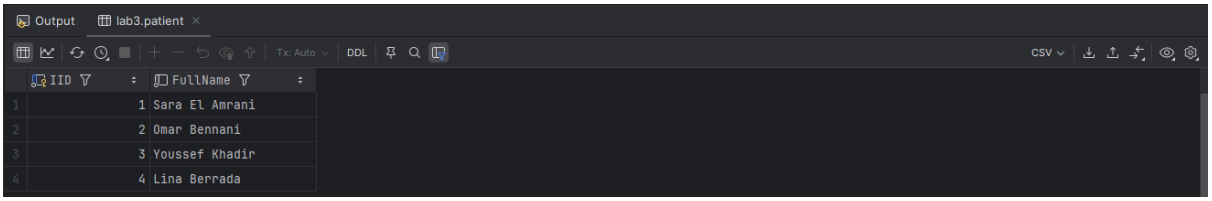
```sql
SELECT D.DEP_ID,
       D.Name AS DepartmentName,
       (SELECT COUNT(*)
        FROM ClinicalActivity CA1
        JOIN Appointment A1 ON CA1.CAID = A1.CAID
        WHERE CA1.DEP_ID = D.DEP_ID
          AND A1.Status = 'Scheduled') AS ScheduledCount,
       (SELECT COUNT(*)
        FROM ClinicalActivity CA2
        JOIN Appointment A2 ON CA2.CAID = A2.CAID
        WHERE CA2.DEP_ID = D.DEP_ID
          AND A2.Status = 'Completed') AS CompletedCount,
       (SELECT COUNT(*)
        FROM ClinicalActivity CA3
        JOIN Appointment A3 ON CA3.CAID = A3.CAID
        WHERE CA3.DEP_ID = D.DEP_ID
          AND A3.Status = 'Cancelled') AS CancelledCount
FROM Department D;
```



	DEP_ID	DepartmentName	ScheduledCount	CompletedCount	CancelledCount
1	1	Cardiology	0	2	0
2	2	Neurology	0	2	0
3	3	Pediatrics	0	0	0
4	4	Surgery	0	1	0
5	5	Radiology	2	0	0

Query 11:

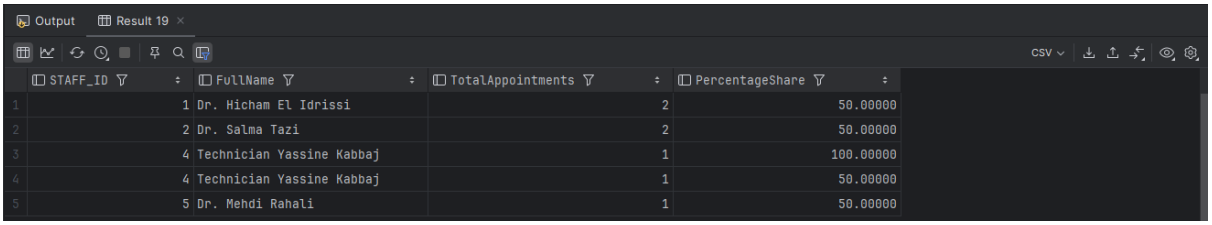
```sql
SELECT P.IID, P.FullName
    FROM Patient P
    WHERE P.IID NOT IN (
        SELECT CA.IID
        FROM Appointment A
        JOIN ClinicalActivity CA ON A.CAID = CA.CAID
        WHERE A.Status = 'Scheduled'
        AND CA.Date BETWEEN CURDATE() AND DATE_ADD(CURDATE(), INTERVAL 30 DAY)
    );
```



	IID	FullName
1	1	Sara El Amrani
2	2	Omar Bennani
3	3	Youssef Khadir
4	4	Lina Berrada

Query 12:

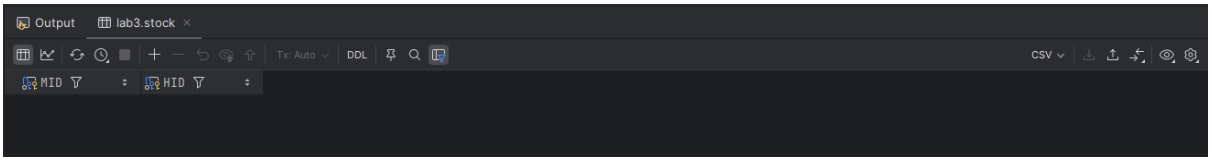
```sql
SELECT S.STAFF_ID,
       S.FullName,
       COUNT(A.CAID) AS TotalAppointments,
       COUNT(A.CAID) * 100.0 /
           (SELECT COUNT(A2.CAID)
            FROM ClinicalActivity CA2
            JOIN Appointment A2 ON CA2.CAID = A2.CAID
            JOIN Department D2 ON CA2.DEP_ID = D2.DEP_ID
            WHERE D2.HID = D.HID) AS PercentageShare
FROM Staff S
JOIN ClinicalActivity CA ON S.STAFF_ID = CA.STAFF_ID
JOIN Appointment A ON CA.CAID = A.CAID
JOIN Department D ON D.DEP_ID = CA.DEP_ID
GROUP BY S.STAFF_ID,S.FullName;
```



	STAFF_ID	FullName	TotalAppointments	PercentageShare
1	1	Dr. Hicham El Idrissi	2	50.00000
2	2	Dr. Salma Tazi	2	50.00000
3	4	Technician Yassine Kabbaj	1	100.00000
4	4	Technician Yassine Kabbaj	1	50.00000
5	5	Dr. Mehdi Rahali	1	50.00000

Query 13:

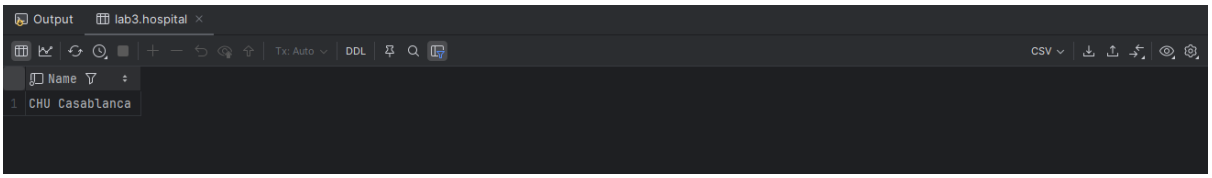
```sql
SELECT MID, HID
    FROM Stock S
    WHERE S.ReorderLevel > S.Qty
    ORDER BY MID;
```



MID	HID

Query 14:

```sql
SELECT H.Name
FROM Hospital H
JOIN Stock S ON S.HID = H.HID
JOIN Medication M ON S.MID = M.MID
WHERE NOT EXISTS ((SELECT MID
                   FROM Medication
                   WHERE TherapeuticClass = 'Antibiotic')
                  EXCEPT
                      (SELECT S2.MID
                       FROM Stock S2
                       JOIN Medication M2 ON M2.MID = S2.MID
                       WHERE S2.HID = H.HID AND M2.TherapeuticClass =
                           'Antibiotic'));
```



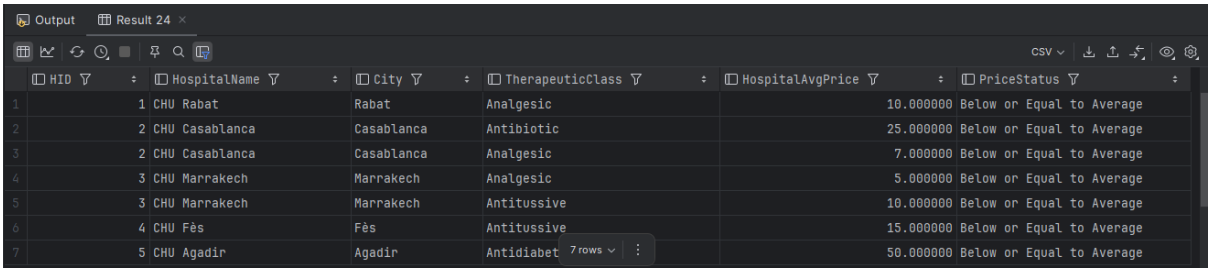
Name
CHU Casablanca

Query 15:

```sql
SELECT H.HID,H.Name AS HospitalName , H.City , M.TherapeuticClass ,
    AVG(S.UnitPrice) AS HospitalAvgPrice , 'Above Average' AS PriceStatus
FROM Hospital H, Stock S, Medication M
WHERE H.HID = S.HID
AND S.MID = M.MID
GROUP BY H.HID, M.TherapeuticClass
HAVING AVG(S.UnitPrice) > (
    SELECT AVG(S2.UnitPrice)
    FROM Stock S2, Hospital H2, Medication M2
    WHERE S2.HID = H2.HID
    AND S2.MID = M2.MID
    AND H2.City = H.City
    AND M2.TherapeuticClass = M.TherapeuticClass
)

UNION

SELECT H.HID,H.Name AS HospitalName , H.City , M.TherapeuticClass ,
    AVG(S.UnitPrice) AS HospitalAvgPrice , 'Below or Equal to Average' AS
    PriceStatus
FROM Hospital H, Stock S, Medication M
WHERE H.HID = S.HID
AND S.MID = M.MID
GROUP BY H.HID, M.TherapeuticClass
HAVING AVG(S.UnitPrice) <= (
    SELECT AVG(S2.UnitPrice)
    FROM Stock S2, Hospital H2, Medication M2
    WHERE S2.HID = H2.HID
    AND S2.MID = M2.MID
    AND H2.City = H.City
    AND M2.TherapeuticClass = M.TherapeuticClass
);
```


Output Result 24

csv

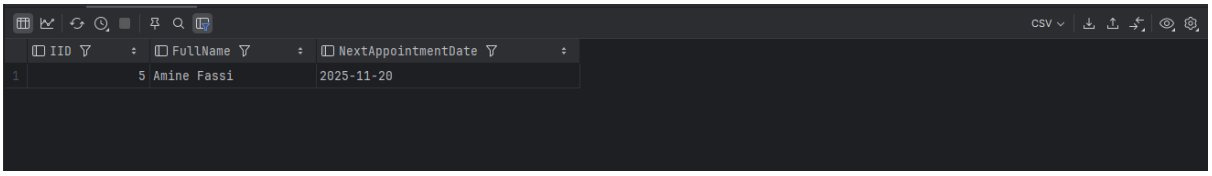
	HID	HospitalName	City	TherapeuticClass	HospitalAvgPrice	PriceStatus
1	1	CHU Rabat	Rabat	Analgesic	10.000000	Below or Equal to Average
2	2	CHU Casablanca	Casablanca	Antibiotic	25.000000	Below or Equal to Average
3	2	CHU Casablanca	Casablanca	Analgesic	7.000000	Below or Equal to Average
4	3	CHU Marrakech	Marrakech	Analgesic	5.000000	Below or Equal to Average
5	3	CHU Marrakech	Marrakech	Antitussive	10.000000	Below or Equal to Average
6	4	CHU Fès	Fès	Antitussive	15.000000	Below or Equal to Average
7	5	CHU Agadir	Agadir	Antidiabet	50.000000	Below or Equal to Average

7 rows

Query 16:

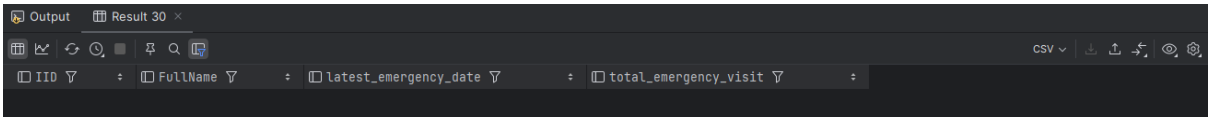
```sql
SELECT P.IID, P.FullName, MIN(CA.Date) AS NextAppointmentDate
FROM Patient P
JOIN ClinicalActivity CA ON P.IID = CA.IID
JOIN Appointment A ON CA.CAID = A.CAID
```

```
WHERE CA.Date >= CURDATE()
AND A.Status = 'Scheduled'
GROUP BY P.IID
HAVING MIN(CA.Date) IS NOT NULL;
```



Query 17:

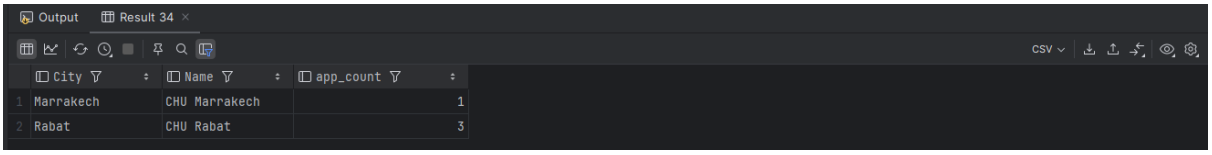
```
SELECT P.IID,P.FullName,MAX(C.Date)as latest_emergency_date,COUNT(*) as
    total_emergency_visit
FROM Patient P
JOIN ClinicalActivity C on P.IID=C.IID
JOIN Emergency E on C.CAID=E.CAID
GROUP BY P.IID
Having count(*)>=2 AND DATEDIFF(CURDATE(), MAX(C.Date)) <= 14;
```



As expected, no patient showed up, and this matches the sample data used in the database.

Query 18:

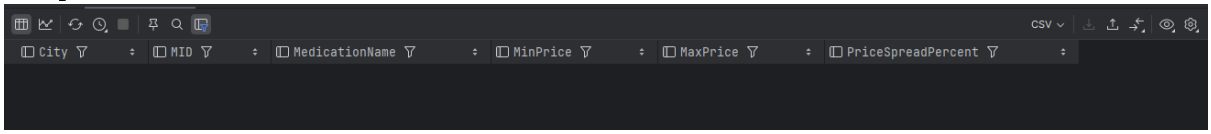
```
SELECT H.City, H.Name, COUNT(DISTINCT A.CAID) as app_count
FROM Hospital H
JOIN Department D ON H.HID = D.HID
JOIN ClinicalActivity CA ON CA.DEP_ID = D.DEP_ID
JOIN Appointment A ON A.CAID = CA.CAID
WHERE CA.Date BETWEEN CURDATE() AND DATE_ADD(CURDATE(), INTERVAL 90 DAY)
AND A.Status = 'Completed'
GROUP BY H.HID
ORDER BY H.City, app_count;
```



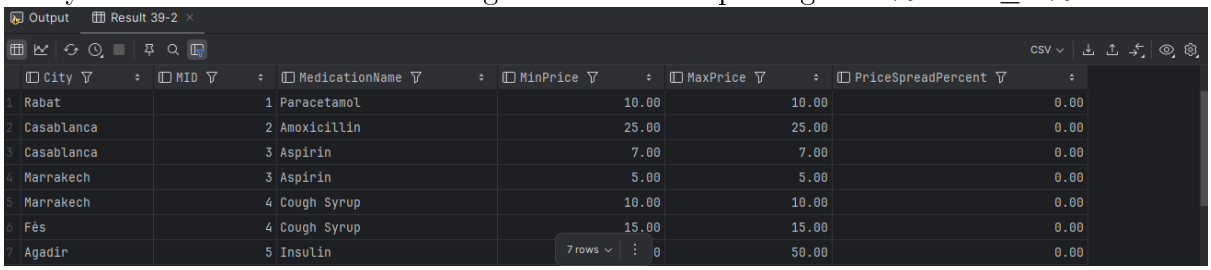
Query 19:

```sql
SELECT
    H.City,
    M.MID,
    M.Name AS MedicationName,
    MIN(ST.UnitPrice) AS MinPrice,
    MAX(ST.UnitPrice) AS MaxPrice,
    ROUND((MAX(ST.UnitPrice) - MIN(ST.UnitPrice)) * 100.0 / MIN(ST.UnitPrice),
        2) AS PriceSpreadPercent
FROM Hospital H
JOIN Stock ST ON H.HID = ST.HID
JOIN Medication M ON ST.MID = M.MID
GROUP BY H.City, M.MID, M.Name
HAVING (MAX(ST.UnitPrice) - MIN(ST.UnitPrice)) * 100.0 / MIN(ST.UnitPrice)>30;
```

In this example, the query doesn't return much information since only around five sample entries were added to each table.



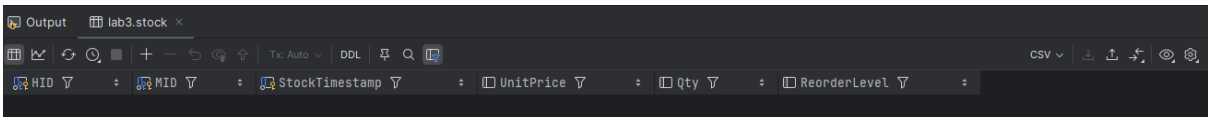
Not a lot of samples were inserted onto the tables, thus, the min and max range are always the same as seen in the image below when replacing $> 30\%$ with $\geq 0\%$




```sql
SELECT
    H.City,
    M.MID,
    M.Name AS MedicationName,
    MIN(ST.UnitPrice) AS MinPrice,
    MAX(ST.UnitPrice) AS MaxPrice,
    ROUND((MAX(ST.UnitPrice) - MIN(ST.UnitPrice)) * 100.0 / MIN(ST.UnitPrice),
        2) AS PriceSpreadPercent
FROM Hospital H
JOIN Stock ST ON H.HID = ST.HID
JOIN Medication M ON ST.MID = M.MID
GROUP BY H.City, M.MID, M.Name
HAVING (MAX(ST.UnitPrice) - MIN(ST.UnitPrice)) * 100.0 / MIN(ST.UnitPrice)>=0;
```

Query 20:

```sql
SELECT *
    FROM Stock
    WHERE Qty < 0 OR UnitPrice <= 0;
```



Since all inserted values were within the expected range and none were negative, the query produces no output.

5 Discussion

Challenges Faced

2 During the implementation of these SQL queries, we encountered several technical challenges that required careful problem-solving. One significant difficulty was managing complex multi-table joins across four or more tables, as seen in Queries 7 and 18, where connecting Hospital through Department to ClinicalActivity and then to Emergency or Appointment tables demanded precise join conditions to avoid incorrect results. Another challenge emerged in queries requiring division operations, such as Query 14, where we had to identify hospitals stocking all antibiotics using NOT EXISTS patterns.

Another area that required particular attention was working with correlated subqueries for analytical calculations. For instance, in Query 12, calculating the percentage share of appointments per hospital while correctly grouping results by staff members proved challenging. We had to carefully structure the query to reference outer query elements like D.HID within subqueries without creating circular dependencies. Furthermore, queries involving date-based filtering, such as Query 16 for upcoming appointments and Query 17 for recent emergency visits, demanded precise use of date functions and interval calculations to ensure accurate results for operational planning.

Observations

Throughout the query development process, we observed several important patterns in healthcare data analysis. Queries involving patient scheduling and appointments (Queries 4, 10, 11, 16) consistently required joining the ClinicalActivity and Appointment tables with careful status filtering. We noticed that medication-related queries (Queries 8, 13, 15, 19) formed a distinct category, typically joining Medication with Stock and Hospital tables to analyze pricing, availability, and distribution patterns.

The aggregation patterns varied significantly based on the analysis objective. Queries like 5 and 7 used straightforward COUNT aggregations, while more complex analyses in Queries 12 and 15 required nested aggregations with subqueries. We also observed that queries dealing with absence or exclusion criteria, such as Query 11 finding patients without appointments, effectively used NOT IN and subquery patterns rather than outer joins with NULL checks.

Lessons Learned

This project taught us useful skills for working with healthcare data. We learned to split complex questions into simpler SQL queries, which made them easier to write and understand. For example, Query 15 showed us how to use UNION to organize hospitals into different price categories, while other queries helped us compare medication costs across different locations.

We also discovered that some query structures work better than others when dealing with large amounts of data. Most importantly, we found that well-planned queries don't just answer one question - they can be reused for future analysis. The work we did here creates a strong base for the hospital system to make better day-to-day decisions and plan for the future.

6 Conclusion

In this part of the project, we applied the principles of database normalization and SQL query formulation to the MNHS dataset. Normalisation allowed us to refine the relational schema by removing redundancy, enforcing data consistency, and ensuring that all relations respected BCNF properties. This guarantees a clean and reliable database foundation for analytical operations. Using the refined schema, we implemented a completed set of SQL queries covering simple selections, multi-table relationships, nested subqueries, aggregate functions, and conditional counting.These queries demonstrate how normalized data can be effectively exploited to extract meaningful insights. Overall, this lab highlights the importance of combining a well-designed schema with expressive SQL queries to support healthcare data management.