University
Mohammed VI
Polytechnic

College of Computing

Deliverable #:Views, Triggers, and Application development

Data Management Course

UM6P College of Computing

Professor: Karima Echihabi **Program:** Computer Engineering
Session: Fall 2025

Team Information

Team Name	QueryMaster
Member 1	El Mehdi Regagui
Member 2	Yasser Jarboua
Member 3	Adam Ibourg-EL Idrissi
Member 4	Salma Mana
Member 5	Hiba Mhirit
Member 6	Sara Qiouame
Member 7	Douaae Mabrouk
Repository Link	`https://github.com/yasserJarboua/QueryMasters/`

1 Introduction

The Moroccan National Health Services (MNHS) database system requires robust management of complex healthcare data including patients, medical staff, hospital departments, appointments, prescriptions, medications, insurance, and billing information. This deliverable addresses three critical aspects of database system enhancement: the implementation of SQL views to optimize query performance and usability, the development of database triggers to enforce business rules and maintain data consistency, and the creation of a comprehensive web application interface. The application layer, built using Python for backend operations with JavaScript, CSS, and HTML for frontend presentation, provides an intuitive platform to interact with the MNHS database while ensuring data integrity through properly implemented database constraints and business logic.

2 Requirements

1. **Views** Define each view in SQL and explain briefly (1–2 sentences) how it can simplify application code or improve query performance (for example by encapsulating complex joins).

   (a) **UpcomingByHospital view.** Build a view that returns, for the next fourteen days, per hospital and per date:`HospitalName`, `ApptDate`, `ScheduledCount`.Use `Appointment` joined through `ClinicalActivity` → `Department` → `Hospital`. Consider only rows with `Appointment.Status = 'Scheduled'`.

   (b) **DrugPricingSummary view.** Build a view that summarizes medication pricing per hospital with the columns:`HID`, `HospitalName`, `MID`, `MedicationName`, `AvgUnitPrice`, `MinUnitPrice`, `MaxUnitPrice`, `LastStockTimestamp`. Use `Stock` with `Hospital` and `Medication`. Group by hospital and medication.

   (c) **StaffWorkloadThirty view.** Build a view that returns per staff member, over the last thirty days:`STAFF_ID`, `FullName`, `TotalAppointments`, `ScheduledCount`, `CompletedCount` ,`CancelledCount`. Source from `Appointment` joined via `ClinicalActivity` → `Staff`. Treat missing counts as zero.

   (d) **PatientNextVisit view.** Build a view that returns, for each patient, the next scheduled visit with:`IID`, `FullName`, `NextApptDate`, `DepartmentName`, `HospitalName`, `City`. Join `Patient` → `ClinicalActivity` → `Appointment` and through `Department` → `Hospital`. Pick the minimum `ClinicalActivity.Date` strictly greater than today among rows with `Status = 'Scheduled'`.

2. **Triggers**

   (a) **Reject double booking for a staff member.** Create a trigger on `Appointment` that rejects an `INSERT` or `UPDATE` if it would schedule two `Appointment` rows at the same `ClinicalActivity.Date` and `ClinicalActivity.Time` for the same`STAFF_ID`. Use `SIGNAL` with a clear error message.

   (b) **Recompute Expense.Total when prescription lines change.** Create triggers on `Includes` for `INSERT`, `UPDATE`, and `DELETE` that recompute the `Expense.Total` of the linked clinical activity as the sum of current `Stock.UnitPrice` for all medications included in the corresponding prescription. Navigate `Includes`

→ `Prescription` → `ClinicalActivity` → `Expense`, and join `Stock` by (`HID` from `Department` → `Hospital` of the activity, `MID`). Use `AFTER INSERT`, `AFTER UPDATE`, and `AFTER DELETE` triggers to recompute the total. If a price is missing for at least one medication, block the change with a clear error using `SIGNAL` (do not update `Expense.Total`).

(c) **Prevent negative or inconsistent stock.** Create `BEFORE INSERT` and `BEFORE UPDATE` triggers on `Stock` that reject any row with `Qty < 0` or `UnitPrice <= 0`, and require `ReorderLevel >= 0`. In addition, require that any change decreasing `Qty` cannot drop below zero. Use `SIGNAL` to reject invalid rows or updates with a clear error message.

(d) **Protect referential integrity on patient delete.** Create a `BEFORE DELETE` trigger on `Patient` that blocks deletion if any `ClinicalActivity` exists for the patient. Use `SIGNAL` to raise an error instructing the user to reassign or delete dependent activities first.

3. **Application development**

- Develop a web application on top of the MNHS database (for example using Python, PHP, or J2EE).
- Implement the following commands in a backend application of your choice (for example using Python, PHP, or another web framework) that connects to the MNHS database.

  (a) **list_patients**
  print the first twenty patients ordered by last name.

  (b) **schedule**
  create a clinical activity and a scheduled appointment in one transaction.

  (c) **low_appt**
  create a clinical activity and a scheduled appointment.

  low_stock: list medications below `ReorderLevel` per hospital with a left join so that medications without stock also appear.

  (d) **staff_share**
  for each staff member compute total number of appointments and percentage share within their hospital. Return a sorted table.

3 Methodology

3.1 Design Approach

We followed a systematic approach: first analyzing the MNHS business requirements, then designing database components (views and triggers), and finally developing the web application layer. Each component was designed to address specific performance, integrity, and usability needs.

3.2 Technology Selection

Database: MySQL for robust relational data management
Backend: Python Flask for rapid web application development

Frontend: HTML/CSS/JavaScript for responsive user interfaces
ORM: SQLAlchemy for database abstraction and security

3.3 Implementation Strategy

Views were designed to encapsulate complex joins and calculations
Triggers were implemented to enforce critical business rules at the database level
The web application was structured using MVC pattern for maintainability
Security measures included parameterized queries and environment variables

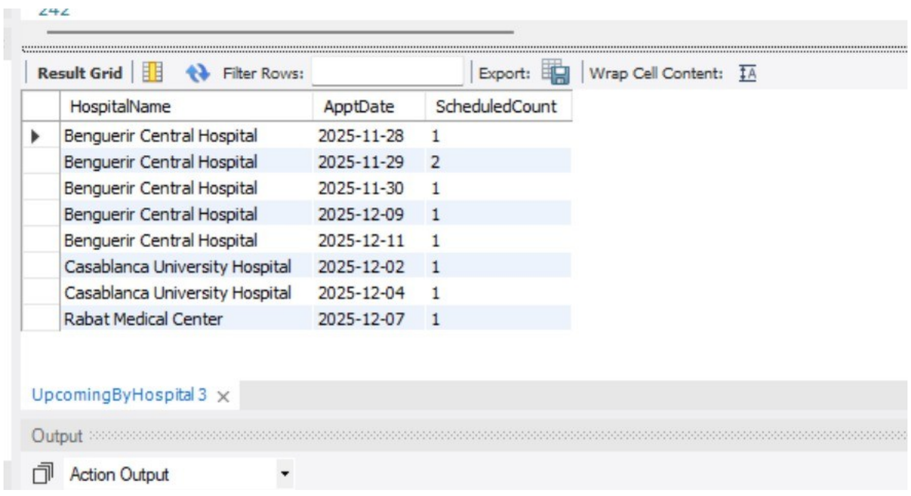
4 Implementation & Results

1. **Views**

   (a) **UpcomingByHospital view.**

```sql
CREATE VIEW UpcomingByHospital AS
SELECT H.Name AS HospitalName,
       CA.Date AS ApptDate,
       COUNT(*) AS ScheduledCount
FROM Appointment A
JOIN ClinicalActivity CA ON A.CAID = CA.CAID
JOIN Department D ON D.DEP_ID = CA.DEP_ID
JOIN Hospital H ON H.HID = D.HID
WHERE A.Status = 'Scheduled'
      AND CA.Date BETWEEN CURDATE() AND DATE_ADD(CURDATE
         (), INTERVAL 14 DAY)
GROUP BY
H.Name,
CA.Date;
-- test1
SELECT * FROM UpcomingByHospital ORDER BY HospitalName,
   ApptDate;
```

   explanation: This view puts all the joins needed to get upcoming appointments for a hospital in one place. It makes queries easier to write and can run faster because the database uses the view's fixed structure.



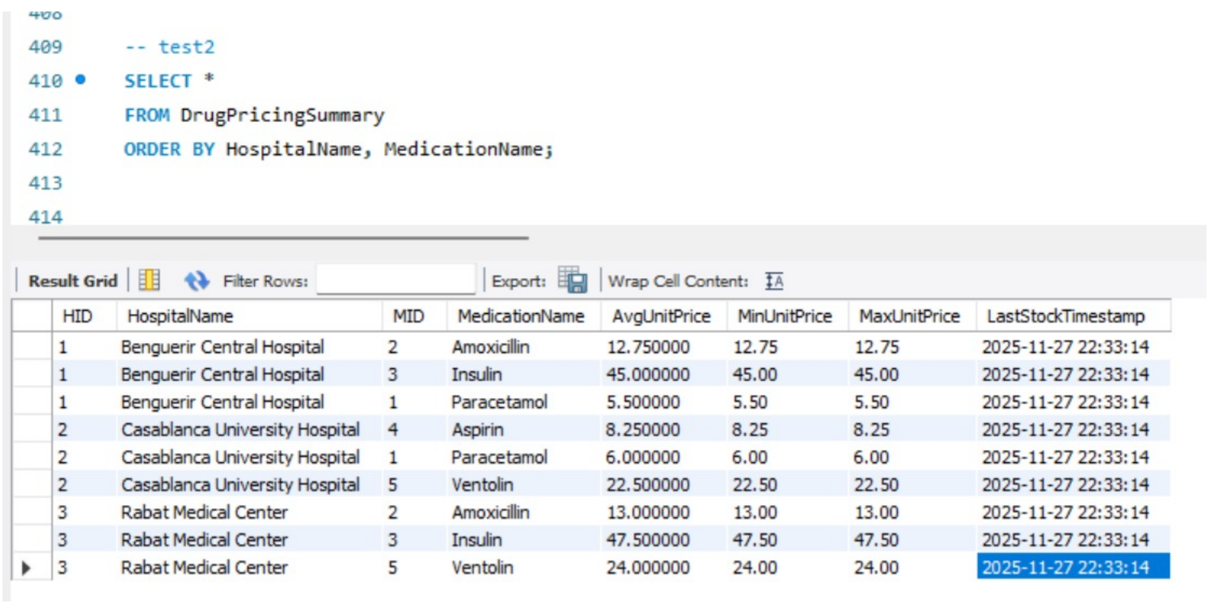
HospitalName	ApptDate	ScheduledCount
Benguerir Central Hospital	2025-11-28	1
Benguerir Central Hospital	2025-11-29	2
Benguerir Central Hospital	2025-11-30	1
Benguerir Central Hospital	2025-12-09	1
Benguerir Central Hospital	2025-12-11	1
Casablanca University Hospital	2025-12-02	1
Casablanca University Hospital	2025-12-04	1
Rabat Medical Center	2025-12-07	1

Figure 1: UpcomingByHospital test

(b) **DrugPricingSummary view.**

```sql
DROP VIEW IF EXISTS DrugPricingSummary;

CREATE VIEW DrugPricingSummary AS
SELECT Hospital.HID,
       Hospital.name AS HospitalName,
       Medication.MID,
       Medication.Name AS MedicationName,
       AVG(Stock.UnitPrice) AS AvgUnitPrice,
       MIN(Stock.UnitPrice) AS MinUnitPrice,
       MAX(Stock.UnitPrice) AS MaxUnitPrice,
       MAX(Stock.StockTimestamp) AS LastStockTimestamp
FROM Stock
JOIN Hospital  ON Hospital.HID = Stock.HID
JOIN Medication ON Medication.MID = Stock.MID
GROUP BY Hospital.HID, Hospital.name, Medication.MID,
    Medication.Name;
-- test2
SELECT *
FROM DrugPricingSummary
ORDER BY HospitalName, MedicationName;
```

explanation: The DrugPricingSummary view provides a consolidated overview of medication prices across hospitals by joining the Stock, Hospital, and Medication tables to recreate the full relationship between each hospital and the medications it stores. For every medication–hospital pair, the view computes essential pricing statistics—including the average, minimum, and maximum prices—as well as the most recent stock update date. These metrics are produced using aggregate functions after grouping the data by hospital and medication, ensuring that each row of the view represents one medication within a specific hospital. By encapsulating these joins and calculations inside a view, the database simplifies future queries, ensures consistency across the application, and improves efficiency whenever pricing information is needed, making it easier to track price variations and monitor stock updates across hospitals.



```
-- test2
SELECT *
FROM DrugPricingSummary
ORDER BY HospitalName, MedicationName;
```

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

HID	HospitalName	MID	MedicationName	AvgUnitPrice	MinUnitPrice	MaxUnitPrice	LastStockTimestamp
1	Benguerir Central Hospital	2	Amoxicillin	12.750000	12.75	12.75	2025-11-27 22:33:14
1	Benguerir Central Hospital	3	Insulin	45.000000	45.00	45.00	2025-11-27 22:33:14
1	Benguerir Central Hospital	1	Paracetamol	5.500000	5.50	5.50	2025-11-27 22:33:14
2	Casablanca University Hospital	4	Aspirin	8.250000	8.25	8.25	2025-11-27 22:33:14
2	Casablanca University Hospital	1	Paracetamol	6.000000	6.00	6.00	2025-11-27 22:33:14
2	Casablanca University Hospital	5	Ventolin	22.500000	22.50	22.50	2025-11-27 22:33:14
3	Rabat Medical Center	2	Amoxicillin	13.000000	13.00	13.00	2025-11-27 22:33:14
3	Rabat Medical Center	3	Insulin	47.500000	47.50	47.50	2025-11-27 22:33:14
3	Rabat Medical Center	5	Ventolin	24.000000	24.00	24.00	2025-11-27 22:33:14

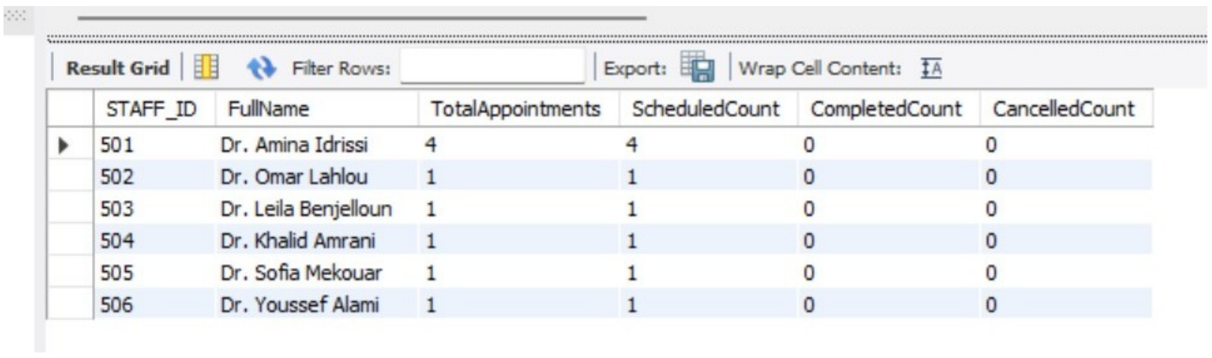
Figure 2: DrugPricingSummary test

(c) **StaffWorkloadThirty view.**

```sql
CREATE OR REPLACE VIEW StaffWorkloadThirty AS
SELECT
    s.STAFF_ID,
    s.FullName,
    COUNT(a.CAID) AS TotalAppointments,
    SUM(a.status = 'Scheduled') AS ScheduledCount,
    SUM(a.status = 'Completed') AS CompletedCount,
    SUM(a.status = 'Cancelled') AS CancelledCount
FROM Staff s
LEFT JOIN ClinicalActivity ca
       ON s.STAFF_ID = ca.STAFF_ID
       AND ca.DATE >= CURDATE() - INTERVAL 30 DAY
LEFT JOIN Appointment a
       ON a.CAID = ca.CAID
GROUP BY s.STAFF_ID, s.FullName;
-- test 3
-- Update the dates so they are within the last 30 days
UPDATE ClinicalActivity
SET Date = CURDATE() - INTERVAL 5 DAY
WHERE CAID = 101;

UPDATE ClinicalActivity
SET Date = CURDATE() - INTERVAL 10 DAY
WHERE CAID = 102;

UPDATE ClinicalActivity
SET Date = CURDATE() - INTERVAL 15 DAY
WHERE CAID = 103;

UPDATE ClinicalActivity
```

```sql
SET Date = CURDATE() - INTERVAL 20 DAY
WHERE CAID IN (104, 105, 106);
-- Check the new dates
SELECT CAID, STAFF_ID, Date,
       DATEDIFF(CURDATE(), Date) as Jours_Depuis
FROM ClinicalActivity
ORDER BY Date DESC;
SELECT * FROM StaffWorkloadThirty ORDER BY
   TotalAppointments DESC;
```

explanation: This view simplifies application code by encapsulating the complex joins and aggregations between Staff, ClinicalActivity, and Appointment, so the application can retrieve each staff's workload with a single query. It can also improve query performance because the database can optimize the view internally, avoiding repeated recomputation of the joins and aggregations every time the data is needed.



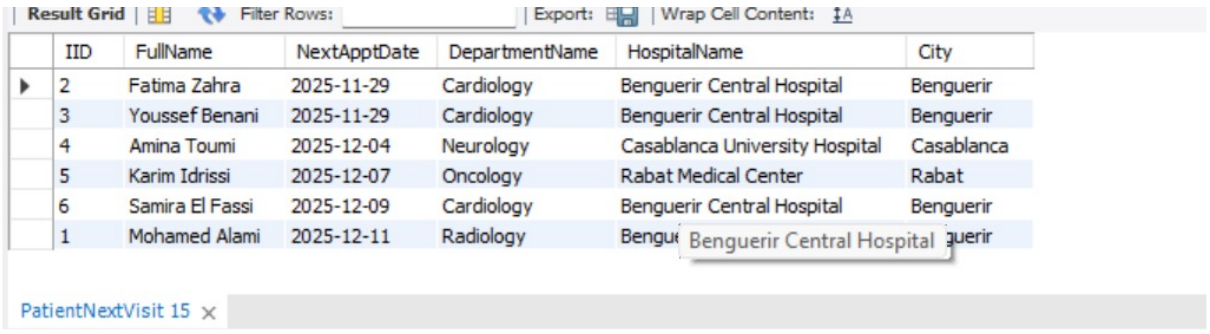
STAFF_ID	FullName	TotalAppointments	ScheduledCount	CompletedCount	CancelledCount
501	Dr. Amina Idrissi	4	4	0	0
502	Dr. Omar Lahlou	1	1	0	0
503	Dr. Leila Benjelloun	1	1	0	0
504	Dr. Khalid Amrani	1	1	0	0
505	Dr. Sofia Mekouar	1	1	0	0
506	Dr. Youssef Alami	1	1	0	0

Figure 3: StaffWorkloadThirty test

(d) **PatientNextVisit view.**

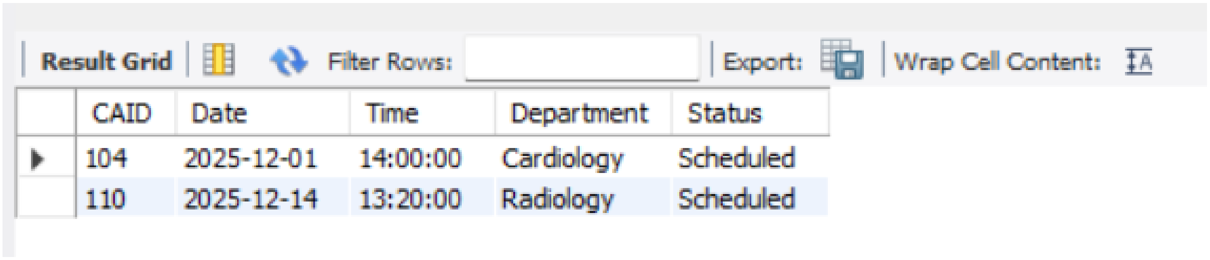
```sql
CREATE VIEW PatientNextVisit AS
SELECT P.IID AS IID,
       P.FullName AS FullName,
       CA.Date AS NextApptDate,
       D.Name AS DepartmentName,
       H.Name AS HospitalName,
       H.City AS City
FROM Patient P
LEFT JOIN (
      SELECT CA.IID,
             MIN(CA.DATE) AS NextDate
          FROM ClinicalActivity CA
      JOIN Appointment A ON CA.CAID = A.CAID
      WHERE A.Status = 'Scheduled'
           AND CA.Date > CURDATE()
          GROUP BY CA.IID
      ) NextVisit ON NextVisit.IID = P.IID
LEFT JOIN ClinicalActivity CA ON CA.IID = P.IID AND CA.
   Date = NextVisit.NextDate
LEFT JOIN Appointment A ON A.CAID = CA.CAID
LEFT JOIN Department D ON D.DEP_ID = CA.DEP_ID
LEFT JOIN Hospital H ON H.HID = D.HID;
SELECT * FROM PatientNextVisit ORDER BY NextApptDate;
--Let's check all future activities of Mohamed Alami (IID
    =1)
SELECT CA.CAID, CA.Date, CA.Time, D.Name as Department, A
    .Status
FROM ClinicalActivity CA
JOIN Department D ON CA.DEP_ID = D.DEP_ID
JOIN Appointment A ON CA.CAID = A.CAID
WHERE CA.IID = 1
  AND A.Status = 'Scheduled'
  AND CA.Date > CURDATE()
ORDER BY CA.Date;
```

explanation: This view gives each patient's next appointment automatically, so the application doesn't need to write the sorting and filtering each time. It makes the code shorter and can speed things up because the database handles the work inside the view.



IID	FullName	NextApptDate	DepartmentName	HospitalName	City
2	Fatima Zahra	2025-11-29	Cardiology	Benguerir Central Hospital	Benguerir
3	Youssef Benani	2025-11-29	Cardiology	Benguerir Central Hospital	Benguerir
4	Amina Toumi	2025-12-04	Neurology	Casablanca University Hospital	Casablanca
5	Karim Idrissi	2025-12-07	Oncology	Rabat Medical Center	Rabat
6	Samira El Fassi	2025-12-09	Cardiology	Benguerir Central Hospital	Benguerir
1	Mohamed Alami	2025-12-11	Radiology	Benguerir Central Hospital	Benguerir

Figure 4: PatientNextVisit test1



CAID	Date	Time	Department	Status
104	2025-12-01	14:00:00	Cardiology	Scheduled
110	2025-12-14	13:20:00	Radiology	Scheduled

Figure 5: PatientNextVisit test2

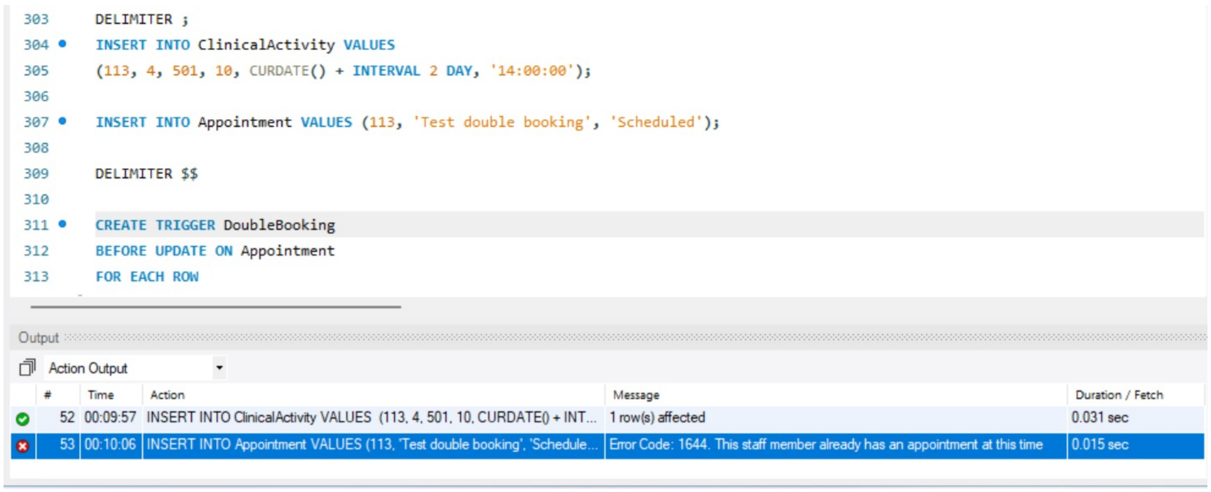
2. **Triggers**

(a) **Reject double booking for a staff member.**

```sql
DELIMITER $$

CREATE TRIGGER DoubleBooking
BEFORE INSERT ON Appointment
FOR EACH ROW
```

```sql
BEGIN
     DECLARE cnt INT;
     DECLARE new_staff INT;
     DECLARE new_date DATE;
     DECLARE new_time TIME;

     SELECT STAFF_ID, Date, time
     INTO new_staff, new_date, new_time
     FROM ClinicalActivity
     WHERE CAID = NEW.CAID;

     SELECT COUNT(*) INTO cnt
     FROM ClinicalActivity
     WHERE STAFF_ID = new_staff
           AND Date = new_date
           AND Time = new_time;

     IF cnt > 0 THEN
           SIGNAL SQLSTATE '45000'
                SET MESSAGE_TEXT = 'This staff member
                   already has an appointment at this
                   time';
         END IF;
END $$
DELIMITER ;
INSERT INTO ClinicalActivity VALUES
(113, 4, 501, 10, CURDATE() + INTERVAL 2 DAY, '14:00:00')
   ;

INSERT INTO Appointment VALUES (113, 'Test double booking
   ', 'Scheduled');

DELIMITER $$

CREATE TRIGGER DoubleBooking
BEFORE UPDATE ON Appointment
FOR EACH ROW
BEGIN
     DECLARE cnt INT;
     DECLARE new_staff INT;
     DECLARE new_date DATE;
     DECLARE new_time TIME;

     SELECT STAFF_ID, Date, time
     INTO new_staff, new_date, new_time
     FROM ClinicalActivity
     WHERE CAID = NEW.CAID;

     SELECT COUNT(*) INTO cnt
     FROM ClinicalActivity
     WHERE STAFF_ID = new_staff
```

```
            AND Date = new_date
            AND Time = new_time
            AND CAID <> NEW.CAID;

    IF cnt > 0 THEN
            SIGNAL SQLSTATE '45000'
                SET MESSAGE_TEXT = 'This staff member
                    already has an appointment at this
                    time';
        END IF;
END $$
DELIMITER ;
```



```
DELIMITER ;
INSERT INTO ClinicalActivity VALUES
(113, 4, 501, 10, CURDATE() + INTERVAL 2 DAY, '14:00:00');

INSERT INTO Appointment VALUES (113, 'Test double booking', 'Scheduled');

DELIMITER $$

CREATE TRIGGER DoubleBooking
BEFORE UPDATE ON Appointment
FOR EACH ROW
```

Output

Action Output

```
#   Time      Action                                                           Message                                                          Duration / Fetch
52  00:09:57  INSERT INTO ClinicalActivity VALUES (113, 4, 501, 10, CURDATE() + INT...  1 row(s) affected                                          0.031 sec
53  00:10:06  INSERT INTO Appointment VALUES (113, 'Test double booking', 'Schedule...  Error Code: 1644. This staff member already has an appointment at this time  0.015 sec
```

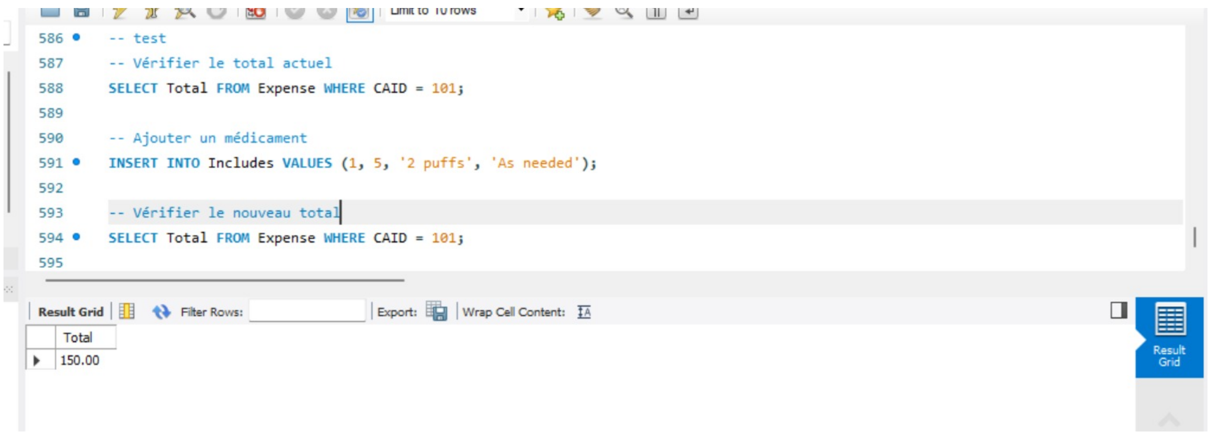
Figure 6: Reject double booking for a staff member test

(b) **Recompute Expense.Total when prescription lines change.**

```
DELIMITER $$
CREATE TRIGGER RecomputeTotal_AfterInsert
AFTER INSERT ON Includes
FOR EACH ROW
BEGIN
    DECLARE hospital_id INT;
    DECLARE computed_total DECIMAL(10,2);

    SELECT Department.HID
    INTO hospital_id
    FROM ClinicalActivity
    JOIN Department ON ClinicalActivity.DEP_ID =
        Department.DEP_ID
    JOIN Prescription ON Prescription.CAID =
        ClinicalActivity.CAID
    WHERE Prescription.PID = NEW.PID;

    IF EXISTS (
        SELECT 1
        FROM Includes inc
        LEFT JOIN Stock stk
            ON stk.MID = inc.MID AND stk.HID =
               hospital_id
        WHERE inc.PID = NEW.PID
        AND stk.UnitPrice IS NULL
    ) THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Error: Missing price for
               at least one medication in this
               prescription.';
    END IF;

    SELECT SUM(stk.UnitPrice)
    INTO computed_total
    FROM Includes inc
    JOIN Stock stk
        ON stk.MID = inc.MID AND stk.HID = hospital_id
    WHERE inc.PID = NEW.PID;

    UPDATE Expense
    SET Total = computed_total
    WHERE CAID = (
        SELECT CAID FROM Prescription WHERE PID = NEW.PID
    );
END$$

DELIMITER ;
DELIMITER $$

CREATE TRIGGER RecomputeTotal_AfterUpdate
AFTER UPDATE ON Includes
FOR EACH ROW
BEGIN
    DECLARE hospital_id INT;
    DECLARE computed_total DECIMAL(10,2);
    DECLARE prescription_id INT;

    SET prescription_id = NEW.PID;

    SELECT Department.HID
    INTO hospital_id
    FROM ClinicalActivity
    JOIN Department ON ClinicalActivity.DEP_ID =
       Department.DEP_ID
    JOIN Prescription ON Prescription.CAID =
       ClinicalActivity.CAID
    WHERE Prescription.PID = prescription_id;

    IF EXISTS (
        SELECT 1
```

```sql
        FROM Includes inc
        LEFT JOIN Stock stk
            ON stk.MID = inc.MID AND stk.HID =
               hospital_id
        WHERE inc.PID = prescription_id
        AND stk.UnitPrice IS NULL
    ) THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Error: Missing price for
               at least one medication in this
               prescription.';
    END IF;

    -- Compute total
    SELECT SUM(stk.UnitPrice)
    INTO computed_total
    FROM Includes inc
    JOIN Stock stk
        ON stk.MID = inc.MID AND stk.HID = hospital_id
    WHERE inc.PID = prescription_id;

    UPDATE Expense
    SET Total = computed_total
    WHERE CAID = (
        SELECT CAID FROM Prescription WHERE PID =
           prescription_id
    );
END$$

DELIMITER ;
DELIMITER $$

CREATE TRIGGER RecomputeTotal_AfterDelete
AFTER DELETE ON Includes
FOR EACH ROW
BEGIN
    DECLARE hospital_id INT;
    DECLARE computed_total DECIMAL(10,2);
    DECLARE prescription_id INT;

    SET prescription_id = OLD.PID;

    SELECT Department.HID
    INTO hospital_id
    FROM ClinicalActivity
    JOIN Department ON ClinicalActivity.DEP_ID =
       Department.DEP_ID
    JOIN Prescription ON Prescription.CAID =
       ClinicalActivity.CAID
    WHERE Prescription.PID = prescription_id;

    IF EXISTS (
        SELECT 1
        FROM Includes inc
        LEFT JOIN Stock stk
            ON stk.MID = inc.MID AND stk.HID =
               hospital_id
        WHERE inc.PID = prescription_id
        AND stk.UnitPrice IS NULL
    ) THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Error: Missing price for
               at least one medication in this
               prescription.';
    END IF;

    SELECT SUM(stk.UnitPrice)
    INTO computed_total
    FROM Includes inc
    JOIN Stock stk
        ON stk.MID = inc.MID AND stk.HID = hospital_id
    WHERE inc.PID = prescription_id;

    UPDATE Expense
    SET Total = computed_total
    WHERE CAID = (
        SELECT CAID FROM Prescription WHERE PID =
           prescription_id
    );
END$$

DELIMITER ;
-- test
--Check the current total
SELECT Total FROM Expense WHERE CAID = 101;

-- Add a drug
INSERT INTO Includes VALUES (1, 5, '2 puffs', 'As needed'
   );

-- Check the new total
SELECT Total FROM Expense WHERE CAID = 101;
-- Check the data before testing
SELECT E.ExpID, E.CAID, E.Total, P.PID, I.MID, M.Name, S.
   UnitPrice
FROM Expense E
JOIN Prescription P ON E.CAID = P.CAID
JOIN Includes I ON P.PID = I.PID
JOIN Medication M ON I.MID = M.MID
JOIN ClinicalActivity CA ON P.CAID = CA.CAID
JOIN Department D ON CA.DEP_ID = D.DEP_ID
LEFT JOIN Stock S ON S.MID = I.MID AND S.HID = D.HID
```

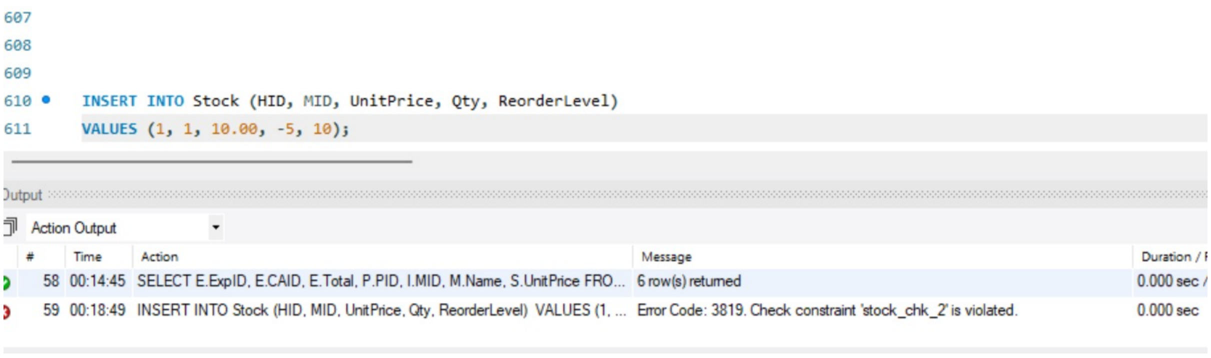
```
ORDER BY E.ExpID, I.MID;

INSERT INTO Stock (HID, MID, UnitPrice, Qty, ReorderLevel
    )
VALUES (1, 1, 10.00, -5, 10);
UPDATE Stock
SET Qty = 80, UnitPrice = 6.00, ReorderLevel = 15
WHERE HID = 1 AND MID = 1;
SELECT HID, MID, Qty, UnitPrice, ReorderLevel
FROM Stock
WHERE HID = 1 AND MID = 1;
DELIMITER $$
```



```
-- test
-- Vérifier le total actuel
SELECT Total FROM Expense WHERE CAID = 101;

-- Ajouter un médicament
INSERT INTO Includes VALUES (1, 5, '2 puffs', 'As needed');

-- Vérifier le nouveau total
SELECT Total FROM Expense WHERE CAID = 101;
```

```
Total
150.00
```

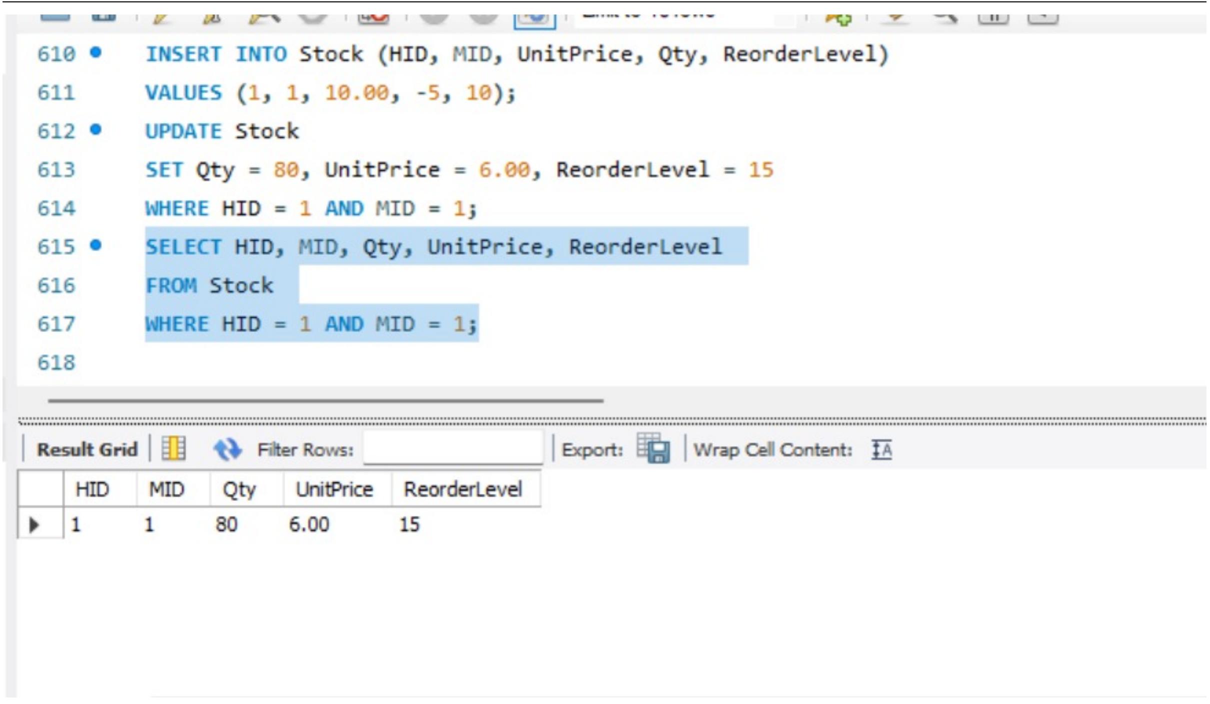
Figure 7: Recompute Expense test



```
INSERT INTO Stock (HID, MID, UnitPrice, Qty, ReorderLevel)
VALUES (1, 1, 10.00, -5, 10);
```

```
Action Output
#   Time      Action                                                              Message                                                 Duration / F
58  00:14:45  SELECT E.ExpID, E.CAID, E.Total, P.PID, I.MID, M.Name, S.UnitPrice FRO...  6 row(s) returned                                       0.000 sec /
59  00:18:49  INSERT INTO Stock (HID, MID, UnitPrice, Qty, ReorderLevel) VALUES (1, ...  Error Code: 3819. Check constraint 'stock_chk_2' is violated.  0.000 sec
```

Figure 8: Recompute Expense test



```
INSERT INTO Stock (HID, MID, UnitPrice, Qty, ReorderLevel)
VALUES (1, 1, 10.00, -5, 10);
UPDATE Stock
SET Qty = 80, UnitPrice = 6.00, ReorderLevel = 15
WHERE HID = 1 AND MID = 1;
SELECT HID, MID, Qty, UnitPrice, ReorderLevel
FROM Stock
WHERE HID = 1 AND MID = 1;
```

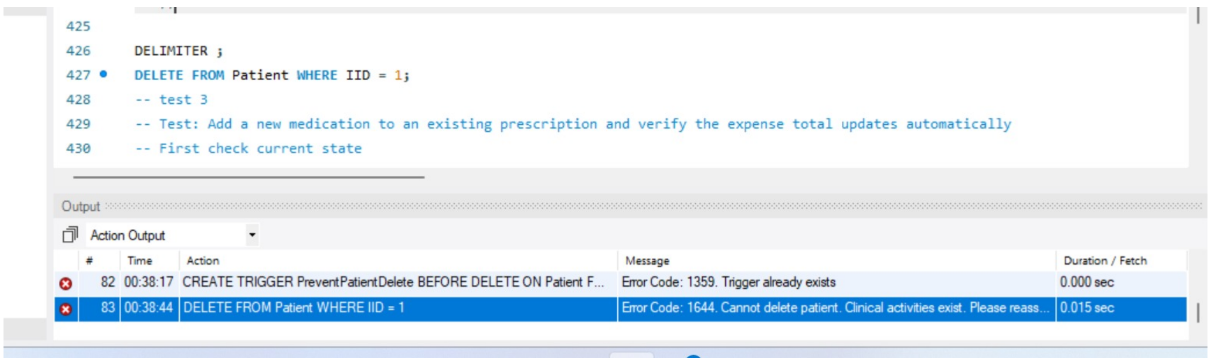
Result Grid | Filter Rows: | Export: | Wrap Cell Content:

HID	MID	Qty	UnitPrice	ReorderLevel
1	1	80	6.00	15

Figure 9: Recompute Expense test

(c) **Prevent negative or inconsistent stock.**

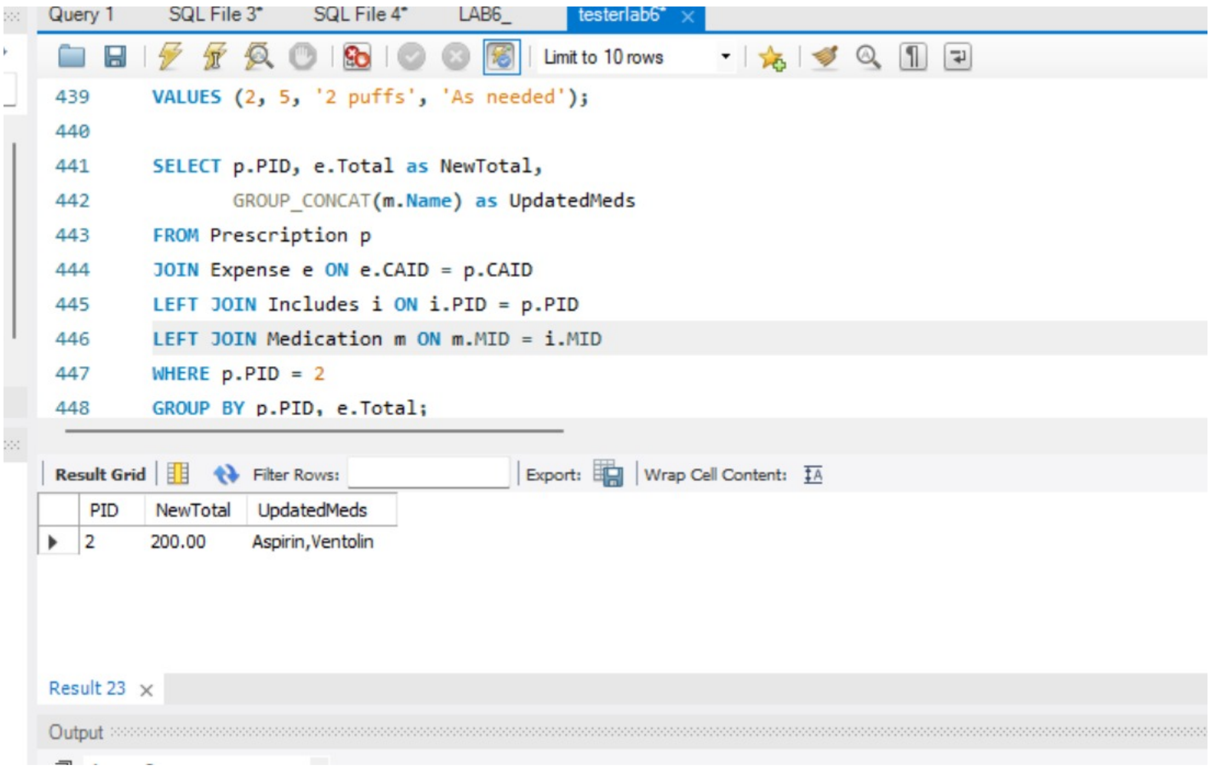
```
DELIMITER //
CREATE TRIGGER insertStock
BEFORE INSERT ON stock
FOR EACH ROW
BEGIN
    IF NEW.Qty < 0 OR NEW.UnitPrice <= 0 OR NEW.
       ReorderLevel < 0 THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = "Cannot insert negative
               stock quantities, unit price, or reorder
               level";
    END IF;
END;
//
DELIMITER;

-- update trigger:

DELIMITER //
CREATE TRIGGER updateStock
BEFORE UPDATE ON stock
FOR EACH ROW
BEGIN
    IF NEW.Qty < 0 OR NEW.UnitPrice <= 0 OR NEW.
       ReorderLevel < 0 THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = "Cannot update to negative
                stock quantities, unit price, or reorder
                level";
    END IF;

    IF NEW.Qty < OLD.Qty AND NEW.Qty < 0 THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = "Cannot decrease Qty below
                zero";
    END IF;
END;
//

DELETE FROM Patient WHERE IID = 1;
-- test 3
-- Test: Add a new medication to an existing prescription
    and verify the expense total updates automatically
-- First check current state
SELECT p.PID, e.Total as CurrentTotal,
       GROUP_CONCAT(m.Name) as CurrentMeds
FROM Prescription p
JOIN Expense e ON e.CAID = p.CAID
LEFT JOIN Includes i ON i.PID = p.PID
LEFT JOIN Medication m ON m.MID = i.MID
WHERE p.PID = 2
GROUP BY p.PID, e.Total;

-- Add Amoxicillin to prescription 1
INSERT INTO Includes (PID, MID, Dosage, Duration)
VALUES (2, 5, '2 puffs', 'As needed');

SELECT p.PID, e.Total as NewTotal,
       GROUP_CONCAT(m.Name) as UpdatedMeds
FROM Prescription p
JOIN Expense e ON e.CAID = p.CAID
LEFT JOIN Includes i ON i.PID = p.PID
LEFT JOIN Medication m ON m.MID = i.MID
WHERE p.PID = 2
GROUP BY p.PID, e.Total;

DELIMITER ;
```



```
DELIMITER ;
DELETE FROM Patient WHERE IID = 1;
-- test 3
-- Test: Add a new medication to an existing prescription and verify the expense total updates automatically
-- First check current state
```

```
Output
Action Output
 #   Time      Action                                                       Message                                                                   Duration / Fetch
 82  00:38:17  CREATE TRIGGER PreventPatientDelete BEFORE DELETE ON Patient F...  Error Code: 1359. Trigger already exists                                 0.000 sec
 83  00:38:44  DELETE FROM Patient WHERE IID = 1                            Error Code: 1644. Cannot delete patient. Clinical activities exist. Please reass...  0.015 sec
```

Figure 10: Prevent negative or inconsistent stock test



```
VALUES (2, 5, '2 puffs', 'As needed');

SELECT p.PID, e.Total as NewTotal,
       GROUP_CONCAT(m.Name) as UpdatedMeds
FROM Prescription p
JOIN Expense e ON e.CAID = p.CAID
LEFT JOIN Includes i ON i.PID = p.PID
LEFT JOIN Medication m ON m.MID = i.MID
WHERE p.PID = 2
GROUP BY p.PID, e.Total;
```

PID	NewTotal	UpdatedMeds
2	200.00	Aspirin,Ventolin

Figure 11: Prevent negative or inconsistent stock test

(d) **Protect referential integrity on patient delete.**

```
DELIMITER $$
CREATE TRIGGER PreventPatientDelete
BEFORE DELETE ON Patient
FOR EACH ROW
BEGIN
    IF EXISTS (SELECT 1 FROM ClinicalActivity WHERE IID =
        OLD.IID) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Cannot delete patient.
           Clinical activities exist. Please reassign or
           delete dependent activities first.';
    END IF;
```

```
END$$

DELIMITER ;
```

3. **Web Application Implementation**
   Our MNHS web application was developed using a Python Flask backend with JavaScript, CSS, and HTML frontend, providing an intuitive interface for database interactions.

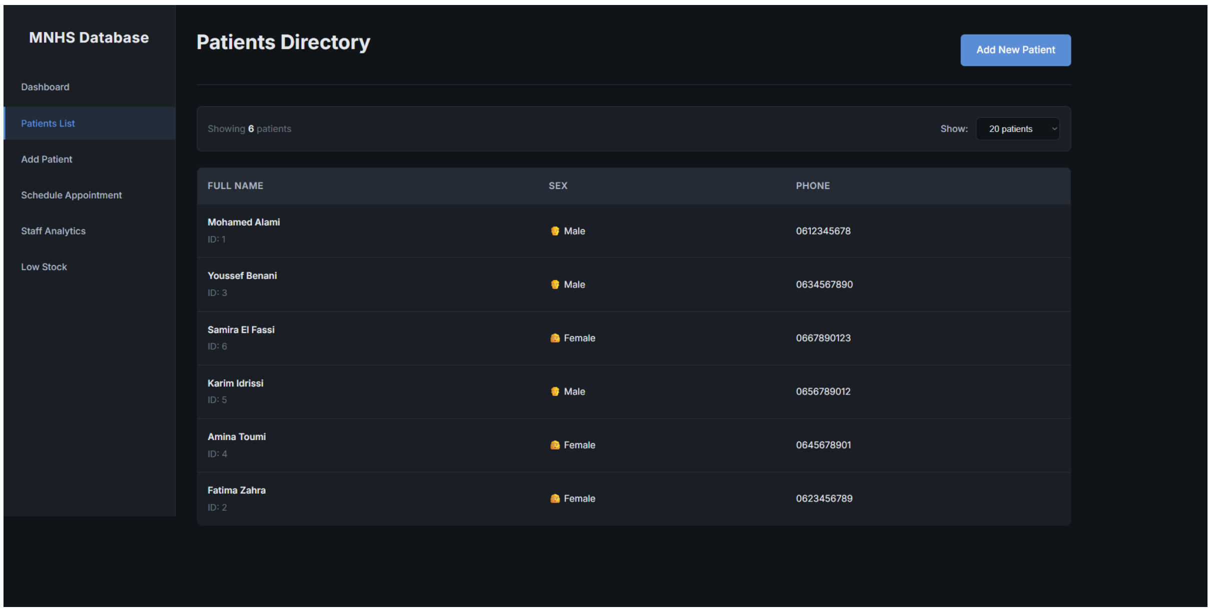
   (a) **Application Architecture**
   Our web application follows a three-tier architecture with Flask handling the backend logic, SQLAlchemy managing database operations, and HTML/CSS/JavaScript providing the user interface. The modular structure separates concerns between data access, business logic, and presentation layers.

```python
import os
from dotenv import load_dotenv
from flask_sqlalchemy import SQLAlchemy
from flask import jsonify


load_dotenv()
db = SQLAlchemy()

def get_database_config():
    """Returns database configuration from environment variables"""
    return dict(
        host=os.getenv("MYSQL_HOST"),
        port=int(os.getenv("MYSQL_PORT", 3306)),
        database=os.getenv("MYSQL_DB"),
        user=os.getenv("MYSQL_USER"),
        password=os.getenv("MYSQL_PASSWORD")
    )

def init_db(app):
    """Initialize database with Flask app"""
    cfg = get_database_config()
    app.config['SQLALCHEMY_DATABASE_URI'] = f'mysql+pymysql://{cfg["user"]}:{cfg["password"]}@{cfg["host"]}/{cfg["database"]}'
    app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
    db.init_app(app)
    return db
```

   (b) **Command list_patients: Backend Implementation**:

```python
    def list_patients_ordered_by_last_name(limit=20):
    query = db.text(f"""
    SELECT IID, FullName, Sex, Phone
    FROM Patient
    ORDER BY SUBSTRING_INDEX(FullName, ' ', -1), FullName
    LIMIT {limit}
    """)
    result = db.session.execute(query).fetchall()
    res = [
        {
            "IID": patient[0],
            "FullName": patient[1],
            "Sex": patient[2],
            "Phone": patient[3],
        }
        for patient in result
    ]
    return res
```

Patients List Interface



Patients Directory

Showing **6** patients

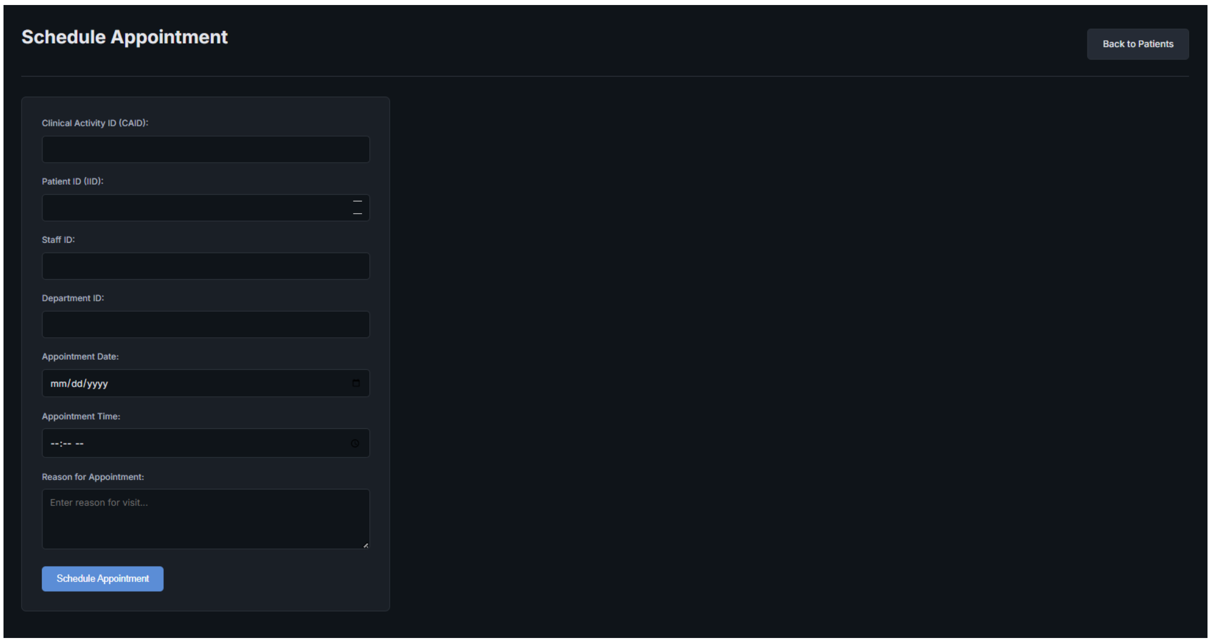
FULL NAME	SEX	PHONE
Mohamed Alami ID: 1	Male	0612345678
Youssef Benani ID: 3	Male	0634567890
Samira El Fassi ID: 6	Female	0667890123
Karim Idrissi ID: 5	Male	0656789012
Amina Toumi ID: 4	Female	0645678901
Fatima Zahra ID: 2	Female	0623456789

Figure 12: Patients list command interface

(c) **Command schedule_appt: Backend Implementation:**

```
    def schedule_appointment(caid, iid, staff_id, dep_id,
        date_str, time_str, reason):
    ins_ca = db.text("""
        INSERT INTO ClinicalActivity(CAID, IID, STAFF_ID,
            DEP_ID, Date, Time)
        VALUES (:caid, :iid, :staff_id, :dep_id, :
           date_str, :time_str)
    """)

    ins_appt = db.text("""
        INSERT INTO Appointment(CAID, Reason, Status)
        VALUES (:caid, :reason, 'Scheduled')
    """)
    try:
        db.session.execute(ins_ca, {
            "caid": caid,
            "iid": iid,
            "staff_id": staff_id,
            "dep_id": dep_id,
            "date_str": date_str,
            "time_str": time_str
        })
        db.session.execute(ins_appt, {
            "caid": caid,
            "reason": reason
        })
        db.session.commit()
    except Exception as e:
        db.session.rollback()
        raise Exception(f"OPERATION FAILED: {e}")
```

Schedule Appointment Interface


Schedule Appointment

Clinical Activity ID (CAID):

Patient ID (IID):

Staff ID:

Department ID:

Appointment Date: mm/dd/yyyy

Appointment Time: --:-- --

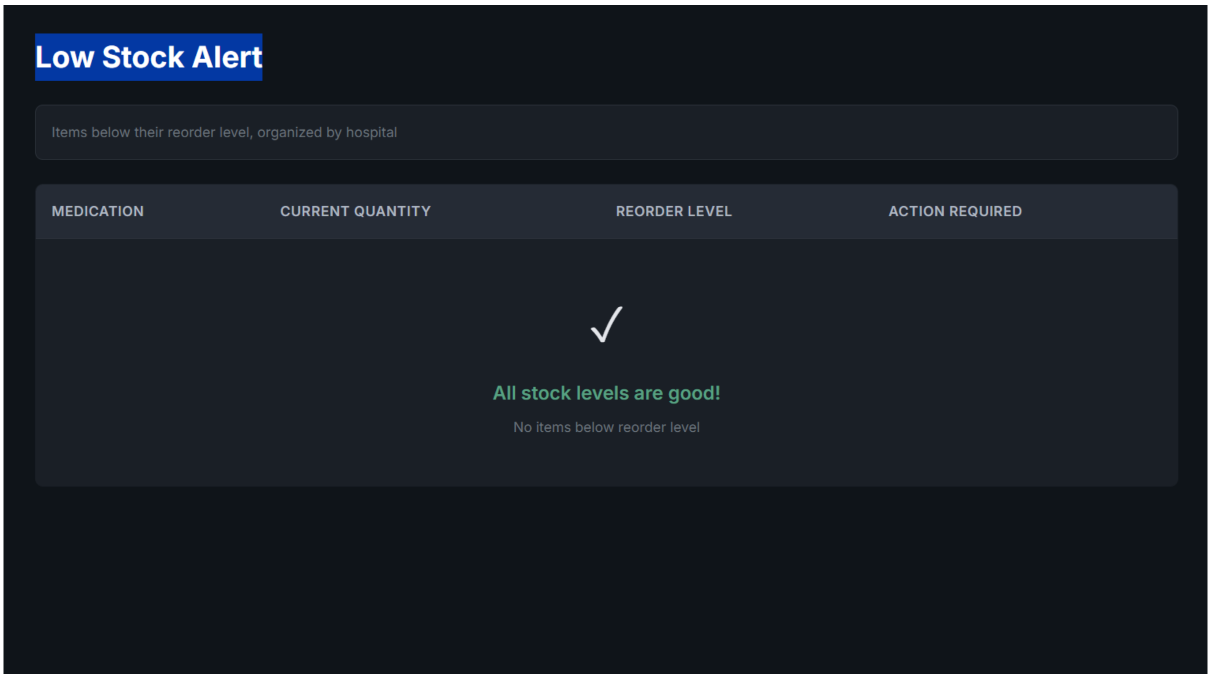
Reason for Appointment: Enter reason for visit...

Figure 13: Schedule Appointment

(d) **Command low_stock: Backend Implementation**:

```
    def get_low_stock():
    query = db.text("""
        SELECT
            h.HID,
            h.Name AS HospitalName,
            m.MID,
            m.Name AS MedicationName,
            COALESCE(s.Qty, 0) AS Quantity,
            COALESCE(s.ReorderLevel, 10) AS ReorderLevel
        FROM Medication m
        LEFT JOIN Stock s ON s.MID = m.MID
        JOIN Hospital h ON s.HID = h.HID
        WHERE COALESCE(s.Qty, 0) < COALESCE(s.
           ReorderLevel, 10)
        ORDER BY h.HID, m.Name
        """)
    try:
        result = db.session.execute(query).fetchall()
        res = [
            {
                "HID": row[0],
                "HospitalName": row[1],
                "MID" : row[2],
                "Medication Name": row[3],
                "Quantity" : row[4],
                "Reorder Level" : row[5]
            }
            for row in result
        ]
        return res
    except Exception as e:
        db.session.rollback()
        raise Exception(f"Operation Failed: {e}")
```

Low Stock Alert Interface


Low Stock Alert

Items below their reorder level, organized by hospital

MEDICATION	CURRENT QUANTITY	REORDER LEVEL	ACTION REQUIRED

✓

All stock levels are good!

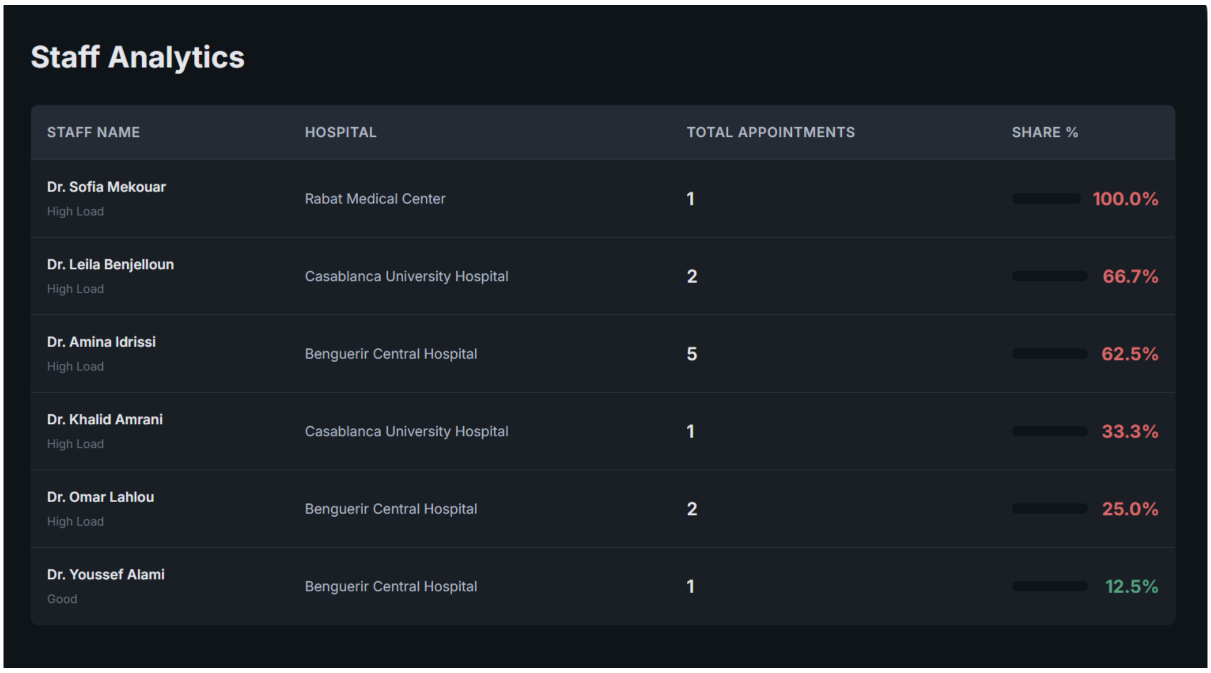
No items below reorder level

Figure 14: low stock alert

(e) **Command staff_share: Backend Implementation**:

```
    def get_staff_share():
    query = db.text("""
    WITH staff_hosp AS (
        SELECT
            S.STAFF_ID,
            S.FullName,
            d.HID,
            COUNT(*) AS n,
            h.Name as HName
        FROM Appointment a
        JOIN ClinicalActivity c ON c.CAID = a.CAID
        JOIN Department d ON d.DEP_ID = c.DEP_ID
        JOIN Staff S ON S.STAFF_ID = c.STAFF_ID
        JOIN Hospital h ON h.HID = d.HID
        GROUP BY S.STAFF_ID, d.HID, S.FullName, h.Name
    ),
    hosp_tot AS (
        SELECT d.HID, COUNT(*) AS total_appointments
        FROM Appointment a
        JOIN ClinicalActivity c ON c.CAID = a.CAID
        JOIN Department d ON d.DEP_ID = c.DEP_ID
        GROUP BY d.HID
    )
    SELECT
        sh.FullName,
        sh.HID,
        sh.n,
        sh.HName,
        ROUND(100.0 * sh.n / ht.total_appointments, 2) AS
            PctOfHospital
    FROM staff_hosp sh
    JOIN hosp_tot ht ON ht.HID = sh.HID
    ORDER BY PctOfHospital DESC
    """)
    try:
        result = db.session.execute(query).fetchall()
        res = [
            {
                "Staff FullName": row.FullName,
                "Hospital Name": row.HName,
                "Total Appointments": row.n,
                "Percentage Share within the Hospital":
                    row.PctOfHospital
            }
            for row in result
        ]

        return res
    except Exception as e:
        db.session.rollback()
        raise Exception(f"Operation Failed: {e}")
```

Staff Analytics Interface



Staff Analytics

STAFF NAME	HOSPITAL	TOTAL APPOINTMENTS	SHARE %
Dr. Sofia Mekouar High Load	Rabat Medical Center	1	100.0%
Dr. Leila Benjelloun High Load	Casablanca University Hospital	2	66.7%
Dr. Amina Idrissi High Load	Benguerir Central Hospital	5	62.5%
Dr. Khalid Amrani High Load	Casablanca University Hospital	1	33.3%
Dr. Omar Lahlou High Load	Benguerir Central Hospital	2	25.0%
Dr. Youssef Alami Good	Benguerir Central Hospital	1	12.5%

Figure 15: Staff Analytics

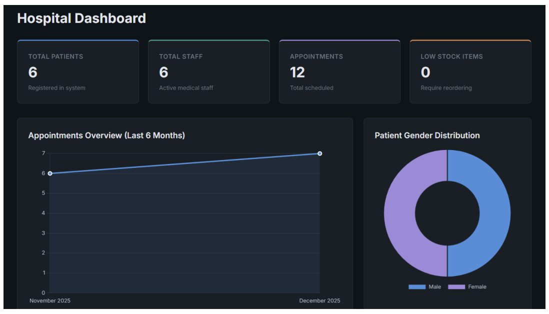
(f) Application Setup and Running

```
#requirements.txt
blinker==1.9.0
cffi==2.0.0
click==8.3.1
cryptography==46.0.3
dotenv==0.9.9
Flask==3.1.2
Flask-SQLAlchemy==3.1.1
greenlet==3.2.4
itsdangerous==2.2.0
Jinja2==3.1.6
MarkupSafe==3.0.3
pycparser==2.23
PyMySQL==1.1.2
python-dotenv==1.2.1
SQLAlchemy==2.0.44
typing_extensions==4.15.0
Werkzeug==3.1.3

# .env file (example)
# MySQL Database Configuration
MYSQL_HOST=localhost
MYSQL_PORT=3306
MYSQL_DB=lab6
MYSQL_USER=root
MYSQL_PASSWORD=root

# Flask Configuration
```

```
FLASK_ENV=development
FLASK_DEBUG=True
SECRET_KEY=your-secret-key-here
```

```
run.bat

============================================================
    MNHS Hospital Management System - Starting...
============================================================

Activating virtual environment...
Starting Flask application...
Press Ctrl+C to stop the server

 * Serving Flask app 'main'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a
     production deployment. Use a production WSGI server
    instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
```

Main application dashboard

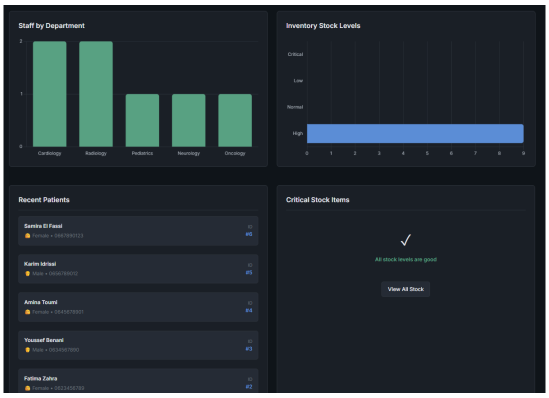


Hospital Dashboard

TOTAL PATIENTS	TOTAL STAFF	APPOINTMENTS	LOW STOCK ITEMS
6	6	12	0
Registered in system	Active medical staff	Total scheduled	Require reordering

Appointments Overview (Last 6 Months)
7
6
5
4
3
2
1
0
November 2025
December 2025
Patient Gender Distribution
Male
Female

Figure 16: application dashboard



Recent Patients

Samira El Fassi — Female • 0667890123 — ID #6

Karim Idrissi — Male • 0656789012 — ID #5

Amina Toumi — Female • 0645678901 — ID #4

Youssef Benani — Male • 0634567890 — ID #3

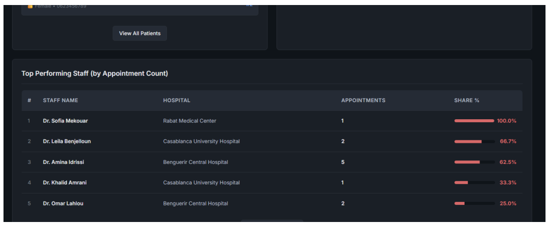
Fatima Zahra — Female • 0623456789 — ID #2

Critical Stock Items

✓

All stock levels are good

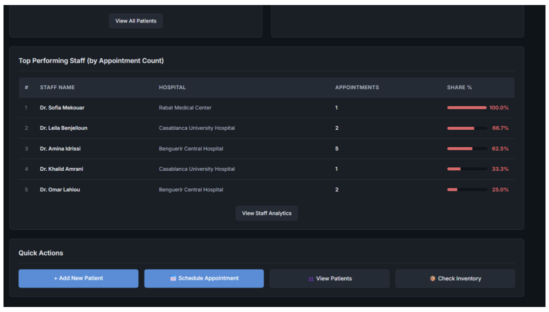
Figure 17: application dashboard



Top Performing Staff (by Appointment Count)

#	STAFF NAME	HOSPITAL	APPOINTMENTS	SHARE %
1	Dr. Sofia Mekouar	Rabat Medical Center	1	100.0%
2	Dr. Leila Benjelloun	Casablanca University Hospital	2	66.7%
3	Dr. Amina Idrissi	Benguerir Central Hospital	5	62.5%
4	Dr. Khalid Amrani	Casablanca University Hospital	1	33.3%
5	Dr. Omar Lahlou	Benguerir Central Hospital	2	25.0%

Figure 18: application dashboard



Top Performing Staff (by Appointment Count)

#	STAFF NAME	HOSPITAL	APPOINTMENTS	SHARE %
1	Dr. Sofia Mekouar	Rabat Medical Center	1	100.0%
2	Dr. Leila Benjelloun	Casablanca University Hospital	2	66.7%
3	Dr. Amina Idrissi	Benguerir Central Hospital	5	62.5%
4	Dr. Khalid Amrani	Casablanca University Hospital	1	33.3%
5	Dr. Omar Lahlou	Benguerir Central Hospital	2	25.0%

Figure 19: application dashboard

5 Discussion

5.1 Technical Challenges

Complex View Implementation: PatientNextVisit required sophisticated subqueries and multiple joins to find the next scheduled appointment for each patient
Trigger Logic: Expense recalculation triggers involved complex navigation through multiple table relationships (Includes → Prescription → ClinicalActivity → Expense → Stock)
Application-Database Integration: Ensuring Flask properly handled database transactions and error scenarios

5.2 Performance Observations

Views significantly improved query performance by pre-computing complex joins
Triggers added overhead but ensured data consistency
The web application provided much faster data access compared to manual SQL queries

5.3 Lessons Learned

Database views are powerful for simplifying application code
Proper trigger design is crucial for maintaining data integrity
Web applications make database interactions more accessible to non-technical users
Environment variables and proper configuration management are essential for security

6 Conclusion

This deliverable successfully enhanced the MNHS database system through three key components: optimized SQL views for improved query performance, robust triggers for business rule enforcement, and an intuitive web application for user-friendly data access. The implementation demonstrates how database features and application development work together to create a comprehensive healthcare management solution that is both technically sound and practically useful for medical staff.

The project met all specified requirements while providing a solid foundation for future enhancements to the Moroccan National Health Services system.