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 Name	DATA BEAKERS
Member 1 Name	ASMAE BOUYA
Member 2 Name	HIBA BOUSSALAA
Member 3 Name	SALMA BAGHAZI
Member 4 Name	ABDELILAH BAAMOUDI
Member 5 Name	ZAINA AITHMADOUBRAHIM

University
Mohammed VI
Polytechnic

1 Introduction

The goal of this deliverable is to enhance the MNHS database system by improving query efficiency, enforcing data consistency, and enabling real-world interaction through an application layer. This work focuses on three main components. First, we design SQL views that simplify complex queries, reduce repetitive joins, and improve usability for developers and analysts. Second, we implement robust SQL triggers that ensure data integrity by preventing invalid operations such as double bookings, negative stock, and inconsistent deletions. Finally, we extend the database with a small backend application capable of performing key MNHS operations, including listing patients, scheduling appointments, analyzing stock levels, and computing staff workload. This deliverable demonstrates how database logic, integrity constraints, and application development can work together to create a reliable, efficient, and secure medical information system.

2 Requirements

This deliverable covers the following tasks:

- Build the **UpcomingByHospital** SQL view summarizing scheduled appointments for the next 14 days.
- Build the **DrugPricingSummary** view for medication pricing per hospital.
- Build the **StaffWorkloadThirty** view summarizing staff activity over the last 30 days.
- Build the **PatientNextVisit** view identifying each patient's next scheduled appointment.
- Implement triggers to:
  - Prevent double booking of staff.
  - Recompute **Expense.Total** when prescription lines change.
  - Block invalid stock updates and enforce consistency rules.
  - Protect patient deletion when linked activities exist.
- Implement a backend application capable of:
  - Listing patients.
  - Scheduling appointments in one transaction.
  - Listing low-stock medications.
  - Computing staff appointment share per hospital.

3 Implementation & Results

UpcomingByHospital View

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

```sql
SELECT HospitalName, ApptDate, ScheduledCount
FROM UpcomingByHospital
WHERE ApptDate = '2025-01-15'
ORDER BY ScheduledCount DESC;
```

This view simplifies application code by encapsulating a complex join across 4 tables (Appointment → ClinicalActivity → Department → Hospital). It filters appointments scheduled for the next 14 days and groups them by hospital and date, providing a clean summary of daily scheduled appointments per hospital.

DrugPricingSummary View

```sql
CREATE VIEW DrugPricingSummary AS
SELECT H.HID AS HID,
                H.Name AS HospitalName,
        M.MID AS MID,
        M.Name AS MedicationName,
                AVG(S.UnitPrice) AS AvgUnitPrice,
        MIN(S.UnitPrice) AS MinUnitPrice,
        MAX(S.UnitPrice) AS MaxUnitPrice,
        MAX(S.StockTimestamp) AS LastStockTimestamp
FROM Hospital H
JOIN Stock S
        ON H.HID = S.HID
JOIN Medication M
        ON M.MID = S.MID
GROUP BY H.HID, H.Name, M.MID , M.Name;
```

```sql
-- Find the most expensive medications on average
SELECT MedicationName, HospitalName, AvgUnitPrice
FROM DrugPricingSummary
ORDER BY AvgUnitPrice DESC
LIMIT 10;
```

This view simplifies medication pricing analysis by aggregating stock data across hospitals and medications. It provides a comprehensive summary of pricing statistics

(average, min, max) and the latest stock update timestamp for each medication at each hospital. Applications can use this view for price comparisons, inventory management, and reporting without writing complex aggregation queries.

StaffWorkloadThirty View

```
CREATE VIEW StaffWorkloadThirty AS
SELECT S.STAFF_ID AS STAFF_ID,
                S.FullName AS FullName,
        COUNT(A.CAID) AS TotalAppointments,
        COUNT(CASE WHEN A.Status = 'Scheduled' THEN 1 END) AS
           ScheduledCount,
                COUNT(CASE WHEN A.Status = 'Completed' THEN 1 END
                   ) AS CompletedCount,
                COUNT(CASE WHEN A.Status = 'Cancelled' THEN 1 END
                   ) AS CancelledCount
FROM Staff S
LEFT JOIN ClinicalActivity C
        ON S.STAFF_ID = C.STAFF_ID
LEFT JOIN Appointment A
        ON A.CAID = C.CAID
    AND C.Date BETWEEN DATE_SUB(CURDATE(), INTERVAL 30 DAY) AND
       CURDATE()
GROUP BY S.STAFF_ID, S.FullName;
```

```
-- Find the busiest staff members
SELECT StaffName, TotalAppointments, CompletedCount
FROM StaffWorkloadThirty
ORDER BY TotalAppointments DESC
LIMIT 10;
```

This view simplifies workload analysis by providing a 30-day summary of each staff member's appointment activities. It counts total appointments and breaks them down by status (scheduled, completed, cancelled) using conditional aggregation.

PatientNextVisit View

```
CREATE VIEW PatientNextVisit AS
SELECT p.IID,p.FullName AS FullName,d.Name AS DepartmentName,h.
   Name AS Appointment,
h.City,MIN(c.Date) AS NextAppDate
FROM ClinicalActivity c
JOIN patient p ON p.IID=c.IID
JOIN Appointment a ON a.CAID=c.CAID
JOIN Deppartment d ON d.DEP_ID=c.DEO_ID
JOIN Hospital h ON h.HID=d.HID

WHERE c.Date>CURRENT_DATE() AND a.Status='Scheduled'
GROUP BY p.IID,p.FullName,d.Name,h.Name,h.City;
```

```sql
-- Find which departments have the most upcoming appointments
SELECT DepartmentName, COUNT(*) as NumberOfAppointments
FROM PatientNextVisit
GROUP BY DepartmentName
ORDER BY NumberOfAppointments DESC;
```

This view simplifies patient appointment tracking by identifying the next scheduled visit for each patient across the healthcare network. It finds the earliest future appointment date by joining through five tables (Patient → ClinicalActivity → Appointment → Department → Hospital) and filtering for scheduled status, providing medical staff with immediate access to upcoming patient visits including department, hospital, and location details without complex date-minimization queries.

4 Triggers

Double Booking Trigger

```sql
DELIMITER //

CREATE TRIGGER Double_booking_insert
BEFORE INSERT ON Appointment
FOR EACH ROW
BEGIN
        DECLARE v_staff_id INT;
    DECLARE v_date DATE;
    DECLARE v_time TIME;
        DECLARE booking_count INT DEFAULT 0;

    SELECT STAFF_ID ,Date ,Time
    INTO v_staff_id, v_date, v_time
    FROM ClinicalActivity
    WHERE CAID = NEW.CAID;

    SELECT COUNT(*)
    INTO booking_count
    FROM Appointment A
    JOIN ClinicalActivity C ON C.CAID=A.CAID
    WHERE C.Date = v_date
                AND C.Time= v_time
        AND C.STAFF_ID=v_staff_id
        AND A.CAID <> NEW.CAID;

    IF booking_count > 0 THEN
                SIGNAL SQLSTATE '45000'
                SET MESSAGE_TEXT = 'Double booking detected
                   during insert.';
    END IF;
END//
DELIMITER ;
```

```
DELIMITER //

CREATE TRIGGER Double_booking_update
BEFORE UPDATE  ON Appointment
FOR EACH ROW
BEGIN
        DECLARE v_staff_id INT;
    DECLARE v_date DATE;
    DECLARE v_time TIME;
        DECLARE booking_count INT DEFAULT 0;

    SELECT STAFF_ID ,Date ,Time
    INTO v_staff_id, v_date, v_time
    FROM ClinicalActivity
    WHERE CAID = NEW.CAID;

    SELECT COUNT(*)
    INTO booking_count
    FROM Appointment A
    JOIN ClinicalActivity C ON C.CAID=A.CAID
    WHERE C.Date = v_date
                AND C.Time= v_time
        AND C.STAFF_ID=v_staff_id
        AND A.CAID <> NEW.CAID;

    IF booking_count > 0 THEN
                SIGNAL SQLSTATE '45000'
                                SET MESSAGE_TEXT = 'Double
                                   booking detected during update
                                   .';
    END IF;
END//
DELIMITER ;
```

```
-- Create first appointment for a staff member
INSERT INTO ClinicalActivity (CAID, IID, STAFF_ID, DEP_ID, Date,
   Time)
VALUES (9991, 1001, 4001, 3001, '2025-12-01', '09:00:00');
INSERT INTO Appointment (CAID, Reason, Status) VALUES (9991, '
   Checkup', 'Scheduled');

-- Try to create second appointment for same staff at same time
INSERT INTO ClinicalActivity (CAID, IID, STAFF_ID, DEP_ID, Date,
   Time)
VALUES (9992, 1002, 4001, 3001, '2025-12-01', '09:00:00');
INSERT INTO Appointment (CAID, Reason, Status) VALUES (9992, '
   Follow-up', 'Scheduled');
-- error : double booking detected during insert
```

This trigger prevents staff members from being double-booked by checking for existing appointments at the same date and time before allowing new appointments to be scheduled. It ensures that no two clinical activities can be created for the same staff member with overlapping schedules, maintaining efficient staff allocation and preventing scheduling conflicts across the healthcare system.

Expense Update Trigger

```
DELIMITER //
CREATE TRIGGER AfterInsertIncludes
AFTER INSERT ON Includes
FOR EACH ROW
BEGIN
        DECLARE total_amount DECIMAL(10,2);
    DECLARE hospital_id INT;
    DECLARE clinical_activity_id INT;

    -- select the CAID for this prescription
    SELECT P.CAID INTO clinical_activity_id
    FROM Prescription P
    WHERE P.PID=NEW.PID;

    -- select HID for the clinical activity
    SELECT H.HID INTO hospital_id
    FROM Prescription P
    JOIN ClinicalActivity C ON P.CAID=C.CAID
    JOIN Department D ON C.DEP_ID = D.DEP_ID
    JOIN Hospital H ON D.HID = H.HID
    WHERE P.PID = NEW.PID ;

    -- calculate the new total for the expense
    SELECT SUM(S.UnitPrice) INTO total_amount
    FROM Includes I
    JOIN Prescription P ON P.PID = I.PID
    JOIN ClinicalActivity C ON P.CAID = C.CAID
    JOIN Department D ON C.DEP_ID = D.DEP_ID
    JOIN Hospital H ON D.HID = H.HID
    JOIN Stock S ON S.HID = H.HID AND S.MID = I.MID
    WHERE P.PID = NEW.PID;

    -- check if any medication price is missing
    IF total_amount IS NULL THEN
                SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Cannot update expense total: missing
           unit price for one or more medications';
        ELSE
                -- update the expense total
        UPDATE Expense E
        JOIN Prescription P ON E.CAID = P.CAID
        SET E.Total = total_amount
```

```
        WHERE P.PID = NEW.PID ;
        END IF ;
END//
DELIMITER ;

DELIMITER //
CREATE TRIGGER AfterUpdateIncludes
AFTER UPDATE ON Includes
FOR EACH ROW
BEGIN
        DECLARE total_amount DECIMAL(10,2);
    DECLARE hospital_id INT;
    DECLARE clinical_activity_id INT;

    -- select the CAID for this prescription
    SELECT P.CAID INTO clinical_activity_id
    FROM Prescription P
    WHERE P.PID=NEW.PID;

    -- select HID for the clinical activity
    SELECT H.HID INTO hospital_id
    FROM Prescription P
    JOIN ClinicalActivity C ON P.CAID=C.CAID
    JOIN Department D ON C.DEP_ID = D.DEP_ID
    JOIN Hospital H ON D.HID = H.HID
    WHERE P.PID = NEW.PID ;

    -- calculate the new total for the expense
    SELECT SUM(S.UnitPrice) INTO total_amount
    FROM Includes I
    JOIN Prescription P ON P.PID = I.PID
    JOIN ClinicalActivity C ON P.CAID = C.CAID
    JOIN Department D ON C.DEP_ID = D.DEP_ID
    JOIN Hospital H ON D.HID = H.HID
    JOIN Stock S ON S.HID = H.HID AND S.MID = I.MID
    WHERE P.PID = NEW.PID;

    -- check if any medication price is missing
    IF total_amount IS NULL THEN
                SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Cannot update expense total: missing
           unit price for one or more medications';
        ELSE
                -- update the expense total
        UPDATE Expense E
        JOIN Prescription P ON E.CAID = P.CAID
        SET E.Total = total_amount
        WHERE P.PID = NEW.PID ;
        END IF ;
END//
DELIMITER ;
```

```
DELIMITER //
CREATE TRIGGER AfterDeleteIncludes
AFTER DELETE ON Includes
FOR EACH ROW
BEGIN
        DECLARE total_amount DECIMAL(10,2);
    DECLARE hospital_id INT;
    DECLARE clinical_activity_id INT;

    -- select the CAID for this prescription
    SELECT P.CAID INTO clinical_activity_id
    FROM Prescription P
    WHERE P.PID=OLD.PID;

    -- select HID for the clinical activity
    SELECT H.HID INTO hospital_id
    FROM Prescription P
    JOIN ClinicalActivity C ON P.CAID=C.CAID
    JOIN Department D ON C.DEP_ID = D.DEP_ID
    JOIN Hospital H ON D.HID = H.HID
    WHERE P.PID = OLD.PID ;

    -- calculate the new total for the expense
    SELECT SUM(S.UnitPrice) INTO total_amount
    FROM Includes I
    JOIN Prescription P ON P.PID = I.PID
    JOIN ClinicalActivity C ON P.CAID = C.CAID
    JOIN Department D ON C.DEP_ID = D.DEP_ID
    JOIN Hospital H ON D.HID = H.HID
    JOIN Stock S ON S.HID = H.HID AND S.MID = I.MID
    WHERE P.PID = OLD.PID;

    -- check if any medication price is missing
    IF total_amount IS NULL THEN
        SET total_amount = 0;
        END IF;
        -- update the expense total
        UPDATE Expense E
        JOIN Prescription P ON E.CAID = P.CAID
        SET E.Total = total_amount
        WHERE P.PID = OLD.PID ;
END//
DELIMITER ;
```

```
-- Create new clinical activity and prescription
INSERT INTO ClinicalActivity (CAID, IID, STAFF_ID, DEP_ID, Date,
   Time)
VALUES (9993, 1005, 4002, 3010, '2025-12-02', '10:00:00');
INSERT INTO Appointment (CAID, Reason, Status) VALUES (9993, '
   Consultation', 'Completed');
```

```
INSERT INTO Prescription (PID, CAID, DateIssued) VALUES (9991,
   9993, '2025-12-02');
INSERT INTO Expense (ExpID, InsID, CAID, Total) VALUES (9991,
   6001, 9993, 0.00);

-- Add medication to prescription (triggers expense update)
INSERT INTO Includes (PID, MID, Dosage, Duration) VALUES (9991,
   8001, '1 capsule daily', '7 days');
-- error : Cannot update expense total: missing unit price for
   one or more medications
```

These triggers automatically maintain financial accuracy by recalculating prescription totals whenever medication lines are added, modified, or removed. They navigate through the complex relationship chain (Includes → Prescription → ClinicalActivity → Expense) and update the corresponding expense total using current medication prices, ensuring that patient billing always reflects the actual prescribed treatments and their current costs.

Stock Integrity Triggers

```
DELIMITER //
CREATE TRIGGER ConsistentStock_before_insert
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 = 'The Qty must be positive and
                   the UnitPrice and ReorderLevel must be
                   strictly positive';
         END IF;
END//
DELIMITER ;

DELIMITER //
CREATE TRIGGER ConsistentStock_before_update
BEFORE UPDATE ON Stock
FOR EACH ROW
BEGIN
     IF (NEW.Qty<OLD.Qty AND NEW.Qty<0) OR  NEW.UnitPrice <= 0 OR
         NEW.ReorderLevel < 0 OR NEW.Qty<0 THEN
                SIGNAL SQLSTATE '45000'
                SET MESSAGE_TEXT = 'The Qty must be positive
                   after decreasing it and the UnitPrice and
                   ReorderLevel must be strictly positive';
         END IF;
END//
DELIMITER ;
```

```
-- Try to insert negative stock quantity
INSERT INTO Stock (HID, MID, UnitPrice, Qty, ReorderLevel)
VALUES (2001, 8001, 150.00, -10, 5);
-- error : The Qty must be positive and the UnitPrice and
   ReorderLevel must be strictly positive

-- Try to insert zero price
INSERT INTO Stock (HID, MID, UnitPrice, Qty, ReorderLevel)
VALUES (2001, 8004, 0.00, 50, 10);
-- error: The Qty must be positive and the UnitPrice and
   ReorderLevel must be strictly positive

-- Valid stock entry
INSERT INTO Stock (HID, MID, UnitPrice, Qty, ReorderLevel)
VALUES (2004, 8006, 120.00, 25, 5);
```

These triggers enforce inventory integrity by validating all stock entries and updates against business rules before they are committed to the database. They prevent negative quantities, zero or negative pricing, and invalid reorder levels, while also ensuring that stock reductions cannot result in negative inventory, maintaining reliable medication availability and accurate pharmacy management.

Patient Deletion Trigger

```
DELIMITER //
CREATE TRIGGER check_patient_delete
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 with
                   existing clinical activities. Reassign or
                   delete activities first.';
        END IF;
END//
DELIMITER ;
```

```
-- Create a patient with no activities
INSERT INTO Patient (IID, CIN, FullName, Birth, Sex, BloodGroup,
   Phone)
VALUES (9999, 'TEST999', 'Test Patient', '1990-01-01', 'M', 'O+',
    '0600000000');

-- Delete patient with NO activities
DELETE FROM Patient WHERE IID = 9999;
```

```
-- Try to delete patient WITH activities (using existing patient)
DELETE FROM Patient WHERE IID = 1001;
-- error: Cannot delete patient with existing clinical activities
   . Reassign or delete activities first.
```

TThis trigger safeguards patient data integrity by blocking deletion attempts for patients who have existing clinical activities in the system. It prevents accidental loss of important medical history and ensures that all patient records remain complete for ongoing care, requiring administrators to first reassign or remove dependent clinical activities before patient deletion can proceed.

5 Application

The MNHS web application is a comprehensive hospital management dashboard that enables medical staff to efficiently coordinate patient care, track medication inventory, and monitor staff workload distribution. Built with a clean web interface, the system provides real-time access to critical healthcare operations including patient records management, appointment scheduling across departments, automated low-stock medication alerts, and detailed staff performance analytics - all seamlessly integrated with a MySQL backend database.

6 Conclusion

This deliverable successfully demonstrates the integration of database views, triggers, and application development to create a robust hospital management system. The implemented SQL views significantly improve query efficiency by encapsulating complex joins across multiple tables, providing healthcare staff with immediate access to critical information including upcoming appointments, medication pricing, staff workload, and patient schedules. The database triggers enforce essential business rules at the data layer, preventing double bookings, maintaining financial accuracy through automatic expense recalculations, ensuring inventory integrity, and protecting patient data from accidental deletion. The web application layer provides an intuitive interface that bridges the gap between the database and end-users, enabling seamless interaction with the MNHS system through patient management, appointment scheduling, inventory monitoring, and staff analytics modules. Together, these components create a comprehensive healthcare management solution that ensures data consistency, operational efficiency, and user-friendly access to critical medical information.

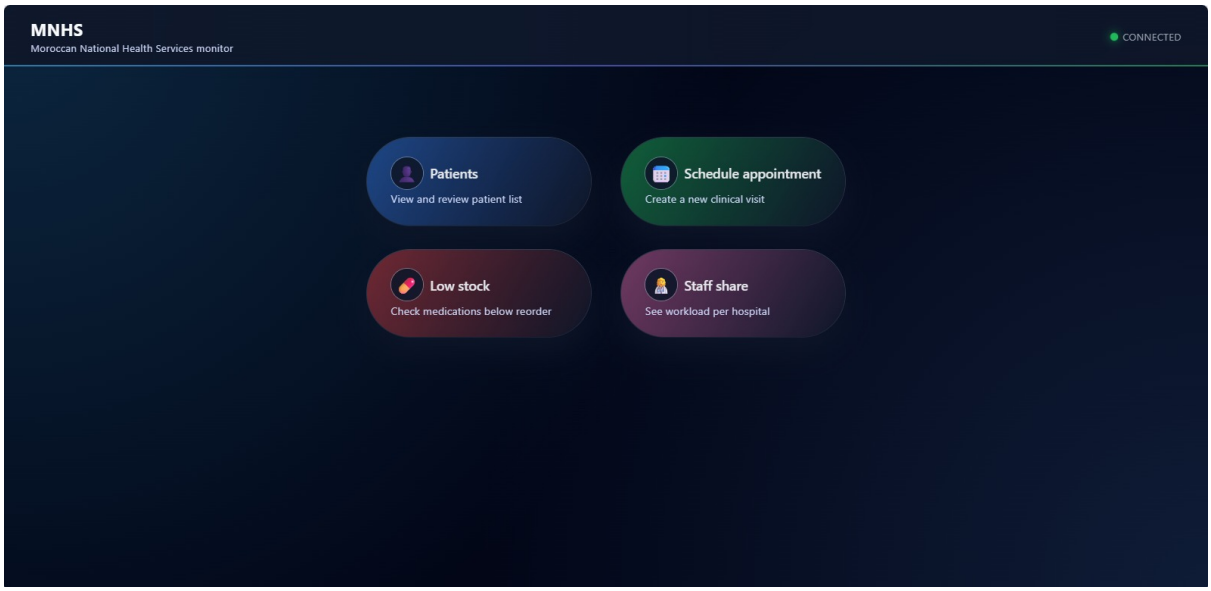

Figure 1: MNHS Dashboard - Main navigation interface showing the four core modules of the hospital management system

Figure 2: Appointment Scheduling Form - Interface for booking new clinical visits with patient, staff, and timing details

Figure 3: Appointment Confirmation - Success message displayed after scheduling a new clinical activity

	CAID	Reason	Status
	7000	apah	Scheduled
	7002	flu symptoms	Scheduled
	7003	routine follow-up	Scheduled
	9000	Test appointment	Scheduled
	NULL	NULL	NULL

Figure 4: Scheduled Appointments List - View showing all currently scheduled appointments with their status and reasons

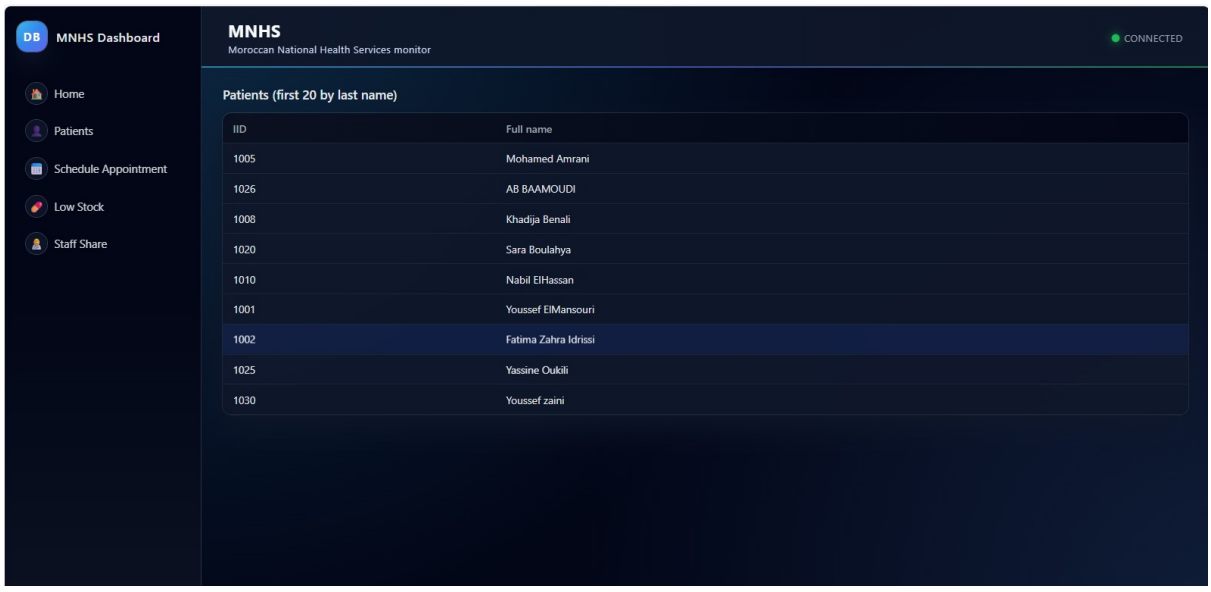

Figure 5: Patient Directory - Alphabetical listing of patients with IDs and full names for medical staff reference

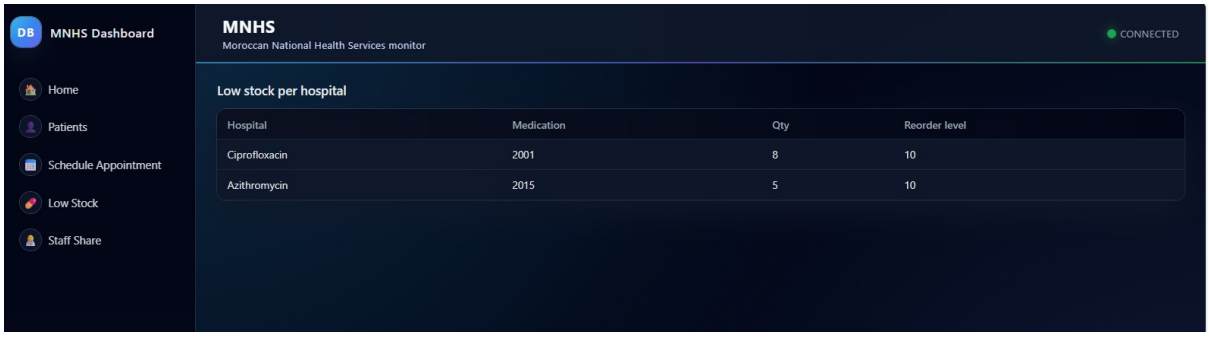

Figure 6: Low Stock Alert Dashboard - Medication inventory monitoring showing items below reorder levels across hospitals

Figure 7: Staff Workload Analytics - Percentage distribution of appointments showing workload balance across hospital network