Deliverable : Moroccan National Health Services Lab 6

Data Management Course

UM6P College of Computing

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

Team Information

Team Name	....
Member 1	Achraf Tata
Member 2	Taha Tahiri
Member 3	Noura Riahi El Idrissi
Member 4	Mohamadou Thiam Taha
Member 5	Adam Yassine
Repository Link	`https://github.com/...`

---

1 Introduction

This deliverable implements the MNHS lab part VI: the required triggers, views and a small web app to exercise them. It enforces business rules (no double booking, stock checks, expense recompute, protect patient deletes) and provides simple endpoints/commands for patients, appointments and inventory. The work follows the lab specification.

2 Requirements

Create four views (upcoming by hospital, drug pricing summary, staff workload 30d, patient next visit).

Implement four groups of triggers (double-booking, expense recompute, stock validation, patient-delete protection).

Build a web app (Flask) exposing endpoints for: list patients, schedule appointment, low stock, staff share, totals.

Provide a CLI wrapper with same commands for quick testing.

3 Methodology

We kept things simple and pragmatic: enforce invariant logic in DB triggers so app stays thin; use BEFORE triggers for input validation and AFTER triggers for recomputing dependent totals. App layer uses transactions for multi-statement ops (create ClinicalActivity + Appointment) and parameterized queries to avoid injection. Assumed input dates/times come ISO (YYYY-MM-DD / HH:MM:SS) and CAID/IID are provided by caller. Flask for HTTP, small argparse CLI for dev testing.

4 Implementation & Results

Include:

Triggers

```
CREATE TRIGGER OVERLAP_insert
BEFORE INSERT ON Appointment
FOR EACH ROW
BEGIN
    IF EXISTS (
        SELECT 1
        FROM ClinicalActivity C
        JOIN Appointment A ON A.CAID = C.CAID
        WHERE C.Date = (SELECT Date FROM ClinicalActivity WHERE CAID = NEW.CAID)
          AND C.Time = (SELECT Time FROM ClinicalActivity WHERE CAID = NEW.CAID)
    ) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'APPOINTMENT-OVERLAP';
    END IF;
END


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

```
BEGIN
    IF EXISTS (
        SELECT 1
        FROM ClinicalActivity C
        JOIN Appointment A ON A.CAID = C.CAID
        WHERE C.Date = (SELECT Date FROM ClinicalActivity WHERE CAID = NEW.CAID)
          AND C.Time = (SELECT Time FROM ClinicalActivity WHERE CAID = NEW.CAID)
          AND A.CAID <> NEW.CAID
    ) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'APPOINTMENT OVERLAP';
    END IF;
END



CREATE TRIGGER RECOMPUTE_insert
AFTER INSERT ON Includes
FOR EACH ROW
BEGIN
    IF EXISTS (
        SELECT 1 FROM Stock S
        WHERE S.UnitPrice IS NULL
    ) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'ERROR: Null unit price in Stock';
    END IF;
    UPDATE Expense E
    SET E.Total = (
        SELECT SUM(S.UnitPrice)
        FROM Includes I
        JOIN Medication M ON M.MID = I.MID
        JOIN Prescription P ON P.PID = I.PID
        JOIN Stock S ON S.MID = M.MID
        WHERE I.PID = NEW.PID
    )
    WHERE E.PID = NEW.PID;
END


CREATE TRIGGER RECOMPUTE_update
AFTER UPDATE ON Includes
FOR EACH ROW
BEGIN
    IF EXISTS (
        SELECT 1 FROM Stock S
        WHERE S.UnitPrice IS NULL
    ) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'ERROR: Null unit price in Stock';
    END IF;

    UPDATE Expense E
    SET E.Total = (
        SELECT SUM(S.UnitPrice)
        FROM Includes I
        JOIN Medication M ON M.MID = I.MID
        JOIN Prescription P ON P.PID = I.PID
        JOIN Stock S ON S.MID = M.MID
        WHERE I.PID = NEW.PID
    )
    WHERE E.PID = NEW.PID;
END
```

```
CREATE TRIGGER RECOMPUTE_delete
AFTER DELETE ON Includes
FOR EACH ROW
BEGIN
    IF EXISTS (
        SELECT 1 FROM Stock S
        WHERE S.UnitPrice IS NULL
    ) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'ERROR: Null unit price in Stock';
    END IF;

    UPDATE Expense E
    SET E.Total = (
        SELECT SUM(S.UnitPrice)
        FROM Includes I
        JOIN Medication M ON M.MID = I.MID
        JOIN Prescription P ON P.PID = I.PID
        JOIN Stock S ON S.MID = M.MID
        WHERE I.PID = OLD.PID
    )
    WHERE E.PID = OLD.PID;
END



 Prevent negative or inconsistent stock.

CREATE TRIGGER BFR_INSERT_Stock
BEFORE INSERT ON stock
FOR EACH ROW

BEGIN
 IF NEW.qty<0 THEN SIGNAL SQLSTATE '45000'
 set MESSAGE_TEXT='Invalid insert. Qty can not be negative';
END if;

BEGIN
 IF NEW.UnitPrice<=0 THEN SIGNAL SQLSTATE '45000'
 set MESSAGE_TEXT='Invalid Insert. UnitPrice can not be negative;
END if;

BEGIN
 IF NEW.RorderLevel<0 THEN SIGNAL SQLSTATE '45000'
 set MESSAGE_TEXT='Invalid Insert. ReorderLevel can not be negative;
END if;
END ;



CREATE TRIGGER BFR_UPDATE_Stock
BEFORE UPDATE ON stock
FOR EACH ROW

BEGIN
 IF NEW.qty<0 THEN SIGNAL SQLSTATE '45000'
 set MESSAGE_TEXT='Invalid Update. Qty can not be negative';
END if;

BEGIN
 IF NEW.UnitPrice<=0 THEN SIGNAL SQLSTATE '45000'
 set MESSAGE_TEXT='Invalid Update. UnitPrice can not be negative;
```

```
END if;

BEGIN
IF NEW.RorderLevel<0 THEN SIGNAL SQLSTATE '45000'
set MESSAGE_TEXT='Invalid Update. ReorderLevel can not be negative;
END if;


BEGIN
IF NEW.Qty<OLD.QTY AND NEW.Qty<0 THEN SIGNAL SQLSTATE '45000'
set MESSAGE_TEXT='Invalid UPDATE. Qty can not be below zero;
END if;

END ;



 Protect referential integrity on patient delete

CREATE TRIGGER trig_patientw
BEFORE DELETE ON patient
FOR EACH ROW

Begin

IF EXISTS (
     SELECT 1
     FROM ClinicalActivity
     where PID=OLD.PID
)Then SIGNAL SQLSTATE '45000'
set Message_text='Deletion blocked:This patient has a clinicalActivity and he have to reassign
END IF ;

END;
```

Views

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

2-CREATE VIEW DrugPricingSummary AS
SELECT
     H.HID,
     H.Name AS HospitalName,
     M.MID AS MID,
     AVG(S.UnitPrice) AS AvgUnitPrice,
     MIN(S.UnitPrice) AS MinUnitPrice,
     MAX(S.UnitPrice) AS MaxUnitPrice,
     MIN(DATEDIFF(CURDATE(), S.StockTimestamp)) AS DaysSinceOldestStock
```

```
FROM Hospital H
JOIN Stock S ON H.HID = S.HID
JOIN Medication M ON M.MID = S.MID
GROUP BY
    H.HID, HospitalName, MID;

3-CREATE VIEW StaffWorkloadThirty AS
SELECT
    S.STAFF_ID,
    S.FullName AS FullName,

    SUM(CASE WHEN A.Status = 'Scheduled' THEN 1 ELSE 0 END) AS ScheduledCount,
    SUM(CASE WHEN A.Status = 'Completed' THEN 1 ELSE 0 END) AS CompletedCount,
    SUM(CASE WHEN A.Status = 'Canceled' THEN 1 ELSE 0 END) AS CanceledCount,

    COUNT(*) AS TotalAppointmentsLast30Days
FROM Staff S
JOIN ClinicalActivity C ON S.STAFF_ID = C.STAFF_ID
JOIN Appointment A ON A.CAID = C.CAID
WHERE
    C.Time BETWEEN DATE_SUB(CURDATE(), INTERVAL 30 DAY) AND CURDATE()
GROUP BY
    S.STAFF_ID, FullName;

4-CREATE VIEW PatientNextVisit AS
SELECT
    P.IID,
    P.FullName AS FullName,
    D.Name AS DepartmentName,
    H.Name AS HospitalName,
    H.City AS HospitalCity,
    MIN(C.Date) AS NextApptDate
FROM Patient P
JOIN ClinicalActivity C ON P.IID = C.IID
JOIN Appointment A ON A.CAID = C.CAID
JOIN Department D ON C.DEP_ID = D.DEP_ID
JOIN Hospital H ON D.HID = H.HID
WHERE
    A.Status = 'Scheduled'
GROUP BY
    P.IID, FullName, DepartmentName, HospitalName, HospitalCity
HAVING
    NextApptDate > CURDATE();
```

for more information on the web App see the section "About Us" in the web app menus

5 Discussion

We hit a few annoyances: writing safe recompute logic that joins stock by hospital required careful joins and blocking when unit price missing; preventing double-booking needed checking ClinicalActivity date/time per staff; testing triggers is fiddly because you must craft exact test rows and rollback often. Also small SQL typos (grouping/semicolon) cost debugging time. Lesson: put basic checks in the app but rely on DB triggers for final integrity. Keep tests small and repeatable.

6 Conclusion

All required triggers, views and a web app were implemented and tested locally. The DB enforces the main business rules, and the app provides the simple UI to use them. This is lightweight, easy to run, and good enough for the course deliverable.