University
Mohammed VI
Polytechnic

College of Computing

Moroccan National Health Services (MNHS)

Data Management Course

Mohammed VI Polytechnic University (UM6P)

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

Session: Fall 2025

Part VI: Views, Triggers, and Application development.

Preparatory Exercises

Before starting Project Part VI, students should review and solve the following textbook exercises.

Textbook Exercises

Reference: Ramakrishnan, Raghu, and Johannes Gehrke, *Database Management Systems*, 3rd Ed. [1].

- Chapter 5 (SQL: Triggers): Exercise 5.9
- Chapter 9 (SQL: Views): Exercise 3.19

Objectives

- Improve performance and usability of MNHS queries with views.
- Enforce business rules and data consistency using triggers.
- Develop a web application on top of the MNHS database (for example using Python, PHP, or J2EE).

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).

1. **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'`.

2. **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.

3. **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.

4. **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'`.

Triggers

1. **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.

2. **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`).

3. **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 that requires `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.

4. **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.

Example of a Python Application Layer with MySQL

Environment Setup

1. Create a dedicated database user with least privilege.

```
CREATE DATABASE IF NOT EXISTS lab3;
USE lab3;

CREATE USER 'mnhs_user'@'%' IDENTIFIED BY 'STRONG_PASSWORD';
GRANT SELECT, INSERT, UPDATE, DELETE ON lab3.* TO
   'mnhs_user'@'%';
FLUSH PRIVILEGES;
```

2. Install Python packages.

```
python -m venv .venv
source .venv/bin/activate
pip install mysql-connector-python SQLAlchemy pandas python-dotenv
```

3. Store secrets in an `.env` file, never in code.

```
MYSQL_HOST=127.0.0.1
MYSQL_PORT=3306
MYSQL_DB=lab3
MYSQL_USER=mnhs_user
MYSQL_PASSWORD=STRONG_PASSWORD
```

Direct connector (mysql_connector_python)

Minimal connection with context managers and parameterized query.

```
import os
from dotenv import load_dotenv
import mysql.connector
from mysql.connector import errorcode

load_dotenv()

cfg = 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 get_connection():
    return mysql.connector.connect(**cfg)

def list_patients_ordered_by_last_name(limit=20):
    sql = """
    SELECT IID, FullName
    FROM Patient
    ORDER BY SUBSTRING_INDEX(FullName, ' ', -1), FullName
    LIMIT %s
    """
    with get_connection() as cnx:
        with cnx.cursor(dictionary=True) as cur:
            cur.execute(sql, (limit,))
            return cur.fetchall()

def insert_patient(iid, cin, full_name, birth, sex, blood, phone):
    sql = """
    INSERT INTO Patient(IID, CIN, FullName, Birth, Sex, BloodGroup, Phone)
    VALUES (%s, %s, %s, %s, %s, %s, %s)
    """
```

```
    with get_connection() as cnx:
        try:
            with cnx.cursor() as cur:
                cur.execute(sql, (iid, cin, full_name, birth, sex, blood, phone))
            cnx.commit()
        except Exception:
            cnx.rollback()
            raise

if __name__ == "__main__":
    for row in list_patients_ordered_by_last_name():
        print(f"{row['IID']}  {row['FullName']}")
```

Transactions across multiple statements.

```
def schedule_appointment(caid, iid, staff_id, dep_id, date_str, time_str, reason):
    ins_ca = """
    INSERT INTO ClinicalActivity(CAID, IID, STAFF_ID, DEP_ID, Date, Time)
    VALUES (%s, %s, %s, %s, %s, %s)
    """
    ins_appt = """
    INSERT INTO Appointment(CAID, Reason, Status)
    VALUES (%s, %s, 'Scheduled')
    """
    with get_connection() as cnx:
        try:
            with cnx.cursor() as cur:
                cur.execute(ins_ca, (caid, iid, staff_id, dep_id, date_str, time_str))
                cur.execute(ins_appt, (caid, reason))
            cnx.commit()
        except Exception:
            cnx.rollback()
            raise
```

Pandas integration for quick analysis

```
import pandas as pd
from sqlalchemy import create_engine, text
import os
from dotenv import load_dotenv

load_dotenv()
url = (
    "mysql+mysqlconnector://"
    f"{os.getenv('MYSQL_USER')}:{os.getenv('MYSQL_PASSWORD')}"
    f"@{os.getenv('MYSQL_HOST')}:{os.getenv('MYSQL_PORT')}/{os.getenv('MYSQL_DB')}"
)
engine = create_engine(url, pool_pre_ping=True)

q = text("""
SELECT M.MID, M.Name AS Drug, H.City, AVG(S.UnitPrice) AS AvgPrice
FROM Stock S
```

```
JOIN Medication M ON M.MID = S.MID
JOIN Hospital H ON H.HID = S.HID
GROUP BY M.MID, M.Name, H.City
""")
df = pd.read_sql(q, engine)
print(df.head())
```

Guidelines for a robust application

- Use environment variables for credentials. Do not commit secrets.
- Use parameterized statements. Never build SQL by string concatenation.
- Prefer context managers for connection and cursor to ensure clean close and commit.
- Wrap multi statement workflows in one transaction with commit or rollback.
- Log executed actions and catch database errors with clear messages.
- Validate data in code and rely on database constraints for final protection.

Example Command Line Interface (CLI) wrapper

```
import argparse

def main():
    parser = argparse.ArgumentParser(description="MNHS CLI")
    sub = parser.add_subparsers(dest="cmd", required=True)

    sub.add_parser("list_patients")

    appt = sub.add_parser("schedule_appt")
    appt.add_argument("--caid", type=int, required=True)
    appt.add_argument("--iid", type=int, required=True)
    appt.add_argument("--staff", type=int, required=True)
    appt.add_argument("--dep", type=int, required=True)
    appt.add_argument("--date", required=True)   # YYYY-MM-DD
    appt.add_argument("--time", required=True)   # HH:MM:SS
    appt.add_argument("--reason", required=True)

    args = parser.parse_args()
    if args.cmd == "list_patients":
        for r in list_patients_ordered_by_last_name():
            print(f"{r['IID']}  {r['FullName']}")
    elif args.cmd == "schedule_appt":
        schedule_appointment(args.caid, args.iid, args.staff, args.dep,
                             args.date, args.time, args.reason)
        print("Appointment scheduled")

if __name__ == "__main__":
    main()
```

Mini tasks to implement

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 MySQL database.

1. Command `list_patients`: print the first twenty patients ordered by last name.
2. Command `schedule_appt`: create a clinical activity and a scheduled appointment in one transaction.
3. Command `low_stock`: list medications below `ReorderLevel` per hospital with a left join so that medications without stock also appear.
4. Command `staff_share`: for each staff member compute total number of appointments and percentage share within their hospital. Return a sorted table.

Deliverables

- SQL script containing all required views (`UpcomingByHospital`, `DrugPricingSummary`, `StaffWorkloadThirty`, `PatientNextVisit`) with at least one example query for each view.
- SQL script containing all required triggers (double booking protection, expense recomputation, stock checks, patient delete protection) with at least one example showing each trigger in action.
- Web application implementing the MNHS project on top of the MySQL database (for example using Python, PHP, or J2EE), including documentation on how to run it.

References

[1] Raghu Ramakrishnan, Johannes Gehrke, and Johannes Gehrke. *Database management systems*, volume 3. McGraw-Hill New York, 2003.