



University
Mohammed VI
Polytechnic



Deliverable #5: 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	Groupe2
Member 1	Abir Fakhreddine
Member 2	Malak El Assali
Member 3	Nada El Farissi
Member 4	Amine Chrif
Member 5	Anass Fertat
Member 6	Yasser Hallou
Repository Link	https://github.com/beaNoBeebea

1 Introduction

This deliverable presents a comprehensive healthcare management web application developed for the Moroccan National Health Services (MNHS), complemented with a robust implementation of triggers and views to maintain business logic rules. This system addresses real-world healthcare challenges such as appointment scheduling, inventory managements with automated low stock alerts, and more. By harnessing MySQL's advanced features such as triggers and views, we were able to simplify data access and enforce data integrity. Built as part of the Data Management course, this project demonstrates the team's core understanding of the course material, and concepts such as complex SQL triggers, and views, and database-driven application development. The system architecture integrates a MySQL backend with a Streamlit-based user interface, providing a highly simplified - and quite rudimentary for the moment - platform for healthcare professionals to manage daily operations efficiently.

2 Requirements

2.1 SQL Views

- **UpcomingByHospital:** Provides 14-day appointment forecasts per hospital
 - Helps staffing decisions based on predicted demand
 - Better resource allocation decisions
- **DrugPricingSummary:** Analyzes medication pricing across hospital network.
 - Helps homogenize drug pricing benchmarking across hospitals
 - Better tracking overall
- **StaffWorkloadThirty:** Tracks staff performance metrics over 30-day periods
 - Balancing workload across departments
 - Helps scheduling decisions
- **PatientNextVisit:** Manages patient appointment scheduling and tracking

- Filters only scheduled future appointments
- Uses correlated subquery to find earliest upcoming visit
- Provides complete hospital and department context

2.2 Database Triggers

- **Double Booking Prevention:** Ensures staff availability for appointments
 - Separate triggers for INSERT and UPDATE operations
 - Conditional execution in UPDATE to avoid unnecessary checks
 - Clear error messages
- **Expense Recalculation:** Automatically updates prescription costs
 - Real-time cost updates for accurate billing
 - Price validation before calculation
- **Stock Validation:** Maintains inventory data integrity
 - Prevents negative quantity errors
 - Maintains data consistency across operations
- **Patient Deletion Protection:** Preserves referential integrity
 - Prevents loss of medical history
 - Maintains referential integrity

2.3 Application Layer

- **CLI Interface:** Four operational commands for daily hospital tasks
- **Streamlit Web Interface:** AI-assisted web interface for better visualization and more seamless insertion of data
- **Database Connectivity:** Secure MySQL connection management
- **Transaction Management:** Atomic operations for critical workflows
- **Error Handling:** Exception management

3 Methodology

3.1 Application Design Approach

1. **Requirements Analysis:** of the functionalities needed for this section of the project.
2. **System Architecture Design:** Designed a three-tier strucutre separating database, backend, and frontend layers for clearer division of roles between teammates, and for more clarity for future scaling/maintenance of the app.
3. **Database Schema Optimization:** Evaluated and refined existing MNHS database relationships
4. **Business Logic Implementation:** Encoded cited rules at three levels:
 - **Database Level:** Implemented triggers for data integrity and business rule enforcement
 - **Backend Level:** Developed Python functions with transaction management and error handling - built on the database logic.
 - **Application Level:** Created validation/error message indicators in both CLI and web interfaces
5. **Integration & Testing:** Sample commands and queries for testing.

4 Implementation & Results

4.1 Database Views Implementation

4.1.1 PatientNextVisit View

This view solves the critical healthcare challenge of tracking patient follow-ups by identifying the next scheduled appointment for each patient.

- This view displays each patient's next scheduled visit with information about the patient, the department and the hospital.
- It does this by joining `Patient` → `ClinicalActivity` → `Appointment` and `Department` → `Hospital` and only keeping the rows where the appointment is actually sched-

uled and with a date greater than today's date (fetched with the SQL command CURDATE).

- The view contains a nested subquery to find the minimum amongst all dates that verify the previous conditions for a given patient.

4.1.2 UpcomingByHospital View

Essential for resource planning, this view provides a 14-day forecast of appointment volumes across the hospital network.

- In this view we have used `Appointment` joined through `ClinicalActivity` → `Department` → `Hospital`. The view only considers appointments with status 'Scheduled' between the current day and the following 14 days. For every hospital and date pair, it returns all hospital details, the clinical activity and the total number of scheduled appointments on that date.

4.1.3 DrugPricingSummary View

Critical for pharmaceutical management and cost control across the MNHS network.

- Without the `DrugPricingSummary` view, any time we need information about medication pricing, we would need to `JOIN` twice and group by `Hospital` and `Medication`. The view reduces these repeated joins and the grouping logic, making the queries shorter and reusable whenever needed.

4.1.4 StaffWorkloadThirty View

Supports workforce management by providing 30-day performance metrics for all staff members.

- The `StaffWorkloadThirty` view provides a statistical summary of appointments for every staff member, breaking down totals into 'Scheduled', 'Cancelled', and 'Completed' counts. It ensures comprehensive reporting by including staff with no current activity and automatically converting any `NULL` values into zeros for accurate analysis.

4.2 Business Logic Triggers

4.2.1 Double Booking Prevention

Ensures healthcare providers cannot be double-booked, maintaining service quality and schedule integrity.

- The two triggers `reject_double_booking_insert` and `reject_double_booking_update` are implemented to prevent double booking of a staff member for a clinical activity.
- The `INSERT` Trigger checks before inserting a new clinical activity if another clinical activity is already assigned to that staff member at the same date and time. If so, an error is raised using a `SIGNAL` to stop the execution of the statement.
- For the `UPDATE` Trigger it does a verification before any modification, which means that the only case where the trigger runs is when the date, time, or staff member is changed, without this verification the trigger will run even if we try to update unrelated fields. We note that in the update version we add the condition '`CAID != OLD.CAID`'. This condition tells the trigger to ignore the appointment that is currently being updated when checking for conflicts. For example, if we only want to change the time of a clinical activity, without this condition, the update would be blocked.
- Together, these two triggers ensure data consistency by ensuring that a staff member cannot perform two different clinical activities at the same time and date.

4.2.2 Expense Recalculation

Automatically maintains financial data integrity when prescription details change.

- The triggers automatically recalculate the total cost of a prescription anytime prescription lines are inserted, deleted, or updated so that it wouldn't have to be manually recomputed and risk inconsistencies or errors. They also prevent the change when a drug has no current price in `Stock`, ensuring that the total is not updated unless all pricing data is valid.

4.2.3 Stock Validation

Protects inventory data quality by enforcing business rules at the database level.

- The `check_stock_ins` and `check_stock_ups` triggers protect the `Stock` table by validating data before any insert or update occurs. They automatically block transactions and raise errors if a user attempts to save negative quantities, negative reorder levels, or non-positive unit prices, ensuring only valid inventory data enters the system.

4.2.4 Patient Deletion Protection

Preserves medical history integrity by preventing accidental patient record deletion.

- The `patient_delete` trigger is a BEFORE DELETE trigger, therefore, it is executed each time we try to delete a patient.
- It checks if the patient has any records in Clinical Activities.
- If so, it blocks the deletion and sends an error message that tells the user to make the necessary modifications to the clinical activities table
- This trigger protects referential integrity by preventing the deletion of patient records that have associated clinical activities.
- In other words, it serves to make sure that we do not find a clinical activity with a non-existing patient.

4.3 Results & Testing

The database triggers and views were rigorously and repeatedly tested to ensure proper functionality. Many test cases were executed, that verified:

- **Trigger Validation**
- **View Performance**
- **Data Consistency**

All tests passed successfully, demonstrating robust database performance. Detailed test queries and their outputs are documented in the `sql/example_queries` and `sql/testing` directories.

4.4 Python Application Layer

4.4.1 Database Connection Management

Secure and efficient database connectivity.

4.4.2 CLI Command Implementation

Comprehensive command set for daily hospital operations.

Operational Commands Analysis:

- **list_patients**: Simple patient list access
- **schedule_appt**: Safe appointment creation
- **low_stock**: Inventory analytics
- **staff_share**: Calculates percentage distributions using dictionaries

4.4.3 Main Application Interface

Robust command-line interface, as well as simplified web interface.

5 Discussion

5.1 Technical Challenges and Solutions

5.1.1 Extensive Data Validation & Testing

Challenge: Testing required massive data insertions to thoroughly test database views, triggers, and SQL commands, which was generated by AI. However, AI-generated test data needed rigorous validation to ensure accuracy, which proved to be maybe almost as time-consuming as if we had generated the data ourselves.

5.1.2 Limited SQL Capabilities

Challenge: Needed to implement sophisticated data integrity checks beyond basic constraints due to many fails of triggers.

Solution:

- Leveraged MySQL's procedural extensions using BEGIN/END blocks and DELIMITER

5.1.3 Python Compatibility Issues

Challenge: Different Python versions caused major compatibility problems. Streamlit and pyarrow were not compatible with Python 3.14, which led to repeated installation errors. Set us back quite some time trying to figure out what was wrong.

Solution:

- Switched back to Python 3.11

5.1.4 Virtual Environment Corrupted

Challenge: .venv broke multiple times (“bad interpreter”, pip pointing to a wrong path), causing Python to not run packages correctly.

Solution:

- Trial and error made us realize it was a problem with the directory name (“deliverable#5”). Changing the name of the directory fixed what multiple debugging attempts couldn't fix.

5.1.5 Handling Transaction Errors

Challenge: The CLI printed “Appointment scheduled” even when the DB insert failed

Solution:

- We updated the CLI to read the return value from schedule_appointment() and only print success if it returns True

5.2 Limitations

- Limited web interface
- Basic error reporting to users
- Lack of user authentication
- Lack of search and filtering functionalities
- etc.

5.3 AI-use disclaimer

This project leveraged AI tools to enhance its quality. AI was utilized in the following regards:

- Data generation was mainly handled by AI with a certain amount of human oversight to verify the accuracy of the sample data.
- Testing was partially handled by AI tools, as it generated a few advanced and complex queries to test triggers. All generated queries were manually reviewed and executed.
- AI was heavily used in interface development (using Streamlit), but ultimately everything was verified by the team.

6 Conclusion

This deliverable successfully demonstrates the practical application of database management concepts in a real-world healthcare context.

Through this lab, we managed to grasp the importance of foundational and complex SQL constructs to maintain data integrity, with the added layer of the relevance of this project through every day health services that impact us all.

The most eye-opening part was working with triggers to automatically apply business rules: preventing double-booking, validating stock values, and recomputing expenses whenever prescriptions changed. This made it clear how the database itself can protect the system from inconsistent or unsafe operations, instead of relying only on application code.

We also faced some technical difficulties, which helped us reinforce some good coding habits and better error handling when something didn't behave as we expected it to.

Ultimately, this lab not only strengthened our technical SQL skills, but also showed us how a well-designed database and application can directly support safer, more efficient healthcare services.