University
Mohammed VI
Polytechnic

College of Computing

Deliverable 4: Relational Schema Normalization

Data Management Course

UM6P College of Computing

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

Team Information

Team Name	alfari9 alkhari9
Member 1	Ilyas Rahmouni
Member 2	Malak Koulat
Member 3	Aymane Raiss
Member 4	Zakaria Harira
Member 5	Youness Latif
Member 6	Rayane Khaldi
Member 7	Younes Lougnidi
Repository Link	`https://github.com/...`

Contents

1 Task 1: Normalization

In this section, we analyze the normalization level of each relation in our database schema. For each relation, we list its functional dependencies (FDs) and assess its adherence to Boyce-Codd Normal Form (BCNF). A relation is in BCNF if for every non-trivial functional dependency $X \to Y$, the determinant X is a superkey of the relation.

1. **Patient**
   - IID → {CIN, FullName, Birth, Sex, BloodGroup, Phone, Email}
   - CIN → {IID, FullName, Birth, Sex, BloodGroup, Phone, Email}
   - *Justification:* The determinants are `IID` (the primary key) and `CIN` (a candidate key), both of which are superkeys. Therefore, the relation is in BCNF.

2. **ContactLocation**
   - CLID → {City, Province, Street, Number, PostalCode, Phone_Location}
   - *Justification:* The determinant `CLID` is the primary key and thus a superkey. The relation is in BCNF.

3. **Staff and ISA Subtypes (Practitioner, Caregiving, Technical)**
   - `Staff`: STAFF_ID → {FullName, Status}
   - `Practitioner`: STAFF_ID → {LicenseNumber, Speciality}
   - `Caregiving`: STAFF_ID → {Grade, Ward}
   - `Technical`: STAFF_ID → {Certifications, Modality}
   - *Justification:* In all four relations, the determinant of every FD is `STAFF_ID`, which is the primary key for each respective table. Consequently, all are in BCNF.

4. **Hospital**
   - HID → {Name, City, Region}
   - *Justification:* The determinant `HID` is the primary key, making it a superkey. The relation is in BCNF.

5. **Department**
   - DEP_ID → {HID, Name, Specialty}
   - *Justification:* The determinant `DEP_ID` is the primary key and a superkey. The relation is in BCNF.

6. **ClinicalActivity and ISA Subtypes (Appointment, Emergency)**
   - `ClinicalActivity`: CAID → {IID, STAFF_ID, DEP_ID, Date, Time}
   - `Appointment`: CAID → {Reason, Status}
   - `Emergency`: CAID → {TriageLevel, Outcome}

- *Justification:* The determinant for all three relations is `CAID`, which is the primary key and thus a superkey. All three relations are in BCNF.

7. **Insurance**
   - InsID → {Type}
   - *Justification:* The determinant `InsID` is the primary key, making it a superkey. The relation is in BCNF.

8. **Expense**
   - ExID → {Total, InsID, CAID}
   - CAID → {ExID, Total, InsID}
   - *Justification:* The determinants are `ExID` (primary key) and `CAID` (a candidate key, since it is unique), both of which are superkeys. The relation is in BCNF.

9. **Medication**
   - DrugID → {Class, Name, Form, Strength, Manufacturer, ActiveIngredient}
   - *Justification:* The determinant `DrugID` is the primary key and therefore a superkey. The relation is in BCNF.

10. **Stock**
    - {HID, DrugID, StockTimestamp} → {Unit_Price, Qty, ReorderLevel}
    - *Justification:* The determinant is the composite primary key, which is a superkey. The relation is in BCNF.

11. **Prescription**
    - PID → {DateIssued, CAID}
    - CAID → {PID, DateIssued}
    - *Justification:* The determinants are `PID` (primary key) and `CAID` (a candidate key), which are both superkeys. The relation is in BCNF.

12. **Many-to-Many Relationship Tables**
    - `Have`: {IID, CLID} → {}
    - `Work_in`: {STAFF_ID, DEP_ID} → {}
    - `Covers`: {IID, InsID} → {}
    - `Include`: {PID, DrugID} → {Dosage, Duration}
    - *Justification:* For `Have`, `Work_in`, and `Covers`, there are no non-key attributes, so no non-trivial FDs exist. For `Include`, the determinant of the only non-trivial FD is its primary key. Thus, all four relations are in BCNF.

Conclusion: Lossless Join and Dependency Preservation

The overall decomposition of our conceptual schema into the final set of relations is both **lossless** and **dependency-preserving**.

Lossless Join The lossless join is guaranteed by the fact that the initial step was to design the ER Model, then map all entities to relations in the relational schema. All relationships between entities were correctly implemented depending on their type: either using foreign keys or, for many-to-many relationships, by creating another table. This approach ensures that when the relations are joined, no data is lost and no incomplete tuples appear.

Dependency Preservation Based on the BCNF analysis, every functional dependency is fully contained within its corresponding relation. Because no functional dependency spans across multiple tables, the DBMS is able to enforce data constraints on each table individually, without needing to perform joins. This confirms that the decomposition is dependency-preserving.

2 Task 2: DDL Schema Creation

DDL Statements for the tables Creation

2.1 Database Creation

```
CREATE DATABASE MNHS;

USE MNHS;
```

Figure 1: Dabase Creation

2.2 Patient Relation

```
-- ======================
-- 1. PATIENT
-- ======================
CREATE TABLE Patient (
  IID INT PRIMARY KEY,
  CIN VARCHAR(10) UNIQUE NOT NULL,
  FullName VARCHAR(100) NOT NULL,
  Birth DATE,
  Sex ENUM('M','F') NOT NULL,
  BloodGroup ENUM('A+','A-','B+','B-','O+','O-','AB+','AB-'),
  Phone VARCHAR(15)
);
```

Figure 2: DDL for the Patient table.

2.3 ContactLocation Relation

```sql
-- =====================
-- 2. CONTACT LOCATION
-- =====================
CREATE TABLE ContactLocation (
  CLID INT PRIMARY KEY,
  City VARCHAR(50),
  Province VARCHAR(50),
  Street VARCHAR(100),
  Number VARCHAR(10),
  PostalCode VARCHAR(10),
  Phone_Location VARCHAR(15)
);
```

Figure 3: DDL for the ContactLocation table.

2.4 Have Relation

```sql
-- HAVE (Patient - ContactLocation)
CREATE TABLE Have (
  IID INT,
  CLID INT,
  PRIMARY KEY (IID, CLID),
  FOREIGN KEY (IID) REFERENCES Patient(IID),
  FOREIGN KEY (CLID) REFERENCES ContactLocation(CLID)
);
```

Figure 4: DDL for the Have junction table.

2.5 Staff Relation (Superclass)

```sql
-- ======================
-- 3. STAFF (Superclass)
-- ======================
CREATE TABLE Staff (
  STAFF_ID INT PRIMARY KEY,
  FullName VARCHAR(100) NOT NULL,
  Status ENUM('Active','Retired') DEFAULT 'Active'
);
```

Figure 5: DDL for the Staff superclass table.

2.6 Staff ISA Subtypes (Practitioner, Caregiving, Technical)

```sql
-- Practitioner (ISA)
CREATE TABLE Practitioner (
  STAFF_ID INT PRIMARY KEY,
  LicenseNumber VARCHAR(100),
  Speciality VARCHAR(200),
  FOREIGN KEY (STAFF_ID) REFERENCES Staff(STAFF_ID) ON DELETE CASCADE
);

-- Caregiving (ISA)
CREATE TABLE Caregiving (
  STAFF_ID INT PRIMARY KEY,
  Grade VARCHAR(50),
  Ward VARCHAR(100),
  FOREIGN KEY (STAFF_ID) REFERENCES Staff(STAFF_ID) ON DELETE CASCADE
);

-- Technical (ISA)
CREATE TABLE Technical (
  STAFF_ID INT PRIMARY KEY,
  Certifications VARCHAR(500),
  Modality VARCHAR(200),
  FOREIGN KEY (STAFF_ID) REFERENCES Staff(STAFF_ID) ON DELETE CASCADE
);
```

Figure 6: DDL for the Practitioner, Caregiving, and Technical subtype tables.

2.7 Hospital Relation

```
-- ====================
-- 4. HOSPITAL
-- ====================
CREATE TABLE Hospital (
  HID INT PRIMARY KEY,
  Name VARCHAR(100) NOT NULL,
  City VARCHAR(50) NOT NULL,
  Region VARCHAR(50)
);
```

Figure 7: DDL for the Hospital table.

2.8 Department Relation

```
-- ======================
-- 5. DEPARTMENT (belongs to Hospital)
-- ======================
CREATE TABLE Department (
  DEP_ID INT PRIMARY KEY,
  HID INT NOT NULL,
  Name VARCHAR(100) NOT NULL,
  Specialty VARCHAR(100),
  FOREIGN KEY (HID) REFERENCES Hospital(HID)
);
```

Figure 8: DDL for the Department table.

2.9 Work_in Relation

```
-- ======================
-- 6. WORK_IN (Staff - Department)
-- ======================
CREATE TABLE Work_in (
  STAFF_ID INT,
  DEP_ID INT,
  PRIMARY KEY (STAFF_ID, DEP_ID),
  FOREIGN KEY (STAFF_ID) REFERENCES Staff(STAFF_ID),
  FOREIGN KEY (DEP_ID) REFERENCES Department(DEP_ID)
);
```

Figure 9: DDL for the Work_in junction table.

2.10 ClinicalActivity Relation (Superclass)

```
-- ======================
-- 7. CLINICAL ACTIVITY
-- ======================
CREATE TABLE ClinicalActivity (
  CAID INT PRIMARY KEY,
  IID INT NOT NULL,
  STAFF_ID INT NOT NULL,
  DEP_ID INT NOT NULL,
  Date DATE NOT NULL,
  Time TIME,
  FOREIGN KEY (IID) REFERENCES Patient(IID),
  FOREIGN KEY (STAFF_ID) REFERENCES Staff(STAFF_ID),
  FOREIGN KEY (DEP_ID) REFERENCES Department(DEP_ID)
);
```

Figure 10: DDL for the ClinicalActivity superclass table.

2.11 ClinicalActivity ISA Subtypes (Appointment, Emergency)

```sql
-- APPOINTMENT (subclass)
CREATE TABLE Appointment (
  CAID INT PRIMARY KEY,
  Reason VARCHAR(500),
  Status ENUM('Scheduled','Completed','Cancelled') DEFAULT 'Scheduled',
  FOREIGN KEY (CAID) REFERENCES ClinicalActivity(CAID) ON DELETE CASCADE
);

-- EMERGENCY (subclass)
CREATE TABLE Emergency (
  CAID INT PRIMARY KEY,
  TriageLevel INT CHECK (TriageLevel BETWEEN 1 AND 5),
  Outcome ENUM('Discharged','Admitted','Transferred','Deceased'),
  FOREIGN KEY (CAID) REFERENCES ClinicalActivity(CAID) ON DELETE CASCADE
);
```

Figure 11: DDL for the Appointment and Emergency subtype tables.

2.12 Insurance and Covers Relations

```sql
-- =====================
-- 8. INSURANCE
-- =====================
CREATE TABLE Insurance (
  InsID INT PRIMARY KEY,
  Type ENUM('CNOPS','CNSS','RAMED','Private','None') NOT NULL
);

-- COVERS (Insurance - Patient)
CREATE TABLE Covers (
  IID INT,
  InsID INT,
  PRIMARY KEY (IID, InsID),
  FOREIGN KEY (IID) REFERENCES Patient(IID),
  FOREIGN KEY (InsID) REFERENCES Insurance(InsID)
);
```

Figure 12: DDL for the Insurance table and the Covers junction table.

2.13 Expense Relation

```sql
-- =====================
-- 9. EXPENSE (linked to ClinicalActivity + Insurance)
-- =====================
CREATE TABLE Expense (
  ExID INT PRIMARY KEY,
  Total DECIMAL(10,2) NOT NULL CHECK (Total >= 0),
  InsID INT,
  CAID INT UNIQUE NOT NULL,
  FOREIGN KEY (InsID) REFERENCES Insurance(InsID) ON DELETE CASCADE,
  FOREIGN KEY (CAID) REFERENCES ClinicalActivity(CAID) ON DELETE CASCADE
);
```

Figure 13: DDL for the Expense table.

2.14 Medication Relation

```sql
-- =====================
-- 10. MEDICATION
-- =====================
CREATE TABLE Medication (
  DrugID INT PRIMARY KEY,
  Class VARCHAR(200),
  Name VARCHAR(200) NOT NULL,
  Form VARCHAR(200),
  Strength VARCHAR(50),
  Manufacturer VARCHAR(200),
  ActiveIngredient VARCHAR(100)
);
```

Figure 14: DDL for the Medication table.

2.15 Stock Relation

```sql
-- =====================
-- 11. STOCK (Hospital - Medication)
-- =====================
CREATE TABLE Stock (
  HID INT,
  DrugID INT,
  Unit_Price DECIMAL(10,2) CHECK (Unit_Price >= 0),
  StockTimestamp DATETIME DEFAULT CURRENT_TIMESTAMP,
  Qty INT DEFAULT 0 CHECK (Qty >= 0),
  ReorderLevel INT DEFAULT 10 CHECK (ReorderLevel >= 0),
  PRIMARY KEY (HID, DrugID, StockTimestamp),
  FOREIGN KEY (HID) REFERENCES Hospital(HID),
  FOREIGN KEY (DrugID) REFERENCES Medication(DrugID)
);
```

Figure 15: DDL for the Stock table.

2.16 Prescription Relation

```sql
-- =====================
-- 12. PRESCRIPTION
-- =====================
CREATE TABLE Prescription (
  PID INT PRIMARY KEY,
  DateIssued DATE NOT NULL,
  CAID INT UNIQUE NOT NULL,
  FOREIGN KEY (CAID) REFERENCES ClinicalActivity(CAID) ON DELETE CASCADE
);
```

Figure 16: DDL for the Prescription table.

2.17 Include Relation

```sql
-- INCLUDE (Prescription - Medication)
CREATE TABLE Include (
  PID INT,
  DrugID INT,
  Dosage VARCHAR(100),
  Duration VARCHAR(100),
  PRIMARY KEY (PID, DrugID),
  FOREIGN KEY (PID) REFERENCES Prescription(PID),
  FOREIGN KEY (DrugID) REFERENCES Medication(DrugID)
);
```

Figure 17: DDL for the Include junction table.

2.18 DDL Alter to Add Attribute

```sql
-- ========================================
-- DDL: ALTER TABLE TO ADD EMAIL TO PATIENT
-- ========================================
ALTER TABLE Patient ADD Email VARCHAR(100);
```

Figure 18: DDL statement to add the Email attribute to the Patient table.

3 Task 3: DML Data Manipulation

DML Statements to populate each table with sample data;

3.1 Patient Data Insertion

```
-- Insert into Patient
INSERT INTO Patient (IID, CIN, FullName, Birth, Sex, BloodGroup, Phone, Email) VALUES
(1, 'AB123456', 'Ahmed Benali', '1985-03-15', 'M', 'A+', '0612345678', 'ahmed.benali@email.com'),
(2, 'CD789012', 'Fatima Zahra', '1990-07-22', 'F', 'O+', '0623456789', 'fatima.zahra@email.com'),
(3, 'EF345678', 'Mohammed Amine', '1978-11-30', 'M', 'B-', '0634567890', 'mohammed.amine@email.com'),
(4, 'GH901234', 'Aicha Toumi', '1982-05-14', 'F', 'AB+', '0645678901', 'aicha.toumi@email.com'),
(5, 'IJ567890', 'Youssef Khalid', '1995-09-08', 'M', 'A-', '0656789012', 'youssef.khalid@email.com');
```

Figure 19: DML INSERT statements for the Patient table.

3.2 ContactLocation Data Insertion

```
-- Insert into ContactLocation
INSERT INTO ContactLocation (CLID, City, Province, Street, Number, PostalCode, Phone_Location) VALUES
(1, 'Casablanca', 'Casablanca-Settat', 'Rue Mohammed V', '123', '20000', '0522123456'),
(2, 'Rabat', 'Rabat-Salé-Kénitra', 'Avenue Hassan II', '45', '10000', '0537123456'),
(3, 'Marrakech', 'Marrakech-Safi', 'Rue de la Koutoubia', '67', '40000', '0524123456'),
(4, 'Fès', 'Fès-Meknès', 'Boulevard Mohammed VI', '89', '30000', '0535123456'),
(5, 'Tanger', 'Tanger-Tétouan-Al Hoceïma', 'Avenue d''Espagne', '101', '90000', '0539123456');
```

Figure 20: DML INSERT statements for the ContactLocation table.

3.3 Have Data Insertion

```
-- Insert into Have (Patient-ContactLocation relationship)
INSERT INTO Have (IID, CLID) VALUES
(1, 1), (2, 2), (3, 3), (4, 4), (5, 5);
```

Figure 21: DML INSERT statements for the Have table.

3.4 Staff Data Insertion

```
-- Insert into Staff
INSERT INTO Staff (STAFF_ID, FullName, Status) VALUES
(1, 'Dr. Kamal El Fassi', 'Active'),
(2, 'Dr. Leila Berrada', 'Active'),
(3, 'Nurse Fatima Ezzahra', 'Active'),
(4, 'Nurse Hassan Mounir', 'Active'),
(5, 'Tech Ahmed Radi', 'Active'),
(6, 'Dr. Samira Alaoui', 'Active'),
(7, 'Nurse Karim Bennani', 'Active'),
(8, 'Tech Younes Cherkaoui', 'Active');
```

Figure 22: DML INSERT statements for the Staff table.

3.5 Practitioner Data Insertion

```sql
-- Insert into Practitioner
INSERT INTO Practitioner (STAFF_ID, LicenseNumber, Speciality) VALUES
(1, 'MED12345', 'Cardiology'),
(2, 'MED12346', 'Pediatrics'),
(6, 'MED12347', 'General Surgery');
```

Figure 23: DML INSERT statements for the Practitioner table.

3.6 Caregiving Data Insertion

```sql
-- Insert into Caregiving
INSERT INTO Caregiving (STAFF_ID, Grade, Ward) VALUES
(3, 'Senior Nurse', 'Emergency'),
(4, 'Junior Nurse', 'Pediatrics'),
(7, 'Head Nurse', 'Surgery');
```

Figure 24: DML INSERT statements for the Caregiving table.

3.7 Technical Data Insertion

```sql
-- Insert into Technical
INSERT INTO Technical (STAFF_ID, Certifications, Modality) VALUES
(5, 'Radiology Technician', 'X-Ray, MRI'),
(8, 'Lab Technician', 'Blood Analysis, Microbiology');
```

Figure 25: DML INSERT statements for the Technical table.

3.8 Hospital Data Insertion

```sql
-- Insert into Hospital
INSERT INTO Hospital (HID, Name, City, Region) VALUES
(1, 'CHU Ibn Rochd', 'Casablanca', 'Casablanca-Settat'),
(2, 'Hôpital Avicenne', 'Rabat', 'Rabat-Salé-Kénitra'),
(3, 'Hôpital Mohammed VI', 'Marrakech', 'Marrakech-Safi'),
(4, 'CHU Hassan II', 'Fès', 'Fès-Meknès'),
(5, 'Hôpital Moulay Hassan', 'Tanger', 'Tanger-Tétouan-Al Hoceïma');
```

Figure 26: DML INSERT statements for the Hospital table.

3.9 Department Data Insertion

```sql
-- Insert into Department
INSERT INTO Department (DEP_ID, HID, Name, Specialty) VALUES
(1, 1, 'Cardiology', 'Heart Diseases'),
(2, 1, 'Emergency', 'Emergency Care'),
(3, 2, 'Pediatrics', 'Child Healthcare'),
(4, 3, 'Surgery', 'General Surgery'),
(5, 4, 'Radiology', 'Medical Imaging');
```

Figure 27: DML INSERT statements for the Department table.

3.10 Work_in Data Insertion

```sql
-- Insert into Work_in
INSERT INTO Work_in (STAFF_ID, DEP_ID) VALUES
(1, 1), (2, 3), (3, 2), (4, 3), (5, 5), (6, 4), (7, 4), (8, 5);
```

Figure 28: DML INSERT statements for the Work_in table.

3.11 ClinicalActivity Data Insertion

```sql
-- Insert into ClinicalActivity
INSERT INTO ClinicalActivity (CAID, IID, STAFF_ID, DEP_ID, Date, Time) VALUES
(1, 1, 1, 1, '2025-11-17', '09:00:00'),
(2, 2, 2, 3, '2025-11-20', '10:30:00'),
(3, 3, 1, 1, '2025-11-21', '14:00:00'),
(4, 4, 6, 4, '2024-01-16', '11:00:00'),
(5, 5, 2, 3, '2024-01-17', '15:30:00'),
(6, 1, 3, 2, '2024-01-18', '08:00:00'),
(7, 2, 3, 2, '2024-01-18', '20:30:00');
```

Figure 29: DML INSERT statements for the ClinicalActivity table.

3.12 Appointment Data Insertion

```sql
-- Insert into Appointment
INSERT INTO Appointment (CAID, Reason, Status) VALUES
(1, 'Routine heart checkup', 'Completed'),
(2, 'Child vaccination', 'Completed'),
(3, 'Cardiac consultation', 'Scheduled'),
(4, 'Pre-surgery consultation', 'Scheduled'),
(5, 'Pediatric follow-up', 'Cancelled');
```

Figure 30: DML INSERT statements for the Appointment table.

3.13 Emergency Data Insertion

```sql
-- Insert into Emergency
INSERT INTO Emergency (CAID, TriageLevel, Outcome) VALUES
(6, 2, 'Admitted'),
(7, 1, 'Discharged');
```

Figure 31: DML INSERT statements for the Emergency table.

3.14 Insurance Data Insertion

```sql
-- Insert into Insurance
INSERT INTO Insurance (InsID, Type) VALUES
(1, 'CNOPS'),
(2, 'CNSS'),
(3, 'RAMED'),
(4, 'Private'),
(5, 'None');
```

Figure 32: DML INSERT statements for the Insurance table.

3.15 Covers Data Insertion

```sql
-- Insert into Covers
INSERT INTO Covers (IID, InsID) VALUES
(1, 1), (2, 2), (3, 3), (4, 4), (5, 5);
```

Figure 33: DML INSERT statements for the Covers table.

3.16 Expense Data Insertion

```sql
-- Insert into Expense
INSERT INTO Expense (ExID, Total, InsID, CAID) VALUES
(1, 500.00, 1, 1),
(2, 300.00, 2, 2),
(3, 1200.00, 4, 4),
(4, 750.00, NULL, 6),
(5, 450.00, 3, 3);
```

Figure 34: DML INSERT statements for the Expense table.

3.17 Medication Data Insertion

```sql
INSERT INTO Medication (DrugID, Class, Name, Form, Strength, Manufacturer, ActiveIngredient) VALUES
(1, 'Antihypertensive', 'Amlodipine', 'Tablet', '5mg', 'PharmaMaroc', 'Amlodipine Besylate'),
(2, 'Analgesic', 'Paracetamol', 'Tablet', '500mg', 'Sothema', 'Paracetamol'),
(3, 'Antibiotic', 'Amoxicillin', 'Capsule', '250mg', 'Cooper', 'Amoxicillin Trihydrate'),
(4, 'Antidiabetic', 'Metformin', 'Tablet', '850mg', 'PharmaMaroc', 'Metformin HCl'),
(5, 'Anticoagulant', 'Warfarin', 'Tablet', '5mg', 'Sothema', 'Warfarin Sodium');
```

Figure 35: DML INSERT statements for the Medication table.

3.18 Stock Data Insertion

```sql
-- Insert into Stock
INSERT INTO Stock (HID, DrugID, Unit_Price, Qty, ReorderLevel) VALUES
(1, 1, 25.50, 100, 20),
(1, 2, 8.75, 250, 50),
(2, 3, 45.00, 80, 15),
(3, 4, 32.25, 120, 25),
(4, 5, 18.90, 60, 10);
```

Figure 36: DML INSERT statements for the Stock table.

3.19 Prescription Data Insertion

```sql
-- Insert into Prescription
INSERT INTO Prescription (PID, DateIssued, CAID) VALUES
(1, '2024-01-15', 1),
(2, '2024-01-15', 2),
(3, '2024-01-16', 3),
(4, '2024-01-16', 4),
(5, '2024-01-17', 5);
```

Figure 37: DML INSERT statements for the Prescription table.

3.20 Include Data Insertion

```sql
-- Insert into Include
INSERT INTO Include (PID, DrugID, Dosage, Duration) VALUES
(1, 1, '1 tablet daily', '30 days'),
(1, 2, '1 tablet when needed', '7 days'),
(2, 3, '1 capsule 3 times daily', '10 days'),
(3, 1, '1 tablet daily', '90 days'),
(4, 4, '1 tablet twice daily', '60 days');
```

Figure 38: DML INSERT statements for the Include table.

3.21 DML Update Operations

This subsection shows the DML statements used to update existing records, specifically a patient's phone number and a hospital's region.

```sql
-- ========================================
-- DML: UPDATE OPERATIONS
-- ========================================

-- Update a patient's phone number
UPDATE Patient
SET Phone = '0666666666'
WHERE IID = 1;

-- Update a hospital's region
UPDATE Hospital
SET Region = 'Grand Casablanca'
WHERE HID = 1;
```

Figure 39: DML UPDATE statements for Patient and Hospital tables.

3.22 DML Delete Operation

This subsection displays the DML statement for deleting a specific record, in this case, a cancelled appointment. The SQL SAFE UPDATES mode is temporarily disabled to allow deletion based on a non-key attribute.

```
-- ========================================
-- DML: DELETE OPERATION
-- ========================================

-- Delete a scheduled appointment that was cancelled
SET SQL_SAFE_UPDATES = 0;
DELETE FROM Appointment WHERE Status = 'Cancelled';
SET SQL_SAFE_UPDATES = 1;

-- MySQL uses a protective feature called SAFE_UPDATES that requires DELETE/UPDATE to use key columns in WHERE clause
-- That's why we need to disable it to allow DELETE without primary key in the WHERE clause
```

Figure 40: DML DELETE statement for the Appointment table.

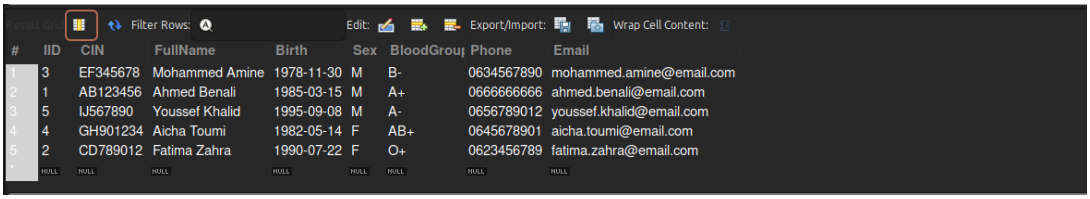
4 Task 4: Queries

This final task showcases a series of 20 queries designed to retrieve and analyze data from the MNHS database. Each query is presented with its SQL statement and the resulting output.

4.1 Select all patients ordered by last name

```sql
-- Query 1
SELECT *
FROM Patient
ORDER BY SUBSTRING_INDEX(FullName, ' ', -1);-- This method extracts the last name if fullname format is "FirstName LastName"
```

Figure 41: Query 1: SQL Statement.



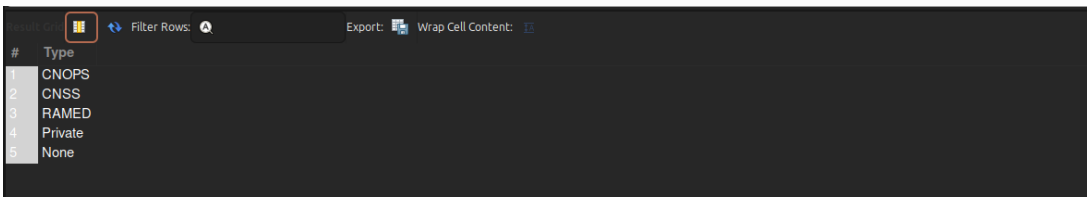
#	IID	CIN	FullName	Birth	Sex	BloodGrou	Phone	Email
1	3	EF345678	Mohammed Amine	1978-11-30	M	B-	0634567890	mohammed.amine@email.com
2	1	AB123456	Ahmed Benali	1985-03-15	M	A+	0666666666	ahmed.benali@email.com
3	5	IJ567890	Youssef Khalid	1995-09-08	M	A-	0656789012	youssef.khalid@email.com
4	4	GH901234	Aicha Toumi	1982-05-14	F	AB+	0645678901	aicha.toumi@email.com
5	2	CD789012	Fatima Zahra	1990-07-22	F	O+	0623456789	fatima.zahra@email.com

Figure 42: Query 1: Output.

4.2 List distinct insurance types

```sql
-- Query 2
SELECT DISTINCT Type
FROM Insurance;
```

Figure 43: Query 2: SQL Statement.



#	Type
1	CNOPS
2	CNSS
3	RAMED
4	Private
5	None

Figure 44: Query 2: Output.

4.3 Retrieve staff who work in hospitals located in Rabat

```sql
-- Query 3
select w.STAFF_ID
from Work_in as w
join Department as dep on dep.DEP_ID = w.DEP_ID
join Hospital as h on h.HID = dep.HID
where h.city = "Rabat";
```

Figure 45: Query 3: SQL Statement.

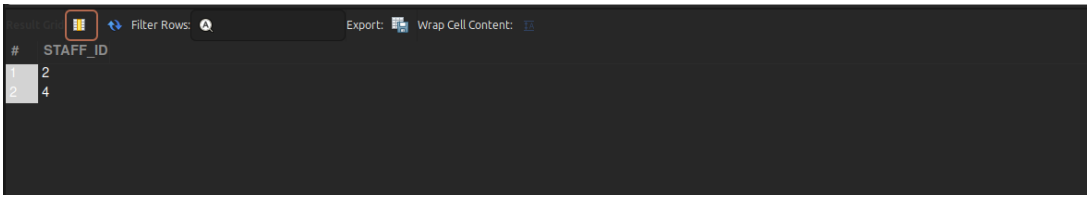


Figure 46: Query 3: Output.

4.4 Find all appointments that are scheduled within the next seven days

```sql
-- Query 4
SELECT A.CAID, CA.IID, CA.STAFF_ID, CA.DEP_ID, CA.Date, CA.Time, A.Reason
FROM Appointment A
JOIN ClinicalActivity CA ON A.CAID = CA.CAID
WHERE A.Status = 'Scheduled'
  AND CA.Date BETWEEN CURDATE() AND DATE_ADD(CURDATE(), INTERVAL 7 DAY);
```

Figure 47: Query 4: SQL Statement.

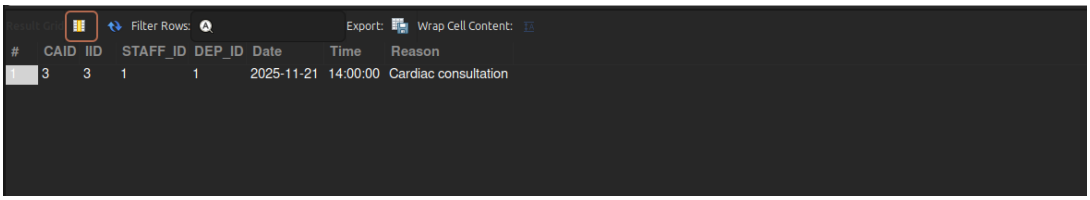


Figure 48: Query 4: Output.

4.5 Count the number of appointments per department

```sql
-- Query 5
SELECT dep.DEP_ID, dep.Name, COUNT(a.CAID) AS AppCount
FROM Appointment a
JOIN ClinicalActivity ca ON ca.CAID = a.CAID
JOIN Department dep ON dep.DEP_ID = ca.DEP_ID
GROUP BY dep.DEP_ID, dep.Name;
```

Figure 49: Query 5: SQL Statement.

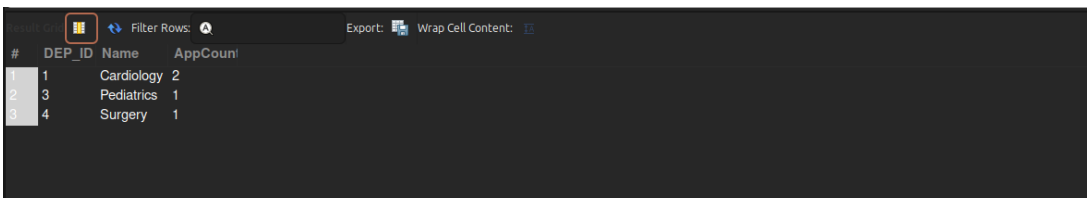


Figure 50: Query 5: Output.

4.6 Compute the average unit price of medications per hospital

```sql
-- Query 6
SELECT h.HID, h.Name AS HospitalName, AVG(s.Unit_Price) AS AverageUnitPrice
FROM Hospital h
JOIN Stock s ON h.HID = s.HID
GROUP BY h.HID, h.Name;
```

Figure 51: Query 6: SQL Statement.

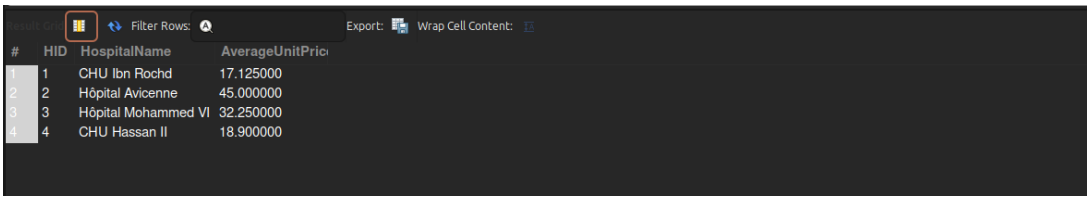


Figure 52: Query 6: Output.

4.7 List hospitals with more than twenty emergency admissions

```sql
-- Query 7
SELECT h.Name, COUNT(*)
FROM Hospital h
JOIN Department d ON h.HID = d.HID
JOIN ClinicalActivity ca ON d.DEP_ID = ca.DEP_ID
JOIN Emergency e ON ca.CAID = e.CAID
GROUP BY h.HID, h.Name
HAVING COUNT(*) > 20;
```

Figure 53: Query 7: SQL Statement.

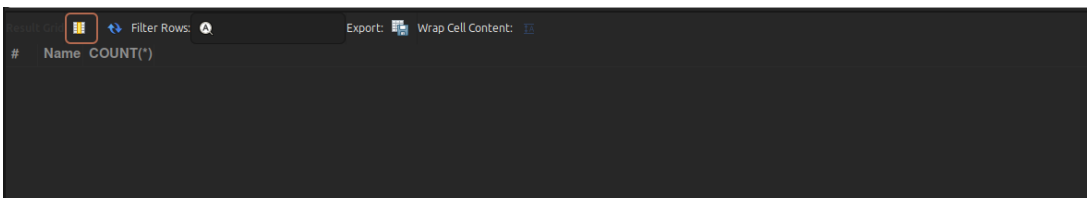


Figure 54: Query 7: Output.

4.8 Find medications in the therapeutic class Antibiotic where the unit price is below 200

```sql
-- Query 8
SELECT DISTINCT
    M.Name AS MedicationName,
    M.Class,
    S.Unit_Price,
    H.Name AS HospitalName
FROM
    Medication AS M
JOIN
    Stock AS S ON M.DrugID = S.DrugID
JOIN
    Hospital AS H ON S.HID = H.HID
WHERE
    M.Class = 'Antibiotic'
    AND S.Unit_Price < 200;
```

Figure 55: Query 8: SQL Statement.

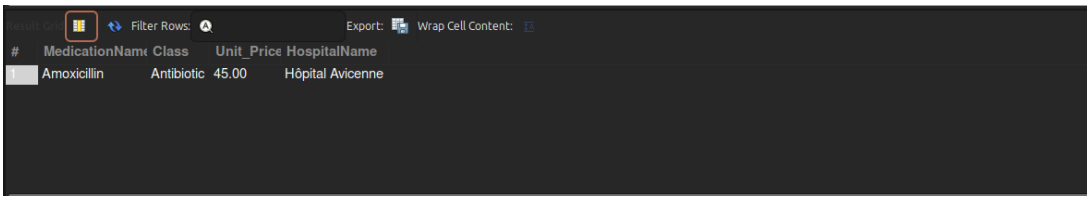
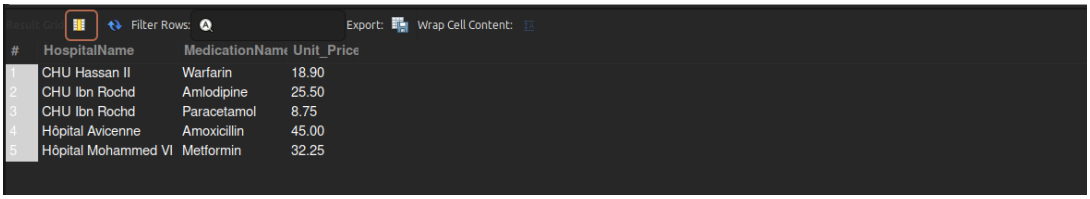


Figure 56: Query 8: Output.

4.9 For each hospital, list the top three most expensive medications

```sql
-- Query 9
SELECT H.Name AS HospitalName, M.Name AS MedicationName, S.Unit_Price
FROM Stock S
JOIN Hospital H ON S.HID = H.HID
JOIN Medication M ON S.DrugID = M.DrugID
WHERE 3 > (
    SELECT COUNT(*)
    FROM Stock S2
    WHERE S2.HID = S.HID AND S2.Unit_Price > S.Unit_Price
)
ORDER BY HospitalName, S.Unit_Price DESC;
```

Figure 57: Query 9: SQL Statement.



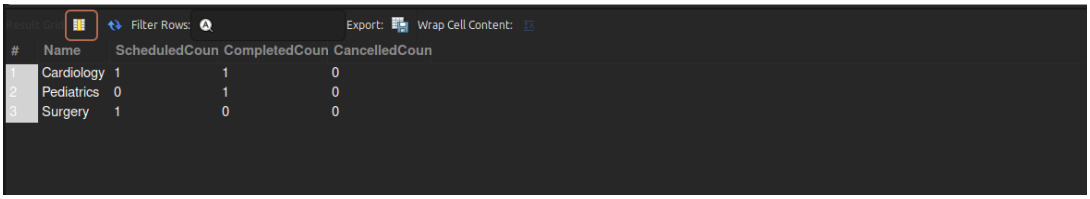
#	HospitalName	MedicationName	Unit_Price
1	CHU Hassan II	Warfarin	18.90
2	CHU Ibn Rochd	Amlodipine	25.50
3	CHU Ibn Rochd	Paracetamol	8.75
4	Hôpital Avicenne	Amoxicillin	45.00
5	Hôpital Mohammed VI	Metformin	32.25

Figure 58: Query 9: Output.

4.10 For each department, return counts of Scheduled, Completed, and Cancelled appointments

```sql
-- Query 10
SELECT d.Name,
       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 Department d
JOIN ClinicalActivity ca ON ca.DEP_ID = d.DEP_ID
JOIN Appointment a ON a.CAID = ca.CAID
GROUP BY d.DEP_ID, d.Name;
```

Figure 59: Query 10: SQL Statement.



#	Name	ScheduledCoun	CompletedCoun	CancelledCoun
1	Cardiology	1	1	0
2	Pediatrics	0	1	0
3	Surgery	1	0	0

Figure 60: Query 10: Output.

4.11 List patients who have no scheduled appointments in the next thirty days

```sql
-- Query 11
SELECT *
FROM Patient
WHERE IID NOT IN (
    SELECT DISTINCT IID
    FROM ClinicalActivity
    WHERE Date <= DATE_ADD(NOW(), INTERVAL 30 DAY)
    AND CAID IN (
        SELECT CAID
        FROM Appointment
        WHERE Status = 'Scheduled'
    )
);
```

Figure 61: Query 11: SQL Statement.

Filter Rows: Edit: Export/Import: Wrap Cell Content:

#	IID	CIN	FullName	Birth	Sex	BloodGroup	Phone	Email
1	1	AB123456	Ahmed Benali	1985-03-15	M	A+	0666666666	ahmed.benali@email.com
2	2	CD789012	Fatima Zahra	1990-07-22	F	O+	0623456789	fatima.zahra@email.com
3	5	IJ567890	Youssef Khalid	1995-09-08	M	A-	0656789012	youssef.khalid@email.com
4	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figure 62: Query 11: Output.

4.12 For each staff member, compute their percentage share of appointments in their hospital

```sql
-- Query 12
select s.STAFF_ID , s.StaffApp , round((s.StaffApp/h.hospitalApp)*100,2) as PercentageShare
from (
select ca.STAFF_ID, dep.HID ,count(a.CAID) as StaffApp
from Appointment a
join ClinicalActivity ca on a.CAID = ca.CAID
join Department dep on dep.DEP_ID = ca.DEP_ID
group by ca.STAFF_ID , dep.HID
) s
join (
select dep.HID ,count(a.CAID) as hospitalAPP
from Appointment a
join ClinicalActivity ca on ca.CAID = a.CAID
join Department dep on dep.DEP_ID = ca.DEP_ID
group by dep.HID
) h on s.HID = h.HID;
```

Figure 63: Query 12: SQL Statement.

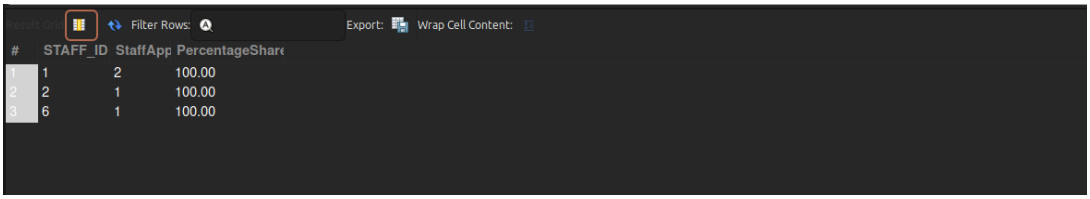


Figure 64: Query 12: Output.

4.13 Show drugs below ReorderLevel and the hospitals where this occurs

```sql
-- Query 13
SELECT M.Name AS MedicationName, GROUP_CONCAT(H.Name SEPARATOR ', ') AS LowStockHospitals
FROM Stock S
JOIN Medication M ON S.DrugID = M.DrugID
JOIN Hospital H ON S.HID = H.HID
WHERE S.Qty < S.ReorderLevel
GROUP BY M.DrugID;
```

Figure 65: Query 13: SQL Statement.

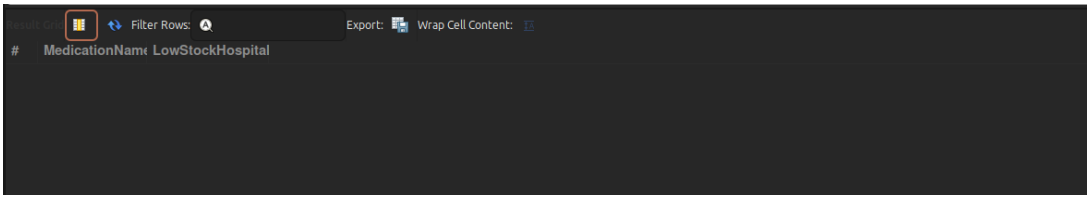


Figure 66: Query 13: Output.

4.14 Find hospitals that stock every antibiotic in the catalog

```sql
-- Query 14
SELECT
    H.Name AS HospitalName
FROM
    Stock AS S
JOIN
    Hospital AS H ON S.HID = H.HID
JOIN
    Medication AS M ON S.DrugID = M.DrugID
WHERE
    M.Class = 'Antibiotic'
GROUP BY
    H.HID, H.Name
HAVING
    COUNT(DISTINCT M.DrugID) = (
        SELECT COUNT(*)
        FROM Medication
        WHERE Class = 'Antibiotic'
    );
```

Figure 67: Query 14: SQL Statement.

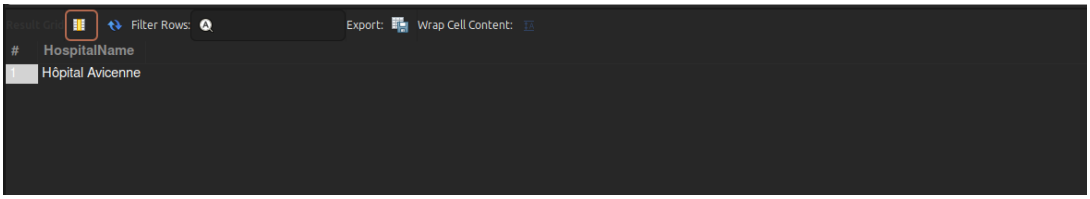


Figure 68: Query 14: Output.

4.15 For each hospital and drug class, flag if its average price is above the citywide average

```sql
-- Query 15
WITH CityAvg AS (
    SELECT
        m.Class AS DrugClass,
        h.City,
        AVG(s.Unit_Price) AS CityAvgPrice
    FROM Stock s
    JOIN Medication m ON s.DrugID = m.DrugID
    JOIN Hospital h ON s.HID = h.HID
    GROUP BY m.Class, h.City
),
HospitalAvg AS (
    SELECT
        h.HID,
        h.Name AS HospitalName,
        h.City,
        m.Class AS DrugClass,
        AVG(s.Unit_Price) AS HospitalAvgPrice
    FROM Stock s
    JOIN Medication m ON s.DrugID = m.DrugID
    JOIN Hospital h ON s.HID = h.HID
    GROUP BY h.HID, h.Name, h.City, m.Class
)
```

Figure 69: Query 15: SQL Statement (Part 1).

```sql
SELECT
    HA.HID,
    HA.HospitalName,
    HA.DrugClass,
    HA.HospitalAvgPrice,
    CA.CityAvgPrice,
    (HA.HospitalAvgPrice > CA.CityAvgPrice) AS AboveCityAverage
FROM HospitalAvg HA
JOIN CityAvg CA
    ON HA.DrugClass = CA.DrugClass
   AND HA.City = CA.City;
```

Figure 70: Query 15: SQL Statement (Part 2).

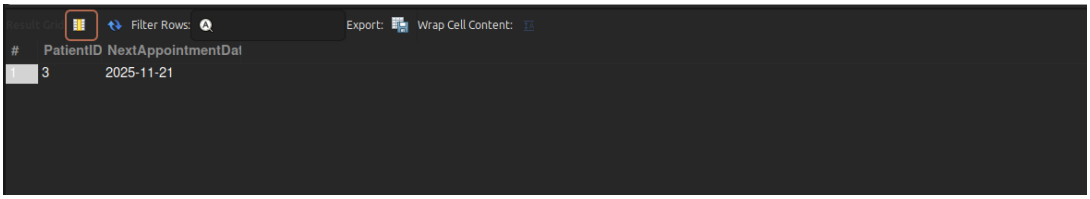
#	HID	HospitalName	DrugClass	HospitalAvgPric	CityAvgPric	AboveCityAverag
1	1	CHU Ibn Rochd	Antihypertensive	25.500000	25.500000	0
2	1	CHU Ibn Rochd	Analgesic	8.750000	8.750000	0
3	2	Hôpital Avicenne	Antibiotic	45.000000	45.000000	0
4	3	Hôpital Mohammed VI	Antidiabetic	32.250000	32.250000	0
5	4	CHU Hassan II	Anticoagulant	18.900000	18.900000	0

Figure 71: Query 15: Output.

4.16 Return the next appointment date for each patient

```sql
-- Query 16
SELECT
    CA.IID AS PatientID,
    MIN(CA.Date) AS NextAppointmentDate
FROM ClinicalActivity CA
JOIN Appointment A ON A.CAID = CA.CAID
WHERE A.Status = 'Scheduled'
  AND CA.Date >= CURDATE()
GROUP BY CA.IID;
```

Figure 72: Query 16: SQL Statement.



#	PatientID	NextAppointmentDat
1	3	2025-11-21

Figure 73: Query 16: Output.

4.17 List patients with 2+ emergency visits whose latest visit was in the last 14 days

```sql
-- Query 17
SELECT CA.IID AS PatientID,
       MAX(CA.Date) AS LatestEmergencyDate,
       COUNT(*) AS TotalEmergencies
FROM ClinicalActivity CA
JOIN Emergency E ON E.CAID = CA.CAID
WHERE CA.Date >= DATE_SUB(CURDATE(), INTERVAL 14 DAY)
GROUP BY CA.IID
HAVING COUNT(*) >= 2;
```

Figure 74: Query 17: SQL Statement.

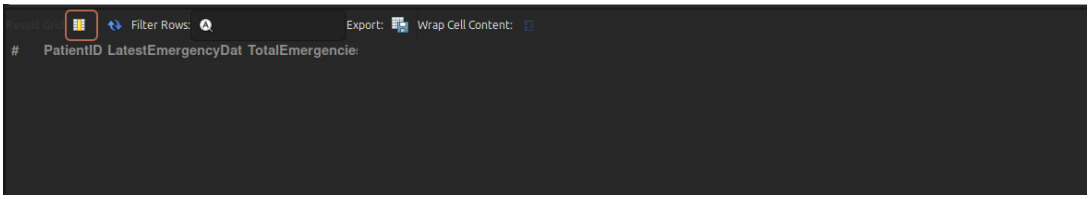


Figure 75: Query 17: Output.

4.18 For each city, rank hospitals by completed appointments in the last 90 days

```sql
-- Query 18
WITH HospitalCounts AS (
    SELECT
        h.City,
        h.Name AS HospitalName,
        COUNT(a.CAID) AS CompletedCount
    FROM
        Hospital h
    JOIN
        Department d ON h.HID = d.HID
    JOIN
        ClinicalActivity ca ON d.DEP_ID = ca.DEP_ID
    JOIN
        Appointment a ON ca.CAID = a.CAID
    WHERE
        a.Status = 'Completed'
        AND ca.Date >= CURDATE() - INTERVAL 90 DAY
    GROUP BY
        h.City, h.Name
)
```

Figure 76: Query 18: SQL Statement (Part 1).

```sql
SELECT
    City,
    HospitalName,
    CompletedCount,
    RANK() OVER (PARTITION BY City ORDER BY CompletedCount DESC) AS HospitalRank
FROM
    HospitalCounts
ORDER BY
    City, HospitalRank;
```

Figure 77: Query 18: SQL Statement (Part 2).

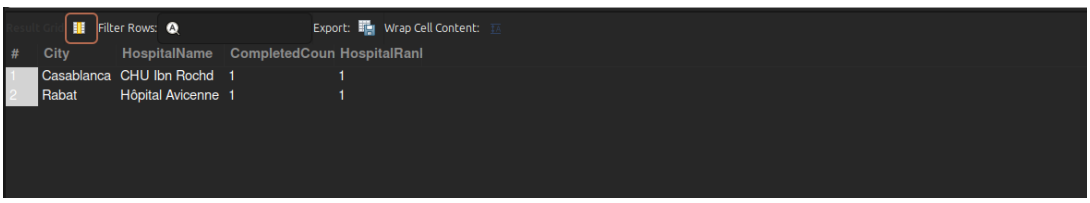


Figure 78: Query 18: Output.

4.19 Find medications with a hospital price spread greater than 30% within a city

```sql
-- Query 19
SELECT H.City, M.Name AS MedicationName
FROM Stock S
JOIN Hospital H ON S.HID = H.HID
JOIN Medication M ON S.DrugID = M.DrugID
GROUP BY H.City, M.DrugID
HAVING (MAX(S.Unit_Price) - MIN(S.Unit_Price)) / MIN(S.Unit_Price) > 0.30;
```

Figure 79: Query 19: SQL Statement.

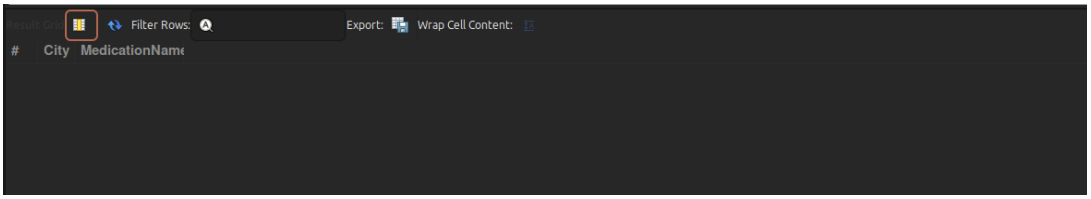


Figure 80: Query 19: Output.

4.20 Data quality check: List stock entries with negative quantity or non-positive unit price

```sql
-- Query 20
SELECT *
FROM Stock
WHERE Qty < 0 OR Unit_Price <= 0;
```

Figure 81: Query 20: SQL Statement.

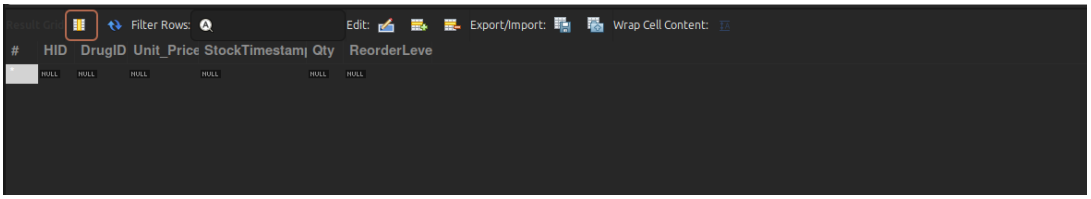


Figure 82: Query 20: Output.