Question 1: Entity Relationship Diagram (ERD)

The goal is to draw a Logical ERD using UML notation, which must include Primary Keys (PKs), Foreign Keys (FKs), and resolve the many-to-many (M:N) relationship[1]. All entities must have surrogate PKs[2].

Steps for the Perfect ERD:

1. **Identify Entities and Attributes:**
   - **Category**: CategoryID (PK, surrogate), Description[3333].
   - **Medication**: MedicationID (PK, surrogate), Name[4].
   - **ActiveIngredient**: ActiveIngredientID (PK, surrogate), Name[5].
2. **Resolve the M:N Relationship (Medication and Active Ingredient):**
   - The relationship is M:N ("one or more active ingredients," "more than one medication") and requires storing a descriptive attribute (Quantity)[6].
   - Create an **Associative Entity**: **MedicationActiveIngredient**.
   - Its Primary Key must be a **Composite Key** made of the two Foreign Keys: MedicationID (PK, FK) and ActiveIngredientID (PK, FK).
   - Add the descriptive attribute: Quantity.
3. **Establish 1:M Relationships and Foreign Keys:**
   - **Medication** and **Category**: Many Medications belong to one Category[7]. Add **CategoryID (FK)** to the Medication entity.
   - **ActiveIngredient** and **Category**: Many Active Ingredients belong to one Category[8]. Add **CategoryID (FK)** to the ActiveIngredient entity.
4. **Determine Multiplicities (Cardinality):**
   - **Category (1)** $\leftrightarrow$ **Medication (1..*)**: *One* Category can have *one or more* Medications[9].
   - **Category (1)** $\leftrightarrow$ **ActiveIngredient (1..*)**: *One* Category can have *one*

or more Active Ingredients.

- **Medication (1)** $\leftrightarrow$ **MedicationActiveIngredient (1..*)**: *One* Medication can contain *one or more* records in the associative table[10].

- **ActiveIngredient (1)** $\leftrightarrow$ **MedicationActiveIngredient (1..*)**: *One* Active Ingredient can be used in *one or more* records in the associative table[11].

Question 2: Normalisation to 2NF and 3NF

This question requires normalizing a 1NF table extract to 2NF (Q.2.1) and then to 3NF (Q.2.2), showing all steps, explanations, and final answers in dependency diagram format[121212].

Initial 1NF Table and Dependencies

To achieve the required 2NF step, we must assume a **Composite Key**. Based on the data's logical purpose, the assumed Composite Primary Key is **(PharmacyID, PharmacistID)**[13].

Attribute	**Dependency**	**Type of Dependency**
Composite Key	(**PharmacyID**, **PharmacistID**)	
Non-Key Attributes	PharmacyName, GroupName, GroupID, PharmacistName	
Partial Dependencies (Violation of 2NF)	**PharmacyID** $\rightarrow$ PharmacyName, GroupID, GroupName (Determined by only part of the key)	Partial Dependency

	PharmacistID $\rightarrow$ PharmacistName (Determined by only part of the key)	Partial Dependency
Transitive Dependency (Violation of 3NF)	GroupID $\rightarrow$ GroupName (A non-key attribute determines another non-key attribute)	Transitive Dependency

Q.2.1: Normalise to Second Normal Form (2NF)

Goal: Resolve **Partial Dependencies**.

Explanation: In 1NF, a non-key attribute is dependent on only *part* of the composite primary key. To reach 2NF, we remove these partial dependencies by creating new tables where the determinant (the part of the key) becomes the new Primary Key[14].

Diagram	Table Name	Relationship	Explanation
Pharmacist (PharmacistID $\rightarrow$ PharmacistName)	**Pharmacist**	PharmacistID (PK) $\rightarrow$ PharmacistName	Creates a table for pharmacist details, removing the partial dependency of PharmacistName on PharmacistID.
Pharmacy (PharmacyID $\rightarrow$ PharmacyName, GroupID, GroupName)	**Pharmacy**	PharmacyID (PK) $\rightarrow$ PharmacyName, GroupID, GroupName	Creates a table for pharmacy location and group details, removing the partial dependency of these attributes on PharmacyID.

Original Table (Junction)	**Pharmacy-Pharmacist**	PharmacyID (PK, FK) + PharmacistID (PK, FK)	The original table is reduced to a junction table to link pharmacies and pharmacists.

Q.2.2: Normalise to Third Normal Form (3NF)

Goal: Resolve **Transitive Dependencies**.

Explanation: In the 2NF Pharmacy table, the non-key attribute GroupID determines the non-key attribute GroupName (GroupID $\rightarrow$ GroupName). This is a transitive dependency, violating 3NF. To resolve it, we create a new table for the transitive relationship and leave a Foreign Key in the original table[15].

Diagram	**Table Name**	**Relationship**	**Explanation**
New Table (Group)	**Group**	GroupID (PK) $\rightarrow$ GroupName	Creates a separate table for the group information, using GroupID as the PK.
Updated Pharmacy Table	**Pharmacy**	PharmacyID (PK) $\rightarrow$ PharmacyName, GroupID (FK)	GroupName is removed, leaving GroupID as a Foreign Key to link to the new Group table.

Final 3NF Dependency Diagrams
Group: GroupID (PK) $\rightarrow$ GroupName
Pharmacist: PharmacistID (PK) $\rightarrow$ PharmacistName

Pharmacy: PharmacyID (PK) $\rightarrow$ PharmacyName, GroupID (FK)	
Pharmacy-Pharmacist: PharmacyID (PK, FK) + PharmacistID (PK, FK)	

Question 3: Practical SQL Script (MySQL)

The solution must be a single SQL script using the corrected statements below. Use comments to indicate which part answers which question[16].

SQL

```sql
-- Ensure the schema is created and used
CREATE SCHEMA IF NOT EXISTS Hospital_db;
USE Hospital_db;

-- Q.3.1: Write an SQL statement to create the Patient table. (5 Marks)
CREATE TABLE Patient(
  PatientID INT AUTO_INCREMENT NOT NULL,
  PatientName VARCHAR(250) NOT NULL, -- Per ERD extract [cite: 257]
  PatientSurname VARCHAR(250) NOT NULL, -- Per ERD extract [cite: 257]
  PatientDOB DATE NOT NULL,
  PRIMARY KEY (PatientID)
);

-- Q.3.2: Write an SQL statement to create the Doctor table. (4 Marks)
CREATE TABLE Doctor(
  DoctorID INT AUTO_INCREMENT NOT NULL,
  DoctorName VARCHAR(250) NOT NULL, -- Per ERD extract [cite: 261]
  DoctorSurname VARCHAR(250) NOT NULL, -- Per ERD extract [cite: 262]
  PRIMARY KEY (DoctorID)
);

-- Q.3.3: Write an SQL statement to create the Appointments table. (9 Marks)
CREATE TABLE Appointments(
  AppointmentID INT AUTO_INCREMENT NOT NULL,
  PatientID INT NOT NULL, -- Foreign Key [cite: 266]
  DoctorID INT NOT NULL,  -- Foreign Key [cite: 267]
  AppointmentDate DATE NOT NULL,
  AppointmentTime VARCHAR(250) NOT NULL, -- Using VARCHAR as per the submitted file for time/duration data [cite: 377]
  AppointmentDuration VARCHAR(250) NOT NULL,
  PRIMARY KEY (AppointmentID),
  FOREIGN KEY (PatientID) REFERENCES Patient(PatientID), -- Links to Patient table
  FOREIGN KEY (DoctorID) REFERENCES Doctor(DoctorID)     -- Links to Doctor table
);

-- Q.3.4: Write SQL statements to insert the provided data. (11 Marks)

-- Insert Patient data [cite: 281, 282]
INSERT INTO Patient (PatientID, PatientName, PatientSurname, PatientDOB) VALUES
(1, 'Debbie', 'Theart', '1980-03-17'),
(2, 'Thomas', 'Duncan', '1976-08-12');

-- Insert Doctor data [cite: 286, 287]
INSERT INTO Doctor (DoctorID, DoctorName, DoctorSurname) VALUES
(1, 'Zintle', 'Nukani'),
(2, 'Ravi', 'Maharaj');

-- Insert Appointments data (Ensure DoctorID and PatientID values are correct based on the data table) [cite: 288, 289]
INSERT INTO Appointments (AppointmentID, AppointmentDate, AppointmentTime,
AppointmentDuration, DoctorID, PatientID) VALUES
(1, '2024-01-15', '9:00', '15', 2, 1), -- Dr 2 (Ravi), Pt 1 (Debbie)
(2, '2024-01-18', '15:00', '30', 2, 2), -- Dr 2 (Ravi), Pt 2 (Thomas)
(3, '2024-01-20', '10:00', '15', 1, 1), -- Dr 1 (Zintle), Pt 1 (Debbie)
(4, '2024-01-21', '11:00', '15', 2, 1); -- Dr 2 (Ravi), Pt 1 (Debbie)

-- Q.3.5: Write an SQL statement to display all appointments between 2024-01-16 and 2024-01-20 (inclusive). (4 Marks)
SELECT *
FROM Appointments
WHERE AppointmentDate BETWEEN '2024-01-16' AND '2024-01-20';

-- Q.3.6: Write a SQL statement to display patient names, surnames, and total appointments, sorted by appointment count (descending). (6 Marks)
SELECT
  P.PatientName,
  P.PatientSurname,
  COUNT(A.AppointmentID) AS NumberOfAppointments
FROM
  Patient P
INNER JOIN
  Appointments A ON P.PatientID = A.PatientID
GROUP BY
  P.PatientID, P.PatientName, P.PatientSurname -- Group by PK and names
ORDER BY
  NumberOfAppointments DESC; -- Sort in descending order [cite: 295]

-- Q.3.7: Write an SQL statement to display all appointments: date (desc), time, doctor's name/surname, patient's name/surname. (10 Marks)
SELECT
  A.AppointmentDate,
  A.AppointmentTime,
  D.DoctorName,
  D.DoctorSurname,
  P.PatientName,
  P.PatientSurname
FROM
  Appointments A
INNER JOIN
  Doctor D ON A.DoctorID = D.DoctorID -- Correct join on DoctorID
INNER JOIN
  Patient P ON A.PatientID = P.PatientID -- Correct join on PatientID
ORDER BY
  A.AppointmentDate DESC; -- Sort in descending order [cite: 298]

-- Q.3.8: Create a view for patients with appointments with Doctor ID 2, displaying name/surname, sorted by surname (asc). (6 Marks)
CREATE VIEW Patients_with_DoctorID2 AS
SELECT
  P.PatientName,
  P.PatientSurname
FROM
  Patient P
INNER JOIN
  Appointments A ON P.PatientID = A.PatientID -- Correct join key
WHERE
  A.DoctorID = 2
ORDER BY
  P.PatientSurname ASC; -- Sort in ascending order [cite: 301]

-- Q.3.9: Create a stored procedure called get_appointments (IN date) to display appointment details
```

```
for that date (time asc, duration, doctor name, patient name). (10 Marks)
DELIMITER //

CREATE PROCEDURE get_appointments (
  IN app_date DATE
)
BEGIN
  SELECT
    A.AppointmentTime, -- In ascending order [cite: 309]
    A.AppointmentDuration,
    D.DoctorName,
    D.DoctorSurname,
    P.PatientName,
    P.PatientSurname
  FROM
    Appointments A
  INNER JOIN
    Doctor D ON D.DoctorID = A.DoctorID
  INNER JOIN
    Patient P ON P.PatientID = A.PatientID
  WHERE
    A.AppointmentDate = app_date -- Filter by the input parameter
  ORDER BY
    A.AppointmentTime ASC; -- Sort by time ascending [cite: 309]
END //

DELIMITER ;
```

Question 4: MongoDB Shell Commands

The solution must be a single text file containing the interactive shell commands[17].

Correct MongoDB Commands:

Question	**Command (Replace <your-student-number>)**	**Explanation**
Q.4.1 (Create Database)	use patients_<your-student-number>	The use command creates the database if it doesn't exist and switches the context to it[18].
Q.4.2 (Create/Insert Data)	db.patients.insertMany([{ "Patient Name": "Debbie", "Patient Surname": "Theart", "Date of Birth": "1980-03-17" }, { "Patient Name": "Thomas", "Patient Surname": "Duncan", "Date of Birth": "1976-08-12" }])	Creates the patients collection and inserts the required data using a single insertMany command[19].
Q.4.3 (Get all notes)	db.patients.find().pretty()	The find() method without arguments returns all documents/notes in the collection[20]. .pretty() formats the output.
Q.4.4 (Query notes born after 1979-01-12)	db.patients.find({ "Date of Birth": { $gt: "1979-01-12" } }).pretty()	Uses the **$gt** operator (greater than) for string comparison, which works correctly for YYYY-MM-DD dates, to find records born after the specified date[21].