**DATABASE DESIGN AND IMPLEMENTATION**

**HOSPITAL DATABASE**

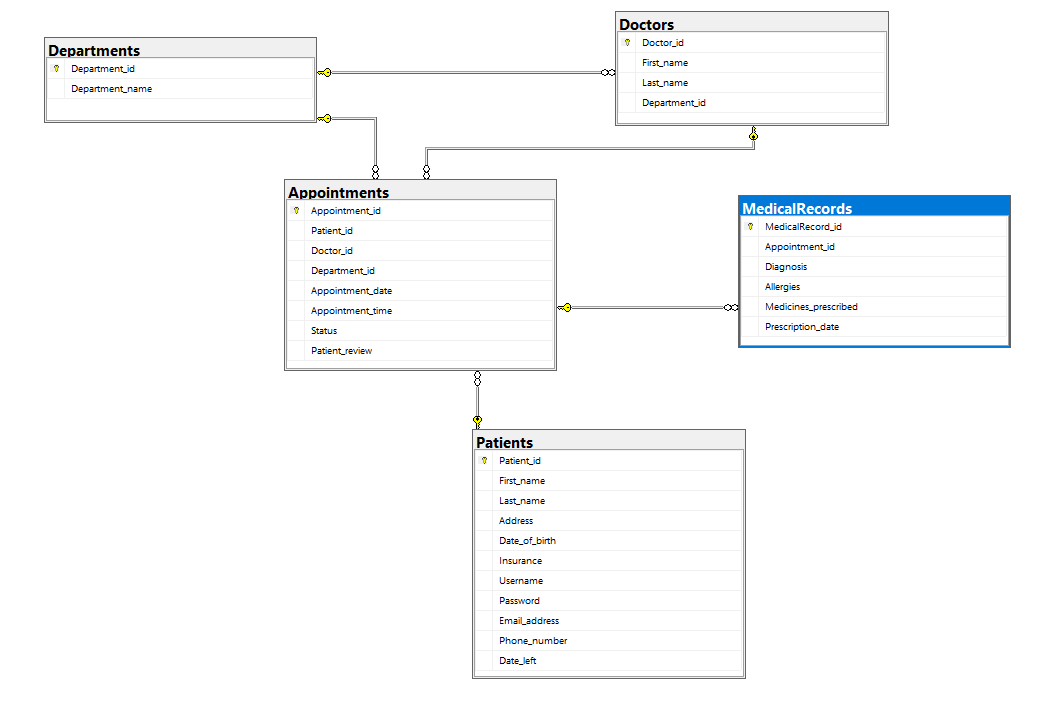
AYEBANENGIYEFA OGAMBA

**INTRODUCTION**

In modern healthcare, effective data management is essential for providing quality care. This report documents the design and implementation of a relational database system for a hospital, focusing on the secure and efficient management of patient, doctor, and appointment data.

Acting as a database consultant, my primary goal was to create a robust system guided by the principles of **normalization**, specifically ensuring the design is in **Third Normal Form (3NF)** to prevent data redundancy and maintain data integrity. The following sections will detail the conceptual schema, the T-SQL code used for implementation, and the practical application of this database in a hospital environment.

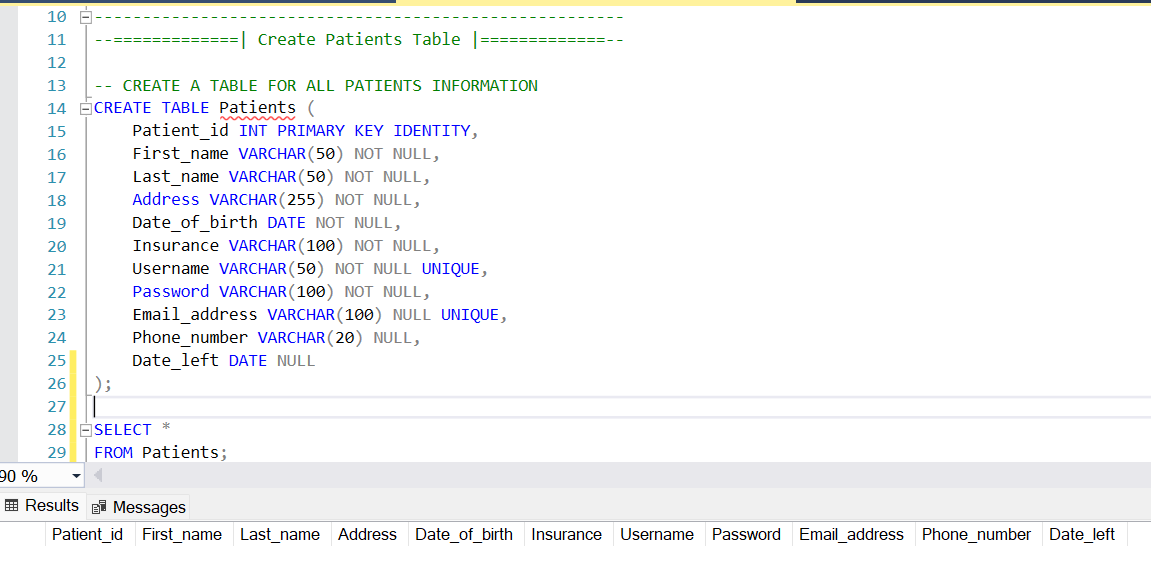
**HOSPITAL DATABASE SCHEMA**



**CREATING TABLES**

Here is a breakdown of my database design:

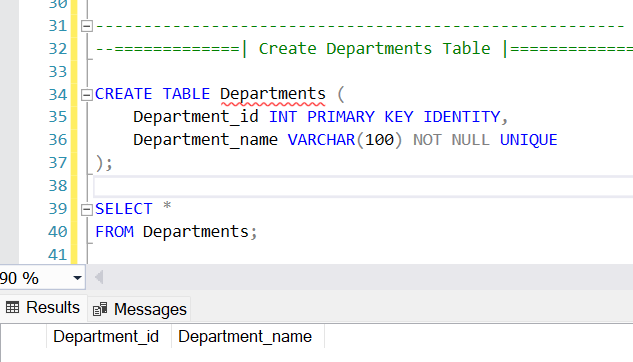
**1. Patients Table**

****

This table is the central repository for all patient information.

* Patient\_id:
  + **Data Type:** INT
  + **Constraint:** PRIMARY KEY IDENTITY
  + **My Justification:** I chose INT as the data type to serve as a numeric, unique identifier for each patient. By adding the PRIMARY KEY IDENTITY constraint, the database automatically generates a unique number for each new patient, ensuring no two records are the same and providing a reliable way to reference each patient.
* First\_name, Last\_name:
  + **Data Type:** VARCHAR(50)
  + **Constraint:** NOT NULL
  + **My Justification:** I opted for VARCHAR(50) to store names, as it's a variable-length string that's efficient for storing names up to 50 characters. The NOT NULL constraint is essential because every patient must have a name on file.
* Address:
  + **Data Type:** VARCHAR(255)
  + **Constraint:** NOT NULL
  + **My Justification:** VARCHAR(255) provides enough space for a full street address. The NOT NULL constraint ensures every patient has a recorded address.
* Date\_of\_birth:
  + **Data Type:** DATE
  + **Constraint:** NOT NULL
  + **My Justification:** The DATE data type is the most precise way to store a birthdate without any time data, making it both accurate and efficient. This is a required field, so I used NOT NULL.
* Insurance:
  + **Data Type:** VARCHAR(100)
  + **Constraint:** None
  + **My Justification:** A VARCHAR(100) field is flexible enough to store various insurance policy numbers or provider names. Since this information might not be immediately available, I allowed it to be NULL.
* Username:
  + **Data Type:** VARCHAR(50)
  + **Constraint:** NOT NULL, UNIQUE
  + **My Justification:** The VARCHAR(50) data type is perfect for a username. The combination of NOT NULL and UNIQUE is crucial because every patient needs a unique username to log in, and it must exist.
* Password:
  + **Data Type:** VARCHAR(100)
  + **Constraint:** NOT NULL
  + **My Justification:** A VARCHAR(100) field is a good size for storing a hashed password, as hashes are typically longer than plain text. The NOT NULL constraint ensures that every patient has a password for their account.
* Email\_address:
  + **Data Type:** VARCHAR(100)
  + **Constraint:** UNIQUE
  + **My Justification:** A VARCHAR(100) is a good choice for email addresses. The UNIQUE constraint prevents two patients from using the same email. Since this is an optional field, I left it as NULL.
* Phone\_number:
  + **Data Type:** VARCHAR(20)
  + **Constraint:** None
  + **My Justification:** This data type can accommodate various phone number formats. Since this is also optional, I allowed it to be NULL.
* Date\_left:
  + **Data Type:** DATE
  + **Constraint:** None
  + **My Justification:** This field is meant to store the date a patient leaves the system, which is not required at the time of registration. I chose a DATE data type and allowed it to be NULL initially.

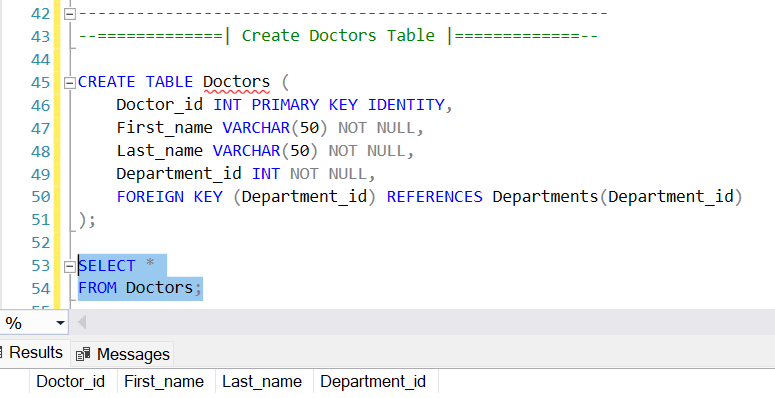
**2. Departments Table**

****

This table is a small, normalized lookup table that centralizes department information to avoid data redundancy in other tables.

* Department\_id:
  + **Data Type:** INT
  + **Constraint:** PRIMARY KEY IDENTITY
  + **My Justification:** This INT serves as a primary key, automatically generating a unique ID for each department.
* Department\_name:
  + **Data Type:** VARCHAR(100)
  + **Constraint:** NOT NULL, UNIQUE
  + **My Justification:** A VARCHAR(100) field stores the department's full name. Both NOT NULL and UNIQUE constraints are vital because every department must have a name, and that name must be unique.

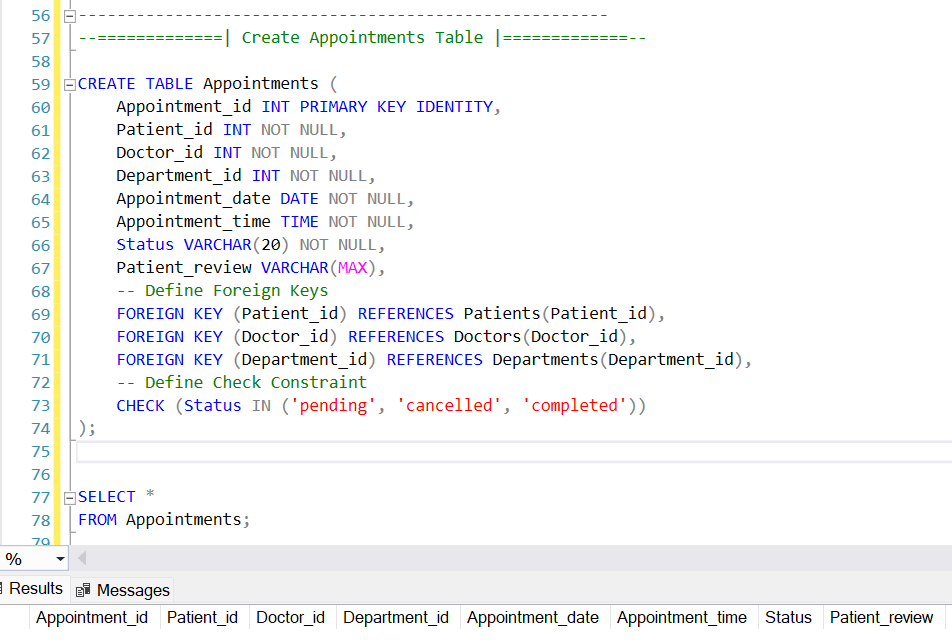
**3. Doctors Table**

****

This table stores information on the doctors and links them to their respective departments.

* Doctor\_id:
  + **Data Type:** INT
  + **Constraint:** PRIMARY KEY IDENTITY
  + **My Justification:** This INT is the unique identifier for each doctor, automatically assigned by the database.
* First\_name, Last\_name:
  + **Data Type:** VARCHAR(50)
  + **Constraint:** NOT NULL
  + **My Justification:** Similar to the Patients table, I used VARCHAR(50) and NOT NULL to ensure every doctor has a first and last name on record.
* Department\_id:
  + **Data Type:** INT
  + **Constraint:** NOT NULL, FOREIGN KEY
  + **My Justification:** I chose INT to match the data type of the primary key in the Departments table. The NOT NULL constraint is critical, as every doctor must be assigned to a department. I used FOREIGN KEY to establish a relationship with the Departments table, ensuring that any department ID entered here actually exists in the Departments table.

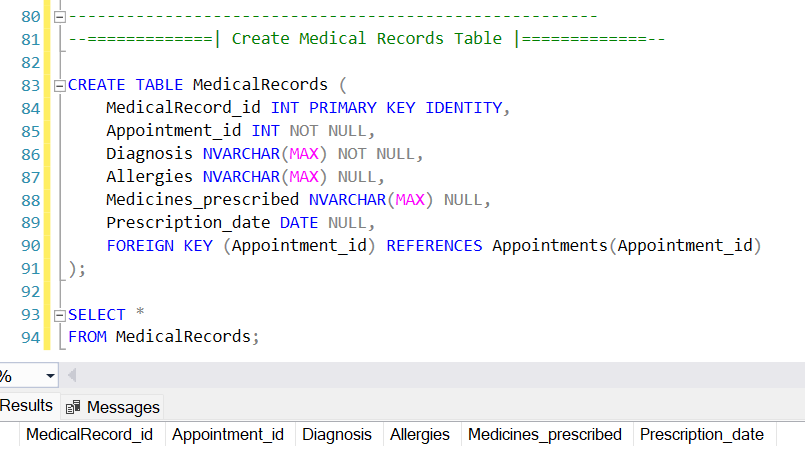
**4. Appointments Table**

****

This table is a bridge between patients, doctors, and departments, recording the details of each appointment.

* Appointment\_id:
  + **Data Type:** INT
  + **Constraint:** PRIMARY KEY IDENTITY
  + **My Justification:** I used INT as the primary key to give each appointment a unique, automatically generated ID.
* Patient\_id, Doctor\_id, Department\_id:
  + **Data Type:** INT
  + **Constraint:** NOT NULL, FOREIGN KEY
  + **My Justification:** These are INT foreign keys linking to the primary keys of the Patients, Doctors, and Departments tables. The NOT NULL constraint is crucial because an appointment must always be linked to a patient, doctor, and department.
* Appointment\_date:
  + **Data Type:** DATE
  + **Constraint:** NOT NULL
  + **My Justification:** A DATE data type is used to store the specific date of the appointment. It's a required field.
* Appointment\_time:
  + **Data Type:** TIME
  + **Constraint:** NOT NULL
  + **My Justification:** The TIME data type is used to store the time of the appointment. It's a required field.
* Status:
  + **Data Type:** VARCHAR(20)
  + **Constraint:** NOT NULL, CHECK
  + **My Justification:** I chose VARCHAR(20) for the status. By adding a CHECK constraint with IN ('pending', 'cancelled', 'completed'), I've ensured that only these three valid status values can be entered into the column. NOT NULL ensures a status is always set.
* Patient\_review:
  + **Data Type:** VARCHAR(MAX)
  + **Constraint:** None
  + **My Justification:** VARCHAR(MAX) is a flexible data type that allows for a variable and potentially very large amount of text, which is ideal for a review or comment field. This is an optional field.

**5. MedicalRecords Table**

****

This table captures specific medical information related to an appointment, like diagnoses and prescriptions.

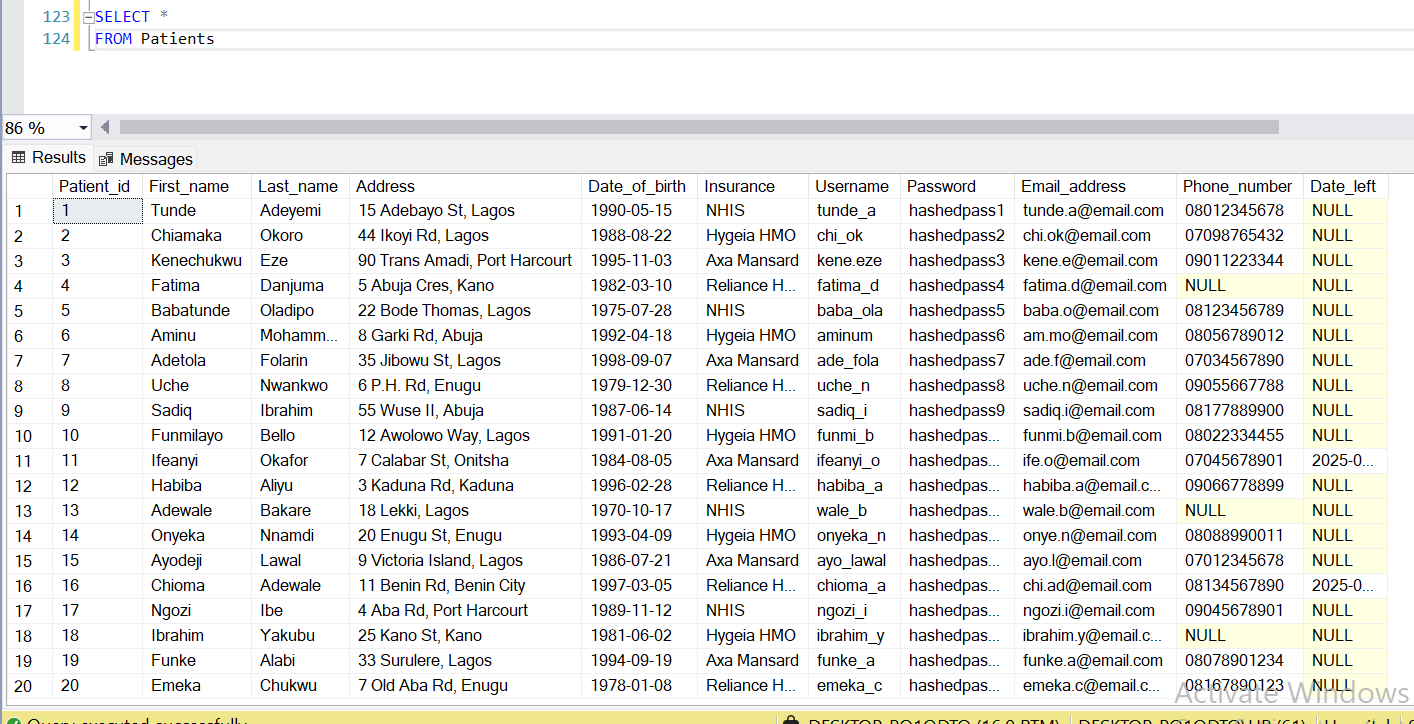
* MedicalRecord\_id:
  + **Data Type:** INT
  + **Constraint:** PRIMARY KEY IDENTITY
  + **My Justification:** This INT acts as the primary key, providing a unique ID for each medical record.
* Appointment\_id:
  + **Data Type:** INT
  + **Constraint:** NOT NULL, FOREIGN KEY
  + **My Justification:** This is an INT foreign key that links a medical record to a specific appointment. The NOT NULL constraint ensures every medical record is tied to an appointment.
* Diagnosis:
  + **Data Type:** NVARCHAR(MAX)
  + **Constraint:** NOT NULL
  + **My Justification:** I chose NVARCHAR(MAX) to accommodate detailed descriptions for a diagnosis. It's a required field, so I used NOT NULL.
* Allergies:
  + **Data Type:** NVARCHAR(MAX)
  + **Constraint:** None
  + **My Justification:** NVARCHAR(MAX) is a flexible type for storing allergy information. Since not every patient has allergies, I allowed this field to be NULL.
* Medicines\_prescribed:
  + **Data Type:** NVARCHAR(MAX)
  + **Constraint:** None
  + **My Justification:** This flexible data type can store one or more prescribed medicines. I allowed it to be NULL as a doctor might not prescribe anything during an appointment.
* Prescription\_date:
  + **Data Type:** DATE
  + **Constraint:** None
  + **My Justification:** I used the DATE data type for the prescription date. It can be NULL if no medicines were prescribed.

**POPULATING THE TABLES**

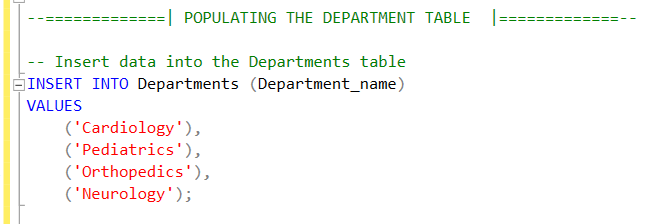
**1. Patients Table**

****

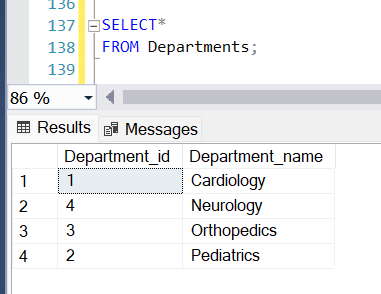
**Result:**



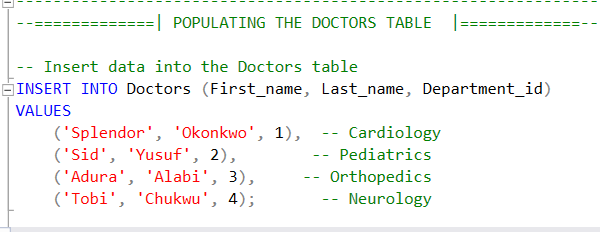
**2. Departments Table**



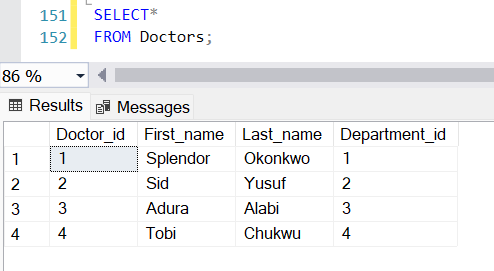
**Result:**

****

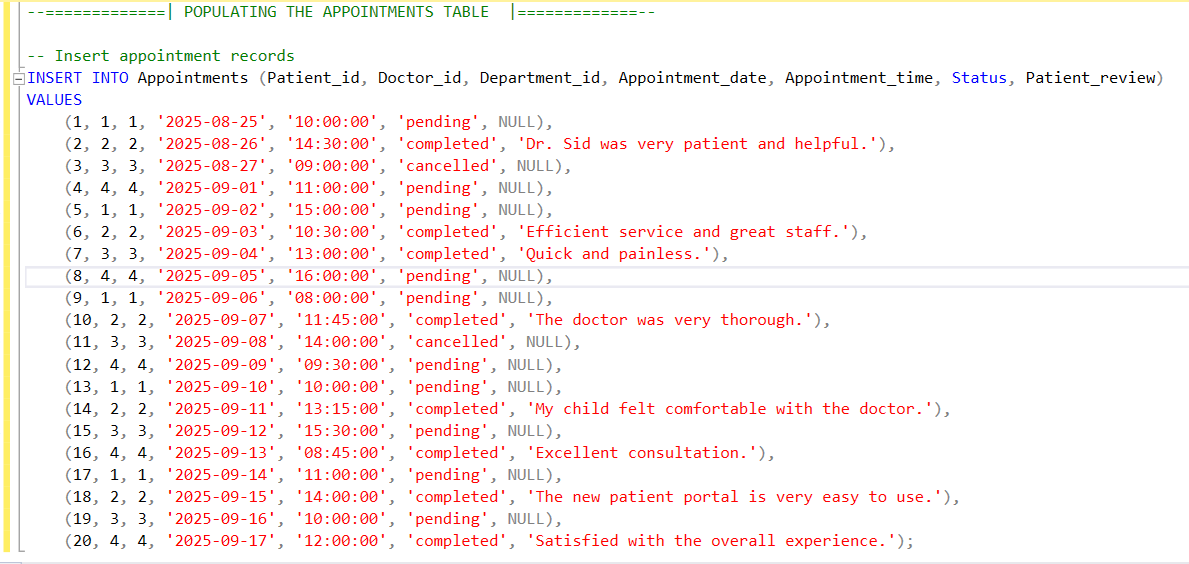
**3. Doctors Table**



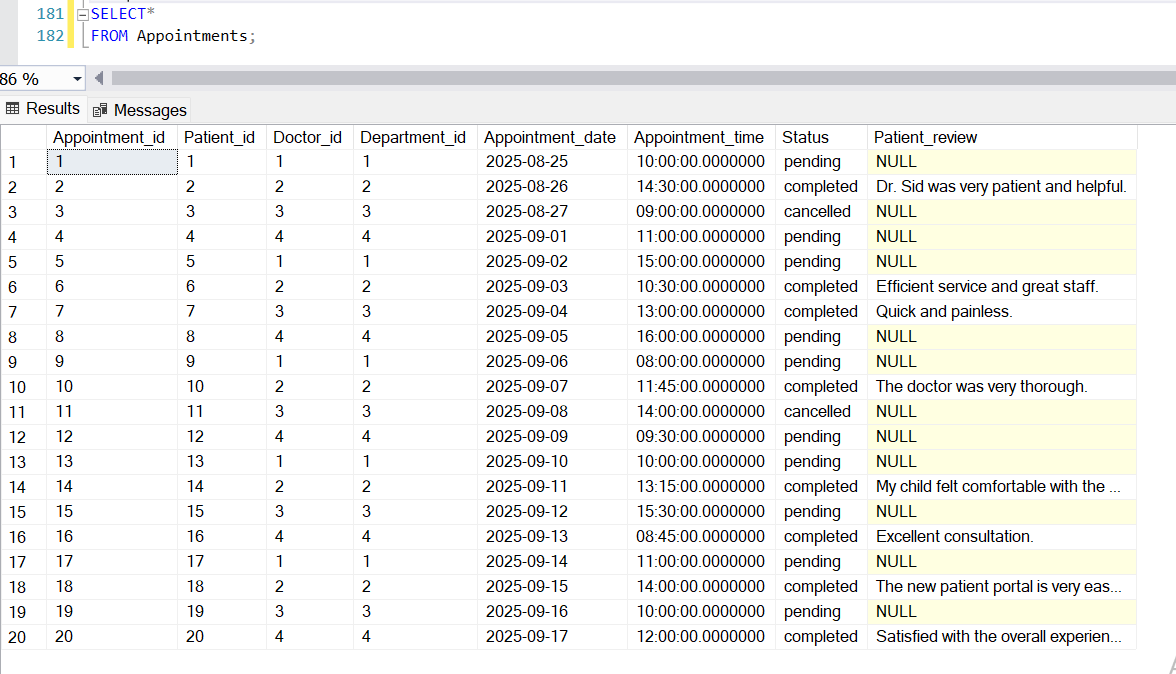
**Result:**



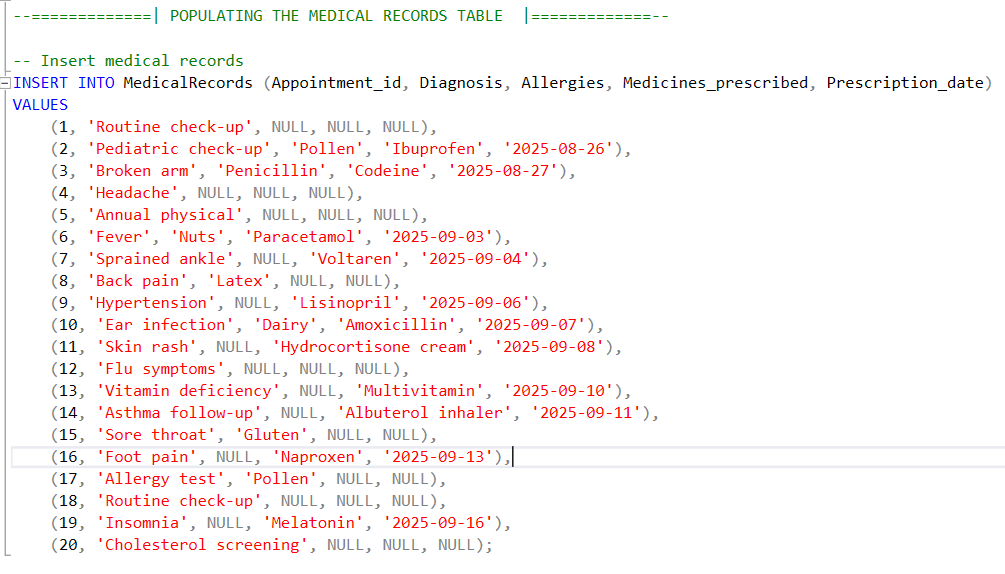
**4. Appointments Table**

****

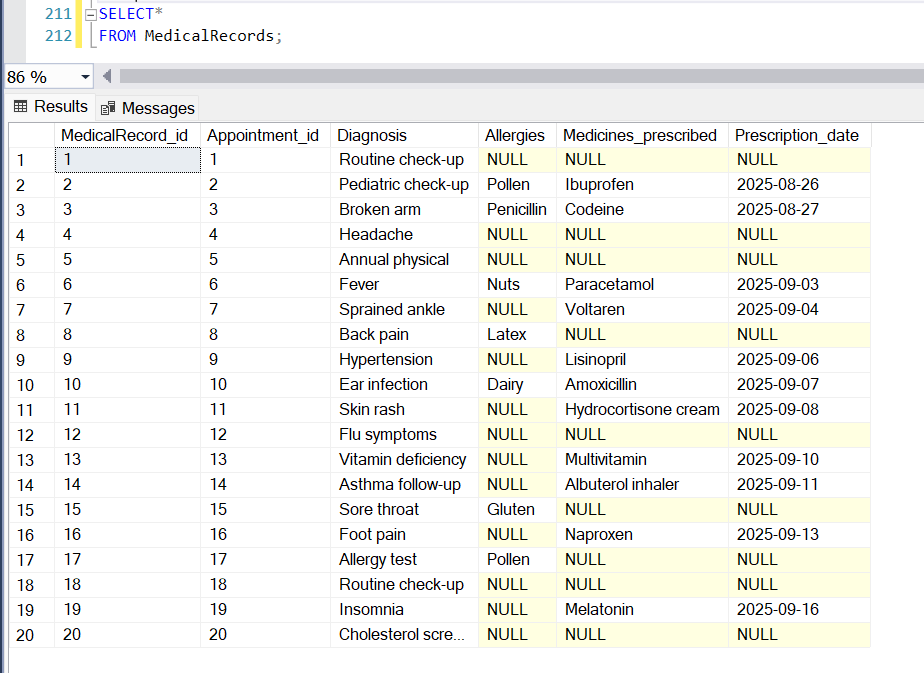
**Result:**



**5. MedicalRecords Table**



**Result:**



**CONCLUSION**

This project successfully delivered a robust and normalized database system for a hospital, designed to manage patients, doctors, medical records, and appointments efficiently. By applying the principles of database normalization, the final schema was designed to be in **Third Normal Form (3NF)**, effectively eliminating data redundancy and ensuring the integrity of the information.

The implementation phase, carried out using **T-SQL** in Microsoft SQL Server Management Studio, demonstrated a practical application of the design. The code for creating tables, defining appropriate **data types**, and establishing crucial **constraints** like primary and foreign keys was executed to build a solid foundation for the hospital's data. Furthermore, the population of the tables with sample data proved the database's functionality and its readiness for real-world use.

The resulting system is not only a secure repository for sensitive medical data but also a scalable and organized solution that supports the hospital’s operational needs. The structure allows for straightforward data retrieval and will be instrumental in improving administrative efficiency and patient care. I have successfully fulfilled all the client's requirements, laying the groundwork for a reliable and high-performing hospital management system.