A logo for a university

Description automatically generated

**UTD- JSOM**

BUAN 6320 - Database Foundations for Business Analytics

Project - Technical Report

Ron Chaudhuri

# **Introduction**

The goal of this project is to design and document a database structure to manage patient records within a hospital. We've created a schema with six main entities: Patient, Doctor, Department, Appointment, and Bill. Each of these entities have been carefully defined to include attributes and relationships that mirror how data would be organized and accessed in a real hospital environment.

In the Entity Relationship Diagram (ERD), we used the Crow’s Feet notation to clearly show relationships and cardinalities between entities. This notation makes it easy to see how each part of the system interacts with others. Additionally, we've normalized the schema to the Third Normal Form (3NF), which helps ensure that data is stored efficiently without unnecessary redundancy, making the database easier to maintain and update.

**Overview**

The Hospital Management System is designed to make managing patients and hospital information easier and more efficient. It helps track important details like patient records, doctor assignments, appointments, bills, and department activities.

This system is built around a powerful database that organizes and stores all the information securely and reliably. It works as a central part of the hospital's operations, connecting directly to systems that handle things like logging in and managing records. It also supports tools like appointment booking and billing, making it easier for staff and patients to access the services they need.

**Literature Review**

“Hospital Management System: A Detailed Guide”, a 2024 blog published by Milit Panchasara describes “a hospital management system [as a] software that allows medical staff to collect, store, retrieve, and share patient information securely across the hospital”. He goes on to state that “the hospital management system (HMS) is such software that provides a seamless approach to managing the operations of an entire hospital and improving patient experience”. Notably, the author goes on to describe the benefits of an HMS system such as efficiency and accuracy; “providing a convenient way to manage patient information and access it universally in the organization, hence reducing administrative burden”.

**Assumptions and Constraints**

**Assumptions**

* Patients can have multiple appointments with the same doctor, but each appointment is unique and tied to a specific date and time to avoid scheduling conflicts.
* Doctors can manage many patients and appointments but are always linked to one department to ensure proper organization.
* Each bill is generated for a single appointment, ensuring a one-to-one relationship between appointments and bills.
* Departments can operate independently of doctors (e.g., for administrative purposes), but every doctor must belong to a department.
* Sensitive patient and doctor information (e.g., phone numbers, addresses) is stored securely to comply with privacy regulations.

**Constraints**

* The **Appointment** and **Bill** entities act as associative entities (join tables) that link Patients, Doctors, and Department data and may not meet the five-attribute minimum guideline for standalone tables.
* Patients can only have one bill per appointment, and each appointment must generate a single bill to maintain clarity in financial transactions.
* Doctors can exist without appointments, but they must be assigned to a department to remain active in the system.
* A patient must always have an assigned doctor to book appointments, ensuring proper care management.
* Appointment times are strictly enforced to prevent overlapping schedules for the same doctor, ensuring accurate availability.

**Design Decisions**

**Key Factors Influencing Design**

The hospital management database was designed using an **entity-relationship model**, as depicted in the ERD. Six entities define the structure of the database:

1. **Patient**: Represents individuals receiving care, including their personal and contact details.
2. **Doctor**: Represents medical staff, with attributes like name, specialty, and start date.
3. **Department**: Represents hospital divisions, defined by name, location, and services.
4. **Appointment**: Captures patient-doctor interactions, with details like date, time, and payment status.
5. **Bill**: Tracks financial transactions tied to appointments, including amount, tax, and payment method.
6. **Relationships**:
   1. Patients are assigned to doctors.
   2. Appointments generate bills.
   3. Doctors belong to departments.

The database includes these views:

* **PatientInfo**: Fetches patient details and their assigned doctor.
* **DoctorInfo**: Provides doctor and department details.
* **DepartmentServices**: Shows department services and doctors.
* **BillingOverview**: Summarizes appointments and bills.

**Functional Design Decisions**

During normal use, the hospital management database will interact with three primary systems to ensure seamless functionality:

1. **Security and Authentication System**:
   1. A login-based authentication system will control access to the database, requiring verified credentials to allow access.
   2. Only authorized users, such as hospital staff, will be able to retrieve or update sensitive data.
2. **Hospital Operations Management System**:
   1. Acts as a "middleman" between users (staff, patients) and the database, interpreting requests and sending SQL commands to the DBMS.
   2. Handles operations such as scheduling appointments, issuing bills, and assigning doctors to patients.
   3. Updates the database automatically in response to events such as new patient registrations, billing transactions, and appointment changes.
3. **File Storage System**:
   1. Assists in storing and retrieving large data files, such as scanned patient records or reports.
   2. Works with the database by using file references (URIs) stored in appropriate tables to ensure efficient file access.

### **Database Management System Decisions**

For the initial implementation, the database will use **MySQL** as the DBMS due to its balance of reliability, performance, and familiarity with the team. The database design focuses on modularity to adapt to future requirements efficiently.

1. **Modularity**:
   1. The database schema is designed to allow easy alterations, such as adding new fields to existing tables using ALTER statements.
   2. Significant changes, such as introducing new entities, will involve revising the entity-relationship model to integrate the new functionality.
2. **Query Language**:
   1. MySQL will use standard SQL for all operations, ensuring portability and compliance with industry standards.
   2. Triggers and stored procedures can be implemented to automate actions like generating bills upon completing appointments.
3. **Scalability**:
   1. The database structure supports growth, handling increasing patient records, appointments, and billing transactions without performance degradation.

### **Security and Privacy Design Decisions**

The system will employ robust security measures to protect sensitive hospital data and ensure user privacy:

1. **Multi-Layer Security**:
   1. **Authentication**: A login-based system verifies user credentials before granting access.
   2. **Mediated Access**: End users (e.g., patients) will access the database only through intermediary systems (e.g., hospital portals), ensuring no direct interaction with the database structure.
2. **User Roles**:
   1. **End Users**: Patients and general staff access the database indirectly via portals or operational systems.
   2. **Internal Users**: Maintenance personnel and administrators have more direct access but are required to work through mediated systems when possible.
   3. **Mediator Systems**: These have full access to the database to perform automated operations such as updating patient records and processing financial transactions.n
3. **Data Encryption**:
   1. All sensitive data, such as patient details and billing information, is encrypted both in transit and at rest to ensure security.
4. **Database Hardening**:
   1. The DBMS is configured to allow only secure, authorized connections, minimizing risks of unauthorized access or tampering.
   2. Audit logs track all database operations for accountability and troubleshooting.

### **Statement of Work**

#### **Overview**

This project oversees the design and creation of a database system for managing patient records and hospital operations. The database will serve as a centralized data store for tracking patients, doctors, appointments, departments, and billing transactions. It is designed to streamline hospital processes by enabling efficient scheduling, patient management, and financial record-keeping. This system aims to improve operational efficiency and ensure better patient care through secure and reliable data management.

#### **Purpose and Objective**

The primary objective of this project is to develop a database system that supports the day-to-day operations of a hospital. The database will:

Store and manage critical data about patients, doctors, departments, appointments, and bills.

* Enable hospital staff to schedule and manage appointments efficiently.
* Ensure accurate and secure financial transactions through detailed billing records.
* Provide flexibility for future scalability, such as adding new fields or integrating advanced features (e.g., analytics or AI-based diagnosis).

The database will act as the backbone of the hospital's information system, facilitating seamless interaction between departments and ensuring data integrity.

#### **Project Scope**

**In-Scope Work**:

* **Project Requirements Documentation**: Clearly outlining the functional and non-functional requirements of the database system.
* **Entity-Relationship Model**: Designing the schema with six main entities (Patient, Doctor, Appointment, Department, Bill) and their relationships.
* **DDL Scripts**: Writing SQL scripts to create and define database tables, constraints, and relationships.
* **Example DML Scripts**: Providing sample data manipulation scripts for adding, updating, and deleting records.
* **Example SQL Scripts**: Demonstrating common queries for operations like fetching patient details, appointment schedules, or billing summaries.
* **Comprehensive Report**: Delivering a detailed project report, including assumptions, constraints, and key design considerations.

**Out-of-Scope Work**:

* Development of patient-facing interfaces (e.g., appointment booking system).
* Implementation of middleware or APIs for external system integration.
* Advanced analytics or reporting tools.

#### **Database Goals, Expectations, and Deliverables**

Upon completion, the database will:

* Contain fields for all essential hospital operations, such as patient details, appointment schedules, and billing records.
* Implement primary and foreign key constraints to ensure data integrity.
* Be normalized to Third Normal Form (3NF) to minimize redundancy and optimize storage.

**Deliverables**:

* Entity-Relationship Diagram (ERD): Visualizing entities, attributes, and relationships.
* DDL Scripts: Defining the database schema.
* Example DML and SQL Scripts: Demonstrating database functionality.
* Final Project Report: Summarizing the project and its outcomes.

#### **Database Benefits**

The hospital management database will bring the following benefits:

* **Efficiency**: Simplify operations like scheduling, billing, and patient management, reducing administrative workload.
* **Data Security**: Safeguard sensitive patient and financial information with robust access controls and encryption.
* **Scalability**: Support growing patient records and hospital data without impacting performance.
* **Interoperability**: Provide a foundation for future integration with hospital systems, such as patient portals or analytics platforms.

#### **Project Hardware and Software Tools**

**Diagram Tool**

ER-Assistant

**Office Productivity Tools**

Microsoft Office 365

**Database**

pgAdmin 4

**Hardware and Software**

Computers capable of running the applications above

**Client Access Method**

In-game level browser, on a consumer-class computer running Windows or macOS

#### **SQL Usage and Style**

*Adapted from Simon Holywell’s SQL style guide, available at* [*http://sqlstyle.guide/*](http://sqlstyle.guide/)*.*

**General**

* Use consistent and descriptive identifiers and names
* Make judicious use of white space and indentation to make code easier to read
* Keep code succinct and devoid of redundant SQL
* Plurals, use the more natural collective term where possible
* Ensure the name is unique and does not exist as a reserved keyword
* Avoid abbreviations and if you must use them make sure they are commonly understood
* Avoid simply using id as the primary identifier for the table
* Use lowercase except where it may make sense not to
* Include the AS keyword – make it easier to read as it is explicit
* The name must contain a verb

**Query Syntax**

* Always use uppercase for the reserved keywords like SELECT and WHERE
* It is best to avoid the abbreviated keywords and use the full-length ones where available
* To make the code easier to read it is important that the correct complement of spacing is used
* Spaces should be used to line up the code so that the root keywords all end on the same character boundary
* Include newlines/vertical space

**Create Syntax**

* Important to maintain human-readable code
* Deciding the column(s) that will form the keys in the definition should be a carefully considered activity as it will affect performance and data integrity
* Tables must have at least one key to be complete and useful
* Constraints should be given a custom name excepting UNIQUE, PRIMARY KEY, and FOREIGN KEY
* Avoid splitting up data that should be in one table across many tables

#### **Project Management Methodology**

The development of the hospital management database system will adopt a phased project management approach that begins with a **linear process** for the initial design and transitions to an **iterative methodology** for subsequent development and refinement.

*Initial Design Phase*

The early implementation of the database will follow a linear approach, similar to the **waterfall model**. This phase focuses on:

* Meeting the initial requirements defined at the outset of the project.
* Delivering a baseline database schema that satisfies core functionalities, such as patient management, doctor assignments, and appointment scheduling.
* Creating foundational deliverables, including the entity-relationship diagram (ERD) and data definition language (DDL) scripts.

This phase ensures that the primary requirements are met before transitioning to a more flexible development process.

*Iterative Development Phase*

Following the completion of the initial database design, the project will shift to a **rapid iteration methodology** to adapt to evolving requirements. This phase emphasizes:

1. **Collaboration**:
   1. Aligning with the hospital's IT and operational teams to ensure database updates align with the overall system's evolving needs.
   2. Maintaining regular communication with any software development teams to ensure seamless integration between the database and external systems, such as patient portals or billing applications.
2. **Flexibility**:
   1. Revising the database schema iteratively to accommodate feedback and changing requirements, such as adding new fields or modifying relationships.
   2. Supporting quick updates to the database using agile principles, ensuring incremental improvements without disrupting existing functionality.
3. **Feedback-Driven Refinements**:
   1. Incorporating feedback from end-users, such as hospital staff, to improve usability and data accessibility.
   2. Using real-world testing data to validate the database design and identify areas for improvement.

*Benefits of Methodology*

1. **Structured Start**: The waterfall approach ensures a solid foundation, reducing the risk of significant redesigning later.
2. **Adaptability**: Iterative development allows for the database to evolve alongside the hospital's operational needs.
3. **Team Synergy**: Using a methodology aligned with the hospital's broader IT strategy fosters better communication and collaboration across teams.

**Requirements Definition Document**

**Entity Descriptions and Attributes**

1. **Patient**
   * **Primary Key**: Patient\_ID
   * **Attributes**:
     + **Patient\_Fname**: First name of the patient.
     + **Patient\_Lname**: Last name of the patient.
     + **DOB**: Date of birth of the patient.
     + **Address**: Address of the patient.
     + **ER\_Number**: Emergency room contact number.
2. **Doctor**
   * **Primary Key**: Doctor\_ID
   * **Attributes**:
     + **Doctor\_Fname**: First name of the doctor.
     + **Doctor\_Lname**: Last name of the doctor.
     + **Specialty**: Area of specialization (e.g., cardiology, pediatrics).
     + **Doctor\_Num**: Contact number for the doctor.
     + **Start\_Date**: Date when the doctor joined the hospital.
3. **Department**
   * **Primary Key**: Dept\_ID
   * **Attributes**:
     + **Dept\_Name**: Name of the department.
     + **Dept\_Location**: Physical location within the hospital.
     + **Services**: Services provided by the department.
     + **Dept\_Phone**: Contact phone number for the department.
4. **Appointment**
   * **Primary Key**: App\_ID
   * **Attributes**:
     + **App\_Date**: Date of the appointment.
     + **App\_Time**: Time of the appointment.
     + **Visit\_Reason**: Reason for the patient’s visit.
     + **Status**: Status of the appointment (e.g., scheduled, completed).
     + **Payment**: Payment status for the appointment.
5. **Bill**
   * **Primary Key**: Bill\_ID
   * **Attributes**:
     + **Bill\_Amt**: Total amount of the bill.
     + **Tax**: Tax applied to the bill amount.
     + **Discount**: Any discount provided on the bill.
     + **Pay\_Status**: Payment status (e.g., paid, pending).
     + **Pay\_Method**: Payment method (e.g., cash, card).

**Foreign Keys Information**

1. **Patient Table**
   * **Foreign Key**: Doctor\_ID
   * **Related Entity**: Doctor
2. **Doctor Table**
   * **Foreign Key**: Dept\_ID
   * **Related Entity**: Department
3. **Appointment Table**
   * **Foreign Keys**:
     + Patient\_ID (Related Entity: Patient)
     + Doctor\_ID (Related Entity: Doctor)
4. **Bill Table**
   * **Foreign Key**: App\_ID
   * **Related Entity**: Appointment

**Relationships and Cardinalities**

* **Patient - Doctor (Doctored)**
* **Relationship**: Each patient is assigned a doctor, and each doctor can have multiple patients.
* **Cardinality**: M:1

(Many patients to One doctor).

* **Business Rule**: Every patient must have an assigned doctor, but a doctor can manage multiple patients.
* **Mandatory/Optional**: Mandatory for patients to have an assigned doctor; a doctor can exist without patients (optional from the doctor’s side).
* **Patient - Appointment (Consultation)**
* **Relationship**: Each patient must have at least one appointment.
* **Cardinality**: 1:M

(One patient to many appointments).

* **Business Rule**: Every appointment is associated with one patient, but each patient can have multiple appointments.
* **Mandatory/Optional**: Mandatory for patients to have at least one appointment; each appointment must be associated with a patient (mandatory from both sides).
* **Patient - Bill (Cost)**
* **Relationship**: Each patient will be issued a bill.
* **Cardinality**: 1:1

(One patient to one bill).

* **Business Rule**: Every patient is associated with one bill and each bill will have one patient for sure.
* **Mandatory/Optional**: Mandatory for patients to have one bill; each bill must be associated with a patient (mandatory from both sides).
* **Doctor - Appointment (Examination)**
* **Relationship**: A doctor can have multiple appointments, but each appointment is associated with only one doctor.
* **Cardinality**: 1:M

(One doctor to many appointments).

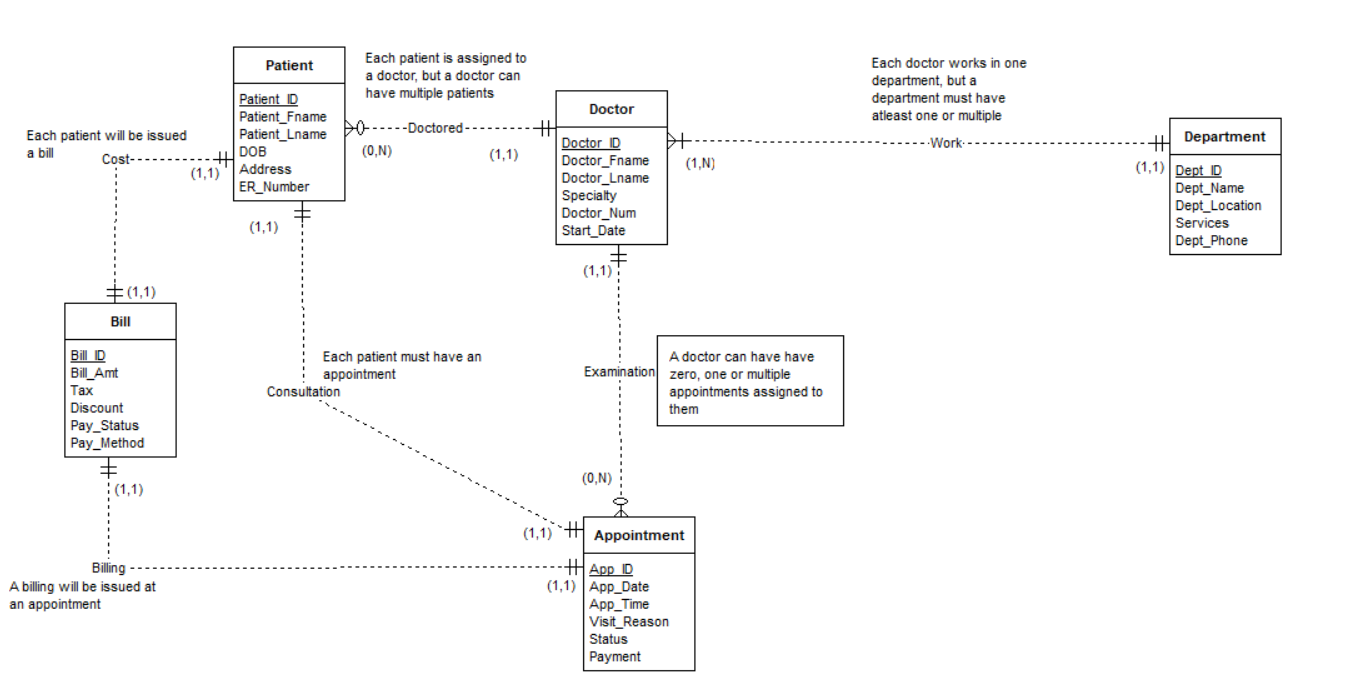
* **Business Rule**: Each appointment involves one doctor, but doctors can handle multiple appointments.
* **Mandatory/Optional**: Mandatory for each appointment to be associated with a doctor; a doctor can exist without any appointments (optional from the doctor’s side).
* **Appointment - Bill (Billing)**
* **Relationship**: Each appointment results in a bill, and each bill is associated with one appointment.
* **Cardinality**: 1:1.
* **Business Rule**: For every appointment, a bill is generated.
* **Mandatory/Optional**: Mandatory for both sides; every appointment must have a bill, and each bill is linked to a specific appointment.
* **Doctor - Department (Work)**
* **Relationship**: Each doctor works in one department, but a department can have multiple doctors.
* **Cardinality**: M:1

(Many doctors to one department).

* **Business Rule**: Departments can employ multiple doctors, and each doctor is assigned to one department.
* **Mandatory/Optional**: Mandatory for doctors to be assigned to a department; departments can exist without doctors (optional from the department’s side).

**Detailed Database Design**

**Entity Relationship Diagram**



**DDL Source Code**

set search\_path to hospital

/\* DROP statements \*/

-- Drop Triggers

DROP TRIGGER IF EXISTS TRG\_Department ON Department;

DROP TRIGGER IF EXISTS TRG\_Doctor ON Doctor;

DROP TRIGGER IF EXISTS TRG\_Patient ON Patient;

DROP TRIGGER IF EXISTS TRG\_Appointment ON Appointment;

DROP TRIGGER IF EXISTS TRG\_Bill ON Bill;

-- Drop Trigger Functions

DROP FUNCTION IF EXISTS dept\_trigger\_function() CASCADE;

DROP FUNCTION IF EXISTS doctor\_trigger\_function() CASCADE;

DROP FUNCTION IF EXISTS patient\_trigger\_function() CASCADE;

DROP FUNCTION IF EXISTS set\_default\_appointment\_values () CASCADE;

DROP FUNCTION IF EXISTS bill\_trigger\_function() CASCADE;

-- Drop Sequences

DROP SEQUENCE IF EXISTS SEQ\_Department\_Dept\_ID;

DROP SEQUENCE IF EXISTS SEQ\_Doctor\_Doctor\_ID;

DROP SEQUENCE IF EXISTS SEQ\_Patient\_Patient\_ID;

DROP SEQUENCE IF EXISTS SEQ\_Appointment\_App\_ID;

DROP SEQUENCE IF EXISTS SEQ\_Bill\_Bill\_ID;

-- Drop Altered Table Columns and Constraints

ALTER TABLE Department DROP COLUMN IF EXISTS Dept\_Head;

ALTER TABLE Department DROP CONSTRAINT IF EXISTS valid\_dept\_phone;

ALTER TABLE Doctor DROP CONSTRAINT IF EXISTS unique\_doctor\_num;

ALTER TABLE Doctor DROP COLUMN IF EXISTS Doctor\_Email;

ALTER TABLE Doctor DROP CONSTRAINT IF EXISTS doctor\_dept\_id\_fkey;

ALTER TABLE Patient DROP COLUMN IF EXISTS Patient\_Phone;

ALTER TABLE Patient DROP CONSTRAINT IF EXISTS patient\_doctor\_id\_fkey;

ALTER TABLE Appointment DROP CONSTRAINT IF EXISTS valid\_status;

ALTER TABLE Appointment DROP CONSTRAINT IF EXISTS valid\_payment;

ALTER TABLE Appointment DROP CONSTRAINT IF EXISTS appointment\_doctor\_id\_fkey;

ALTER TABLE Appointment DROP CONSTRAINT IF EXISTS appointment\_patient\_id\_fkey;

ALTER TABLE Bill DROP COLUMN IF EXISTS Pay\_Due\_Date;

ALTER TABLE Bill DROP CONSTRAINT IF EXISTS bill\_app\_id\_fkey;

-- Drop Views

DROP VIEW IF EXISTS view\_patient\_doctors;

DROP VIEW IF EXISTS view\_patient\_history;

DROP VIEW IF EXISTS view\_upcoming\_appointments;

DROP VIEW IF EXISTS view\_unpaid\_bills;

-- Drop Tables

DROP TABLE IF EXISTS Department CASCADE;

DROP TABLE IF EXISTS Doctor CASCADE;

DROP TABLE IF EXISTS Patient CASCADE;

DROP TABLE IF EXISTS Appointment CASCADE;

DROP TABLE IF EXISTS Bill CASCADE;

/\* CREATE statements \*/

-- Create Department table

CREATE TABLE Department (

Dept\_ID INT PRIMARY KEY,

Dept\_Name VARCHAR(100),

Dept\_Location VARCHAR(100),

Services VARCHAR(255),

Dept\_Phone VARCHAR(15)

);

-- Create Doctor table

CREATE TABLE Doctor (

Doctor\_ID INT PRIMARY KEY,

Doctor\_Fname VARCHAR(100),

Doctor\_Lname VARCHAR(100),

Specialty VARCHAR(100),

Doctor\_Num VARCHAR(15),

Start\_Date DATE,

Dept\_ID INT,

FOREIGN KEY (Dept\_ID) REFERENCES Department(Dept\_ID)

);

-- Create Patient table

CREATE TABLE Patient (

Patient\_ID INT PRIMARY KEY,

Patient\_Fname VARCHAR(100),

Patient\_Lname VARCHAR(100),

DOB DATE,

Address VARCHAR(255),

ER\_Number VARCHAR(15),

Doctor\_ID INT,

FOREIGN KEY (Doctor\_ID) REFERENCES Doctor(Doctor\_ID)

);

-- Create Appointment table

CREATE TABLE Appointment (

App\_ID INT PRIMARY KEY,

App\_Date DATE,

App\_Time TIME,

Visit\_Reason VARCHAR(255),

Status VARCHAR(50),

Payment VARCHAR(50),

Patient\_ID INT,

Doctor\_ID INT,

FOREIGN KEY (Patient\_ID) REFERENCES Patient(Patient\_ID),

FOREIGN KEY (Doctor\_ID) REFERENCES Doctor(Doctor\_ID)

);

-- Create Bill table

CREATE TABLE Bill (

Bill\_ID INT PRIMARY KEY,

Bill\_Amt DECIMAL(10, 2),

Tax DECIMAL(10, 2),

Discount DECIMAL(10, 2),

Pay\_Status VARCHAR(50),

Pay\_Method VARCHAR(50),

App\_ID INT,

FOREIGN KEY (App\_ID) REFERENCES Appointment(App\_ID)

);

/\* Alter Tables to add constraints and columns \*/

-- Department table constraints

ALTER TABLE Department

ADD COLUMN Dept\_Head VARCHAR(100);

ALTER TABLE Department

ADD CONSTRAINT valid\_dept\_phone CHECK (LENGTH(Dept\_Phone) <= 15);

-- Doctor table constraints

ALTER TABLE Doctor

ADD CONSTRAINT unique\_doctor\_num UNIQUE (Doctor\_Num);

ALTER TABLE Doctor

ADD COLUMN Doctor\_Email VARCHAR(100);

-- Patient table constraints

ALTER TABLE Patient

ADD COLUMN Patient\_Phone VARCHAR(15);

-- Appointment table constraints

ALTER TABLE Appointment

ADD CONSTRAINT valid\_status CHECK (Status IN ('Scheduled', 'Completed', 'Cancelled'));

ALTER TABLE Appointment

ADD CONSTRAINT valid\_payment CHECK (Payment IN ('Paid', 'Pending', 'Unpaid'));

-- Bill table constraints

ALTER TABLE Bill

ADD COLUMN Pay\_Due\_Date DATE;

/\* CREATE Views \*/

-- View to show patients and their assigned doctors

CREATE VIEW view\_patient\_doctors AS

SELECT

Patient\_ID,

Patient\_Fname || ' ' || Patient\_Lname AS Patient\_Name,

Doctor\_ID,

ER\_Number

FROM Patient;

-- View to list upcoming appointments

CREATE VIEW view\_upcoming\_appointments AS

SELECT

App\_ID,

App\_Date,

App\_Time,

Visit\_Reason,

Patient\_ID,

Doctor\_ID,

Status

FROM Appointment

WHERE Status = 'Scheduled';

-- View to show unpaid bills

CREATE VIEW view\_unpaid\_bills AS

SELECT

Bill\_ID,

Bill\_Amt,

Tax,

Discount,

Pay\_Status,

App\_ID

FROM Bill

WHERE Pay\_Status = 'Pending';

-- View to show patient history of completed appointments

CREATE VIEW view\_patient\_history AS

SELECT

Patient\_ID,

App\_ID,

App\_Date,

Visit\_Reason,

Status

FROM Appointment

WHERE Status = 'Completed';

/\* CREATE Sequences \*/

-- Sequence for Department

CREATE SEQUENCE SEQ\_Department\_Dept\_ID

START WITH 101 INCREMENT BY 1;

-- Sequence for Appointment

CREATE SEQUENCE SEQ\_Appointment\_App\_ID

START WITH 11 INCREMENT BY 1;

-- Sequence for Doctor

CREATE SEQUENCE SEQ\_Doctor\_Doctor\_ID

START WITH 1 INCREMENT BY 1;

-- Sequence for Patient

CREATE SEQUENCE SEQ\_Patient\_Patient\_ID

START WITH 1001 INCREMENT BY 1;

-- Sequence for Bill

CREATE SEQUENCE SEQ\_Bill\_Bill\_ID

START WITH 11 INCREMENT BY 1;

/\* CREATE Triggers \*/

-- Trigger for Department

CREATE OR REPLACE FUNCTION dept\_trigger\_function()

RETURNS TRIGGER AS $$

BEGIN

IF NEW.Dept\_ID IS NULL THEN

NEW.Dept\_ID = NEXTVAL('SEQ\_Department\_Dept\_ID');

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER TRG\_Department

BEFORE INSERT ON Department

FOR EACH ROW

EXECUTE FUNCTION dept\_trigger\_function();

-- Trigger for Appointment

CREATE OR REPLACE FUNCTION set\_default\_appointment\_values()

RETURNS TRIGGER AS $$

BEGIN

IF NEW.App\_ID IS NULL THEN

NEW.App\_ID = NEXTVAL('SEQ\_Appointment\_App\_ID');

END IF;

IF NEW.App\_Date IS NULL THEN

NEW.App\_Date = CURRENT\_DATE + INTERVAL '7 days';

END IF;

IF NEW.App\_Time IS NULL THEN

NEW.App\_Time = '14:00';

END IF;

IF NEW.Payment IS NULL THEN

NEW.Payment = 'Paid';

END IF;

IF NEW.Status IS NULL THEN

NEW.Status = 'Scheduled';

END IF;

IF NEW.Visit\_Reason IS NULL THEN

NEW.Visit\_Reason = 'Consultation';

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER TRG\_Appointment

BEFORE INSERT ON Appointment

FOR EACH ROW

EXECUTE FUNCTION set\_default\_appointment\_values();

-- Trigger for Doctor

CREATE OR REPLACE FUNCTION doctor\_trigger\_function()

RETURNS TRIGGER AS $$

BEGIN

IF NEW.Doctor\_ID IS NULL THEN

NEW.Doctor\_ID = NEXTVAL('SEQ\_Doctor\_Doctor\_ID');

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER TRG\_Doctor

BEFORE INSERT ON Doctor

FOR EACH ROW

EXECUTE FUNCTION doctor\_trigger\_function();

-- Trigger for Patient

CREATE OR REPLACE FUNCTION patient\_trigger\_function()

RETURNS TRIGGER AS $$

BEGIN

IF NEW.Patient\_ID IS NULL THEN

NEW.Patient\_ID = NEXTVAL('SEQ\_Patient\_Patient\_ID');

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER TRG\_Patient

BEFORE INSERT ON Patient

FOR EACH ROW

EXECUTE FUNCTION patient\_trigger\_function();

-- Trigger for Bill

CREATE OR REPLACE FUNCTION bill\_trigger\_function()

RETURNS TRIGGER AS $$

BEGIN

IF NEW.Bill\_ID IS NULL THEN

NEW.Bill\_ID = NEXTVAL('SEQ\_Bill\_Bill\_ID');

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER TRG\_Bill

BEFORE INSERT ON Bill

FOR EACH ROW

EXECUTE FUNCTION bill\_trigger\_function();

**DML and Query Source Code**

--Inserting data into the tables

INSERT INTO Department (Dept\_ID, Dept\_Name, Dept\_Location, Services, Dept\_Phone, Dept\_Head)

VALUES

(NEXTVAL('SEQ\_Department\_Dept\_ID'), 'Cardiology', 'Building A', 'Heart health', '1234567890', 'Dr. Alice Johnson'),

(NEXTVAL('SEQ\_Department\_Dept\_ID'), 'Neurology', 'Building B', 'Brain health', '2345678901', 'Dr. Bob Smith'),

(NEXTVAL('SEQ\_Department\_Dept\_ID'), 'Orthopedics', 'Building C', 'Bone health', '3456789012', 'Dr. Carol Davis'),

(NEXTVAL('SEQ\_Department\_Dept\_ID'), 'Pediatrics', 'Building D', 'Child health', '4567890123', 'Dr. Daniel Thompson'),

(NEXTVAL('SEQ\_Department\_Dept\_ID'), 'Dermatology', 'Building E', 'Skin health', '5678901234', 'Dr. Evelyn Clarke'),

(NEXTVAL('SEQ\_Department\_Dept\_ID'), 'Oncology', 'Building F', 'Cancer treatment', '6789012345', 'Dr. Frank Harris'),

(NEXTVAL('SEQ\_Department\_Dept\_ID'), 'Gastroenterology', 'Building G', 'Digestive health', '7890123456', 'Dr. Grace Adams'),

(NEXTVAL('SEQ\_Department\_Dept\_ID'), 'Psychiatry', 'Building H', 'Mental health', '8901234567', 'Dr. Hank Brown'),

(NEXTVAL('SEQ\_Department\_Dept\_ID'), 'Radiology', 'Building I', 'Imaging services', '9012345678', 'Dr. Ian White'),

(NEXTVAL('SEQ\_Department\_Dept\_ID'), 'Ophthalmology', 'Building J', 'Eye health', '0123456789', 'Dr. Jill Wilson');

INSERT INTO Doctor (Doctor\_ID, Doctor\_Fname, Doctor\_Lname, Specialty, Doctor\_Num, Start\_Date, Dept\_ID, Doctor\_Email)

VALUES

(NEXTVAL('SEQ\_Doctor\_Doctor\_ID'), 'Alice', 'Johnson', 'Cardiology', 'D001', '2020-01-15', 101, '[alice.johnson@hospital.com](mailto:alice.johnson@hospital.com)'),

(NEXTVAL('SEQ\_Doctor\_Doctor\_ID'), 'Bob', 'Smith', 'Neurology', 'D002', '2019-06-01', 102, '[bob.smith@hospital.com](mailto:bob.smith@hospital.com)'),

(NEXTVAL('SEQ\_Doctor\_Doctor\_ID'), 'Carol', 'Davis', 'Orthopedics', 'D003', '2018-09-21', 103, '[carol.davis@hospital.com](mailto:carol.davis@hospital.com)'),

(NEXTVAL('SEQ\_Doctor\_Doctor\_ID'), 'Daniel', 'Thompson', 'Pediatrics', 'D004', '2021-03-12', 104, '[daniel.thompson@hospital.com](mailto:daniel.thompson@hospital.com)'),

(NEXTVAL('SEQ\_Doctor\_Doctor\_ID'), 'Evelyn', 'Clarke', 'Dermatology', 'D005', '2017-12-01', 105, '[evelyn.clarke@hospital.com](mailto:evelyn.clarke@hospital.com)'),

(NEXTVAL('SEQ\_Doctor\_Doctor\_ID'), 'Frank', 'Harris', 'Oncology', 'D006', '2022-01-25', 106, '[frank.harris@hospital.com](mailto:frank.harris@hospital.com)'),

(NEXTVAL('SEQ\_Doctor\_Doctor\_ID'), 'Grace', 'Adams', 'Gastroenterology', 'D007', '2020-07-15', 107, '[grace.adams@hospital.com](mailto:grace.adams@hospital.com)'),

(NEXTVAL('SEQ\_Doctor\_Doctor\_ID'), 'Hank', 'Brown', 'Psychiatry', 'D008', '2021-08-05', 108, '[hank.brown@hospital.com](mailto:hank.brown@hospital.com)'),

(NEXTVAL('SEQ\_Doctor\_Doctor\_ID'), 'Ian', 'White', 'Radiology', 'D009', '2019-04-17', 109, '[ian.white@hospital.com](mailto:ian.white@hospital.com)'),

(NEXTVAL('SEQ\_Doctor\_Doctor\_ID'), 'Jill', 'Wilson', 'Ophthalmology', 'D010', '2022-09-10', 110, '[jill.wilson@hospital.com](mailto:jill.wilson@hospital.com)');

INSERT INTO Patient (Patient\_ID, Patient\_Fname, Patient\_Lname, DOB, Address, ER\_Number, Doctor\_ID, Patient\_Phone)

VALUES

(NEXTVAL('SEQ\_Patient\_Patient\_ID'), 'John', 'Doe', '1985-05-12', '123 Main St', 'ER1001', 1, '3216549870'),

(NEXTVAL('SEQ\_Patient\_Patient\_ID'), 'Jane', 'Smith', '1990-08-25', '456 Oak St', 'ER1002', 10, '9876543210'),

(NEXTVAL('SEQ\_Patient\_Patient\_ID'), 'Sam', 'Johnson', '1975-12-11', '789 Pine St', 'ER1003', 3, '6549873210'),

(NEXTVAL('SEQ\_Patient\_Patient\_ID'), 'Mary', 'Brown', '1968-03-04', '101 Maple St', 'ER1004', 4, '9638527410'),

(NEXTVAL('SEQ\_Patient\_Patient\_ID'), 'Mike', 'Davis', '2002-11-19', '202 Cedar St', 'ER1005', 5, '7418529630'),

(NEXTVAL('SEQ\_Patient\_Patient\_ID'), 'Sue', 'Adams', '1995-09-30', '303 Birch St', 'ER1006', 6, '2589631470'),

(NEXTVAL('SEQ\_Patient\_Patient\_ID'), 'Tom', 'White', '1988-02-22', '404 Elm St', 'ER1007', 7, '8529637410'),

(NEXTVAL('SEQ\_Patient\_Patient\_ID'), 'Sara', 'Wilson', '1997-07-14', '505 Walnut St', 'ER1008', 8, '1597534860'),

(NEXTVAL('SEQ\_Patient\_Patient\_ID'), 'Paul', 'Thomas', '1981-01-01', '606 Willow St', 'ER1009', 9, '1234567890'),

(NEXTVAL('SEQ\_Patient\_Patient\_ID'), 'Anna', 'Lee', '1993-06-15', '707 Spruce St', 'ER1010', 10, '7896541230'),

(NEXTVAL('SEQ\_Patient\_Patient\_ID'), 'Rock', 'Lee', '1993-11-15', '709 Hunt St', 'ER1010', 10, '78965412355');

INSERT INTO Appointment (App\_ID, App\_Date, App\_Time, Visit\_Reason, Status, Payment, Patient\_ID, Doctor\_ID)

VALUES

(NEXTVAL('SEQ\_Appointment\_App\_ID'), '2024-12-01', '09:00', 'Routine Checkup', 'Scheduled', 'Pending', 1001, 1),

(NEXTVAL('SEQ\_Appointment\_App\_ID'), '2024-12-03', '10:30', 'Headache', 'Scheduled', 'Paid', 1002, 10),

(NEXTVAL('SEQ\_Appointment\_App\_ID'), '2024-12-05', '14:00', 'Back Pain', 'Scheduled', 'Unpaid', 1003, 3),

(NEXTVAL('SEQ\_Appointment\_App\_ID'), '2024-12-08', '16:00', 'Vaccination', 'Scheduled', 'Paid', 1004, 4),

(NEXTVAL('SEQ\_Appointment\_App\_ID'), '2024-12-10', '12:00', 'Skin Rash', 'Scheduled', 'Pending', 1005, 5),

(NEXTVAL('SEQ\_Appointment\_App\_ID'), '2024-12-12', '11:00', 'Cancer Follow-up', 'Scheduled', 'Paid', 1006, 6),

(NEXTVAL('SEQ\_Appointment\_App\_ID'), '2024-12-15', '15:00', 'Stomach Pain', 'Scheduled', 'Unpaid', 1007, 7),

(NEXTVAL('SEQ\_Appointment\_App\_ID'), '2024-12-18', '13:00', 'Anxiety Consultation', 'Scheduled', 'Paid', 1008, 8),

(NEXTVAL('SEQ\_Appointment\_App\_ID'), '2024-12-20', '08:30', 'X-Ray', 'Scheduled', 'Pending', 1009, 9),

(NEXTVAL('SEQ\_Appointment\_App\_ID'), '2024-12-22', '14:00', 'Vision Check', 'Scheduled', 'Paid', 1010, 10),

(NEXTVAL('SEQ\_Appointment\_App\_ID'), '2024-12-22', '15:00', 'Vision Check', 'Scheduled', 'Paid', 1011, 10);

INSERT INTO Bill (Bill\_ID, Bill\_Amt, Tax, Discount, Pay\_Status, Pay\_Method, App\_ID, Pay\_Due\_Date)

VALUES

(NEXTVAL('SEQ\_Bill\_Bill\_ID'), 200.00, 10.00, 0.00, 'Paid', 'Credit Card', 11, NULL),

(NEXTVAL('SEQ\_Bill\_Bill\_ID'), 150.00, 7.50, 5.00, 'Pending', 'Cash', 12, '2024-12-05'),

(NEXTVAL('SEQ\_Bill\_Bill\_ID'), 300.00, 15.00, 10.00, 'Unpaid', 'Insurance', 13, '2024-12-08'),

(NEXTVAL('SEQ\_Bill\_Bill\_ID'), 100.00, 5.00, 0.00, 'Paid', 'Credit Card', 14, NULL),

(NEXTVAL('SEQ\_Bill\_Bill\_ID'), 250.00, 12.50, 0.00, 'Pending', 'Cash', 15, '2024-12-12'),

(NEXTVAL('SEQ\_Bill\_Bill\_ID'), 400.00, 20.00, 50.00, 'Paid', 'Credit Card', 16, NULL),

(NEXTVAL('SEQ\_Bill\_Bill\_ID'), 500.00, 25.00, 0.00, 'Unpaid', 'Insurance', 17, '2024-12-18'),

(NEXTVAL('SEQ\_Bill\_Bill\_ID'), 350.00, 17.50, 15.00, 'Pending', 'Cash', 18, '2024-12-22'),

(NEXTVAL('SEQ\_Bill\_Bill\_ID'), 100.00, 5.00, 0.00, 'Paid', 'Credit Card', 19, NULL),

(NEXTVAL('SEQ\_Bill\_Bill\_ID'), 200.00, 10.00, 0.00, 'Pending', 'Cash', 20, '2024-12-31'),

(NEXTVAL('SEQ\_Bill\_Bill\_ID'), 200.00, 10.00, 0.00, 'Pending', 'Cash', 21, '2024-12-31');

-- Select \* from doctor

-- Select \* from appointment

-- Select \* from bill

-- Select \* from patient

-- Select \* from department

-- select \* from view\_patient\_doctors

-- example query (Appointment details of a doctor who has more than 2 appointments)

SELECT

d.Doctor\_ID,

d.Doctor\_Fname || ' ' || d.Doctor\_Lname AS Doctor\_Name,

a.App\_ID,

a.App\_Date,

a.App\_Time,

a.Visit\_Reason,

a.Status,

a.Payment,

a.Patient\_ID

FROM

Doctor d

JOIN

Appointment a ON d.Doctor\_ID = a.Doctor\_ID

WHERE

d.Doctor\_ID IN (

SELECT Doctor\_ID

FROM Appointment

GROUP BY Doctor\_ID

HAVING COUNT(App\_ID) > 2

)

ORDER BY

d.Doctor\_ID, a.App\_Date;

**DDL and DML Query Output**

**A close-up of a white background

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**A close-up of a white background

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**A close-up of a white background

Description automatically generated**

**A screenshot of a computer

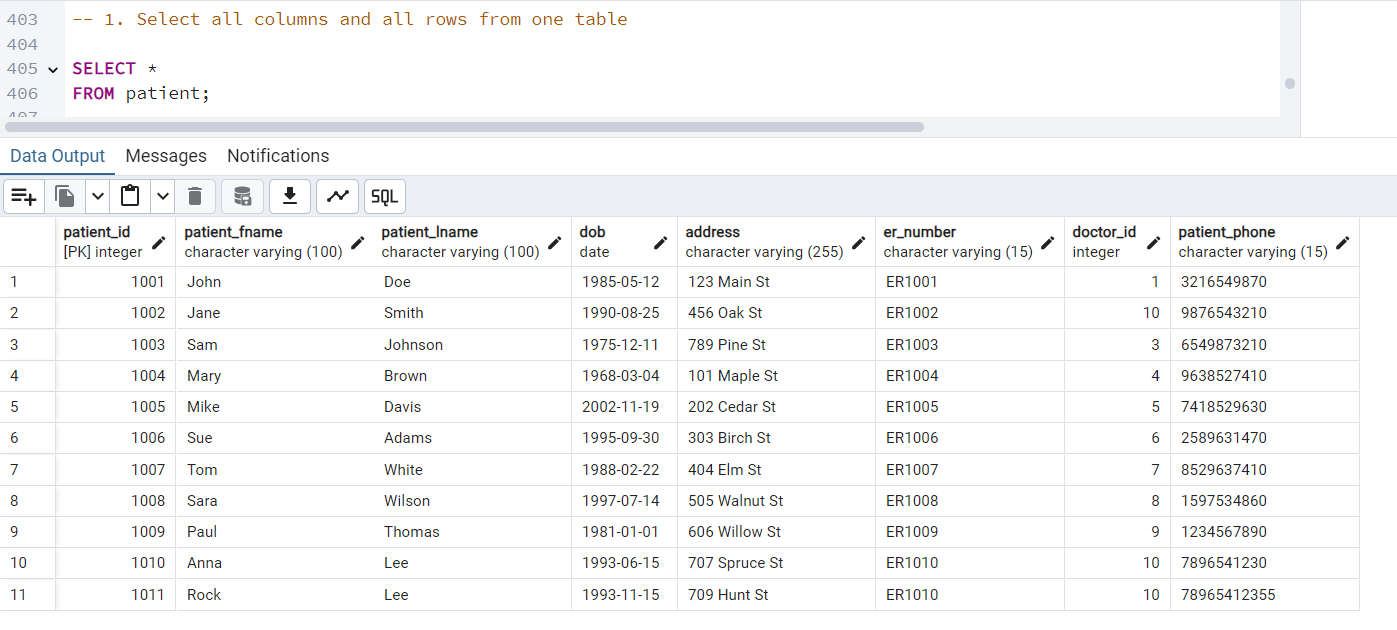
Description automatically generated**

**A screenshot of a computer

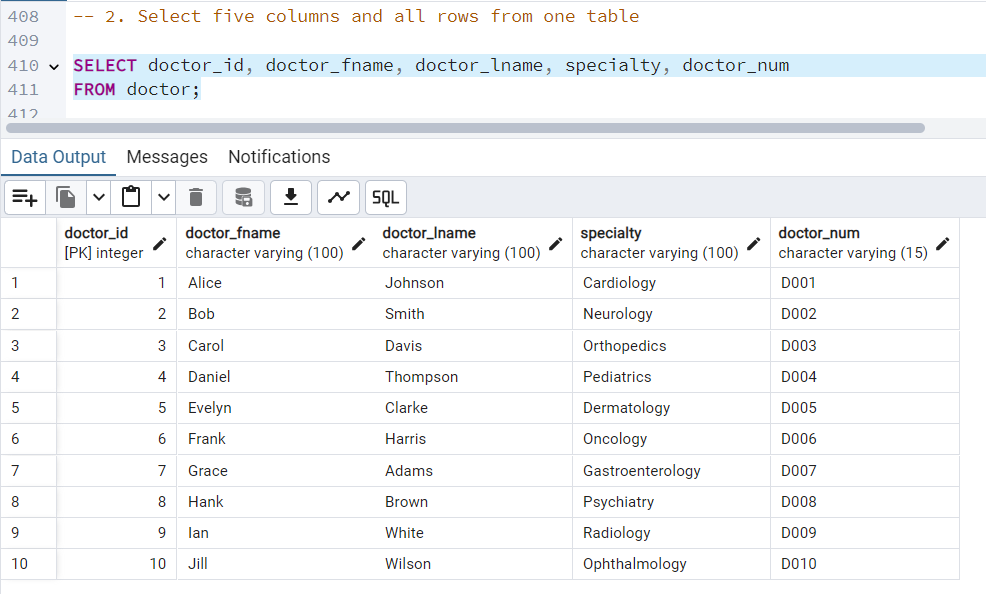
Description automatically generated**

**SQL Query Output**

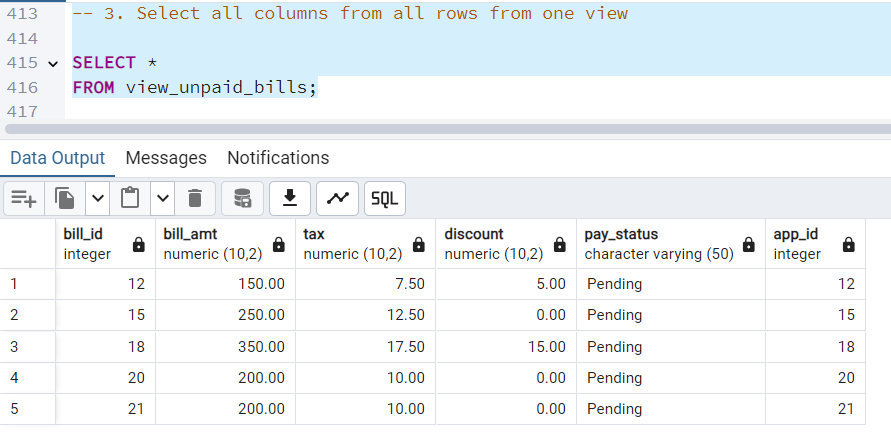
* **Query 1: Select all columns and all rows from one table (5 points)**

****

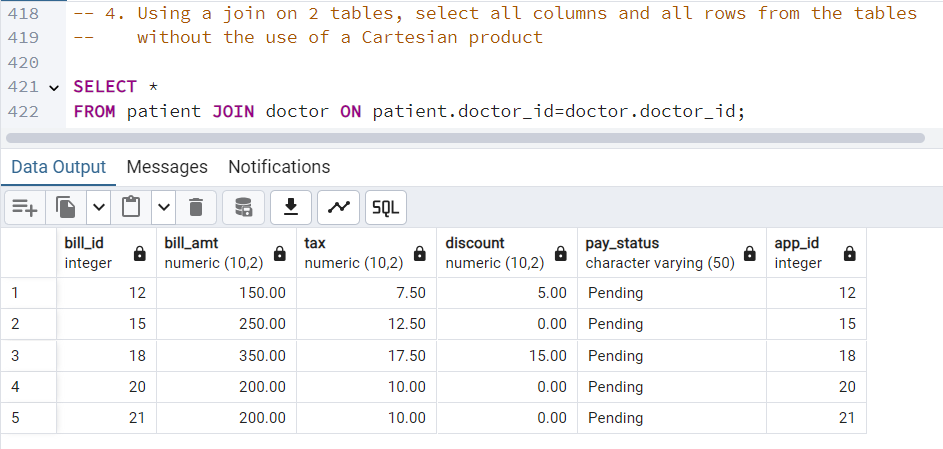
* **Query 2: Select five columns and all rows from one table (5 points)**

****

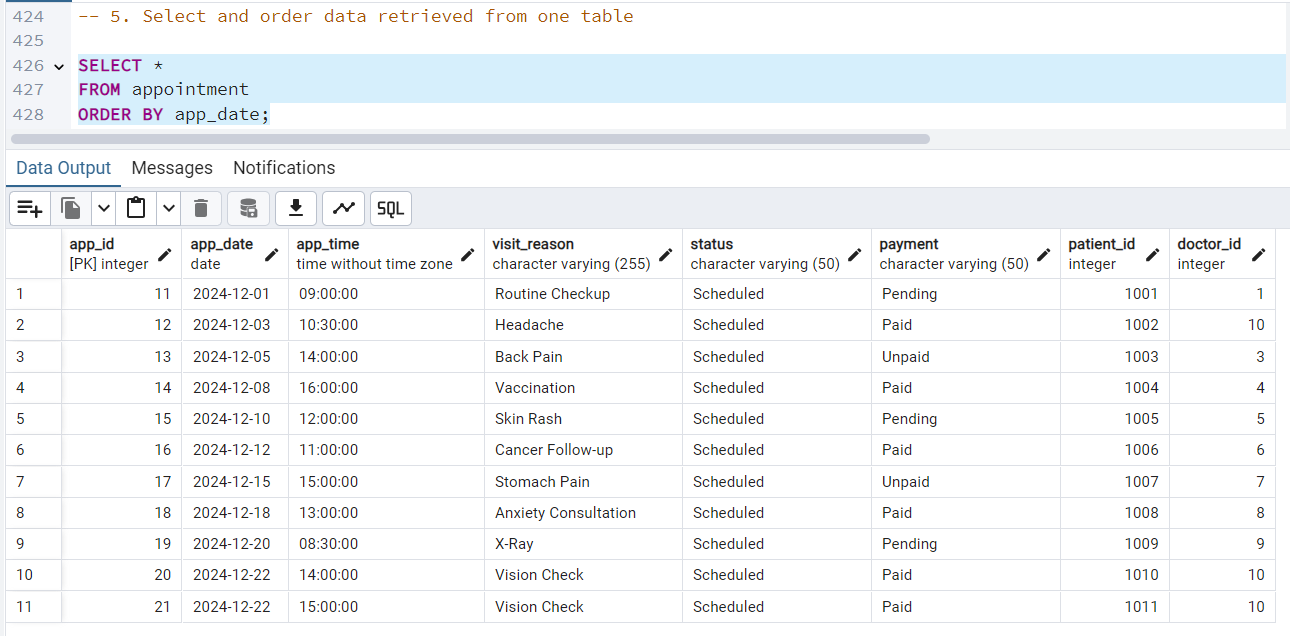
* **Query 3: Select all columns from all rows from one view (5 points)**

****

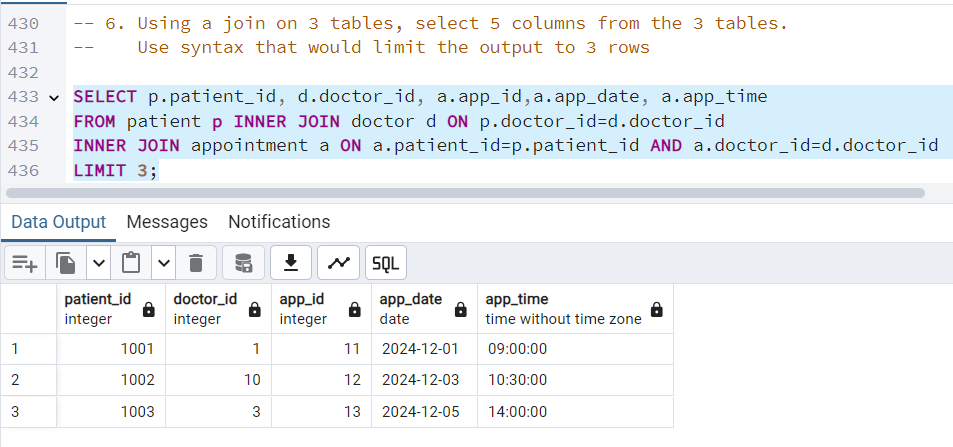
* **Query 4: Using a join on 2 tables, select all columns and all rows from the tables without the use of a Cartesian product (5 points)**

****

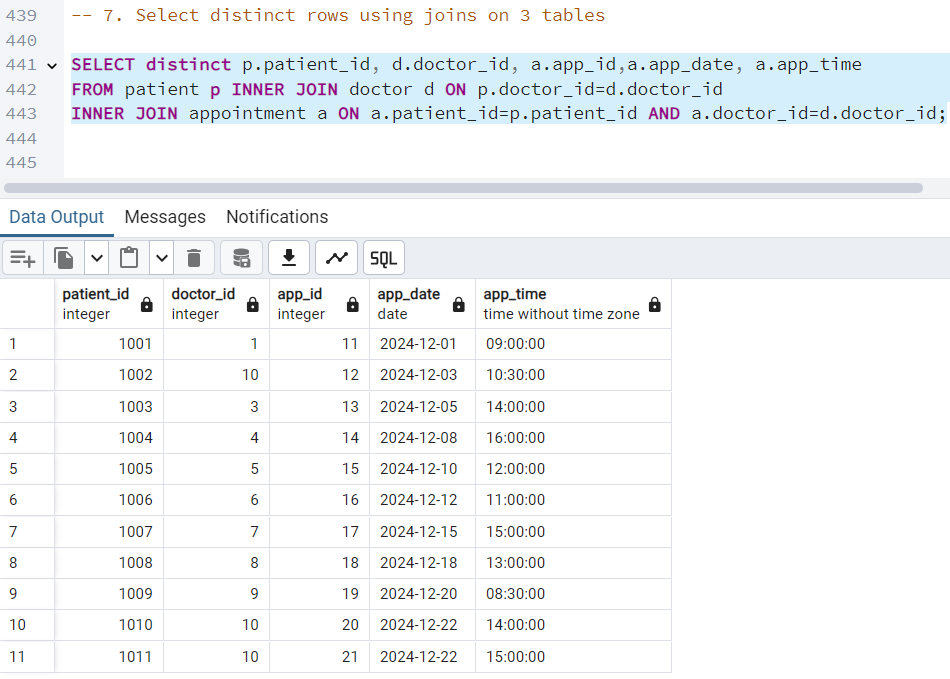
* **Query 5: Select and order data retrieved from one table (5 points)**

****

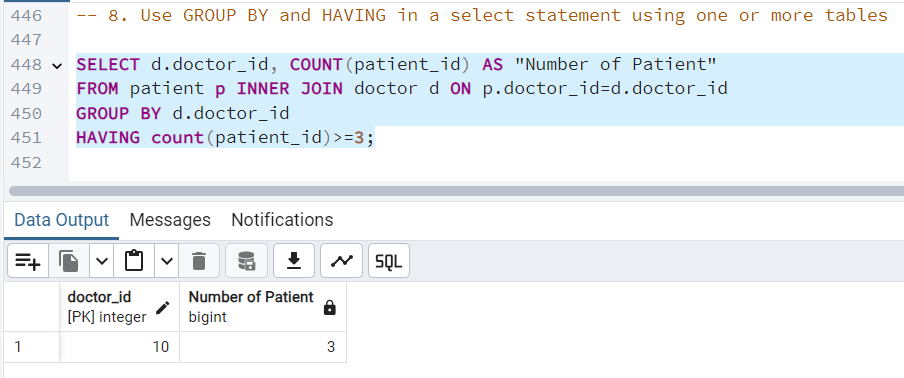
* **Query 6: Using a join on 3 tables, select 5 columns from the 3 tables. Use syntax that would limit the output to 3 rows (5 points)**

****

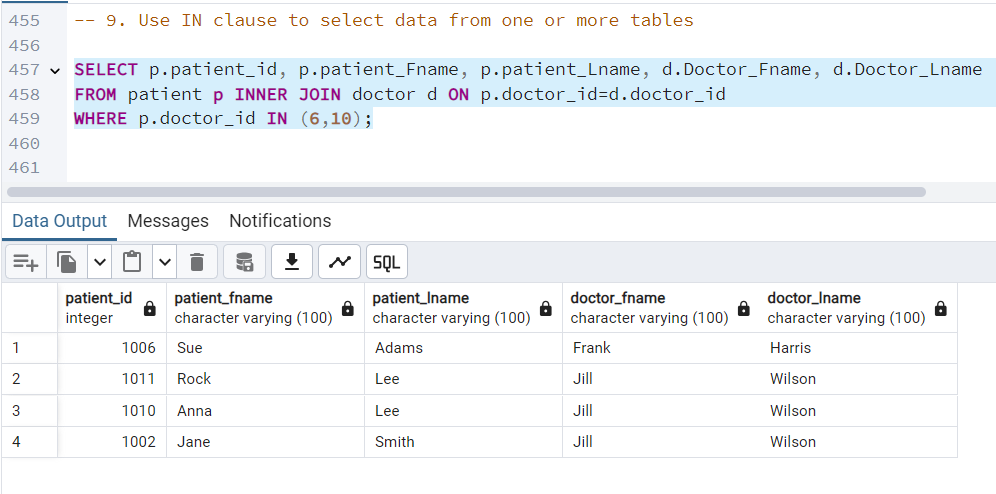
* **Query 7: Select distinct rows using joins on 3 tables (5 points)**

****

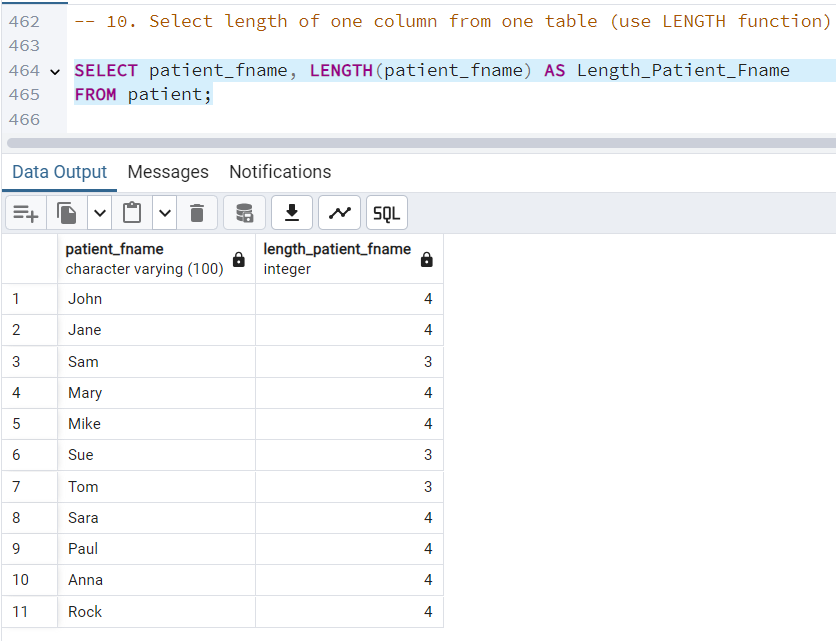
* **Query 8: Use GROUP BY and HAVING in a select statement using one or more tables (5 points)**

****

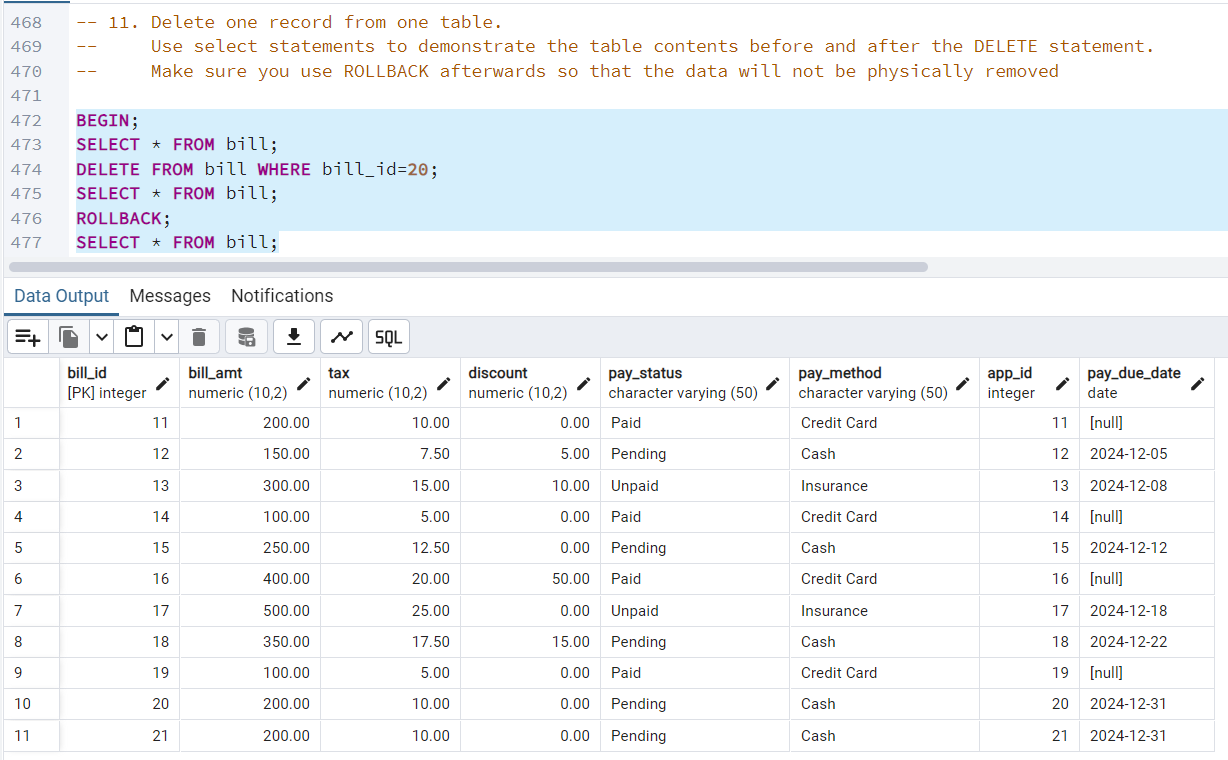
* **Query 9: Use IN clause to select data from one or more tables (5 points)**

****

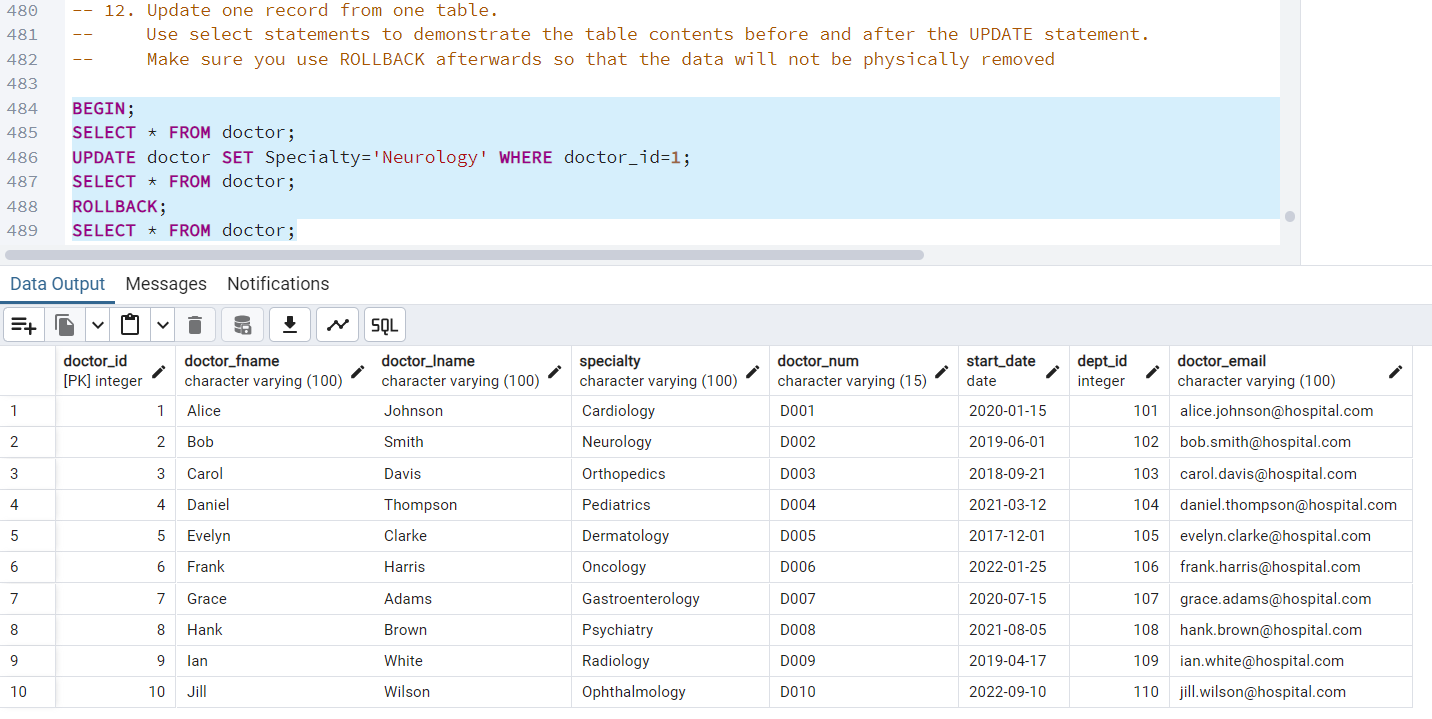
* **Query 10: Select length of one column from one table (use LENGTH function) (5 points)**

****

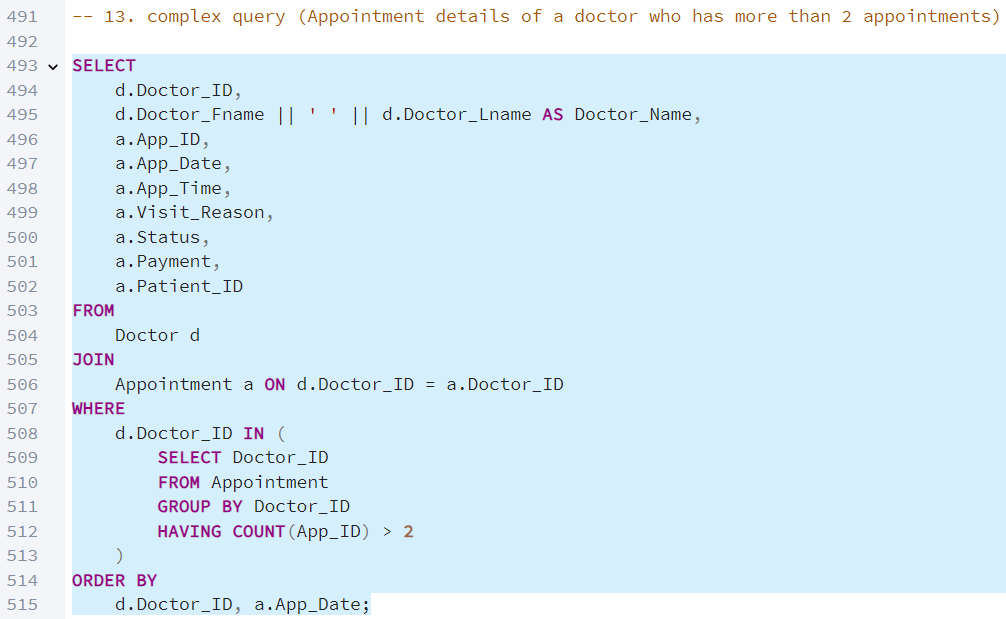
* **Query 11: Delete one record from one table. Use select statements to demonstrate the table contents before and after the DELETE statement. Make sure you use ROLLBACK afterwards so that the data will not be physically removed (5 points)**

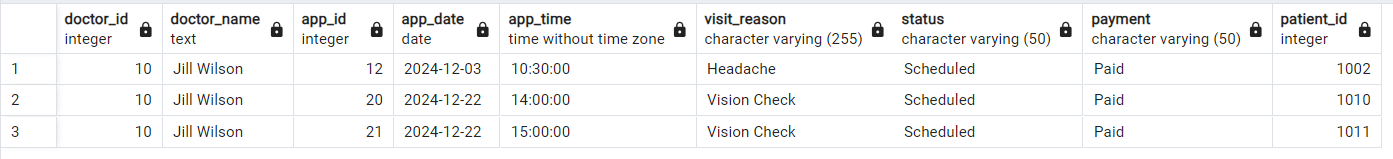
****

* **Query 12: Update one record from one table. Use select statements to demonstrate the table contents before and after the UPDATE statement. Make sure you use ROLLBACK afterwards so that the data will not be physically removed (5 points)**

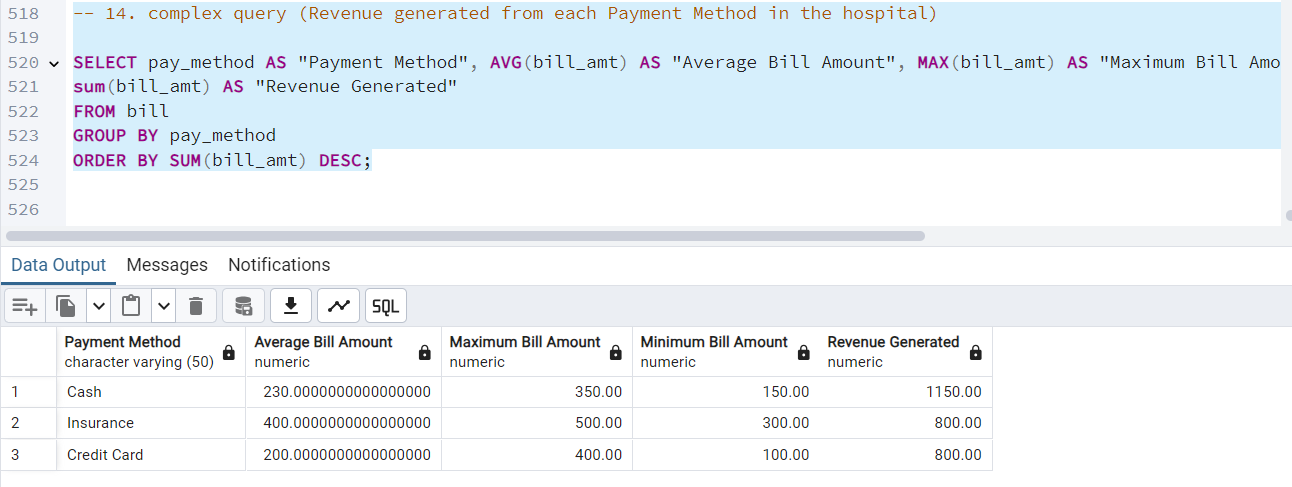
****

* **Query 13 (Advanced Query) – Find out the appointment details of all doctors who have more than 2 appointments**

****

****

* **Query 14 (Advanced Query) – Find out which payment method generates the most revenue for the hospital.**

****

**Database Administration and Monitoring**

**Roles and Responsibilities**

1. **Database Administrator (DBA)**:
   1. Maintains the database, performs backups, and optimizes performance.
   2. Develops and updates SQL scripts to meet hospital requirements.
   3. Ensures consistency with the database schema and ERD.
2. **System Administrator**:
   1. Manages the DBMS server, including software updates and hardware maintenance.
   2. Ensures server uptime and resolves performance issues in collaboration with the DBA.
3. **Security Administrator**:
   1. Safeguards database security through encryption and monitoring access logs.
   2. Addresses vulnerabilities and updates security policies.
   3. Works with DBAs and system admins to maintain a secure environment.

#### **Monitoring and Maintenance**

* **Performance**: Optimizes slow queries and monitors server resource usage.
* **Data Integrity**: Regularly checks relationships like Patient–Doctor and Appointment–Bill for consistency.
* **Backups**: Ensures routine backups and tests recovery processes.
* **Security**: Conducts audits of permissions and addresses vulnerabilities.

**System Information**

**Performance Monitoring and Database Efficiency**

The performance of the hospital management database, DBMS, and servers will be a shared responsibility between the **Database Administration** and **System Administration** teams:

* **Database Administration Team**:
  + Monitors query performance and optimizes slow queries.
  + Maintains database indices and ensures efficient execution of common operations like scheduling, billing, and record retrieval.
* **System Administration Team**:
  + Manages server resources (CPU, memory, storage) and ensures optimal hardware utilization.
  + Collaborates with the DBA team to address DBMS-related performance issues.

**Data Formats**

The database will handle the following types of data:

* **String, Integer, Date, and Time**: Stored directly in the database for fields like patient names, appointment times, and bill amounts.
* **Image Data**: Patient reports or scanned documents are stored externally as image files (e.g., PNG) with URIs linking them to database records.
* **External Data Files**: Large documents or diagnostic files are stored outside the database, with the database maintaining references for efficient access.

**Backup and Recovery**

To protect against data loss and ensure continuity, the following backup strategy will be implemented:

* **Delta Backups**:
  + Performed twice daily to record incremental changes, such as new appointments, billing updates, and patient registrations.
* **Full Backups**:
  + Conducted weekly during scheduled maintenance (3 AM EST on Tuesdays) to create a complete snapshot of the database.

**References**

* Holywell, Simon. “SQL Style Guide.” *SQL Style Guide by Simon Holywell*, 25 Nov. 2024, www.sqlstyle.guide/.
* Panchasara, Milit. “Hospital Management System: A Detailed Guide.” *Radixweb*, Radixweb, 6 May 2024, radixweb.com/blog/hospital-management-system-features-and-benefits.