# A. Title Page

Lewis University  
CPSC 50900: Database Systems   
Spring 2025 Term Project

Smart Healthcare Management System

Vamshi Krishna Akuthota, VamshiKrishnaAkuth@lewisu.edu

Bharath Kumar Pola, BharathKumarPola@lewisu.edu

Harini Radhavaram, Hariniradhavaram@lewisu.edu

Work products stored in the Github repository <https://github.com/BharathKumar0206/Database-Project.git>

Table of Contents

[A. Title Page 1](#_Toc192964418)

[Schedule of Milestones 1](#_Toc192964419)

[B. Initial Proposal 2](#_Toc192964420)

[C. Data Sources 5](#_Toc192964421)

[D. Alternative Ways to Store the Data 10](#_Toc192964422)

[E. Conceptual and Logical Models 14](#_Toc192964423)

[F. Physical Model 23](#_Toc192964424)

[G. Populate the database with data 29](#_Toc192964425)

[H. Data Manipulation Language (DML) Scripts 37](#_Toc192964426)

[I. Indexes 46](#_Toc192964427)

[J. Views 50](#_Toc192964428)

[K. Stored Programs (Stored Procedures, Stored Functions, Triggers) 53](#_Toc192964429)

[L. Transactions 55](#_Toc192964430)

[M. Database Security 58](#_Toc192964431)

[N. Locking and Concurrent Access 61](#_Toc192964432)

[O. Backing Up Your Database 64](#_Toc192964433)

[P. Programming 67](#_Toc192964434)

[Q. Suggested Future Work 73](#_Toc192964435)

[R. Activity Log 76](#_Toc192964436)

# Schedule of Milestones

Here is a schedule that shows when each milestone is due and what sections comprise it.

|  |  |
| --- | --- |
| Deadline | Sections for which you must demonstrate significant progress |
| February 4 at 11:59pm | a. Title page  b. Initial proposal  c. Data sources  d. Alternative ways to store the data  r. Activity Log – at least six entries covering the first two weeks |
| February 18  at 11:59pm | e. Conceptual and logical models  f. Physical model  g. Populate the database with data  r. Activity Log – at least six entries covering the past two weeks |
| March 4 at 11:59pm | h. Data manipulation language (DML) scripts  i. Indexes  j. Views  l. Transactions  m. Security  r. Activity Log – at least six entries covering the past two weeks |

The remaining sections – Triggers, Locking and Concurrency, Backup, and Programming, will be turned in with the final report, which is due March 16 at 11:59pm.

# B. Initial Proposal

*Description: You will describe the data you aim to store. What data will be storing? Why are you interested in this data? Why is it important? Where will the data come from? Who will use this data? What kind of application do you plan to build with it?*

*Rubric: Your response to each of these six questions will be graded out of 3 points.*

* *3 points: clear, complete descriptions that convey the importance and meaning of your data*
* *2 points: mostly clear descriptions, although some additional data would have helped in some sections*
* *1 point: necessary details are lacking in many of your responses.*

*You will also earn 2 additional points for coming up with a descriptive title for your project.*

*As you consider various ideas for your project, keep in mind that your database is going to have to store data for at least 8 different types of things. Each of these different “types of things” will become a table in the database you design and build. So, the idea can’t be so narrow that you can’t identify at least eight different types of things in it that you’d store data about.*

*Total points possible: 20*

ENTER YOUR INITIAL PROPOSAL HERE

**Project Description:**

Designed to improve the accessibility, accuracy, and efficiency of healthcare services, the centralized, database-driven Smart Healthcare Management System (SHMS) would the technology will help with electronic health record (EHR) management, appointment scheduling, medical prescriptions, billing, hospital inventories, lab test results.

Designed for hospitals, clinics, and other healthcare institutions, this database system will help to maximize their operations and enhance patient treatment. Ensuring data integrity and compliance with healthcare laws will be the main goals of the structured, safe, and quick approach to handle hospital data that will be offered.

Through a user-friendly web and mobile app, the system will let several users including doctors, nurses, hospital managers, pharmacists, and patients to engage with pertinent data. Storing several kinds of medical and administrative records to support hospital operations and decision-making procedures, the database will be the center of this system.

**Data to be stored:**

The database will keep several kinds of data connected to healthcare. Among the main data categories will be:

* **Patient Records:** Personal details, contact information, insurance details, medical history, allergies, and emergency contacts.
* **Doctor Profiles:** Professional credentials, specializations, availability, and assigned patients.
* **Appointments:** Scheduling details, consultation times, patient-doctor assignments, and appointment statuses.
* **Medical History:** Diagnoses, treatment plans, past surgeries, and ongoing conditions.
* **Prescriptions:** Medication details, dosage, duration, and refill schedules.
* **Billing Information:** Invoices, payments, insurance claims, and transaction records.
* **Hospital Inventory:** Stock of medical supplies, available equipment, and procurement records.
* **Lab Test Results:** Diagnostic test details, radiology reports, and test results linked to patient profiles.

Effective storage of this data will help the system to lower paperwork, decrease mistakes, and increase patient data accessibility.

**Interest in data:**

Though many hospitals still rely on ineffective paper-based or fragmented digital systems, the healthcare sector produces a great volume of vital patient and operational data. Through the development of a consolidated and orderly database, this project will:

* Less administrative work will help to improve hospital efficiency.
* Make sure doctors and nurses have immediate access to medical records will help to improve patient care.
* Maintaining accurate prescriptions and treatment histories will help to reduce medical mistakes.
* For improved hospital operations, simplify billing and appointment scheduling.
* Guarantee data security and regulatory compliance will help to safeguard patient privacy.

Given digital transformation as a top priority in healthcare, this system will be crucial first step toward modernized medical data management.

**Importance of data:**

Among the most delicate and important kinds of data available is healthcare-related one. For a number of reasons, correct data management is absolutely vital.

First, fast and better-informed decisions made by clinicians enabled by instant access to patient medical information will enhance patient outcomes. Correct prescribing and treatment monitoring will help lower medical errors which might be fatal.

Second, automation of administrative tasks will raise hospital efficiency. The system will manage appointment scheduling, billing, and insurance claims digitally instead of depending on hand records, therefore relieving hospital staff of work and preventing financial transaction mistakes.

At last, the system will help with medical research and analytics. Trend analysis of patient data can help hospitals to spot prevalent health problems, maximize resource allocation, and enhance general quality of healthcare.

**Data Gathering:**

Several sources within a healthcare institution will provide the data for this system.

Patient data will be recorded manually at first during registration; doctors and nurses will update medical history, diagnosis, and prescriptions following every session. Hospital transactions will cause appointments and billing records to be created automatically.

Pharmacists will handle prescription records and monitor inventory levels; lab technicians will upload test findings. Certain medical tools might also be linked to automatically save patient vitals and monitoring data.

Combining manual input, automated updates, and system linkages will help the database guarantee that medical practitioners always have real-time, accurate, and current data.

**Intended users:**

Many people inside a hospital will utilize this database, each with different access rights and obligations:

* Doctors will access patient medical histories, update treatment records, and prescribe medications.
* Nurses will monitor patient progress, track vitals, and assist in updating records.
* Patients will use a web or mobile interface to view their medical history, book appointments, and check prescriptions.
* Hospital Administrators will oversee hospital operations, staff schedules, and billing.
* Pharmacists will manage prescriptions and maintain an up-to-date inventory of available medicines.
* Lab Technicians will upload diagnostic test results and link them to patient records.

Every user will have role-based access control so that private data stays under protection and only available to authorized staff.

**Application to be developed:**

A Smart Healthcare Management System (SHMS) with an integrated web and mobile application will come of this project. The main characteristics of this system will include in:

* Electronic Health Records (EHRs): Structured and safe patient data storage.
* Online Appointment Scheduling: Patients can schedule and handle doctor visits.
* Prescription Management: Pharmacists will monitor drug status; doctors will write prescriptions electronically.
* Billing & Insurance Processing: Automated insurance claim handling and billing.
* Hospital Inventory Management: Track medical supply and equipment stock levels.
* Lab Test Integration: Simple test result uploading and access.
* Analytics & Reporting: AI-based discoveries for improved hospital resource allocation.

This program will be tailored to fit the particular requirements of clinics and hospitals, therefore enabling medical professionals and managers to effectively handle healthcare data while preserving security and privacy standards compliance.

# C. Data Sources

*Description:* *Gather your data in text files. The text files may be csv, tab-delimited, xml, json, or some other custom format. Not all the files need be of the same type. Identify what each file contains by indicating where it came from, explaining in detail how it is structured, and describing how you will reorganize the data into a relational database. Post your data files to your GitHub repository, and provide samples of the data in your Word doc.*

*Rubric: Your work will be graded as follows:*

* *5 points: you gathered multiple data files that contain the data that will populate your databases. If you do not use multiple data files, you will not receive credit.*
* *5 points: you described the contents of the data files in detail, including referencing their origin and explaining how they were structured.*
* *3 points: you identify which fields you plan to include in your database, including their data types and any constraints you expect to impose on the data or steps you'll have to take to clean up the data.*
* *2 points: you post the data files to your GitHub account and make it possible for me to see them.*

*Total points possible: 15*

ENTER YOUR DATA SOURCES DESCRIPTION HERE

Using several data sources, the Smart Healthcare Management System (SHMS) will fill its relational database. Collected from several hospital departments and outside healthcare systems, these data sources CSV, JSON, and XML among others will be in varied forms.

The gathered data will be organized and converted into a structured relational form guaranteeing security, data integrity, and efficiency.

**Data files overview:**

|  |  |  |  |
| --- | --- | --- | --- |
| **File Name** | **Format** | **Source** | **Purpose** |
| patients.csv | CSV | Patient registration system | Stores patient demographic and medical details |
| doctors.csv | CSV | Hospital HR records | Contains doctor profiles and specialization |
| appointments.json | JSON | Online scheduling system | Stores patient-doctor appointment details |
| prescriptions.xml | XML | Pharmacy system | Holds medication prescription details |
| billing.csv | CSV | Finance department | Stores billing, payments, and insurance records |
| inventory.csv | CSV | Hospital inventory system | Tracks medical supplies and stock levels |
| lab\_results.json | JSON | Laboratory database | Stores diagnostic test results |
| staff.csv | CSV | HR system | Maintains hospital staff information |

Every file will be organized and standardized to match a relational database paradigm, therefore enabling effective data retrieval and searching.

**Structure of Files and Transformation:**

**Patients.csv:**

|  |
| --- |
| patient\_id, first\_name, last\_name, dob, gender, contact, address, insurance, medical\_history  1001, Sai, Teja, 1998-06-15, Male, 1234567890, "123 Illinois", Blue Cross, "Diabetes, Hypertension" |

**Transformation:**

* Normalize insurance details into a separate table (insurance)
* Extract medical history and store in medical\_records table
* Split contact details into contacts table

**Doctors.csv:**

|  |
| --- |
| doctor\_id, first\_name, last\_name, specialization, phone, email, availability  D001, Nandini, Roy, Cardiology, 555-111-2222, nroy@hospital.com, "Mon-Fri 9AM-5PM" |

**Transformation:**

* Store doctor specialization in specializations table
* Separate contact details into contacts table

**Appointments.json:**

|  |
| --- |
| [  {  "appointment\_id": "A1001",  "patient\_id": "1001",  "doctor\_id": "D001",  "date": "2025-03-05",  "time": "10:30 AM",  "status": "Confirmed"  },  {  "appointment\_id": "A1002",  "patient\_id": "1002",  "doctor\_id": "D002",  "date": "2025-03-06",  "time": "02:00 PM",  "status": "Pending"  }  ] |

**Transformation:**

* Convert JSON to relational format and store in appointments table
* Extract appointment status and store in appointment\_status lookup table

**Prescriptions.xml:**

|  |
| --- |
| <prescriptions>  <prescription>  <prescription\_id>P1001</prescription\_id>  <patient\_id>1001</patient\_id>  <doctor\_id>D001</doctor\_id>  <medication>Metformin</medication>  <dosage>500mg</dosage>  <duration>30 days</duration>  </prescription>  </prescriptions> |

**Transformation:**

* Extract medications and store in medications table
* Normalize prescriptions into a relational structure

**Billing.csv:**

|  |
| --- |
| invoice\_id, patient\_id, amount, insurance\_covered, payment\_status, date\_issued, date\_paid  INV001, 1001, 500, Yes, Paid, 2025-03-05, 2025-03-07 |

**Transformation:**

* Store insurance data in insurance table
* Normalize payment statuses in payment status table

**Inventory.csv:**

|  |
| --- |
| item\_id,name,category,stock\_level,supplier\_id  I001,Ventilator,Medical Equipment,10,S001 |

**Transformation:**

* Extract item\_id, name, category, stock\_level, supplier\_id from CSV.
* Normalize data by moving supplier\_id to a separate suppliers table to avoid redundancy.

**Lab\_results.json:**

|  |
| --- |
| {  "lab\_id": "L001",  "patient\_id": "1001",  "test\_name": "Blood Test",  "result": "Positive",  "test\_date": "2025-03-10"  } |

**Transformation:**

* Extract lab\_id, patient\_id, test\_name, result, test\_date
* Normalize data by linking patient\_id to the patients table.

**Staff\_data.csv:**

|  |
| --- |
| staff\_id,first\_name,last\_name,role,contact  S001,Anil,Singh,Nurse,999881235 |

**Transformation:**

* Extract staff\_id, first\_name, last\_name, role, contact
* Normalize by storing roles in a separate roles table if needed.

**Relational Database Schema Design:**

**Patients Table:**

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Constraints** |
| patient\_id | INT | PRIMARY KEY, AUTO\_INCREMENT |
| first\_name | VARCHAR(50) | NOT NULL |
| last\_name | VARCHAR(50) | NOT NULL |
| dob | DATE | NOT NULL |
| gender | ENUM | NOT NULL |
| contact | VARCHAR(15) | UNIQUE, NOT NULL |
| address | TEXT | NOT NULL |
| insurance\_id | INT | FOREIGN KEY REFERENCES insurance(insurance\_id) |
| medical\_history | TEXT | NULL |

**Doctors Table:**

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Constraints** |
| doctor\_id | INT | PRIMARY KEY, AUTO\_INCREMENT |
| first\_name | VARCHAR(50) | NOT NULL |
| last\_name | VARCHAR(50) | NOT NULL |
| specialization | VARCHAR(100) | NOT NULL |
| phone | VARCHAR(15) | UNIQUE, NOT NULL |
| email | VARCHAR(100) | UNIQUE, NOT NULL |
| availability | VARCHAR(100) | NOT NULL |

**Appointments Table:**

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Constraints** |
| appointment\_id | INT | PRIMARY KEY, AUTO\_INCREMENT |
| patient\_id | INT | FOREIGN KEY REFERENCES patients(patient\_id) |
| doctor\_id | INT | FOREIGN KEY REFERENCES doctors(doctor\_id) |
| date | DATE | NOT NULL |
| time | TIME | NOT NULL |
| status | ENUM | NOT NULL |

**Prescriptions Table:**

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Constraints** |
| prescription\_id | INT | PRIMARY KEY, AUTO\_INCREMENT |
| appointment\_id | INT | FOREIGN KEY REFERENCES appointments(appointment\_id) |
| medication | VARCHAR(100) | NOT NULL |
| dosage | VARCHAR(50) | NOT NULL |
| duration | INT | NOT NULL |

**Billing Table:**

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Constraints** |
| invoice\_id | INT | PRIMARY KEY, AUTO\_INCREMENT |
| patient\_id | INT | FOREIGN KEY REFERENCES patients(patient\_id) |
| amount | DECIMAL(10,2) | NOT NULL |
| insurance\_covered | BOOLEAN | NOT NULL |
| payment\_status | ENUM | NOT NULL |
| date\_issued | DATE | NOT NULL |
| date\_paid | DATE | NULL |

**Inventory Table:**

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Constraints** |
| item\_id | INT | PRIMARY KEY, AUTO\_INCREMENT |
| name | VARCHAR(100) | NOT NULL |
| stock\_level | INT | NOT NULL |
| supplier\_id | INT | FOREIGN KEY REFERENCES suppliers(supplier\_id) |

**Lab Results Table:**

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Constraints** |
| lab\_id | INT | PRIMARY KEY, AUTO\_INCREMENT |
| patient\_id | INT | FOREIGN KEY REFERENCES patients(patient\_id) |
| test\_name | VARCHAR(100) | NOT NULL |
| result | ENUM | NOT NULL |
| test\_date | DATE | NOT NULL |

**Staff Table:**

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Constraints** |
| staff\_id | INT | PRIMARY KEY, AUTO\_INCREMENT |
| first\_name | VARCHAR(50) | NOT NULL |
| last\_name | VARCHAR(50) | NOT NULL |
| role | VARCHAR(100) | NOT NULL |
| contact | VARCHAR(15) | UNIQUE, NOT NULL |

**Constraints and Data Integrity Measures:**

* **Primary Keys:** Verify originality for every record in a table.
* **Foreign Keys:** Link tables to provide referential integrity.
* **Enums:** For categorical data like gender, appointment status, and test findings, set limited possible values.
* **Auto-Increment:** Create fresh IDs automatically for main keys.
* **Not Null Constraints:** Verify always in important areas including names, contacts, and payment status.

**Data Cleaning & Processing:**

* These actions will be done before adding the data to the relational database:
* Remove duplicates to maintain consistency.
* Standardize formats (dates in YYYY-MM-DD, phone numbers in XXX-XXX-XXXX).
* Normalize data by splitting complex fields into separate tables.

# D. Alternative Ways to Store the Data

*Description: We will study alternatives to storing data in a relational database. Some of the alternatives come from several decades ago, including the hierarchical and network models. Some are newer options, such as NoSQL databases that use JSON or some other encoding. Describe in detail how to store the data using two alternatives to relational databases. Be sure to describe how you would implement the alternatives and the advantages and disadvantages of each.*

*Rubric: Your work will be graded as follows*

* *5 points for clearly describing how your data could be stored using one alternative to relational databases and what the advantages and disadvantages of that approach would be.*
* *5 points for clearly describing how your data could be stored using another alternative to relational databases and what the advantages and disadvantages of that approach would be.*

*Total points possible: 10*

ENTER YOUR ALTERNATIVE DATA STORAGE IDEAS HERE

**Overview of the Hierarchical Model**

With a single parent but several child records, the Hierarchical Database Model arranges data into a tree-like structure. Like a computer file system's directory organization, data is kept in a parent-child relationship.   
Within a healthcare management system, data can be kept in a hierarchical structure whereby every patient serves as the root node and related records including visits, medications, and billing records are arranged as child nodes.

**Implementation of the Hierarchical Model in Healthcare**

XML-based databases or specialist hierarchical database management solutions like IBM Information Management System (IMS) allow one to use a hierarchical approach. One might depict the data's structure as follows:

|  |
| --- |
| <Patient ID="1001">  <Name>Rahul Sharma</Name>  <DOB>1985-06-15</DOB>  <Appointments>  <Appointment ID="A101">  <Doctor>Name="Dr. Mehta"</Doctor>  <Date>2025-03-15</Date>  <Time>10:30 AM</Time>  </Appointment>  </Appointments>  <Prescriptions>  <Prescription ID="P205">  <Medication>Metformin</Medication>  <Dosage>500mg</Dosage>  <Duration>30 Days</Duration>  </Prescription>  </Prescriptions>  <Billing>  <Invoice ID="B301">  <Amount>5000</Amount>  <Status>Paid</Status>  </Invoice>  </Billing>  </Patient> |

Under this approach, every Patient node contains child nodes for billing, prescriptions, and appointments. In XML databases or via navigational techniques in hierarchical database systems, inquiries are conducted using XPath or XQuery.

**Advantages of the Hierarchical Model**

Fast data retrieval speed of hierarchical databases is one of their main features. Data is pre-organized into a rigid hierarchy, hence searches following a top-down traversal run faster than sophisticated SQL joins in relational databases. For read-heavy applications like obtaining patient medical histories, hierarchical databases are therefore quite effective.

Consistency and data integrity provide still another benefit. Relationships are predefined, so the database imposes rigorous parent-child dependencies to lower the possibility of referential integrity problems or orphan records.

**Disadvantages of the Hierarchical Model**

Hierarchical architecture has some restrictions even if it has certain benefits. The stiff framework makes it challenging to allow adjustments in relationships. Should a new entity type such as insurance claims had to be included, the whole hierarchy could have to be rebuilt, which takes time.

Furthermore, for many-to- many relationships hierarchical databases are not adaptable. A doctor treating several patients, for instance, would need repeated doctor entries under several patient nodes, therefore causing data duplication and ineffective storage.

**Overview of the NoSQL Document-Oriented Model**

Rather than structured tables, NoSQL document databases—MongoDB, CouchDB, Firebase Firestore store data in JSON-like documents. Dealing with dynamic and semi-structured data that could change with time, this model is especially helpful.

Data is kept in collections of documents instead of tables, where every document has nested fields enabling customizable data representation.

**Implementation of a NoSQL Document Database in Healthcare**

Under a NoSQL system like MongoDB, a patient's record would be kept as a JSON document housed within a collection. The organization would resemble this:

|  |
| --- |
| {  "PatientID": 1001,  "Name": "Rahul Sharma",  "DOB": "1985-06-15",  "Appointments": [  {  "AppointmentID": "A101",  "Doctor": "Dr. Mehta",  "Date": "2025-03-15",  "Time": "10:30 AM"  }  ],  "Prescriptions": [  {  "PrescriptionID": "P205",  "Medication": "Metformin",  "Dosage": "500mg",  "Duration": "30 Days"  }  ],  "Billing": [  {  "InvoiceID": "B301",  "Amount": 5000,  "Status": "Paid"  }  ]  } |

Every patient is shown as a single document housed inside embedded arrays that captures all relevant information including visits, medicines, and billing records. MongoDB Query Language (MQL) replaces conventional SQL for searches.

**Advantages of NoSQL Document-Oriented Databases**

Schema adaptability of NoSQL databases is one of their main benefits. Unlike relational databases which demand preset table structures, NoSQL lets you store dynamic and changing data. In healthcare especially, where new data fields such as wearable health data or telemedicine records may need to be included without rewriting the whole database, this is especially helpful.   
Horizontal scalability is also another main benefit. NoSQL databases are perfect for managing vast amounts of data with high transaction volumes since they may be dispersed across several servers unlike relational databases that scale vertically by including more powerful hardware.   
NoSQL databases also maximize for API-based uses. Many healthcare apps rely on JSON-based APIs, hence storing data in a JSON format reduces the requirement for conversion, thereby enhancing performance and responsiveness in mobile and web applications.

**Disadvantages of NoSQL Document-Oriented Databases**

NoSQL has various shortcomings even if it offers benefits. The dearth of ACID compliance in most NoSQL systems raises one of the main issues. Unlike relational databases, which provide rigorous transactional integrity, NoSQL databases give performance and scalability first priority, which could result in ultimate consistency instead of immediate consistency. For billing transactions and medical records, where data accuracy is vital, this might be challenging.   
Complicated searches are another drawback. Because NoSQL databases do not enable SQL joins, retrieving related data across several collections becomes challenging. If medications are kept apart from patient records, for instance, obtaining a full medical history could call for several searches, therefore affecting performance.

# E. Conceptual and Logical Models

*Description: First, come up with a conceptual model. The conceptual model identifies the entity sets and the relationships among them. For each relationship, identify the connectivity and the participation (optional or mandatory).*

*Now that you know the entity sets, the next step is to develop the logical model by adding attributes. For each entity set, identify the attributes that describe the entity set. This may include references to other entity sets that are involved in relationships. Then, identify the functional dependencies that exist among them. For each functional dependency, identify the determinants and the fields they determine, like this:*

*determinant, or, determinants  attributes, they, determine*

*This becomes the basis for identifying your entity sets, which will become your tables when we move to the physical model in the next section. The attributes listed on the left of the arrows are candidates to become your primary key attributes. Attributes that are references to other entity sets are candidates to become the foreign keys.*

*For entity sets that have multi-attribute determinants, replace them with surrogate keys. This makes it easier to identify each entity in the set and to define foreign keys.*

*Then apply normalization to make sure that your design satisfies First, Second, and Third Normal forms. For 1st Normal Form, make sure that all attributes are indivisible. This may require adding an entity set that lists values that appear in comma-separated lists as individual entities. For 2nd Normal Form, make sure there are no partial dependencies (this won’t be a problem if all your entity sets have single-attribute determinants). Finally, make sure all your entity sets are in 3rd Normal Form. This means that you have to split transitive dependencies into separate entity sets and add relationships between the original entity set and the new ones.*

*Finally, draw the logical model as an ERD. At this point, your design will have entity sets, their relationships, and their attributes. M:N relationships are acceptable at this point, as we’ll remove them in the physical model.*

*Rubric: Your work will be graded as follows:*

* *5 points for identifying all entity sets*
* *5 points for writing each relationship between entity sets as two sentences and correctly identifying their connectivity and participation.*
* *5 points for adding attributes to entity sets and writing the functional dependencies correctly. Replace multi-attribute determinants with surrogate keys.*
* *4 points for performing the normalization steps. Make sure your design is in 3rd Normal Form.*
* *5 points for drawing the ERD for the logical model. At this point, the ERD will show entity sets, relationships, attributes, and primary identifiers. The design may include M:N relationships at this point. We’ll get rid of those in the physical model.*

*Total points possible: 24*

ENTER YOUR RELATIONAL DATABASE DESIGN DESCRIPTION HERE. INCLUDE A PICTURE OF YOUR ERD.

**Conceptual Model:**

Without exploring specific features or implementation details, a conceptual model captures the main elements and relationships in the database. Within the Smart Healthcare Management System (SHMS), the following significant entities have been noted:

Entity Set:

1. **Patient** – Represents the individuals receiving medical care.
2. **Doctor** – Represents medical professionals providing healthcare services.
3. **Appointment** – Links patients and doctors through scheduled visits.
4. **Medical History** – Stores records of past diagnoses and treatments.
5. **Prescription** – Maintains details of medications prescribed by doctors.
6. **Billing** – Tracks payments, insurance coverage, and transactions.
7. **Hospital Inventory** – Stores details of medical equipment and pharmaceutical stock.
8. **Lab Test** – Records laboratory test results for patients.
9. **Staff** – Includes non-medical personnel such as nurses, administrators, and technicians.
10. **Insurance** – Manages patient insurance policies and claims.

**Entity Relationships & Participation Constraints**

* **A patient** must have at least one medical history record (Mandatory Participation).
* **A doctor** can have multiple patients, but each patient is assigned to only one primary doctor at a time (1:M Relationship).
* **A patient** can schedule multiple appointments with multiple doctors (M:N Relationship).
* **An appointment** must involve one patient and one doctor (1:1 Relationship).
* **A prescription** is issued for a particular patient and linked to a doctor's diagnosis (1:M Relationship).
* **A bill** is generated for each patient visit and can be paid partially or fully (1:1 Relationship with Optional Participation).
* **A patient** can have multiple lab test results stored over time (1:M Relationship).
* **Hospital inventory** stores medical supplies linked to prescriptions (M:N Relationship).
* **Staff** members can manage multiple departments but have a primary designation (1:M Relationship).
* **Insurance providers** cover multiple patients, but each patient is associated with a single provider at a time (M:1 Relationship).

**Logical Model**

A logical model extends the conceptual model by defining attributes for each entity and their functional dependencies.

**Entity Attributes and Primary Keys**

**Patient**

* Patient\_ID (Primary Key)
* First\_Name
* Last\_Name
* Date\_of\_Birth
* Gender
* Contact\_Number
* Address
* Insurance\_ID (Foreign Key)

**Doctor**

* Doctor\_ID (Primary Key)
* First\_Name
* Last\_Name
* Specialization
* Contact\_Number
* Email
* Availability

**Appointment**

* Appointment\_ID (Primary Key)
* Patient\_ID (Foreign Key)
* Doctor\_ID (Foreign Key)
* Date
* Time
* Status

**Medical\_History**

* Record\_ID (Primary Key)
* Patient\_ID (Foreign Key)
* Diagnosis
* Treatment
* Date

**Prescription**

* Prescription\_ID (Primary Key)
* Patient\_ID (Foreign Key)
* Doctor\_ID (Foreign Key)
* Medication
* Dosage
* Duration

**Billing**

* Invoice\_ID (Primary Key)
* Patient\_ID (Foreign Key)
* Amount
* Insurance\_Covered (Boolean)
* Payment\_Status
* Date\_Issued
* Date\_Paid

**Hospital\_Inventory**

* Item\_ID (Primary Key)
* Name
* Category
* Stock\_Level
* Supplier\_ID (Foreign Key)

**Lab Test**

* Lab\_ID (Primary Key)
* Patient\_ID (Foreign Key)
* Test\_Name
* Result
* Test\_Date

**Staff**

* Staff\_ID (Primary Key)
* First\_Name
* Last\_Name
* Role
* Contact

**Functional Dependencies and Normalization**

To ensure data integrity, we analyze the functional dependencies and normalize the schema to Third Normal Form (3NF).

**Functional Dependencies**

1. Patient Table

* PatientID → FirstName, LastName, DOB, Gender, Address, InsuranceID
* InsuranceID → CoverageDetails, ProviderName, PolicyNumber (Derived from the Insurance table)

2. Patient\_Contact Table

* ContactID → PatientID, Contact\_Number
* PatientID → ContactID (if each patient has only one contact)

3. Billing Table

* InvoiceID → PatientID, PaymentStatusID, InsuranceCovered, DateIssued, DatePaid, Amount
* PaymentStatusID → Status (From the Payment\_Status table)

4. Payment\_Status Table

* PaymentStatusID → Status

5. Insurance Table

* InsuranceID → CoverageDetails, ProviderName, PolicyNumber

6. Appointment Table

* AppointmentID → PatientID, DoctorID, Date, Time, AppointmentStatusID
* AppointmentStatusID → Status (From the Appointment\_Status table)
* DoctorID → FirstName, LastName, ContactNumber, Email

7. Appointment\_Status Table

* AppointmentStatusID → Status

8. Doctor Table

* DoctorID → FirstName, LastName, ContactNumber, Email
* DoctorID → SpecializationID (From Doctor\_Specialization table)

9. Doctor\_Specialization Table

* SpecializationID, DoctorID → Specialization

10. Prescription Table (Bridge Between Appointment & Medication)

* PrescriptionID → AppointmentID, DoctorID
* AppointmentID → PatientID, DoctorID (Since each appointment links a doctor and a patient)
* PrescriptionID → MedicationID, Duration, Dosage

11. Medication Table

* MedicationID → PrescriptionID, Duration, Medication, Dosage

12. Lab Test Table

* LabID → Test\_Name, Test\_Date, Result

13. Patient\_Test Table (Bridge Between Patient & Lab Test)

* PatientID, LabID → Test\_Date, Result

14. Hospital\_Inventory Table

* ItemID → Name, Category, StockLevel, SupplierID
* SupplierID → Name, Contact

15. Staff Table

* StaffID → FirstName, LastName, ContactNumber
* StaffID → RoleID (From Staff\_Role Table)

16. Staff\_Role Table

* RoleID, StaffID → Role

**Normalization Process**

* **First Normal Form (1NF)**: Ensures atomicity by removing multi-valued attributes.
* **Second Normal Form (2NF)**: Eliminates partial dependencies by ensuring that non-key attributes depend on the whole primary key.
* **Third Normal Form (3NF)**: Eliminates transitive dependencies.

**First Normal Form (1NF):**

* There are no multivalued attributes so schema is already in 1st Normalization Form

**Second Normal Form (2NF)**

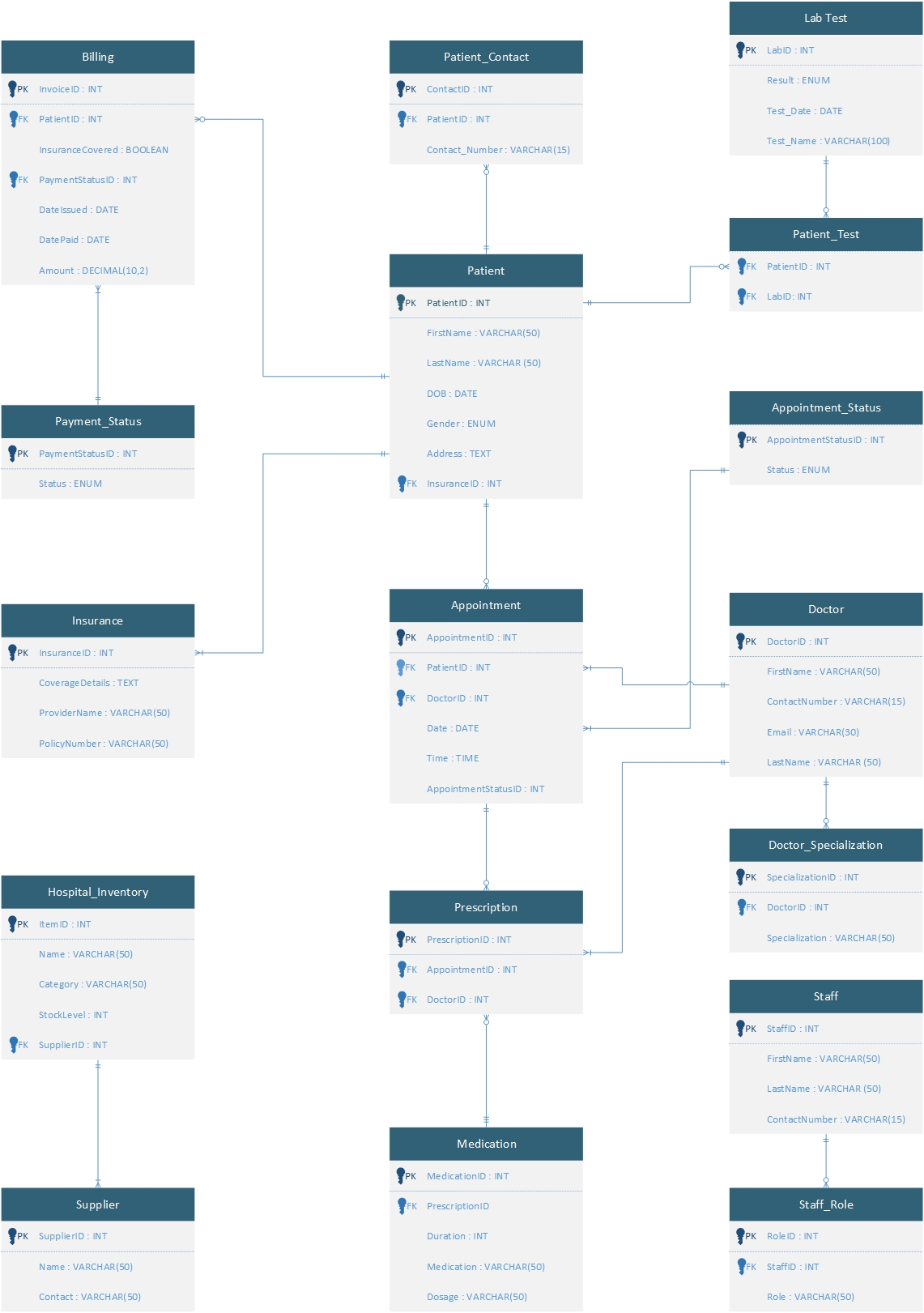
* Schema doesn’t gave any partial key dependency means it is already in 2nd Normalization Form

**Third Normal Form (3NF)**

* There is no transitive dependency so schema is already in 3rd Normalization Form.

**Tables after 3rd NF:**

|  |  |
| --- | --- |
| **Table Name** | **Attributes** |
| **Patient** | Patient\_ID (PK), First\_Name, Last\_Name, DOB, Gender, Address, Insurance\_ID (FK) |
| **Patient\_Contact** | Contact\_ID (PK), Patient\_ID (FK), Contact\_Number |
| **Insurance** | Insurance\_ID (PK), Provider\_Name, Policy\_Number, Coverage\_Details |
| **Doctor** | Doctor\_ID (PK), First\_Name, Last\_Name, Contact\_Number, Email |
| **Doctor\_Specialization** | Specialization\_ID (PK), Doctor\_ID (FK), Specialization |
| **Appointment** | Appointment\_ID (PK), Patient\_ID (FK), Doctor\_ID (FK), Date, Time |
| **Appointment\_Status** | Appointment\_ID (FK), Status |
| **Prescription** | Prescription\_ID (PK), Appointment\_ID (FK), Doctor\_ID (FK) |
| **Medications** | Medication\_ID (PK), Prescription\_ID (FK), Medication, Dosage, Duration |
| **Billing** | Invoice\_ID (PK), Patient\_ID (FK), Amount, Insurance\_Covered, Payment\_Status\_ID (FK), Date\_Issued, Date\_Paid |
| **Payment\_Status** | Payment\_Status\_ID (PK), Status |
| **Hospital\_Inventory** | Item\_ID (PK), Name, Category, Stock\_Level, Supplier\_ID (FK) |
| **Lab\_Test** | Lab\_ID (PK), Test\_Name, Result, Test\_Date |
| **Patient\_Test** | Patient\_ID (FK), Lab\_ID (FK) |
| **Staff** | Staff\_ID (PK), First\_Name, Last\_Name, Contact |
| **Staff\_Role** | Role\_ID (PK), Staff\_ID (FK), Role |
| **Supplier** | Supplier\_ID (PK), Name, Contact |



# F. Physical Model

*Description: This is where you will complete your database design. Add data types, including size constraints, uniqueness constraints, and auto-incrementing for all attributes. Implement relationships using foreign keys. Replace many-to-many relationships with two one-to-many relationships using bridge entity sets. Add additional entity sets that you think could be helpful for storing the acceptable values of particular attributes. (For example, if you were storing student data, valid student statuses might include Good Standing, Graduated, On Probation, Expelled. Put those in a table and create a relationship back to the student table). Draw the ERD for the physical model.*

*Using the final ERD, write the SQL DDL statements needed to create the database, its tables, and the relationships among them. Run these statements in MySQL to build your database. Provide screen shots that show the database you built in MySQL, including its tables and descriptions of some of the tables. To show a list of databases and a list of the tables in a particular database, use the show command. To see a description for a table, use the describe command.*

*Rubric: Your work will be graded as follows:*

* *3 points for introducing bridge entity sets (if necessary)*
* *3 points for adding data types and other constraints on the data.*
* *3 points for introducing other entity sets and their relationships that help enforce what values can be assigned to particular attributes (if necessary)*
* *5 points for drawing the ERD for the physical model. If you used Vertabelo, the resulting ERD must be free of errors and warnings*
* *6 points for generating the SQL scripts that build the database and then running the script in mysql. Demonstrate that the script built the database and its tables with screenshots that show that you ran the show and describe commands.*

*You will be penalized 4 points if your database doesn’t have at least 8 appropriately defined tables.*

*Total points possible: 20*

DESCRIBE THE STEPS YOU TOOK TO COMPLETE THE PHYSICAL MODEL. THEN SHOW THE ERD FOR THE PHYSICAL MODEL. THEN SHOW THE SQL COMMANDS THAT BUILD THE DATABASE. (FOR THIS, YOU MAY REFER TO A PARTICULAR FILE IN YOUR GITHUB REPOSITORY. MAKE SURE YOU INVITE ME AS A COLLABORATOR ON YOUR REPOSITORY SO THAT I CAN ACCESS THE SCRIPT.) FINALLY, SHOW SCREEN SHOTS THAT PROVE THAT YOU BUILT THE DATABASE AND ITS STRUCTURES IN MYSQL.

**Data Types & Constraints**

* **Primary Keys (PK)**: Defined as INT AUTO\_INCREMENT PRIMARY KEY.
* **Foreign Keys (FK)**: Defined with FOREIGN KEY constraints ensuring referential integrity.
* **Data Types**: VARCHAR, ENUM, TEXT, DATE, BOOLEAN, DECIMAL chosen appropriately.
* **Uniqueness Constraints**: Applied to attributes like Email, Contact\_Number, Policy\_Number.
* **Normalization**: All transitive and partial dependencies are removed.

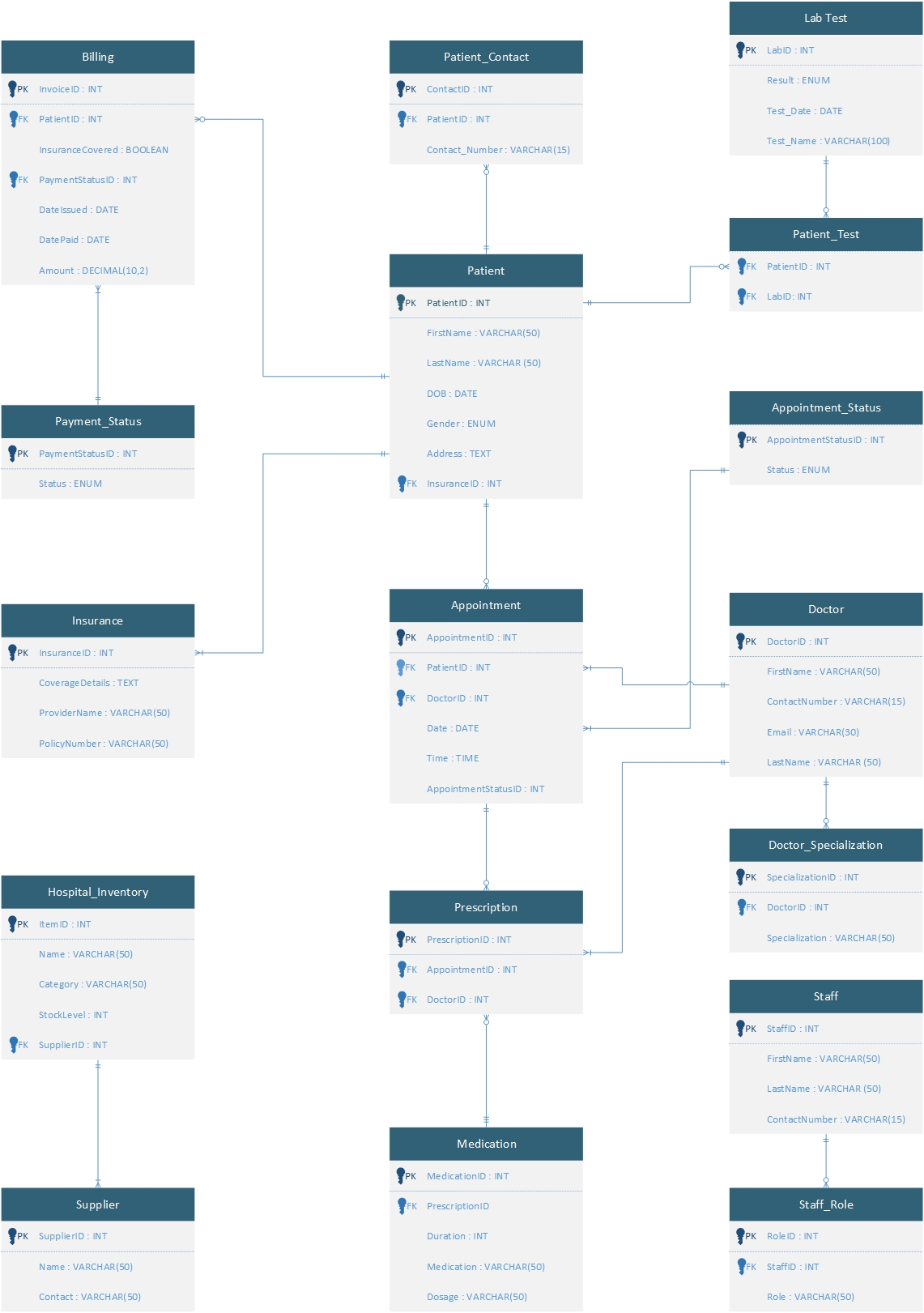
**Bridge Entity Sets**

Patient\_Test (Bridges Patient and Lab\_Test)

Prescription (Bridges Appointment and Medication)

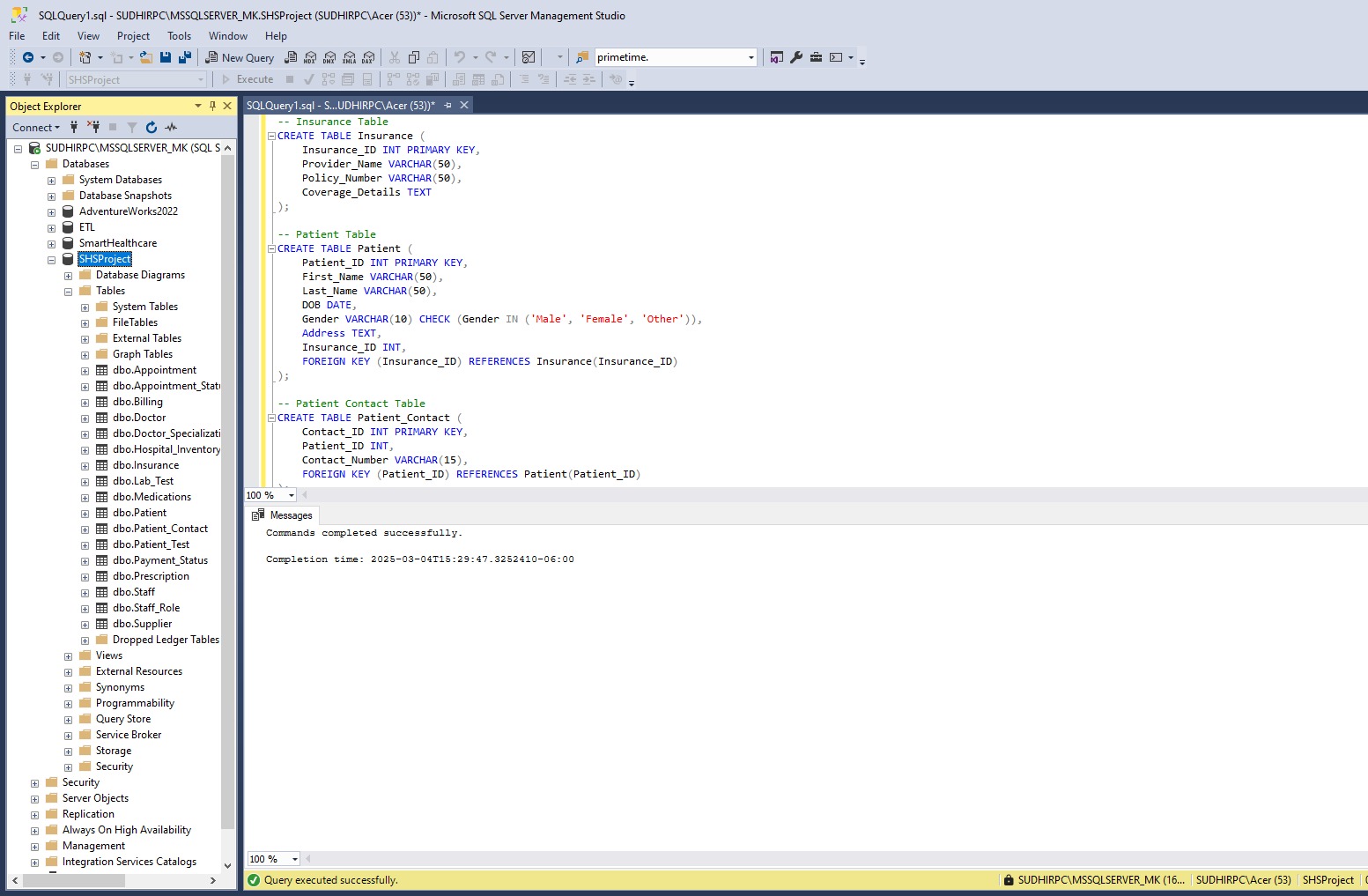
**ERD:**

It was already completed in previous section when 3NF ERD was created.



**DDLs**

|  |
| --- |
| -- Insurance Table  CREATE TABLE Insurance (  Insurance\_ID INT PRIMARY KEY,  Provider\_Name VARCHAR(50),  Policy\_Number VARCHAR(50),  Coverage\_Details TEXT  );  -- Patient Table  CREATE TABLE Patient (  Patient\_ID INT PRIMARY KEY,  First\_Name VARCHAR(50),  Last\_Name VARCHAR(50),  DOB DATE,  Gender VARCHAR(10) CHECK (Gender IN ('Male', 'Female', 'Other')),  Address TEXT,  Insurance\_ID INT,  FOREIGN KEY (Insurance\_ID) REFERENCES Insurance(Insurance\_ID)  );  -- Patient Contact Table  CREATE TABLE Patient\_Contact (  Contact\_ID INT PRIMARY KEY,  Patient\_ID INT,  Contact\_Number VARCHAR(15),  FOREIGN KEY (Patient\_ID) REFERENCES Patient(Patient\_ID)  );  -- Doctor Table  CREATE TABLE Doctor (  Doctor\_ID INT PRIMARY KEY,  First\_Name VARCHAR(50),  Last\_Name VARCHAR(50),  Contact\_Number VARCHAR(15),  Email VARCHAR(80)  );  -- Doctor Specialization Table  CREATE TABLE Doctor\_Specialization (  Specialization\_ID INT PRIMARY KEY,  Doctor\_ID INT,  Specialization VARCHAR(50),  FOREIGN KEY (Doctor\_ID) REFERENCES Doctor(Doctor\_ID)  );  -- Appointment Table  CREATE TABLE Appointment (  Appointment\_ID INT PRIMARY KEY,  Patient\_ID INT,  Doctor\_ID INT,  Date DATE,  Time TIME,  FOREIGN KEY (Patient\_ID) REFERENCES Patient(Patient\_ID),  FOREIGN KEY (Doctor\_ID) REFERENCES Doctor(Doctor\_ID)  );  -- Appointment Status Table  CREATE TABLE Appointment\_Status (  Appointment\_ID INT PRIMARY KEY,  Status VARCHAR(20) CHECK (Status IN ('Scheduled', 'Completed', 'Cancelled', 'Rescheduled')),  FOREIGN KEY (Appointment\_ID) REFERENCES Appointment(Appointment\_ID)  );  -- Prescription Table  CREATE TABLE Prescription (  Prescription\_ID INT PRIMARY KEY,  Appointment\_ID INT,  Doctor\_ID INT,  FOREIGN KEY (Appointment\_ID) REFERENCES Appointment(Appointment\_ID),  FOREIGN KEY (Doctor\_ID) REFERENCES Doctor(Doctor\_ID)  );  -- Medications Table  CREATE TABLE Medications (  Medication\_ID INT PRIMARY KEY,  Prescription\_ID INT,  Medication VARCHAR(50),  Dosage VARCHAR(50),  Duration INT,  FOREIGN KEY (Prescription\_ID) REFERENCES Prescription(Prescription\_ID)  );  -- Payment Status Table  CREATE TABLE Payment\_Status (  Payment\_Status\_ID INT PRIMARY KEY,  Status VARCHAR(20) CHECK (Status IN ('Paid', 'Pending', 'Cancelled', 'Failed'))  );  -- Billing Table  CREATE TABLE Billing (  Invoice\_ID INT PRIMARY KEY,  Patient\_ID INT,  Amount DECIMAL(10,2),  Insurance\_Covered BIT, -- BOOLEAN is replaced with BIT in SSMS  Payment\_Status\_ID INT,  Date\_Issued DATE,  Date\_Paid DATE,  FOREIGN KEY (Patient\_ID) REFERENCES Patient(Patient\_ID),  FOREIGN KEY (Payment\_Status\_ID) REFERENCES Payment\_Status(Payment\_Status\_ID)  );  -- Supplier Table  CREATE TABLE Supplier (  Supplier\_ID INT PRIMARY KEY,  Name VARCHAR(50),  Contact VARCHAR(50)  );  -- Hospital Inventory Table  CREATE TABLE Hospital\_Inventory (  Item\_ID INT PRIMARY KEY,  Name VARCHAR(50),  Category VARCHAR(50),  Stock\_Level INT,  Supplier\_ID INT,  FOREIGN KEY (Supplier\_ID) REFERENCES Supplier(Supplier\_ID)  );  -- Lab Test Table  CREATE TABLE Lab\_Test (  Lab\_ID INT PRIMARY KEY,  Test\_Name VARCHAR(100),  Result VARCHAR(20) CHECK (Result IN ('Positive', 'Negative', 'Inconclusive')),  Test\_Date DATE  );  -- Patient Test Table (Bridge between Patient and Lab\_Test)  CREATE TABLE Patient\_Test (  Patient\_ID INT,  Lab\_ID INT,  PRIMARY KEY (Patient\_ID, Lab\_ID),  FOREIGN KEY (Patient\_ID) REFERENCES Patient(Patient\_ID),  FOREIGN KEY (Lab\_ID) REFERENCES Lab\_Test(Lab\_ID)  );  -- Staff Table  CREATE TABLE Staff (  Staff\_ID INT PRIMARY KEY,  First\_Name VARCHAR(50),  Last\_Name VARCHAR(50),  Contact VARCHAR(15)  );  -- Staff Role Table  CREATE TABLE Staff\_Role (  Role\_ID INT PRIMARY KEY,  Staff\_ID INT,  Role VARCHAR(50),  FOREIGN KEY (Staff\_ID) REFERENCES Staff(Staff\_ID)  ); |



# G. Populate the database with data

*Description: You built the database in section F, and it now exists in mysql. Now populate it with your data. Take your original data source or sources and generate insert statements from them. Store the insert statements in a text file, and then use the mysql source command to run these insert statements to populate the various table structures. Generating the necessary insert statements may require writing Python scripts or manipulating Excel databases to convert the data from your original data sources.*

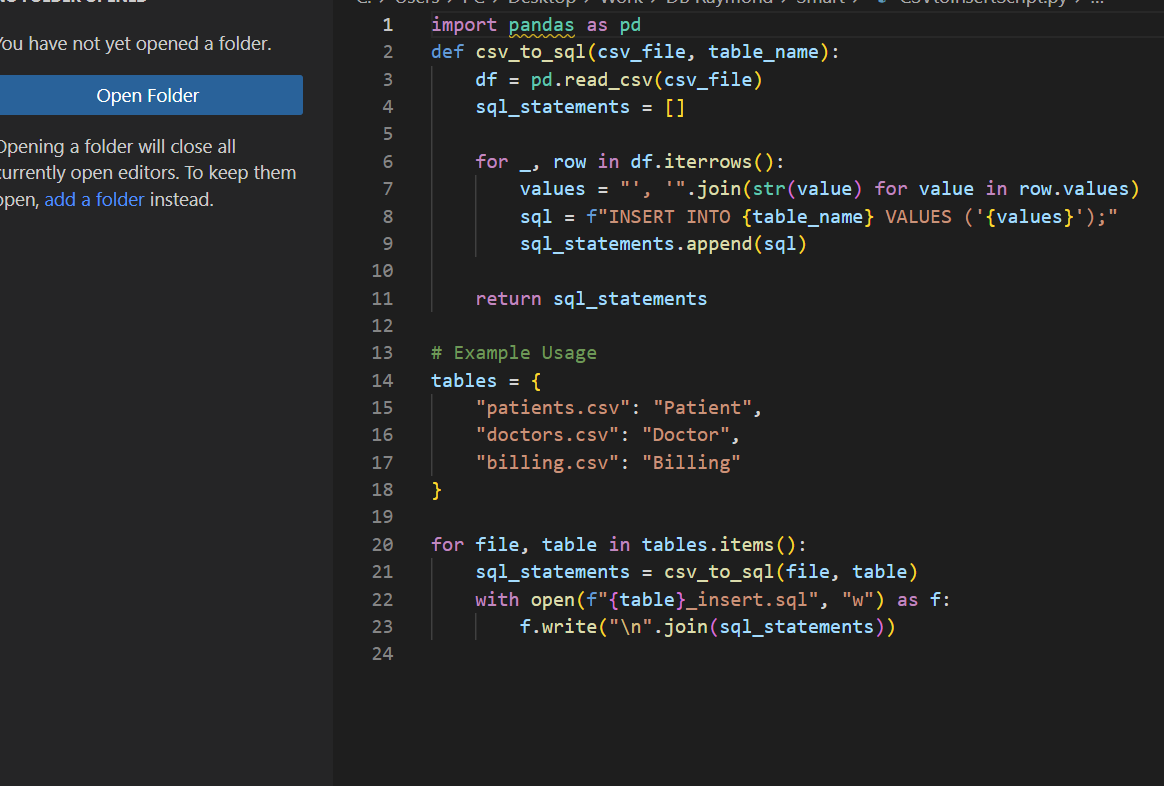
*Rubric: Your work will be grades as follows:*

* *Explain step-by-step and very clearly how you created the required SQL statements from your initial data. Write it as a set of instructions. 5 points*
* *Show the file of insert statements that you ran in MySQL. You may do this either by including the listing in this report or by identifying the file in your GitHub that contains the insert statements. Make sure I have access to your GitHub repository. 4 points*
* *Show screenshots of the data in your MySQL database. To do this, run select statements for each table and show screen shots of what is displayed: 5 points*

*Total points possible: 14*

ENTER YOUR DDL WORK HERE

Reading CSV files to generate Insert statements:

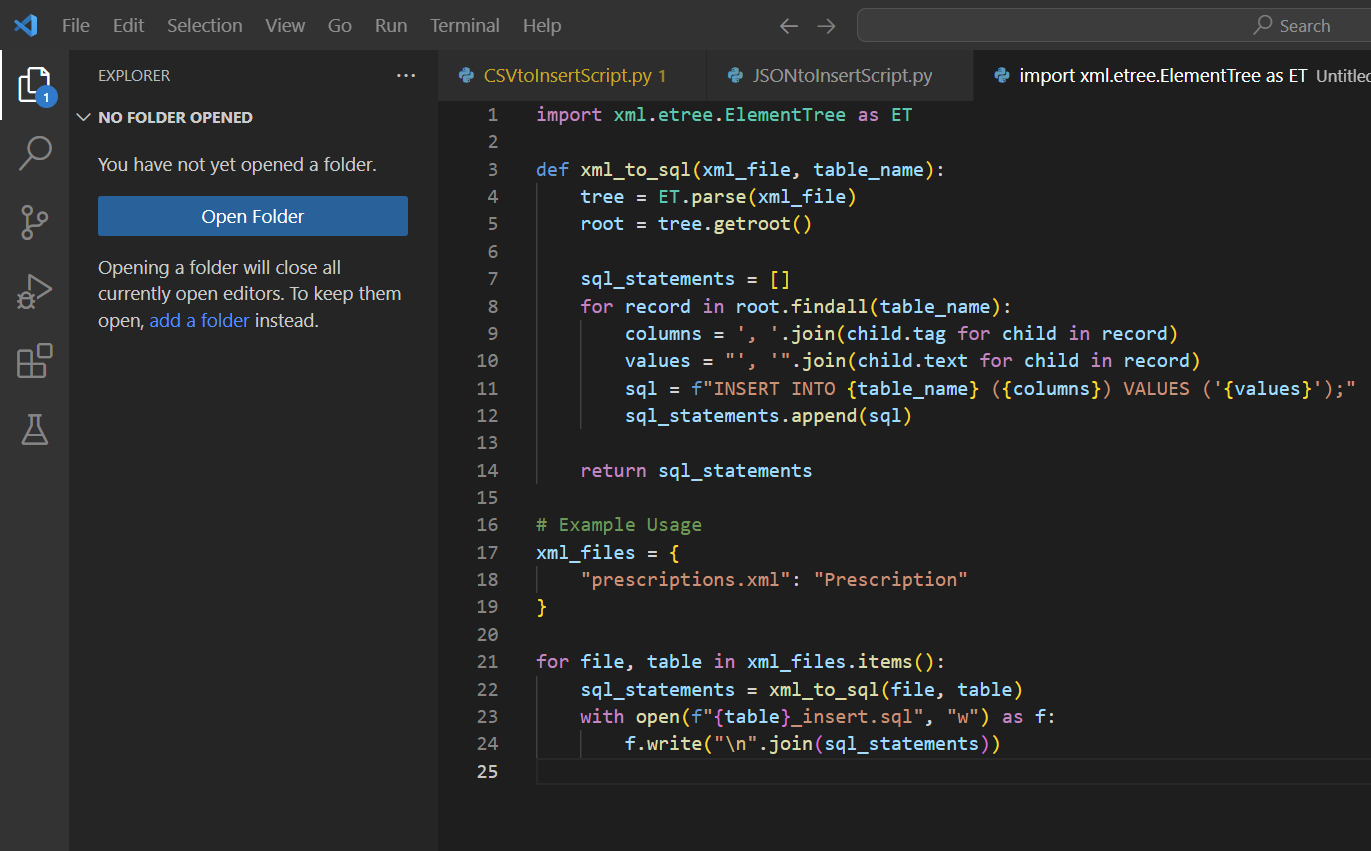


Reading JSON files to generate Insert statements:

A screenshot of a computer program

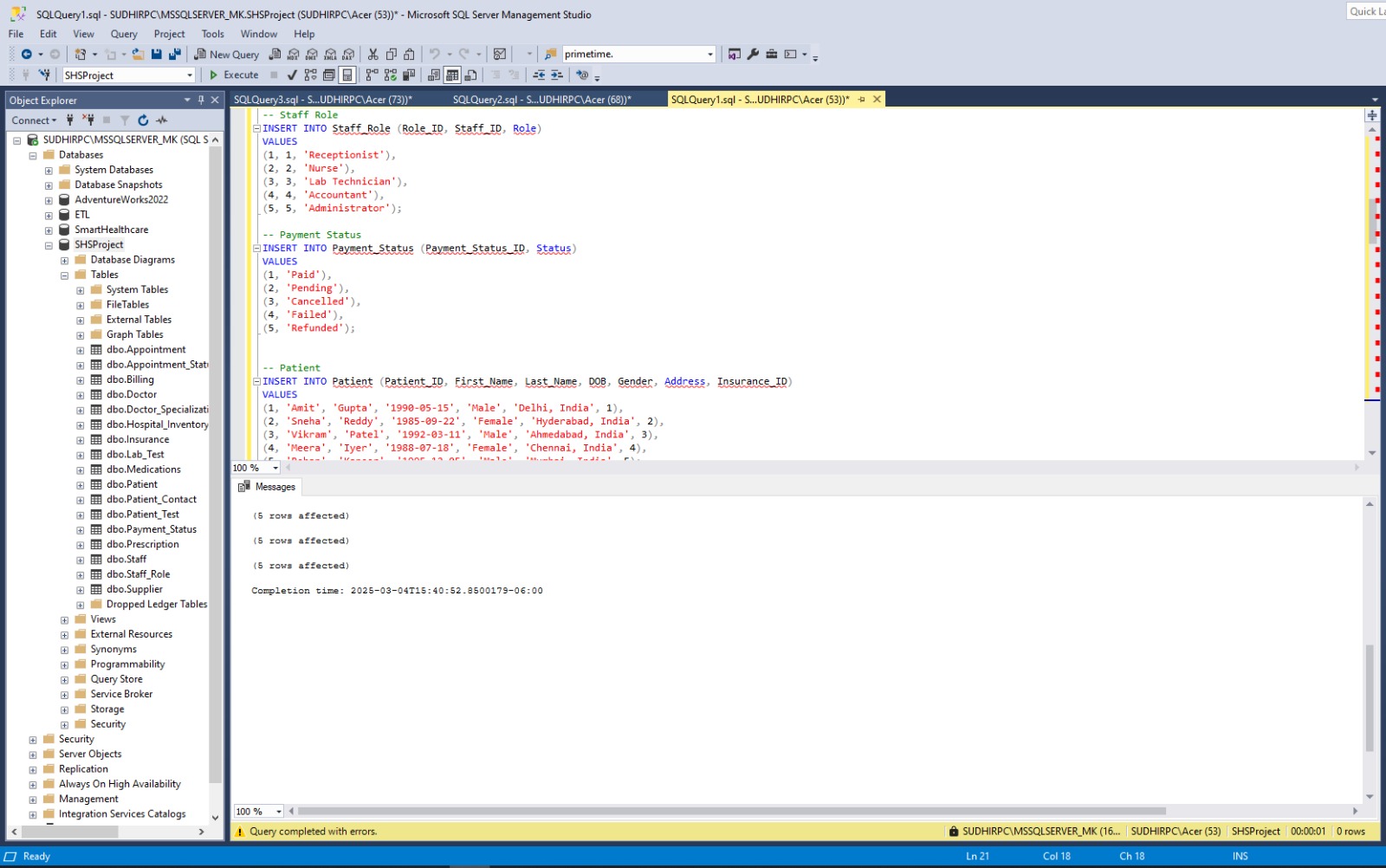
AI-generated content may be incorrect.

Reading XML files to generate Insert statements:



DDLs:

|  |
| --- |
| -- Insurance  INSERT INTO Insurance (Insurance\_ID, Provider\_Name, Policy\_Number, Coverage\_Details)  VALUES  (1, 'Apollo Munich', 'POL123456', 'Covers hospitalization and critical illness'),  (2, 'Max Bupa', 'POL789012', 'Covers general OPD and consultation'),  (3, 'Star Health', 'POL345678', 'Comprehensive family coverage'),  (4, 'HDFC Ergo', 'POL901234', 'Covers pre and post-hospitalization'),  (5, 'Bajaj Allianz', 'POL567890', 'Cashless claim benefits');  -- Doctor  INSERT INTO Doctor (Doctor\_ID, First\_Name, Last\_Name, Contact\_Number, Email)  VALUES  (1, 'Rajesh', 'Sharma', '9876543210', 'rajesh.sharma@gmail.com'),  (2, 'Anita', 'Verma', '8765432109', 'anita.verma@gmail.com'),  (3, 'Sandeep', 'Joshi', '9123456780', 'sandeep.joshi@gmail.com'),  (4, 'Priya', 'Desai', '9234567891', 'priya.desai@gmail.com'),  (5, 'Amit', 'Singh', '9345678902', 'amit.singh@gmail.com');  -- Supplier  INSERT INTO Supplier (Supplier\_ID, Name, Contact)  VALUES  (1, 'MediCare Supplies', 'medicare@gmail.com'),  (2, 'Apollo Medicals', 'apollo.meds@gmail.com'),  (3, 'HealthCare Ltd.', 'healthcare@gmail.com'),  (4, 'QuickMeds', 'quickmeds@gmail.com'),  (5, 'Wellness Pharma', 'wellnesspharma@gmail.com');  -- Staff  INSERT INTO Staff (Staff\_ID, First\_Name, Last\_Name, Contact)  VALUES  (1, 'Suresh', 'Kumar', '8899776655'),  (2, 'Priya', 'Mishra', '7788996655'),  (3, 'Alok', 'Tiwari', '6677889900'),  (4, 'Deepika', 'Rana', '5566778899'),  (5, 'Rahul', 'Jain', '4455667788');  -- Staff Role  INSERT INTO Staff\_Role (Role\_ID, Staff\_ID, Role)  VALUES  (1, 1, 'Receptionist'),  (2, 2, 'Nurse'),  (3, 3, 'Lab Technician'),  (4, 4, 'Accountant'),  (5, 5, 'Administrator');  -- Payment Status  INSERT INTO Payment\_Status (Payment\_Status\_ID, Status)  VALUES  (1, 'Paid'),  (2, 'Pending'),  (3, 'Cancelled'),  (4, 'Failed'),  (5, 'Pending');  -- Patient  INSERT INTO Patient (Patient\_ID, First\_Name, Last\_Name, DOB, Gender, Address, Insurance\_ID)  VALUES  (1, 'Amit', 'Gupta', '1990-05-15', 'Male', 'Delhi, India', 1),  (2, 'Sneha', 'Reddy', '1985-09-22', 'Female', 'Hyderabad, India', 2),  (3, 'Vikram', 'Patel', '1992-03-11', 'Male', 'Ahmedabad, India', 3),  (4, 'Meera', 'Iyer', '1988-07-18', 'Female', 'Chennai, India', 4),  (5, 'Rohan', 'Kapoor', '1995-12-05', 'Male', 'Mumbai, India', 5);  -- Patient Contact  INSERT INTO Patient\_Contact (Contact\_ID, Patient\_ID, Contact\_Number)  VALUES  (1, 1, '9988776655'),  (2, 2, '9876554433'),  (3, 3, '9765432109'),  (4, 4, '9654321098'),  (5, 5, '9543210987');  -- Doctor Specialization  INSERT INTO Doctor\_Specialization (Specialization\_ID, Doctor\_ID, Specialization)  VALUES  (1, 1, 'Cardiology'),  (2, 2, 'Pediatrics'),  (3, 3, 'Orthopedics'),  (4, 4, 'Neurology'),  (5, 5, 'General Medicine');  -- Appointment  INSERT INTO Appointment (Appointment\_ID, Patient\_ID, Doctor\_ID, Date, Time)  VALUES  (1, 1, 1, '2025-03-05', '10:00:00'),  (2, 2, 2, '2025-03-06', '14:30:00'),  (3, 3, 3, '2025-03-07', '09:45:00'),  (4, 4, 4, '2025-03-08', '11:15:00'),  (5, 5, 5, '2025-03-09', '16:00:00');  -- Appointment Status  INSERT INTO Appointment\_Status (Appointment\_ID, Status)  VALUES  (1, 'Scheduled'),  (2, 'Completed'),  (3, 'Cancelled'),  (4, 'Rescheduled'),  (5, 'Scheduled');  -- Prescription  INSERT INTO Prescription (Prescription\_ID, Appointment\_ID, Doctor\_ID)  VALUES  (1, 1, 1),  (2, 2, 2),  (3, 3, 3),  (4, 4, 4),  (5, 5, 5);  -- Medications  INSERT INTO Medications (Medication\_ID, Prescription\_ID, Medication, Dosage, Duration)  VALUES  (1, 1, 'Aspirin', '75mg', 7),  (2, 2, 'Paracetamol', '500mg', 5),  (3, 3, 'Ibuprofen', '400mg', 10),  (4, 4, 'Amoxicillin', '250mg', 7),  (5, 5, 'Cough Syrup', '10ml', 5);  INSERT INTO Billing (Invoice\_ID, Patient\_ID, Amount, Insurance\_Covered, Payment\_Status\_ID, Date\_Issued, Date\_Paid)  VALUES  (1, 1, 2500.00, 1, 1, '2025-03-05', '2025-03-05'), -- Paid (Valid)  (2, 2, 1800.00, 0, 2, '2025-03-06', NULL), -- Pending (Valid)  (3, 3, 3200.00, 1, 1, '2025-03-07', '2025-03-07'), -- Paid (Valid)  (4, 4, 5000.00, 0, 3, '2025-03-08', NULL), -- Cancelled (Valid)  (5, 5, 1500.00, 1, 4, '2025-03-09', NULL); -- Failed (Valid)  -- Hospital Inventory  INSERT INTO Hospital\_Inventory (Item\_ID, Name, Category, Stock\_Level, Supplier\_ID)  VALUES  (1, 'Syringes', 'Medical Equipment', 100, 1),  (2, 'Bandages', 'First Aid', 200, 2),  (3, 'Thermometers', 'Medical Equipment', 50, 3),  (4, 'Stethoscopes', 'Diagnostic Tools', 30, 4),  (5, 'Gloves', 'Protective Gear', 500, 5);  -- Lab Test  INSERT INTO Lab\_Test (Lab\_ID, Test\_Name, Result, Test\_Date)  VALUES  (1, 'Blood Test', 'Positive', '2025-03-05'),  (2, 'X-Ray', 'Negative', '2025-03-06'),  (3, 'MRI Scan', 'Inconclusive', '2025-03-07'),  (4, 'ECG', 'Positive', '2025-03-08'),  (5, 'Urine Test', 'Negative', '2025-03-09');  -- Patient Test (Bridge Between Patient and Lab Test)  INSERT INTO Patient\_Test (Patient\_ID, Lab\_ID)  VALUES  (1, 1),  (2, 2),  (3, 3),  (4, 4),  (5, 5); |



Select Queries:

|  |
| --- |
| -- Retrieve all patients  SELECT \* FROM Patient;  -- Retrieve all patient contact details  SELECT \* FROM Patient\_Contact;  -- Retrieve all insurance details  SELECT \* FROM Insurance;  -- Retrieve all records from each table  SELECT \* FROM Patient;  SELECT \* FROM Patient\_Contact;  SELECT \* FROM Insurance;  SELECT \* FROM Doctor;  SELECT \* FROM Doctor\_Specialization;  SELECT \* FROM Appointment;  SELECT \* FROM Appointment\_Status;  SELECT \* FROM Prescription;  SELECT \* FROM Medications;  SELECT \* FROM Billing;  SELECT \* FROM Payment\_Status;  SELECT \* FROM Hospital\_Inventory;  SELECT \* FROM Lab\_Test;  SELECT \* FROM Patient\_Test;  SELECT \* FROM Staff;  SELECT \* FROM Staff\_Role;  SELECT \* FROM Supplier; |

A screenshot of a computer

AI-generated content may be incorrect.

**Note: Github repository is updated.**

# H. Data Manipulation Language (DML) Scripts

*Description: Write the SQL commands for twelve queries. Two queries should be insert statements, two should update statements, one should be a delete statement, one should be a simple select statement that selects a subset of the rows and columns from one table, two should be a select statements that select data from a joining of two tables, two should use summary functions to generate statistics about the data, one should be a multi-table query, and one should be another query of your choice. Show the queries and screenshots of the results in your Word document, and save your queries in a commented sql script to GitHub.*

*Rubric: Your work will be graded as follows:*

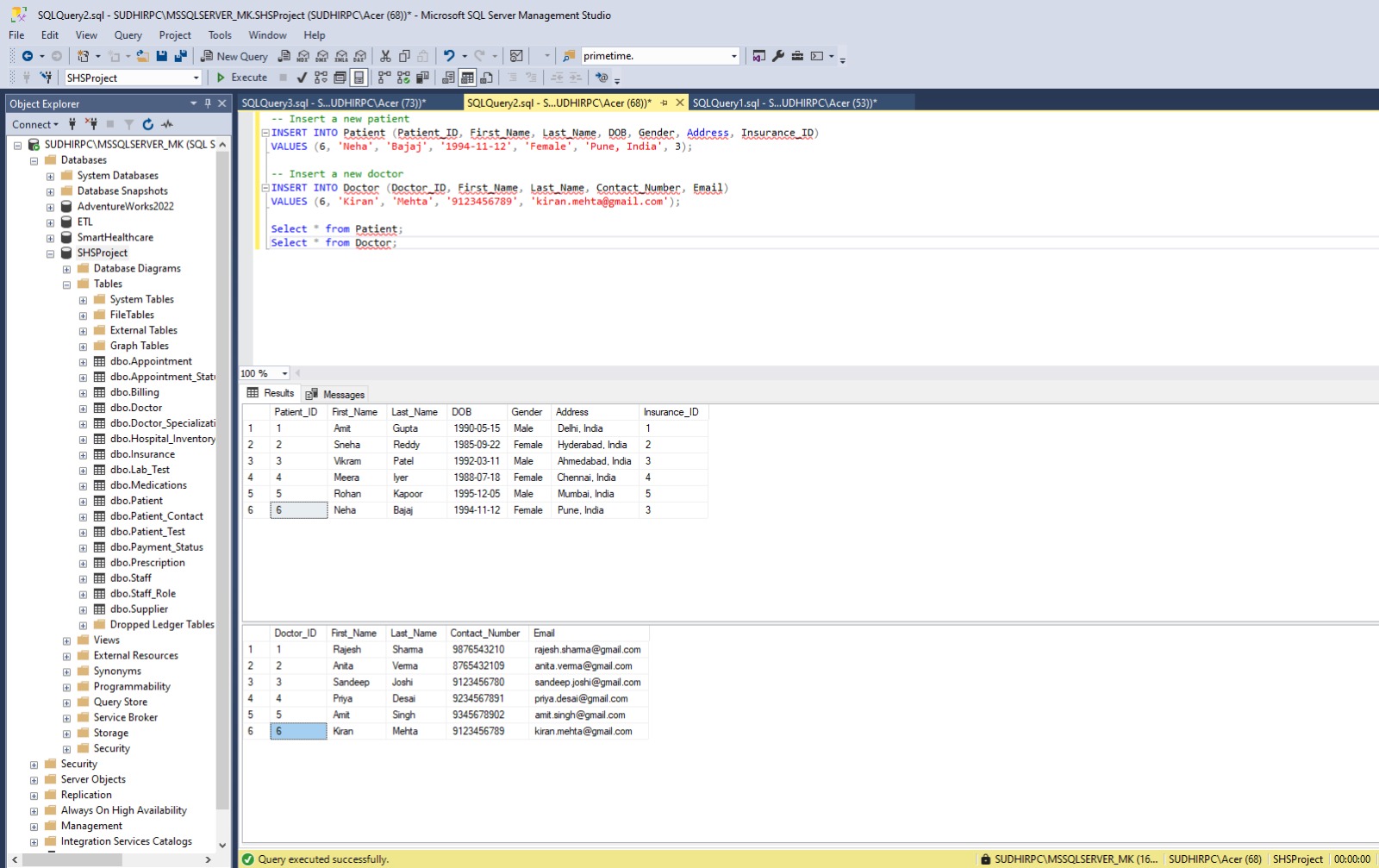
* *1 point each for the two insert statements*
* *1 point each for the two update statements*
* *1 point for the delete statement*
* *1 point for the simple select statement*
* *2 points each for the 2 join statements*
* *2 points each for the two that use summary statements*
* *2 points for the multi-table query*
* *2 points for the query of your choice.*
* *6 points for showing the query and a screenshot of the corresponding result set back-to-back for each of these queries in your Word document.*

*Total points possible: 24*

ENTER DML WORK HERE

Insert statements:

|  |
| --- |
| -- Insert a new patient  INSERT INTO Patient (Patient\_ID, First\_Name, Last\_Name, DOB, Gender, Address, Insurance\_ID)  VALUES (6, 'Neha', 'Bajaj', '1994-11-12', 'Female', 'Pune, India', 3);  -- Insert a new doctor  INSERT INTO Doctor (Doctor\_ID, First\_Name, Last\_Name, Contact\_Number, Email)  VALUES (6, 'Kiran', 'Mehta', '9123456789', 'kiran.mehta@gmail.com'); |



Update Statements:

|  |
| --- |
| -- Update the contact number of a patient  UPDATE Patient\_Contact  SET Contact\_Number = '9998887776'  WHERE Patient\_ID = 6;  -- Update the status of an appointment  UPDATE Appointment\_Status  SET Status = 'Completed'  WHERE Appointment\_ID = 2; |

A screenshot of a computer

AI-generated content may be incorrect.

Delete statement:

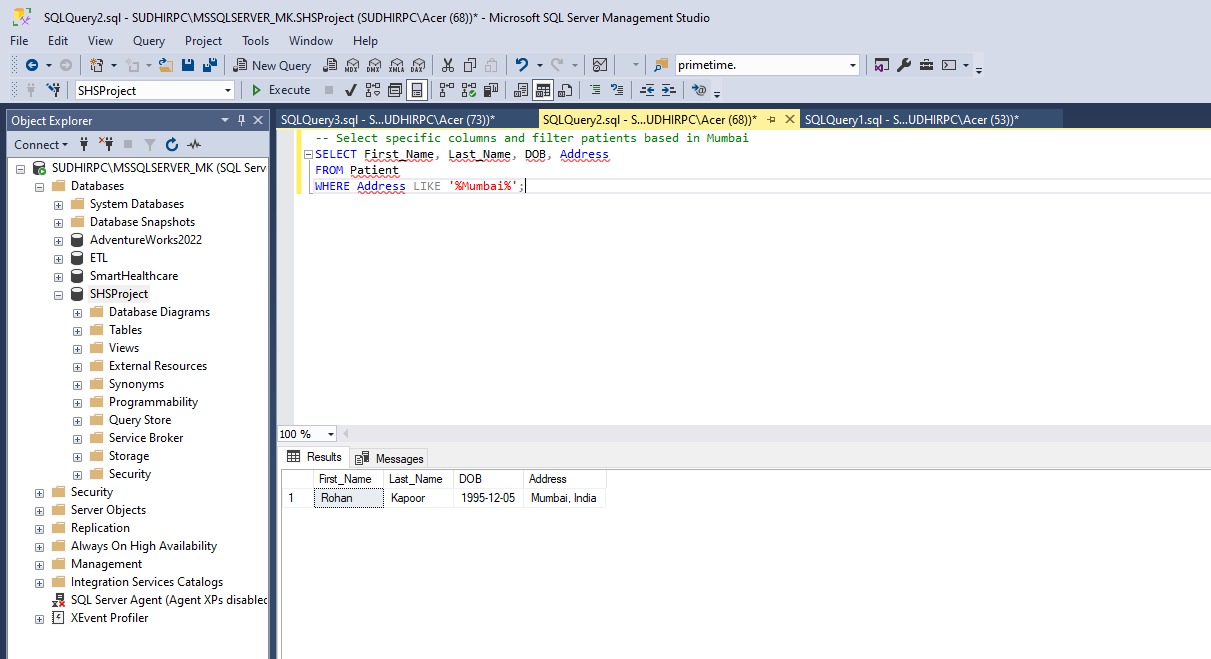
|  |
| --- |
| DELETE FROM Billing WHERE Invoice\_ID = 3; |

A computer screen shot of a computer

AI-generated content may be incorrect.

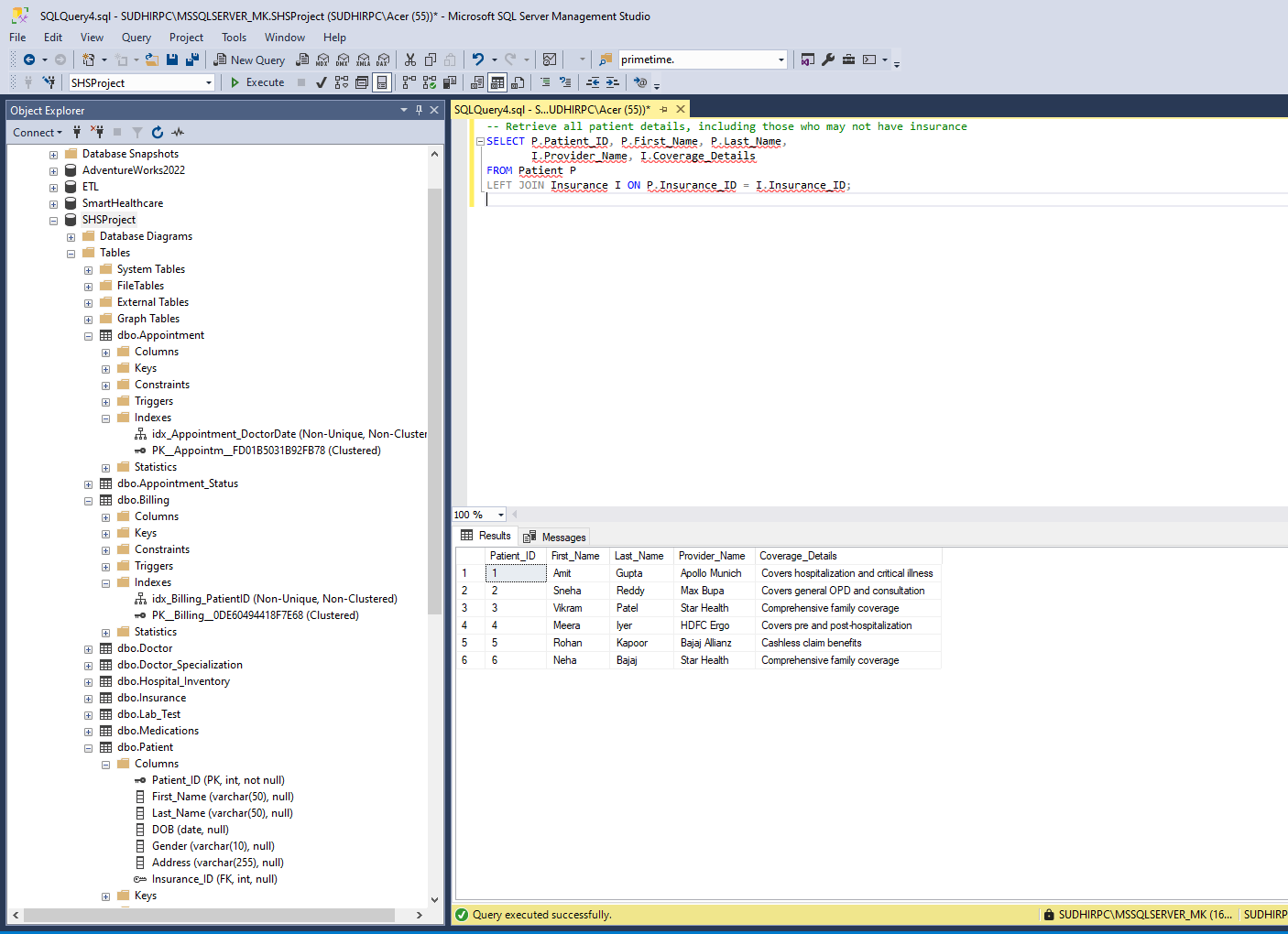
Simple select Statement

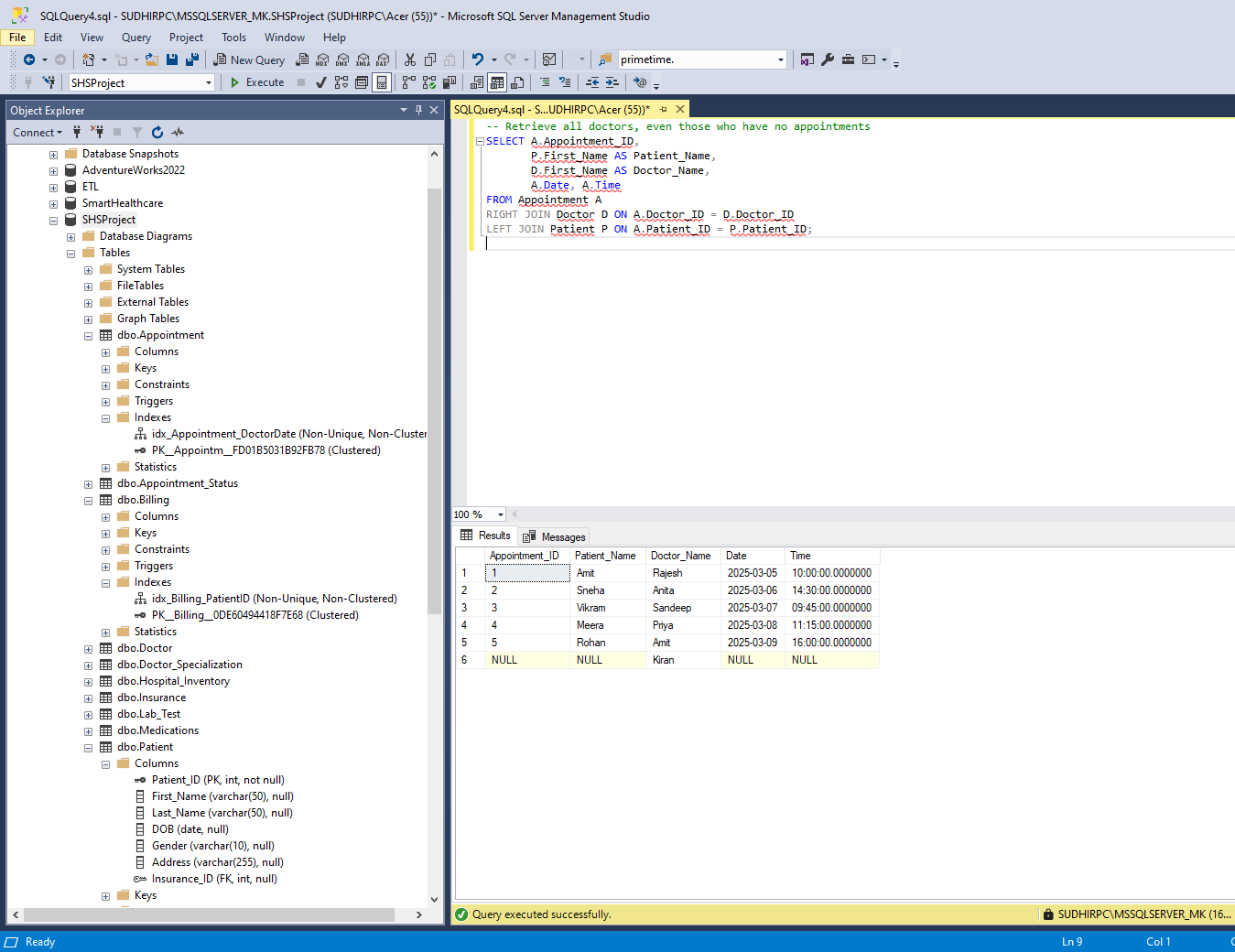
|  |
| --- |
| -- Select specific columns and filter patients based in Mumbai  SELECT First\_Name, Last\_Name, DOB, Address  FROM Patient  WHERE Address LIKE '%Mumbai%'; |



Join Statement

|  |
| --- |
| -- Retrieve all doctors, even those who have no appointments  SELECT A.Appointment\_ID,  P.First\_Name AS Patient\_Name,  D.First\_Name AS Doctor\_Name,  A.Date, A.Time  FROM Appointment A  RIGHT JOIN Doctor D ON A.Doctor\_ID = D.Doctor\_ID  LEFT JOIN Patient P ON A.Patient\_ID = P.Patient\_ID;  -- Retrieve all patient details, including those who may not have insurance  SELECT P.Patient\_ID, P.First\_Name, P.Last\_Name,  I.Provider\_Name, I.Coverage\_Details  FROM Patient P  LEFT JOIN Insurance I ON P.Insurance\_ID = I.Insurance\_ID; |





Summary Functions Statement

|  |
| --- |
| -- Count the total number of patients per city  SELECT Address, COUNT(Patient\_ID) AS Patient\_Count  FROM Patient  GROUP BY Address;  -- Find the average billing amount for all patients  SELECT AVG(Amount) AS Avg\_Bill\_Amount  FROM Billing; |

A screenshot of a computer

AI-generated content may be incorrect.

Multi-Table Query Statement

|  |
| --- |
| -- Retrieve appointment details, the assigned doctor, and prescription details  SELECT A.Appointment\_ID, P.First\_Name AS Patient\_Name, D.First\_Name AS Doctor\_Name, M.Medication, M.Dosage  FROM Appointment A  JOIN Patient P ON A.Patient\_ID = P.Patient\_ID  JOIN Doctor D ON A.Doctor\_ID = D.Doctor\_ID  JOIN Prescription PR ON A.Appointment\_ID = PR.Appointment\_ID  JOIN Medications M ON PR.Prescription\_ID = M.Prescription\_ID; |

A computer screen shot of a computer screen

AI-generated content may be incorrect.

Custom Query of Choice Statement

|  |
| --- |
| -- Find doctors who have prescribed more than one medication  SELECT D.First\_Name, D.Last\_Name, COUNT(M.Medication\_ID) AS Total\_Prescriptions  FROM Doctor D  JOIN Prescription PR ON D.Doctor\_ID = PR.Doctor\_ID  JOIN Medications M ON PR.Prescription\_ID = M.Prescription\_ID  GROUP BY D.First\_Name, D.Last\_Name  HAVING COUNT(M.Medication\_ID) > 1; |

A computer screen shot of a computer

AI-generated content may be incorrect.

# I. Indexes

*Description: Improve the performance of your design by adding indexes to various tables. Show the SQL needed to add the indexes. Explain why you chose the ones you added. Explain how you would demonstrate the impact the indexes had on the performance of various queries.*

*Rubric: Your work will be graded as follows:*

* *3 points for clearly defining at least three indexes and explaining why you chose them.*
* *3 points for showing the sql needed to generate the indexes*
* *2 points for explaining how you would demonstrate the performance improvement afforded by the indexes.*

*Total points possible: 8*

ENTER YOUR INDEX WORK HERE

**Index 1:** Billing (Payment\_Status\_ID, Date\_Issued)

**Justification:**

* Helps quickly find unpaid or completed bills based on payment status and issue date.

|  |
| --- |
| CREATE INDEX idx\_Billing\_PaymentStatusDate ON Billing (Payment\_Status\_ID, Date\_Issued); |

**Index 2:** Lab\_Test (Test\_Name, Result)

**Justification:**

* Speeds up searches for patients undergoing specific tests. Lab staff often filter tests by name and result.

|  |
| --- |
| CREATE INDEX idx\_LabTest\_NameResult ON Lab\_Test (Test\_Name, Result); |

**Index 3:** idx\_Appointment\_DoctorDate on Appointment(Doctor\_ID, Date)

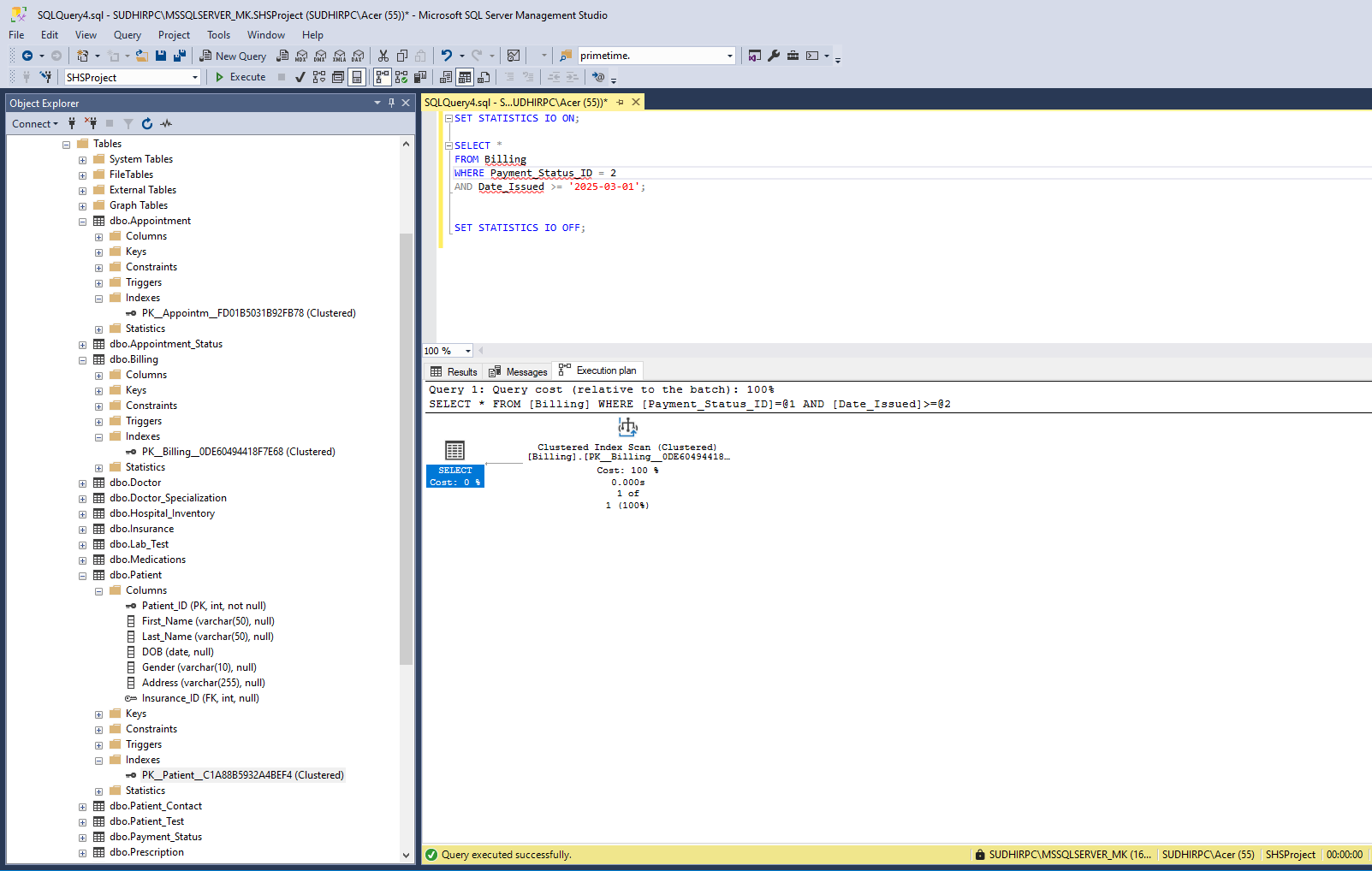
**Justification:**

* Doctors' schedules are queried frequently

|  |
| --- |
| CREATE INDEX idx\_Appointment\_DoctorDate ON Appointment(Doctor\_ID, Date); |



**Index Performance Using SQL Server Execution Plan and Statistics IO:**



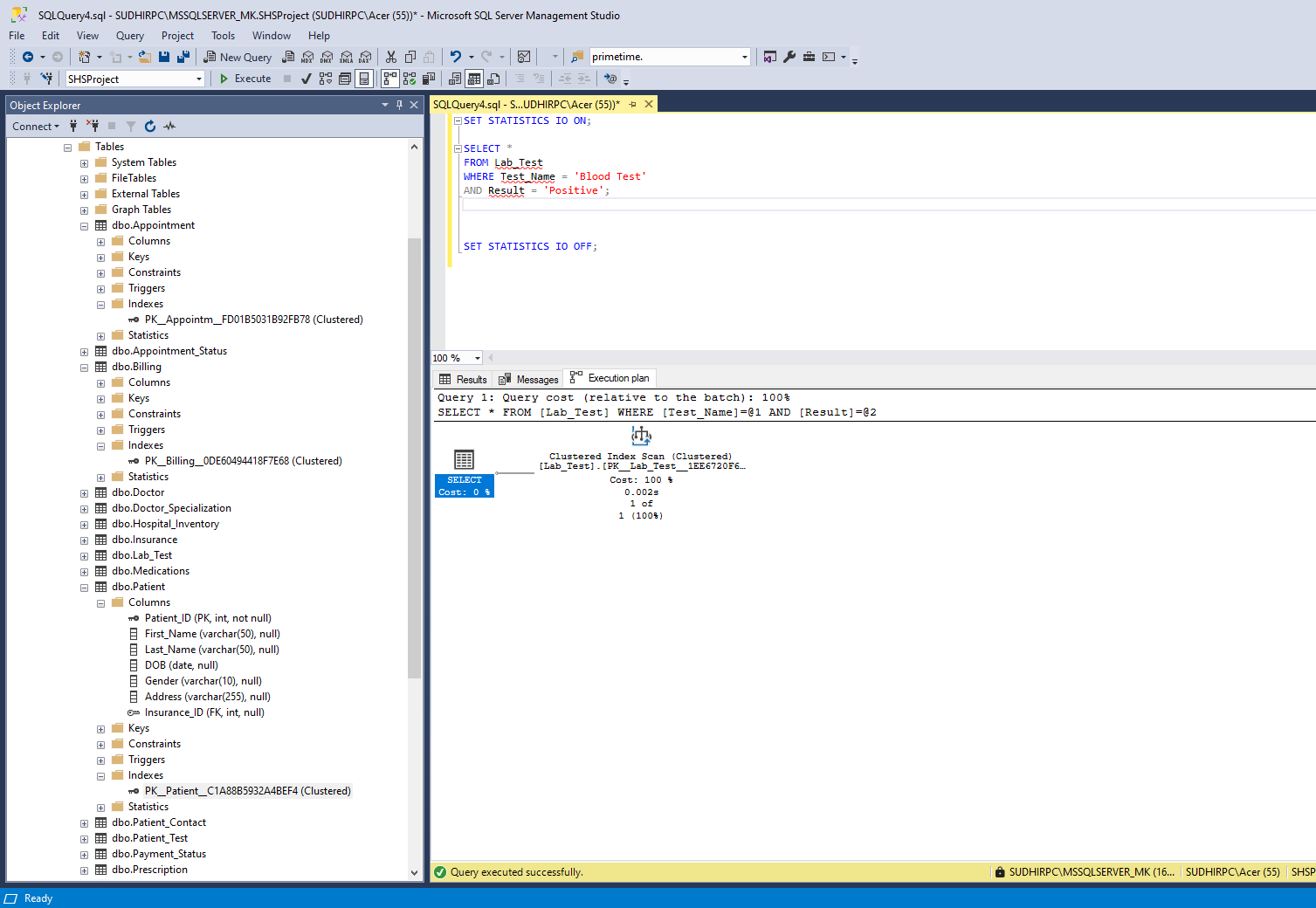
**Query Message:**

|  |
| --- |
| (1 row affected)  Table 'Billing'. Scan count 1, logical reads 2, physical reads 0, page server reads 0, read-ahead reads 0, page server read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob page server reads 0, lob read-ahead reads 0, lob page server read-ahead reads 0.  (1 row affected)  Completion time: 2025-03-06T14:46:59.5390636-06:00 |

**Interpretation:**

* Logical Reads = 2
  + The query scanned only 2 pages in the Billing table, indicating that the database engine efficiently retrieved the necessary records.
  + This confirms that the index on Billing (Payment\_Status\_ID, Date\_Issued) has successfully optimized data access, reducing unnecessary reads.
* Clustered Index Scan in Execution Plan
  + The execution plan indicates a Clustered Index Scan, which means SQL Server scanned the entire index rather than using an index seek.
  + However, since the number of logical reads is minimal (2 pages), this suggests that the scan was efficient due to the indexing strategy.
  + In cases where the dataset is relatively small, SQL Server may still choose a scan instead of an index seek if it determines that scanning a small number of pages is more efficient.
* Query Cost: 100%
  + The query cost is relative to the batch execution and does not indicate a performance issue.
  + The low number of logical reads suggests that the indexing strategy successfully reduced the number of scanned records.

**Index Performance for Lab\_Test Table Using SQL Server Execution Plan and Statistics IO**



**Query message:**

|  |
| --- |
| (1 row affected)  Table 'Lab\_Test'. Scan count 1, logical reads 2, physical reads 1, page server reads 0, read-ahead reads 0, page server read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob page server reads 0, lob read-ahead reads 0, lob page server read-ahead reads 0.  (1 row affected)  Completion time: 2025-03-06T14:52:00.5866568-06:00 |

**Interpretation:**

* Logical Reads = 2
  + The query only required 2 logical reads, indicating that the database engine retrieved the necessary records efficiently.
  + This demonstrates that the index on Lab\_Test (Test\_Name, Result) is being utilized effectively to minimize unnecessary data scans.
* Physical Reads = 1
  + The system accessed one page from disk rather than scanning the entire table.
  + A low number of physical reads indicates that the index has reduced disk I/O, leading to faster query execution.
* Clustered Index Scan in Execution Plan
  + The execution plan shows a Clustered Index Scan, meaning that SQL Server scanned the clustered index rather than performing an index seek.
  + Since logical reads are very low (only 2 pages), this suggests that SQL Server optimized the scan efficiently due to the small dataset.
  + If the dataset were larger, a non-clustered index might be preferable to allow for an index seek.
* Query Cost: 100%
  + This value represents the relative cost within the execution batch and does not necessarily indicate inefficiency.
  + Given the low logical reads and minimal physical reads, the index is effectively improving performance.

# J. Views

*Description: Add two views to your database to provide easy access to combinations of data from multiple tables.*

*Rubric: Your work will be graded as follows:*

* *3 points for including the SQL for generating the two views in your Word document*
* *3 points for including screenshots for the data contained in each view in your Word document*
* *3 points for explaining why each view is a valuable addition to your database*

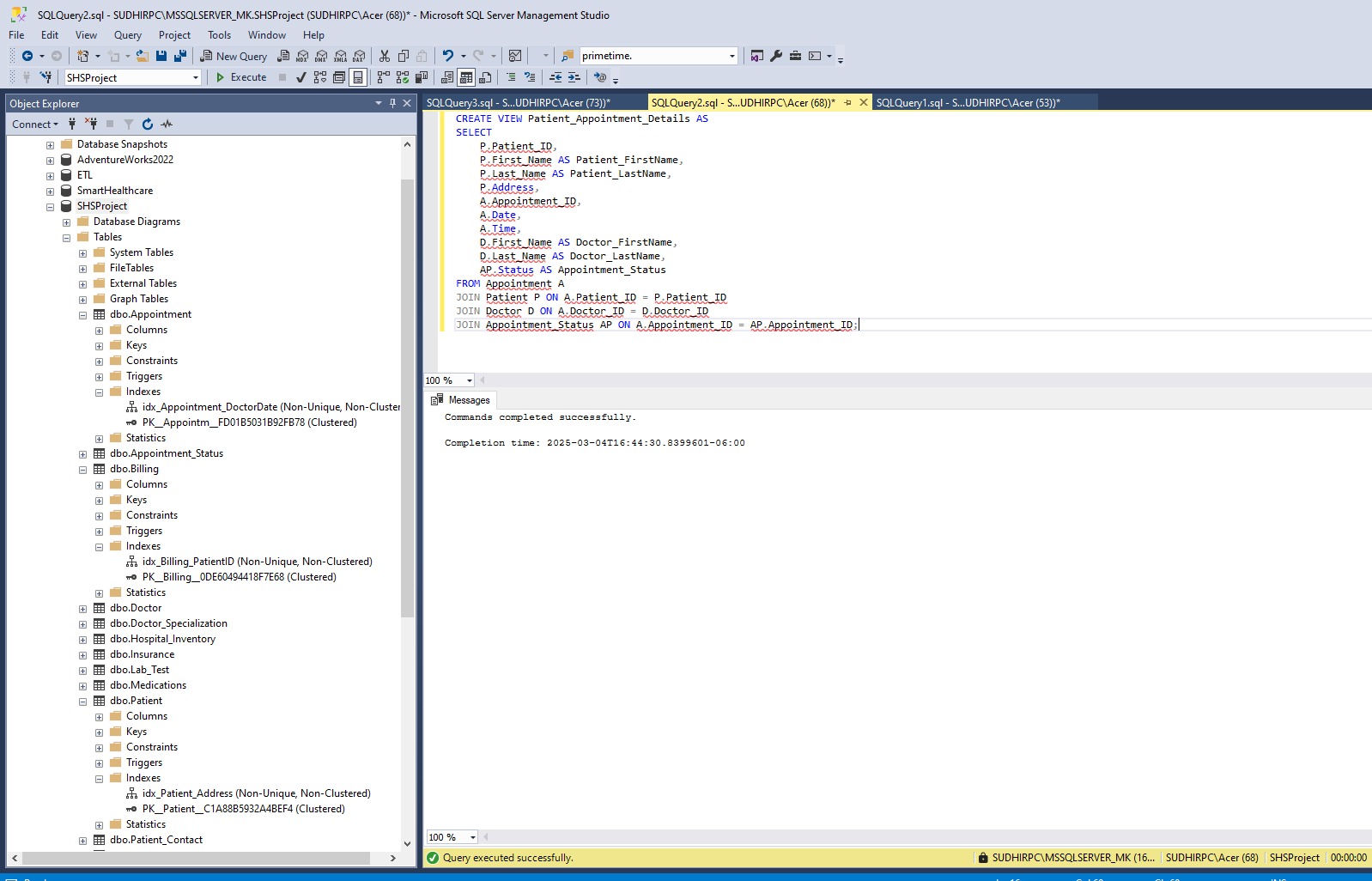
*Total points possible: 9*

ENTER YOUR WORK WITH VIEWS HERE

View 1: Patient\_Appointment\_Details

Provides a single view of patient details along with their appointment information and assigned doctor. Helps quickly retrieve patient appointments without needing complex joins.

|  |
| --- |
| CREATE VIEW Patient\_Appointment\_Details AS  SELECT  P.Patient\_ID,  P.First\_Name AS Patient\_FirstName,  P.Last\_Name AS Patient\_LastName,  P.Address,  A.Appointment\_ID,  A.Date,  A.Time,  D.First\_Name AS Doctor\_FirstName,  D.Last\_Name AS Doctor\_LastName,  AP.Status AS Appointment\_Status  FROM Appointment A  JOIN Patient P ON A.Patient\_ID = P.Patient\_ID  JOIN Doctor D ON A.Doctor\_ID = D.Doctor\_ID  JOIN Appointment\_Status AP ON A.Appointment\_ID = AP.Appointment\_ID; |



View 2: Billing\_Payment\_Status

Displays billing details along with patient name and payment status. Allows quick financial tracking by combining Billing, Patient, and Payment\_Status data.

|  |
| --- |
| CREATE VIEW Billing\_Payment\_Status AS  SELECT  B.Invoice\_ID,  P.First\_Name AS Patient\_FirstName,  P.Last\_Name AS Patient\_LastName,  B.Amount,  B.Insurance\_Covered,  B.Date\_Issued,  B.Date\_Paid,  PS.Status AS Payment\_Status  FROM Billing B  JOIN Patient P ON B.Patient\_ID = P.Patient\_ID  JOIN Payment\_Status PS ON B.Payment\_Status\_ID = PS.Payment\_Status\_ID; |

A screenshot of a computer

AI-generated content may be incorrect.

Select views:

A screenshot of a computer

AI-generated content may be incorrect.

# K. Stored Programs (Stored Procedures, Stored Functions, Triggers)

*Description: Add a stored procedure, stored function or trigger to a table and demonstrate using it.*

*Rubric: Your work will be graded as follows:*

* *3 points for including the SQL for the stored program (procedure, function, or trigger in your Word document*
* *3 points for clearly explaining the purpose of the stored program*
* *3 points for a screenshot and explanation that shows the stored program in action.*

*Total points possible: 9*

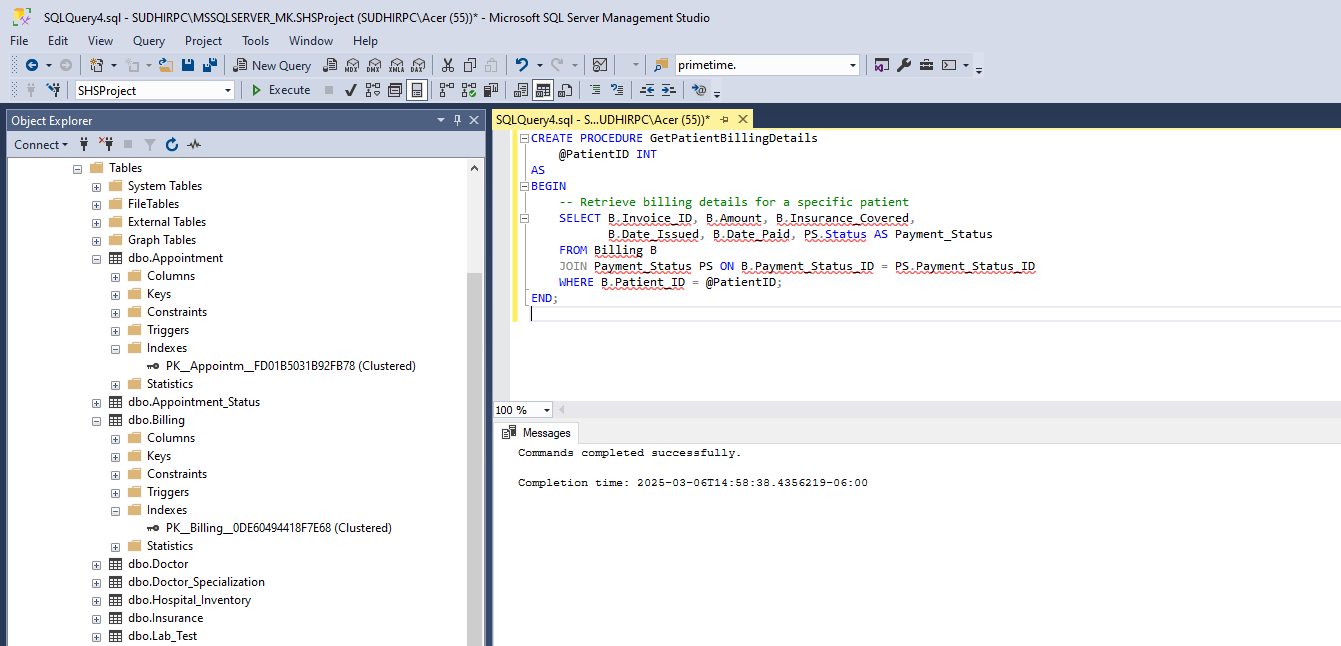
ENTER YOUR WORK WITH STORED PROGRAMS HERE

Among Stored Procedures, Functions, and Triggers, a Stored Procedure is the best choice for this demonstration because:

* It allows parameterized execution.
* It improves query efficiency by executing precompiled SQL statements.
* It is easier to test and demonstrate compared to triggers.

We created a stored procedure to retrieve billing details for a specific patient. This allows users to quickly look up pending or completed bills by providing a Patient\_ID.

|  |
| --- |
| CREATE PROCEDURE GetPatientBillingDetails  @PatientID INT  AS  BEGIN  -- Retrieve billing details for a specific patient  SELECT B.Invoice\_ID, B.Amount, B.Insurance\_Covered,  B.Date\_Issued, B.Date\_Paid, PS.Status AS Payment\_Status  FROM Billing B  JOIN Payment\_Status PS ON B.Payment\_Status\_ID = PS.Payment\_Status\_ID  WHERE B.Patient\_ID = @PatientID;  END; |

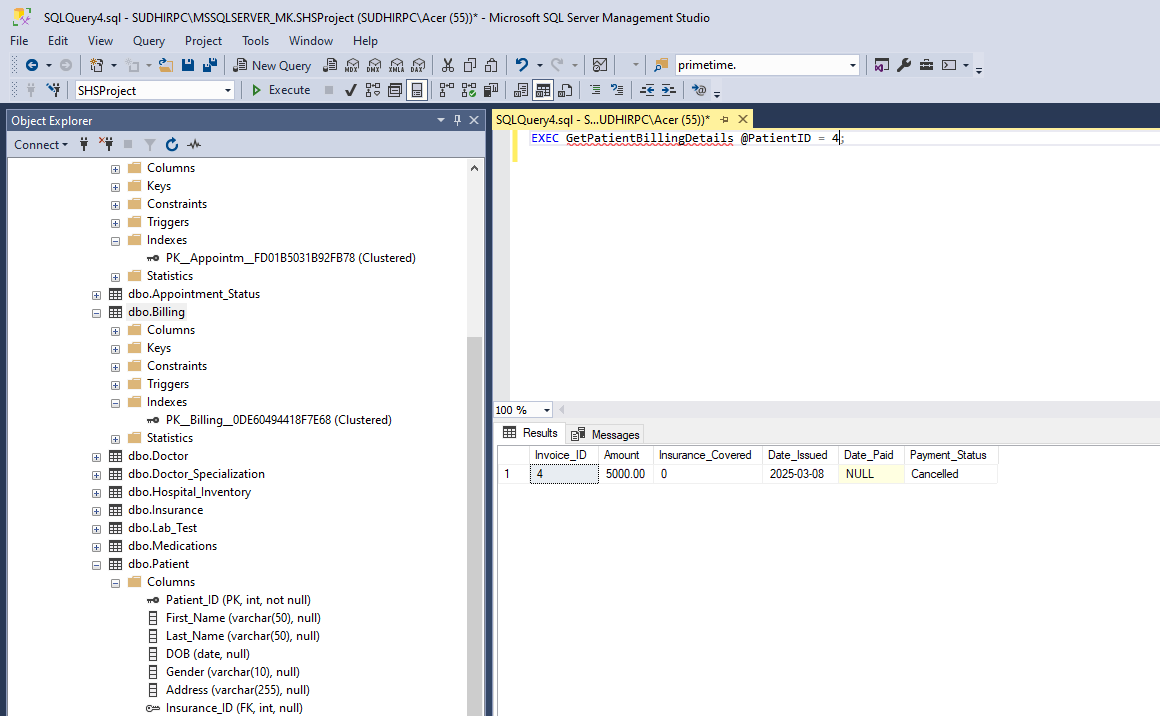


**Purpose of this stored procedure:**

* Allows staff to quickly retrieve billing records for a specific patient.
* Reduces the need to write repetitive SQL queries every time billing details are needed.
* Ensures that only relevant billing information is displayed for each patient.

**Test:**

|  |
| --- |
| EXEC GetPatientBillingDetails @PatientID = 4; |



The stored procedure was successfully tested by executing EXEC GetPatientBillingDetails @PatientID = 4;. The result set displayed billing records for the given patient, proving its functionality.

# L. Transactions

*Description: Demonstrate that you know how to define and use a transaction. Why are transactions important for ensuring ACID behavior?*

*Rubric: Your work will be graded as follows:*

* *5 points for clearly explaining the importance of transactions to ensuring ACID behavior*
* *3 points for including a screenshot and accompanying explanation of a MySQL transaction.*

*Total points possible: 8*

ENTER YOUR WORK WITH TRANSACTIONS HERE

What is a Transaction?

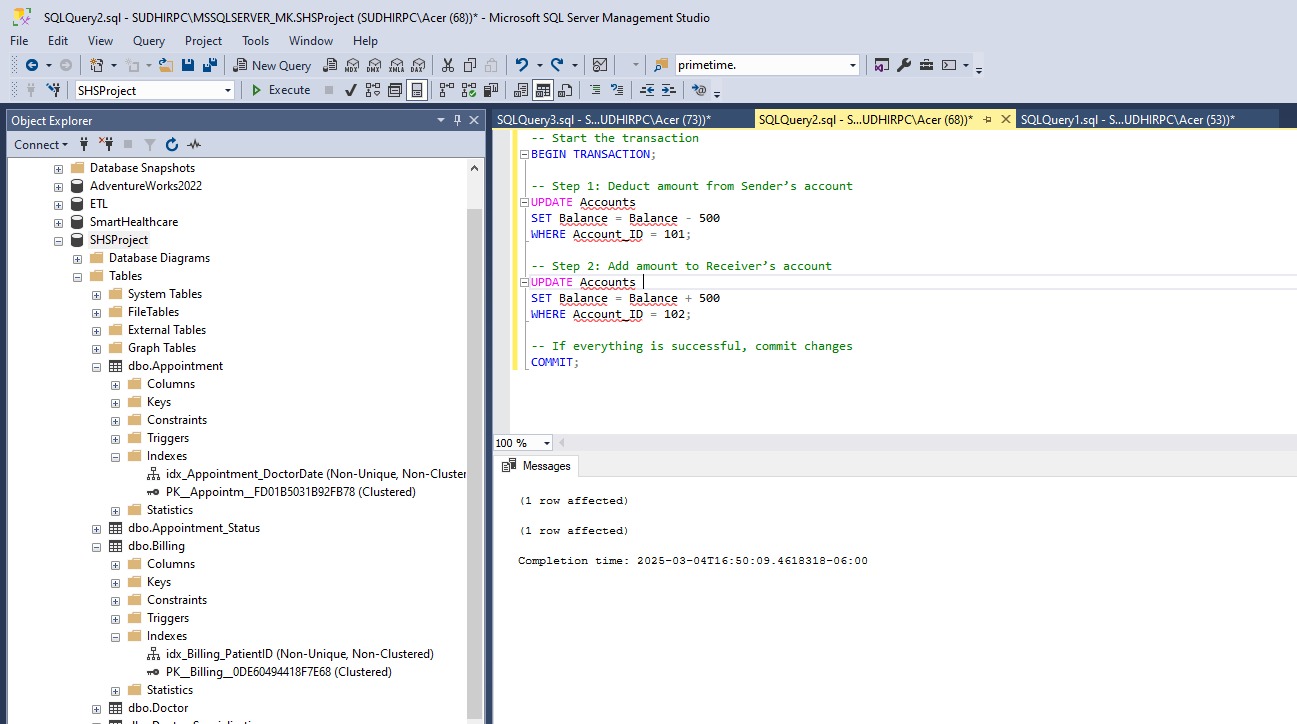
A transaction in SQL is a logical unit of work that consists of one or more database operations. It ensures that all operations within the transaction are either fully executed or fully undone, maintaining data integrity and consistency. Transactions are essential in scenarios where multiple related operations must be treated as a single unit to avoid partial updates and inconsistencies.

ACID Properties Through Transactions:

* Atomicity ("All or Nothing")
  + Ensures that a transaction is executed in its entirety or not executed at all.
  + If any part of a transaction fails, the entire transaction is rolled back, leaving the database unchanged.
  + Example:
    - If transferring funds between two bank accounts, both the deduction from the sender and the deposit to the receiver must succeed. If one fails, the transaction is rolled back, preventing an inconsistent state.
* Consistency ("Valid State Transitions")
  + Ensures that the database moves from one valid state to another while maintaining data integrity.
  + If a transaction violates database rules (such as foreign key constraints), it is aborted and rolled back.
  + Example:
    - A bank transfer must ensure that the total sum of all account balances remains the same before and after the transaction.
* Isolation ("Transactions Do Not Interfere")
  + Ensures that transactions execute independently without affecting each other.
  + Prevents concurrency issues such as dirty reads, non-repeatable reads, and phantom reads.
  + Example:
    - If two users try to update the same bank account balance simultaneously, isolation ensures that one transaction is completed before the other starts.
* Durability ("Permanent Changes")
  + Once a transaction is committed, the changes are permanently stored in the database, even in case of system failure.
  + SQL Server ensures durability using transaction logs that store changes until they are written to disk.

Smooth transaction:

|  |
| --- |
| -- Create Accounts table if it does not exist  CREATE TABLE Accounts (  Account\_ID INT PRIMARY KEY,  Account\_Holder VARCHAR(100),  Balance DECIMAL(10,2)  );  -- Insert Sample Data  INSERT INTO Accounts (Account\_ID, Account\_Holder, Balance)  VALUES  (101, 'Ravi Kumar', 10000.00),  (102, 'Asha Verma', 8000.00); |



Roll Back Transaction:

|  |
| --- |
| BEGIN TRANSACTION;  UPDATE Accounts  SET Balance = Balance - 500  WHERE Account\_ID = 101;  -- Simulating an error by trying to update a non-existing account  UPDATE Accounts  SET Balance = Balance + 500  WHERE Account\_ID = 999;  -- If error occurs, rollback  ROLLBACK;  -- Check balances (No changes should be applied)  SELECT \* FROM Accounts; |

A computer screen with a computer screen

AI-generated content may be incorrect.

# M. Database Security

*Description: Identify the different kinds of users who will use your database. Write GRANT statements to define the privileges for these different kinds of users.*

*Rubric: Your work will be graded as follows:*

* *4 points for clearly identifying and describing the various kinds of users who will use the databases and identifying and justifying what privileges each should have.*
* *4 points for writing GRANT statements that assign privileges to these different kinds of users.*
* *4 points for demonstrating with screenshots that your GRANT statements do distinguish among different kinds of users in regard to what they can do with the database.*

*Total points possible: 12*

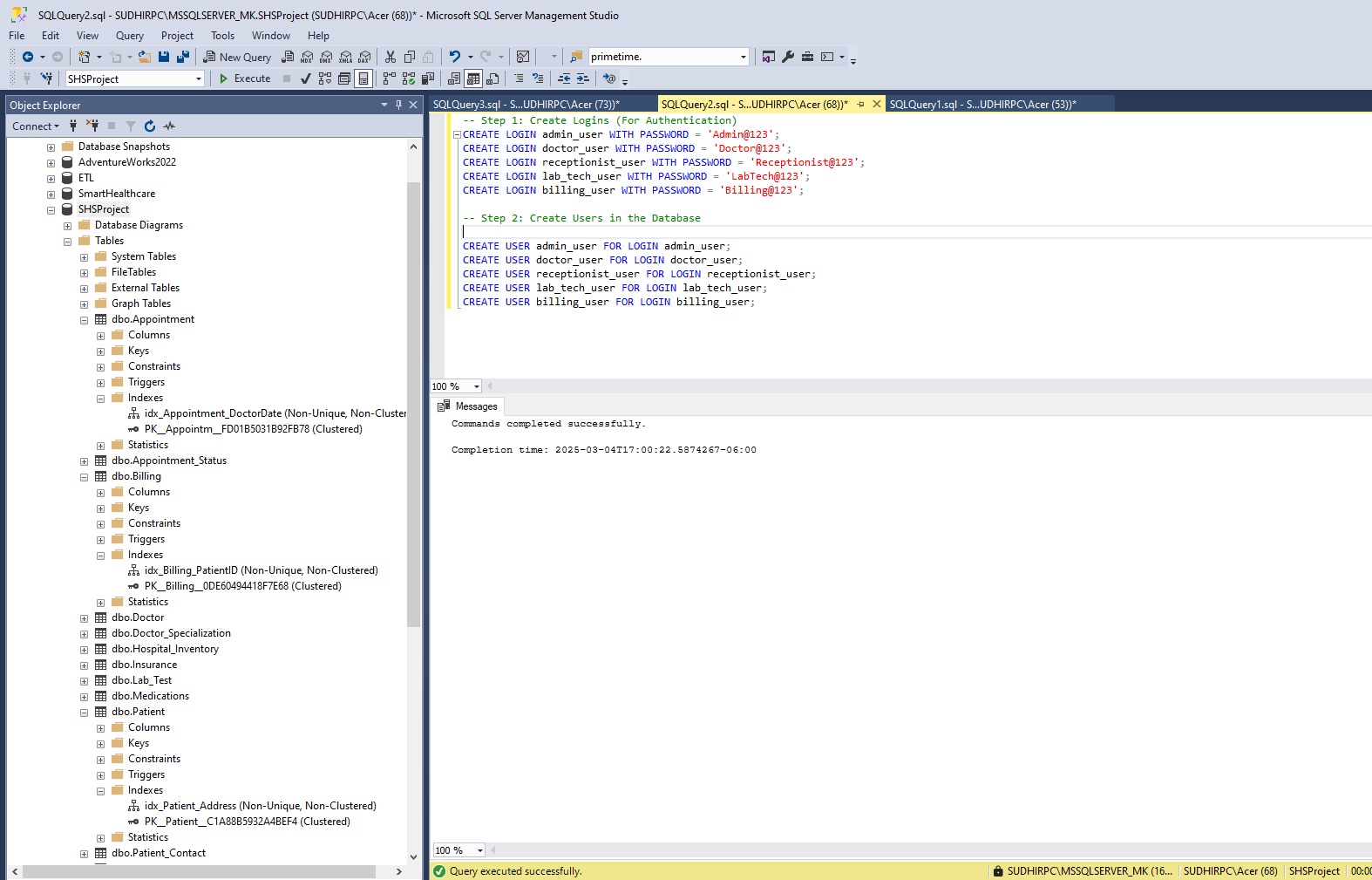
ENTER YOUR WORK WITH DATABASE SECURITY HERE

Identifying Users:

|  |  |  |
| --- | --- | --- |
| **User Role** | **Description** | **Privileges Required** |
| **Administrator** | Manages the entire database, including creating, modifying, and deleting tables and user accounts. | ALL PRIVILEGES (Full control) |
| **Doctor** | Needs to view patient details, update prescriptions, and check appointments. | SELECT, INSERT, UPDATE on Patient, Appointment, Prescription |
| **Receptionist** | Manages patient appointments and billing information, but cannot modify medical records. | SELECT, INSERT, UPDATE on Appointment, Billing |
| **Lab Technician** | Needs to add and update lab test results, but cannot access sensitive patient details. | SELECT, INSERT, UPDATE on Lab\_Test |
| **Billing Staff** | Handles billing and payments, but should not access medical records. | SELECT, UPDATE on Billing |

Creating users:

|  |
| --- |
| -- Step 1: Create Logins (For Authentication)  CREATE LOGIN admin\_user WITH PASSWORD = 'Admin@123';  CREATE LOGIN doctor\_user WITH PASSWORD = 'Doctor@123';  CREATE LOGIN receptionist\_user WITH PASSWORD = 'Receptionist@123';  CREATE LOGIN lab\_tech\_user WITH PASSWORD = 'LabTech@123';  CREATE LOGIN billing\_user WITH PASSWORD = 'Billing@123';  -- Step 2: Create Users in the Database  CREATE USER admin\_user FOR LOGIN admin\_user;  CREATE USER doctor\_user FOR LOGIN doctor\_user;  CREATE USER receptionist\_user FOR LOGIN receptionist\_user;  CREATE USER lab\_tech\_user FOR LOGIN lab\_tech\_user;  CREATE USER billing\_user FOR LOGIN billing\_user; |



Grant Permissions:

|  |
| --- |
| -- 1. Grant Full Access to Administrators  GRANT CONTROL ON DATABASE::SHSProject TO admin\_user;  -- 2. Grant Limited Access to Doctors  GRANT SELECT, INSERT, UPDATE ON Patient TO doctor\_user;  GRANT SELECT, INSERT, UPDATE ON Appointment TO doctor\_user;  GRANT SELECT, INSERT, UPDATE ON Prescription TO doctor\_user;  -- 3. Grant Receptionists Access to Manage Appointments and Billing  GRANT SELECT, INSERT, UPDATE ON Appointment TO receptionist\_user;  GRANT SELECT, INSERT, UPDATE ON Billing TO receptionist\_user;  -- 4. Grant Lab Technicians Access to Lab Tests Only  GRANT SELECT, INSERT, UPDATE ON Lab\_Test TO lab\_tech\_user;  -- 5. Grant Billing Staff Access to Billing and Payments Only  GRANT SELECT, UPDATE ON Billing TO billing\_user; |

A computer screen shot of a computer

AI-generated content may be incorrect.

Verifying Permissions:

|  |
| --- |
| SELECT  princ.name AS UserName,  perm.permission\_name AS Permission,  perm.state\_desc AS PermissionState,  obj.name AS ObjectName  FROM sys.database\_permissions perm  JOIN sys.database\_principals princ ON perm.grantee\_principal\_id = princ.principal\_id  LEFT JOIN sys.objects obj ON perm.major\_id = obj.object\_id  WHERE princ.name IN ('admin\_user', 'doctor\_user', 'receptionist\_user', 'lab\_tech\_user', 'billing\_user'); |

A screenshot of a computer

AI-generated content may be incorrect.

# N. Locking and Concurrent Access

*Description: Explain the purpose of locking tables and show how to do that to prevent inconsistencies that may arise in your data when concurrent transactions take place.*

*Rubric: Your work will be graded as follows:*

* *3 points for clearly explaining an example that shows why you should lock tables to prevent inconsistencies.*
* *3 points for providing a screenshot and accompanying explanation of locking tables.*

*Total points possible: 5*

ENTER YOUR WORK WITH LOCKING AND CONCURRENT ACCESS HERE

**Purpose of Table Locking**

In a healthcare management system, multiple users, including doctors, receptionists, billing staff, and patients, may simultaneously access and modify the database. Without proper table locking mechanisms, data inconsistencies may arise due to concurrent transactions. Table locking ensures that critical updates to appointments, billing, and patient records are processed in a synchronized manner, preventing data corruption and conflicts.

**Example Scenario: Appointment Scheduling Conflict**

Problem: Double Booking of a Doctor’s Appointment Slot

In a hospital, the appointment scheduling system allows patients to book available time slots with doctors. Consider the following scenario:

* Patient A attempts to book an appointment with Doctor X at 10:00 AM.
* Patient B, at the same time, tries to book the same time slot with the same doctor.
* The system processes both transactions simultaneously due to high user activity.

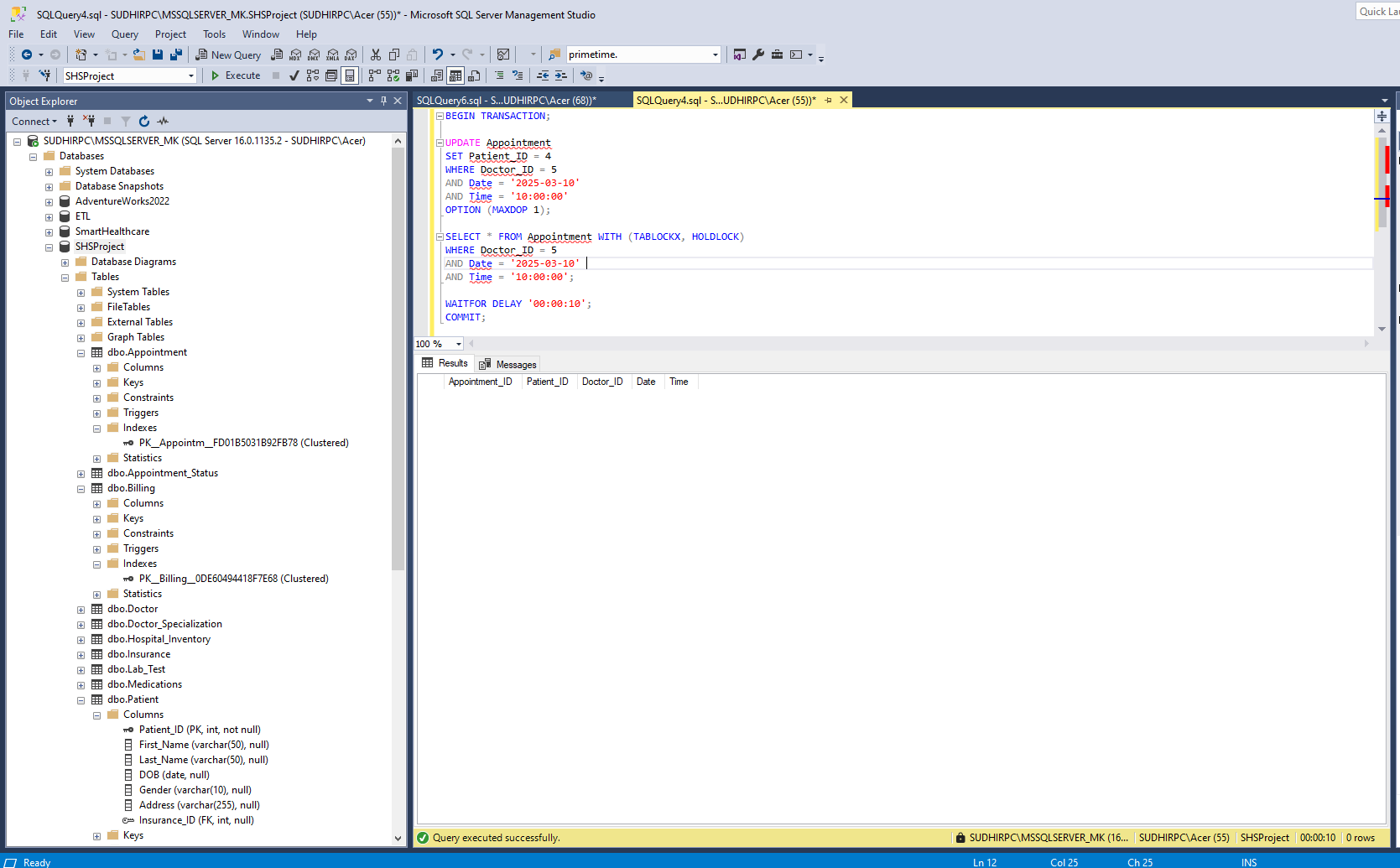
**Possible Issue (Race Condition):**

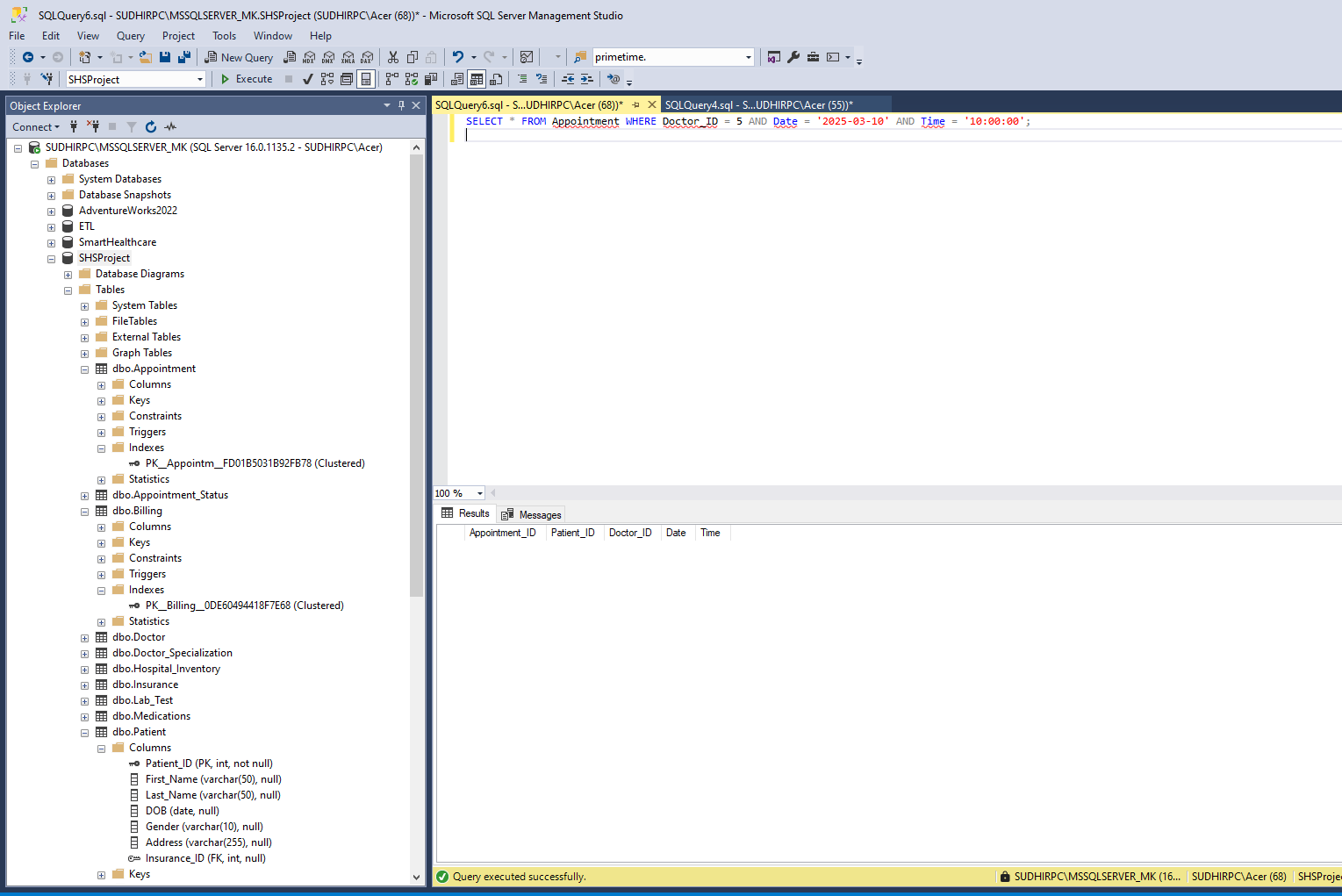
* If both transactions read the available appointment slots before either transaction updates the database, both appointments may be booked for the same doctor at the same time.
* This results in double booking, which leads to confusion and scheduling errors.

**Solution: Applying Table Locking to Prevent Double Booking**

To prevent two patients from booking the same doctor at the same time, an exclusive lock (TABLOCKX) is applied to the Appointment table during the booking process. This ensures that only one transaction can access the table at a time, preventing simultaneous updates.

|  |
| --- |
| BEGIN TRANSACTION;  UPDATE Appointment  SET Patient\_ID = 4  WHERE Doctor\_ID = 5  AND Date = '2025-03-10'  AND Time = '10:00:00'  OPTION (MAXDOP 1);  SELECT \* FROM Appointment WITH (TABLOCKX, HOLDLOCK)  WHERE Doctor\_ID = 5  AND Date = '2025-03-10'  AND Time = '10:00:00';  WAITFOR DELAY '00:00:10';  COMMIT;  ---------------------------------------------------------------------------------------------------------------------------  SELECT \* FROM Appointment WHERE Doctor\_ID = 5 AND Date = '2025-03-10' AND Time = '10:00:00'; |





* The UPDATE statement with TABLOCKX locks the specific doctor’s appointment slot.
* Any other transaction attempting to modify or read this appointment slot will be blocked until the first transaction completes.
* The second query (run in another session) remains in a waiting state and executes only after the lock is released.

Table locking is essential in healthcare management systems to prevent data inconsistencies, such as double booking of appointments. By applying exclusive table locks (TABLOCKX), the system ensures that only one patient can book a specific time slot at a time, eliminating race conditions and scheduling conflicts.

This approach ensures accurate patient scheduling and prevents conflicts in appointment management, leading to a more reliable healthcare system.

# O. Backing Up Your Database

*Description: How you will back up your database. What commands will you issue? How frequently will the commands run? How can they be automated? Where will the backups be stored?*

*Rubric: Your work will be graded as follows:*

* *6 points for clearly explaining and justifying your database backup strategy, including the frequency with which you will back up the database, how you will automate backups, where you will store them, and how you will secure them. You will earn three points for addressing each factor (frequency, location, automation, and security)*
* *2 points for providing a screenshot of the command you would issue to back up the database and for including a portion of the resulting file.*

*Total points possible: 8*

ENTER YOUR WORK ON DATABASE BACKUPS HERE

**Justification for the Backup Strategy**

A well-structured database backup plan ensures data integrity, protection against failures, and compliance with healthcare regulations such as HIPAA (Health Insurance Portability and Accountability Act). The healthcare system stores critical patient information, appointments, billing details, and lab test results, making regular backups essential.

**Backup Frequency**

Backup frequency depends on the criticality of the data and system usage. The strategy includes:

* **Full Backup:** Performed daily at midnight to capture the entire database.
* **Differential Backup:** Taken every 6 hours to store only the changes made since the last full backup.
* **Transaction Log Backup:** Runs every 15 minutes to minimize data loss in case of failure.

This approach ensures that in the event of failure, data loss is minimized to at most 15 minutes.

**Backup Storage Location**

To ensure data redundancy and security, backups will be stored in multiple locations:

1. Primary Local Storage: The database server’s dedicated backup disk (D:\SQLBackups\).
2. Remote Storage: A secure network-attached storage (NAS) system (\\BackupServer\HealthcareDB\).
3. Cloud Backup: A secure AWS S3 or Azure Blob Storage configured for long-term retention.

Using multiple backup locations prevents data loss due to hardware failures, ransomware attacks, or accidental deletions.

**Backup Process:**

Best way is to assign jobs to SQL Server Agent to make the backup automatically, but here we are not implementing this. But we are mentioning the commands that can be used to do it automatically by using them in job scheduler.

Full Backup - Runs Daily at Midnight

|  |
| --- |
| BACKUP DATABASE HealthcareDB  TO DISK = 'D:\SQLBackups\HealthcareDB\_Full.bak'  WITH FORMAT, INIT, NAME = 'Full Backup of HealthcareDB', STATS = 10; |

Differential Backup - Runs Every 6 Hours

|  |
| --- |
| BACKUP DATABASE HealthcareDB  TO DISK = 'D:\SQLBackups\HealthcareDB\_Diff.bak'  WITH DIFFERENTIAL, INIT, NAME = 'Differential Backup of HealthcareDB', STATS = 10; |

Transaction Log Backup - Runs Every 15 Minutes

|  |
| --- |
| BACKUP LOG HealthcareDB  TO DISK = 'D:\SQLBackups\HealthcareDB\_TransLog.trn'  WITH INIT, NAME = 'Transaction Log Backup', STATS = 10; |

**How to Secure the Backups:**

We can secure the backup by using encryption or through Access control.

**Encryption:**

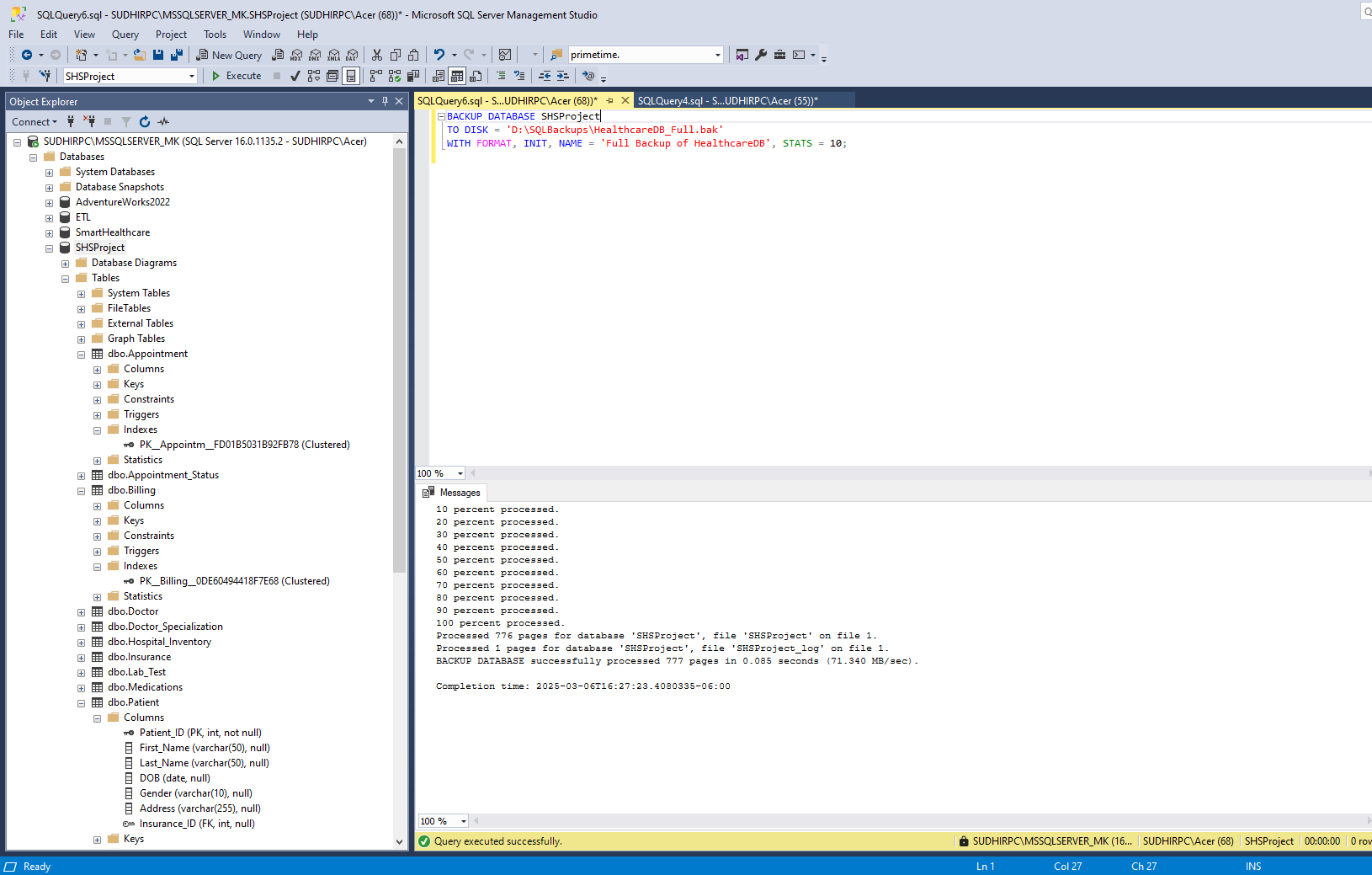
Backups will be encrypted using AES-256 to prevent unauthorized access.

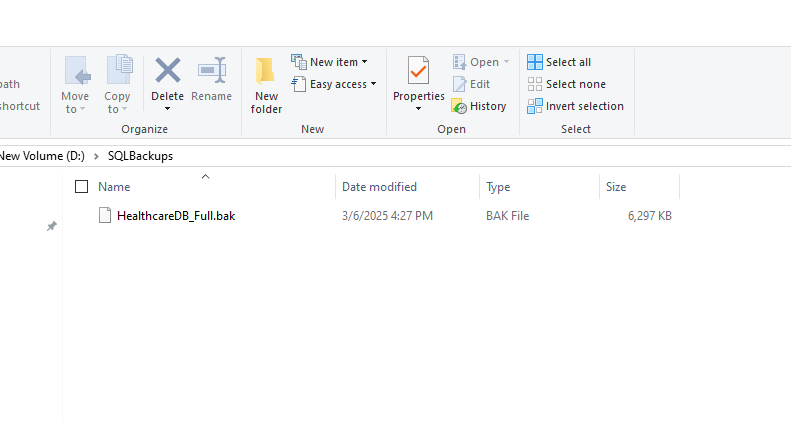
|  |
| --- |
| BACKUP DATABASE HealthcareDB  TO DISK = 'D:\SQLBackups\HealthcareDB\_Encrypted.bak'  WITH ENCRYPTION (ALGORITHM = AES\_256, SERVER CERTIFICATE = BackupCert); |

**Access Control:**

* Only database administrators will have permission to access backups.
* Backup files will have restricted file system permissions.

Below screenshots showing the full backup command run on SSMS and it created the backup in “D” Drive in SQLBackup folder.





# P. Programming

*Description: Write a Python, Java, or PHP program that generates a report that contains a subset of the data from your database. Include the code for your Python program in your Word document, and also post the program to your GitHub repository.*

*Rubric: Your work will be graded as follows:*

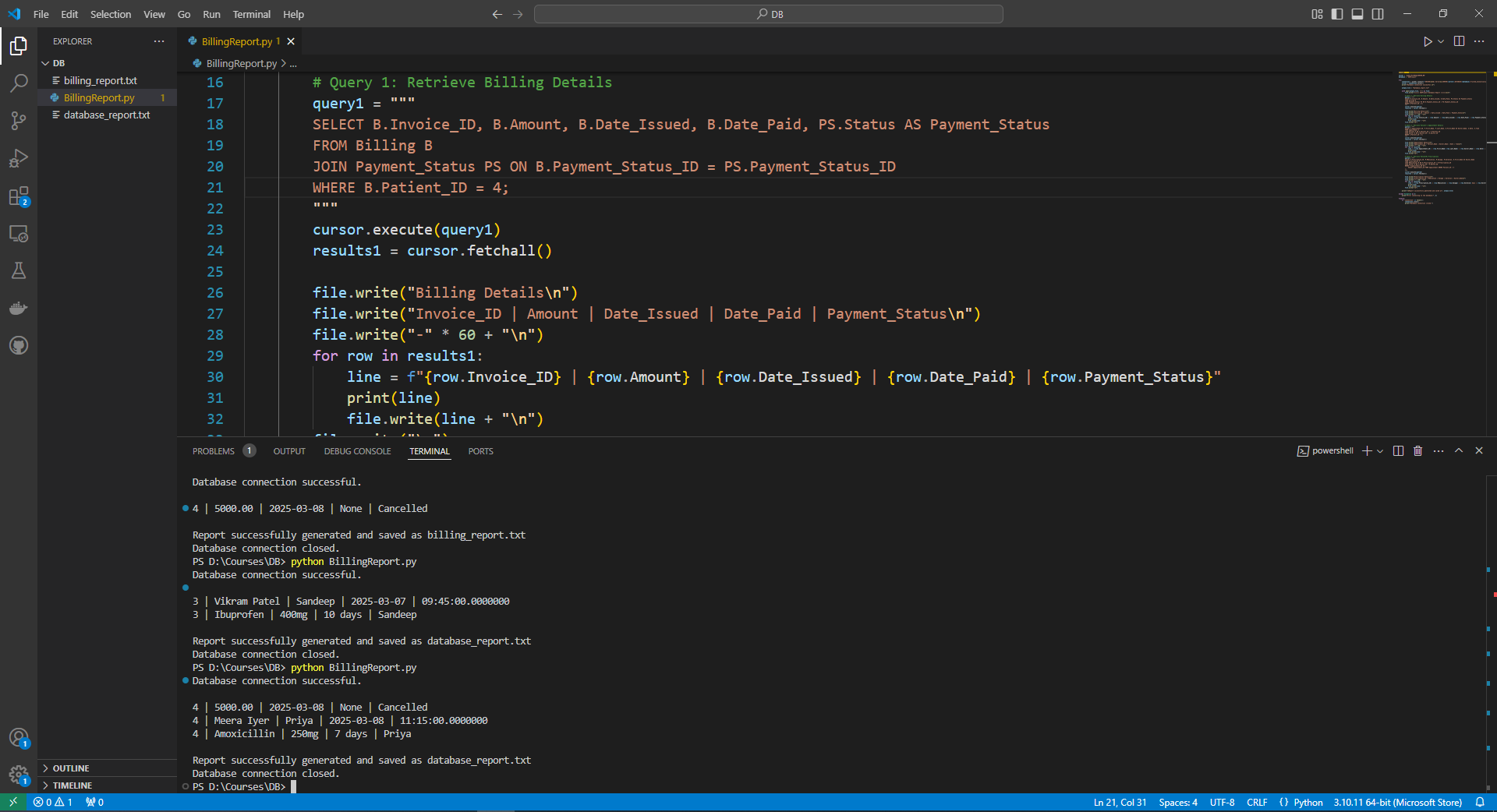
* *10 points for writing a Python script (and including its code in the Word doc) that will pull data from a database and store it to a text file and present it to the screen. Your code must have comments in it that explain how it works. You will be awarded 3 points for successfully connecting to the database, 3 points for successfully querying it, and 4 points for presenting the data to the screen and to a file. Internal comments count for 2 points.*
* *2 points for posting the code to GitHub*
* *6 points for showing a screenshot of your running the script and showing the results it produces on the screen.*

*Total points possible: 18*

ENTER YOUR PYTHON, PHP, or JAVA DATABASE PROGRAMMING WORK HERE

Code:

|  |
| --- |
| import pyodbc  server = 'SudhirPC\MSSQLSERVER\_MK'  database = 'SHSProject'  try:      connection = pyodbc.connect(f'DRIVER={{SQL Server}};SERVER={server};DATABASE={database};Trusted\_Connection=yes')      cursor = connection.cursor()      print("Database connection successful.\n")      output\_file = "database\_report.txt"      with open(output\_file, 'w') as file:          file.write("====== SHSProject Database Report ======\n\n")          # Query 1: Retrieve Billing Details          query1 = """          SELECT B.Invoice\_ID, B.Amount, B.Date\_Issued, B.Date\_Paid, PS.Status AS Payment\_Status          FROM Billing B          JOIN Payment\_Status PS ON B.Payment\_Status\_ID = PS.Payment\_Status\_ID          WHERE B.Patient\_ID = 4;          """          cursor.execute(query1)          results1 = cursor.fetchall()          file.write("Billing Details\n")          file.write("Invoice\_ID | Amount | Date\_Issued | Date\_Paid | Payment\_Status\n")          file.write("-" \* 60 + "\n")          for row in results1:              line = f"{row.Invoice\_ID} | {row.Amount} | {row.Date\_Issued} | {row.Date\_Paid} | {row.Payment\_Status}"              print(line)              file.write(line + "\n")          file.write("\n")          # Query 2: Retrieve Patient's Appointment Details          query2 = """          SELECT A.Appointment\_ID, P.First\_Name, P.Last\_Name, D.First\_Name AS Doctor\_Name, A.Date, A.Time          FROM Appointment A          JOIN Patient P ON A.Patient\_ID = P.Patient\_ID          JOIN Doctor D ON A.Doctor\_ID = D.Doctor\_ID          WHERE A.Patient\_ID = 4;          """          cursor.execute(query2)          results2 = cursor.fetchall()          file.write("Appointment Details\n")          file.write("Appointment\_ID | Patient\_Name | Doctor\_Name | Date | Time\n")          file.write("-" \* 60 + "\n")          for row in results2:              line = f"{row.Appointment\_ID} | {row.First\_Name} {row.Last\_Name} | {row.Doctor\_Name} | {row.Date} | {row.Time}"              print(line)              file.write(line + "\n")          file.write("\n")          # Query 3: Retrieve Patient’s Prescriptions          query3 = """          SELECT P.Prescription\_ID, M.Medication, M.Dosage, M.Duration, D.First\_Name AS Doctor\_Name          FROM Prescription P          JOIN Medications M ON P.Prescription\_ID = M.Prescription\_ID          JOIN Doctor D ON P.Doctor\_ID = D.Doctor\_ID          WHERE P.Appointment\_ID IN (              SELECT Appointment\_ID FROM Appointment WHERE Patient\_ID = 4          );          """          cursor.execute(query3)          results3 = cursor.fetchall()          file.write("Prescription Details\n")          file.write("Prescription\_ID | Medication | Dosage | Duration | Doctor\_Name\n")          file.write("-" \* 60 + "\n")          for row in results3:              line = f"{row.Prescription\_ID} | {row.Medication} | {row.Dosage} | {row.Duration} days | {row.Doctor\_Name}"              print(line)              file.write(line + "\n")          file.write("\n")      print("\nReport successfully generated and saved as", output\_file)  except Exception as e:      print("Error connecting to the database:", e)  finally:      if 'connection' in locals():          connection.close()          print("Database connection closed.") |





# Q. Suggested Future Work

*Description: Describe the limitations of your current database and explain how you or someone else could improve the design to address these shortcomings. Also describe how you might take advantage of leverage cloud services to increase the performance and availability of your database. Finally, explain the advantages and disadvantages of storing your data in a NoSQL format instead.*

*Rubric: Your work will be graded as follows:*

* *3 points for clearly describing the limitations of your databases*
* *3 points for explaining how you would address these shortcomings*
* *3 points for explaining how you might migrate the database to the cloud and describing what advantages you might gain from doing that.*
* *3 points for explaining the advantages and disadvantages of storing your data in a document-based NoSQL format instead.*

*Total points possible: 12*

ENTER YOUR SUGGESTED FUTURE WORK IDEAS HERE

**Limitations and Future Improvements of the Healthcare Management System Database**

Designed to handle structured patient records, appointments, invoicing, and medicines, the SHSProject database is used in healthcare management. Although the system effectively manages medical data, its scalability, performance, and security could be limited as usage rises from numerous factors. Ensuring long-term efficiency, availability, and regulatory compliance with regard for healthcare depends on addressing these constraints.

**Limitations of the Current Database Design**

Scalability of the present database is one of its main drawbacks. For structured healthcare data, the relational database model is appropriate; but, the system could suffer performance degradation as transaction count rises. Particularly for complicated searches requiring many table joins, the growing volume of patient records, appointment reservations, and financial transactions may cause delayed query response times. Retrieving patient history, billing data, or medical prescriptions may become ineffective over time without appropriate indexing strategy and query optimization methods.

The low fault tolerance of the system raises still another important issue. The database lacks high availability features like automated failover or replication since the on-site SQL Server instance hosting it is single. This implies that the whole system becomes unreachable if the server has downtime resulting from hardware failure, maintenance, or unplanned breakdowns. In a healthcare environment when fast access to patient records and scheduling is essential, such unavailability could cause operational inefficiencies and disturbance of medical services.

Moreover, handling intricate searches in the database could cause performance restrictions. Inquiries including obtaining appointment details, prescription histories, or patient invoicing may take more time to execute as the volume of patients and medical records increases. Lack of modern indexing, table partitioning, and query optimization strategies can cause delays that would influence administrative processes and patient experience.

The security and compliance issues of the technology represent still another major constraint. There is a concern in terms of illegal access to private patient data since the present database does not apply encryption for kept data. Under laws like HIPAA (Health Insurance Portability and Accountability Act), safeguarding patient data is a legal mandate in the medical field. Without strong security measures including audit logging, transparent data encryption (TDE), and role-based access control (RBAC), the system stays open to security concerns including data breaches and illegal changes.

**Addressing the Shortcomings of the Database**

Many changes can be done to improve the security, availability, and performance of the system so mitigating these constraints. Using advanced indexing techniques is among the best ways to raise scalability and performance. Search activities can be greatly accelerated by using composite indexes on often searched values, such as indexing the Doctor\_ID and Date in the Appointment database. Furthermore, partitioning big tables like billing helps to maximize data retrieval by keeping entries in several physical locations, hence lowering query running times.

The application of high availability and replication systems marks even another significant advance. Configuring SQL Server Always On Availability Groups guarantees that the database stays accessible even should a server fail. This system lets one server take over should the main server fail by means of real-time data replication over several servers. Using database mirroring and log shipping can also offer backup redundancy, therefore guaranteeing less data loss and downtime during system faults.

Improving query optimization will help to increase database effectiveness still more. By precomputing and storing results using materialized views for often requested data, complicated joins during execution are less necessary. By running precompiled query plans and reducing processing overhead, the use of stored procedures rather than raw SQL searches can increase performance.

Using data encryption methods is absolutely crucial if one is addressing security issues. By allowing transparent data encryption (TDE) in SQL Server, important patient records remain encrypted even should illegal users have access to the storage system. Data at rest is encrypted. Using role-based access control (RBAC) also guarantees that only authorized users such as administrative personnel and doctors have access to view or change particular data. Additionally used to track access and changes to patient records, regular audit logging and monitoring systems guarantees compliance with healthcare laws.

**Migrating the Database to Cloud Services**

Moving the healthcare database to a cloud-based Database-as- a- Service (DBaaS) solution can help to enhance its security, availability, and performance among other things. Significant benefits over conventional on-site databases abound from cloud platforms as Microsoft Azure SQL Database, Amazon RDS (Relational Database Service), and Google Cloud SQL.

Scalability is a main benefit of moving clouds. By means of auto-scaling features of cloud databases, the system can dynamically modify computer resources depending upon demand. For a hospital system in particular, where consumption surges could result from heavy patient traffic, seasonal demand, or emergency circumstances, this is especially helpful. Dynamic resource scaling guarantees that performance stays ideal free from human involvement.

Moreover improving high availability and catastrophe recovery are cloud platforms. Unlike on-site databases, which depend on a single server, cloud databases store duplicates of the data in several geographical locations by means of multi-region replication. Automated failover systems guarantee that, should a failure occur, the database stays operational without interruption of services. Additionally included into the program are cloud backups, which offer automated, safe storage free of human involvement.

Still another major benefit of cloud-based databases is security. For HIPAA, GDPR, and SOC2, cloud providers have built-in HIPAA, GDPR, and SOC2 compliance certifications together with access restrictions and encryption. Healthcare companies may guarantee that patient data stays safeguarded against cyber threats and illegal access by using cloud security elements. Furthermore improving data security are managed security solutions include access monitoring, intrusion detection, and firewall protection.

Moving to the cloud comes with difficulties even if its advantages are clear. The main issue is cost. Although cloud services cut on-site hardware and maintenance expenses, they also bring monthly membership prices depending on storage, compute use, and data transfer volume. Should improper optimization not be achieved, cloud costs may become a major financial load. Complexity of data movement presents still another obstacle. To guarantee that no data is lost during the transfer of a live production database to the cloud, careful preparation, database replication, and synchronizing are absolutely necessary.

**Evaluating NoSQL as an Alternative to Relational Databases**

Although relational databases are perfect for organized healthcare data, some uses can gain from a NoSQL document-based solution. Particularly for unstructured medical records, doctor notes, and photos, NoSQL databases as MongoDB and Firebase give flexible data storage.

Schematical flexibility is one of NoSQL's main benefits. NoSQL lets data be stored in a flexible JSON format, unlike relational databases which demand predetermined table structures, therefore facilitating handling of dynamic and changing medical records. Electronic health records (EHRs), patient histories, and imaging data—which do not fit well into organized tables—especially benefit from this.

Additionally providing great performance for big datasets are NoSQL databases. Their horizontal scale lets distributed storage among several servers, so enhancing response speeds for real-time searches. Furthermore geared for API-based interactions, NoSQL databases are perfect for cloud-based patient portals and mobile healthcare projects.

NoSQL databases do, nevertheless, also have drawbacks. Lack of ACID transactions presents one of the toughest difficulties. Most NoSQL databases give availability and partition tolerance top priority, unlike relational databases which enforce tight consistency through ACID features. This could result in disparities in financial transactions, medical billing, and appointment scheduling. NoSQL is therefore inappropriate for uses requiring great data consistency and referential integrity.

The complexity of searching NoSQL presents still another drawback. NoSQL databases lack SQL-style joins, so it is challenging to execute sophisticated searches requiring linking several datasets, including retrieving a patient's medical history together with billing and prescription records.

# R. Activity Log

*Description: As an appendix, the team will keep a frequently updated diary or log of their activity. What did you or your team study in this class each day? What did you learn? What did you accomplish or build or design? You don't have to enter something every day, but there should be at least three entries each week. Since we have eight weeks, that means you should make 3 posts to the Activity Log each week, for a total of at least 24 posts. Each post will be worth 1 point.*

*If you are working as part of a team, make sure you clearly identify which team member worked on which tasks. The Activity Log should help me figure out how each team member contributed to the project. If I cannot discern who worked on what aspects of the project from the activity log, no points will be awarded for it.*

*Total points possible: 24*

MAKE AT LEAST THREE ENTRIES PER WEEK. CLEARLY IDENTIFY WHAT EACH PERSON ON YOUR TEAM ACCOMPLISHED. YOU MUST SHARE THE RESPONSIBILITY OF COMPLETING THE PROJECT.

|  |  |  |
| --- | --- | --- |
| Weeks | Student Name | Activity |
| Week 1 | Vamshi | Researched on various topics and came up with 3 topics |
|  | Bharath | Researched on various topics and came up with 2 topics |
|  | Harini | Researched on various topics and came up with 4 topics |
|  | Vamshi, Harini, Bharath | Discussed each project and finalized 1 topic |
| Week 2 | Vamshi | Worked on Initial process |
|  | Bharath | Worked on gathering data |
|  | Harini | Worked on Alternative Data Storage Models |
|  | Vamshi | Worked on structure of data |
|  | Bharath | Worked on how to transform the data into relational tables |
|  | Harini | Created finalized tables |
| Week 3 | Harini | Conversion of raw data into tables |
|  | Bharath | Found constraints in data |
|  | Harini | Found functional dependencies |
|  | Vamshi | Converted the schema into 3NF |
|  | Vamshi | Created 3NF ERD |
|  | Bharath | Wrote Create table queries |
|  | Harini | Wrote script to get insert query statements |
|  |  | Worked on insertion statements |
| Week 4 | Bharat Vamshi h Harini Vamshi | Revised the design according to the feedback |
| Week 5 | Harini | Worked on DML |
|  | Bharath | Worked on Indexing |
|  | Vamshi | Worked on Views |
| Week 6 | Harini | Worked on transactions |
|  | Bharath Harini Vamshi | Security |
| Week 7 | Bharath Harini Vamshi | Revised the joins and indexing issues as per feedback |
|  | Bharath | Worked on Suggestions, Limitations |
|  | Harini | Worked on Stored Procedures |
|  | Vamshi | Worked on Locking |
|  | Vamshi | Worked on Backup Strategies |
|  | Bharath Harini Vamshi | Worked on python Program |