**Electronic Medical Record System**

**Project -2**

**Name : Nitin Sai Talluri**

**Student ID: 801324883**

**Due date: May 5th 2023**

**Introduction:**

**Project Overview:**

Every hospital or clinic should have a system in place for handling electronic health records. This system maintains a record of all pertinent patient information, including name, diagnosis of sickness, contact information (phone number, email address, address, etc.), and diagnosis and prescription information for clinical care.  It also keeps a list of doctors with information about each one, including name, specialization, and phone and email numbers. Additionally, it logs information about the exam room and materials. The system also keeps track of information concerning insurance, such as the provider and plan specifics. Last but not least, visit information upkeep information pertaining to visit date and time.This project is an extension of project 1 on which we use an application programming interface using stored procedures that allows an EMR system to work without having a database and writing sql queries. I have used python programming language to implement. This model also provides features like authentication on authorized people who are supposed to use the database and certain constraints are implemented so that only doctors can update prescriptions. And also features like saving the the history every time a record is accessed.

Functional Requirements specification : This system should be able to create efficient information management system for an student clinic with use of API so no need to use the work bench . Here clinic uses a database to store patient information, visit history, insurance details, care details, prescriptions, exam room details and supplies , billing records and Auth table is add for authentication. Also, the system should make it possible to manage medications, schedule visits, and handle billing procedures effectively. The main objective is to develop a system that uses api to intract with the user without use of workbench for that I have chosen visual studio to work and the system is all efficient to store the audits if any changes are made.

New tables added to the database to satisfy the requirement specified in the porject I have taken the new table related to the audit to store the details of the changes. All the tables are specifiend in functional requirement

**Database Design:**

The database design consists of several tables that are interrelated with each other using foreign key constraints. The following is the database schema design for the given tables:

Patients\_details:

Patient\_ID (Primary Key, INT): Unique identifier for each patient

First\_name (VARCHAR(100)): First name of the patient

Last\_name (VARCHAR(100)): Last name of the patient

dob (DATE): Date of birth of the patient

Phone\_number (VARCHAR(20)): Phone number of the patient

Email\_address (VARCHAR(100)): Email address of the patient

Address (VARCHAR(150)): Address of the patient

City (VARCHAR(50)): City of the patient

State (VARCHAR(50)): State of the patient

ZipCode (VARCHAR(15)): Zip code of the patient

Insurance\_details:

Insurance\_provider (VARCHAR(50)): Name of the insurance provider

Insurance\_id (Primary Key, INT): Unique identifier for each insurance policy

Insurance\_group (VARCHAR(50)): Group name of the insurance policy

Insurance:

Patient\_ID (Foreign Key, INT): Unique identifier of the patient

Insurance\_id (Foreign Key, INT): Unique identifier of the insurance policy

Doctor\_details:

Provider\_id (Primary Key, INT): Unique identifier for each doctor

First\_name (VARCHAR(100)): First name of the doctor

Last\_name (VARCHAR(100)): Last name of the doctor

Specialty (VARCHAR(50)): Specialty of the doctor

Phone\_number (VARCHAR(20)): Phone number of the doctor

Email\_address (VARCHAR(100)): Email address of the doctor

Visit\_details:

Visit\_id (Primary Key, INT): Unique identifier for each visit

Patient\_id (Foreign Key, INT): Unique identifier of the patient

Provider\_id (Foreign Key, INT): Unique identifier of the doctor

Facility (VARCHAR(100)): Name of the facility where the visit took place

Visit\_date (DATE): Date of the visit

Visit\_time (TIME): Time of the visit

Exam\_room (VARCHAR(20)): Room number where the exam took place

Billing\_code (VARCHAR(20)): Billing code for the visit

Clinicalcare\_details:

Care\_id (Primary Key, INT): Unique identifier for each care provided

Visit\_id (Foreign Key, INT): Unique identifier of the visit

Diagnosis (VARCHAR(700)): Diagnosis provided during the visit

Symptoms (VARCHAR(700)): Symptoms observed during the visit

Prescription (VARCHAR(700)): Prescription given during the visit

Lab\_order (VARCHAR(700)): Lab order given during the visit

Lab\_results (VARCHAR(700)): Lab results received during the visit

Exam\_Room:

Room\_id (Primary Key, INT): Unique identifier for each exam room

Facility (VARCHAR(100)): Name of the facility where the exam room is located

Room\_number (VARCHAR(20)): Room number of the exam room

Supplies:

Supply\_id (Primary Key, INT): Unique identifier for each supply

Name (VARCHAR(100)): Name of the supply

Description (VARCHAR(700)): Description of the supply

Quantity (INT): Quantity of the supply

Supplier\_name (VARCHAR(100)): Name of the supplier of the supply

Supplier\_phonenumber (VARCHAR(20)): Phone number of the supplier of the supply

Supplier\_emailadress (VARCHAR(100)): Email address of the supplier of the supply

Billing\_details:

Billing\_id (Primary Key, INT): Unique identifier for each billing

Visit\_id (Foreign Key, INT): Unique identifier of the visit

Total\_charge(Decimal): expenses paid total by patient

Payment\_amount(Decimal): expenses cleared by patient

Balance\_due(Decimal): Due amount to clear

\*Auth Table: is used to store the values of the Authenication of the user a d their passwords.

username (varchar): Name of the user

userpassword (varchar): Password of the user

user\_is varchar( NOT NULL): The user is either doctor or patient

user\_id(INT) : User id of the person.

**Functionl Requirments:**

Patients\_details: Entity to store patient details including patient ID, first name, last name, date of birth, phone number, email address, address, city, state, and zip code, action\_type, action\_date, action\_by Action\_name one show ne made the change teir details are stored here.

Insurance\_details: Entity to store insurance provider details including insurance provider name, insurance ID, and insurance group.

Insurance: Entity to store insurance information for each patient including patient ID and insurance ID. This entity has foreign key constraints to Patients\_details and Insurance\_details tables.

Doctor\_details: Entity to store doctor details including provider ID, first name, last name, specialty, phone number, and email address.

Visit\_details: Entity to store visit details including visit ID, patient ID, provider ID, facility name, visit date, visit time, exam room number, and billing code. This entity has foreign key constraints to Patients\_details and Doctor\_details tables.

Clinicalcare\_details: Entity to store clinical care details including care ID, visit ID, diagnosis, symptoms, prescription, lab order, and lab results. This entity has a foreign key constraint to Visit\_details table.

Exam\_Room: Entity to store exam room details including room ID, facility name, and room number.

Supplies : The system should allow the clinic to manage supplies, including details such as supply ID, name, description, quantity, supplier name, supplier phone number, and supplier email address also the ability to add, edit, and delete supply information. Also use search and filter supply information by various criteria such as name and quantity.

Billing Management: The system should allow the clinic to manage billing, including the recording of visit details and billing amounts such as total charge, payment amount, and balance due. This table should include a foreign key referencing the Visit\_details table.

New Tables

\*Auth: Information about user authentication is kept in a system's "Auth" table. In addition to a unique identification, it has areas for the user's username and password. The table's layout is intended to support safe user authentication and authorisation inside the framework of the system.

history\_Patients\_details:

Record patient details (first name, last name, DOB, phone number, email, address, city, state, zip code) and track any updates/changes made to these details (action type, action date, action by, action name).

Allow authorized users to view the patient details and their history of changes.

history\_Doctor\_details:

Record doctor details (first name, last name, specialty, phone number, email) and track any updates/changes made to these details (action type, action date, action by, action name).

Allow authorized users to view the doctor details and their history of changes.

history\_Insurance\_details:

Record insurance details (insurance provider, insurance ID, insurance group) and track any updates/changes made to these details (action type, action date, action by, action name).

Allow authorized users to view the insurance details and their history of changes.

history\_Insurance:

Record the relationship between a patient and their insurance provider (patient ID, insurance ID) and track any updates/changes made to this relationship (action type, action date, action by, action name).

Allow authorized users to view the patient-insurance relationship and its history of changes.

history\_Clinicalcare\_details:

Record clinical care details (care ID, visit ID, diagnosis, symptoms, prescription, lab order, lab results) and track any updates/changes made to these details (action type, action date, action by, action name).Allow authorized users to view the clinical care details and their history of changes.

history\_Visit\_details:

Record visit details (visit ID, patient ID, provider ID, facility, visit date, visit time, exam room, billing code) and track any updates/changes made to these details (action type, action date, action by, action name).Allow authorized users to view the visit details and their history of changes.

history\_Billing\_details:

Record billing details (billing ID, visit ID, total charge, payment amount, balance due) and track any updates/changes made to these details (action type, action date, action by, action name).Allow authorized users to view the billing details and their history of changes.

history\_Supplies:

Record supply details (supply ID, name, description, quantity, supplier name, supplier phone number, supplier email address) and track any updates/changes made to these details (action type, action date, action by, action name).Allow authorized users to view the supply details and their history of changes.

history\_Exam\_Room:

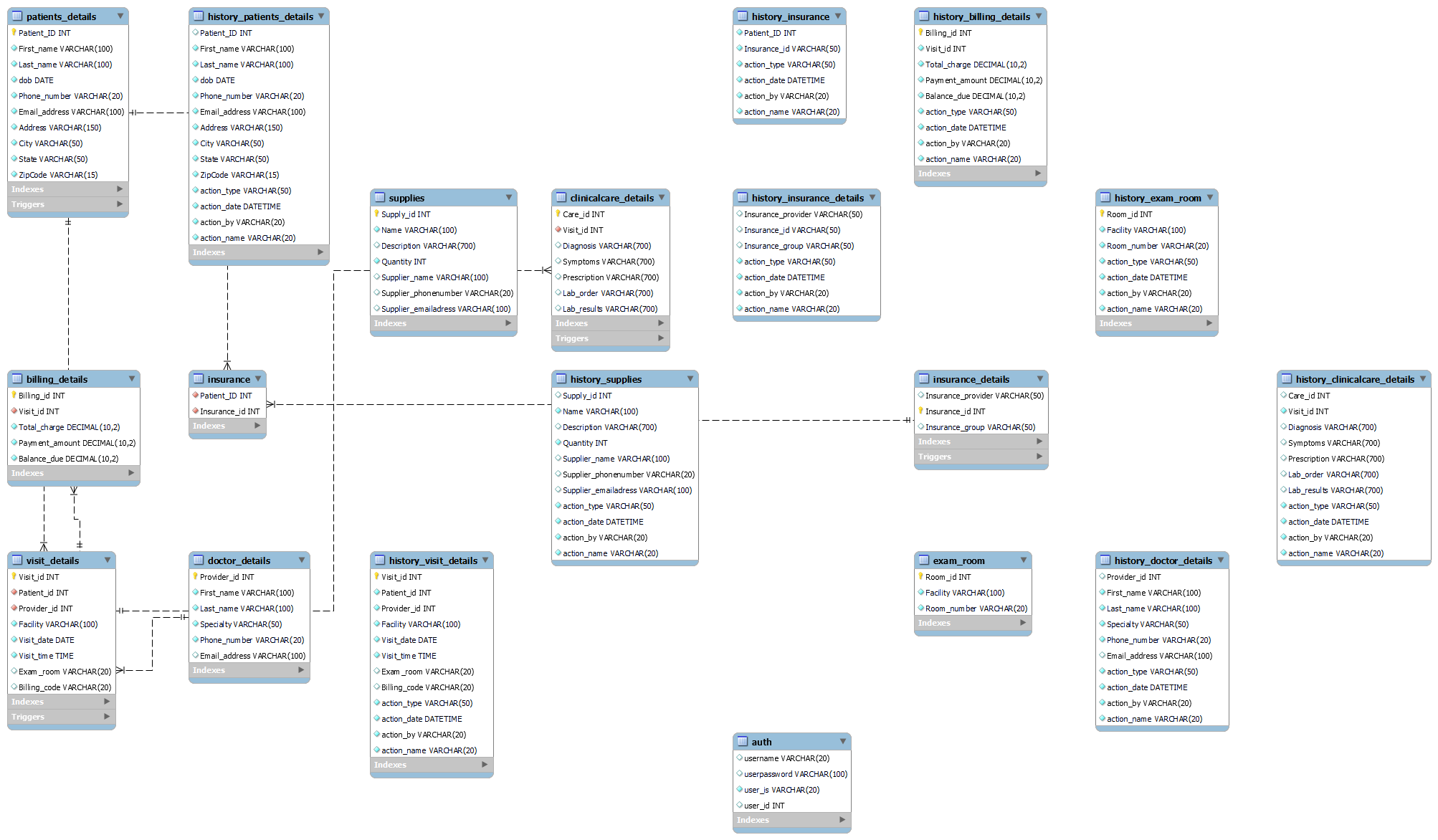
Record exam room details (room ID, facility, room number) and track any updates/changes made to these details (action type, action date, action by, action name).Allow authorized users to view the exam room details and their history of changes.

**Entity relationship:**

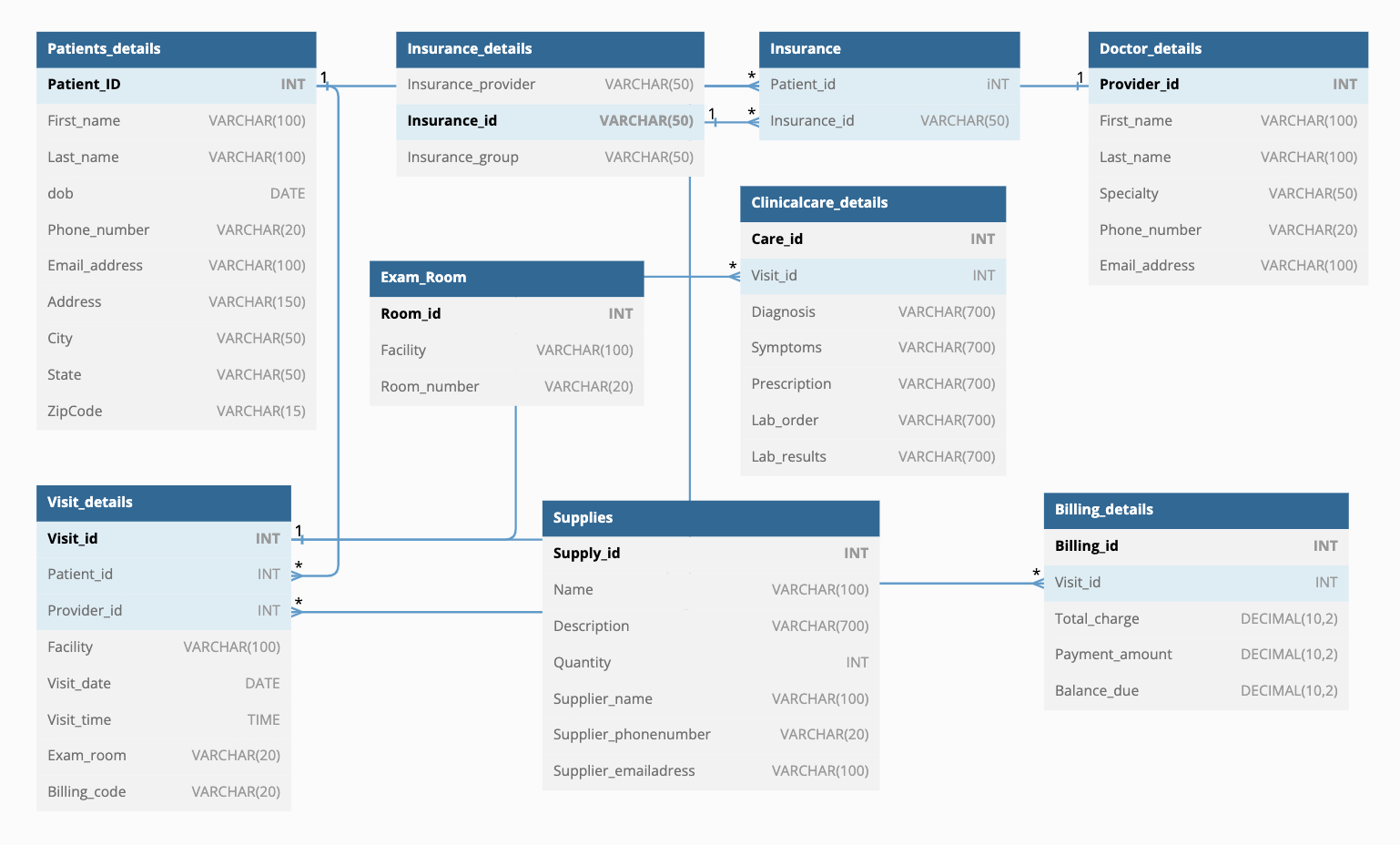
ER Diagram:

ER Diagram or ER model, is a type of structural diagram for use in database design

The below diagram is the updated er diagram of project one newly we have created the authentication table and the other table related to the audit this tables are used stored to the history what access are functioned.



UML Diagram: A system or software design is graphically represented in a UML (Unified Modeling Language) diagram. It is used to depict, describe, and record the composition, operation, and relationships of various system components. Use case diagrams, class diagrams, sequence diagrams, activity diagrams, and others are examples of UML diagrams. They offer a standardized means of communication and comprehension of the system design and requirements for developers, designers, and stakeholders.



**Proof of BCNF:**

To prove that the relationship is in BCNF (Boyce-codd normal form). If all non-key characteristics of a table are completely dependent on the candidate key and there are nontrivial functional relationships among non-key attributes, then the table is in BCNF. BCNF is a higher level of database normalization than the third normal form (3NF). BCNF eliminates redundancy in a database and makes it more efficient and easier to maintain.

Patients\_details: The primary key is Patient\_ID, and all other attributes are dependent on it. There are no functional dependencies between non-key attributes.

Insurance\_details: The primary key is Insurance\_id, and all other attributes are dependent on it. There are no functional dependencies between non-key attributes.

Insurance: This table has a composite primary key (Patient\_ID, Insurance\_id), and both attributes are necessary to uniquely identify a row. The foreign key constraints ensure that the table satisfies the BCNF.

Doctor\_details: The primary key is Provider\_id, and all other attributes are dependent on it. There are no functional dependencies between non-key attributes.

Visit\_details: The primary key is Visit\_id, and all other attributes are dependent on it. There are no functional dependencies between non-key attributes.

Clinicalcare\_details: The primary key is Care\_id, and all other attributes are dependent on it. There are no functional dependencies between non-key attributes.

Exam\_Room: The primary key is Room\_id, and all other attributes are dependent on it. There are no functional dependencies between non-key attributes.

Supplies: The primary key is Supply\_id, and all other attributes are dependent on it. There are no functional dependencies between non-key attributes.

Billing\_details: The primary key is Billing\_id, and all other attributes are dependent on it. There are no functional dependencies between non-key attributes.

New Tables : the new table created to store the user authentation details this table have no connection with other and no primary key and forigen key constraints. There are no functional dependencies between non-key attributes.

**Table information:**

Patients\_details - stores details of patients including their unique patient ID, first name, last name, date of birth, phone number, email address, address, city, state, and zip code.

Insurance\_details - stores details of insurance providers including their name, ID, and group.

Insurance - stores the relationship between patients and their insurance providers using foreign keys to link to the Patients\_details and Insurance\_details tables.

Doctor\_details - stores details of doctors including their unique provider ID, first name, last name, specialty, phone number, and email address.

Visit\_details - stores details of visits including the unique visit ID, patient ID, provider ID, facility, visit date, visit time, exam room, and billing code.

Clinicalcare\_details - stores details of clinical care provided during a visit including the unique care ID, visit ID, diagnosis, symptoms, prescription, lab order, and lab results.

Exam\_Room - stores details of exam rooms including the unique room ID, facility, and room number.

Supplies - stores details of supplies including the unique supply ID, name, description, quantity, supplier name, supplier phone number, and supplier email address.

Billing\_details - stores details of billing including the unique billing ID, visit ID, total charge, payment amount, and balance due.

NEW Tables

\*Auth - stores authentication details including username, hashed user password, user role, and user ID.

history\_Patients\_details: contains historical information about patients, including their personal information (name, DOB, phone number, email, address), and any actions performed on their record (type of action, date, by whom, and the name of the action).

history\_Doctor\_details: contains historical information about doctors, including their personal information (name, specialty, phone number, email), and any actions performed on their record (type of action, date, by whom, and the name of the action).

history\_Insurance\_details: contains historical information about insurance providers, including their name, ID, and group, and any actions performed on their record (type of action, date, by whom, and the name of the action).

history\_Insurance: contains historical information about patients' insurance, including their ID and any actions performed on their record (type of action, date, by whom, and the name of the action).

history\_Clinicalcare\_details: contains historical information about clinical care, including the care and visit IDs, diagnosis, symptoms, prescription, lab order, lab results, and any actions performed on their record (type of action, date, by whom, and the name of the action).

history\_Visit\_details: contains historical information about patient visits, including the visit ID, patient and provider IDs, facility, visit date, visit time, exam room, billing code, and any actions performed on their record (type of action, date, by whom, and the name of the action).

history\_Billing\_details: contains historical information about billing, including the billing ID, visit ID, total charge, payment amount, balance due, and any actions performed on their record (type of action, date, by whom, and the name of the action).

history\_Supplies: contains historical information about supplies, including the supply ID, name, description, quantity, supplier name, phone number, email address, and any actions performed on their record (type of action, date, by whom, and the name of the action).

history\_Exam\_Room: contains historical information about exam rooms, including the room ID, facility, room number, and any actions performed on their record (type of action, date, by whom, and the name of the action).

**API IMPLEMENTATION:**

User Authentication:

User authentication is the process of confirming a user's identity. Typically, it involves asking the user for some kind of identification, like a username and password. Only those users whose identities have been verified by the authentication procedure are given access to the resources or services they are allowed to use. This assists in maintaining the data's security and integrity and preventing illegal access or change.

When we log on to the home page it askes to enter weather your are user , patient or other new entry. If you are an registred user like doctor or patient it will display to enter username and password if the details are correct it shows next features if invaid display to reenter details.

List of users how can access

('jack', '1234','patient',2 ),

('john', '1234','patient',1 ),

('smith', '1234','patient',3 ),

('smith', '12345','patient',4 ),

('joye', '1234','patient',5 ),

('John','1234','doctor',1),

( 'Jack', '1234','doctor',2),

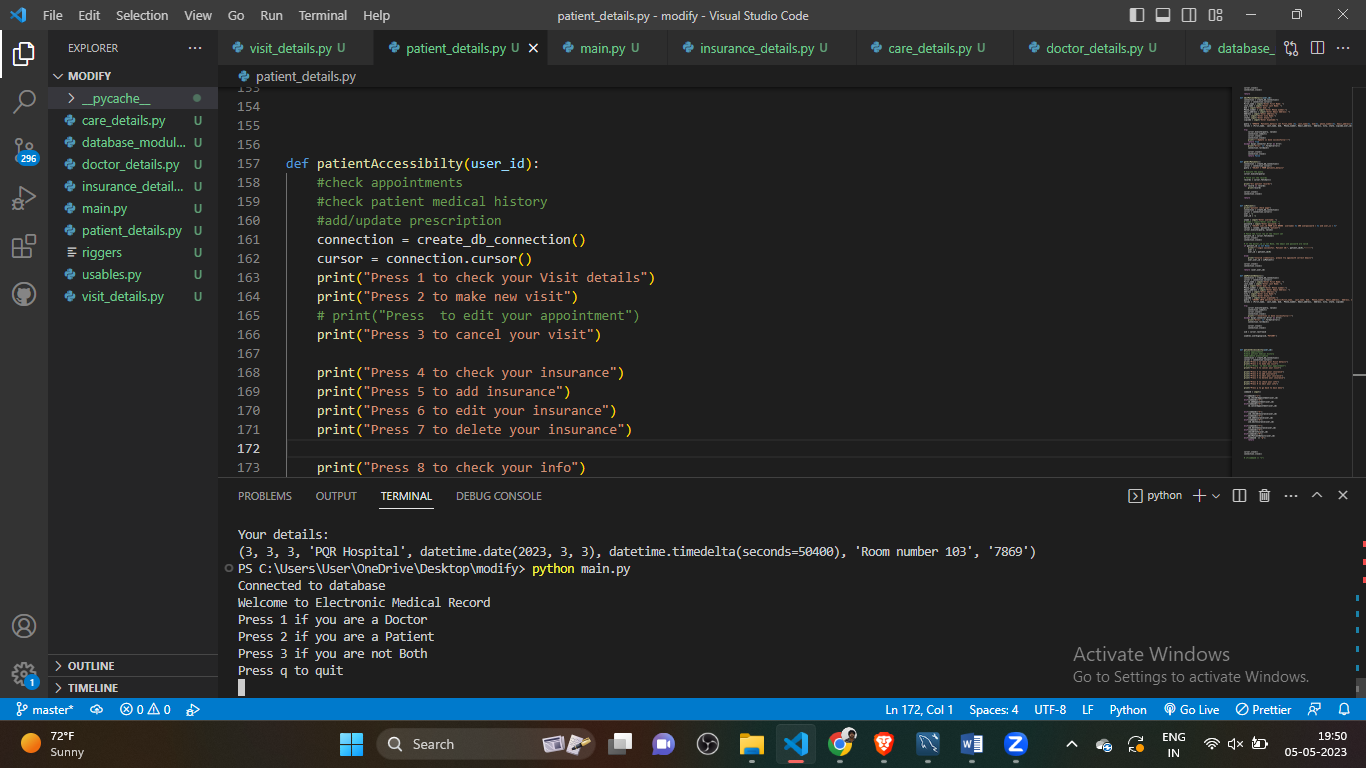
('Smith','1234','doctor',3),

('Jack','1234','doctor',4),

('Daniel','1234','doctor',5),

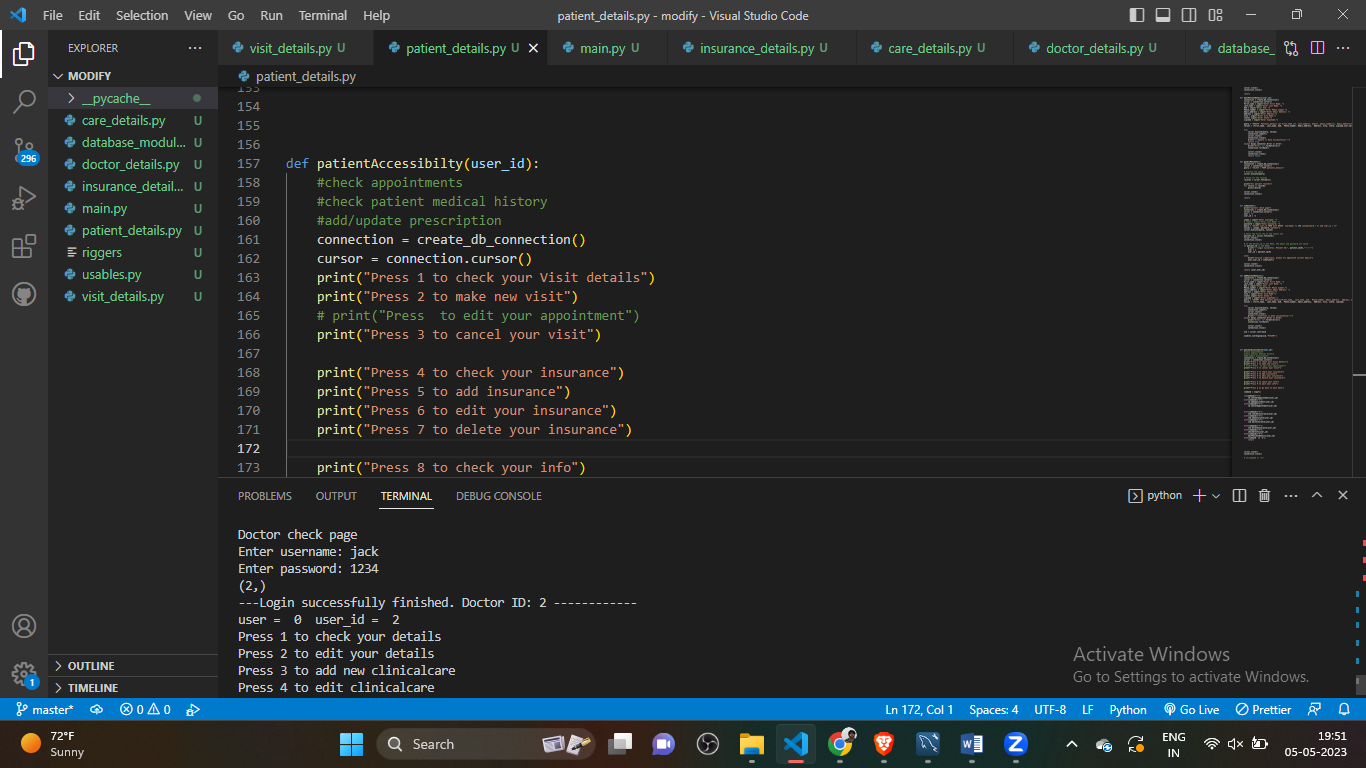
( 'Alicia','1234','doctor',6);

1.1 user interface by running main

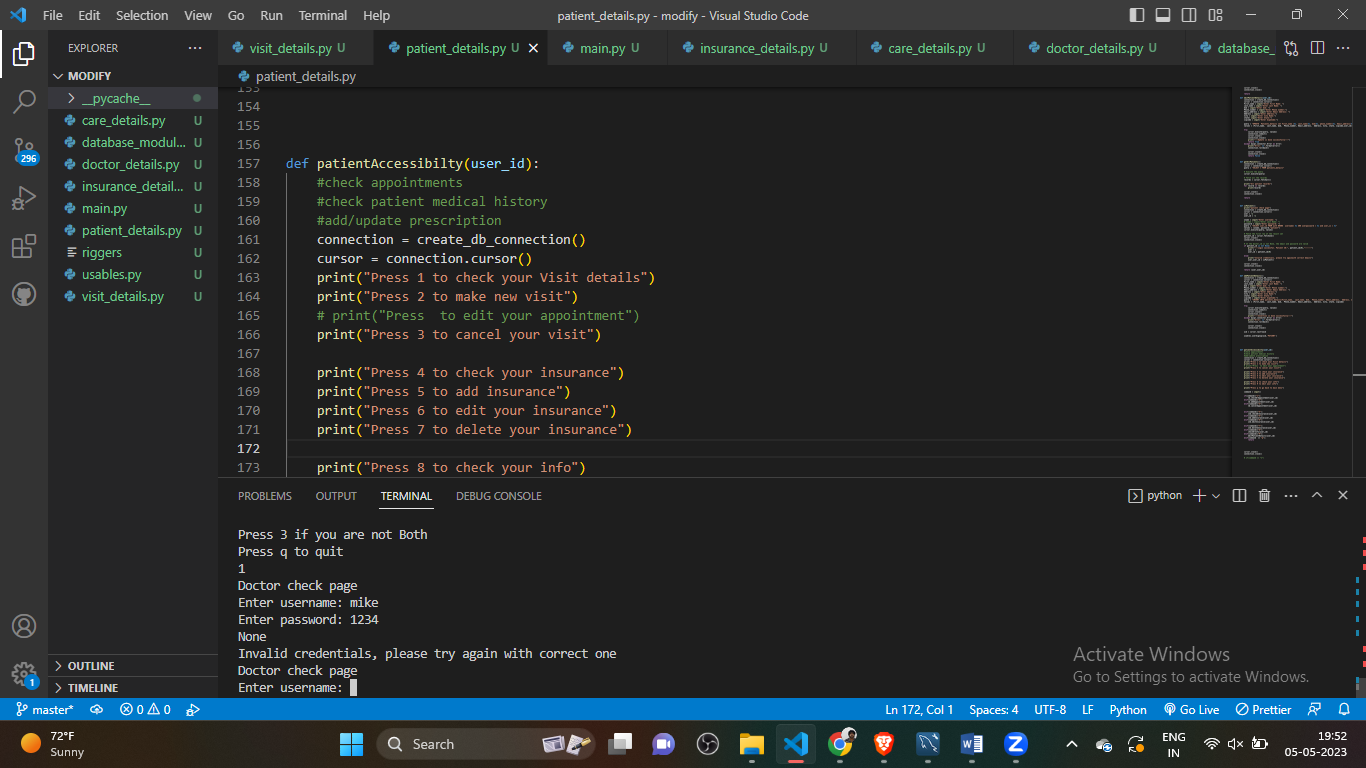


1.2

Enter valid doctor details gets accepted so logins



1.3 invalid details give its not logged in shows to try with correct details

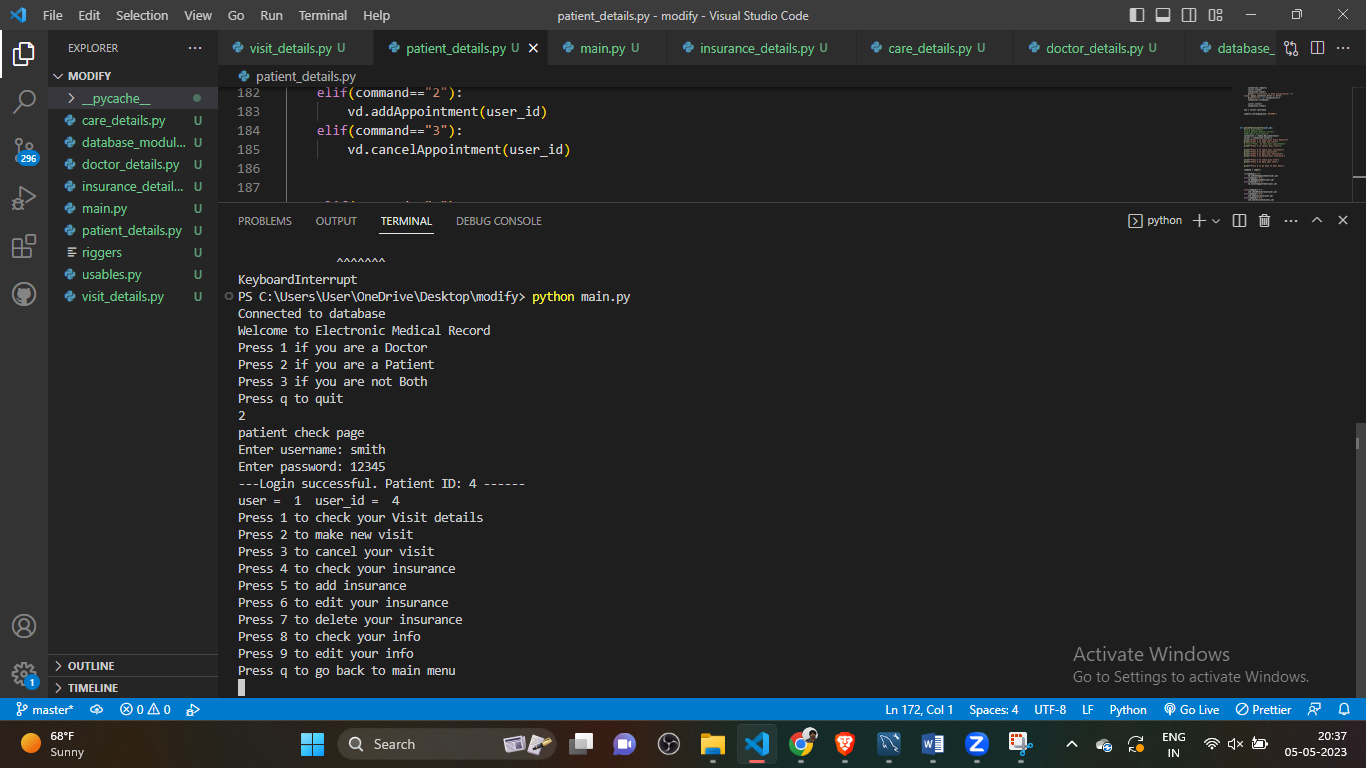


Role Based Access:

The method of limiting access to database resources based on the roles given to certain users within an organization is known as role-based access control (RBAC). According to RBAC, users are given access to resources based on their position. This helps to prevent unauthorized access to sensitive information and ensures that users may only access the resources they require to carry out their duties.

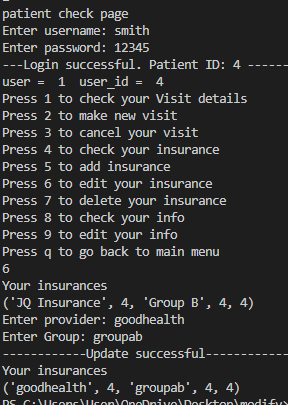
I have used role based access so that the patient cannot edit the prescription details or modify it. The access is given to doctor only based on the role the access is permited. It is based on a user's role or job responsibilities, which determine the permissions and level of access they have to the system's resources.

Student have no option to edit care details which is precription in my table

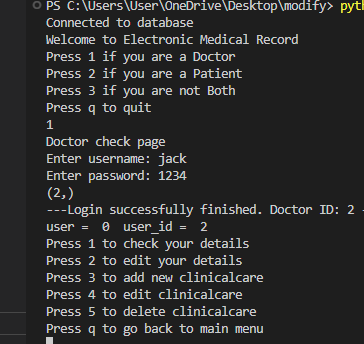


The patient have feature like check visit details, make visits, edit insurance etc..,

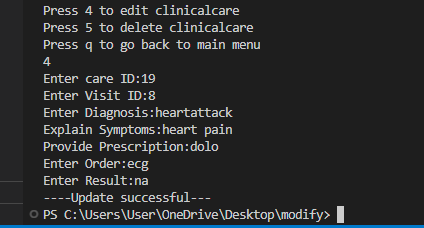
Below is the example for running one feature which is editing insurance



Doctor have access to edit clinical care detail which is prescription in my database

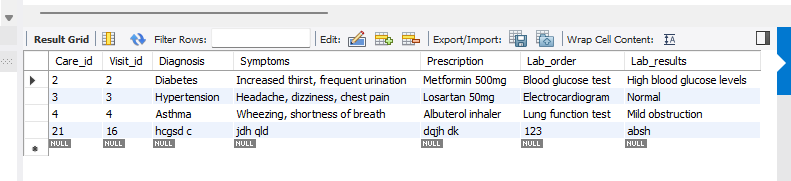


I have tested doctor feature to edit the clinicalcare details below is the successful update performed.

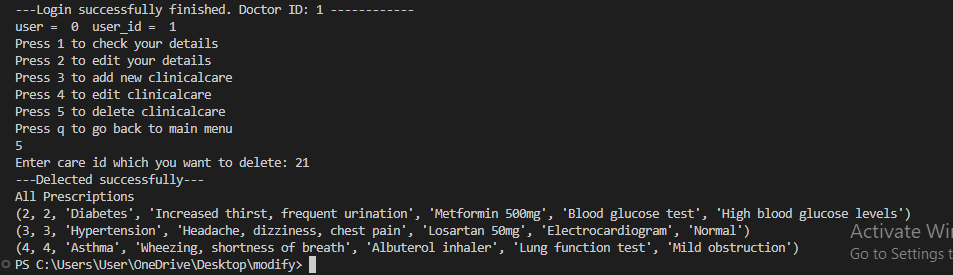


The doctor who logged into the system and delected the clinical care details which patient have no feature to delete care details.

Before delete



After successfully deleted

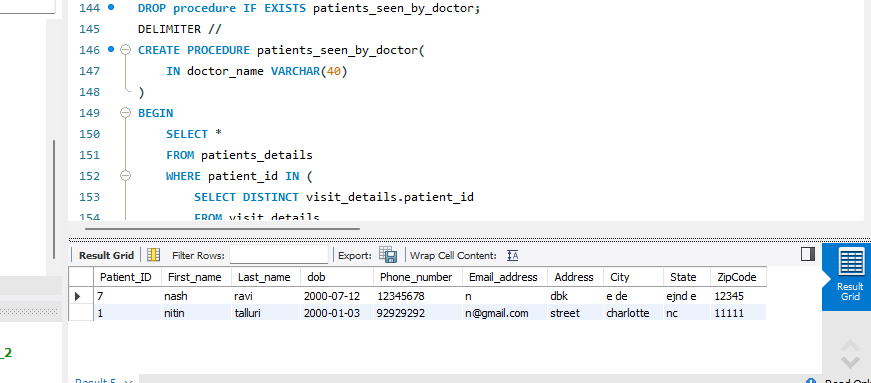


Stored Procedures:

The ability to perform complex SQL statements and logical operations in a single step is provided by stored procedures, which are precompiled database programs that are saved. Code reuse and performance are the two key benefits of using stored procedures. Data validation, constraints, and triggers can all be implemented in the database using stored procedures. They can also be used to carry out database maintenance procedures including data archiving, index rebuilding, and backups.

You can encapsulate and reuse SQL queries and logic by using stored procedures, which are database objects that have already been built. By giving users a single point of access to run complex queries and update many tables, they may streamline complex tasks and improve speed while lowering network traffic. The ability to grant access to the procedure only rather than the underlying tables is another way that stored procedures can improve security.

I have done stored procedure for patients\_seen\_by\_doctor this give the patiens details where doctor last name is doe.



Indexes:

A database table's index is a data structure that accelerates data retrieval processes. A table's data can be effectively accessed by queries using an index rather than a full table scan by acting as a pointer to the data's location in the table. SELECT queries, which obtain data from a database table, are sped up by indexes. They function by building a different data structure with a duplicate of the data that is being indexed. It is made simpler and quicker to search for certain data thanks to the way this data structure is set up.

I have used index for the below data index names for each table:

Patients\_details: Patient\_ID,Insurance\_

details: Insurance\_id

Insurance: Patient\_ID, Insurance\_id

Doctor\_details: Provider\_id

Visit\_details: Visit\_id

Clinicalcare\_details: Care\_id

Exam\_Room: Room\_id

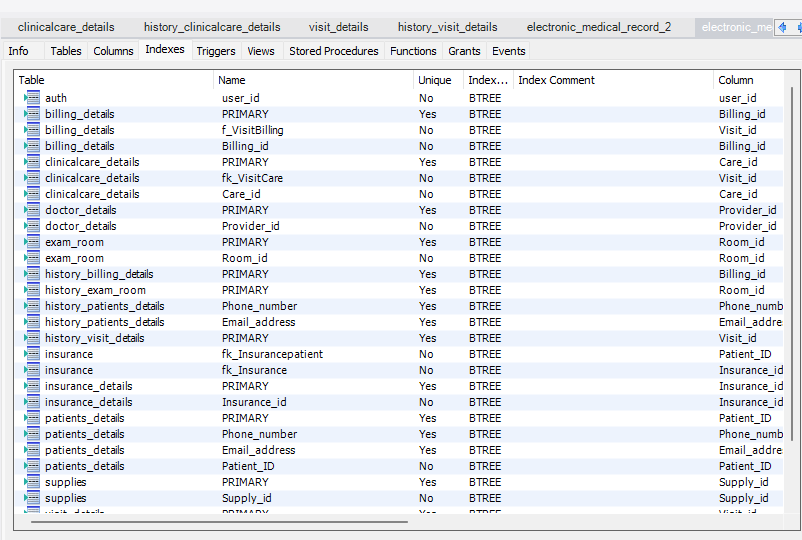
Supplies: Supply\_id

Billing\_details: Billing\_id

Auth: user\_id

Indexes are used to improve database query performance by allowing the database system to locate and retrieve data more quickly and efficiently. I have used inxdexes in below tables

Indexes screenshot indexe are present in this table

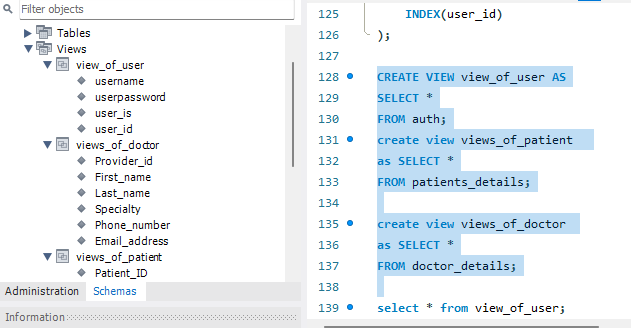


Views:

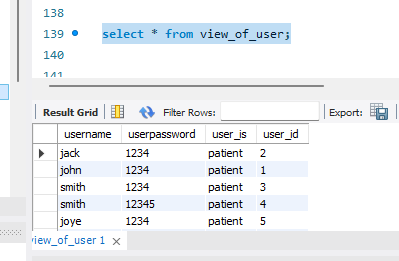
A view in SQL is a fictitious table that shows information from one or more tables. A SELECT statement that extracts and formats data from one or more tables is defined, and then the statement is saved as a view. A view can be used in the same way as a table once it has been constructed.

Views are virtual tables that provide a specific perspective on the data from one or more underlying tables. They are used to simplify complex queries, restrict data access, and enhance security. Views can also be used to summarize or aggregate data and provide a consistent interface for users with varying needs

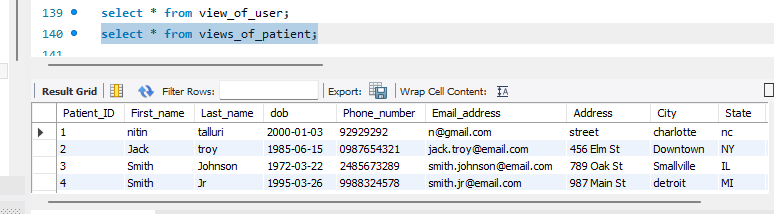
I have used views tolower the time for the tables below are the screenshots of views of auth, patient and doctor.



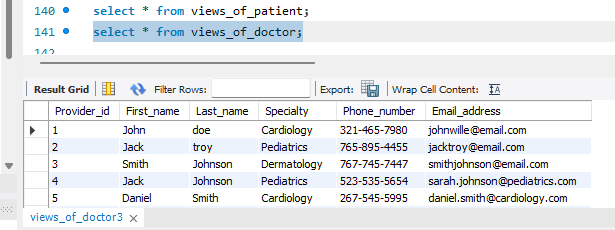
View of user



Views of patient



Views of doctor



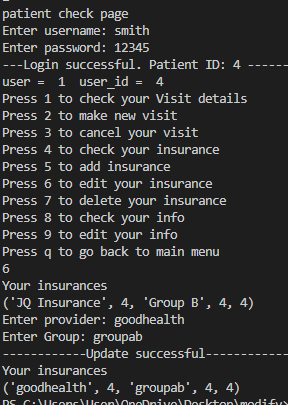
Audit Trails:

A system's audit trail is a record of all its operations or modifications. You may view who performed what actions when, where, and when in the system. A mechanism to track any illegal or inadvertent alterations to data or configuration settings is provided by an audit trail. This can assist in locating security holes, data loss, and other problems that might affect the system's functionality. Triggers and stored procedures can be used to implement audit trails in databases. Data changes, such as when a record was created, edited, or destroyed, can be tracked using audit trails. They can also be used to monitor database schema changes, such as the addition of a new table or column. I have used using triggers.

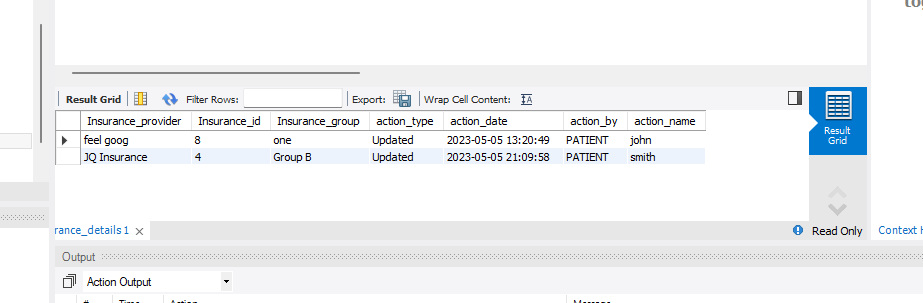
An audit trail is used to track and record changes made to a system or data, providing a chronological history of who accessed it, when, and what changes were made. It can help in identifying errors, unauthorized access, and data breaches, and assist in investigating and resolving any issues.

Audit is taken as history in my database the audit stores the previously modified data

Below is the example of edit insurance the details are stored in audit which is in history\_insurance\_details



Here is the screen shot referring that the data is stored in history\_insurance\_details



**Tools Used:**

Visual Studio

MySQL Workbench