**A green and gold logo

Description automatically generated**  
  
Project 2

Electronic Medical Record System Database based on Dermatology Clinic

Team members

Anushka Santosh Padyal – 801379909

Nishant Acharekar – 801363902

Shivangi Saxena – 801372350

Bulbul Roy – 801365911

Contents

[Introduction 3](#_Toc165666945)

[Functional Requirements 3](#_Toc165666946)

[Entity Relationship 4](#_Toc165666947)

[Proof of BCNF 6](#_Toc165666948)

[Table Information 9](#_Toc165666949)

[Stored Procedure Information 11](#_Toc165666950)

[User authentication/ Role-based access control 12](#_Toc165666951)

[Tables for audit trail (triggers) 14](#_Toc165666952)

[Indexes 14](#_Toc165666953)

[Views 15](#_Toc165666954)

[Tools Used 15](#_Toc165666955)

[References 15](#_Toc165666956)

# Introduction

In healthcare, the emergence of digital technologies has brought about a revolutionary change in the delivery and management of patient care. To encompass essential aspects of EMR systems, we designed the 'Electronic Medical Record System Database based on the Dermatology Clinic project. This database system has been designed to streamline the processes involved in a dermatology clinic and promises to improve the efficiency of managing patient records, scheduling appointments, and tracking treatments. The system ensures a seamless integration of clinical operations by meticulously structuring tables to capture patient information, staff credentials, insurance details, and transaction records. Every effort is made to make this project error-free and efficient, ensuring the robustness of the system.

# Functional Requirements

1. Patient Management
   1. The system shall allow the registration of new patients, including capturing personal and insurance information.
   2. The system shall provide the ability to update and maintain patient records.
2. Appointment Scheduling
   1. The system shall enable staff to schedule, reschedule, and cancel appointments for patients.
3. Staff Management
   1. The system shall maintain staff records, including personal details, professional credentials, and employment information.
   2. The system shall support the assignment of roles and responsibilities to staff members.
4. Insurance Handling
   1. The system shall record and track insurance details and verify insurance coverage for treatments.
   2. The system shall process insurance claims and handle billing procedures.
5. Medical Record Keeping
   1. The system shall allow doctors to enter and access patient medical records, including dermatological treatment history.
   2. The system shall ensure the confidentiality and security of patient medical records.
6. Payment Processing
   1. The system shall support various payment methods and process payments for services rendered.
   2. The system shall generate and provide detailed invoices to patients.
7. Reporting
   1. The system shall generate reports on patient visits, financial transactions, and staff performance.
   2. The system shall provide analytics for clinic performance and patient demographics.
8. Security and Compliance
   1. The system shall enforce user authentication and authorization for system access.
   2. The system shall comply with relevant healthcare regulations and data protection laws.

# Entity Relationship

1. E-R diagram

A screenshot of a computer

Description automatically generated

The database is structured to facilitate the storage and management of various dermatological health-related data efficiently. It comprises multiple tables, each designed to handle specific aspects of patient care and clinic operations. The tables are interconnected using foreign key relationships to ensure data integrity and consistency.

Patient Table  
Stores patient details including first name, last name, gender, address, phone number, email, date of birth, and insurance ID.

Staff Table  
Manages information about clinic staff such as identification number, first name, last name, email, designation, and salary.

Insurance Table  
Records insurance details including provider and plan name.

Appointment Table  
Handles appointment scheduling with details like appointment ID, patient ID, registering staff ID, doctor ID, start time and end time.

Payment Table  
Manages payment details for appointments, including payment ID, appointment ID, payment method, payment date, and total amount.

Patient symptoms Table  
Captures patient symptoms related to various skin conditions identified by appointment ID.

Patient diagnosis Table  
Records diagnosis details for patients, including diagnosis ID, patient ID, and treatment prescribed.

Medication Table  
Tracks prescribed medications for patients with medicine ID, diagnosis ID, appointment ID, and medication dose.

Medicine avail Table  
Manages available medications with details like medicine ID, name, brand, and price.

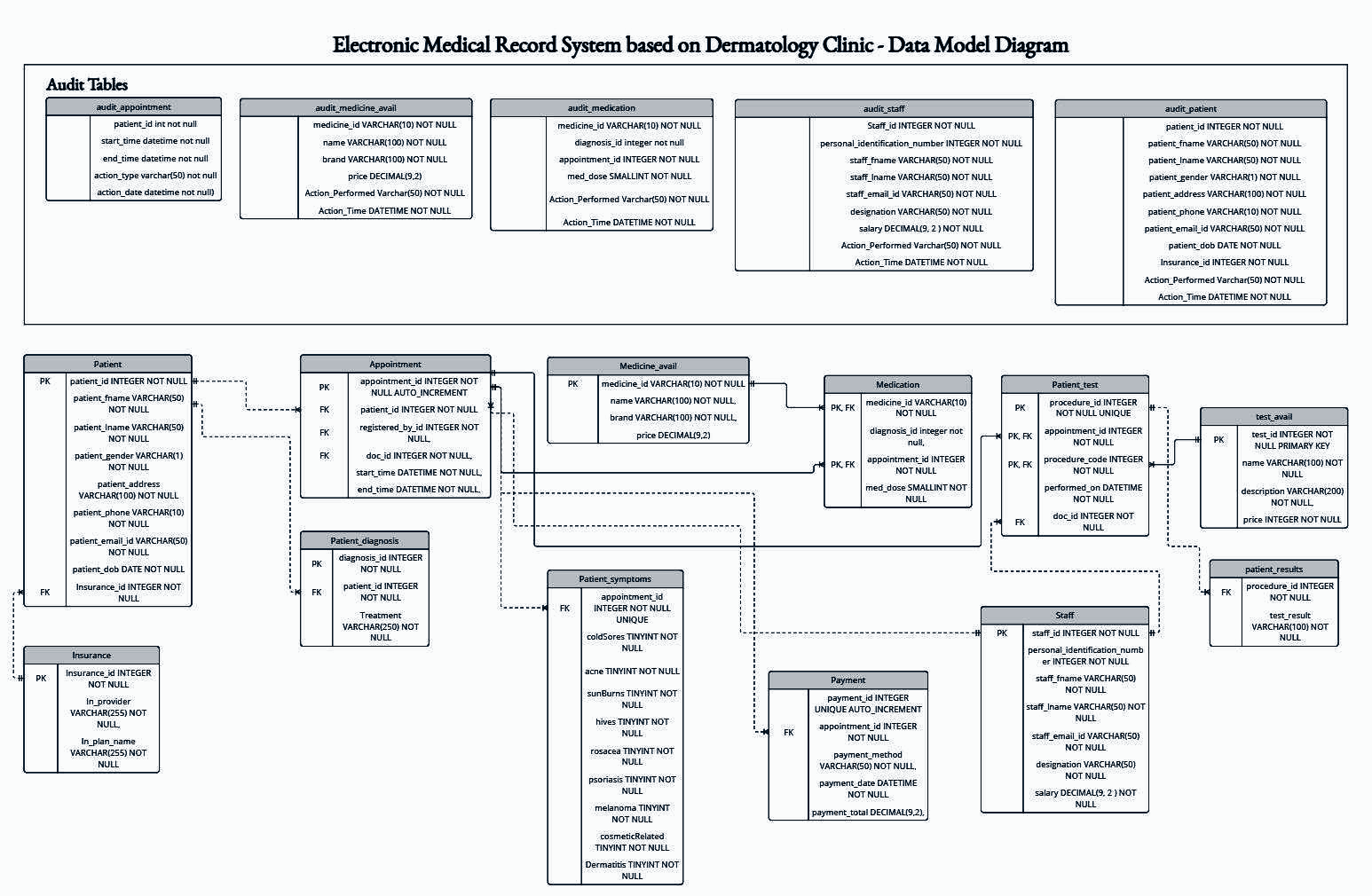
Test avail Table  
Lists available tests or procedures with a unique test ID, name, description, and price.

Patient test Table  
Records tests or procedures performed for patients with procedure ID, appointment ID, procedure code, performed date, and doctor ID.

Patient results Table: Stores test results for patients with procedure ID and test result details.

Audit Tables  
Maintains audit logs for appointments, patients, staff, medication availability, and medication changes with action performed and timestamp.

1. UML diagram



# Proof of BCNF

The tables are in BCNF if they are in 1NF, 2NF, and 3NF form and if they adhere to the below rules –

1. Each table has a primary key that uniquely identifies each record.
2. There are no attributes in a table that are dependent on only a part of a composite primary key.
3. There are no attributes in a table that are dependent on non-key attributes.
4. All columns that determine other columns are candidate keys themselves. This means that every non-trivial functional dependency in the table is of the form **key -> attribute**.

Below is a brief explanation of the tables with their relational schema and their functional dependencies –

1. Patient (patient\_id, patient\_fname, patient\_lname, patient\_gender, patient\_address, patient\_phone, patient\_email\_id, patient\_dob, Insurance\_id)  
     
   Primary key dependency:

patient\_id -> patient\_fname, patient\_lname, patient\_gender, patient\_address, patient\_phone, patient\_email\_id, patient\_dob, Insurance\_id  
  
Foreign key dependency:

Insurance\_id -> Insurance details  
The details of the insurance would be defined in the Insurance table referenced by Insurance\_id.  
  
There are no non-trivial functional dependencies other than the primary key.

The primary key is patient\_id, and all other attributes are dependent on it.

Hence, the Patient table satisfies BCNF.

1. Staff (Staff\_id, personal\_identification\_number, staff\_fname, staff\_lname, staff\_email\_id, designation, salary)  
     
   Primary key dependency:  
   Staff\_id -> (personal\_identification\_number, staff\_fname, staff\_lname, staff\_email\_id, designation, salary)  
     
   Similar to the Patient table, all attributes are dependent on the primary key Staff\_id. Hence, the Staff table satisfies BCNF.
2. Appointment(appointment\_id, patient\_id, registered\_by\_id, doc\_id, start\_time, end\_time)  
     
   Primary key dependency:  
   appointment\_id -> (patient\_id, registered\_by\_id, doc\_id, start\_time, end\_time)  
     
   The attributes patient\_id, registered\_by\_id, and doc\_id are foreign keys and are thus not considered in the FDs. The attributes start\_time and end\_time are dependent on the primary key appointment\_id. Hence, the Appointment table satisfies BCNF.
3. Patient\_symptoms(appointment\_id, coldSores, acne, sunBurns, hives, rosacea, psoriasis, melanoma, cosmeticRelated, Dermatitis)  
     
   Primary key dependency:  
   appointment\_id -> (coldSores, acne, sunBurns, hives, rosacea, psoriasis, melanoma, cosmeticRelated, Dermatitis)  
     
   All attributes are dependent on the primary key appointment\_id. Hence, the Patient\_symptoms table satisfies BCNF.
4. Patient\_test(procedure\_id, appointment\_id, procedure\_code, performed\_on, doc\_id)  
     
   Composite Primary key dependency:  
   (appointment\_id, procedure\_code) -> (procedure\_id, performed\_on, doc\_id)  
     
   All attributes are dependent on the composite primary key (appointment\_id, procedure\_code). Hence, the Patient\_test table satisfies BCNF.
5. Medication(medicine\_id, diagnosis\_id, appointment\_id, med\_dose)  
     
   Composite Primary key dependency:  
   (appointment\_id, medicine\_id) -> (diagnosis\_id, med\_dose)  
     
   All attributes are dependent on the composite primary key (appointment\_id, medicine\_id). Hence, the Medication table satisfies BCNF.
6. Payment(payment\_id, appointment\_id, payment\_method, payment\_date, payment\_total)  
     
   Primary key dependency:  
   payment\_id -> (appointment\_id, payment\_method, payment\_date, payment\_total)  
     
   All attributes are dependent on the primary key payment\_id. Hence, the Payment table satisfies BCNF.
7. Insurance(Insurance\_id, In\_provider, In\_plan\_name)  
     
   Primary key dependency:  
   Insurance\_id -> (In\_provider, In\_plan\_name)  
     
   The primary key is Insurance\_id, and all other attributes are dependent on it. Hence, the Insurance table satisfies BCNF.
8. Patient\_diagnosis(diagnosis\_id, patient\_id, Treatment)  
     
   Primary key dependency:  
   diagnosis\_id -> (patient\_id, Treatment)  
     
   All attributes are dependent on the primary key diagnosis\_id. Hence, the Patient\_diagnosis table satisfies BCNF.
9. Medicine\_avail(medicine\_id, name, brand, price)

Primary key dependency:

medicine\_id -> (name, brand, price)  
  
All attributes are dependent on the primary key medicine\_id. Hence, the Medicine\_avail table satisfies BCNF.

1. test\_avail(test\_id, name, description, price)  
     
   Primary key dependency:  
   {test\_id -> (name, description, price)  
     
   All attributes are dependent on the primary key test\_id. Hence, the test\_avail table satisfies BCNF.
2. patient\_results(procedure\_id, test\_result)  
     
   Foreign key dependency:  
   procedure\_id ->(test\_result)  
     
   All attributes are dependent on the primary key procedure\_id. Hence, the patient\_results table satisfies BCNF.
3. For the Audit Tables:

All the audit tables have a non-trivial functional dependency on the primary key of the respective tables they are auditing. The action performed and action time are dependent on the primary keys of the respective tables. Therefore, they also satisfy BCNF.

# Table Information

|  |  |  |
| --- | --- | --- |
| **Table Name** | **Column Names** | **Attributes** |
| Patient | patient\_id | INTEGER |
| patient\_fname | VARCHAR |
| patient\_lname | VARCHAR |
| patient\_gender | VARCHAR |
| patient\_address | VARCHAR |
| patient\_phone | VARCHAR |
| patient\_email\_id | VARCHAR |
| patient\_dob | DATE |
| Insurance\_id | INTEGER |
|  | | |
| Staff | Staff\_id | INTEGER |
| personal\_identification\_number | INTEGER |
| staff\_fname | VARCHAR |
| staff\_lname | VARCHAR |
| staff\_email\_id | VARCHAR |
| designation | VARCHAR |
| salary | DECIMAL |
|  | | |
| Insurance | Insurance\_id | INTEGER |
| In\_provider | VARCHAR |
| In\_plan\_name | VARCHAR |
|  | | |
| Appointment | appointment\_id | INTEGER |
| patient\_id | INTEGER |
| registered\_by\_id | INTEGER |
| doc\_id | INTEGER |
| start\_time | DATETIME |
| end\_time | DATETIME |
|  | | |
| Payment | payment\_id | INTEGER |
| appointment\_id | INTEGER |
| payment\_method | VARCHAR |
| payment\_date | DATETIME |
| payment\_total | DECIMAL |
|  | | |
| Patient\_symptoms | appointment\_id | INTEGER |
| coldSores | TINYINT |
| acne | TINYINT |
| sunBurns | TINYINT |
| hives | TINYINT |
| rosacea | TINYINT |
| psoriasis | TINYINT |
| cosmeticRelated | TINYINT |
| melanoma | TINYINT |
| Dermatitis | TINYINT |
|  | | |
| Patient\_diagnosis | diagnosis\_id | INTEGER |
| patient\_id | INTEGER |
| Treatment | VARCHAR |
|  | | |
| Medication | medicine\_id | INTEGER |
| diagnosis\_id | INTEGER |
| appointment\_id | INTEGER |
| med\_dose | SMALLINT |
|  | | |
| Medicine\_avail | medicine\_id | VARCHAR |
| name | VARCHAR |
| brand | VARCHAR |
| price | DECIMAL |
|  | | |
| test\_avail | test\_id | INTEGER |
| name | VARCHAR |
| description | VARCHAR |
| price | INTEGER |
|  | | |
| Patient\_test | procedure\_id | INTEGER |
| appointment\_id | INTEGER |
| procedure\_code | INTEGER |
| performed\_on | DATETIME |
| doc\_id | INTEGER |
|  | | |
| patient\_results | procedure\_id | INTEGER |
| test\_result | VARCHAR |
|  | | |
| audit\_appointment | patient\_id | INTEGER |
| start\_time | DATETIME |
| end\_time | DATETIME |
| action\_type | VARCHAR |
| action\_date | DATETIME |
|  | | |
| audit\_patient | patient\_id | INTEGER |
| patient\_fname | VARCHAR |
| patient\_lname | VARCHAR |
| patient\_gender | VARCHAR |
| patient\_address | VARCHAR |
| patient\_phone | VARCHAR |
| patient\_email\_id | VARCHAR |
| patient\_dob | DATE |
| Insurance\_id | INTEGER |
| Action\_Performed | VARCHAR |
| Action\_Time | DATETIME |
|  | | |
| audit\_staff | Staff\_id | INTEGER |
| personal\_identification\_number | INTEGER |
| staff\_fname | VARCHAR |
| staff\_lname | VARCHAR |
| staff\_email\_id | VARCHAR |
| designation | VARCHAR |
| salary | DECIMAL |
| Action\_Performed | VARCHAR |
| Action\_Time | DATETIME |
|  | | |
| audit\_medicine\_avail | medicine\_id | VARCHAR |
| name | VARCHAR |
| brand | VARCHAR |
| price | DECIMAL |
| Action\_Performed | VARCHAR |
| Action\_Time | DATETIME |
|  | | |
| audit\_medication | medicine\_id | VARCHAR |
| diagnosis\_id | INTEGER |
| appointment\_id | INTEGER |
| med\_dose | SMALLINT |
| Action\_Performed | VARCHAR |
| Action\_Time | DATETIME |
|  | | |

# Stored Procedure Information

There are several stored procedure created and we are using four of them i.e getAllPatientNames, PatientInsert, patientUpdate, and delete\_patientInfo in our project.

|  |  |
| --- | --- |
| **Procedure Names** | **Description** |
| TestAvailInsert | Inserts data into the test\_avail table, typically for adding test availability records. |
| StaffInsert | to insert staff data in staff table |
| insert\_medicaiton | to insert data into medication table |
| AppointmentInsert | to insert data appointment table. |
| PatientInsert | to Insert data into patient table. |
| pid\_match\_testing | Stored Procedure for retrieving a patient from the system |
| AppointmentUpdate | using the Stored Procedure we are updating the data in Appointment table |
| TestAvailUpdate | using the Stored Procedure we are Updating data in test\_avail table. |
| patientUpdate | updating the patient table for address phone no. and insurance id using procedure. |
| update\_medication | Updating medication table |
| StaffUpdate | To find update the staff table using stored procedure |
| update\_PatientSymptoms | to update patient symptoms |
| delete\_patientInfo | for deleting patient info from patient table |
| delete\_StaffInformation | for deleting staff information |
| delete\_Appointment | for deleting appointment |
| delete\_testAvail | for deleting test availability in the clinic |
| delete\_medciation | for deleting medication |
| delete\_medAvail | for deleting a particular medicine avaialbility |
| paymentInfo | for selecting payment info of patient |
| patientSymptoms | for selecting patient symptoms |
| patientName | for selecting patient name from patient table |
| getAllPatientNames | for selecting patient name from patient table |

# User authentication/ Role-based access control

There are several user authentication role and we are using two of them i.e admin\_staff and test&technical\_staff in our project.

|  |  |
| --- | --- |
| **User** | **Description** |
| doctor\_staff | The user can   * perform any operation on the Patient\_test table. * view patient-related data, audit records related to patients,view patient diagnosis-related data. * perform any operation on the medication and audit\_medication tables. * view patient results-related data. * The subsequent GRANT EXECUTE ON PROCEDURE statements provide the user account ‘doctor\_staff’ with execution rights for specific stored procedures within the proj\_dermatC\_db database. * These procedures include update\_PatientSymptoms, patientSymptoms, insert\_medicaiton, delete\_medciation, and update\_medication. The user can execute these procedures to perform specific actions within the database. |
| receptionist\_staff | The user can:   * perform any operation on the patient and Appointment table. * view data from the Patient\_Symptoms and staff table. * modify and view data from the payment table. * view audit-related data from the Audit\_Patient table. * execute these procedures to perform specific actions within the database. * The subsequent GRANT EXECUTE ON PROCEDURE statements provide the user account ‘receptionist\_staff’ with execution rights for specific stored procedures within the proj\_dermatC\_db database. * These procedures include StaffUpdate, StaffInsert, paymentInfo, pid\_match\_testing, patientName, patientInsert, patientUpdate, delete\_StaffInformation, delete\_patientInfo, delete\_Appointment, AppointmentUpdate, AppointmentInsert, and getAllPatientNames. |
| nursing\_staff | The user can:   * perform any operation on the Patient\_Symptoms table. * view patient data and audit records related to patients. * modify and view medication-related data. * view audit records related to medication,patient test-related data, and information about available medicines. * The subsequent GRANT EXECUTE ON PROCEDURE statements provide the user account ‘nursing\_staff’ with execution rights for specific stored procedures within the proj\_dermatC\_db database.These procedures include update\_PatientSymptoms, patientSymptoms, insert\_medicaiton, delete\_medciation, and update\_medication.The user can execute these procedures to perform specific actions within the database. |
| admin\_staff | All grants are supplied to administrative staff to ensure that the application is running properly. |
| test&technical\_staff | The user can:   * view audit-related data related to patients and appointments tables. * perform any operation on the patient\_results, patient\_test, and test\_avail tables. * view medication-related data. * The subsequent GRANT EXECUTE ON PROCEDURE statements provide the user account ‘test&technical\_staff’ with execution rights for specific stored procedures within the proj\_dermatC\_db database. These procedures include delete\_testAvail, TestAvailInsert, and TestAvailUpdate. The user can execute these procedures to perform specific actions within the database. |

# Tables for audit trail (triggers)

|  |  |
| --- | --- |
| **Triggers Name** | **Description** |
| after\_UpdateAppointment | Creating a trigger for maintaining an audit record to keep track of a patient's appointment history |
| after\_InsertAppointment | Creating a trigger for preserving an audit record to keep track of a patient's appointment change history. |
| after\_DeleteAppointment | Creating a trigger for maintaining an audit record for keeping the Record/track of appointment changes history of a patient |
| after\_InsertPatientInfo  after\_UpdatePatientInfo  after\_DeletePatientInfo | Creating a trigger for keeping an audit record of changes in patient information. |
| after\_InsertStaffInfo  after\_UpdateStaffInfo  after\_DeleteStaffInfo | Creating a trigger for keeping an audit record of changes in patient information. |
| after\_insertMedicineAvail  after\_updateMedicineAvail  after\_deleteMedicineAvail | Creating a trigger for establishing an audit record for the recording/tracking of changes in medicine availability information |
| after\_insertMedication  after\_updateMedication  after\_deleteMedication | Creating a trigger for preserving an audit record for the record/track of changes in medication availability information. |
| InsertPaymentTable\_medication | Insert into payment table using triggers from medicine and patient\_test tables. |
| UpdatePaymentTable\_medication | Use the medication table trigger to update data into payment. |
| InsertPaymentTable\_test | Use the patient\_test table to enter data into the payment. |
| UpdatePaymentTable\_test | Update the data in the payment table using the patient\_test. |

# Indexes

|  |  |
| --- | --- |
| **Indexes** | **Description** |
| patient\_information | Index to identify patients based on their first names. |
| patient\_insurance\_info | Index for identifying patients on patient insurance IDs |
| Doctor\_info | Index for identifying physicians by name. |
| Doctor\_designation\_info | Index for identifying physicians based on designation. |
| Doctor\_PIN\_info | Index for identifying physicians based on unique identification numbers. |
| medicine\_avail\_info | Index for medicine availability by name. |
| test\_avail\_info | Index to get test availability information by name |
| patient\_payment\_info | Index to obtain payment information based on payment ID |

# Views

|  |  |
| --- | --- |
| **Views** | **Description** |
| appointmentTimeInfo | View for appointment time information. |
| staffBasicInfo | View basic staff information. |
| patientContactInfo | View patient contact information. |
| paymentSumInfo | View for payment sum-to-dateinformation. |
| patientDiagnosisInfo | View for data on patients who have received specific diagnoses and medications for a certain diagnosis. |

# Tools Used

The database was designed and implemented using MySQL Workbench. The project files, including SQL scripts for table creation, triggers, stored procedures, sample data insertion, user profiles, and main test scripts, were utilized to develop and test the database.

We are using Jupyter notebook as our programming interface and developing our code in Python Programming language. You can refer to the instruction document provided in the project folder and additionally we have provided a readme file to follow the steps to execute the project.

# References

1. https://www.w3schools.com/MySQL/default.asp
2. https://www.aad.org/public/fad/what-is-a-derm
3. <https://ieeexplore.ieee.org/abstract/document/6498195>
4. <https://www.python.org/downloads/>
5. <https://jupyter.org/install>
6. <https://dev.mysql.com/downloads/workbench/>