

Health Care Data Management System

Introduction

The main goal of this Health Care Data Management System project is to leverage data management techniques to create an efficient and reliable platform for managing patient records, appointments, and medical histories. By integrating data from hospital systems, electronic health records (EHR), and patient feedback, the system aims to streamline administrative workflows, enhance patient care, and support data-driven decision-making.

Primary Objectives

- **Efficient Patient Record Management:** Centralized storage and easy access to patient data.
- **Employee Performance Monitoring:** By analyzing the ratings given by the patients, tracking appointments handled by each employee we can monitor the employee performance and take necessary steps for better healthcare delivery.
- **Data – Driven Medical Insights:** We can identify the high – risk patients, identify the cases recorded of any disease and any operational inefficiencies.
- **Optimized Scheduling & Appointments:** Reducing waiting times and improving hospital workflow.

Challenges Addressed

- **Fragmented Patient Records:** Patient data is often spread across multiple systems/manual records, making it difficult to access a complete medical history.
- **Inefficient Scheduling:** Patients face long wait times and scheduling conflicts, leading to dissatisfaction.
- **High Operational Costs:** Redundant tests, inefficient resource utilization, and administrative overhead increase costs.

Data Pipeline

Data Source

- In this project, for analysis purposes, the data is **not collected externally** from any real-world sources. Instead, we **generated mock data** using **MOCKAROO**, a data generation tool that allows for creating structured and realistic datasets for **testing and analysis**.

The generated data captures various key aspects of a healthcare management system, including:

- **Patient & Employee Demographics** – Names, age, gender, blood group, contact details, and addresses.
- **Appointments** – Details of scheduled and canceled appointments, assigned doctors, visit dates.
- **Billing & Financial Data** – Total billing amounts, payments made, outstanding balances.
- **Diagnostic Information** – Lab test results, disease diagnoses, abnormal test values, and medical reports.
- **Ratings & Feedback** – Patient reviews and ratings for doctors, nurses, and hospital services.
- **Hospital Employee Information** – Doctors, nurses, lab assistants, and administrative staff with their respective roles.

For a comprehensive **patient analysis**, we created **1000 patient records**, which is the **maximum limit allowed by Mockaroo**. This dataset was structured and stored in a **relational database**, serving as the foundation for **query execution, data transformation, and analytics** in this project.

Data Storage

All data is stored in a **structured relational database**, ensuring efficient organization, retrieval, and processing of healthcare-related information. The **database schema** consists of multiple interconnected tables, each serving a specific function within the healthcare management system. The key tables include:

- **Patient** – Stores demographic details of patients, including **Name, Age, Gender, Weight, Height, Blood Group, and Contact Information**.
- **Appointment** – Stores appointment details, including **scheduled, completed, canceled, and no-show appointments**, along with the assigned doctor and visit date.
- **Employee** – Contains details of hospital employees, including **their roles, departments, demographic information, and employment status**.
- **Department** – Stores information about the various **departments** within the hospital, such as **Emergency, Cardiology, Neurology, Pediatrics, and Radiology**.
- **Role** – Defines different **roles** within the hospital, such as **Doctor, Nurse, Lab Technician, etc.**
- **Patient_Register** – Tracks **patient visits**, including details such as **Admitted On, Discharged On, and reason for the visit**.
- **Feedback** – Stores **ratings and reviews given by patients** for doctors, nurses, and hospital services during their visit.

- **Disease** – Maintains a **catalog of diseases**, including their **name, description, symptoms, and severity**.
- **Address** – Stores **address details** for patients and employees, ensuring proper mapping of locations within the database.
- **Lab Test** – Contains a list of **medical tests available at the hospital**, such as **Blood Tests, MRI, etc.**
- **Patient_Lab_Report** – Stores details of **lab reports**, including **test results, date of test, and flagged abnormalities**, if any.
- **Patient_Billing** – Tracks **financial transactions** related to patient visits, storing **billing details, payment status, and transaction type** (e.g., **Consultation Fee, Lab Test Fee, Surgery Costs**).
- **Patient_Disease** – Maps **patients to diagnosed diseases**, helping in identifying **chronic patients and disease trends**.

Data Transformation

Once data is ingested, it must be **cleaned, standardized, and transformed** to ensure **accuracy, consistency, and reliability** for analytical purposes. Proper transformation eliminates inconsistencies, ensures logical correctness, and prepares the data for effective **query execution and reporting**.

Key Transformations:

- **Standardizing Date Formats & Numerical Values**
 - Ensuring all dates (e.g., **Appointment Date, Admission Date, Discharge Date**) follow a **consistent format** (e.g., YYYY-MM-DD HH:MM:SS).
 - **Height, weight, age, and other numerical values** should be **realistic** (e.g., no negative values or impossible numbers).
- **Eliminating Duplicate Entries**
 - A **patient cannot have multiple appointments at the same time**—ensuring **one appointment per patient per time slot**.
 - Preventing **duplicate feedback entries** for a single visit to maintain accuracy in employee performance tracking.
- **Storing Minimum and Maximum Values for Lab Tests**
 - Each **lab test** (e.g., **Blood Sugar, Cholesterol, Hemoglobin levels**) is stored with **minimum and maximum reference values**.
 - Ensures **test result validation** by comparing **patient test values** against these thresholds to flag abnormal results.
- **Ensuring Logical Consistency in Patient Records**

- **Discharge DateTime must always be after Admission DateTime**—ensuring no inconsistencies in hospital stay durations.
- **Classifying Patients into Age Groups**
 - Patients are categorized based on **age** to enable demographic-based analysis:
 - **Children** → (0-15 years)
 - **Young Adults** → (16-35 years)
 - **Middle-Aged Adults** → (36-50 years)
 - **Senior Adults** → (51+ years)

These transformations ensure that the data is **accurate, reliable, and ready for analysis**.

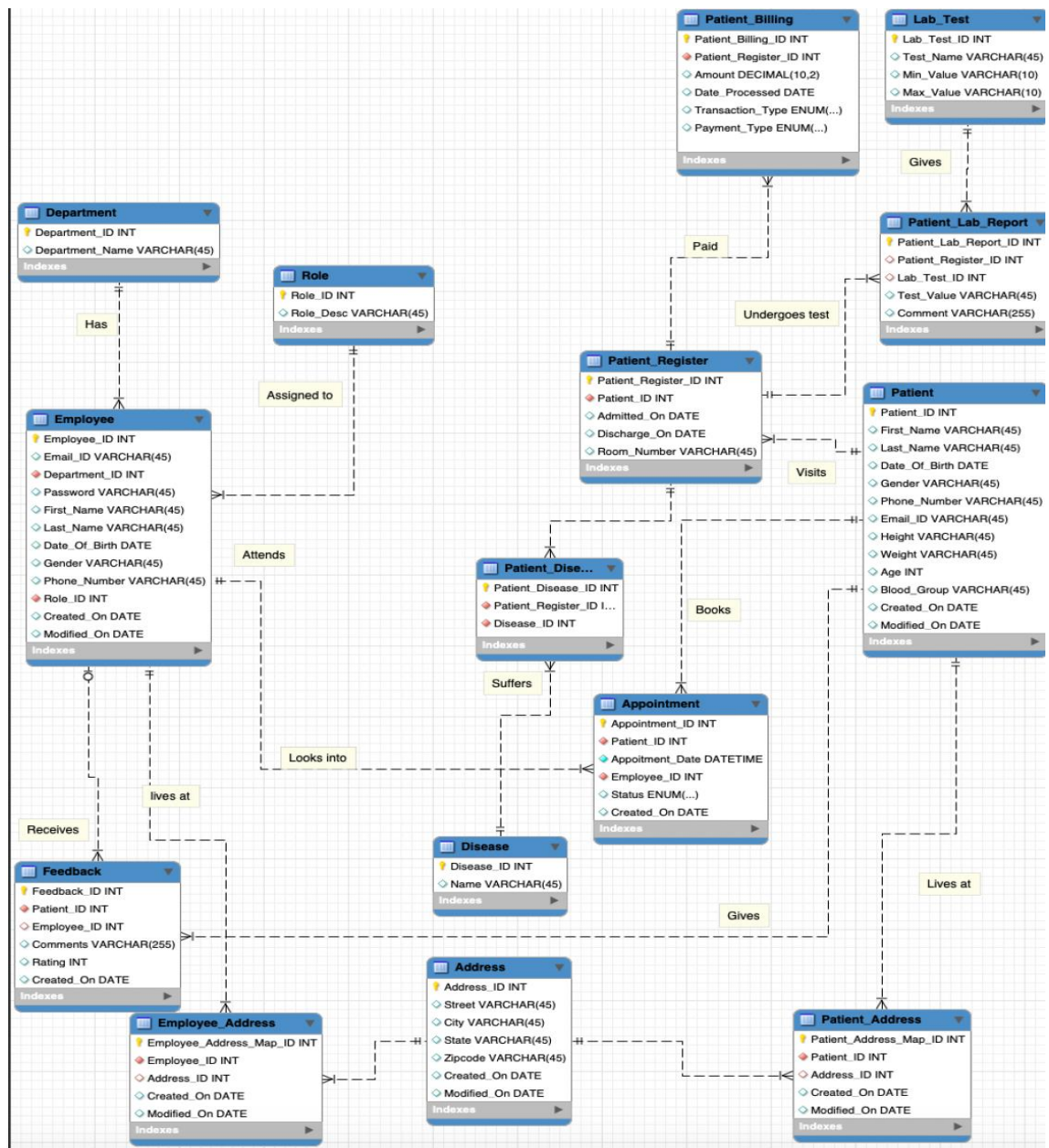
Data Model

The **data model** defines how various **entities (tables)** are **structured, related, and stored** within the relational database. It represents the logical **blueprint** of the healthcare management system.

Below are the relationships between entities.

- Address can have multiple (current, previous) address for each patient and Hospital employees.
- Each doctor / Employee may take one or more appointments.
- Each patient may book one or more appointments.
- Each patient may have more attendant (Doctor, Nurse, Lab assistant) per visit.
- Patient Register will have information of patient visit to hospital. One patient can have multiple visits.
- Feedback will be given by a patient to an employee.
- Billing will have information related to a patient visit. It can also have multiple entry depend on type of changes (Transaction Type: Appointment Bill, Lab Bill etc.)

Below is the ERD model



Query Execution & Analytics

This stage involves running **SQL queries** to generate **actionable insights** for health care data.

Query1: Patient History

Relevance

- This SQL query is designed to extract comprehensive details regarding patient appointments. It includes essential information such as patient demographics—like age, gender, and contact details—as well as specific appointment information such as date, time, and location. Additionally, the query provides insights into the healthcare professionals assigned to each appointment, including their names, roles, and specialties.
- The results are organized in descending order based on the appointment date, allowing for easy identification of the most recent appointments. By joining the Patient, Appointment, Employee, and Role tables, this query effectively consolidates all relevant data, enabling a seamless overview of who received treatment, when, and from whom.

```
SELECT
  p.Patient_ID,
  CONCAT(p.First_Name, ' ', p.Last_Name) AS Patient_Name,
  p.Gender,
  p.Age,
  p.Blood_Group,
  p.Height,
  p.Weight,
  a.Appointment_Date,
  a.Employee_ID,
  CONCAT(e.First_Name, ' ', e.Last_Name) AS Employee_Name,
  r.Role_Desc
FROM Patient p
JOIN Appointment a ON p.Patient_ID = a.Patient_ID
JOIN Employee e ON a.Employee_ID = e.Employee_ID
JOIN Role r ON e.Role_ID = r.Role_ID
WHERE p.Patient_ID = 13
ORDER BY a.Appointment_Date DESC;
```

Output:

| Patient_ID | Patient_Name | Gender | Age | Blood_Group | Height | Weight | Appointment_Date | Employee_ID | Employee_Name | Role_Desc |
|------------|--------------|--------|-----|-------------|--------|--------|---------------------|-------------|-------------------|-------------------------|
| 13 | Remus Duly | Male | 28 | B- | 76 | 131 | 2022-07-11 07:03:59 | 29 | Brittani Jaan | Surgeon |
| 13 | Remus Duly | Male | 28 | B- | 76 | 131 | 2020-01-30 13:40:46 | 216 | Cullen Scurry | Lab Technician |
| 13 | Remus Duly | Male | 28 | B- | 76 | 131 | 2019-09-01 14:30:30 | 74 | Renie Greeson | Nurse |
| 13 | Remus Duly | Male | 28 | B- | 76 | 131 | 2010-07-30 21:11:52 | 54 | Tiphani Brierton | Lab Technician |
| 13 | Remus Duly | Male | 28 | B- | 76 | 131 | 2006-07-02 20:27:40 | 276 | Conway Hennington | Patient Care Technician |

Query2: Most diagnosed diseases in the last 2 years

Relevance

- This query effectively retrieves the total number of diseases diagnosed over the past two years by performing a join operation on the Disease, Patient_Disease, and Patient_Register tables. It meticulously counts the recorded cases for each specific disease and organizes the results in descending order based on the total count of diagnoses. This approach provides a comprehensive overview of disease prevalence within the specified timeframe, highlighting the most frequently diagnosed conditions.

```

- SELECT d.Name AS Disease_Name,
-       COUNT(pd.Patient_Disease_ID) AS Diagnosed_Count
- FROM Disease d
- JOIN Patient_Disease pd ON d.Disease_ID = pd.Disease_ID
- JOIN Patient_Register pr ON pd.Patient_Register_ID = pr.Patient_Register_ID
- WHERE pr.Admitted_On >= DATE_SUB(NOW(), INTERVAL 2 YEAR)
- GROUP BY d.Name
- ORDER BY Diagnosed_Count DESC;

```

Output:

| | Disease_Name | Diagnosed_Count |
|--|----------------------|-----------------|
| | Glaucoma | 6 |
| | Hepatitis | 5 |
| | Anemia | 5 |
| | Multiple Sclerosis | 5 |
| | Influenza | 4 |
| | Depression | 4 |
| | Cholera | 4 |
| | Asthma | 4 |
| | Common Cold | 3 |
| | Cataracts | 3 |
| | Hemorrhoids | 3 |
| | Appendicitis | 3 |
| | Kidney Stones | 3 |
| | Cancer | 3 |
| | Parkinson's Disease | 3 |
| | Diabetes | 3 |
| | Bronchitis | 3 |
| | Bipolar Disorder | 3 |
| | Chickenpox | 3 |
| | Rheumatoid Arthritis | 3 |
| | HIV/AIDS | 2 |
| | Zika Virus | 2 |
| | Measles | 2 |
| | Ulcer | 2 |
| | Gout | 2 |
| | Hypothyroidism | 2 |
| | Schizophrenia | 2 |
| | Leukemia | 2 |
| | Lupus | 2 |
| | Yellow Fever | 2 |
| | Pneumonia | 2 |
| | Alzheimer's Disease | 2 |
| | Anxiety Disorder | 1 |
| | Gallstones | 1 |
| | Allergies | 1 |
| | Psoriasis | 1 |
| | Eczema | 1 |
| | Osteoporosis | 1 |
| | Arthritis | 1 |
| | SARS | 1 |
| | Lyme Disease | 1 |
| | Tuberculosis | 1 |
| | Ebola | 1 |
| | Malaria | 1 |

Query3: Patient Visits by Age Group

Relevance:

- This query organizes hospital visitors into distinct age categories, such as Children, Young Adults, Middle-Aged Adults, Senior Adults, and the Elderly, by employing a CASE statement. It effectively counts the number of unique patients within each age group by combining data from the Patient and Appointment tables. This detailed categorization helps in understanding the distribution of visitors by age, which can be valuable for resource allocation and targeted healthcare services.

```

SELECT
CASE

```

```

    WHEN p.Age BETWEEN 0 AND 15 THEN '0-15 (Children)'
    WHEN p.Age BETWEEN 16 AND 35 THEN '16-35 (Young Adults)'
    WHEN p.Age BETWEEN 36 AND 50 THEN '36-50 (Middle Aged)'
    WHEN p.Age BETWEEN 51 AND 100 THEN '51-100 (Senior Adults)'
END AS Age_Group,
COUNT(DISTINCT a.Patient_ID) AS Total_Visits
FROM Patient p
JOIN Appointment a ON a.Patient_ID = p.Patient_ID
GROUP BY Age_Group
ORDER BY Total_Visits DESC;

```

Output:

| Age_Group | Total_Visits |
|------------------------|--------------|
| 16-35 (Young Adults) | 241 |
| 0-15 (Children) | 187 |
| 36-50 (Middle Aged) | 157 |
| 51-100 (Senior Adults) | 52 |

QUERY 4: State wise cases Recorded for any specific disease

Relevance

- The "Total Diagnosed Cases for a Specific Disease in Each State" query is designed to provide a comprehensive overview of the number of patients diagnosed with a particular disease across all states. This is achieved by carefully joining four key tables: Patient, Patient Address, Disease, and Patient Disease. The query specifically filters the results to focus on a chosen disease name, such as "Diabetes," and systematically counts the total number of diagnosed cases in each state. This information is invaluable for understanding the geographic distribution of disease prevalence and can aid public health efforts and resource allocation.

Query

```

SELECT a.State,
       d.Name AS Disease,
       COUNT(pd.Patient_Disease_ID) AS Total_Cases
FROM Patient_Disease pd
JOIN Patient_Register pr ON pd.Patient_Register_ID = pr.Patient_Register_ID
JOIN Patient p ON pr.Patient_ID = p.Patient_ID
JOIN Patient_Address pa ON p.Patient_ID = pa.Patient_ID
JOIN Address a ON pa.Address_ID = a.Address_ID
JOIN Disease d ON pd.Disease_ID = d.Disease_ID
WHERE d.Name = 'Diabetes' -- Replace 'Diabetes' with any disease name
GROUP BY a.State
ORDER BY Total_Cases DESC;

```

Output

| | State | Disease | Total_Cases |
|--|----------------------|----------|-------------|
| | Texas | Diabetes | 3 |
| | Florida | Diabetes | 2 |
| | Nevada | Diabetes | 2 |
| | Illinois | Diabetes | 1 |
| | Ohio | Diabetes | 1 |
| | California | Diabetes | 1 |
| | District of Columbia | Diabetes | 1 |
| | New Jersey | Diabetes | 1 |
| | Kentucky | Diabetes | 1 |
| | Indiana | Diabetes | 1 |
| | Oklahoma | Diabetes | 1 |

Query 5: Revenue generated by each Department

Relevance

- This query is designed to extract a detailed overview of the revenue generated by each department within the healthcare facility by aggregating comprehensive patient billing data. It effectively joins multiple tables, including the Department, Employee, Appointment, Patient Register, and Patient Billing tables.
- This integration allows for a thorough analysis of which departments are the most profitable, revealing key insights into the total revenue generated, the volume of bills processed per department, and the average bill amount attributed to each patient. Such detailed insights are crucial for understanding departmental performance and improving financial strategies.

Query

```

SELECT
    d.Department_Name,
    COUNT(pb.Patient_Billing_ID) AS Total_Bills,
    SUM(pb.Amount) AS Total_Department_Revenue,
    ROUND(AVG(pb.Amount), 2) AS Avg_Bill_Per_Patient,
    (SUM(pb.Amount) * 100.0 / (SELECT SUM(Amount) FROM Patient_Billing)) AS Revenue_Percentage
FROM Department d
JOIN Employee e ON d.Department_ID = e.Department_ID
JOIN Appointment a ON e.Employee_ID = a.Employee_ID
JOIN Patient_Register pr ON a.Patient_ID = pr.Patient_ID
JOIN Patient_Billing pb ON pr.Patient_Register_ID = pb.Patient_Register_ID
GROUP BY d.Department_Name
ORDER BY Total_Department_Revenue DESC;

```

Output

| | Department_Name | Total_Bills | Total_Department_Revenue | Avg_Bill_Per_Patient | Revenue_Percentage |
|--|--------------------|-------------|--------------------------|----------------------|--------------------|
| | Cardiology | 121 | 29147.22 | 240.89 | 12.8113831 |
| | Gastroenterology | 104 | 27891.95 | 268.19 | 12.2596411 |
| | Orthopedics | 106 | 27647.69 | 260.83 | 12.1522790 |
| | Emergency Medicine | 96 | 25480.78 | 265.42 | 11.1998343 |
| | Neurology | 95 | 23737.00 | 249.86 | 10.4333724 |
| | Oncology | 90 | 21322.80 | 236.92 | 9.3722338 |
| | Urology | 80 | 20556.96 | 256.96 | 9.0356161 |
| | Dermatology | 80 | 19661.53 | 245.77 | 8.6420384 |
| | Radiology | 78 | 17667.77 | 226.51 | 7.7657001 |
| | Pediatrics | 50 | 12064.33 | 241.29 | 5.3027614 |

Query 6: Patients with the Most Visits

Relevance:

- Identifying patients with the highest hospital visits is vital for understanding healthcare utilization and managing frequent care. This analysis helps hospitals recognize individuals needing ongoing medical supervision, particularly those with chronic conditions or long-term treatments. By examining visit frequency, providers can create personalized treatment plans, improve patient engagement, and ensure timely interventions.
- Additionally, this data aids in optimizing resource allocation, reducing unnecessary admissions, and enhancing patient satisfaction through proactive care solutions. Ultimately, recognizing these patterns fosters a more efficient healthcare environment and supports better patient outcomes.

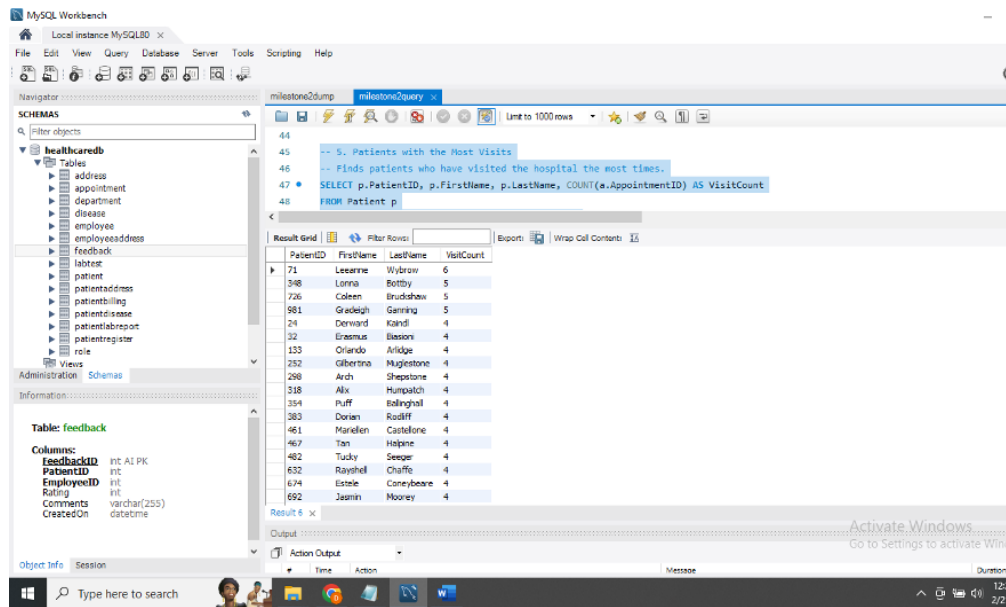
Query

```

SELECT p.PatientID, p.FirstName, p.LastName, COUNT(a.AppointmentID) AS VisitCount
FROM Patient p
JOIN Appointment a ON p.PatientID = a.PatientID
GROUP BY p.PatientID, p.FirstName, p.LastName
ORDER BY VisitCount DESC;

```

Output



Query 7: Total Appointments Per Doctor

Relevance

- Monitoring the number of appointments handled by each doctor is essential for effective hospital management. This analysis reveals how workloads are distributed, ensuring no doctor is overwhelmed while others are underutilized. It also highlights trends in patient visits across departments, allowing for informed decisions on staff allocation and scheduling.
- By examining appointment data, administrators can achieve a balanced workload, enhance operational efficiency, and prioritize the quality of patient care, ultimately improving patient satisfaction and outcomes.

Query

```
SELECT e.EmployeeID, e.FirstName, e.LastName, d.DepartmentName, COUNT(a.AppointmentID) AS TotalAppointments
FROM Employee e
JOIN Appointment a ON e.EmployeeID = a.EmployeeID
JOIN Department d ON e.DepartmentID = d.DepartmentID
-- Filter only doctors
WHERE e.RoleID = 2
GROUP BY e.EmployeeID, e.FirstName, e.LastName, d.DepartmentName
ORDER BY TotalAppointments DESC;
```

Output

The screenshot shows the MySQL Workbench interface. On the left, the 'SCHEMAS' pane displays the 'healthcaredb' database with various tables like 'address', 'appointment', 'department', etc. The main editor window shows a SQL query in the 'milestone2query' tab. The query is as follows:

```

-- 1. Total Appointments Per Doctor
-- This query counts how many appointments each doctor has handled.
SELECT e.EmployeeID, e.FirstName, e.LastName, d.DepartmentName, COUNT(a.AppointmentID) AS TotalAppointments
FROM Employee e
JOIN Appointment a ON e.EmployeeID = a.EmployeeID

```

The 'Result Grid' shows the output of this query, listing doctors and their total appointments:

| EmployeeID | FirstName | LastName | DepartmentName | TotalAppointments |
|------------|-----------|------------|---------------------------|-------------------|
| 378 | Sholan | Antoons | Psychiatry | 4 |
| 41 | Marybelle | de Verson | Emergency Department(ED) | 3 |
| 328 | Thelma | Crichton | Ophthalmology | 3 |
| 394 | Wally | Keddey | Ophthalmology | 3 |
| 389 | Hazel | Boutfour | Anesthesiology | 3 |
| 817 | Debera | Stinton | Gastroenterology | 3 |
| 926 | Elinor | Margaram | Infectious Diseases | 3 |
| 987 | Mariabel | Rowena | Emergency Department(ED) | 3 |
| 167 | Irene | Kennet | Pediatrics | 2 |
| 51 | Hart | Imason | Otorhinolaryngology (ENT) | 2 |
| 58 | Stu | Gimlet | Ophthalmology | 2 |
| 69 | Anastole | Stearndale | Radiology | 2 |
| 99 | Micki | Brighting | Psychiatry | 2 |
| 124 | Sabrina | Dauney | Intensive Care Unit (ICU) | 2 |
| 125 | Garvin | Marchant | Neurology | 2 |

The bottom status bar indicates that 610 rows were returned in 0.063 seconds.

Query 8: List of Canceled Appointments

Relevance

- Tracking canceled appointments is vital for improving hospital scheduling and patient care efficiency. This query reveals canceled appointments, detailing the involved patients and doctors. By analyzing this data, hospital administrators can identify trends, such as peak cancellation times or specific departments with higher rates.
- Understanding the reasons behind cancellations allows for targeted strategies, like automated reminders, flexible rescheduling, and better patient communication. Reducing cancellations enhances resource utilization, minimizes doctor downtime, and ensures that appointment slots are efficiently filled, ultimately improving patient satisfaction and healthcare delivery.

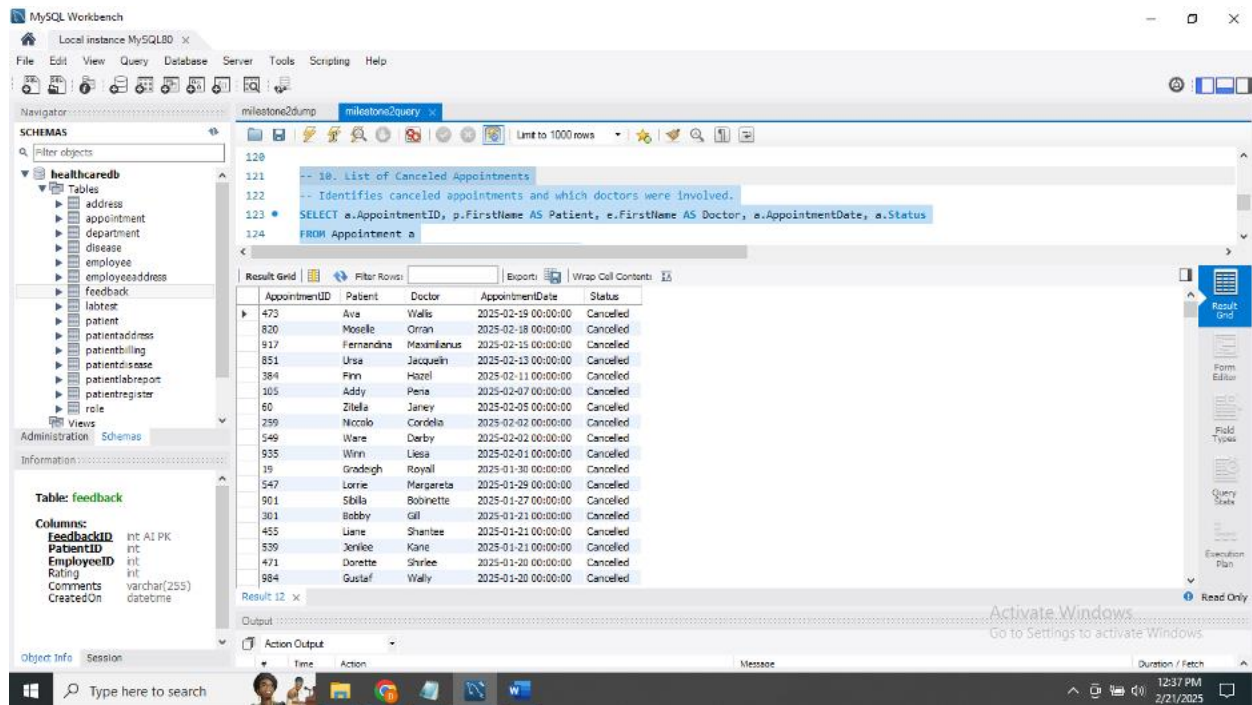
Query

```

SELECT a.AppointmentID, p.FirstName AS Patient, e.FirstName AS Doctor, a.AppointmentDate, a.Status
FROM Appointment a
JOIN Patient p ON a.PatientID = p.PatientID
JOIN Employee e ON a.EmployeeID = e.EmployeeID
WHERE a.Status = 'Cancelled'
ORDER BY a.AppointmentDate DESC;

```

Output



Query 9: Most Expensive Treatments Given

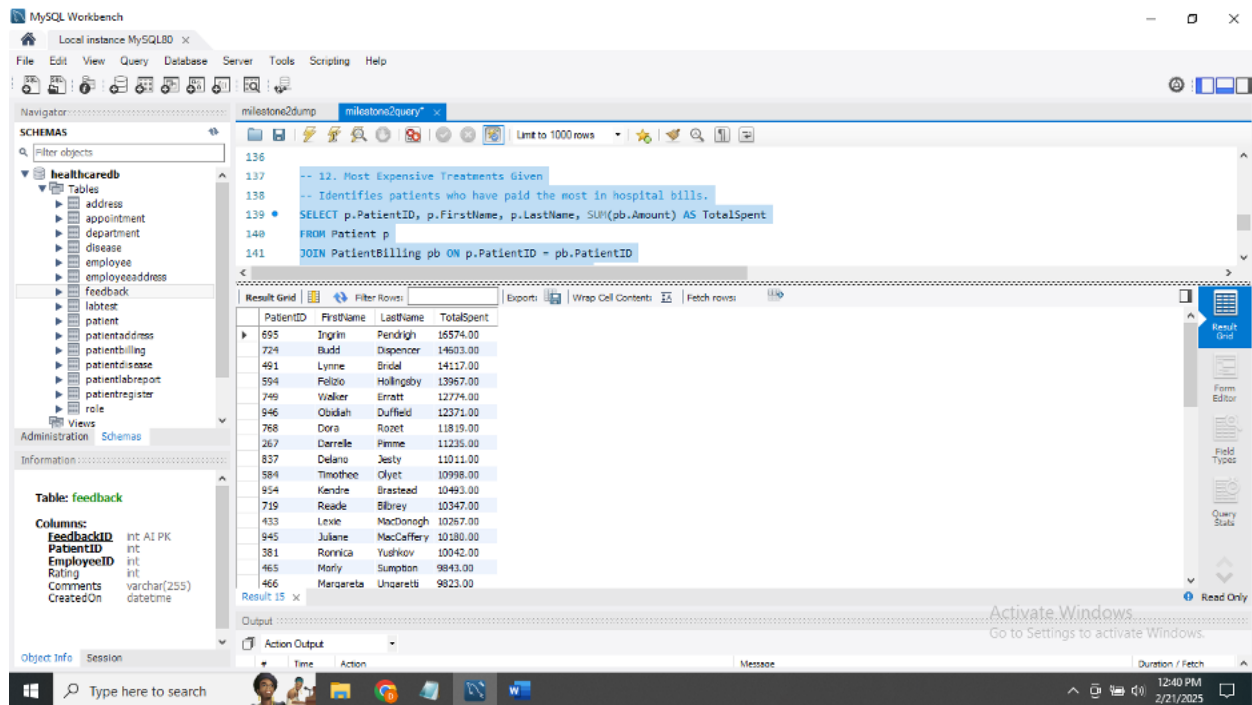
Relevance

- Identifying the most expensive treatments provided by the hospital helps in financial planning, resource allocation, and assessing healthcare affordability. This query retrieves a list of patients who have incurred the highest medical expenses, allowing administrators to analyze the costliest procedures and treatments. Understanding these trends helps hospitals evaluate pricing structures, negotiate better rates with suppliers, and ensure fair billing practices. Additionally, this information can be used to assess the financial burden on patients and explore possible solutions such as insurance coverage options, financial aid programs, or cost-effective treatment alternatives to improve accessibility to quality healthcare.

Query

```
SELECT p.PatientID, p.FirstName, p.LastName, SUM(pb.Amount) AS TotalSpent
FROM Patient p
JOIN PatientBilling pb ON p.PatientID = pb.PatientID
GROUP BY p.PatientID, p.FirstName, p.LastName
ORDER BY TotalSpent DESC
LIMIT 50;
```

Output



Query 10: Percentage of Appointments Completed vs Canceled

Relevance

- Tracking the percentage of completed versus canceled appointments is crucial for evaluating hospital efficiency and patient commitment. This query provides a clear measure of how many scheduled appointments were successfully completed and how many were canceled, helping administrators identify potential scheduling issues. By excluding scheduled appointments, the calculation focuses only on appointments that have reached an outcome. A high cancellation rate may indicate issues such as poor appointment scheduling, long wait times, or patient dissatisfaction, which could be addressed by implementing automated reminders, better rescheduling options, or improved patient communication. Monitoring this metric enables hospital management to take proactive steps in optimizing appointment availability, reducing cancellations, and ensuring effective utilization of medical resources.

Query

```

SELECT
    COUNT(CASE WHEN Status = 'Completed' THEN 1 END) AS CompletedAppointments,
    COUNT(CASE WHEN Status = 'Cancelled' THEN 1 END) AS CanceledAppointments,
    (COUNT(CASE WHEN Status = 'Completed' THEN 1 END) /
     COUNT(CASE WHEN Status IN ('Completed', 'Cancelled') THEN 1 END)) * 100 AS CompletionRate
FROM Appointment;

```

Output

The screenshot shows the MySQL Workbench interface. The 'Query' tab is active, displaying a SQL query that calculates the completion rate of appointments. The 'Result Grid' shows the output of the query, which includes three columns: CompletedAppointments, CanceledAppointments, and CompletionRate. The output shows 303 completed appointments, 357 canceled appointments, and a completion rate of 45.9091.

| CompletedAppointments | CanceledAppointments | CompletionRate |
|-----------------------|----------------------|----------------|
| 303 | 357 | 45.9091 |

Stored Procedure

Stored Procedure 1: Retrieve Employee Ratings and Performance Insights

- This stored procedure, named `Get_Employee_Rating`, dynamically calculates an employee's average rating, total feedback received, and appointment completion rate based on their `Employee_ID`. By joining the `Employee`, `Feedback`, and `Appointment` tables, it evaluates the performance of doctors or staff members based on patient feedback.
- Additionally, this stored procedure aids in identifying top-performing employees by analyzing feedback ratings and completion rates.

```

DELIMITER $$

CREATE PROCEDURE Get_Employee_Rating(IN employee_id INT)
BEGIN
    SELECT e.Employee_ID,
           CONCAT(e.First_Name, ' ', e.Last_Name) AS Employee_Name,
           d.Department_Name,
           COUNT(f.Feedback_ID) AS Total_Feedbacks,

```



```

-      IFNULL(ROUND(AVG(f.Rating), 2), 'No Ratings') AS Avg_Rating,
-      (SELECT COUNT(*)
-      FROM Appointment a
-      WHERE a.Employee_ID = employee_id
-      AND a.Status = 'Completed') AS Completed_Appointments,
-      (SELECT COUNT(*)
-      FROM Appointment a
-      WHERE a.Employee_ID = employee_id) AS Total_Appointments,
-      IFNULL(ROUND(
-      (SELECT COUNT(*)
-      FROM Appointment a
-      WHERE a.Employee_ID = employee_id
-      AND a.Status = 'Completed')
-      * 100.0 /
-      (SELECT COUNT(*)
-      FROM Appointment a
-      WHERE a.Employee_ID = employee_id), 2), 'No Appointments')
-      AS Completion_Rate
-      FROM Employee e
-      JOIN Department d ON e.Department_ID = d.Department_ID
-      LEFT JOIN Feedback f ON e.Employee_ID = f.Employee_ID
-      WHERE e.Employee_ID = employee_id
-      GROUP BY e.Employee_ID, d.Department_Name;
-      END $$
-
-      DELIMITER ;

```

Procedure Invocation

CALL Get_Employee_Rating(90);

Output

| Employee_ID | Employee_Name | Department_Na... | Total_Feedbacks | Avg_Rating | Completed_Appointme... | Total_Appointments | Completion_Rate |
|-------------|----------------|------------------|-----------------|------------|------------------------|--------------------|-----------------|
| 90 | Marijn Jickles | Gastroenterology | 4 | 1.75 | 2 | 4 | 50.00 |

Stored Procedure 2: Analyze Patient Diseases and Abnormal Lab Results

- This stored procedure (Check_Patient_Health) evaluates a patient's health status by checking the number of diseases diagnosed and identifying abnormal lab test results. It first classifies the patient as "Chronic" if they have more than three diagnosed diseases. Then, it scans the Patient Lab Reports table to count abnormal test values. By joining the Patient, Disease, Lab Test, and Patient Lab Report tables, this procedure helps doctors monitor high-risk.
- This procedure helps to monitor the patient and risk management particularly for the chronic patients.

Query

```

DELIMITER $$

DELIMITER $$
DROP PROCEDURE IF EXISTS Check_Patient_Health;

```



```

CREATE PROCEDURE Check_Patient_Health(IN patient_id INT)
BEGIN
    DECLARE disease_count INT DEFAULT 0;
    DECLARE chronic_status VARCHAR(20);
    DECLARE abnormal_tests INT DEFAULT 0;

    -- Count number of diseases a patient has
    SELECT COUNT(pd.Disease_ID) INTO disease_count
    FROM Patient_Register pr
    JOIN Patient_Disease pd ON pr.Patient_Register_ID = pd.Patient_Register_ID
    WHERE pr.Patient_ID = patient_id;

    -- Determine if the patient is chronic
    IF disease_count > 3 THEN
        SET chronic_status = 'Chronic Patient';
    ELSE
        SET chronic_status = 'Non-Chronic';
    END IF;

    -- Count abnormal lab tests
    SELECT COUNT(*) INTO abnormal_tests
    FROM Patient_Lab_Report plr
    JOIN Lab_Test lt ON plr.Lab_Test_ID = lt.Lab_Test_ID
    JOIN Patient_Register pr ON plr.Patient_Register_ID = pr.Patient_Register_ID
    WHERE pr.Patient_ID = patient_id
    AND ((CAST(plr.Test_Value AS DECIMAL(10,2)) - CAST(lt.Min_Value AS DECIMAL(10,2)) < 0.5)
        OR (CAST(plr.Test_Value AS DECIMAL(10,2)) - CAST(lt.Max_Value AS DECIMAL(10,2)) < 0.5));

    SELECT p.Patient_ID,
        CONCAT(p.First_Name, ' ', p.Last_Name) AS Patient_Name,
        disease_count AS Total_Diseases,
        chronic_status AS Health_Status,
        abnormal_tests AS Abnormal_Lab_Test_Count
    FROM Patient p
    WHERE p.Patient_ID = patient_id;

END $$

DELIMITER ;

```

Procedure Invocation:

CALL Check_Patient_Health(974);

Output:

| | Patient_ID | Patient_Name | Total_Diseases | Health_Status | Abnormal_Lab_Test_Count |
|--|------------|-----------------|----------------|-----------------|-------------------------|
| | 974 | Minny MacGaffey | 5 | Chronic Patient | 3 |

Data Presentation Via Python API (TkinterGUI)

- The Healthcare Database Query System is a graphical user interface (GUI) application developed using Python and Tkinter. It allows users to query a healthcare database hosted on MySQL and view the results in a tabular format with support for horizontal and

vertical scrolling. The application includes error handling for user inputs and database connections.

Features

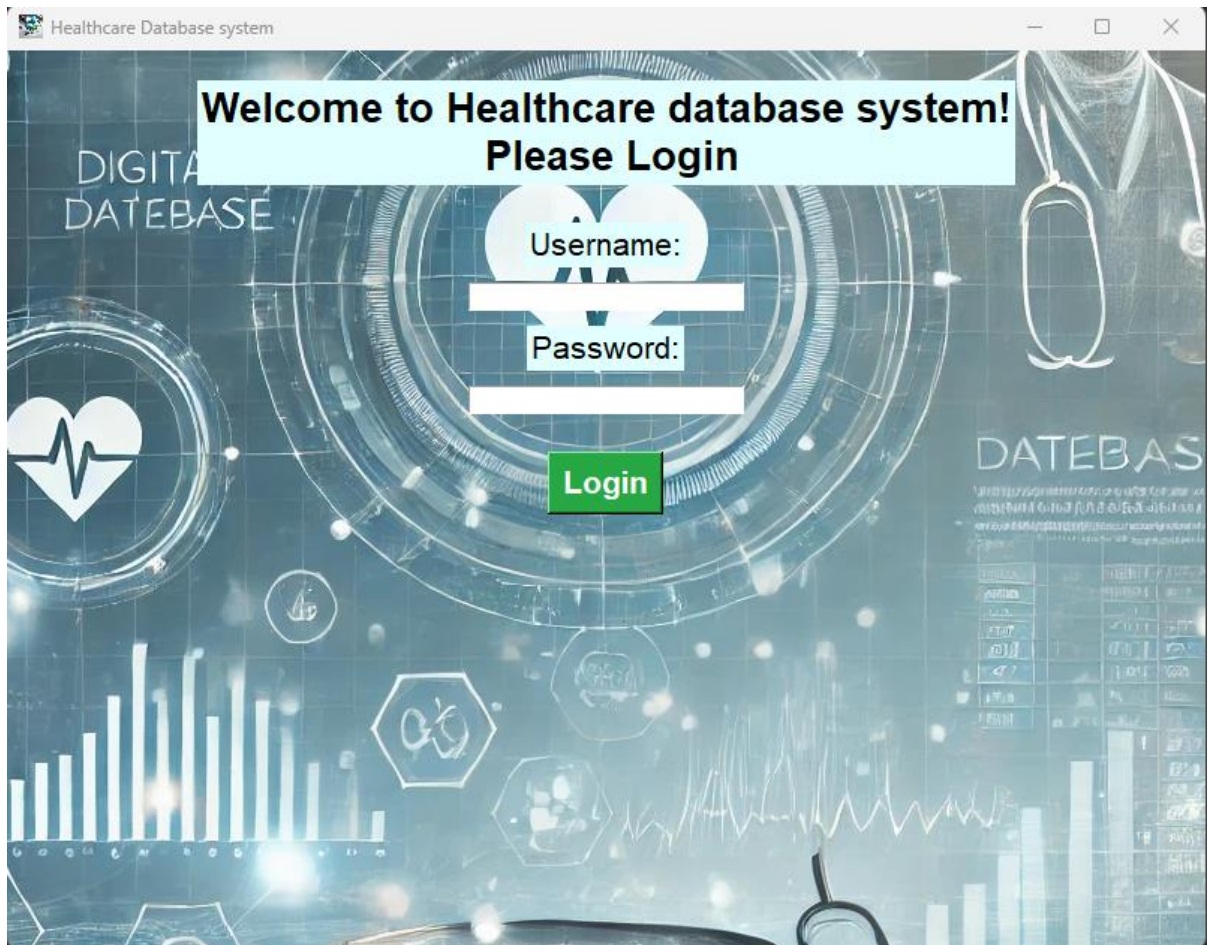
- Connect to a MySQL healthcare database securely. The `'database_mysql_connection'` function securely connects to the MySQL database.
- Execute pre-defined queries with parameter input from the user. The `'execution_query'` function runs the user-selected query and handles any errors encountered during execution. A combination of labels, entry fields, and dropdown menus allows the user to input parameters and select queries.
- Display query results in a table.
- Handle input validation and display error messages when needed. A `'Treeview'` widget shows query results in a scrollable format.
- Easy-to-use graphical interface with dropdown selection and input fields.
- Included the authentication where a user can authenticate if and only if the credentials provided matches with the database values.

Prerequisites

- Python 3.x
- Required Libraries: *pip install mysql-connector-python, pip install tkinter*
- Connect to Database via credentials (mentioned in code file)
- To run: Open the Milestone3 folder in python software or VS and open terminal. Then type “python milestone3.py”. Click enter.

Application structure

- **MySQL Connection Function:** Establishes a connection to the database with error handling.
- **Query Execution Function:** Executes the user-selected query while passing the input parameters securely. Also handles any errors encountered during query execution and informs the user.
- **Results Display Function:** Displays the query results in a Treeview widget with a structured format. Clears previous results before updating the table with new results.
- **GUI Setup:** Built the main application window with dropdown menus, input fields, and a results display area.



CODE

The code consists of two main parts:

1. **User Authentication:** Allows users to log in with their email and password.
2. **Healthcare Query System:** Once logged in, users can execute predefined SQL queries and view the results in a structured format.

```
import mysql.connector
from mysql.connector import Error
import tkinter as tk
from tkinter import *
from tkinter import ttk, messagebox
```

- `mysql.connector` Used to connect and interact with a MySQL database.
- `tkinter` A standard GUI toolkit in Python for creating desktop applications.
- `ttk`: Provides themed widgets for Tkinter.

- `messagebox`: Used to show dialog boxes with messages, such as errors or alerts.

```
# Function to establish MySQL conn
def database_mysql_connection():
    try:
        conn = mysql.connector.connect(
            host="cssql.seattleu.edu",
            user="am_dpadala",
            password="KLsijv1KlJGN+PH0",
            database="am_dpadala"
        )
        if conn.is_connected():
            print("Connected to MySQL Database")
        return conn
    except Error as e:
        messagebox.showerror("Connection Error", f"Error connecting to MySQL: {e}")
    return None
```

This function attempts to establish a connection to the MySQL database using the provided credentials. If successful, it returns the connection object; otherwise, it shows an error message.

```
def authenticate_user(Email_ID, Password):
    allowed_roles = {2, 3, 6} #allowed employees based on their role i.e.,
    Doctor, Surgeon, Patient Care Technician
    conn = database_mysql_connection()
    if not conn:
        return False
    try:
        cursor = conn.cursor()
        # Fetch the Role_ID along with the email and password
        cursor.execute("SELECT Role_ID FROM Employee WHERE Email_ID = %s AND
Password = %s", (Email_ID, Password))
        user = cursor.fetchone()
        cursor.close()

        if user is not None:
            role_id = user[0] # Assuming Role_ID is the first column in the
result
            return role_id in allowed_roles # Checking if the role_id is in the
allowed roles
        else:
            return False # Invalid credentials
    except Error as e:
        messagebox.showerror("Query Error", f"Error executing query: {e}")
    return False
```

```
finally:
    conn.close() # Ensure the connection is closed in all cases
```

- This function checks if the user's email and password match an entry in the database. If the user exists and their role is allowed, it returns True. Otherwise, it returns False.
- It also verifies if the user has the necessary role to access the system.
- The result of the query is retrieved using `fetchone()`, which returns the first matching row or None if no match is found.

```
def healthcare_query_page():
    login_page.destroy()
```

This function is called upon successful login to transition from the login page to the query execution interface.

```
# This Function will execute queries and display results
def execution_query():
    key = var.get() #getting the selected queries from the dropdown box

    if key not in queries:
        messagebox.showwarning("Input Error", "Please select a valid query.")
        return

    query = queries[key] #extracting SQL queries from dictionary
    params = ()

    # Handling parameterized queries or queries that need user input
    if key == "Patient History":
        patient_id = userEntryParameter.get()
        if not patient_id.isdigit():
            messagebox.showerror("Input Error", "Please enter a valid Patient
ID.")
            return
        params = (patient_id,)#given patient ID as parameter

    elif key == "State-wise Cases for a Disease":
        disease_name = userEntryParameter.get()
        if not disease_name:
            messagebox.showerror("Input Error", "Please enter a Disease
Name.")
            return
        params = (disease_name,) # disease name as parmeter

    elif key == "Total Appointments Per Doctor":
        Employee_ID = userEntryParameter.get()
```

```

        if not Employee_ID.isdigit():
            messagebox.showerror("Input Error", "Please enter a valid Doctor
ID.")
            return
        params = (Employee_ID,)

    elif key == "List of Appointments by Status":
        status = userEntryParameter.get().strip().lower()
        if status not in ["completed", "cancelled", "scheduled"]:
            messagebox.showerror("Input Error", "Please enter 'Canceled' as
status.")
            return
        params = (status,)

    elif key == "Patients by Blood Group":
        blood_group = userEntryParameter.get().strip().upper()
        valid_blood_groups = ['A+', 'A-', 'B+', 'B-', 'AB+', 'AB-', 'O+', 'O-
']
        if blood_group not in valid_blood_groups:
            messagebox.showerror("Input Error", "Please enter a valid Blood
Group (e.g., A+, O-).")
            return
        params = (blood_group,)

    conn = database_mysql_connection()
    if not conn:
        return

    try:
        cursor = conn.cursor(dictionary=True) #showing column names in output
        cursor.execute(query, params)
        results = cursor.fetchall()
        cursor.close()
        conn.close()

        if results:
            display_results(results) #Diplaying output in GUI
        else:
            messagebox.showinfo("No Data", "No results found for this
query.")
    except Error as e:
        messagebox.showerror("Query Error", f"Error executing query: {e}")

```

This function retrieves the selected query from a dropdown, checks for required parameters, executes the query, and displays the results. It handles specific queries that require user input for parameters, such as patient IDs or disease names.

```
# This Function will display query results in Tkinter
def display_results(results):
    for widget in frame_results.winfo_children():
        widget.destroy() # clears old data before showing new results

    if not results:
        return

    columns = list(results[0].keys()) #getting the column names
    frame = tk.Frame(frame_results)
    frame.pack(expand=True, fill="both")
    tree = ttk.Treeview(frame, columns=columns, show="headings")# creating a
table Treeview to display query results.
    '''A treeview widget displays a hierarchy of items and allows users to
browse through it.
    One or more attributes of each item can be displayed as columns to
the right of the tree.'''
    h_scroll = ttk.Scrollbar(frame, orient="horizontal", command=tree.xview)
    v_scroll = ttk.Scrollbar(frame, orient="vertical", command=tree.yview)
    tree.configure(xscrollcommand=h_scroll.set, yscrollcommand=v_scroll.set)

    for col in columns:
        tree.heading(col, text=col) #setting column headers
        tree.column(col, width=120, anchor="center") #setting column width

    for row in results:
        tree.insert("", "end", values=list(row.values())) #Inserting the data

    tree.pack(side="left", fill="both", expand=True)
    v_scroll.pack(side="right", fill="y")
    h_scroll.pack(side="bottom", fill="x")
```

This function takes the query results and populates a Treeview widget to present the data in a table format. It clears previous results before displaying new ones.

```
# Healthcare Database GUI Setup
root = tk.Tk()
root.title("Healthcare Database - Group 4")
```

```

root.geometry("1000x700") #window size
# Load background image
background = PhotoImage(file="logo.png")

# Show image using label and make it fill the entire window
label_bg = tk.Label(root, image=background)
label_bg.place(relwidth=1, relheight=1)

# Header Label
header_label = tk.Label(root, text="Healthcare Database Query System",
font=("Arial", 24, "bold"), bg="#F0F2F5", fg="#333")
header_label.pack(pady=10)

```

A new Tkinter window (**root**) is created. Initializes the main application window for the healthcare query interface. Loads a background image and adds a header label for the application.

```

# Main Frame
frame_top = tk.Frame(root, bg="ffffff", relief=tk.GROOVE, borderwidth=2,
pady=10, padx=20)
frame_top.pack(pady=20, padx=10, fill="x") #selecting the queries

# Frame for displaying the result of the query
frame_results = tk.Frame(root, bg="#f4f4f4", relief=tk.GROOVE) #displaying
the result of that query
frame_results.pack(expand=True, fill="both", padx=20, pady=10)

# Creating a Dropdown menu for Query Selection
var = tk.StringVar()
labelQuery = tk.Label(frame_top, text="Select Query:", font=("Arial", 14),
bg="ffffff")
labelQuery.grid(row=0, column=0, padx=10, pady=20)

query_options = [
    "Patient History",
    "Most Diagnosed Diseases in Last 2 Years",
    "Patient Visits by Age Group",
    "State-wise Cases for a Disease",

```



```

        "Revenue Generated by Each Department",
        "Patients with the Most Visits",
        "Total Appointments Per Doctor",
        "List of Appointments by Status",
        "Patients by Blood Group"
    ]
    queryBox = ttk.Combobox(frame_top, textvariable=var, values=query_options,
state="readonly", width=40)
    queryBox.grid(row=0, column=1, padx=10)

    # Adding Entry field/ Input box for for users to enter values
    userEntryParameter = tk.Entry(frame_top, font=("Arial", 12), width=25,
relief=tk.SOLID, borderwidth=1)
    userEntryParameter.grid(row=0, column=2, padx=10, pady=10)

    # Run query Button
    runButton = tk.Button(frame_top, text="Run Query", command=execution_query,
font=("Arial", 12, "bold"), bg="#28A745", fg="#FFF", relief=tk.FLAT, padx=10)
    runButton.grid(row=0, column=3, padx=10, pady=10)

```

This section of the code sets up a user interface for selecting and executing queries in a healthcare database application. It includes a frame for organizing the query selection elements, a dropdown for choosing queries, an entry box for parameters, and a button to run the selected query. This structure facilitates user interaction with the database, allowing for seamless data retrieval and display of results.

- A new frame (frame_top) is created within the main application window (root).
- Another frame (frame_results) is created to display the results of the executed queries.
- A StringVar (var) is created to hold the value selected from the dropdown.
- A list of options (query_options) is defined, representing various SQL queries that users can execute.
- A Combobox (queryBox) is created to allow users to select one of the predefined queries.
- A button (runButton) is created to execute the selected query when clicked.

```

# SQL Queries Dictionary
queries = {
    "Patient History": """
        SELECT
            p.Patient_ID,
            CONCAT(p.First_Name, ' ', p.Last_Name) AS Patient_Name,
            p.Gender, p.Age, p.Blood_Group, p.Height, p.Weight,
            a.Appointment_Date, a.Employee_ID,
            CONCAT(e.First_Name, ' ', e.Last_Name) AS Employee_Name,

```

```

        r.Role_Desc
    FROM Patient p
    JOIN Appointment a ON p.Patient_ID = a.Patient_ID
    JOIN Employee e ON a.Employee_ID = e.Employee_ID
    JOIN Role r ON e.Role_ID = r.Role_ID
    WHERE p.Patient_ID = %s
    ORDER BY a.Appointment_Date DESC;
    """

    "Most Diagnosed Diseases in Last 2 Years": """
        SELECT d.Name AS Disease_Name,
               COUNT(pd.Patient_Disease_ID) AS Diagnosed_Count
        FROM Disease d
        JOIN Patient_Disease pd ON d.Disease_ID = pd.Disease_ID
        JOIN Patient_Register pr ON pd.Patient_Register_ID =
pr.Patient_Register_ID
        WHERE pr.Admitted_On >= DATE_SUB(NOW(), INTERVAL 2 YEAR)
        GROUP BY d.Name
        ORDER BY Diagnosed_Count DESC;
    """

    "Patient Visits by Age Group": """
        SELECT
            CASE
                WHEN p.Age BETWEEN 0 AND 15 THEN '0-15 (Children)'
                WHEN p.Age BETWEEN 16 AND 35 THEN '16-35 (Young Adults)'
                WHEN p.Age BETWEEN 36 AND 50 THEN '36-50 (Middle Aged)'
                WHEN p.Age BETWEEN 51 AND 100 THEN '51-100 (Senior Adults)'
            END AS Age_Group,
            COUNT(DISTINCT a.Patient_ID) AS Total_Visits
        FROM Patient p
        JOIN Appointment a ON a.Patient_ID = p.Patient_ID
        GROUP BY Age_Group
        ORDER BY Total_Visits DESC;
    """

    "State-wise Cases for a Disease": """
        SELECT a.State, d.Name AS Disease, COUNT(pd.Patient_Disease_ID) AS
Total_Cases
        FROM Patient_Disease pd
        JOIN Patient_Register pr ON pd.Patient_Register_ID =
pr.Patient_Register_ID
        JOIN Patient p ON pr.Patient_ID = p.Patient_ID

```

```

        JOIN Patient_Address pa ON p.Patient_ID = pa.Patient_ID
        JOIN Address a ON pa.Address_ID = a.Address_ID
        JOIN Disease d ON pd.Disease_ID = d.Disease_ID
        WHERE d.Name = %s
        GROUP BY a.State
        ORDER BY Total_Cases DESC;
    """

    "Revenue Generated by Each Department": """
        SELECT
            d.Department_Name,
            COUNT(pb.Patient_Billing_ID) AS Total_Bills,
            SUM(pb.Amount) AS Total_Department_Revenue
        FROM Department d
        JOIN Employee e ON d.Department_ID = e.Department_ID
        JOIN Appointment a ON e.Employee_ID = a.Employee_ID
        JOIN Patient_Register pr ON a.Patient_ID = pr.Patient_ID
        JOIN Patient_Billing pb ON pr.Patient_Register_ID =
pb.Patient_Register_ID
        GROUP BY d.Department_Name
        ORDER BY Total_Department_Revenue DESC;
    """

    "Patients with the Most Visits": """
        SELECT p.Patient_ID, CONCAT(p.First_Name, ' ', p.Last_Name) AS
Patient_Name, COUNT(a.Appointment_ID) AS Visit_Count
        FROM Patient p
        JOIN Appointment a ON p.Patient_ID = a.Patient_ID
        GROUP BY p.Patient_ID
        ORDER BY Visit_Count DESC
        LIMIT 10;
    """

    "Total Appointments Per Doctor": """
        SELECT
            e.Employee_ID,
            CONCAT(e.First_Name, ' ', e.Last_Name) AS Doctor_Name,
            COUNT(a.Appointment_ID) AS Total_Appointments
        FROM Employee e
        JOIN Appointment a ON e.Employee_ID = a.Employee_ID
        WHERE e.Employee_ID= %s
        GROUP BY e.Employee_ID, Doctor_Name
        ORDER BY Total_Appointments DESC;
    """

```

```

"""
"List of Appointments by Status":
"""
    SELECT
        a.Appointment_ID,
        CONCAT(p.First_Name, ' ', p.Last_Name) AS Patient_Name,
        CONCAT(e.First_Name, ' ', e.Last_Name) AS Doctor_Name,
        a.Appointment_Date,
        a.Status
    FROM Appointment a
    JOIN Patient p ON a.Patient_ID = p.Patient_ID
    JOIN Employee e ON a.Employee_ID = e.Employee_ID
    WHERE a.Status = %s
    ORDER BY a.Appointment_Date DESC;
"""
,
"Patients by Blood Group":"""
    SELECT
        p.Blood_Group,
        COUNT(p.Patient_ID) AS Patient_Count
    FROM Patient p
    WHERE p.Blood_Group = %s
    GROUP BY p.Blood_Group
    ORDER BY Patient_Count DESC;
"""

}
# Run Tkinter Application by keeping the window open
root.mainloop()

```

Stores various SQL queries in a dictionary for easy access based on user selection.

```

# Function to handle login button click
def on_login():
    Email_ID = entry_username.get()
    Password = entry_password.get()

    if authenticate_user(Email_ID, Password):
        messagebox.showinfo("Login Success", "You are successfully logged in!")
        healthcare_query_page()
    else:
        messagebox.showerror("Login Failed", "Invalid username or password.")

# Creating a Login GUI Setup for authorized users only
login_page = tk.Tk()
login_page.title("Healthcare Database system")

```

```

login_page.geometry("800x600")# window size
icon = tk.PhotoImage(file="logo.png")
login_page.iconphoto(False, icon)

login_page.configure(bg="#E0FFFF")
background= PhotoImage(file="healthcare_bg.png")

# Showing image using label and make it fill the entire window
label_bg = tk.Label(login_page, image=background)
label_bg.place(relwidth=1, relheight=1) # Fill the entire window

# Header Label
header_label = tk.Label(login_page, text="Welcome to Healthcare database
system!\n Please Login", font=("Arial", 20, "bold"), bg="#E0FFFF")
header_label.pack(pady=20)

# Username and Password Labels and Entries
label_username = tk.Label(login_page, text="Email ID:", bg="#E0FFFF",
font=("Arial", 16))
label_username.pack(pady=5)

entry_username = tk.Entry(login_page, width=30, relief=tk.GROOVE )
entry_username.pack(pady=5)

label_password = tk.Label(login_page, text="Password:",bg="#E0FFFF",
font=("Arial", 16))
label_password.pack(pady=5)

entry_password = tk.Entry(login_page, show='*', width=30, relief=tk.GROOVE )
entry_password.pack(pady=5)

# Login Button
login_button = tk.Button(login_page, text="Login", command=on_login,
font=("Arial", 16, "bold"), bg="#28A745", fg="#FFF", relief=tk.RAISED )
login_button.pack(pady=20)

# Start the login window
login_page.mainloop()

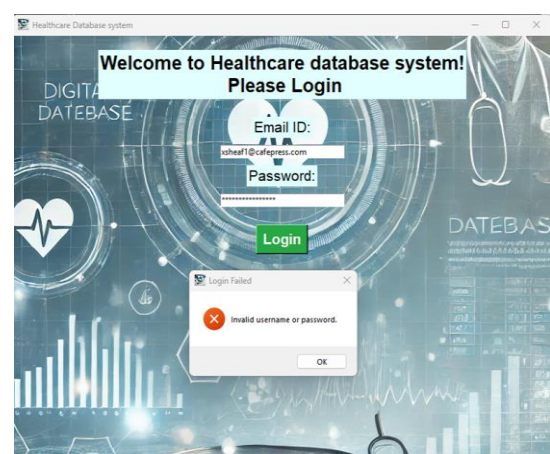
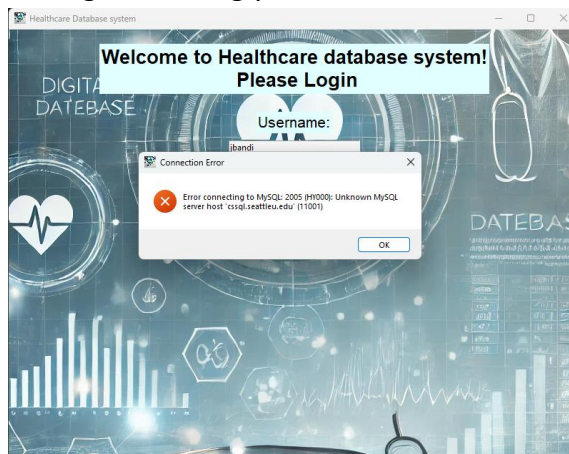
```

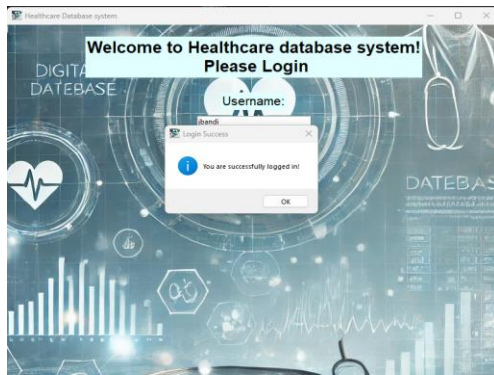
- This function is triggered when the user clicks the "Login" button. It retrieves the email and password entered by the user using the get() method on the respective entry fields.
- The authenticate_user function is called with the provided email and password:

- If authentication is successful, a success message box is displayed, and the user is directed to the healthcare query page (healthcare_query_page()).
- If authentication fails, an error message box informs the user about invalid credentials.
- A new Tkinter window is initialized for the login page, titled "Healthcare Database system." prompts for the "Email ID," and an entry field allows the user to input their email. Similarly, a label prompts for the "Password," and an entry field is created, with the show='*' attribute to mask the password input.
- The mainloop() method starts the Tkinter event loop, allowing the application to wait for user interactions, such as button clicks.
- Establishes a comprehensive login interface for users of the healthcare database system. It includes input fields for email and password, a visually appealing layout with background images, and effective feedback mechanisms for login success or failure. The design emphasizes usability while ensuring that the application is visually engaging and user-friendly.

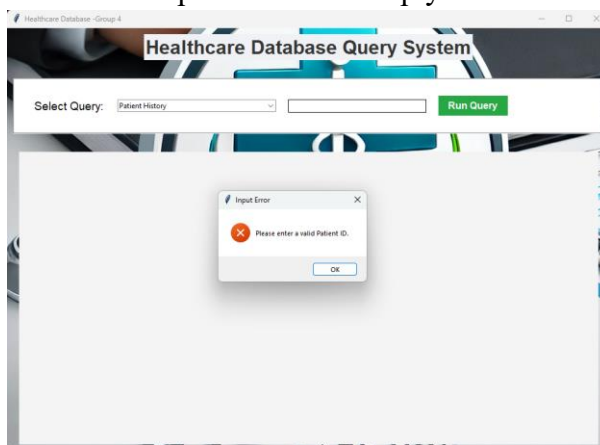
Error Handling

- **Database Connection Errors:** Alerts the user if the connection to the database fails. The User is able to logging to the Healthcare database by valid credentials and authorized roles.
- Similarly, The user will not be able to login if the user enters invalid credentials or if that users are not authorized and will display the error message. The error message is also displayed her the system is not connected to the database and will display the error message accordingly.

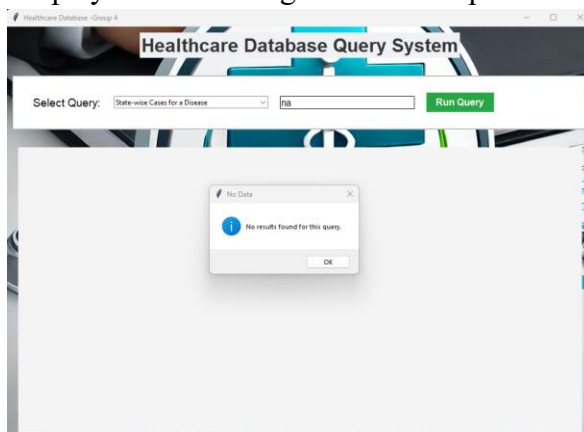




Input Validation: Provides specific error messages if the user input is invalid, such as non-numeric patient IDs or empty disease names.



- Displays Error message when the Input data is irrelevant to the selected query.



Usage

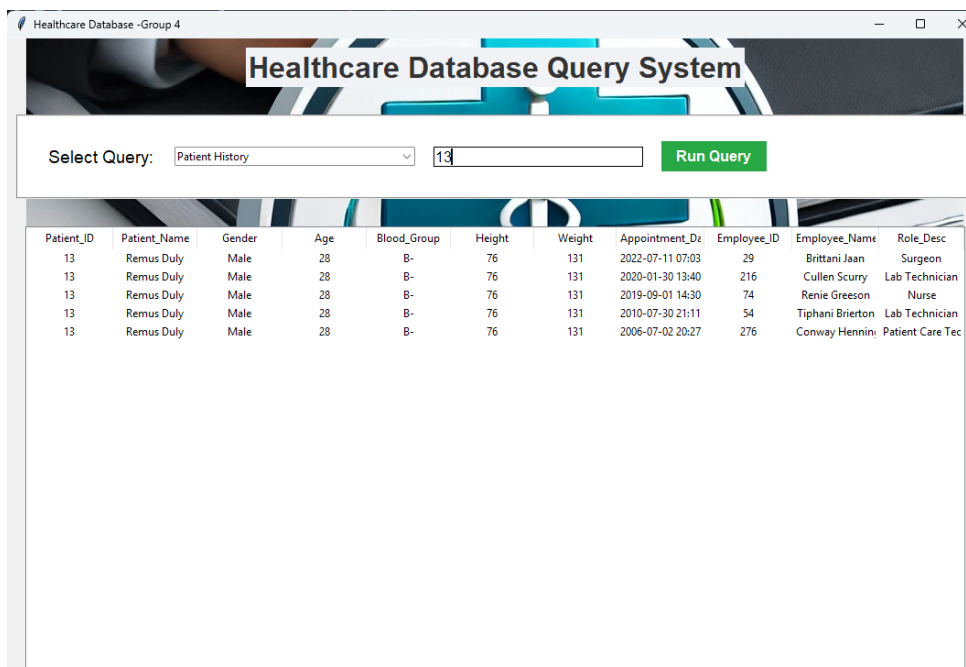
- Launch the application.
- Select a query from the "Select Query" dropdown.
- Enter the required parameters (e.g., Patient ID or Disease Name).

- Click "Run Query" to execute and view the results.

Query Execution From GUI

1. Patient History:

- Displays a detailed history of a patient's visits, treatments, and diagnoses. Requires a valid numeric Patient ID as input.
- Displays information such as Name, Age Gender, Blood group, Height, Weights Appointment dates, doctors consulted.
- Essential for tracking a patient's medical journey and ensuring continuity of care. Requires a valid numeric Patient ID as input.

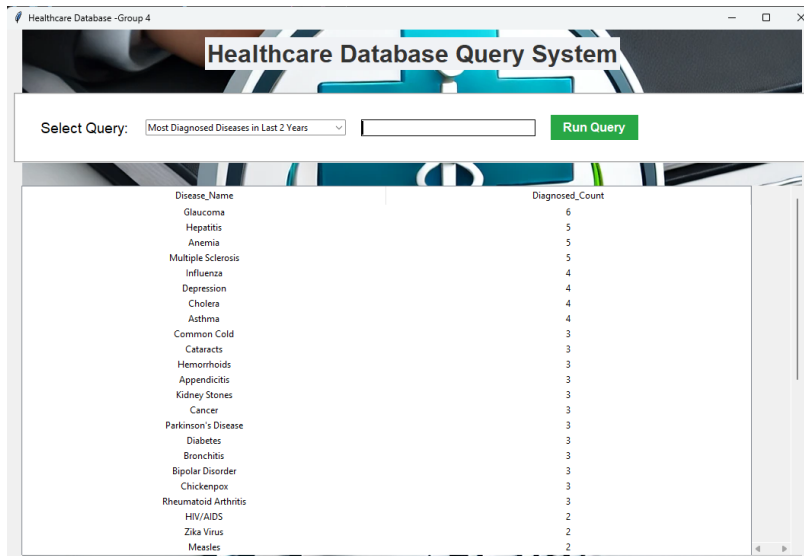


The screenshot shows a web application titled "Healthcare Database Query System". It features a search bar with a dropdown menu set to "Patient History" and a text input field containing the number "13". A green "Run Query" button is positioned to the right of the input field. Below the search bar, a table displays the query results for Patient ID 13. The table has 11 columns: Patient_ID, Patient_Name, Gender, Age, Blood_Group, Height, Weight, Appointment_De, Employee_ID, Employee_Name, and Role_Desc. The results show five entries for Patient ID 13, all with the name "Remus Duly" and gender "Male". The appointments are dated 2022-07-11, 2020-01-30, 2019-09-01, 2010-07-30, and 2006-07-02. The employees involved are Brittani Jaan (Surgeon), Cullen Scurry (Lab Technician), Renie Greeson (Nurse), Tiphani Brierton (Lab Technician), and Conway Hennin (Patient Care Tec).

| Patient_ID | Patient_Name | Gender | Age | Blood_Group | Height | Weight | Appointment_De | Employee_ID | Employee_Name | Role_Desc |
|------------|--------------|--------|-----|-------------|--------|--------|------------------|-------------|------------------|------------------|
| 13 | Remus Duly | Male | 28 | B- | 76 | 131 | 2022-07-11 07:03 | 29 | Brittani Jaan | Surgeon |
| 13 | Remus Duly | Male | 28 | B- | 76 | 131 | 2020-01-30 13:40 | 216 | Cullen Scurry | Lab Technician |
| 13 | Remus Duly | Male | 28 | B- | 76 | 131 | 2019-09-01 14:30 | 74 | Renie Greeson | Nurse |
| 13 | Remus Duly | Male | 28 | B- | 76 | 131 | 2010-07-30 21:11 | 54 | Tiphani Brierton | Lab Technician |
| 13 | Remus Duly | Male | 28 | B- | 76 | 131 | 2006-07-02 20:27 | 276 | Conway Hennin | Patient Care Tec |

2. Most Diagnosed Diseases in Last 2 Years:

- Shows the top diseases diagnosed within the last two years, useful for trend analysis.
- Displays information such as visit dates, doctors consulted, treatments administered, and outcomes.
- Essential for tracking a patient's medical journey and ensuring continuity of care.



The screenshot shows a web application titled "Healthcare Database Query System". Below the title is a "Select Query:" dropdown menu with "Most Diagnosed Diseases in Last 2 Years" selected. To the right of the dropdown is a text input field and a green "Run Query" button. Below the query selection area is a table with two columns: "Disease_Name" and "Diagnosed_Count". The table lists various diseases and their corresponding counts, sorted in descending order of count.

| Disease_Name | Diagnosed_Count |
|----------------------|-----------------|
| Glaucoma | 6 |
| Hepatitis | 5 |
| Anemia | 5 |
| Multiple Sclerosis | 5 |
| Influenza | 4 |
| Depression | 4 |
| Cholera | 4 |
| Asthma | 4 |
| Common Cold | 3 |
| Cataracts | 3 |
| Hemorrhoids | 3 |
| Appendicitis | 3 |
| Kidney Stones | 3 |
| Cancer | 3 |
| Parkinson's Disease | 3 |
| Diabetes | 3 |
| Bronchitis | 3 |
| Bipolar Disorder | 3 |
| Chickenpox | 3 |
| Rheumatoid Arthritis | 3 |
| HIV/AIDS | 2 |
| Zika Virus | 2 |
| Measles | 2 |

3. Patient visits by Age group:

- Breaks down the number of patient visits based on predefined age groups to understand demographics.
- Analyzes the volume of patient visits segmented by age groups (e.g., 0-15, 16-35, 36-50, 51+).
- Useful for identifying age-specific healthcare needs and tailoring services accordingly.

Healthcare Database - Group 4

Healthcare Database Query System

Select Query: Patient Visits by Age Group

| Age_Group | Total_Visits |
|------------------------|--------------|
| 16-35 (Young Adults) | 241 |
| 0-15 (Children) | 187 |
| 36-50 (Middle Aged) | 157 |
| 51-100 (Senior Adults) | 52 |

4. State-wise Cases for a Disease:

- Displays the count of cases for a specific disease, categorized by state. Shows state names, case counts, and possibly trends over time.
- Requires a Disease Name as input. Helpful for regional health monitoring and resource allocation.

Healthcare Database - Group 4

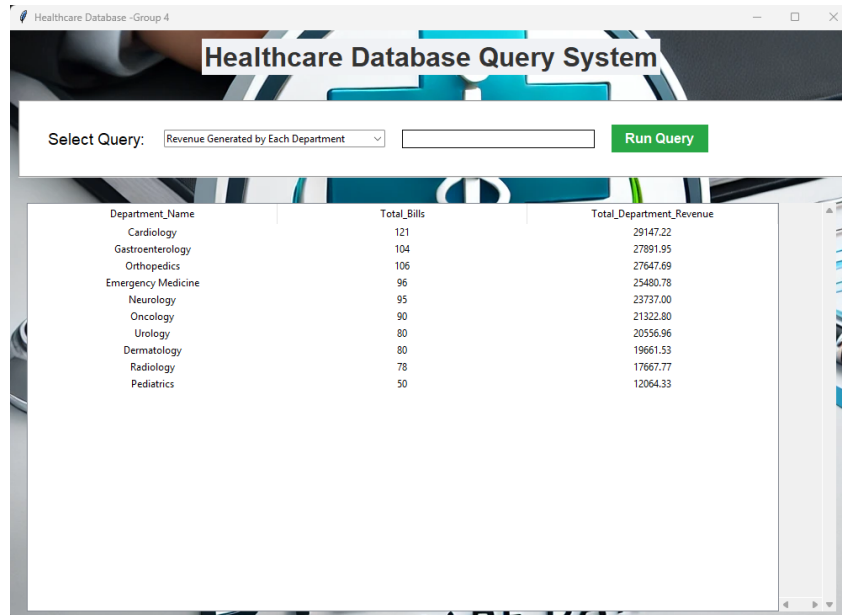
Healthcare Database Query System

Select Query: State-wise Cases for a Disease

| State | Disease | Total_Cases |
|----------|---------|-------------|
| New York | Malaria | 2 |
| Texas | Malaria | 2 |
| Nevada | Malaria | 1 |
| Colorado | Malaria | 1 |
| Montana | Malaria | 1 |
| Indiana | Malaria | 1 |
| Georgia | Malaria | 1 |

5. Revenue Generated by each Department :

- Provides a financial overview by showing the revenue generated by each department in the healthcare facility.
- Displays department names and the corresponding revenue generated, assisting in budget planning and financial management.



| Department_Name | Total_Bills | Total_Department_Revenue |
|--------------------|-------------|--------------------------|
| Cardiology | 121 | 29147.22 |
| Gastroenterology | 104 | 27891.95 |
| Orthopedics | 106 | 27647.69 |
| Emergency Medicine | 96 | 25480.78 |
| Neurology | 95 | 23737.00 |
| Oncology | 90 | 21322.80 |
| Urology | 80 | 20556.96 |
| Dermatology | 80 | 19661.53 |
| Radiology | 78 | 17667.77 |
| Pediatrics | 50 | 12064.33 |

6. Patients with the Most Visits:

- Lists the patients who have the highest number of visits, indicating frequent care or chronic conditions.
- Displays patient names, visit counts.
- Useful for managing chronic disease programs and personalized care.

Healthcare Database - Group 4

Healthcare Database Query System

Select Query: Patientz with the Most Visits Run Query

| Patient_ID | Patient_Name | Visit_Count |
|------------|--------------------|-------------|
| 558 | Giralda Samber | 5 |
| 107 | Tymothy Mitchel | 5 |
| 152 | Dalton Blazdell | 5 |
| 181 | Gabriel Gorgler | 5 |
| 13 | Remus Duly | 5 |
| 319 | Renate Buckham | 4 |
| 327 | Gardiner Romanelli | 4 |
| 676 | Kerry Meiklem | 4 |
| 137 | Mariana Silcox | 4 |
| 579 | Mycah Wyrall | 4 |

7. Total Appointments Per Doctor:

- Shows the total appointments scheduled for each doctor, assisting in workload analysis.
- Assists in balancing workload, scheduling efficiency, and identifying potential overbooking issues.

Healthcare Database - Group 4

Healthcare Database Query System

Select Query: Total Appointments Per Doctor 21 Run Query

| Employee_ID | Doctor_Name | Total_Appointments |
|-------------|----------------|--------------------|
| 21 | Derward Foulds | 7 |

8. List of Canceled Appointments:

- Displays appointments that were canceled, useful for understanding patient behavior and scheduling efficiency.

- Assists in balancing workload, scheduling efficiency, and identifying potential overbooking issues.

Healthcare Database - Group 4

Healthcare Database Query System

Select Query: List of Appointments by Status Run Query

| Appointment_ID | Patient_Name | Doctor_Name | Appointment_Date | Status |
|----------------|---------------------|-------------------|---------------------|-----------|
| 681 | Dona Dmitrienko | Lorena Readie | 2025-02-13 02:09:14 | Scheduled |
| 370 | Jania Scroggen | Edes Cheesworth | 2025-02-01 08:16:32 | Scheduled |
| 434 | Berly Enigo | Jennee Greatbach | 2024-12-28 03:41:30 | Scheduled |
| 443 | Donny Donald | Kerianne Roscam | 2024-09-07 04:59:47 | Scheduled |
| 857 | Mayer Frotton | Flynn Ogger | 2024-08-14 02:52:41 | Scheduled |
| 436 | Mada Felge | Robin Roswarne | 2024-08-01 10:40:27 | Scheduled |
| 986 | Lynne Bridal | Anthony Enga | 2024-04-02 07:28:32 | Scheduled |
| 628 | Fielding Barnhill | Dacia Rackham | 2024-03-23 08:11:26 | Scheduled |
| 121 | Judie Miles | Jennee Greatbach | 2024-03-07 19:58:21 | Scheduled |
| 844 | Sacha Salls | Chip Egdal | 2024-01-23 03:23:22 | Scheduled |
| 579 | Helene Seeger | Elissa McDonagh | 2024-01-19 03:08:33 | Scheduled |
| 315 | Perri Perfit | Norman Figgers | 2024-01-10 14:48:00 | Scheduled |
| 359 | Karole MacDonald | Paule Kobiera | 2023-07-23 19:46:48 | Scheduled |
| 9 | Marnia O'Hallihane | Constantia Sebert | 2023-07-16 05:03:15 | Scheduled |
| 652 | Reinhold Welham | Tamqrah Kidsley | 2023-07-11 13:13:30 | Scheduled |
| 845 | Zolly Lott | Wilmer Skippen | 2023-07-10 22:44:54 | Scheduled |
| 518 | Nathaniel Slowgrove | Eva Tranckle | 2023-05-05 16:44:53 | Scheduled |
| 111 | Julietta Cowins | Karoly Hooke | 2023-04-04 13:36:24 | Scheduled |
| 93 | Werner Simoncelli | Kerianne Roscam | 2023-02-12 21:57:09 | Scheduled |
| 328 | Sam Isip | Alina Bottell | 2023-02-11 19:13:41 | Scheduled |
| 913 | Melodie Rockswill | Cynthia Jamblin | 2022-12-23 08:29:37 | Scheduled |
| 161 | Tedman Radish | Irita Lavigne | 2022-11-29 21:25:52 | Scheduled |
| 367 | Trevor Diboll | Andromache Gigg | 2022-11-06 19:17:37 | Scheduled |

Healthcare Database - Group 4

Healthcare Database Query System

Select Query: List of Appointments by Status Run Query

| Appointment_ID | Patient_Name | Doctor_Name | Appointment_Date | Status |
|----------------|-------------------|--------------------|---------------------|-----------|
| 918 | Shara Matiewe | Thane Charlwood | 2025-01-30 09:17:47 | Completed |
| 802 | Datha Hadden | Humfrey Pumphrey | 2025-01-28 13:08:56 | Completed |
| 172 | Carver Krinks | Owen Andriuzzi | 2025-01-22 17:06:14 | Completed |
| 696 | Jenilee Willgrass | Maxim Van Leeuwen | 2024-12-27 13:13:47 | Completed |
| 302 | Etty Tourret | Staci Paula | 2024-12-23 06:14:13 | Completed |
| 581 | Willis Bradder | Tanner Bell | 2024-10-21 11:55:16 | Completed |
| 425 | Harriet Satterly | Lorelei Dellatorre | 2024-10-17 10:38:39 | Completed |
| 452 | Emera Kissick | Niall Staines | 2024-09-11 04:38:37 | Completed |
| 714 | Neda Yanov | Rasia Yendle | 2024-05-21 14:59:15 | Completed |
| 778 | Evelina Steaning | Thacher Rodway | 2024-04-21 03:39:10 | Completed |
| 515 | Kathlin Goring | Vickie Marcussen | 2024-04-19 19:27:40 | Completed |
| 350 | Gareth Tyreman | Karena Abbey | 2024-04-10 20:35:58 | Completed |
| 263 | Barron Meeke | Calida Fairlav | 2024-02-07 21:19:49 | Completed |
| 289 | Isac Spataro | Ambrose Mabbitt | 2024-01-11 07:23:33 | Completed |
| 856 | Weber Baudon | Reider Spur | 2023-12-08 07:30:45 | Completed |
| 235 | Eileen Iston | Nick Woodrough | 2023-11-17 00:23:23 | Completed |
| 728 | Dulciana Drayson | Marijn Jickles | 2023-11-04 04:05:06 | Completed |
| 48 | Corenda Amburyzy | Audrey Olexa | 2023-10-05 10:42:29 | Completed |
| 781 | Neill Duvelly | Brock Kagan | 2023-09-27 03:42:13 | Completed |
| 46 | Natty Marcham | Brock Kagan | 2023-08-10 12:29:36 | Completed |
| 914 | Casper Iddins | Orazio Hunsforth | 2023-07-28 15:03:29 | Completed |
| 3 | Omar Northwood | Xena Sheaf | 2023-07-19 20:42:22 | Completed |
| 769 | Lambert Fanton | Rozalie Duesbury | 2023-03-16 20:21:52 | Completed |

Healthcare Database - Group 4

Healthcare Database Query System

Select Query: List of Appointments by Status Run Query

| Appointment_ID | Patient_Name | Doctor_Name | Appointment_Date | Status |
|----------------|---------------------|-----------------------|---------------------|-----------|
| 445 | Timothée Olyet | Carl Brigman | 2025-01-20 16:09:31 | Cancelled |
| 786 | Jerrald Tomkins | Arda Cruzer | 2024-12-27 16:03:40 | Cancelled |
| 114 | Julian Kahan | Orazio Hunsworth | 2024-12-16 20:41:34 | Cancelled |
| 791 | Christoph McLae | Jolee Servis | 2024-12-16 14:43:03 | Cancelled |
| 888 | Tera Gasson | Brittani Jaan | 2024-11-21 20:21:43 | Cancelled |
| 509 | Rafaël Jakšić | Carlene Tohill | 2024-10-22 05:03:16 | Cancelled |
| 921 | Hennieta Scotchford | Farlie Mapam | 2024-10-15 22:33:16 | Cancelled |
| 69 | Shirline Bernardo | Gabbey Beau | 2024-08-29 16:19:20 | Cancelled |
| 977 | Neda Yanov | Carmelina Kenn | 2024-07-08 19:36:15 | Cancelled |
| 966 | Cherlynn Blackall | Maridel Mapis | 2024-07-01 07:13:12 | Cancelled |
| 567 | Eran Varlow | Eva Tranckle | 2024-06-23 11:20:01 | Cancelled |
| 338 | Martin Bariball | Hakim Mancser | 2024-05-18 17:35:01 | Cancelled |
| 78 | Willie Shine | Hedvig Haquin | 2024-05-19 02:17:54 | Cancelled |
| 732 | Corliss Heiner | Archibald Gullford | 2024-03-16 21:30:39 | Cancelled |
| 10 | Derwin Vavton | Calida Fairlaw | 2024-01-31 14:34:38 | Cancelled |
| 689 | Dalton Blazdell | Myrta Wylt | 2023-12-02 00:18:16 | Cancelled |
| 717 | Adham Schurig | Celestine De Courtney | 2023-11-26 11:17:50 | Cancelled |
| 377 | Corliss Heiner | Niall Staines | 2023-11-09 20:36:59 | Cancelled |
| 404 | Roxie Mitchenson | Randell Dowry | 2023-10-29 12:22:01 | Cancelled |
| 843 | Branden Woodyatt | Erl Wicks | 2023-09-28 10:06:04 | Cancelled |
| 188 | Penelope Shenton | Tawsha Brigge | 2023-08-23 01:16:07 | Cancelled |
| 957 | Pepita Itzkovici | Shanna Hughman | 2023-06-03 23:57:13 | Cancelled |
| 789 | Tanney Catherod | Elissa McDonagh | 2023-05-25 12:42:35 | Cancelled |

9. Patients by Blood Group:

- Groups patients by their blood type (e.g., A+, O-, AB).
- Displays patient demographics and blood group distribution, which can support blood bank management and emergency readiness.

Healthcare Database - Group 4

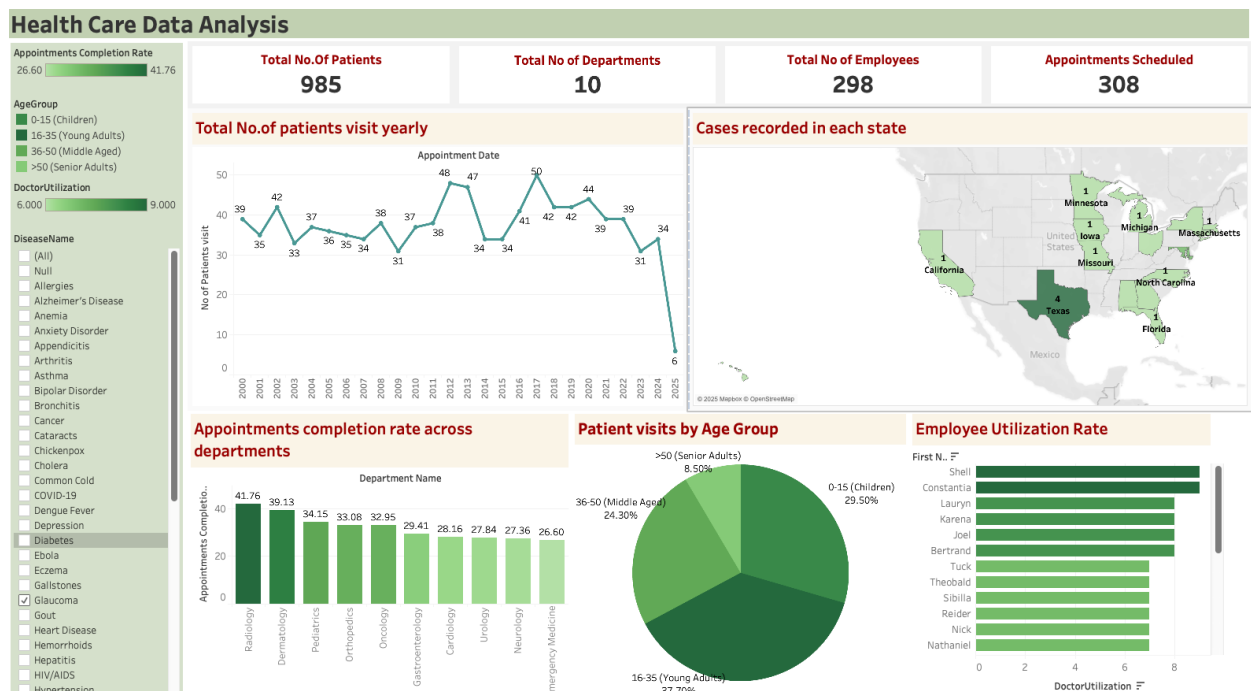
Healthcare Database Query System

Select Query: Patients by Blood Group Run Query

| Blood_Group | Patient_Count |
|-------------|---------------|
| A+ | 139 |

Data Visualization

- We have created the dashboard in Tableau which provides a comprehensive overview of hospital performance by tracking patient visits, disease trends, appointment efficiency, and resource utilization. The visualizations align with the project goals of improving patient care, optimizing operations, and enhancing data-driven decision-making.



KPIs (Top Metrics)

- **Total No. of Patients (985):** Helps track hospital **patient intake**.
- **Total No. of Departments (10):** Gives an overview of available **medical specialties**.
- **Total No. of Employees (295):** Ensures proper **staffing and resource allocation**.
- **Appointments Scheduled (308)**
- **):** Measures **hospital workload and appointment demand**.

Patient Visit Trends (Line Chart)

This Shows total patient visits yearly from 2000 to 2025. Identifies trends in hospital traffic and seasonal patient fluctuations. Declining or increasing trends help in forecasting hospital resource needs.

- Enables data-driven resource planning to reduce wait times & manage peak hours efficiently.

Cases Recorded in Each State (Geographical Heatmap)

Visualizes disease spread across different states. This helps to identify regions with higher cases recorded. Helps in tracking regional outbreaks or location-specific health trends.

- Supports data-driven healthcare policies & disease prevention efforts.

Appointment Completion Rate Across Departments (Bar Chart)

Displays how efficiently departments complete appointments. Radiology (41.76%) has the highest completion rate, while Emergency Medicine is lowest. Helps identify departments struggling with scheduling & patient flow.

- Ensures efficient scheduling & reduced wait times by improving departmental workflows.

Patient Visits by Age Group (Pie Chart)

- Breaks down hospital visits by age:
 - 16-35 (Young Adults) = 37.7% (Highest)
 - 0-15 (Children) = 29.5%
 - 36-50 (Middle Aged) = 24.3%
 - 50+ (Senior Adults) = 8.5% (Lowest)
- Shows which demographic requires the most care. Helps in personalized healthcare planning for different age groups.

Employee Utilization Rate (Bar Chart)

Displays workload distribution among doctors/employees. This helps to identify overworked and underutilized staff.

- Optimizes workforce efficiency and reduces doctor burnout.

Link to the work in Tableau Public Cloud

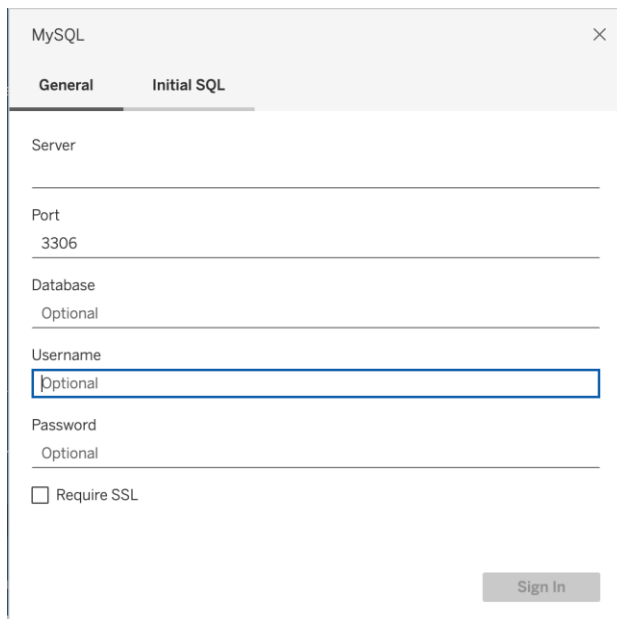
https://public.tableau.com/app/profile/devi.sowjanya.padala/viz/HealthcareData_TableauReport/Dashboard1?publish=yes

Getting Started

- Full export of database, including stored procedures and data are in the folder SQLDump. The 'HealthCareData.sql' file inside the 'SQLDump' folder consists of all the queries. Also the 'am_dpadala_routines.sql' file has stored procedures.
- We can also run queries by uploading the ERD diagram file (HospitalManagement_DBModel 2.mwb).
- Now this ERD model is converted into MYSQL model in the workbench which creates the tables based on the ERD model.
- Once the tables are created the csv files with data of each table is imported. While importing we have to make sure that the column in the dataset matches with the database columns.
- Once the data in the folder 'Datasets' is imported to tables, we can run queries by joining multiple table and derive insights.
- In case of stored procedure we have to invoke the procedure by passing the variables according to the function defined.

Creating visualizations in tableau

- In order to connect MYSQL database with tableau we need to download the ODBC driver.
- Once the driver is installed the tableau needs to be connected to database by providing the DB credentials as shown in the image below.



The image shows a 'MySQL' connection dialog box with a close button (X) in the top right corner. It has two tabs: 'General' (selected) and 'Initial SQL'. The 'General' tab contains the following fields: 'Server' (empty), 'Port' (3306), 'Database' (Optional), 'Username' (Optional, highlighted with a blue border), 'Password' (Optional), and a checkbox for 'Require SSL' which is unchecked. A 'Sign In' button is located at the bottom right.

- Once the connection is established all the tables will be loaded to the tableau dashboard in datasource and used to create visualizations which is shown below.
- The tables must be joined based on the matching primary keys and valid then only we can create charts.

The screenshot displays the Tableau Desktop interface for a project named 'am_dpadata'. On the left, the 'Connections' pane shows a connection to 'cssql.seattleu.edu MySQL'. Below it, the 'Database' dropdown is set to 'am_dpadata'. The 'Table' list on the left includes various tables such as 'actor', 'Address', 'Appointment', 'category', 'city', 'country', 'customer', 'customer_list', 'Customers', 'Department', 'Disease', 'Employee', 'Employee_Address', 'Expensive_Check', 'Feedback', 'film', 'New Custom SQL', 'New Union', and 'New Table Extension'. The main workspace shows a data model diagram with 'Patient' as the central entity, connected to 'Appointment', 'Patient_Address', and 'Patient_Register'. 'Appointment' is connected to 'Employee', which is further connected to 'Department' and 'Feedback'. 'Patient_Register' is connected to 'Patient_Disease', which is connected to 'Disease'. Below the diagram, a dropdown menu is set to 'Patient', and a data preview table is shown with columns: Patient ID (Patient), First Name, Last Name, Date Of Birth, Gender, and Phone Number. Below the preview are buttons for 'Update Now' and 'Update Automatically'. The bottom status bar shows 'Data Source' and a list of sheets from 'Sheet 1' to 'Sheet 11', along with 'Dashboard 1'.

- Multiple sheets are created for each chart or text fields and integrated all the sheets in the dashboard.

Conclusion

The **Health Care Data Management System** successfully integrates **patient records, appointments, billing, employee management, and analytics** into a **structured and efficient framework**.

- By leveraging a **relational database model**, we ensured **data consistency, integrity, and scalability**, enabling seamless **data retrieval, storage, and processing**.
- **SQL-based queries and stored procedures** automate **key hospital operations**, providing real-time insights into **patient history, doctor performance, disease prevalence, and financial analytics**.
- **Tableau dashboards** provide **interactive visualizations**, helping hospital administrators make **data-driven decisions** to **improve patient care, optimize resources, and enhance operational efficiency**.