Healthcare System Database

By

Ezennaya, Chinenye Kate

Mogollón Vargas, Ingrid

Naw Mu Aye, Naw Mu Aye

University of Niagara Falls Canada

A Project Report Submitted to

Prof.Abbas Hamze

CPSC500-1: SQL DATABASE

Faculty of Master's in Data Analytics

Healthcare System Database

The provided database schema is designed for a healthcare management system, offering a comprehensive framework to manage various aspects of healthcare facilities, patients, and their interactions. This schema facilitates efficient organization, storage, and retrieval of critical information, supporting operations like patient registration, medical consultations, prescriptions, billing, and inventory management.

Benefits of the Schema

1. **Data Integrity**: The use of primary and foreign keys ensures consistency and prevents data duplication.
2. **Scalability**: The modular design allows for easy expansion to accommodate new features or additional entities.
3. **Operational Efficiency**: Automates repetitive tasks such as appointment tracking, inventory restocking, and insurance processing.
4. **Improved Patient Care**: Provides healthcare professionals with quick access to patient records and history, enabling informed decision-making.

Database Design and Schema Overview

Schema Overview

The schema is composed of `14 interconnected tables, each serving a specific purpose within the healthcare management system. These tables work together to facilitate seamless data management, ensuring accurate and consistent information across the organization.

Description of the Schema

Following figure shows the description of the schema components and their purpose:

Figure 1:

Description of Schema Components and Their Purposes

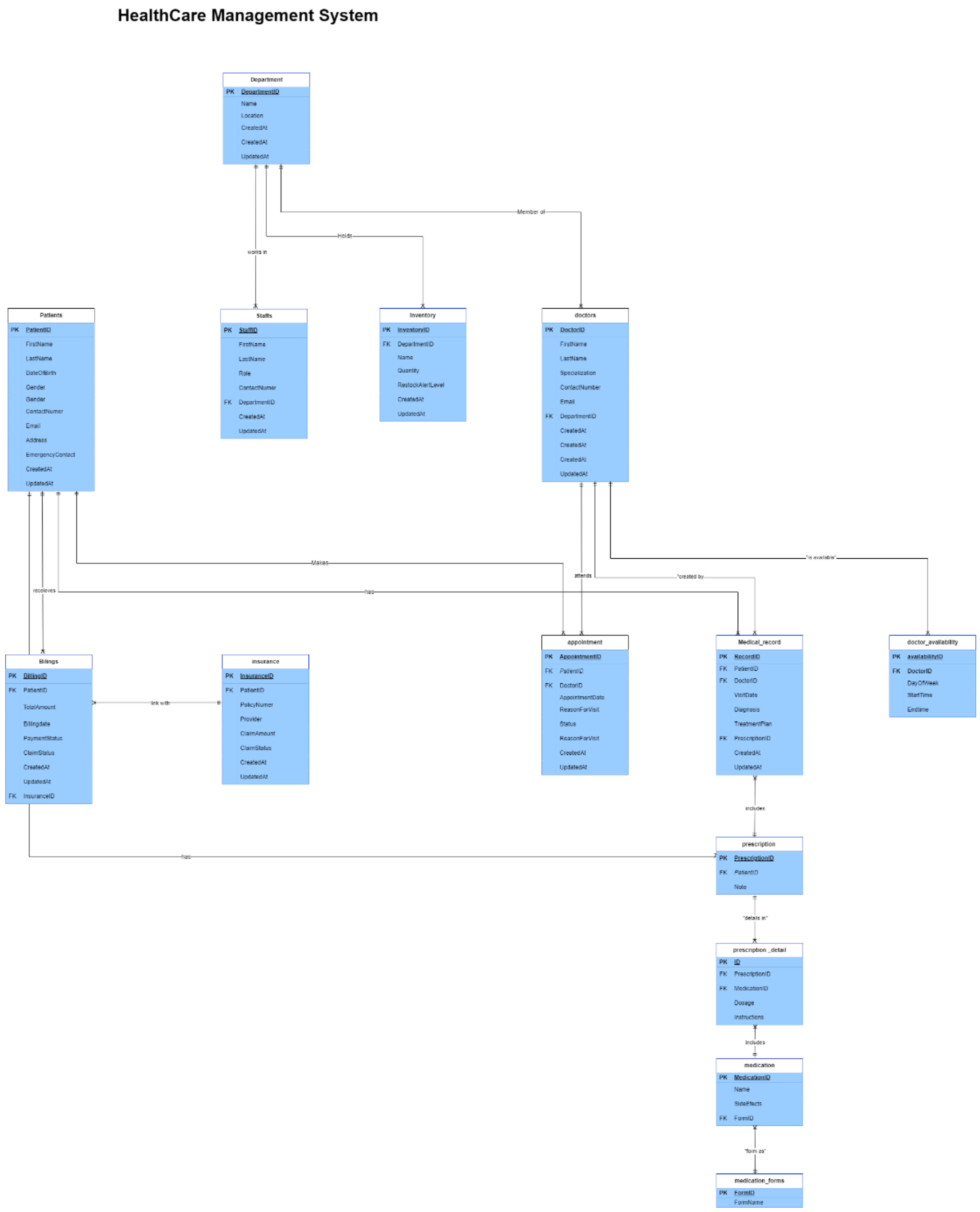
Schema Table Overview		
Table Name	Purpose	Key Attributes
Patients	Stores patient details, including personal and emergency contact information.	PatientID, FirstName, LastName, DateOfBirth, Gender, ContactNumber, Email, Address, EmergencyContact
Doctors	Contains information about doctors, their specializations, and department affiliations.	DoctorID, FirstName, LastName, Specialization, ContactNumber, Email, DepartmentID
Appointments	Tracks appointments between patients and doctors, including statuses and reasons for visits.	AppointmentID, PatientID, DoctorID, AppointmentDate, ReasonForVisit, Status
Medical Records	Documents details of patient visits, diagnoses, treatments, and prescriptions.	RecordID, PatientID, DoctorID, VisitDate, PrescriptionID, Diagnosis, TreatmentPlan
Prescriptions	Stores prescriptions issued to patients by doctors.	PrescriptionID, PatientID, DoctorID, Note
Prescription Details	Tracks specific medications, dosages, and instructions within a prescription.	ID, PrescriptionID, MedicationID, Dosage, Instructions
Medications	Provides information about available medications, including side effects and forms.	MedicationID, Name, FormID, SideEffects
Medication Forms	Defines various forms of medications (e.g., tablets, syrups).	FormID, FormName
Insurance	Maintains details of insurance policies and claim statuses for patients.	InsuranceID, PolicyNumber, Provider, ClaimAmount, ClaimStatus
Billings	Records billing details, including total amounts and payment statuses.	BillingID, PatientID, TotalAmount, BillingDate, PaymentStatus, InsuranceID
Departments	Organizes information about hospital departments and their locations.	DepartmentID, Name, Location
Inventory	Tracks medical supplies and equipment, including stock levels and restocking requirements.	InventoryID, ItemName, Quantity, RestockLevel, SupplierID
Staffs	Stores details of hospital staff, their roles, and associated departments.	StaffID, FirstName, LastName, Role, ContactNumber, DepartmentID
Doctor Availability	Details the working schedules of doctors, specifying days and time slots.	AvailabilityID, DoctorID, DayOfWeek, StartTime, EndTime

ER Diagram

The Entity-Relationship (ER) Diagram illustrates the relationships between the entities in the database schema, showing how they interact and the rules governing these interactions. Draw IO diagram for ER Diagram can be accessed from this link.

Figure 2:

ER Diagram of Healthcare Management System



Description of the relationships between entities and the rules governing these relationships are described in the following table.

Table 1:

List of Entities and Their Relationship Table

Relationship	Entities Involved	Relationship Type	Rule Governing the Relationship
Patient ↔ Appointments	Patients, Appointments	One-to-Many	A single patient can have multiple appointments, but each appointment is linked to one patient.
Doctor ↔ Appointments	Doctors, Appointments	One-to-Many	A doctor can have multiple appointments, but each appointment is associated with only one doctor.
Patient ↔ Medical Records	Patients, Medical Records	One to many	A patient can have multiple medical records, with each record tied to one specific patient.
Medical Records ↔ Prescriptions	Medical Records, Prescriptions	One to many	A Prescription can be linked to many Medical Records, but each Medical Record can be linked to one Prescription.
Prescription ↔ Prescription Details	Prescriptions, Prescription Details	One-to-Many	A prescription can contain multiple medication details, but each detail refers to one prescription.
Medications ↔ Medication Forms	Medications, Medication Forms	One to many	One Medication Form can have many Medications
Patient ↔ Billings	Patients, Billings	One-to-Many	A patient can have multiple billing records, but each billing record is linked to one patient.
Patient ↔ Insurance	Patients, Insurance	One-to-many	Each patient can have one insurance policy, but a patient might not have insurance.
Department ↔ Doctors	Departments, Doctors	One-to-Many	A department can have multiple doctors, but each doctor is assigned to one department.
Department ↔ Inventory	Departments, Inventory	One-to-Many	A department can manage multiple inventory items, but each inventory item belongs to one department.
Department ↔ Staffs	Departments, Staffs	One-to-Many	A department can have multiple staff members, but each staff

			member is assigned to one department.
Doctor ↔ Doctor Availability	Doctors, Doctor Availability	One-to-Many	A doctor can have multiple availability slots, but each slot belongs to one doctor.
Billing ↔ Insurance	Billings, Insurance	One-to-many	One Patient can have many Billings,
Doctors ↔ Medical Records:	Doctors, Medical Record	:One-to-Many	A single doctor can create multiple medical records, but each record is created by one doctor.

Implementation with SQL Statement

Data Definition Statements (DDL)

The followings are the SQL statements use to create the database schema. These statements define the structure of the tables, relationships between them and constraint to ensure data integrity. The following lists only explain key tables that used in this project.

Creating Patients Table

Figure 3:

SQL Script to Create Patient Table

```
/*--------- Table 1: Patient Table ------------ */
CREATE TABLE patients(
    PatientID INT PRIMARY KEY,
    FirstName VARCHAR(32) NOT NULL,
    LastName VARCHAR(32) NOT NULL,
    DateOfBirth DATE NOT NULL,
    GENDER  ENUM('Male','Female','Other') NOT NULL,
    ContactNumber VARCHAR(64) NOT NULL,
    email   VARCHAR(64),
    Address VARCHAR(128),
    EmergencyContact VARCHAR(64) NOT NULL,
    CreatedAt DATETIME NOT NULL DEFAULT NOW(),
    UpdatedAt DATETIME NOT NULL DEFAULT NOW()
);
```

- Primary Key: PatientID uniquely identifies each patient in the table.
- NOT NULL: Ensures essential fields like FirstName, LastName, ContactNumber and EmergencyContact cannot be left empty.

- DEFAULT: Set default values for CreatedAt and UpdatedAt column to capture timestamps automatically.
- ENUM: Restricts Gender to specific values (Male, Female, Other)

Creating Doctors Table

Figure 4:

SQL Script to Create Patient Table

```
/*--------- Table 2: Doctor Table ------------ */
CREATE TABLE doctors(
    DoctorID INT PRIMARY KEY,
    FirstName VARCHAR(32) NOT NULL,
    LastName VARCHAR(32) NOT NULL,
    Specialization VARCHAR(128) NOT NULL,
    ContactNumber VARCHAR(64) NOT NULL,
    email VARCHAR(64) NOT NULL,
    DepartmentID INT NOT NULL,
    CreatedAt DATETIME NOT NULL DEFAULT NOW(),
    UpdatedAt DATETIME NOT NULL DEFAULT NOW()
);
```

- Primary Key: DoctorID uniquely identifies each Doctor in the table.
- NOT NULL: Ensures essential fields like FirstName, LastName, ContactNumber, email, specialization and DepartmentID cannot be left empty.
- DEFAULT: Set default values for CreatedAt and UpdatedAt column to capture timestamps automatically.
- Doctors Table is later linked to Department table using DepartmentID as foreign key using ALTER statement after Department table was created.

Figure 5:

Using ALTER DDL Statement to Add Relationship Constraint in Doctors Table

```
/*------- Add relationship Constraints -------------*/
ALTER TABLE doctors ADD  FOREIGN KEY(DepartmentID) REFERENCES departments(DepartmentID) ON DELETE CASCADE;
```

- Foreign Key: DepartmentID establishes the relationship between doctors table and departments table. This ensures that doctor can have association with only valid department.

- ON DELTE CASCADE: This is to ensure to remove the records if a parent record is deleted. Example, deleting the department will delete the records of the doctors that works in associated department.

Creating Insurance Table

Figure 6:

SQL Script to Create Insurance Table

```
/*--------- Table 13: Insurance Table ------------ */

CREATE TABLE insurance(
    InsuranceID INT PRIMARY KEY,
    PatientID INT NOT NULL,
    PolicyNumber VARCHAR(64) NOT NULL,
    Provider VARCHAR(64) NOT NULL,
    ClaimAmount DECIMAL(10,2) NOT NULL DEFAULT 0.00,
    ClaimStatus ENUM('Approved','Pending','Rejected'),
    CreatedAt DATETIME NOT NULL DEFAULT NOW(),
    UpdatedAt DATETIME NOT NULL DEFAULT NOW(),
    FOREIGN KEY(PatientID) REFERENCES  patients(PatientID)
);
```

- Primary Key: InsuranceID uniquely identifies each insurance record.
- Foreign Key: PatientID references the patients table, establishing a relationship between patients and their insurance records.
- ON DELETE CASCADE: Automatically deletes insurance records if the associated patient record is deleted.

In order to maintain data consistency and normalize the tables,

- PaitentID foreign key was removed form insurance table and replaced with billingID from billings table. Insurance table is then linked to Patients table via billing table using InsuranceID as foreign key in billing table.
- Modify the CalimAmount column property by adding CHECK condition so that CliamAmount will not be negative values.

The followings figures show the SQL statements to make the modifications. Note that the relationship constraint has to drop first before dropping the attribute from the table.

Figure 7:

ALTER Statements to Alter the Attributes in Insurance Table

```
ALTER TABLE billings ADD InsuranceID INT;
ALTER TABLE billings ADD FOREIGN KEY(InsuranceID) REFERENCES insurance(InsuranceID) ON DELETE CASCADE;
ALTER TABLE insurance DROP CONSTRAINT insurance_ibfk_1;
ALTER TABLE insurance DROP COLUMN PatientID;
ALTER TABLE insurance MODIFY COLUMN ClaimAmount DECIMAL(10,2) NOT NULL CHECK (ClaimAmount >= 0);
```

Creating Medical Record Table

Figure 8:

SQL Statement to Create Medical Record Table

```
/*--------- Table 5: Medical Record Table ------------ */
CREATE TABLE medical_records(
    RecordID INT AUTO_INCREMENT PRIMARY KEY,
    PatientID INT NOT NULL,
    DoctorID INT NOT NULL,
    VisitDate DATE NOT NULL,
    PrescriptionID INT,
    Diagnosis TEXT NOT NULL,
    TreatmentPlan TEXT NOT NULL,
    Notes TEXT,
    CreatedAt DATETIME NOT NULL DEFAULT NOW(),
    UpdatedAt DATETIME NOT NULL DEFAULT NOW(),
    FOREIGN KEY(DoctorID) REFERENCES doctors(DoctorID),
    FOREIGN KEY(PatientID) REFERENCES patients(PatientID)
);
```

- Primary Key: PrescriptionID uniquely identifies each prescription.
- Foreign Keys:

PatientID references patients.PatientID to link prescriptions to patients.

DoctorID references doctors.DoctorID to associate prescriptions with doctors.

PrescriptionID references prescriptions.Prescription ID to associate with which prescriptions has been made to patients. Based on above script, either of these attributes are dropped, associated medical records will not be deleted. In order to have data consistency and optimize the database storage, the constraint to delete the medical record data when the patient ID or prescription ID is dropped. The following statements are used to make the required modifications.

Figure 9:

Modifying Key Constraint in Medical Record Table.

```
ALTER TABLE medical_records ADD FOREIGN KEY(PatientID) REFERENCES patients(PatientID) ON DELETE CASCADE;
ALTER TABLE medical_records ADD FOREIGN KEY(PrescriptionID) REFERENCES prescription(PrescriptionID) ON DELETE CASCADE;
```

Data Modification Statements

After tables are created, Data are generated via AI tools or python scripts. The generated date is inserted into tables using INSERT DML statements.

Figure 10:

Using INSERT Statement to Insert Data into Medication Form Table

```
INSERT INTO MedicationForms (FormName)
VALUES ('Tablet'), ('Capsule'), ('Liquid'), ('Syrup'), ('Injection'),
       ('Topical'), ('Suppository'), ('Inhaler'), ('Drops'), ('Patch'),
       ('Powder'), ('Lozenge'), ('Granule'), ('Spray');
```

InsuranceID column was added to billings table after billings table was created and filled up with data. Therefore, after adding InsuranceID column to billings table, the data to InsuranceID column in billings table was filled using UPDATE DML statement.

Figure 11:

Adding Values into InsuranceID Column of Billings Table Using UPDATE Statement

```
UPDATE billings
SET InsuranceID = CASE BillingID
    -- Assign non-null unique values
    WHEN 1 THEN 1
    WHEN 2 THEN 2
    WHEN 3 THEN 3
    WHEN 4 THEN 4
    WHEN 5 THEN 5
    WHEN 6 THEN 6
    WHEN 7 THEN 7
    WHEN 8 THEN 8
    WHEN 9 THEN 9
    WHEN 10 THEN 10
    WHEN 11 THEN 11
    WHEN 12 THEN 12
    WHEN 13 THEN 13
    WHEN 14 THEN 14
    WHEN 15 THEN 15
    --...........

    WHEN 89 THEN 282
    WHEN 90 THEN 283
    WHEN 91 THEN 284
    WHEN 92 THEN 285
    WHEN 93 THEN 286
    WHEN 94 THEN 287
    WHEN 95 THEN 288
    WHEN 96 THEN 289
    WHEN 97 THEN 290
    WHEN 98 THEN 291
    WHEN 99 THEN 292
    WHEN 100 THEN 293
    WHEN 101 THEN 294
    WHEN 102 THEN 295
    WHEN 103 THEN 296
    WHEN 104 THEN 297
    WHEN 105 THEN 298
    WHEN 106 THEN 299
    WHEN 107 THEN 300
    -- Assign NULL values to the remaining rows
    ELSE NULL
END
WHERE BillingID BETWEEN 1 AND 203;
```

Data Retrieval

Purpose of Data Retrieval

The main purpose of the healthcare management database is to provide structured datasets that enable analysis and decision-making within the hospital care field through the implementation of data retrieval.

The data obtained in this dataset not only supports daily operations but can also be used to create queries using machine learning and predictive analytics. One of the key points in the retrieval and organization of data for the healthcare system database includes patient records, diagnoses, specialists, departments, and inventory management.

Queries Samples for Machine Learning

This section lists the sample Queries from the database to prepare dataset for various Machine Learning purposes.

Query 1. Patient Treatment Outcome Analysis

This dataset is prepared to use when predicting the patient treatment outcome. It makes used of the following data in the database.

- Patient:
  - PatientID → this ID is used to merge the tables to collect required data
  - Gender → Analysis or prediction may be able to make based on Gender
- MedicalRecord:
  - Diagnosis → Prediction can be based on which disease condition is diagnosed.
  - TreatmentPlan → One can analyse which treatment is more effective and which are not.
  - Difference between UpdatedAt and CreatedAt dates can serve as Treatment Duration and Treatment Duration can be categorized as Success or Failure by setting a threshold value.

- Prescription
    - PrescriptionID: prescribed medication can be obtained by linking medication, prescription_detail and prescription table using PrescriptionID, MedicationID and Medication Name. A view for prescribed medication named ‘ prescribe_med’ has been created so that it can be reused in future queries.

The columns in Output tables contains the columns PatientID, FirstName, LastName, Age, Gender, Region, ConditionDiagnosed, TreatmentProvdied, PrescribedMedication, TreatmentOutcome, TreatmentStartDate, TreatmentEndDate and TreatmentDurtaion. The following figures show the snapshots of the output data and SQL queries.

Figure 12:

Output of the SQL Queries for Predicting Patient Outcomes

PatientID	FirstName	LastName	Age	Gender	Region	ConditionDiagnosed	TreatmentProvided	PrescribedMedications	TreatmentOutcome	TreatmentStartDate	TreatmentEndDate	TreatmentDurationDays
45	Katherine	Curtis	2.1	Male	Other	Arthritis	Use of inhaler as needed	Benazepril,Isosorbide	Failure	2023-07-24 12:29:56	2024-10-29 11:52:00	463
45	Katherine	Curtis	2.1	Male	Other	Upper Respiratory Infection	Bronchodilators	Diclofenac,Doxycycline,Fluticasone,Metoprolol,...	Success	2024-12-10 01:27:23	2024-12-29 07:05:28	19
66	Megan	Perez	2.5	Male	Other	Acute Bronchitis	Pain management and rest	Naproxen	Success	2024-09-14 11:15:47	2024-10-07 23:21:50	23
82	Evan	Barnett	2.9	Other	Other	Chronic Back Pain	Physical therapy and pain relief	Zonisamide	Failure	2023-03-27 10:42:20	2024-08-30 06:45:44	522
81	Patricia	Lewis	9.2	Female	Other	Upper Respiratory Infection	Physiotherapy sessions	Citalopram,Doxycycline	Failure	2023-05-29 03:32:56	2023-11-25 06:43:15	180
81	Patricia	Lewis	9.2	Female	Other	Upper Respiratory Infection	Lifestyle modification and medication	Citalopram	Failure	2023-03-12 18:13:19	2023-09-06 11:32:18	178
97	George	Cunningham	9.2	Female	Other	Arthritis	Cognitive behavioral therapy	Tramadol	Failure	2024-04-04 18:18:19	2024-08-23 08:52:32	141

Figure 13:

SQL Queries for Predicting Patient Outcomes

```
/*--------- Query 1 (POINT 1). Patient Treatment Outcome Analysis ------------ */

CREATE VIEW prescribe_med AS(
SELECT
    pd.PrescriptionID,
    # pd.MedicationID,
    GROUP_CONCAT( m.Name ORDER BY m.Name SEPARATOR ',') AS precribed_medication
FROM prescription_detail as pd
    JOIN medication AS m ON pd.MedicationID = m.MedicationID
GROUP BY pd.PrescriptionID
ORDER BY pd.PrescriptionID) ;
SELECT
    P.PatientID,
    P.FirstName,
    P.LastName,
    ROUND(DATEDIFF(CURDATE(), P.DateOfBirth) / 365.25, 1) AS Age,
    P.Gender,
    CASE
        WHEN P.Address LIKE '%Street%' THEN 'Urban'
        WHEN P.Address LIKE '%Village%' THEN 'Rural'
        ELSE 'Other'
    END AS Region,
    Mr.Diagnosis AS ConditionDiagnosed,
    Mr.TreatmentPlan AS TreatmentProvided,
    pm.precribed_medication AS PrescribedMedications,
    CASE
        WHEN DATEDIFF(Mr.UpdatedAt, Mr.CreatedAt) <= 30 THEN 'Success'
        WHEN DATEDIFF(Mr.UpdatedAt, Mr.CreatedAt) > 30 THEN 'Failure'
        ELSE 'Unknown'
    END AS TreatmentOutcome,
    Mr.CreatedAt AS TreatmentStartDate,
    Mr.UpdatedAt AS TreatmentEndDate,
    DATEDIFF(Mr.UpdatedAt, Mr.CreatedAt) AS TreatmentDurationDays
FROM
    Patients AS P
JOIN
    medical_records AS Mr ON P.PatientID = Mr.PatientID
LEFT JOIN
    prescription AS Pr ON Mr.PrescriptionID = Pr.PrescriptionID
JOIN
    prescribe_med AS pm ON pm.PrescriptionID = pr.PrescriptionID
WHERE
    Mr.CreatedAt >= '2023-01-01'
ORDER BY
    Age, Gender, TreatmentOutcome;
```

Query 2. Inventory Resource Optimization and Demand Forecasting

This query is designed to use in prediction hospital resource such as medications, equipment while hospital revenue in check. It makes used of the following data in the database.

- Doctors:
  - DoctorID: to estimate number of appointments are made for each department by linking Doctors table with Appointment table
  - DepartmentID: to link between inventory table and appointment table via doctor table to count resource demand (number of appointments) and current supply(stock available in inventory)
- Appointment
  - AppointmentID: counting this value group by department will result in estimate demand for each department
  - AppointmentDate: The attribute is used to organize the status by monthly.
- Inventory
  - InventoryID, Name, Quantity: use this values to reflect inventory status
  - RestockAlertLevel and InventoryStatus: If Quantity ≤ RestockAlertLevel → Set Status to ‘Restock Needed’; Else ‘Sufficient Stock’ is set.

The columns in Output tables are Month, DepartmentID, NumberOfAppointment, TotalRenvenueGenerated, InventoryID, InventoryName, CurrentInventory (Quantity), RestockAlertLevel and StockStatus. The following figures show the snapshots of the output data and SQL queries.

Figure 14:

Output of SQL query for Resource Optimization and Demand Forecasting

Month	DepartmentID	NumberOfAppointments	TotalRevenueGenerated	InventoryID	InventoryName	CurrentInventory	ResotckAlertLevel	StockStatus
2024-01	1	12	3121.24	1	Syringes	150	10	Sufficient Stock
2024-01	1	12	3121.24	2	Bandages	250	20	Sufficient Stock
2024-01	1	12	3121.24	3	Gloves	500	50	Sufficient Stock
2024-01	1	12	3121.24	4	Thermometers	80	10	Sufficient Stock
2024-01	1	12	3121.24	5	Needles	200	15	Sufficient Stock
2024-01	1	12	3121.24	6	Stethoscopes	75	5	Sufficient Stock
2024-01	1	12	3121.24	7	Alcohol Swabs	300	30	Sufficient Stock
2024-01	1	12	3121.24	8	Masks	450	40	Sufficient Stock

Figure 15:

SQL Queries for Resource Optimization and Demand Forecasting

```
/*--------- Query 2: (POINT 2) Analysis of Hospital Inventories and
            Demand Based on Appointments and Economic Activity  ------------ */

SELECT
    DATE_FORMAT(Ap.AppointmentDate, '%Y-%m') AS Month,
    Dr.DepartmentID,
    COUNT(Ap.AppointmentID) AS NumberOfAppointments,
    SUM(B.TotalAmount) AS TotalRevenueGenerated,
    i.InventoryID,
    i.Name AS InventoryName,
    i.Quantity AS CurrentInventory,
    i.ResotckAlertLevel,
    CASE
        WHEN i.Quantity <= i.ResotckAlertLevel THEN 'Restock Needed'
        ELSE 'Sufficient Stock'
    END AS StockStatus
FROM
    appointments AS Ap
JOIN
    doctors AS Dr ON Ap.DoctorID = Dr.DoctorID
LEFT JOIN
    Inventory AS i ON Dr.DepartmentID = i.DepartmentID
LEFT JOIN
    billings AS B ON Ap.PatientID = B.PatientID
WHERE
    Ap.AppointmentDate >= '2023-01-01'
GROUP BY
    Month, Dr.DepartmentID, i.InventoryID
ORDER BY
    Month, Dr.DepartmentID;
```

Query 3. Disease Trend and Patient Condition Analysis Across Department

This query is designed to generate the dataset to be used in identifying trend in diseases based on demographics and history. The query makes used of the following data in the database.

- Patients:

  - DateOfBirth: Age is calculated using the difference between current date and DateOfBirth. Patients' ages are grouped as '0-17', '18-35', '36-60' and '60+' and average age is calculated for each group.
  - Address: Postal codes are scraped and used as 'Regions'
- MedicalRecord:
  - Diagnosis: categorized 'diabetes' and 'hypertension' as 'Chronics', 'infection' as 'Acute' and the rest as 'unspecified' and displayed them as 'ConditionStatus'
  - VisitDate: Extract the month to get VisitMonth
  - DoctorID: DoctorID in medical record is to link Patient data to Medical department, so that the record can be viewed together with the associated Medical department.

The columns in Output tables are AgeGroup, Gender, Region, VisitMonth, DepartmentName, ConditionDiagnosed, ConditionStatus, AverageAge. The following figures show the snapshots of the output data and SQL queries.

Figure 16:

Output of the SQL Queries for Disease Trend and Patient Condition Analysis

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

AgeGroup	Gender	Region	VisitMonth	DepartmentName	ConditionDiagnosed	ConditionStatus	AverageAge
0-17	Male	AR 38690	2023-05	Cardiology	Hypertension	Chronic	10.0000
0-17	Male	CT 80088	2023-08	Pediatrics	Common Cold	Unspecified	2.0000
0-17	Female	DC 77625	2023-06	Pediatrics	Common Cold	Unspecified	13.0000
0-17	Male	FL 56318	2023-04	Orthopedics	Diabetes Mellitus	Chronic	14.0000
0-17	Male	FL 56318	2024-06	Neurology	Upper Respiratory Infection	Acute	14.0000
0-17	Other	FM 96925	2023-11	obstetrics and gynaecology	Migraine	Unspecified	6.0000
0-17	Female	IA 94585	2023-03	Surgery	Migraine	Unspecified	13.0000

Vertical Output | Result 19

Figure 17:

SQL Query for Disease Trends and Patient Condition Analysis

```
/*--------- Query 3: (POINT 3) Disease Trends and Patient Condition Analysis Across Departments
(2023-2024) ------------ */

SELECT
    CASE
        WHEN YEAR(CURDATE()) - YEAR(P.DateOfBirth) < 18 THEN '0-17'
        WHEN YEAR(CURDATE()) - YEAR(P.DateOfBirth) BETWEEN 18 AND 35 THEN '18-35'
        WHEN YEAR(CURDATE()) - YEAR(P.DateOfBirth) BETWEEN 36 AND 60 THEN '36-60'
        ELSE '60+'
    END AS AgeGroup,
    P.Gender,
    TRIM(SUBSTRING_INDEX(Address, ',', -1)) AS Region,
    DATE_FORMAT(Mr.VisitDate, '%Y-%m') AS VisitMonth,
    COALESCE(Dept.Name, 'Unknown Department') AS DepartmentName,
    COALESCE(Mr.Diagnosis, 'Unknown Diagnosis') AS ConditionDiagnosed,
    CASE
        WHEN Mr.Diagnosis LIKE '%diabetes%' OR Mr.Diagnosis LIKE '%hypertension%' THEN 'Chronic'
        WHEN Mr.Diagnosis LIKE '%infection%' OR Mr.Diagnosis LIKE '%acute%' THEN 'Acute'
        ELSE 'Unspecified'
    END AS ConditionStatus,
    AVG(YEAR(CURDATE()) - YEAR(P.DateOfBirth)) AS AverageAge
FROM
    Patients AS P
INNER JOIN
    medical_records AS Mr ON P.PatientID = Mr.PatientID
LEFT JOIN
    Doctors AS Dr ON Mr.DoctorID = Dr.DoctorID
LEFT JOIN
    Departments AS Dept ON Dr.DepartmentID = Dept.DepartmentID
WHERE
    Mr.VisitDate BETWEEN '2023-01-01' AND '2024-12-31'
GROUP BY
    AgeGroup, Gender, Region, VisitMonth, DepartmentName, ConditionDiagnosed, ConditionStatus
ORDER BY
    AgeGroup, Region, VisitMonth, DepartmentName;
```

Query 4. Analysis of Medication Effectiveness

This query is designed to generate the dataset to be used in determining the effectiveness of medication based on patient data and treatment duration. The query makes used of the following data in the database.

- Patients:
    - DateOfBirth: It is used to calculate patient's age.
    - PatientID: to link to patient's medical records
- MedicalRecord:
    - CreateAt, UpdatedAt: Difference of these dates are used as TreatmentDuartionDay.
    - If TreatmentDuartionDay ≤ 15, EffectivenessRating is set to 'Highly Effective', between 16 to 30 is set to 'Moderate Effective', and 'Low Effective' is set otherwise.

The queries extract the data with date larger than 2023-01-01 and the prescribed medication is not Null. The columns in Output table are PatientID, FirstName, LastName, Age, Gender, ConditionDiagnosed, TreatmentProvided, PrescribedMedications, EffectivenessRating, TreatmentStartDate, TreatmentEndDate, and TreatmentDurationDays. The following figures show the snapshots of the output data and SQL queries.

Figure 18:

Output of the SQL Queris for Analysis of Medication Effectiveness

PatientID	FirstName	LastName	Age	Gender	ConditionDiagnosed	TreatmentProvided	PrescribedMedications	EffectivenessRating	TreatmentStartDate	TreatmentEndDate	TreatmentDurationDays
45	Katherine	Curtis	2.1	Male	Upper Respiratory Infection	Bronchodilators	Diclofenac,Doxycycline,Fluticasone,Metoprolol,...	Moderate Effectiveness	2024-12-10 01:27:23	2024-12-29 07:05:28	19
66	Megan	Perez	2.5	Male	Acute Bronchitis	Pain management and rest	Naproxen	Moderate Effectiveness	2024-09-14 11:15:47	2024-10-07 23:21:50	23
9	Belinda	Chaney	49.8	Female	Diabetes Mellitus	7-day course of antibiotics	Fluoxetine	Moderate Effectiveness	2024-11-30 09:24:35	2024-12-25 20:24:01	25
28	Heather	Allen	69.7	Other	Asthma	Pain management and rest	Baclofen	Moderate Effectiveness	2023-06-25 21:54:13	2023-07-13 23:41:34	18
61	Donna	Alvarado	78.6	Female	Diabetes Mellitus	Physical therapy and pain relief	Zolpidem	Moderate Effectiveness	2024-04-06 22:25:49	2024-05-05 00:59:58	29
68	Melanie	Chen	80.2	Other	Diabetes Mellitus	Bronchodilators	Famotidine,Hydrocodone	Moderate Effectiveness	2024-09-29 07:07:34	2024-10-21 17:13:09	22
45	Katherine	Curtis	2.1	Male	Arthritis	Use of inhaler as needed	Benazepril,Isosorbide	Low Effectiveness	2023-07-24 12:29:56	2024-10-29 11:52:00	463
82	Evan	Barnett	2.9	Other	Chronic Back Pain	Physical therapy and pain relief	Zonisamide	Low Effectiveness	2023-03-27 10:42:20	2024-08-30 06:45:44	522

Figure 19:

SQL Query for Analysis of Medication Effectiveness

```sql
/*--------- Query 4: (POINT 4) Analysis of Medication Effectiveness Based on Patient Data and
Treatment Duration  ------------ */

SELECT
    P.PatientID,
    P.FirstName,
    P.LastName,
    ROUND(DATEDIFF(CURDATE(), P.DateOfBirth) / 365.25, 1) AS Age,
    P.Gender,
    Mr.Diagnosis AS ConditionDiagnosed,
    Mr.TreatmentPlan AS TreatmentProvided,
    pm.precribed_medication AS PrescribedMedications,
    CASE
        WHEN DATEDIFF(Mr.UpdatedAt, Mr.CreatedAt) <= 15 THEN 'High Effectiveness'
        WHEN DATEDIFF(Mr.UpdatedAt, Mr.CreatedAt) BETWEEN 16 AND 30 THEN 'Moderate Effectiveness'
        ELSE 'Low Effectiveness'
    END AS EffectivenessRating,
    Mr.CreatedAt AS TreatmentStartDate,
    Mr.UpdatedAt AS TreatmentEndDate,
    DATEDIFF(Mr.UpdatedAt, Mr.CreatedAt) AS TreatmentDurationDays
FROM
    Patients AS P
JOIN
    medical_records AS Mr ON P.PatientID = Mr.PatientID
LEFT JOIN
    prescribe_med AS pm ON Mr.PrescriptionID = pm.PrescriptionID
WHERE
    Mr.CreatedAt >= '2023-01-01' AND pm.precribed_medication IS NOT NULL
ORDER BY
    EffectivenessRating DESC, Age, Gender;
```

Query 5. Insurance Claim Prediction

This query is designed to generate the dataset to be used in predicting whether an insurance claim will be approved or rejected based on Patient's age and Chronic conditions. The following data from the database are used in the query to generate the dataset.

- Patients
    - DateOfBirth: it is used to calculate patient's age
- MedicalRecord:

- Diagnosis: The diagnosis are merged together with a comma separator for each patient and displayed as ‘ConditionDiagnosed’

The data from insurance, billings, medical records and patient tables are merged and the output the columns PatientID, Age, ConditionDiagnosed, InsuranceID, PolicyNumber, PolicyProvider, ClaimAmount, BillingID, BillAmount (TotalAmount from billings), and ClaimStatus. The following figures show the snapshots of the output data and SQL queries.

Figure 20:

Output of SQL Query for Insurance Claim Prediction

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

PatientID	Age	ConditionDiagnosed	InsuranceID	PolicyNumber	Provider	ClaimAmount	BillingID	BillAmount	ClaimStatus
1	20.8	Hypertension, Arthritis, Asthma	1	PN123456789	HealthFirst	1500.50	1	237.51	Approved
1	20.8	Hypertension, Arthritis, Asthma	2	PN987654321	CareHealth	2300.00	2	334.55	Pending
2	82.5	Common Cold	3	PN111223344	MedPlus	1500.00	3	436.31	Rejected
2	82.5	Common Cold	4	PN555666777	WellCare	1200.25	4	396.54	Approved
2	82.5	Common Cold	5	PN333222111	MediHelp	500.75	5	209.97	Pending
3	61.6	Diabetes Mellitus	6	PN444888222	CareMed	1800.00	6	234.79	Approved
3	61.6	Diabetes Mellitus	7	PN123987654	HealthPartners	2000.00	7	209.25	Pending
3	61.6	Diabetes Mellitus	8	PN333555777	WellbeingCo	1100.00	8	355.63	Approved
4	15.8	Upper Respiratory Infection	9	PN888777666	TrueCare	1800.50	9	149.91	Pending
4	15.8	Upper Respiratory Infection	10	PN123456789	FitCare	2200.75	10	128.89	Approved

Figure 21:

SQL Query for Insurance Claim Prediction

```sql
/*--------- Q5: Insurance Claim Prediction ------------*/
-- Q5: Query with CTE
WITH Patient_mr_info1 AS(
SELECT
    p.PatientID,
    ROUND(DATEDIFF(CURDATE(), p.DateOfBirth) / 365.25, 1) AS Age,
    GROUP_CONCAT(mr.Diagnosis SEPARATOR ', ') AS ConditionDiagnosed
FROM healthcaresystem.patients AS p
    JOIN healthcaresystem.medical_records AS mr ON p.PatientID = mr.PatientID
GROUP BY p.PatientID)
SELECT
    b.PatientID,
    ptmr.Age,
    ptmr.ConditionDiagnosed,
    ins.InsuranceID,
    ins.PolicyNumber,
    ins.Provider,
    ins.ClaimAmount,
    b.BillingID,
    b.TotalAmount AS BillAmount,
    ins.ClaimStatus
FROM healthcaresystem.billings AS b
    JOIN healthcaresystem.insurance AS ins ON b.InsuranceID = ins.InsuranceID
    JOIN Patient_mr_info1 AS ptmr ON b.PatientID = ptmr.PatientID
ORDER BY b.PatientID;
```

Query 6: Staffing and Schedule Optimization

This query is designed to generate the dataset to be used in analysis to optimize the staffing and scheduling. Staffs' schedules are based on available appointment data, staff roles and department data. The following data from the database are used in the query to generate the dataset.

- Appointments:
    - Appointment Date, DoctorID: DoctorID is used to link with department table to generate how many appointments are made per day for each department. A view named 'department_load' is created to simplify the further query.
- Staffs:
    - StaffID, Department: StaffID is counted for each department and used as CTE. In this query, only Role value 'nurse' are considered as an example.

The query make uses of these two tables and generate output to compare the department load (number of appointment per day) and number of nurses for each department. The output columns of the query are DepartmentID, AppointmentDate, number_of_appointment, and number_of_nurses. The following figures show the snapshots of the output data and SQL queries.

Figure 22:

Output of SQL Query for Staffing and Scheduling Optimization Analysis

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

DepartmentID	AppointmentDate	number_of_appointment	number_of_nurses
1	2024-01-01 10:00:00	1	5
1	2024-01-12 11:00:00	1	5
1	2024-01-20 19:00:00	1	5
1	2024-01-21 10:00:00	1	5
1	2024-02-02 11:00:00	1	5
1	2024-02-10 19:00:00	1	5
1	2024-02-11 10:00:00	1	5
3	2024-01-16 15:00:00	1	1
3	2024-02-06 15:00:00	1	1
4	2024-01-06 15:00:00	1	5

Figure 23:

SQL Query for Staffing and Scheduling Optimization Analysis

```
/*---------- Q6: Staffing Scheduling Optimization(only nurses are considered here) ----
/*------ Get number of Appointment per day for department */
CREATE  VIEW department_load AS(
    SELECT
        dept.DepartmentID,
        appt.AppointmentDate,
        COUNT(appt.AppointmentID) AS number_of_appointment
    FROM appointments AS appt
        JOIN doctors AS doc USING(DoctorID)
        JOIN departments AS dept ON doc.DepartmentID = dept.DepartmentID
    WHERE NOT appt.Status = 'Cancelled'
    GROUP BY dept.DepartmentID,appt.AppointmentDate
    ORDER BY dept.DepartmentID,appt.AppointmentDate);
/*------------ compare with staff load ---------------*/
WITH staff_avail AS(
    SELECT
        DepartmentID,
        COUNT(StaffID) AS number_of_nurses
    FROM staffs
    WHERE Role = 'nurse'
    GROUP BY DepartmentID
)
SELECT
    dl.DepartmentID,
    dl.AppointmentDate,
    dl.number_of_appointment,
    s_av.number_of_nurses
FROM staff_avail AS s_av
    JOIN department_load AS dl ON dl.DepartmentID = s_av.DepartmentID;
```

Query 7. Prediction of Cancel or No-show appointment

This query is designed to generate the dataset to be used in prediction of cancelled or no-show appointment by analyzing appointment history. In this query, the prediction is made based on in which region the patient resides. The following data from the database are used in the query to generate the dataset.

- Appointments
    - Status: Number of appointments are counted for each patient with status ‘Cancelled’
- Patients:
    - Address: Postal code is scraped from Address and set as Region.

The output columns of this query are PatientID, no_of_missed_appointment, Region. The following figures show the snapshots of the output data and SQL queries.

Figure 24:

Output of the SQL Query for Predicting Cancelled Appointments

Result Grid | Filter Rows: | Export:

PatientID	no_of_missed_appointment	Region
3	1	FM 89199
8	1	TN 25371
13	1	MT 28676
18	1	KS 10215
23	1	FL 56318
28	1	VA 61944
38	1	CO 65064
43	1	ME 15708

Figure 25:

SQL Query for Predicting Cancelled Appointments

```
/*--------------- Q7: Cancelled Appointments ------------------*/
WITH patient_postal AS(
    SELECT
        PatientID,
        TRIM(SUBSTRING_INDEX(Address, ',', -1)) AS Region
    FROM patients
)
SELECT
    appt.PatientID,
    COUNT(AppointmentID) AS no_of_missed_appointment,
    pp.Region
FROM appointments AS appt
    JOIN patient_postal AS pp ON pp.PatientID = appt.PatientID
WHERE appt.Status = 'Cancelled'
GROUP BY appt.PatientID;
```

Query 8. Financial Analysis and Cost Prediction

This query is designed to generate the dataset to predict hospital revenue and patient treatment cost based on the available data in billings, insurance and medical record of the database.

Predicted Treatment Cost Prediction

- Medical Record:
- CreatedAt , UpdatedAt : For each medical reords, Treatment duration are calculated by calculating the difference of CreatedAt and UpdatedAt in days
- PatientID: Each patient ID, Max days of treatment in medical record are used to determine Chronic or normal conditions. If (treatment duration) ≤ ROUND(365/2,0) then Normal, otherwise Chronic.
- Insurance Data:

- ClaimAmount: Based on the Chronic condition status, Treatment cost is calculated using the ClaimAmount. If Chronic condition then TreatmentCost = 1.2x Claim Amount, else TreatmentCost = Claim Amount.

Revenue Prediction:

- Billing: Total_Amount, Payment_Status
- Insurance:
  - Claim_Amount, Claim_Status:

    If Claim_Status = Approved:

    - The hospital receives the full ClaimAmount from the insurance.
    - If the Payment_Status is Paid, the patient also pays any remaining balance between the Total_Amount and ClaimAmount.

      Revenue = Claim_Amount + (Total_Amount – Claim_Amount)
    - If the Payment_Status is Pending, only the Claim_Amount is considered, as the patient hasn't paid the remaining balance.

      Revenue = Claim_Amount

    If Claim_Status = Pending

    - 80% of the Claim_Amount is expected to be reimbursed by the insurance provider
    - If the Payment_Status is Paid, the patient pays any remaining balance (Total_Amount – Claim_Amount).

      Revenue = 0.8 * Claim_Amount + (Total_Amount – Claim_Amount)
    - If the Payment_Status is Pending, only 80% of the ClaimAmount is considered, as the patient hasn't paid the remaining balance.

      Revenue = 0.8 * Claim_Amount

    If Claim_Status = Rejected

1. Since the insurance claim is rejected, the revenue comes entirely from the Total_Amount paid by the patient

   Revenue = Total_Amount

2. Payment_Status = Pending, then Revenue = 0

The output columns of this query are PatientID, ChronicConditions, InsuranceID, ClaimStatus, ClaimAmount, BillAmount, PaymentStatus, PredictedTreatmentCost, PredictedRevenue. The following figures show the snapshots of the output data and SQL queries.

Figure 26:

Output of SQL query for Financial Analysis and Cost Prediction

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

PatientID	ChronicConditions	InsuranceID	ClaimStatus	ClaimAmount	BillAmount	PaymentStatus	PredictedTreatmentCost	PredictedRevenue
1	Yes	1	Approved	1500.50	237.51	Pending	1800.600	1500.500
1	Yes	2	Pending	2300.00	334.55	Paid	2760.000	-125.450
2	No	3	Rejected	1500.00	436.31	Pending	1500.000	0.000
2	No	4	Approved	1200.25	396.54	Pending	1200.250	1200.250
2	No	5	Pending	500.75	209.97	Pending	500.750	400.600
3	Yes	6	Approved	1800.00	234.79	Pending	2160.000	1800.000
3	Yes	7	Pending	2000.00	209.25	Paid	2400.000	-190.750
3	Yes	8	Approved	1100.00	355.63	Pending	1320.000	1100.000
4	No	9	Pending	1800.50	149.91	Paid	1800.500	-210.190

Figure 27:

SQL Query for Financial Analysis and Cost Prediction

```sql
/*--------------- Q8: Financial Analysis and Cost Prediction  -----------------------*/
/*--------- Treatment Cost  and Revenue Predicition --------*/
WITH PatientChornicCond AS(
    SELECT
        mr.PatientID,
        MAX(DATEDIFF(mr.UpdatedAt, mr.CreatedAt)) AS Treatment_duration,
        CASE
            -- considered as chronic condition if the treatment period is more than 6 months
            WHEN MAX(DATEDIFF(mr.UpdatedAt, mr.CreatedAt)) >  ROUND(365/2,0) THEN 'Yes'
            ELSE 'No'
        END AS ChronicConditions
    FROM medical_records AS mr
    GROUP BY mr.PatientID
    ORDER BY mr.PatientID)
SELECT DISTINCT
    i.PatientID,
    pc.ChronicConditions,
    i.InsuranceID,
    i.ClaimStatus,
    i.ClaimAmount,
    b.TotalAmount,
    b.PaymentStatus,
    -- Calculate Predicted Treatment Cost: Increase cost if chronic conditions exist
    CASE
        WHEN pc.ChronicConditions = 'Yes' THEN i.ClaimAmount * 1.2  -- 20% higher cost for chronic conditions
        ELSE i.ClaimAmount
    END AS PredictedTreatmentCost,
    -- Revenue Calculation
    CASE
        -- For approved claims: Total revenue includes ClaimAmount + patient payment (if Paid)
        WHEN i.ClaimStatus = 'Approved' AND b.PaymentStatus = 'Paid' THEN i.ClaimAmount + (b.TotalAmount - i.ClaimAmount)
        WHEN i.ClaimStatus = 'Approved' AND b.PaymentStatus = 'Pending' THEN i.ClaimAmount  -- Insurance revenue only

        -- For pending claims: Assume 80% of ClaimAmount + patient payment (if Paid)
        WHEN i.ClaimStatus = 'Pending' AND b.PaymentStatus = 'Paid' THEN (i.ClaimAmount * 0.8) + (b.TotalAmount - i.ClaimAmount)
        WHEN i.ClaimStatus = 'Pending' AND b.PaymentStatus = 'Pending' THEN i.ClaimAmount * 0.8  -- Expected insurance revenue only

        -- For rejected claims: Total revenue comes only from patient payment (if Paid)
        WHEN i.ClaimStatus = 'Rejected' AND b.PaymentStatus = 'Paid' THEN b.TotalAmount
        WHEN i.ClaimStatus = 'Rejected' AND b.PaymentStatus = 'Pending' THEN 0  -- No revenue for unpaid rejected claims

        ELSE 0  -- Default case
    END AS PredictedRevenue
FROM
    insurance i
    JOIN PatientChornicCond AS pc ON i.PatientID = pc.PatientID
    JOIN billings b ON i.PatientID = b.PatientID
WHERE
    i.ClaimAmount IS NOT NULL;
```

Advanced Concepts Implemented

The inclusion of Common Table Expressions (CTEs), CASE Statements, JOINs, and Indexing was essential in improving both the functionality and efficiency of the database system

VIEW

VIEW behaves like tables in a database. Instead of storing data like database, they store the query that generates the data. View generate data dynamically based on the query that it stores whenever it is referenced. VIEW allows the user to reference from different set of queries unlike WITH statement CTE. It is useful when different set of queries that have common sub-query. For example, in Query 1, a view named "prescribe_med" is created to combine the medicine prescribed to a patient and it is reference when generating the patient treatment prediction dataset to obtain the prescribe medication to each patient. And it is again referenced in Query 4 to analyze the effectiveness of medication without needing to write the query again.

Figure 28:

'prescribe_med' VIEW to Generate the Prescribed Medication to Each Patient

```
CREATE VIEW prescribe_med AS(
SELECT
    pd.PrescriptionID,
    # pd.MedicationID,
    GROUP_CONCAT( m.Name ORDER BY m.Name SEPARATOR ',') AS precribed_medication
FROM prescription_detail as pd
    JOIN medication AS m ON pd.MedicationID = m.MedicationID
GROUP BY pd.PrescriptionID
ORDER BY pd.PrescriptionID) ;
```

Figure 29:

'prescribe_med' VIEW Referenced in Query 1

```
    Patients AS P
JOIN
    medical_records AS Mr ON P.PatientID = Mr.PatientID
LEFT JOIN
    prescription AS Pr ON Mr.PrescriptionID = Pr.PrescriptionID
JOIN
    prescribe_med AS pm ON pm.PrescriptionID = pr.PrescriptionID
WHERE
    Mr.CreatedAt >= '2023-01-01'
ORDER BY
    Age, Gender, TreatmentOutcome;
```

Figure 30:

'prescribe_med' VIEW Referenced in Query 4

```
    END AS EffectivenessRating,
    Mr.CreatedAt AS TreatmentStartDate,
    Mr.UpdatedAt AS TreatmentEndDate,
    DATEDIFF(Mr.UpdatedAt, Mr.CreatedAt) AS TreatmentDurationDays
FROM
    Patients AS P
JOIN
    medical_records AS Mr ON P.PatientID = Mr.PatientID
LEFT JOIN
    prescribe_med AS pm ON Mr.PrescriptionID = pm.PrescriptionID
WHERE
    Mr.CreatedAt >= '2023-01-01' AND pm.precribed_medication IS NOT NULL
ORDER BY
    EffectivenessRating DESC, Age, Gender;

/*--------- Q5: Insurance Claim Prediction ------------*/
```

Common Table Expressions (CTE)

The use of Common Table Expressions (CTE) was employed to break down complex queries into simpler and more understandable parts, such as identifying the effectiveness of treatments and generating summaries of inventory usage in different hospital departments.

The design of the queries was simplified, improving the clarity of the code, leading to a better understanding of it, which made it easier to maintain and reuse parts of the analysis without having to repeat calculations. For example, a CTE is used to use Aggregate function

MAX to decide the chronic condition and later joined in normal query without aggregation to calculate predicted treatment cost.

Figure 31:

CTE Created to Use Aggregate Function Separately to Simplify the Query

```
/*--------- Treatment Cost  and Revenue Predicition --------*/
WITH PatientChornicCond AS(
    SELECT
        mr.PatientID,
        MAX(DATEDIFF(mr.UpdatedAt, mr.CreatedAt)) AS Treatment_duration,
        CASE
            -- considered as chronic condition if the treatment period is more than 6 months
            WHEN MAX(DATEDIFF(mr.UpdatedAt, mr.CreatedAt)) >  ROUND(365/2,0) THEN 'Yes'
            ELSE 'No'
        END AS ChronicConditions
    FROM medical_records AS mr
    GROUP BY mr.PatientID
    ORDER BY mr.PatientID)
SELECT DISTINCT
```

CASE Statement

Case statements are the expression which are useful for implementing conditional logic within the queries. In this project, CASE statements are used for categorization and classifications throughout this project. For example, for the Query 3 of this project, CASE statement is used for grouping patients into age ranges (Age Group) and classifying diagnoses as "Chronic," "Acute," or "Unspecified." This allows users to evaluate different conditions and return corresponding value and making it easier to manipulate existing data to additional data.

Figure 32:

Example of CASE Statement Usage in Query 3

```
SELECT
    CASE
        WHEN YEAR(CURDATE()) - YEAR(P.DateOfBirth) < 18 THEN '0-17'
        WHEN YEAR(CURDATE()) - YEAR(P.DateOfBirth) BETWEEN 18 AND 35 THEN '18-35'
        WHEN YEAR(CURDATE()) - YEAR(P.DateOfBirth) BETWEEN 36 AND 60 THEN '36-60'
        ELSE '60+'
    END AS AgeGroup,
    P.Gender,
    TRIM(SUBSTRING_INDEX(Address, ',', -1)) AS Region,
    DATE_FORMAT(Mr.VisitDate, '%Y-%m') AS VisitMonth,
    COALESCE(Dept.Name, 'Unknown Department') AS DepartmentName,
    COALESCE(Mr.Diagnosis, 'Unknown Diagnosis') AS ConditionDiagnosed,
    CASE
        WHEN Mr.Diagnosis LIKE '%diabetes%' OR Mr.Diagnosis LIKE '%hypertension%' THEN 'Chronic
        WHEN Mr.Diagnosis LIKE '%infection%' OR Mr.Diagnosis LIKE '%acute%' THEN 'Acute'
        ELSE 'Unspecified'
    END AS ConditionStatus,
```

JOIN (INNER, LEFT JOIN)

Joins are fundamental concept in working with relational databases. They act allows users to combine data from multiple tables by acting as the bridge based on defined KEYs. There are three types of JOINs, such as INNER JOIN, RIGHT JOIN and LEFT JOIN. In this project, INNER JOIN(JOIN) and LEFT JOIN are used to combine the tables to manipulate the required data in preparing sample datasets. For example, in Query 2, a LEFT JOIN was used to combine data from inventories and medical appointments.

Figure 33:

Using JOIN to Merge Data in Query 2 to Retrieve Inventory Resource Information

```
FROM
    appointments AS Ap
JOIN
    doctors AS Dr ON Ap.DoctorID = Dr.DoctorID
LEFT JOIN
    Inventory AS i ON Dr.DepartmentID = i.DepartmentID
LEFT JOIN
    billings AS B ON Ap.PatientID = B.PatientID
WHERE
    Ap.AppointmentDate >= '2023-01-01'
GROUP BY
    Month, Dr.DepartmentID, i.InventoryID
ORDER BY
    Month, Dr.DepartmentID;
```

Aggregate Functions

The aggregate functions summarize the data by performing calculations on group values and returning a single result. Aggregation functions such as SUM, AVG, COUNT, MAX, and the use of GROUP BY optimized the organization of data within the queries. Aggregate functions are used throughout this project to obtain average age within age group, to count total number of appointments made in each department, to get the maximum number of treatment days in each patient, etc. For example, in Query 8, it was used to calculate the maximum duration of treatments and classify patients as chronic or non-chronic. These functions allowed for detailed analysis by segmenting the data into specific groups without the need for additional queries, making the analysis more efficient.

Figure 34:

Use of Aggregate function MAX together with GROUP BY in Query 8

```
WITH PatientChornicCond AS(
    SELECT
        mr.PatientID,
        MAX(DATEDIFF(mr.UpdatedAt, mr.CreatedAt)) AS Treatment_duration,
        CASE
            -- considered as chronic condition if the treatment period is more than 6 months
            WHEN MAX(DATEDIFF(mr.UpdatedAt, mr.CreatedAt)) >  ROUND(365/2,0) THEN 'Yes'
            ELSE 'No'
        END AS ChronicConditions
    FROM medical_records AS mr
    GROUP BY mr.PatientID
    ORDER BY mr.PatientID)
```

The commands listed in this sections were used according to the specific needs and purposes of each query.

Challenges and Solutions

1. Data consistency in generating dummy data to insert into database

- PatientID and DoctorID appears in appointment, medical record, billing and doctor_availability should be tallied.
- CreatedAt and UpdatedAt date are set to 2023-01-01 00:00:00 to 2024-12-31 23:59:59 to have consistency time frame and add constraints such as CreatedAt < UpdatedAt
- AI and Python scripts to generate dummy data and inserted using SQL script or CSV files.

2. Maintaining data consistency, understanding that a user has different appointments, different diagnoses, treatments, and visits with different specialists. Considering the entire scope was a challenge.

Solution: Primary and foreign keys were reviewed to link related tables (e.g., patients, appointments, and medical records), ensuring data integrity. Views were also created to consolidate critical patient information, allowing for consistent and efficient data retrieval without duplication.

Example:

During the construction and writing of each record, there was duplication of attributes errors.

Previous design: PatientID: Patient→Insurance, Patient →billing

Problem1: No consolidation between bill ID and insurance ID except patient ID. Therefore, no way of telling which bill is covered by which insurance.

Solution: Added InsuranceID foreign Key in billing Table.

Problem: After adding, since PatientID includes in both insurance and billing table while they are already related using InsuranceID. This results in return in duplication of data when using queries with joining tables.

Solution: Remove PatientID from Insurance table. Therefore, the relations will be Patient→billings (Key: PatientID), billings→Insurance (InsuranceID).

A few similar cases are found during the Query design phase and fixed the errors along the way.

Conclusion

The Healthcare System Database successfully integrates various aspects of healthcare operations, offering a robust solution for managing patient data, billing, insurance, and treatment records. Designed with scalability and data integrity in mind, the system ensures consistency through the use of primary and foreign keys while enabling seamless expansions for future needs.

Key accomplishments include:

- Optimized Data Management: Efficient table relationships and constraints prevent redundancy and maintain data integrity across the system.
- Advanced Querying: The use of CTEs, aggregate functions, and `JOIN` operations allows for complex data retrieval and meaningful analysis, such as treatment cost predictions and financial insights.

- Support for Predictive Analytics: The system generates structured datasets for machine learning tasks like claim prediction and staffing optimization, enhancing decision-making capabilities.
- Practical Implementation: Realistic dummy data and optimized SQL queries address real-world challenges such as financial analysis, resource allocation, and operational efficiency.

This project highlights the practical application of SQL in healthcare, demonstrating its ability to streamline data operations and support advanced analytics. The system sets a strong foundation for integrating emerging technologies like AI to further enhance patient care and decision-making.