Hospital Management System (HMS) SQL Project Documentation

1. Project Overview

This project implements a fully functional **Hospital Management System (HMS)** using **MySQL**. It is designed to manage core hospital operations, including patient admission, doctor appointments, room tracking, treatments, and comprehensive financial billing.

The project demonstrates advanced database concepts, real-world data modeling, and practical SQL usage, ensuring a **clean, normalized relational structure**.

Primary Goals Achieved

- **Database Design:** Creation of seven inter-related tables with established Primary and Foreign Key constraints.
- **Workflow Simulation:** Modeling real hospital workflows such as admission-to-billing, appointment scheduling, and room status updates.
- **SQL Proficiency:** Extensive use of CRUD (Create, Read, Update, Delete) operations, Joins, Aggregate Functions, Subqueries, Views, Stored Procedures, and Triggers.
- **Reporting:** Generation of key analytical reports (e.g., revenue trends, top patients by expense).

2. Database Schema Design (Tables)

The system is built on seven core tables, linked via Foreign Key constraints to maintain data integrity and consistency.

Table Name	Primary Key	Key Foreign Keys	Description
Doctors	doctor_id	N/A	Stores doctor profiles and specialization.
Patients	patient_id	N/A	Holds patient demographic and contact information.
Rooms	room_id	N/A	Manages room types (General, Private, ICU), status, and daily charges.
Admissions	admission_id	patient_id, room_id	Tracks patient hospital stays, admission, and discharge dates.
Appointments	appointment_id	patient_id, doctor_id	Records scheduled appointments and their status.

Table Name	Primary Key	Key Foreign Keys	Description
Treatments	treatment_id	patient_id, doctor_id	Stores diagnoses and prescribed medicines for patient visits.
Billing	bill_id	patient_id, admission_id	Stores financial records, total amounts, and payment status.

3. SQL Implementation Details

The following sections contain the complete, functional SQL code used to build and operate the HMS.

3.1. Schema Creation and Data Insertion

SQL

```
-- Database Initialization
CREATE DATABASE hospitaldb;
USE hospitaldb;
-- 1. DOCTORS TABLECREATE TABLE Doctors (
   doctor_id INT PRIMARY KEY,
   name VARCHAR(100),
   specialization VARCHAR(50),
   phone VARCHAR(15),
   email VARCHAR(100),
   joining_date DATE
);
INSERT INTO Doctors (doctor_id, name, specialization, phone, email, joining_date)VALUES
(1, 'Satoru Gojo', 'Neurosurgery', '9001200001', 'gojo@hospital.jp', '2021-05-14'),
(2, 'Suguru Geto', 'Psychiatry', '9001200002', 'geto@hospital.jp', '2020-09-20'),
(3, 'Kento Nanami', 'Cardiology', '9001200003', 'nanami@hospital.jp', '2022-02-10'),
(4, 'Yuji Itadori', 'Emergency Medicine', '9001200004', 'itadori@hospital.jp', '2023-01-
05'),
(5, 'Megumi Fushiguro', 'Orthopedics', '9001200005', 'megumi@hospital.jp', '2023-03-
22'),
(6, 'Nobara Kugisaki', 'Dermatology', '9001200006', 'nobara@hospital.jp', '2022-07-
17'),
(7, 'Maki Zenin', 'Ophthalmology', '9001200007', 'maki@hospital.jp', '2021-11-11'),
(8, 'Toge Inumaki', 'ENT Specialist', '9001200008', 'toge@hospital.jp', '2022-12-01'),
(9, 'Panda', 'Pediatrics', '9001200009', 'panda@hospital.jp', '2023-04-09'),
(10, 'Yuta Okkotsu', 'Oncology', '9001200010', 'yuta@hospital.jp', '2021-08-25');
-- 2. PATIENTS TABLECREATE TABLE Patients (
   patient_id INT PRIMARY KEY,
   name VARCHAR(100),
   gender VARCHAR(10),
   dob DATE,
   phone VARCHAR(15),
   address VARCHAR(255),
   blood_group VARCHAR(5),
```

```
  admission_date DATE
);
INSERT INTO Patients (patient_id, name, gender, dob, phone, address, blood_group,
admission_date)VALUES
(1, 'Tanjiro Kamado', 'Male', '2000-07-14', '9876501111', 'Tokyo, Japan', 'O+', '2025-
01-10'),
(2, 'Nezuko Kamado', 'Female', '2002-12-28', '9876502222', 'Tokyo, Japan', 'B+', '2023-
02-15'),
(3, 'Zenitsu Agatsuma', 'Male', '2001-09-03', '9876503333', 'Kyoto, Japan', 'A+', '2025-
03-05'),
(4, 'Inosuke Hashibira', 'Male', '2001-04-18', '9876504444', 'Osaka, Japan', 'O-', '2024-
01-25'),
(5, 'Kanao Tsuyuri', 'Female', '2002-05-19', '9876505555', 'Nagoya, Japan', 'AB+',
'2025-04-02'),
(6, 'Giyu Tomioka', 'Male', '1995-02-08', '9876506666', 'Tokyo, Japan', 'B-', '2020-02-
20'),
(7, 'Shinobu Kocho', 'Female', '1996-07-24', '9876507777', 'Yokohama, Japan', 'A+',
'2025-03-18'),
(8, 'Kyojuro Rengoku', 'Male', '1994-05-10', '9876508888', 'Sendai, Japan', 'O+', '2025-
01-30'),
(9, 'Tengen Uzui', 'Male', '1993-10-31', '9876509999', 'Osaka, Japan', 'AB-', '2021-02-
25'),
(10, 'Mitsuri Kanroji', 'Female', '1997-06-01', '9876510000', 'Kobe, Japan', 'B+', '2025-
04-15');
-- 3. APPOINTMENTS TABLECREATE TABLE Appointments (
  appointment_id INT PRIMARY KEY,
  patient_id INT,
  doctor_id INT,
  appointment_date DATE,
  appointment_time TIME,
  reason VARCHAR(255),
  status VARCHAR(20),
  FOREIGN KEY (patient_id) REFERENCES Patients(patient_id),
  FOREIGN KEY (doctor_id) REFERENCES Doctors(doctor_id)
);
INSERT INTO Appointments (appointment_id, patient_id, doctor_id, appointment_date,
appointment_time, reason, status)VALUES
(1, 1, 3, '2025-10-20', '10:00:00', 'Chest pain and fatigue', 'Completed'),
(2, 2, 1, '2025-10-22', '11:30:00', 'Frequent headaches', 'Scheduled'),
(3, 3, 6, '2025-10-21', '09:45:00', 'Skin allergy and rashes', 'Completed'),
(4, 4, 4, '2025-10-23', '14:15:00', 'Minor injury checkup', 'Scheduled'),
(5, 5, 5, '2025-10-19', '16:00:00', 'Back pain and stiffness', 'Completed'),
(6, 6, 8, '2025-10-25', '12:00:00', 'Throat irritation', 'Scheduled'),
(7, 7, 2, '2025-10-24', '10:30:00', 'Stress and anxiety', 'Cancelled'),
(8, 8, 10, '2025-10-26', '15:00:00', 'Follow-up for tumor screening', 'Scheduled'),
(9, 9, 7, '2025-10-18', '09:15:00', 'Eye irritation and dryness', 'Completed'),
(10, 10, 9, '2025-10-27', '13:00:00', 'Pediatric vaccination', 'Scheduled');
```

```
-- 4. ROOMS TABLECREATE TABLE Rooms (
   room_id INT PRIMARY KEY,
   room_type VARCHAR(50),
   status VARCHAR(20),
   charges_per_day DECIMAL(10,2)
);
INSERT INTO Rooms (room_id, room_type, status, charges_per_day)VALUES
(101, 'General', 'Occupied', 1500.00),
(102, 'General', 'Available', 1500.00),
(201, 'Private', 'Occupied', 3500.00),
(202, 'Private', 'Available', 3500.00),
(301, 'ICU', 'Occupied', 8000.00),
(302, 'ICU', 'Available', 8000.00);
-- 5. ADMISSIONS TABLECREATE TABLE Admissions (
   admission_id INT PRIMARY KEY,
   patient_id INT,
   room_id INT,
   admit_date DATE,
   discharge_date DATE,
   FOREIGN KEY (patient_id) REFERENCES Patients(patient_id),
   FOREIGN KEY (room_id) REFERENCES Rooms(room_id)
);
INSERT INTO Admissions (admission_id, patient_id, room_id, admit_date, discharge_date)VALUES
(1, 1, 101, '2025-01-10', NULL),
(2, 3, 201, '2025-03-05', NULL),
(3, 5, 202, '2025-04-02', NULL),
(4, 7, 301, '2025-03-18', NULL),
(5, 10, 102, '2025-04-15', NULL);
-- 6. TREATMENTS TABLECREATE TABLE Treatments (
   treatment_id INT PRIMARY KEY,
   patient_id INT,
   doctor_id INT,
   diagnosis VARCHAR(255),
   prescribed_medicines VARCHAR(255),
   treatment_date DATE,
   FOREIGN KEY (patient_id) REFERENCES Patients(patient_id),
   FOREIGN KEY (doctor_id) REFERENCES Doctors(doctor_id)
);
INSERT INTO Treatments (treatment_id, patient_id, doctor_id, diagnosis, prescribed_medicines,
treatment_date)VALUES
(1, 1, 4, 'Fever & Weakness', 'Paracetamol, ORS', '2025-01-11'),
(2, 3, 2, 'Anxiety Episodes', 'Vitamin B Complex, Sleep Aid', '2025-03-06'),
(3, 5, 6, 'Seasonal Allergies', 'Cetirizine', '2025-04-03'),
(4, 7, 8, 'Respiratory Infection', 'Azithromycin, Steam Inhalation', '2025-03-19'),
(5, 10, 5, 'Muscle Cramps', 'Magnesium Supplements', '2025-04-16');
-- 7. BILLING TABLECREATE TABLE Billing (
   bill_id INT PRIMARY KEY,
   patient_id INT,
```

```
  admission_id INT,
  total_amount DECIMAL(10,2),
  payment_status VARCHAR(20), -- Paid / Pending
  payment_date DATE,
  FOREIGN KEY (patient_id) REFERENCES Patients(patient_id),
  FOREIGN KEY (admission_id) REFERENCES Admissions(admission_id)
);
INSERT INTO Billing (bill_id, patient_id, admission_id, total_amount, payment_status,
payment_date)VALUES
(1, 1, 1, 6500.00, 'Paid', '2025-01-15'),
(2, 3, 2, 12500.00, 'Pending', NULL),
(3, 5, 3, 9800.00, 'Paid', '2025-04-05'),
(4, 7, 4, 18500.00, 'Pending', NULL),
(5, 10, 5, 4500.00, 'Paid', '2025-04-17');
-- Additional tables for advanced reporting:CREATE TABLE Medicine (
  med_id INT PRIMARY KEY,
  name VARCHAR(100),
  type VARCHAR(50)
);
CREATE TABLE Prescriptions (
  prescription_id INT PRIMARY KEY,
  patient_id INT,
  med_id INT,
  date_prescribed DATE,
  dosage VARCHAR(100),
  FOREIGN KEY (patient_id) REFERENCES Patients(patient_id),
  FOREIGN KEY (med_id) REFERENCES Medicine(med_id)
);
INSERT INTO Medicine VALUES
(1, 'Paracetamol', 'Tablet'),
(2, 'Amoxicillin', 'Capsule'),
(3, 'Ibuprofen', 'Tablet'),
(4, 'Cetrizine', 'Tablet'),
(5, 'Pantoprazole', 'Tablet');
INSERT INTO Prescriptions VALUES
(1, 1, 1, '2024-10-01', '500mg'),
(2, 2, 2, '2024-10-04', '250mg'),
(3, 1, 3, '2024-10-10', '200mg'),
(4, 3, 1, '2024-10-06', '500mg'),
(5, 4, 1, '2024-10-09', '500mg'),
(6, 5, 4, '2024-10-12', '10mg');
```

3.2. Core Query Operations

Basic Queries (SELECT, WHERE)

SQL

```
-- Display all patients admitted after 2024-01-01.SELECT *FROM AdmissionsWHERE
admit_date > '2024-01-01';
```

```
-- Retrieve all doctors who specialize in Cardiology.SELECT *FROM DoctorsWHERE specialization = 'Cardiology';
-- Find all available rooms of type ICU.SELECT *FROM RoomsWHERE room_type = 'ICU' AND status = 'Available';
```

Joins (Retrieving Combined Data)

SQL

```
-- List all patients with their assigned doctor names from Appointments.SELECT
   P.name AS patient_name,
   D.name AS doctor_name,
   A.appointment_date,
   A.statusFROM Appointments AJOIN Patients P ON A.patient_id = P.patient_idJOIN Doctors D ON A.doctor_id = D.doctor_id;
-- Show billing details with patient name and total bill amount.SELECT
   P.name AS patient_name,
   R.charges_per_day,
   B.total_amount,
   B.payment_statusFROM Billing BJOIN Patients P ON B.patient_id = P.patient_idJOIN Admissions A ON B.admission_id = A.admission_idJOIN Rooms R ON A.room_id = R.room_id;
```

Aggregate Functions (Data Summarization)

SQL

```
-- Count the number of patients currently admitted.SELECT
   COUNT(*) AS currently_admitted_patientsFROM AdmissionsWHERE discharge_date IS NULL;
-- Find the total revenue collected (sum of all paid bills).SELECT
   SUM(total_amount) AS total_revenueFROM BillingWHERE payment_status = 'Paid';
-- List doctors along with the number of patients they have treated.SELECT
   D.name AS doctor_name,
   COUNT(T.patient_id) AS total_patients_treatedFROM Doctors DLEFT JOIN Treatments T ON D.doctor_id = T.doctor_idGROUP BY D.doctor_id, D.name;
```

Subqueries (Advanced Filtering)

SQL

```
-- Find patients who have never booked an appointment.SELECT *FROM PatientsWHERE patient_id NOT IN (
   SELECT patient_id FROM Appointments
);
-- Find doctors who have not treated any patients yet.SELECT *FROM DoctorsWHERE doctor_id NOT IN (
   SELECT doctor_id FROM Treatments
);
```

3.3. CRUD Operations (Updates & Deletes)

SQL

```
-- Update the payment status of a bill after receiving payment.
UPDATE BillingSET payment_status = 'Paid',
   payment_date = CURDATE()WHERE bill_id = 4;
-- Change the phone number of a doctor.
```

```
UPDATE DoctorsSET phone = '9001209999'WHERE doctor_id = 2;
-- Delete all appointments that are marked as 'Cancelled'.SET SQL_SAFE_UPDATES =
0;DELETE FROM AppointmentsWHERE status = 'Cancelled';SET SQL_SAFE_UPDATES = 1;
```

4. Advanced Database Features (PL/SQL)

4.1. Triggers

A trigger is used to automatically update a room's status to **'Occupied'** whenever a new patient is inserted into the Admissions table.

```
SQL
DELIMITER $$
CREATE TRIGGER update_room_status_on_admission
AFTER INSERT ON AdmissionsFOR EACH ROWBEGIN
  UPDATE Rooms
  SET status = 'Occupied'
  WHERE room_id = NEW.room_id;END$$

DELIMITER ;
```

4.2. Views

A view, Patient_Treatment_Billing_View, simplifies complex join queries by creating a virtual table for quick reporting that combines patient, doctor, treatment, and billing information.

```
SQL
CREATE VIEW Patient_Treatment_Billing_View ASSELECT
  p.name AS patient_name,
  d.name AS doctor_name,
  t.diagnosis,
  b.total_amount AS bill_amountFROM Treatments tJOIN Patients p ON t.patient_id =
p.patient_idJOIN Doctors d ON t.doctor_id = d.doctor_idLEFT JOIN Billing b ON b.patient_id =
p.patient_id;
SELECT * FROM Patient_Treatment_Billing_View;
```

4.3. Stored Procedure for Billing Generation

The GenerateBill procedure automates the calculation and insertion of a bill based on the duration of the patient's stay (using DATEDIFF) and the daily room charges.

```
SQL
DROP PROCEDURE IF EXISTS GenerateBill;
DELIMITER $$
CREATE PROCEDURE GenerateBill(IN p_admission_id INT)BEGIN
  DECLARE v_patient_id INT;
  DECLARE v_room_id INT;
  DECLARE v_days INT;
  DECLARE v_charge DECIMAL(10,2);
```

```
  DECLARE v_total DECIMAL(10,2);
  DECLARE v_new_bill_id INT;

  -- Calculate next bill ID
  SELECT IFNULL(MAX(bill_id),0)+1 INTO v_new_bill_id FROM Billing;

  -- Retrieve admission details
  SELECT patient_id, room_id
  INTO v_patient_id, v_room_id
  FROM Admissions
  WHERE admission_id = p_admission_id;

  -- Calculate days stayed (using CURDATE() if not discharged)
  SELECT DATEDIFF(IFNULL(discharge_date,CURDATE()), admit_date)
  INTO v_days
  FROM Admissions
  WHERE admission_id = p_admission_id;

  -- Get room charges
  SELECT charges_per_day INTO v_charge
  FROM Rooms
  WHERE room_id = v_room_id;

  -- Calculate total
  SET v_total = v_days * v_charge;

  -- Insert the new bill record
  INSERT INTO Billing
  VALUES (v_new_bill_id, v_patient_id, p_admission_id, v_total, 'Pending', NULL);END $$

DELIMITER ;
-- Execution:CALL GenerateBill(1);
```

4.4. Transaction Control (Safe Insert)

The SafeInsertBill procedure uses a **transaction** to ensure atomicity. The bill record is inserted *only if* the corresponding admission ID exists in the Admissions table. If the admission does not exist, the transaction is **ROLLED BACK**.

SQL

```
DROP PROCEDURE IF EXISTS SafeInsertBill;

DELIMITER $$
CREATE PROCEDURE SafeInsertBill(IN p_admission_id INT)BEGIN
  DECLARE v_exists INT;
  DECLARE v_new_bill_id INT;

  START TRANSACTION;

  -- Check if admission exists
```

```
  SELECT COUNT(*) INTO v_exists
  FROM Admissions
  WHERE admission_id = p_admission_id;

  IF v_exists = 0 THEN
    ROLLBACK;
    SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'Admission does not exist';
  END IF;

  -- Safely generate next bill_id and lock table for update
  SELECT IFNULL(MAX(bill_id), 0) + 1
  INTO v_new_bill_id
  FROM Billing
  FOR UPDATE;

  -- Insert bill
  INSERT INTO Billing (bill_id, patient_id, admission_id, total_amount, payment_status,
payment_date)
  VALUES (v_new_bill_id, p_admission_id, p_admission_id, 12000.00, 'Pending', NULL);

  COMMIT;END$$

DELIMITER ;
-- Execution:CALL SafeInsertBill(3);
```

5. Reports and Analysis

These queries serve as key business intelligence reports for hospital administration.

Report	SQL Query
Top 5 Patients by Expense	SELECT p.name, SUM(b.total_amount) AS total_expense FROM Patients p JOIN Billing b ON p.patient_id = b.patient_id GROUP BY p.name ORDER BY total_expense DESC LIMIT 5;
Most Frequent Medicine	SELECT m.name, COUNT(DISTINCT pr.patient_id) AS total_patients FROM Medicine m JOIN Prescriptions pr ON m.med_id = pr.med_id GROUP BY m.name ORDER BY total_patients DESC LIMIT 1;
Monthly Revenue Trend	SELECT DATE_FORMAT(payment_date, '%Y-%m') AS month, SUM(total_amount) AS total_revenue FROM Billing WHERE payment_status = 'Paid' AND payment_date IS NOT NULL GROUP BY month ORDER BY month;
Max Revenue Room Type	SELECT R.room_type, SUM(B.total_amount) AS total_room_revenue FROM Billing B JOIN Admissions A ON B.admission_id = A.admission_id JOIN Rooms R ON A.room_id = R.room_id WHERE B.payment_status = 'Paid' GROUP BY R.room_type ORDER BY total_room_revenue DESC LIMIT 1;
Doctor with Most Appointments	SELECT d.name, COUNT(a.appointment_id) AS total_appointments FROM Doctors d JOIN Appointments a ON d.doctor_id = a.doctor_id GROUP BY d.name ORDER BY total_appointments DESC LIMIT 1;

6. Project Window