PROJECT

DBMS-LAB

Submitted By:

UMER RAZA(F2023332021)

MUHAMMAD BIN KHALID(F2023332022)

MUNEEB IRFAN(F2023332023)

NOMAN ALI(F2023332076)

Submitted To:

MAM.ZEENAT TANVEER

Bachelor of Science in Data Science Department of
Computer Science School of Science and
Technology
(THE UNIVERSITY OF MANAGEMENT AND TECHNOLOGY)

Hospital management system

Overview :

A Hospital Management System (HMS) is a comprehensive software solution designed to streamline the administrative, clinical, and financial operations of a hospital or healthcare facility. It integrates various functions into a unified system, enhancing efficiency, reducing errors, and improving patient care.

Key Features of a Hospital Management System

- Patient Registration and Admission:
- Electronic Medical Records (EMR):
- Appointment Scheduling:
- Billing and Invoicing:
- Pharmacy Management:
- Laboratory and Diagnostic Management:

Benefits of hospital managment system

Improved Operational Efficiency

- **Automation of Routine Tasks:** By automating administrative and clinical processes like patient registration, appointment scheduling, and billing, HMS reduces the workload on hospital staff, allowing them to focus on patient care.
- **Streamlined Workflow:** Integrates different hospital departments, ensuring seamless communication and coordination, which minimizes delays and errors.

Enhanced Patient Care

- **Quick Access to Patient Information:** Centralized storage of patient records, including medical history, diagnoses, and treatment plans, allows healthcare providers to access critical information quickly, improving decision-making and patient care.
- **Personalized Treatment:** Detailed patient data enables more personalized and accurate treatment plans.

Better Financial Management

- **Accurate Billing:** Automated billing systems ensure that all services, medications, and procedures are accurately billed, reducing the likelihood of errors and financial losses.
- **Efficient Revenue Cycle Management:** Streamlines the process of generating invoices, tracking payments, and managing insurance claims, leading to faster payment cycles.

Increased Data Accuracy

- **Minimized Human Error:** Digital records and automated processes reduce the risk of errors in patient data, billing, and reporting.
- **Consistent Record Keeping:** Ensures that all patient and hospital data are consistently recorded and updated, maintaining high levels of accuracy and reliability.

Database of a Hospital Management System (HMS)

The database of a Hospital Management System (HMS) is a structured collection of data specifically designed to store, manage, and retrieve all the information required to operate a hospital or healthcare facility. This database serves as the central repository for a wide range of data, including patient information, staff details, medical records, billing data, inventory levels, and more. It is the backbone of the HMS, enabling the integration and efficient management of hospital operations.

Benefits of Database of a Hospital Management System (HMS)

Centralized Data Storage

- **Unified Data Repository:** All patient records, medical histories, billing information, staff details, and other critical data are stored in a single, centralized database, making it easy to access and manage.
- **Elimination of Redundancies:** Centralization reduces data duplication, ensuring that the most current and accurate information is available across all departments.

Improved Data Accessibility

- **Real-Time Access:** Authorized users can access and update patient records, treatment plans, and other data in real-time, which is crucial for timely decision-making and patient care.
- **Multi-User Access:** Multiple users across different departments can access the database simultaneously without data conflicts, enhancing collaboration and efficiency.

Enhanced Data Security

- **Access Control:** The database system allows for role-based access, ensuring that only authorized personnel can view or modify sensitive information, thereby protecting patient privacy and hospital data integrity.
- **Data Encryption:** Stored data can be encrypted, providing an additional layer of security against unauthorized access or data breaches.

Efficient Data Management

- **Data Integrity:** Database management systems (DBMS) enforce rules and constraints that maintain data consistency and accuracy, reducing the risk of errors.
- **Automated Backups:** Regular backups can be automated, ensuring that data is always recoverable in case of hardware failures, cyber-attacks, or other disasters.

Scalability

- **Handling Growth:** A database-driven system can easily scale to accommodate growing amounts of data as the hospital expands its operations, patient base, or services.
- **Flexible Structure:** Databases can be modified or expanded without significant downtime, allowing the hospital to adapt to changing needs and technologies.

Streamlined Reporting and Analytics

- **Custom Reports:** The database can generate detailed and customizable reports on various aspects of hospital operations, such as patient demographics, treatment outcomes, financial performance, and resource utilization.
- **Data Analytics:** Advanced analytics can be performed on the data stored in the database to uncover trends, improve decision-making, and optimize hospital performance.

Creating a Tables :

Patients Table:

- patients
- patient_id (Primary Key)
- first_name
- last_name
- dob (Date ofBirth)
- gender
- address
- phone
- email
- emergency_contact
- insurance_details

Patients Table Relationships:

- **One-to-Many:**
  - **Patients ↔ Appointments:** One patient can have many appointments.
  - **Patients ↔ Medical Records:** One patient can have many medical records.

- **Many-to-Many** (through patient_doctor intermediate table):
    - **Patients ↔ Doctors:** A patient can be treated by many doctors, and a doctor can treat many patients.

Doctors Table

- Doctors
- doctor_id (Primary Key)
- first_name
- last_name
- specialization
- phon
- email
- department_id (Foreign Key referencing `departments`)

Doctors Table Relationships:

- **One-to-Many:**
    - **Doctors ↔ Appointments:** One doctor can have many appointments.
    - **Doctors ↔ Medical Records:** One doctor can have many medical records.
- **One-to-Many:**
    - **Doctors ↔ Departments:** A doctor is associated with one department (assuming one to-many for a simpler model).
- **Many-to-Many** (through patient_doctor intermediate table):
    - **Doctors ↔ Patients:** A doctor can treat many patients, and a patient can be treated by many doctors.
- **Many-to-Many** (through doctor_department intermediate table):
    - **Doctors ↔ Departments:** A doctor can belong to multiple departments, and a department can have multiple doctors.

Departments Table

- departments
- department_id (Primary Key)
- department_name
- location

Departments Table Relationships:

- **One-to-Many:**
    - **Departments ↔ Doctors:** One department can have many doctors.
- **Many-to-Many** (through doctor_department intermediate table):
    - **Departments ↔ Doctors:** A department can have many doctors, and a doctor can belong to multiple departments.

Appointments Table

- appointments
- appointment_id(Primary Key)
- patient_id (Foreign Key referencing `patients`)
- doctor_id (Foreign Key referencing `doctors`)
- appointment_date
- appointment_time
- status

Appointments Table Relationships

- **One-to-Many:**
    - **Appointments ↔ Patients:** One patient can have many appointments.
    - **Appointments ↔ Doctors:** One doctor can have many appointments.

Medical Records Table

- medicals
- record_id (Primary Key)
- patient_id (Foreign Key referencing `patients`)
- doctor_id (Foreign Key referencing `doctors`)
- diagnosis
- treatment
- prescription
- date

Medical Records Table Relationships

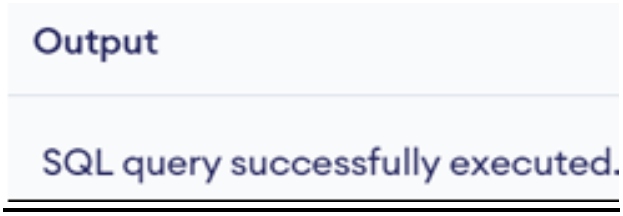
- **One-to-Many:**
    - **Medical Records ↔ Patients:** One patient can have many medical records.
    - **Medical Records ↔ Doctors:** One doctor can have many medical records.

SQL Querry:

```sql
-- Create Patients Table
CREATE TABLE patients (
    patient id INT PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    dob DATE NOT NULL,
    gender VARCHAR(10),
    address TEXT,
    phone VARCHAR(15),
    email VARCHAR(100),
    emergency_contact VARCHAR(100),
    insurance_details TEXT
);

-- Create Departments Table
CREATE TABLE departments (
    department_id INT PRIMARY KEY ,
    department name VARCHAR(100) NOT NULL,
    location VARCHAR(100)
);

-- Create Doctors Table
CREATE TABLE doctors (
    doctor_id INT PRIMARY KEY ,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    specialization VARCHAR(100),
    phone VARCHAR(15),
    email VARCHAR(100),
    department id INT,
    FOREIGN KEY (department_id) REFERENCES departments(department_id)
);

-- Create Appointments Table
CREATE TABLE appointments (
    appointment_id INT PRIMARY KEY ,
    patient_id INT,
    doctor_id INT,
    appointment_date DATE NOT NULL,
    appointment_time TIME NOT NULL,
    status VARCHAR(50),
```

```sql
    FOREIGN KEY (patient_id) REFERENCES patients(patient_id),
    FOREIGN KEY (doctor_id) REFERENCES doctors(doctor_id)
);

-- Create Medical Records Table
CREATE TABLE medical_records (
    record_id INT PRIMARY KEY ,
    patient_id INT,
    doctor_id INT,
    diagnosis TEXT,
    treatment TEXT,
    prescription TEXT,
    date DATE NOT NULL,
    FOREIGN KEY (patient_id) REFERENCES patients(patient_id),
    FOREIGN KEY (doctor_id) REFERENCES doctors(doctor_id)
);
```

Execution:



Appointments

appointment_id	patient_id	doctor_id	appointment_date	appointment_time	status
empty					

Departments

department_id	department_name	location
empty		

Doctors

doctor_id	first_name	last_name	specialization	phone	email	department_id
empty						

Medical_records

record_id	patient_id	doctor_id	diagnosis	treatment	prescription	date
empty						

Patients

patient_id	first_name	last_name	dob	gender	address	phone	email	emergency_
empty								

Normalization

- There are no repeating groups or arrays (1NF).
- All non-key attributes fully depend on the primary key, avoiding partial dependencies (2NF).
- There are no transitive dependencies, ensuring all attributes are directly dependent on the primary key (3NF).

RESULTS

Design is efficient, reduces redundancy, and maintains data integrity across the system.