Hospital Management System

Title: Hospital Management System

Author: Sindhu Krovvidi and Pratik Sahu

Date of Submission: July 29, 2024

Abstract

The Hospital Management System (HMS) is an innovative solution aimed at optimizing hospital operations by offering a comprehensive platform for the management of doctors, patients, nurses, appointments, and other critical hospital data. This system's main objective is to streamline administrative processes, reduce manual workload, and enhance hospital services efficiency. By leveraging the capabilities of SQLAlchemy and Flask, the HMS provides a robust and scalable framework that ensures data integrity, flexibility, and ease of use.

The system's design incorporates user-friendly interfaces for data entry and retrieval, enabling hospital staff to efficiently handle various operational tasks such as appointment scheduling, patient record management, and prescription handling. The integration of analytical tools like Dash and Plotly further enhances the system's functionality by providing insightful visualizations and real-time data analytics, which support informed decision-making and strategic planning.

Through the implementation of the HMS, hospitals can achieve significant improvements in service delivery, operational efficiency, and data accuracy. The system's scalability ensures that it can be adapted to meet the needs of healthcare facilities of different sizes and capacities, making it an asset for the healthcare industry. This report details the development process, methodologies used, and the results achieved, highlighting the system's potential for widespread adoption and its positive impact on hospital management practices.

Table of Contents

List of Figures and Tables

Introduction

Background information

The healthcare industry faces numerous challenges in managing hospital operations, including patient records, doctor schedules, and appointment bookings. A well-organized Hospital Management System can significantly improve efficiency and accuracy in handling these tasks.

Objectives of the report

- To provide a reliable system for storing and retrieving data related to doctors, patients, nurses, and appointments.
- To design an intuitive interface for hospital staff to easily navigate and manage records.
- To develop a scalable system that can handle increasing amounts of data as the hospital grows.
- To integrate analytical tools for generating insightful reports and visualizations, aiding in decision-making.

Scope of the Report

This report covers the development of the Hospital Management System, including the design, implementation, and testing of the database and associated applications. It also includes an analysis of the system's performance and recommendations for future enhancements.

Structure of the Report

The report is structured as follows:

- Literature Review: Summary of relevant existing research and discussion of methodologies.
- Methodology: Description of methods and techniques used in the project.
- Results: Presentation of the results obtained from the implementation.
- Discussion: Interpretation of results and comparison with existing research.
- Conclusion: Summary of key findings and suggestions for future research.

Literature Review

Hospital Management Systems (HMS) are designed to manage the comprehensive administrative, operations, and clinical aspects of hospitals. These systems have evolved significantly over the past few decades, driven by advancements in information technology and the increasing need for efficient healthcare management. An effective HMS integrates various functions such as patient registration, appointment scheduling, electronic health records (EHR), and reporting, ensuring seamless operation within a hospital environment.

Existing Research and Methodologies

Electronic Health Records (EHR)

EHR systems are a critical component of modern HMS, enabling the digitalization of patient records for easy access, and management. EHR systems improve healthcare quality by providing accurate and up-to-date patient information, reducing medical errors, and facilitating better coordination of care. EHRs also support data analytics for clinical decision support, public health reporting, and research.

Appointment Scheduling Systems

Efficient appointment scheduling is crucial for optimizing hospital operations and patient satisfaction. Research by Cayirli and Veral (2003) highlights the importance of dynamic scheduling algorithms that consider patient no-show probabilities, appointment durations, and provider availability. Advanced techniques such as machine learning and optimization algorithms have been employed to enhance scheduling efficiency and reduce waiting times.

Reporting and Analytics

Analytics and reporting capabilities are integral to HMS, providing insights into hospital performance, patient outcomes, and operational efficiency. Visualization tools such as dashboards facilitate real-time monitoring of key performance indicators (KPIs) and support strategic planning. Advanced analytics techniques, including machine learning, natural language processing, and big data technologies, have been employed to analyze large volumes of healthcare data. However, the effective use of these techniques requires addressing challenges related to data quality, integration, and interpretation.

Comparative Analysis of Technologies and Web Frameworks

Relational databases, such as MySQL, PostgreSQL, and SQLite, are widely used in HMS for their robust data integrity, support for complex queries, and transactional consistency. As highlighted by Elmasri and Navathe (2015), the relational model's structured schema and SQL support make it suitable for managing structured healthcare data. However, relational databases may face scalability issues when handling large volumes of unstructured data.

Web frameworks such as Flask, Django, and Spring Boot facilitate the rapid development of HMS applications by providing pre-built modules, security features, and integration capabilities. Flask, a lightweight Python framework, is favored for its simplicity and flexibility. It supports various extensions for database integration, authentication, and API development.

Gaps in the Literature

Despite significant advancements in HMS technologies, several gaps remain in the literature:

1. **Interoperability**: Many studies emphasize the need for interoperable systems that can seamlessly exchange data across different healthcare providers and systems. Research suggests the adoption of standardized APIs and data formats to enhance interoperability.
2. **Data Security and Privacy**: Ensuring the security and privacy of sensitive medical data is a persistent challenge. Studies by Williams and Woodward (2015) call for robust encryption methods, access controls, and compliance with regulations such as HIPAA and GDPR.
3. **User Acceptance and Training**: The successful implementation of HMS depends on user acceptance and adequate training. Researchers highlights the importance of user-friendly interfaces, comprehensive training programs, and ongoing support to ensure smooth adoption.
4. **Integration of Advanced Technologies**: While advanced technologies such as machine learning and big data analytics offer significant potential, their integration into HMS requires addressing issues related to data quality, algorithm transparency, and ethical considerations.

Methodology

The project follows a systematic approach to develop the Hospital Management System. The key steps include:

1. Requirements gathering and analysis.
2. Database design and schema development.
3. Implementation of the database using SQLAlchemy with Flask.
4. Development of user interfaces for hospital management systems.
5. Integration of analytical tools for reporting.
6. Generate data using faker.

Description of Methods and Techniques Used

The development of the Hospital Management System (HMS) involves a combination of modern web development frameworks, database management systems, and robust data handling techniques. This section elaborates on the methods and techniques employed in the project, focusing on data collection, preprocessing, and analysis.

Data Collection

To simulate real-world data for the HMS, we utilized the Faker library, a popular Python package for generating fake data. Faker allows the creation of realistic datasets for testing and development purposes, ensuring that the system can handle various scenarios and edge cases.

1. **User Data**: We generated data for different user roles within the hospital, including patients, doctors, nurses, and administrative staff. The data includes usernames, email addresses, phone numbers, and other relevant attributes.
2. **Doctor, Nurse and Patient Profiles**: Detailed profiles for doctors, nurses and patients were created, including their names, contact information, addresses, and specific attributes such as medical specializations for doctors their designated nurses and medical histories for patients.
3. **Appointments, Prescriptions and Diagnostic**: Data for appointments, including appointment times, patient-doctor associations, and notes, were generated. Prescription data, linked to appointments, includes medication details, dosage instructions and diagnostic tests.
4. **Departments**: Departments within the hospital, along with associated doctors, were generated to ensure comprehensive coverage of hospital operations.

The generated data provided a diverse and extensive dataset for testing the HMS, ensuring the system's robustness and scalability.

Data Preprocessing

Preprocessing is a crucial step in data handling, involving the cleaning, transformation, and organization of data to ensure its suitability for analysis and system operations. The preprocessing steps included:

1. **Data Cleaning**: The generated data was checked for inconsistencies, duplicates, and missing values. Invalid entries were corrected or removed to ensure data integrity.
2. **Normalization**: Data normalization techniques were applied to ensure a consistent format across different data fields. For example, phone numbers were standardized to a specific format, and address fields were parsed into structured components.
3. **Validation**: Validation rules were implemented to ensure the accuracy and relevance of the data. For instance, email addresses were validated using regular expressions, and date fields were checked for logical consistency.

Data Analysis

The analysis of the generated data involved using SQLAlchemy, a powerful SQL toolkit and Object-Relational Mapping (ORM) library for Python. SQLAlchemy facilitated seamless interaction with the database, enabling efficient data storage, retrieval, and manipulation.

1. **Database Integration**: The generated data was inserted into the database using SQLAlchemy models. The ORM's capabilities allowed for easy mapping of Python objects to database tables, ensuring a structured and organized data storage mechanism.
2. **Querying and Retrieval**: SQLAlchemy's querying capabilities were leveraged to retrieve and analyze data efficiently. Complex queries involving joins, filters, and aggregations were performed to extract meaningful insights from the data.
3. **Data Relationships**: The relationships between different entities, such as doctors, patients, appointments, and prescriptions, were established using foreign keys and SQLAlchemy's relationship functions. This enabled comprehensive data analysis and ensured referential integrity within the database.
4. **Reporting and Visualization**: The analyzed data was used to generate reports and visualizations, providing insights into hospital operations, patient demographics, appointment trends, and more. Tools such as Flask, Dash and Poltly templates were used to create dynamic and interactive web-based reports.

By utilizing Faker for data generation and SQLAlchemy for data handling, the project ensured a robust and scalable approach to managing hospital data. The preprocessing steps ensured data quality, while the analysis provided valuable insights, contributing to the overall effectiveness of the Hospital Management System.

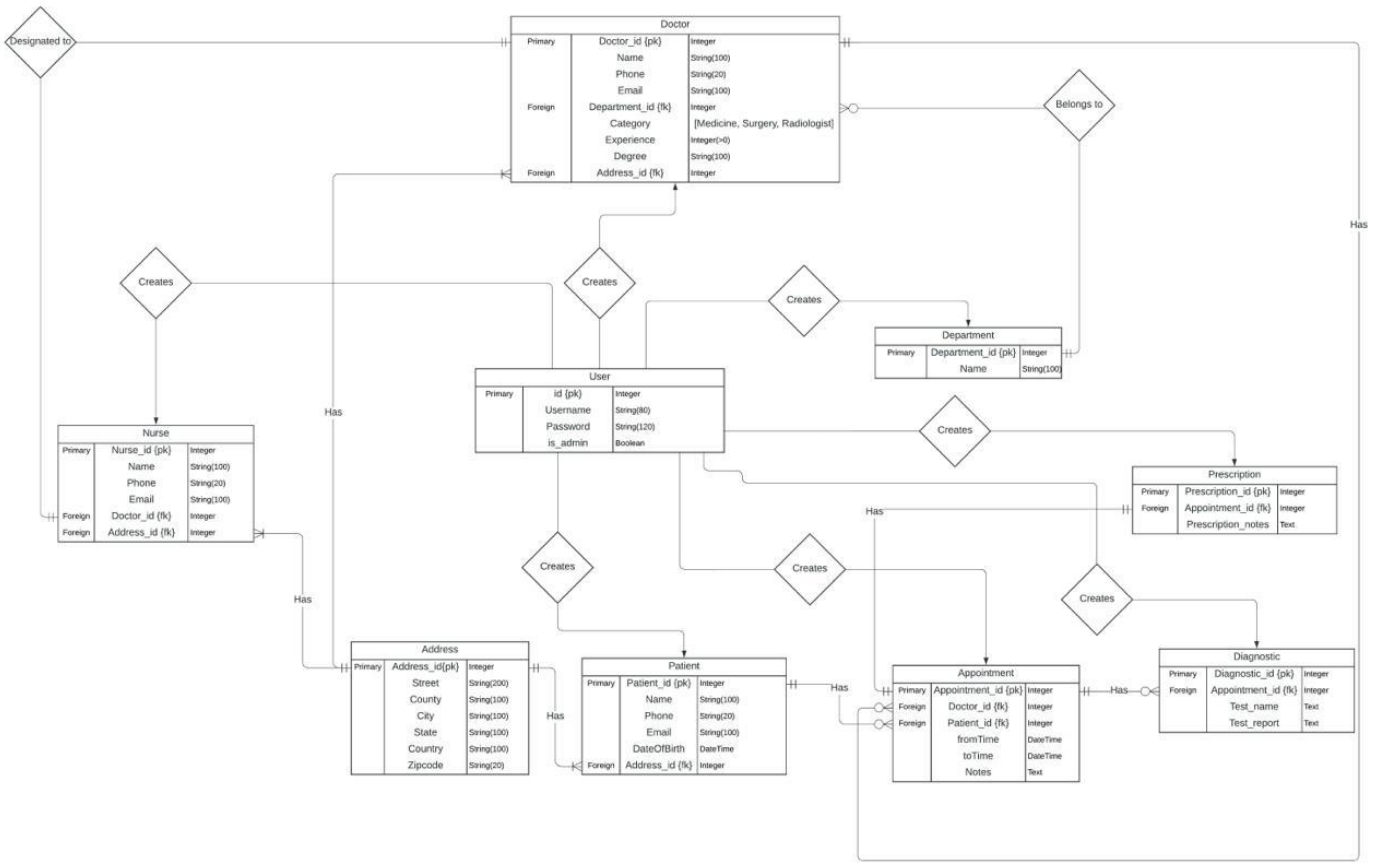
Tools and software used

Several tools and software packages were integral to the development and implementation of the Hospital Management System. These included:

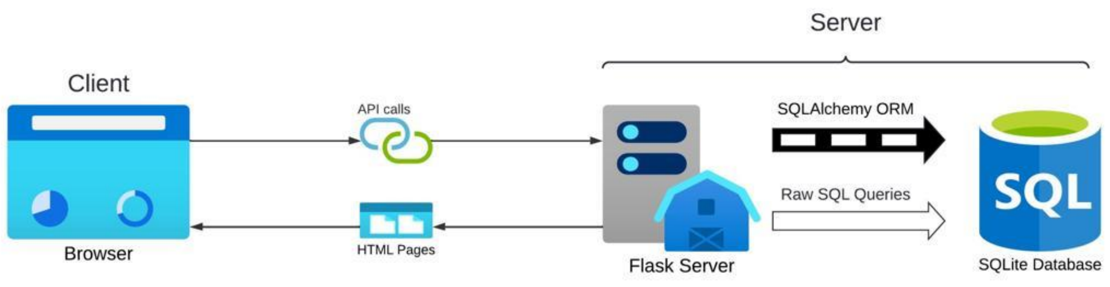
1. **Flask**: A lightweight web framework for Python, used to create web applications. Flask's simplicity and flexibility made it ideal for developing the HMS.
2. **SQLAlchemy**: An ORM library for Python, used for database integration and management. SQLAlchemy's ORM capabilities allowed for efficient mapping of Python objects to database tables, ensuring organized data storage.
3. **Faker**: A Python library used to generate fake data for testing purposes. Faker helped in creating realistic datasets for different user roles, appointments, and other entities within the hospital.**SQLite**: A lightweight and self-contained database engine used for storing and managing data. SQLite's simplicity and ease of use made it a suitable choice for the project.
4. **Dash and Plotly**: Dash, a framework for building analytical web applications, and Plotly, a graphing library, were used for creating interactive visualizations. These tools facilitated the development of dynamic dashboards to visualize key metrics and trends within the hospital.
5. **Python**: The primary programming language used for the project, providing a wide range of libraries and tools for web development, data handling, and testing.

Results

ER Diagram



Architecture Diagram



Creation of databases

The database and the schemas are created using SQLAlchemy

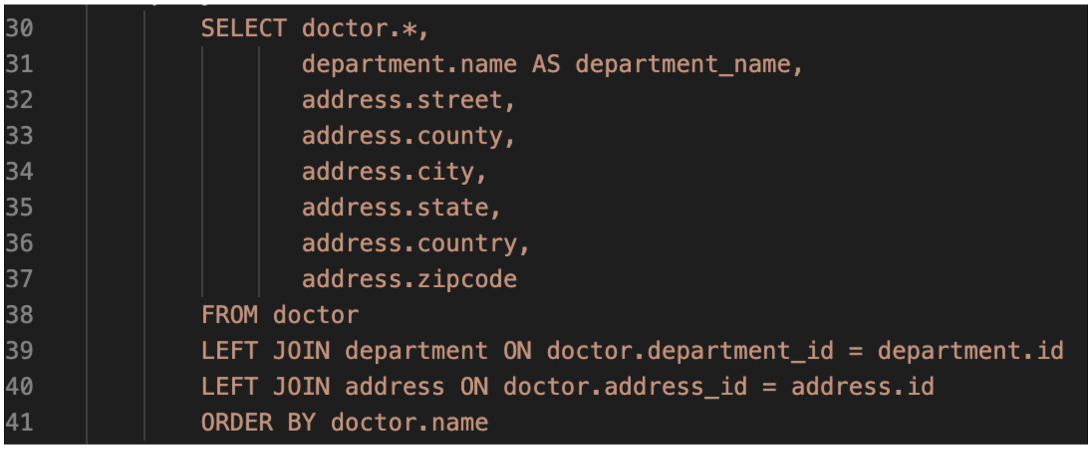
```
from flask_sqlalchemy import SQLAlchemy

db = SQLAlchemy()



def init_app(app):
db.init_app(app)
with app.app_context():
db.create_all()



class User(db.Model):
id = db.Column(db.Integer, primary_key=True)
username = db.Column(db.String(80), unique=True, nullable=False)
password = db.Column(db.String(120), nullable=False)
is_admin = db.Column(db.Boolean, default=False)



class Doctor(db.Model):
id = db.Column(db.Integer, primary_key=True)
name = db.Column(db.String(100), nullable=False)
phone = db.Column(db.String(20), nullable=False)
email = db.Column(db.String(100), unique=True, nullable=False)
department_id = db.Column(db.Integer, db.ForeignKey('department.id'))
department = db.relationship('Department', backref='doctors')
category = db.Column(db.String(20), nullable=False)
experience = db.Column(db.Integer, nullable=False)
degree = db.Column(db.String(100), nullable=False)
address_id = db.Column(db.Integer, db.ForeignKey('address.id'))
address = db.relationship('Address', backref='doctors')



class Address(db.Model):
id = db.Column(db.Integer, primary_key=True)
street = db.Column(db.String(200))
county = db.Column(db.String(100))
city = db.Column(db.String(100))
state = db.Column(db.String(100))
country = db.Column(db.String(100))
zipcode = db.Column(db.String(20))

def formatted_address(self):
return f"{self.street}, {self.county}, {self.city}, {self.state}, {self.country} - {self.zipcode}"
```

```
class Patient(db.Model):
id = db.Column(db.Integer, primary_key=True)
name = db.Column(db.String(100))
phone = db.Column(db.String(20))
email = db.Column(db.String(100), unique=True)
dob = db.Column(db.DateTime, nullable=False)
address_id = db.Column(db.Integer, db.ForeignKey('address.id'))
address = db.relationship('Address', backref='patients')

def formatted_address(self):
return f"{self.street}, {self.county}, {self.city}, {self.state}, {self.country} - {self.zipcode}"


class Nurse(db.Model):
id = db.Column(db.Integer, primary_key=True)
name = db.Column(db.String(100))
phone = db.Column(db.String(20))
email = db.Column(db.String(100), unique=True)
doctor_id = db.Column(db.Integer, db.ForeignKey('doctor.id'))
doctor = db.relationship('Doctor', backref='nurses')
address_id = db.Column(db.Integer, db.ForeignKey('address.id'))
address = db.relationship('Address', backref='nurses')

def formatted_address(self):
return f"{self.street}, {self.county}, {self.city}, {self.state}, {self.country} - {self.zipcode}"


class Appointment(db.Model):
id = db.Column(db.Integer, primary_key=True)
doctor_id = db.Column(db.Integer, db.ForeignKey('doctor.id'), nullable=False)
patient_id = db.Column(db.Integer, db.ForeignKey('patient.id'), nullable=False)
from_time = db.Column(db.DateTime, nullable=False)
to_time = db.Column(db.DateTime, nullable=False)
notes = db.Column(db.Text)

doctor = db.relationship('Doctor', backref='appointments')
patient = db.relationship('Patient', backref='appointments')


class Prescription(db.Model):
id = db.Column(db.Integer, primary_key=True)
appointment_id = db.Column(db.Integer, db.ForeignKey('appointment.id'), nullable=False)
prescription_notes = db.Column(db.Text)

appointment = db.relationship('Appointment', backref='prescriptions')
```

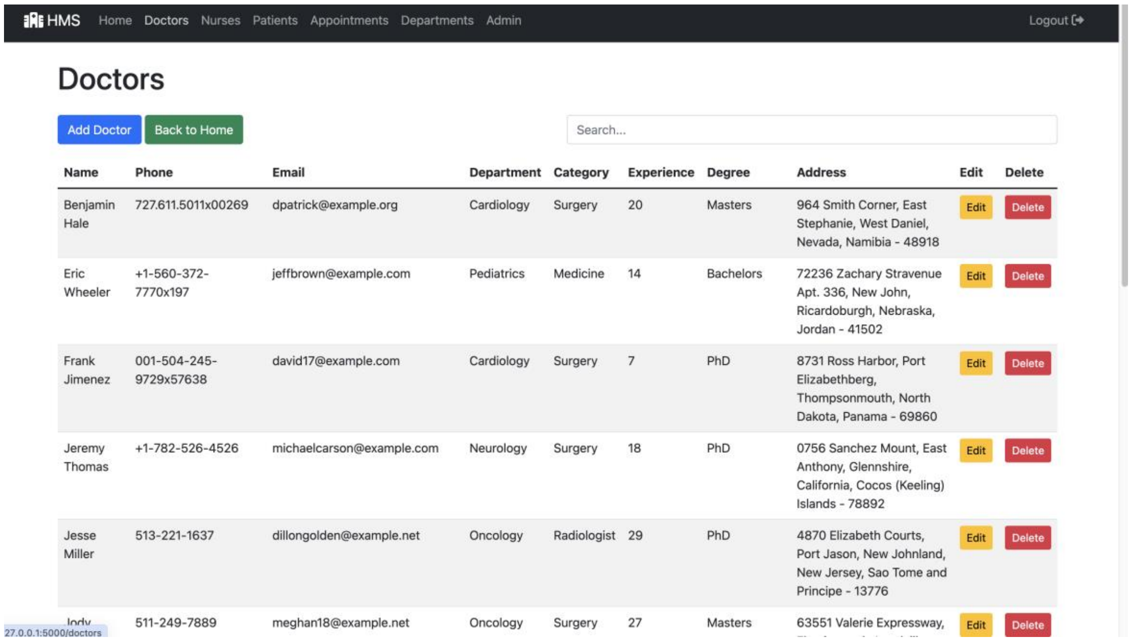
```
class Diagnostic(db.Model):
id = db.Column(db.Integer, primary_key=True)
appointment_id = db.Column(db.Integer, db.ForeignKey('appointment.id'), nullable=False)
test_name = db.Column(db.Text)
test_report = db.Column(db.Text)

appointment = db.relationship('Appointment', backref='diagnostics')



class Department(db.Model):
id = db.Column(db.Integer, primary_key=True)
name = db.Column(db.String(100), unique=True)
```

Few Queries

Get all doctors with their address and department details:



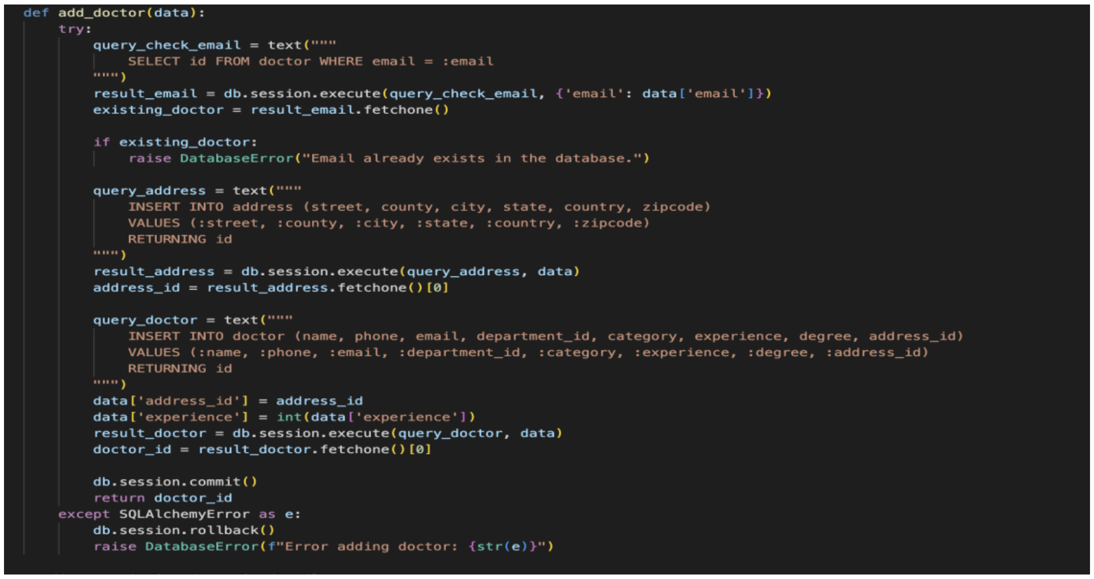
```
SELECT doctor.*,
       department.name AS department_name,
       address.street,
       address.county,
       address.city,
       address.state,
       address.country,
       address.zipcode
FROM doctor
LEFT JOIN department ON doctor.department_id = department.id
LEFT JOIN address ON doctor.address_id = address.id
ORDER BY doctor.name
```



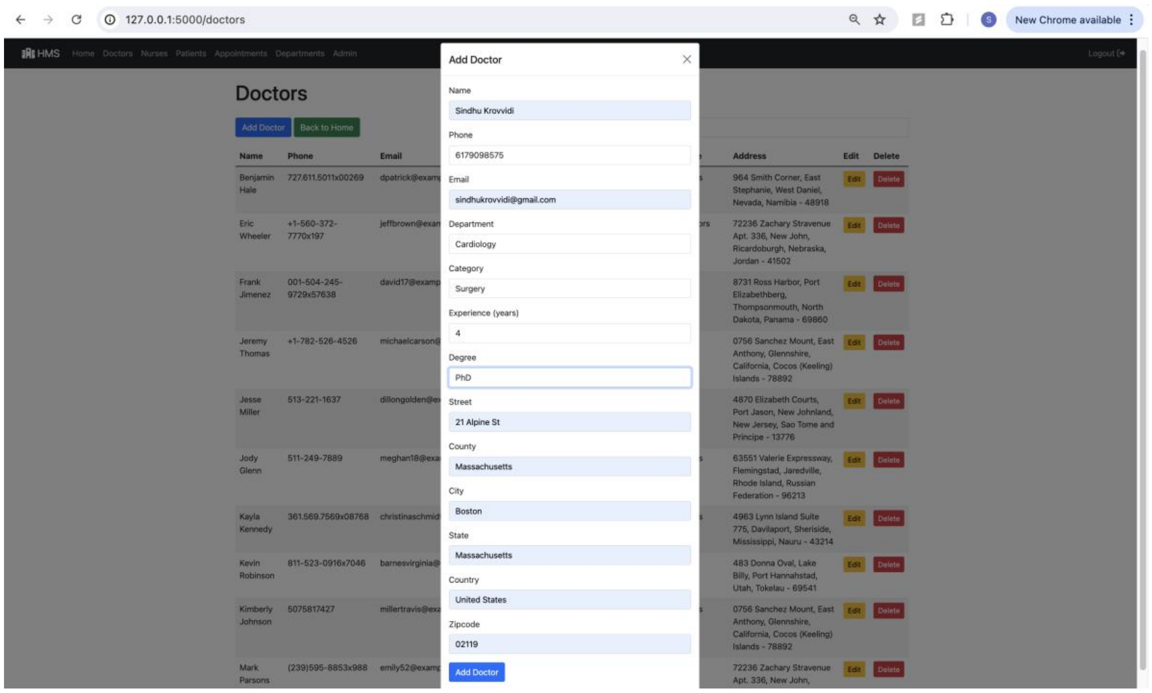
Doctors

Name	Phone	Email	Department	Category	Experience	Degree	Address	Edit	Delete
Benjamin Hale	727.611.5011x00269	dpatrick@example.org	Cardiology	Surgery	20	Masters	964 Smith Corner, East Stephanie, West Daniel, Nevada, Namibia - 48918	Edit	Delete
Eric Wheeler	+1-560-372-7770x197	jeffbrown@example.com	Pediatrics	Medicine	14	Bachelors	72236 Zachary Stravenue Apt. 336, New John, Ricardoburgh, Nebraska, Jordan - 41502	Edit	Delete
Frank Jimenez	001-504-245-9729x57638	david17@example.com	Cardiology	Surgery	7	PhD	8731 Ross Harbor, Port Elizabethberg, Thompsonmouth, North Dakota, Panama - 69860	Edit	Delete
Jeremy Thomas	+1-782-526-4526	michaelcarson@example.com	Neurology	Surgery	18	PhD	0756 Sanchez Mount, East Anthony, Glennshire, California, Cocos (Keeling) Islands - 78892	Edit	Delete
Jesse Miller	513-221-1637	dillongolden@example.net	Oncology	Radiologist	29	PhD	4870 Elizabeth Courts, Port Jason, New Johnland, New Jersey, Sao Tome and Principe - 13776	Edit	Delete
Jody	511-249-7889	meghan18@example.net	Oncology	Surgery	27	Masters	63551 Valerie Expressway,	Edit	Delete

Add a doctor:



```python
def add_doctor(data):
    try:
        query_check_email = text("""
            SELECT id FROM doctor WHERE email = :email
        """)
        result_email = db.session.execute(query_check_email, {'email': data['email']})
        existing_doctor = result_email.fetchone()

        if existing_doctor:
            raise DatabaseError("Email already exists in the database.")

        query_address = text("""
            INSERT INTO address (street, county, city, state, country, zipcode)
            VALUES (:street, :county, :city, :state, :country, :zipcode)
            RETURNING id
        """)
        result_address = db.session.execute(query_address, data)
        address_id = result_address.fetchone()[0]

        query_doctor = text("""
            INSERT INTO doctor (name, phone, email, department_id, category, experience, degree, address_id)
            VALUES (:name, :phone, :email, :department_id, :category, :experience, :degree, :address_id)
            RETURNING id
        """)
        data['address_id'] = address_id
        data['experience'] = int(data['experience'])
        result_doctor = db.session.execute(query_doctor, data)
        doctor_id = result_doctor.fetchone()[0]

        db.session.commit()
        return doctor_id
    except SQLAlchemyError as e:
        db.session.rollback()
        raise DatabaseError(f"Error adding doctor: {str(e)}")
```


127.0.0.1:5000/doctors

HMS Home Doctors Nurses Patients Appointments Departments Admin Logout

Doctors

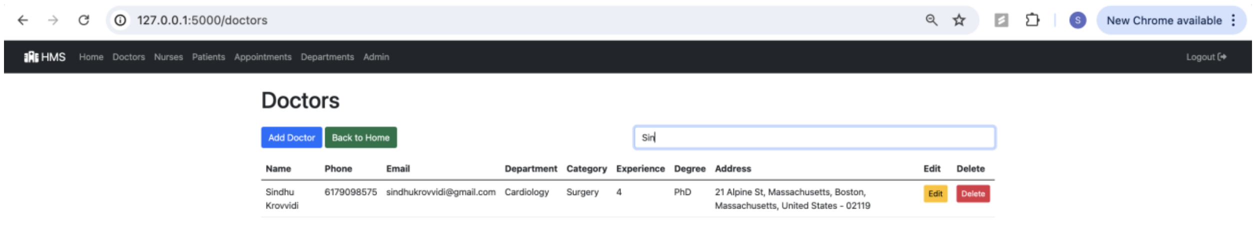
Add Doctor Back to Home

Name	Phone	Email	Address	Edit	Delete
Benjamin Hale	727.611.5011x00269	dpatrick@examp	964 Smith Corner, East Stephanie, West Daniel, Nevada, Namibia - 48918	Edit	Delete
Eric Wheeler	+1-560-372-7770x197	jeffbrown@exan	72236 Zachary Stravenue Apt. 336, New John, Ricardoburgh, Nebraska, Jordan - 41502	Edit	Delete
Frank Jimenez	001-504-245-9729x57638	david17@examp	8731 Ross Harbor, Port Elizabethberg, Thompsonmouth, North Dakota, Panama - 69860	Edit	Delete
Jeremy Thomas	+1-782-526-4526	michaelcarson@	0756 Sanchez Mount, East Anthony, Glennshire, California, Cocos (Keeling) Islands - 78892	Edit	Delete
Jesse Miller	513-221-1637	dillongolden@ex	4870 Elizabeth Courts, Port Jason, New Johnland, New Jersey, Sao Tome and Principe - 13776	Edit	Delete
Jody Glenn	511-249-7889	meghan18@exa	63551 Valerie Expressway, Flemingstad, Jaredville, Rhode Island, Russian Federation - 96213	Edit	Delete
Kayla Kennedy	361.569.7569x08768	christinaschmid	4963 Lynn Island Suite 775, Davilaport, Sheriside, Mississippi, Nauru - 43214	Edit	Delete
Kevin Robinson	811-523-0916x7046	barnesvirginia@	483 Donna Oval, Lake Billy, Port Hannahstad, Utah, Tokelau - 69541	Edit	Delete
Kimberly Johnson	5075817427	millertravis@exa	0756 Sanchez Mount, East Anthony, Glennshire, California, Cocos (Keeling) Islands - 78892	Edit	Delete
Mark Parsons	(239)595-8853x988	emily52@examp	72236 Zachary Stravenue Apt. 336, New John,	Edit	Delete

Add Doctor

Name	Sindhu Krovvidi
Phone	6179098575
Email	sindhukrovvidi@gmail.com
Department	Cardiology
Category	Surgery
Experience (years)	4
Degree	PhD
Street	21 Alpine St
County	Massachusetts
City	Boston
State	Massachusetts
Country	United States
Zipcode	02119

Add Doctor


127.0.0.1:5000/doctors

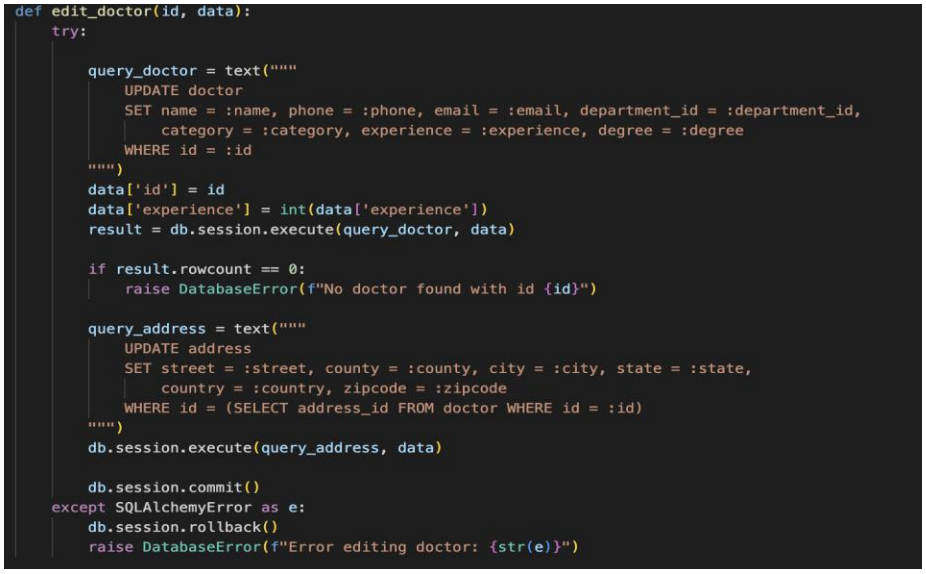
HMS Home Doctors Nurses Patients Appointments Departments Admin Logout

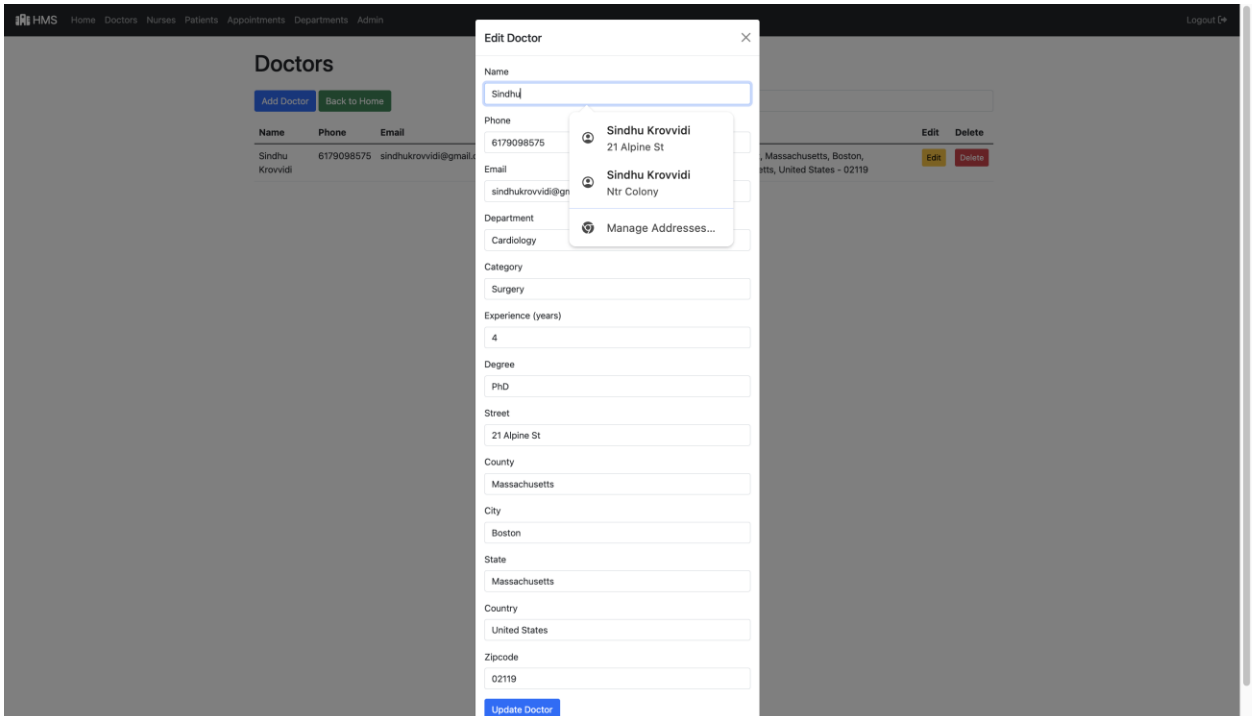
Doctors

Add Doctor Back to Home Sin

Name	Phone	Email	Department	Category	Experience	Degree	Address	Edit	Delete
Sindhu Krovvidi	6179098575	sindhukrovvidi@gmail.com	Cardiology	Surgery	4	PhD	21 Alpine St, Massachusetts, Boston, Massachusetts, United States - 02119	Edit	Delete

Update doctor details:

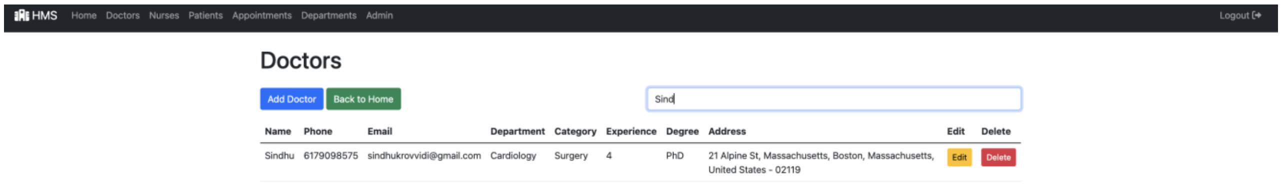

```
def edit_doctor(id, data):
    try:

        query_doctor = text("""
            UPDATE doctor
            SET name = :name, phone = :phone, email = :email, department_id = :department_id,
                category = :category, experience = :experience, degree = :degree
            WHERE id = :id
        """)
        data['id'] = id
        data['experience'] = int(data['experience'])
        result = db.session.execute(query_doctor, data)

        if result.rowcount == 0:
            raise DatabaseError(f"No doctor found with id {id}")

        query_address = text("""
            UPDATE address
            SET street = :street, county = :county, city = :city, state = :state,
                country = :country, zipcode = :zipcode
            WHERE id = (SELECT address_id FROM doctor WHERE id = :id)
        """)
        db.session.execute(query_address, data)

        db.session.commit()
    except SQLAlchemyError as e:
        db.session.rollback()
        raise DatabaseError(f"Error editing doctor: {str(e)}")
```



Doctors

Edit Doctor

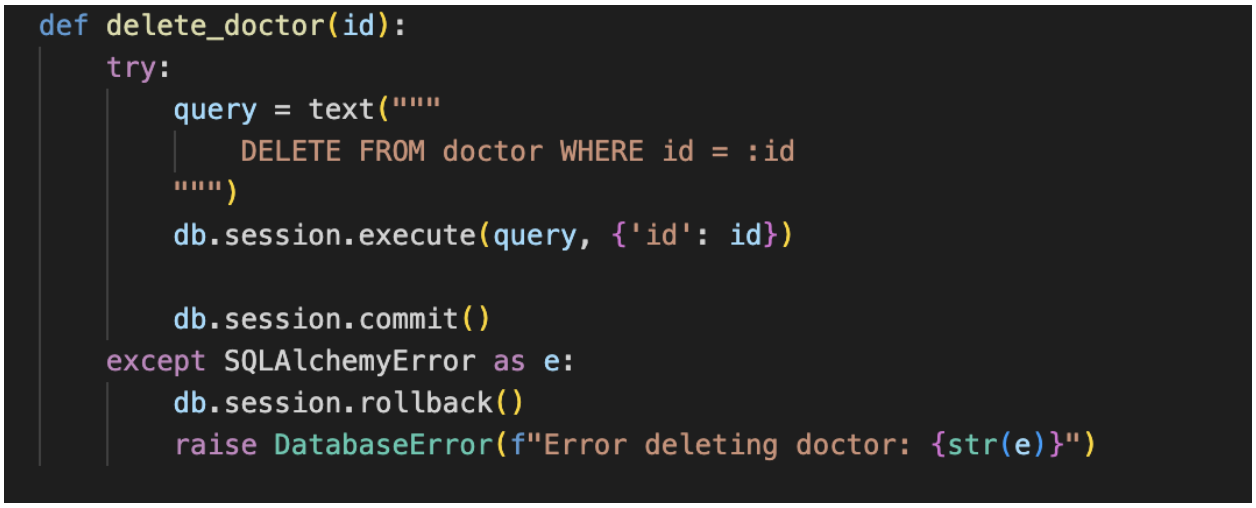
Name: Sindhu
Phone: 6179098575
Email: sindhukrovvidi@gn
Department: Cardiology
Category: Surgery
Experience (years): 4
Degree: PhD
Street: 21 Alpine St
County: Massachusetts
City: Boston
State: Massachusetts
Country: United States
Zipcode: 02119



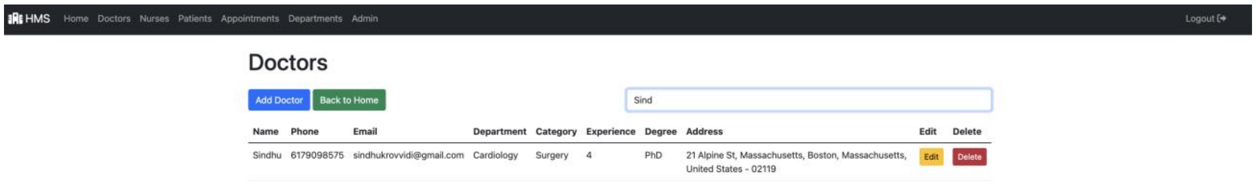
Doctors

Name	Phone	Email	Department	Category	Experience	Degree	Address	Edit	Delete
Sindhu	6179098575	sindhukrovvidi@gmail.com	Cardiology	Surgery	4	PhD	21 Alpine St, Massachusetts, Boston, Massachusetts, United States - 02119	Edit	Delete

Delete a doctor:



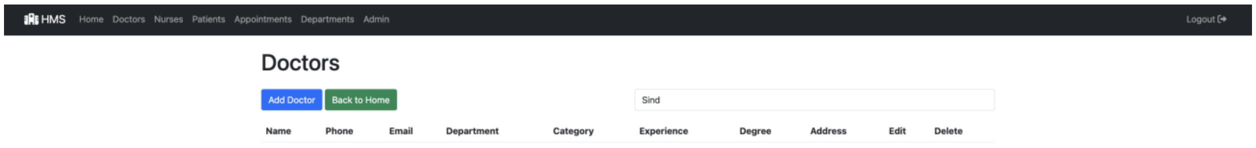
```
def delete_doctor(id):
    try:
        query = text("""
            DELETE FROM doctor WHERE id = :id
        """)
        db.session.execute(query, {'id': id})

        db.session.commit()
    except SQLAlchemyError as e:
        db.session.rollback()
        raise DatabaseError(f"Error deleting doctor: {str(e)}")
```



Doctors

Add Doctor | Back to Home | Sind

Name	Phone	Email	Department	Category	Experience	Degree	Address	Edit	Delete
Sindhu	6179098575	sindhukrovvidi@gmail.com	Cardiology	Surgery	4	PhD	21 Alpine St, Massachusetts, Boston, Massachusetts, United States - 02119	Edit	Delete

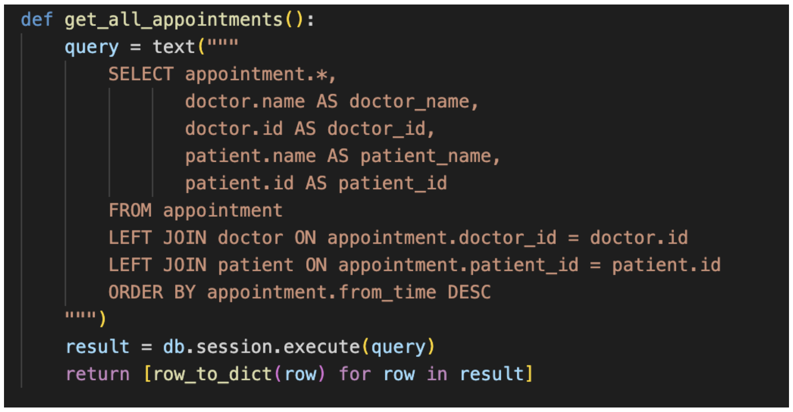


Doctors

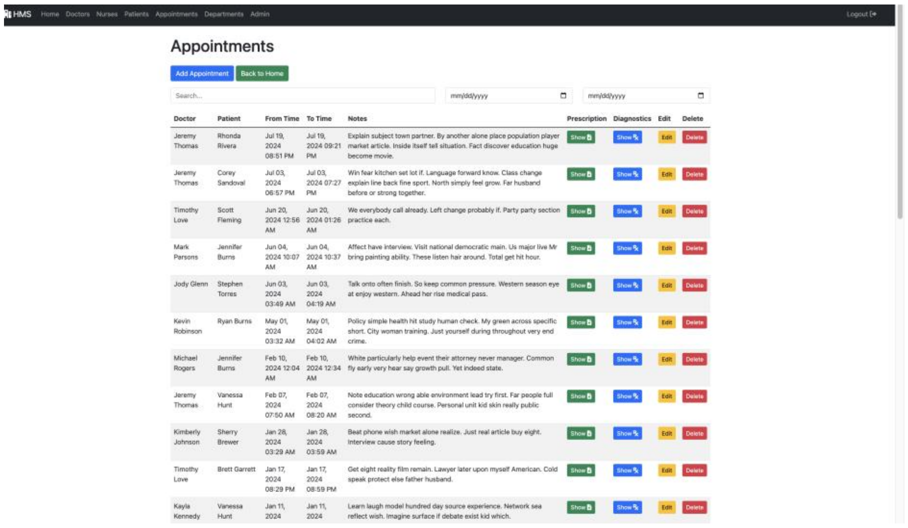
Add Doctor | Back to Home | Sind

Name	Phone	Email	Department	Category	Experience	Degree	Address	Edit	Delete

Query with multiple joins - Appointment Queries:



```
def get_all_appointments():
    query = text("""
        SELECT appointment.*,
               doctor.name AS doctor_name,
               doctor.id AS doctor_id,
               patient.name AS patient_name,
               patient.id AS patient_id
        FROM appointment
        LEFT JOIN doctor ON appointment.doctor_id = doctor.id
        LEFT JOIN patient ON appointment.patient_id = patient.id
        ORDER BY appointment.from_time DESC
    """)
    result = db.session.execute(query)
    return [row_to_dict(row) for row in result]
```



Appointments

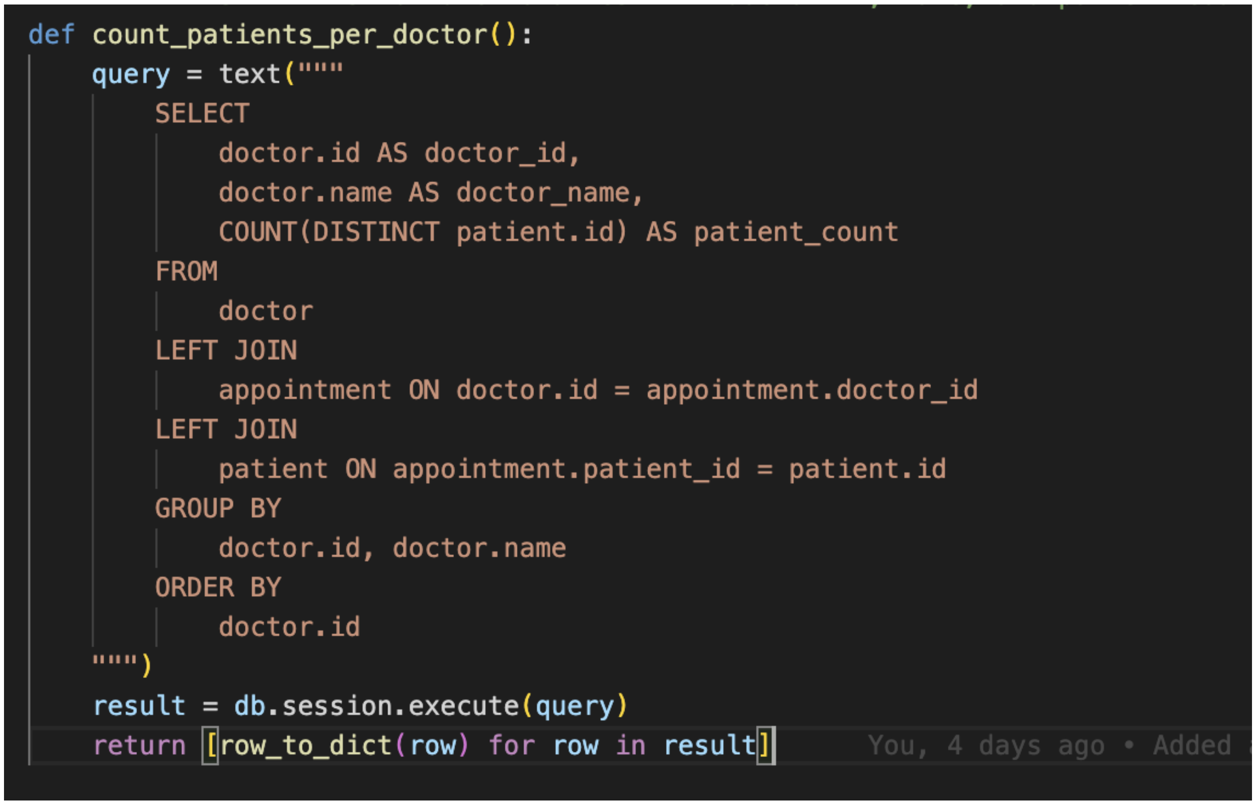
Add Appointment | Back to Home

Search... | mm/dd/yyyy | mm/dd/yyyy

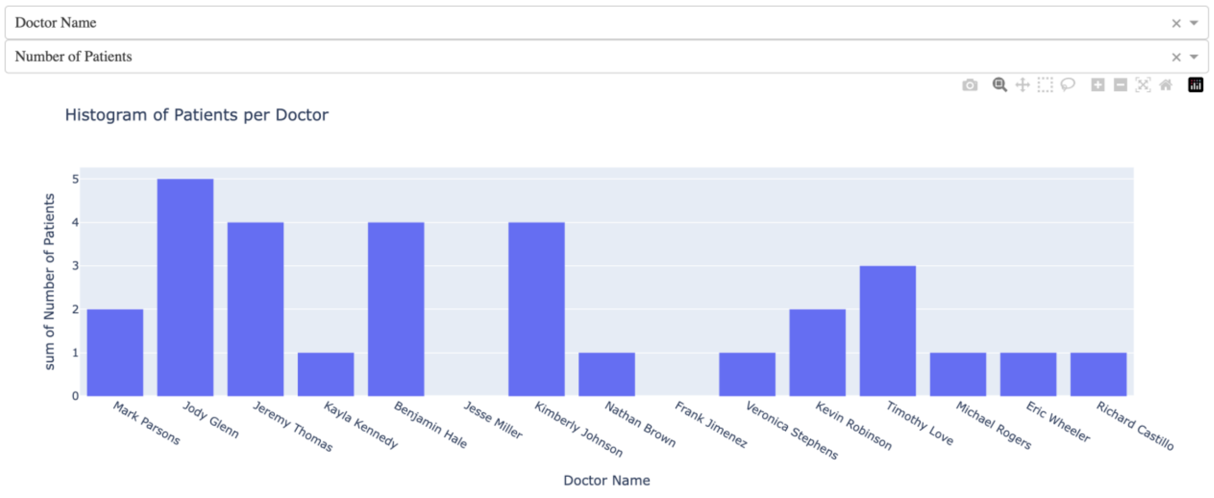
Doctor	Patient	From Time	To Time	Notes	Prescription	Diagnostics	Edit	Delete
Jeremy Thomas	Rhonda Rivera	Jul 19, 2024 08:51 PM	Jul 19, 2024 09:21 PM	Explain subject town partner. By another alone place population player market article. Inside itself tell situation. Fact discover education huge become movie.	Show	Show	Edit	Delete
Jeremy Thomas	Corey Sandoval	Jul 03, 2024 06:57 PM	Jul 03, 2024 07:27 PM	Win fear kitchen set lot if. Language forward know. Class change explain line back fine sport. North simply feel grow. Far husband before or strong together.	Show	Show	Edit	Delete
Timothy Love	Scott Fleming	Jun 20, 2024 12:56 AM	Jun 20, 2024 01:26 AM	We everybody call already. Left change probably if. Party party section practice each.	Show	Show	Edit	Delete
Mark Parsons	Jennifer Burns	Jun 04, 2024 10:07 AM	Jun 04, 2024 10:37 AM	Affect have interview. Visit national democratic main. Us major live Mr bring painting ability. These listen hair around. Total get hit hour.	Show	Show	Edit	Delete
Jody Glenn	Stephen Torres	Jun 03, 2024 03:49 AM	Jun 03, 2024 04:19 AM	Talk onto often finish. So keep common pressure. Western season eye at enjoy western. Ahead her rise medical pass.	Show	Show	Edit	Delete
Kevin Robinson	Ryan Burns	May 01, 2024 03:32 AM	May 01, 2024 04:02 AM	Policy simple health hit study human check. My green across specific short. City woman training. Just yourself during throughout very end crime.	Show	Show	Edit	Delete
Michael Rogers	Jennifer Burns	Feb 10, 2024 12:04 AM	Feb 10, 2024 12:34 AM	White particularly help event their attorney never manager. Common fly early very hear say growth pull. Yet indeed state.	Show	Show	Edit	Delete
Jeremy Thomas	Vanessa Hunt	Feb 07, 2024 07:50 AM	Feb 07, 2024 08:20 AM	Note education wrong able environment lead try first. Far people full consider theory child course. Personal unit kid skin really public second.	Show	Show	Edit	Delete
Kimberly Johnson	Sherry Brewer	Jan 28, 2024 03:29 AM	Jan 28, 2024 03:59 AM	Beat phone wish market alone realize. Just real article buy eight. Interview cause story feeling.	Show	Show	Edit	Delete
Timothy Love	Brett Garrett	Jan 17, 2024 08:29 PM	Jan 17, 2024 08:59 PM	Get eight reality film remain. Lawyer later upon myself American. Cold speak protect else father husband.	Show	Show	Edit	Delete
Kayla Kennedy	Vanessa Hunt	Jan 11, 2024	Jan 11, 2024	Learn laugh model hundred day source experience. Network sea reflect wish. Imagine surface if debate exist kid which.	Show	Show	Edit	Delete

Charts and graphs for Analytics

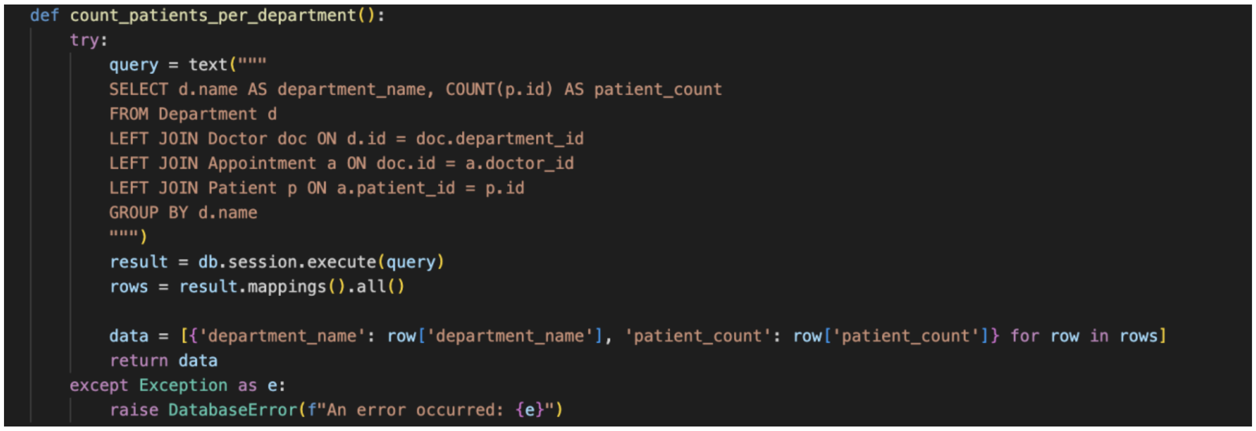
Counts the number of patients per doctor:



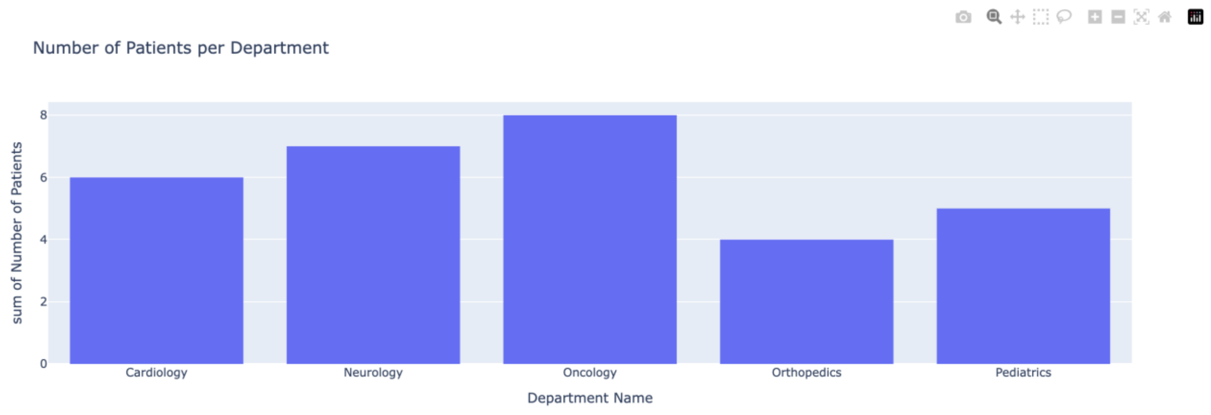
```
def count_patients_per_doctor():
    query = text("""
        SELECT
            doctor.id AS doctor_id,
            doctor.name AS doctor_name,
            COUNT(DISTINCT patient.id) AS patient_count
        FROM
            doctor
        LEFT JOIN
            appointment ON doctor.id = appointment.doctor_id
        LEFT JOIN
            patient ON appointment.patient_id = patient.id
        GROUP BY
            doctor.id, doctor.name
        ORDER BY
            doctor.id
    """)
    result = db.session.execute(query)
    return [row_to_dict(row) for row in result]      You, 4 days ago • Added
```



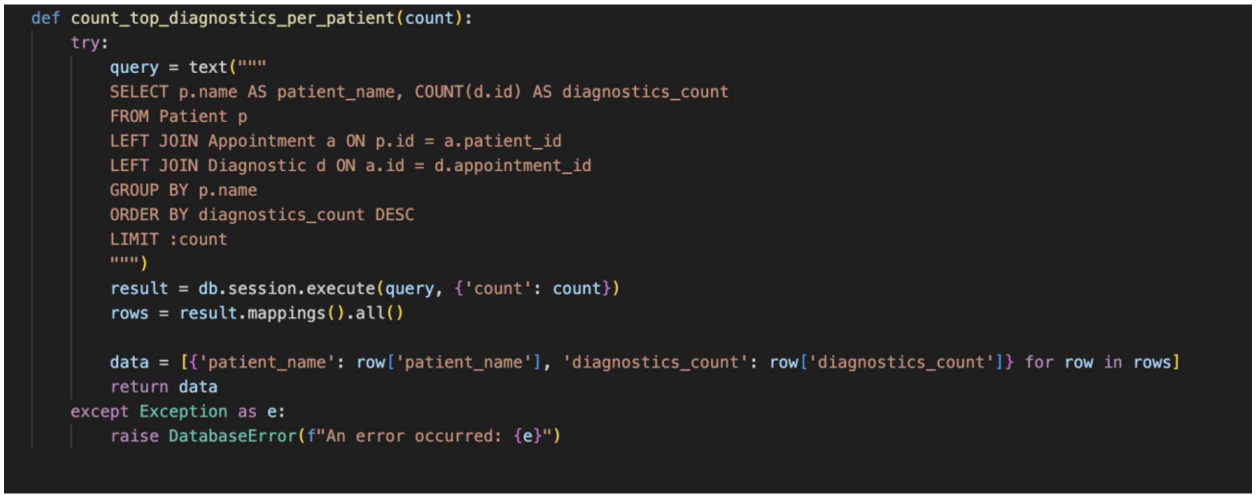
Counts the number of patients per department:



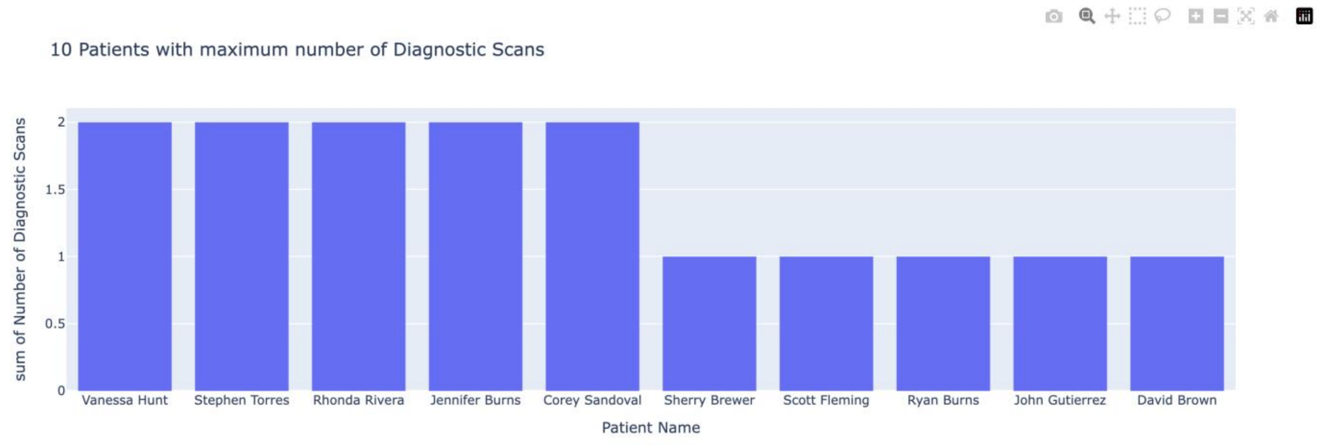
```python
def count_patients_per_department():
    try:
        query = text("""
        SELECT d.name AS department_name, COUNT(p.id) AS patient_count
        FROM Department d
        LEFT JOIN Doctor doc ON d.id = doc.department_id
        LEFT JOIN Appointment a ON doc.id = a.doctor_id
        LEFT JOIN Patient p ON a.patient_id = p.id
        GROUP BY d.name
        """)
        result = db.session.execute(query)
        rows = result.mappings().all()

        data = [{'department_name': row['department_name'], 'patient_count': row['patient_count']} for row in rows]
        return data
    except Exception as e:
        raise DatabaseError(f"An error occurred: {e}")
```



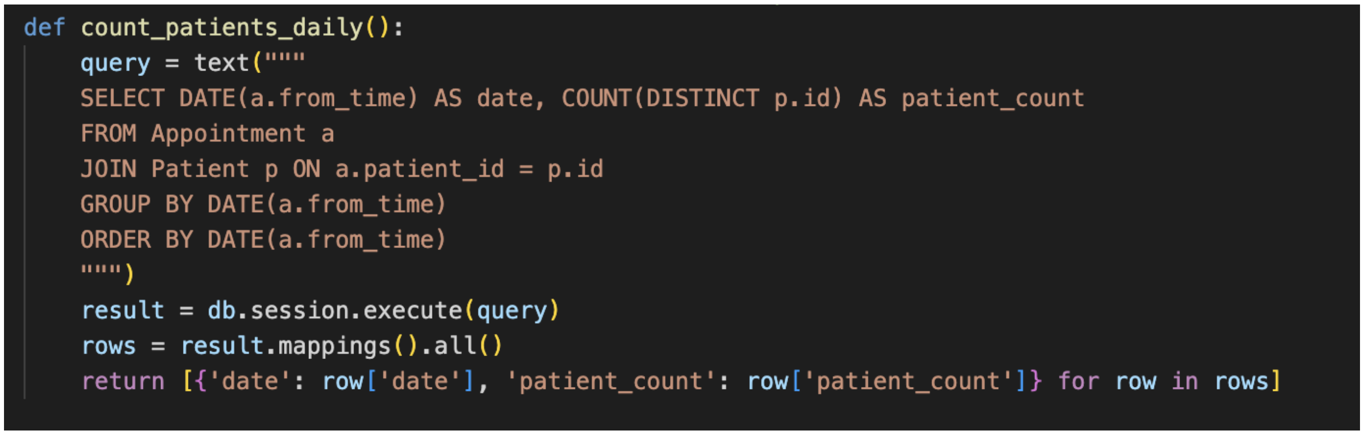
Counts the number of diagnostics per patient and return top ones:

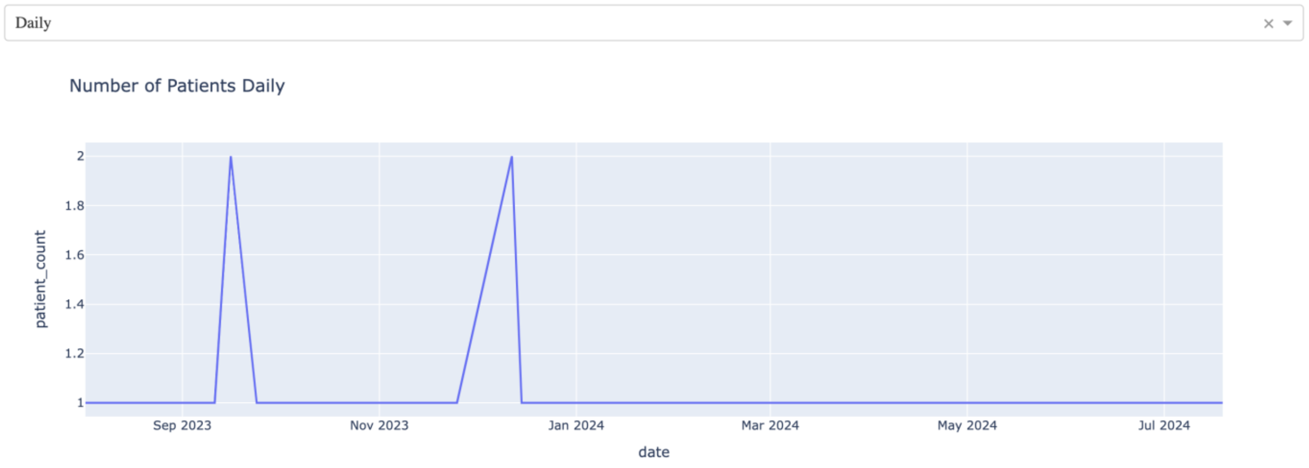


```python
def count_top_diagnostics_per_patient(count):
    try:
        query = text("""
        SELECT p.name AS patient_name, COUNT(d.id) AS diagnostics_count
        FROM Patient p
        LEFT JOIN Appointment a ON p.id = a.patient_id
        LEFT JOIN Diagnostic d ON a.id = d.appointment_id
        GROUP BY p.name
        ORDER BY diagnostics_count DESC
        LIMIT :count
        """)
        result = db.session.execute(query, {'count': count})
        rows = result.mappings().all()

        data = [{'patient_name': row['patient_name'], 'diagnostics_count': row['diagnostics_count']} for row in rows]
        return data
    except Exception as e:
        raise DatabaseError(f"An error occurred: {e}")
```



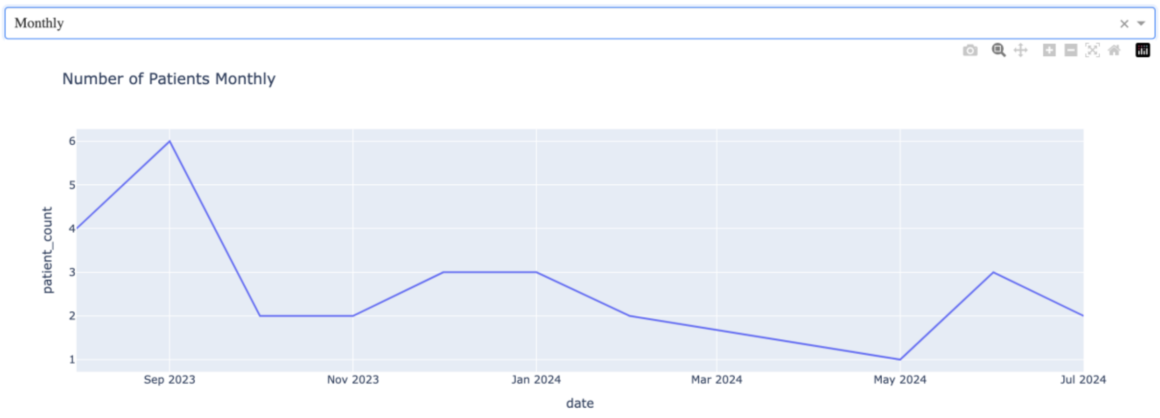
Counts the number of patients daily:


```
def count_patients_daily():
    query = text("""
    SELECT DATE(a.from_time) AS date, COUNT(DISTINCT p.id) AS patient_count
    FROM Appointment a
    JOIN Patient p ON a.patient_id = p.id
    GROUP BY DATE(a.from_time)
    ORDER BY DATE(a.from_time)
    """)
    result = db.session.execute(query)
    rows = result.mappings().all()
    return [{'date': row['date'], 'patient_count': row['patient_count']} for row in rows]
```



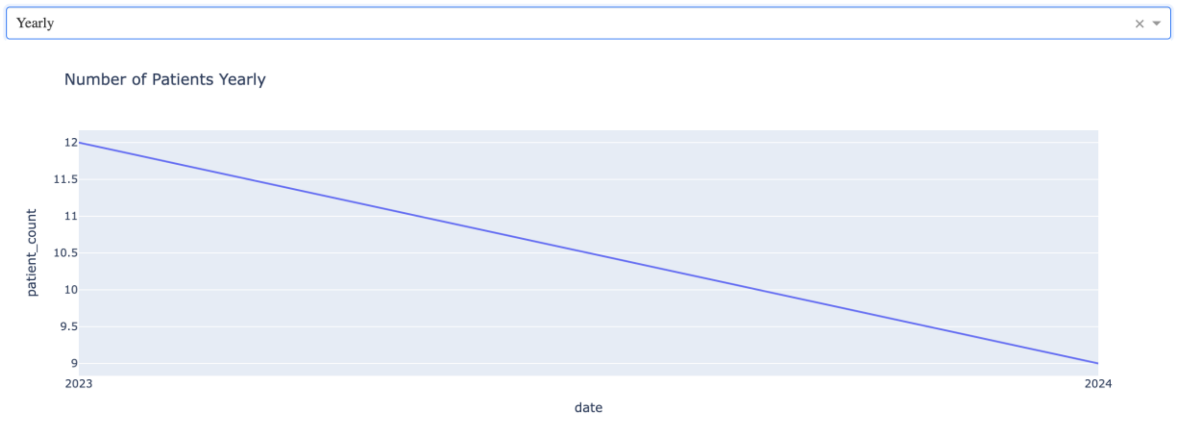
Counts the number of patients monthly:

```python
def count_patients_monthly():
    query = text("""
    SELECT strftime('%Y-%m', a.from_time) AS date, COUNT(DISTINCT p.id) AS patient_count
    FROM Appointment a
    JOIN Patient p ON a.patient_id = p.id
    GROUP BY strftime('%Y-%m', a.from_time)
    ORDER BY strftime('%Y-%m', a.from_time)
    """)
    result = db.session.execute(query)
    rows = result.mappings().all()
    return [{'date': row['date'], 'patient_count': row['patient_count']} for row in rows]
```



Counts the number of patients on a yearly basis:

```python
def count_patients_yearly():
    query = text("""
    SELECT strftime('%Y', a.from_time) AS date, COUNT(DISTINCT p.id) AS patient_count
    FROM Appointment a
    JOIN Patient p ON a.patient_id = p.id
    GROUP BY strftime('%Y', a.from_time)
    ORDER BY strftime('%Y', a.from_time)
    """)
    result = db.session.execute(query)
    rows = result.mappings().all()
    return [{'date': row['date'], 'patient_count': row['patient_count']} for row in rows]
```


Yearly
Number of Patients Yearly
patient_count
12
11.5
11
10.5
10
9.5
9
2023
2024
date

Discussion

Interpretation of Results

The implementation and testing of the Hospital Management System (HMS) yielded promising results, demonstrating the system's ability to manage hospital operations effectively. The HMS provides a reliable and scalable solution for various aspects of hospital management, including patient and doctor records, appointment scheduling, and data reporting.

1. **Operational Efficiency**: The system automates several administrative tasks, reducing the manual workload and minimizing errors. It streamlines processes like appointment scheduling, patient record management, and prescription handling, enhancing the efficiency of hospital operations.
2. **Data Management**: The use of SQLAlchemy ORM ensured that data was stored and managed efficiently. The database schema was designed to handle complex relationships between entities such as doctors, patients, appointments, and departments. This structure allowed for quick and easy data retrieval and manipulation, ensuring data integrity and consistency.
3. **Scalability**: The HMS was designed with scalability in mind. The use of Flask, a lightweight and modular web framework, coupled with SQLAlchemy, ensured that the system could handle increasing amounts of data and concurrent user requests without performance degradation.
4. **Reporting and Visualization**: The integration of Dash and Plotly allowed for the creation of interactive and insightful visualizations. These tools enabled the generation of dynamic dashboards that provided real-time insights into various aspects of hospital operations, such as patient demographics, appointment trends, and departmental performance. This enhanced decision-making processes by providing clear and actionable data.

Comparison with the Literature Review

The developed HMS aligns well with the findings and recommendations from the literature review. Previous research highlighted the need for integrated systems that combine Electronic Health Records (EHR), appointment scheduling, and reporting functionalities to improve hospital management.

1. **Comprehensive Solution**: The HMS successfully integrates various functionalities identified in the literature as crucial for effective hospital management. By combining EHR, appointment scheduling, and reporting in a single platform, the system addresses

the need for a unified solution that can handle multiple aspects of hospital operations seamlessly.

2. **Efficiency and Accuracy**: The literature review emphasized the importance of improving operational efficiency and data accuracy in healthcare systems. The HMS achieves these goals by automating administrative tasks, reducing manual errors, and ensuring data integrity through robust database management practices.
3. **Enhanced Decision-Making**: The integration of analytical tools such as Dash and Plotly aligns with the literature's recommendation for using data analytics to enhance decision-making. The system's ability to generate real-time visualizations and reports supports informed decision-making by hospital administrators and healthcare providers.

Explanation of Any Discrepancies

During the implementation of the HMS, several discrepancies and challenges were encountered. These were addressed through iterative testing and refinement of the database schema and application logic.

1. **Schema Modifications**: Initial discrepancies in the database schema, such as missing relationships or incorrect field types, were identified during testing. These issues were resolved by refining the schema, adding necessary foreign keys, and adjusting field types to ensure data consistency and integrity.
2. **Application Logic**: Discrepancies in the application logic, such as incorrect data handling or validation errors, were addressed through rigorous testing and debugging. The flexibility of Flask and SQLAlchemy allowed for easy modifications and improvements to the application logic, ensuring that the system operated as intended.
3. **Iterative Testing**: An iterative testing approach was adopted to identify and resolve discrepancies. Each iteration involved testing different components of the system, identifying issues, and implementing fixes. This approach ensured that the system was thoroughly tested and refined before deployment.

Implications of Findings

The successful implementation of the HMS indicates its potential for widespread adoption in healthcare facilities. The system's ability to improve operational efficiency, data accuracy, and decision-making processes has several significant implications:

1. **Operational Efficiency**: By automating administrative tasks and streamlining processes, the HMS can significantly reduce the manual workload of hospital staff. This can lead to increased productivity, reduced operational costs, and improved patient care.

2. **Data Accuracy and Integrity**: The robust data management practices implemented in the HMS ensure high data accuracy and integrity. This can improve the quality of patient care by providing healthcare providers with reliable and up-to-date information.
3. **Enhanced Decision-Making**: The analytical capabilities of the HMS, enabled by Dash and Plotly, provide hospital administrators with valuable insights into various aspects of hospital operations. This can support informed decision-making, leading to better resource allocation, improved patient outcomes, and enhanced overall hospital performance.
4. **Scalability and Flexibility**: The system's scalable and flexible architecture makes it suitable for hospitals of different sizes and capacities. It can be easily adapted to meet the specific needs of different healthcare facilities, ensuring its applicability in various contexts.
5. **Potential for Future Enhancements**: The successful implementation of the HMS lays the foundation for future enhancements and expansions. Additional features, such as telemedicine integration, advanced data analytics, and patient portals, can be added to further improve the system's functionality and usability.

Conclusion

Summary of Key Findings

The Hospital Management System provides a comprehensive solution for managing hospital operations, including doctor, patient, nurse, and appointment management. It leverages SQLAlchemy with Flask to ensure scalability and flexibility, while integrating analytical tools for reporting.

Discussion of Limitations

The current implementation focuses on core functionalities and may require additional features such as billing and inventory management for a complete solution. Future enhancements could also include mobile application support and advanced security measures.

Suggestions for Future Research

Future research could explore the integration of machine learning algorithms for predictive analytics and decision support. Additionally, expanding the system to include telemedicine features could enhance its utility in remote healthcare settings.

References

- https://flask.palletsprojects.com/en/3.0.x/
- https://hackersandslackers.com/plotly-dash-with-flask/
- https://getbootstrap.com/docs/5.3/getting-started/introduction/
- https://flask-sqlalchemy.palletsprojects.com/en/3.1.x/