Databases Project Phase 2

Team Members: Aryavrat Gupta, Winston Li
Section: 415
Emails: agupt110@jh.edu, wli111@jh.edu

Target Domain (Healthcare Management Database): To provide a comprehensive system for managing patient records, tracking medical appointments, and handling billing information, aimed at streamlining hospital operations and improving healthcare service delivery.

The **Interface** can be accessed locally using the steps described in the README:
http://localhost:8000

Note: Our Project Phase 1 submission is attached in the Appendix.

Set Up Instructions

The **Steps in the README** are crucial to accessing the project and have been reiterated below:

Start by setting the working directory to the project root.

To create synthetic data:

Set up a python virtual environment: python3 -m venv .venv
Activate the environment: source .venv/bin/activate
Install dependencies: pip install faker
Generate data: python3 DML/generate_data.py

To set up the database:

mysql.server start (On MacOS or Start the MySQL server using GUI in settings)
mysql -u root -p (Log in to MySQL)
CREATE DATABASE healthcare_db;
USE healthcare_db;
SOURCE path/to/create_tables.sql (SOURCE DDL/create_tables.sql)
SOURCE path/to/load_data.sql (SOURCE DML/load_data.sql)

To run queries:

SOURCE path/to/simple_queries.sql (SOURCE Queries/simple_queries.sql)
SOURCE path/to/complex_queries.sql (SOURCE Queries/complex_queries.sql)

To use the web server:

php -v (verify PHP installation)
cd app
Make sure MySQL is running
Locate app/db.php and change configuration ($user = "root"; $password = "";) as needed
php -S localhost:8000
http://localhost:8000

Debugging:

If the files don't have read permissions:
chmod +r path/to/file

How has the Project Trajectory Changed

1. Database Tables:

Original Plan:
The tables were defined as:

- Patients, Medical_Staff, Departments, Appointments, Medical_Records, Insurance_Providers, Billing, Procedures, Appointment_Procedures.
- Primary keys were specified, and foreign key relationships were planned to ensure data integrity.

Changes in Phase II:

- Appointment_Procedures Table: Adjusted for improved data consistency by generating synthetic data that avoids duplicates.
- Added timestamps (created_at) for auditing changes and logging data updates, improving traceability.
- Foreign key constraints were updated with ON DELETE CASCADE for more robust relationship handling.

2. Data Loading Plan:

Original Plan:

- Planned to use a combination of pre-approved hospital data and simulated datasets from APIs. The goal was to clean and map the data to fit the schema.

Changes in Phase II:

- Switched to synthetic data:

Reasons for Change:

1) Availability Issues: Pre-approved hospital data or mock APIs often had limited or outdated datasets that did not match the schema's requirements.
2) Control over Data: Synthetic data allowed us to create consistent, complete, and schema-compliant datasets, avoiding issues such as missing values or formatting mismatches.
3) Testing Robustness: Generated data ensured better coverage of edge cases, such as overlapping appointment times or extreme billing values. Developed Python scripts to generate synthetic data while avoiding duplicate primary keys or foreign key violations.

3. Output Plans:

Original Plan:

- Envisioned using a basic HTML/CSS interface for data viewing and entry, complemented by SQL views and CSV reports.

Changes in Phase II:

- Shifted to a PHP-based web interface for:
- Dynamic patient information display and editing.
- Improved interactivity for data management.
- Basic reporting features were replaced with real-time data views in the PHP interface.

4. Specialized Topics:

Original Focus:

- Aimed at developing a GUI with report generation, integrating triggers, and using stored procedures for advanced SQL functionalities.

Changes in Phase II:

- Simplified SQL Focus: Focused on basic SQL operations instead of advanced triggers or stored procedures.
- Interface Extensions: The PHP interface can be extended to include features like user permissions and access control.

Data Loading:

1. Data Sources:

- We initially intended to use pre-approved hospital data or public healthcare APIs. However, due to limitations in data availability and compatibility with our schema, we opted to use synthetic data generated programmatically.

- The synthetic data was generated using a Python script leveraging the Faker library, which allowed us to create realistic, randomized data while ensuring schema compliance.

2. Specialized Code for Data Generation:

- Data Generation Script: Developed a Python script to generate synthetic data for all database tables, ensuring consistency with foreign key relationships and schema constraints.
- The script included:
  - Primary Key Handling: Avoiding duplicate primary keys across tables.
  - Foreign Key Mapping: Ensuring foreign key values were valid and matched existing primary keys in related tables.
  - Field Formatting: Formatting phone numbers, email addresses, and dates to match schema requirements.
  - Data Cleaning: Addressed potential issues like escaping special characters and truncating oversized text fields.
- The script generated SQL INSERT statements for each table and saved them into a load_data.sql file.

3. Data Preprocessing:

- While generating synthetic data, the script included preprocessing steps to handle potential data mismatches:
  - Escaping Special Characters: Used custom functions to escape special characters in text fields to prevent SQL syntax errors.
  - Truncating Large Fields: Automatically truncated values that exceeded column limits (e.g., phone numbers and email addresses).
  - Avoiding Duplicates: Ensured no duplicate rows were created for tables like Appointment_Procedures by verifying the composite primary key.

4. Database Loading:

- The generated load_data.sql file was executed using the MySQL command-line client:
- mysql -u root -p healthcare_db < load_data.sql

Challenges Addressed:

- Formatting Issues: Handled inconsistencies like overly long text fields or invalid date formats by preprocessing data in the script.
- Dependency Resolution: Loaded data in the correct order (e.g., populating Departments and Insurance_Providers before dependent tables like Medical_Staff and Patients).
- Testing and Validation: Verified data integrity by running SQL queries after loading to ensure foreign key relationships and expected row counts.

Software/Hardware:

We did not use the dbase.cs.jhu.edu server for our database. Instead, we set up a local development environment for the project. Below are the details of the software and hardware platform used:

Hardware:

- Device: MacBook Pro
- Processor: Apple M1 or Intel Core i7
- RAM: 16 GB
- Storage: 512 GB SSD
- Operating System: macOS Ventura (13.x or later)

Software:

1. Database Management System:
   - MySQL: Version 8.0
   - Installed locally using Homebrew (brew install mysql).
2. Programming Language and Tools:
   - Python: Version 3.10
     - Used for data generation and preprocessing with the Faker library.
   - PHP: Version 8.4
     - Used to develop the web-based interface for interacting with the database.
3. Web Server:
   - PHP Development Server:
     - Command used to start the server: php -S localhost:8000
4. Database Connection Tools: (MySQL Command-Line Client)
   - Used for executing SQL scripts and running queries.
   - Command used to connect: mysql -u root -p
5. IDE/Text Editors:
   - Visual Studio Code: For editing SQL scripts, Python code, and PHP files.
   - Terminal: Used to run scripts and manage the database.

User's Guide:

A detailed guide for running the code and interacting with the project is included in the README.md file located in the root directory of the project. The README.md covers the following:

1. System Requirements:
   - Details the software dependencies (e.g., MySQL, PHP, Python) and hardware requirements.
2. Setup Instructions:
   - Step-by-step guidance to install required tools, set up the database, and configure the web interface.
3. Running the Project:

- Instructions for starting the MySQL server, generating data, and launching the PHP web interface.

4. Data Loading:
   - Steps to load the generated data into the database using the provided load_data.sql script.
5. Using the Web Interface:
   - Explanation of features available in the PHP interface, including patient management, appointment viewing, and billing queries.
6. Common Troubleshooting Steps:
   - Tips for resolving common errors or issues during setup or execution.

Areas of Specialization

Major Area of Specialization:

1. Interactive Web Interface with Data Management
   - Description: The project focused on developing an interactive web interface using PHP. This interface allowed users to manage patient records, view and edit appointment information, and handle billing operations.
   - Significance: The interface demonstrated significant accomplishment in user-friendliness and practicality, catering to real-world healthcare management needs. It also provided a foundation for future enhancements such as access control and permissions.

Minor Areas of Specialization

1. Synthetic Data Generation
   - Description: Instead of extracting real-world data, we generated synthetic data to populate the database. A Python script ensured the data conformed to the schema while handling edge cases like foreign key constraints and data type mismatches.
   - Reason: This approach was chosen to ensure data consistency, avoid privacy concerns, and simulate real-world scenarios with a controlled dataset.
2. Simple SQL Queries and Data Modeling
   - Description: The project used simple yet effective SQL queries to manage data. These included complex joins, grouping, and aggregation for reporting purposes. The schema design focused on a relational model optimized for healthcare management.
3. Potential for Security Enhancements
   - Description: Although basic SQL was implemented, the web interface is designed to support role-based access control (RBAC) in the future. This sets the groundwork for secure data handling and restricted access based on user roles.

Project Strengths and Selling Points

1. User-Friendly Web Interface
   - The PHP-based interface is intuitive and allows easy management of patient records, appointments, and billing.

- Features include data editing capabilities and integration with a robust backend database.

2. Extensible Design
   - The schema and interface are designed with scalability in mind, allowing for future enhancements such as role-based access control (RBAC), appointment calendars, and reporting dashboards.

3. Focus on Practicality and Real-World Applicability
   - The system targets realistic healthcare operations, addressing key use cases like patient registration, billing, and appointment scheduling.
   - Designed to reflect real-world challenges, such as managing large datasets and ensuring referential integrity.

4. Performance Optimization
   - Efficient SQL joins, indexing of primary/foreign keys, and query optimization ensure performance even with large datasets.

Project Limitations:

1. Limited Real-World Data Usage
   - The project exclusively uses synthetic data, which may lack the nuanced complexities of real-world datasets.
   - While the synthetic data is realistic, it doesn't capture unexpected anomalies often found in real healthcare data.

2. Basic Interface Functionality
   - The current PHP-based interface provides fundamental CRUD operations but lacks advanced features like interactive dashboards, calendar views, and role-based access control.

3. No Implementation of Triggers or Stored Procedures
   - Despite initial plans, advanced database features like triggers for logging changes or stored procedures for report generation were not implemented.

4. Minimal Security Features
   - Input sanitization and basic security were implemented, but comprehensive features like encryption, multi-factor authentication, or SQL injection prevention are not included.

5. Static Reporting
   - Reports are generated manually through SQL queries rather than dynamic, user-driven reporting or automated generation.

6. No Integration with Real-Time Systems

- The project does not support real-time updates or live tracking, such as immediate appointment notifications or billing updates.

7. Basic Data Validation
   - While the system validates data during insertion, there's limited validation on the interface side for user inputs.

Suggested Improvements and Extensions:

1. Integration of Real Data
   - Incorporate real-world healthcare data from public datasets or APIs to test the system's robustness and relevance.
   - Include functionality to handle edge cases like missing or malformed data.

2. Advanced User Interface
   - Add features such as a dynamic appointment calendar, real-time notifications, and analytics dashboards.
   - Implement role-based access control to differentiate between administrators, doctors, and patients.

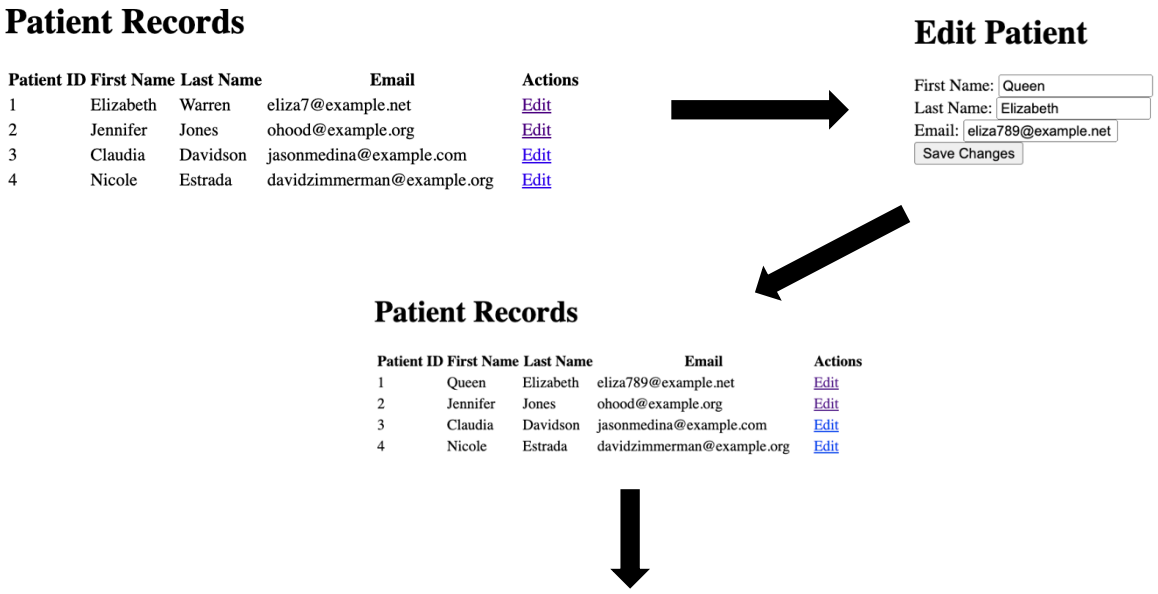
3. Triggers and Stored Procedures
   - Introduce triggers for logging changes to sensitive records (e.g., billing or appointments).
   - Create stored procedures for complex reporting tasks, such as summarizing patient histories or generating invoices.

4. Enhanced Security
   - Implement secure authentication (e.g., hashed passwords, OAuth).
   - Use parameterized queries or ORM to prevent SQL injection.
   - Encrypt sensitive data, like patient contact details or billing information.

5. Dynamic and Automated Reporting
   - Develop a feature for user-defined report generation, enabling custom queries through the interface.
   - Automate scheduled report generation and email delivery for management purposes.
6. Advanced Analytics and Machine Learning (An interesting extension)
   - Use the database for predictive analytics, such as identifying high-risk patients or optimizing staff schedules.
   - Integrate basic machine learning models for medical record analysis or billing predictions.

Output

- Query Outputs are in /Queries/simple_queries.txt and /Queries/complex.queries.txt
- Screenshots of the user-interface below show that you can edit patient information which reflects in the database (there are 100 patients).




```
mysql> select * from Patients;
+------------+------------+------------+---------------+--------+-----------------------------------------------------------------+----------------+---------------------+
| patient_id | first_name | last_name  | date_of_birth | gender | address                                                         | phone          | email               |
| insurance_provider_id | created_at          |
+------------+------------+------------+---------------+--------+-----------------------------------------------------------------+----------------+---------------------+
|          1 | Queen      | Elizabeth  | 1970-07-30    | F      | 508 Roger Turnpike
South Rhondaview, SC 37619           | +1-591-320-2631 | eliza7@example.net            |                 66 | 2024-12-20 00:43:26 |
|          2 | Jennifer   | Jones      | 2024-05-08    | M      | 3276 Matthew Highway
Jonesburgh, LA 42446               | (679)897-2620   | ohood@example.org             |                 14 | 2024-12-20 00:43:26 |
|          3 | Claudia    | Davidson   | 1994-09-24    | M      | 477 Michael Trail Suite 193
Matthewville, MH 00577       | (599)782-2761   | jasonmedina@example.com       |                 91 | 2024-12-20 00:43:26 |
|          4 | Nicole     | Estrada    | 1962-01-19    | M      | 52872 Shannon Light Apt. 996
Shariport, CT 71758          | 6462630701      | davidzimmerman@example.org    |                  2 | 2024-12-20 00:43:26 |
```

References

1. https://www.w3schools.com/sql/
2. https://dev.mysql.com/doc/
3. Course materials provided during the semester were very useful in understanding relational database design principles and creating and populating tables in SQL.
4. https://faker.readthedocs.io/

5. https://stackoverflow.com/questions/55143102/how-to-connect-to-mysql-database-in-php (and many other for understanding file permissions, installing php, setting up venv for python, etc.)

Appendix

Databases Project Phase 1

1. Team Members

- Winston Li, Aryavrat Gupta

2. Target Domain

- Healthcare Management Database:
    - Manage patient records, track medical appointments, and billing information to help hospitals manage operations

3. System Queries

1. What is the total revenue generated from patient visits in the last month?
2. Which doctors have the highest number of appointments scheduled for next week?
3. How many patients are currently waiting for specific types of procedures?
4. What is the average wait time between appointment scheduling and actual visit dates?
5. Which insurance providers have the highest claim rejection rates?
6. What are the most common diagnoses among patients aged 65 and above?
7. How many patients have missed their follow-up appointments in the last quarter?
8. What is the average length of stay for patients admitted for different procedures?
9. Which departments generate the highest revenue per patient visit?
10. What percentage of patients require prescription medication after their visits?
11. How many patients were referred to specialists in the last month?
12. What is the distribution of patient ages across different medical departments?
13. Which procedures have the highest insurance reimbursement rates?
14. What is the average number of follow-up visits required per initial diagnosis?
15. How many patients have outstanding medical bills over 90 days?

4. Relational Data Model

Tables:

Patients (patient_id, first_name, last_name, date_of_birth, gender, address, phone, email, *insurance_provider_id*)
Medical_Staff (staff_id, first_name, last_name, specialization, *department_id*, phone, email)
Departments (department_id, name, location, head_staff_id)
Appointments (appointment_id, *patient_id, staff_id*, appointment_date, status, reason)
Medical_Records (record_id, *patient_id, staff_id*, diagnosis, treatment_plan, prescription, notes, visit_date)
Insurance_Providers (provider_id, name, contact_person, phone, email)
Billing (bill_id, *patient_id, appointment_id*, amount, insurance_claim_id, status, bill_date, due_date)
Procedures (procedure_id, name, description, base_cost, *department_id*)
Appointment_Procedures (appointment_id, procedure_id, notes)

* Highlights = primary keys
* Italics = foreign keys

```
-- Patients table to store patient information
CREATE TABLE Patients (
    patient_id INT PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    date_of_birth DATE NOT NULL,
    gender CHAR(1),
    address VARCHAR(200),
    phone VARCHAR(15),
    email VARCHAR(100),
    insurance_provider_id INT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (insurance_provider_id) REFERENCES Insurance_Providers(provider_id)
);

-- Medical Staff table for doctors and other healthcare providers
CREATE TABLE Medical_Staff (
    staff_id INT PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    specialization VARCHAR(100) NOT NULL,
    department_id INT,
    phone VARCHAR(15),
    email VARCHAR(100),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (department_id) REFERENCES Departments(department_id)
);

-- Departments within the hospital
CREATE TABLE Departments (
    department_id INT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    location VARCHAR(100),
    head_staff_id INT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Appointments tracking
CREATE TABLE Appointments (
    appointment_id INT PRIMARY KEY,
    patient_id INT NOT NULL,
    staff_id INT NOT NULL,
    appointment_date DATETIME NOT NULL,
    status VARCHAR(20) NOT NULL, -- Scheduled, Completed, Canceled, No-Show
    reason VARCHAR(200),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (patient_id) REFERENCES Patients(patient_id),
    FOREIGN KEY (staff_id) REFERENCES Medical_Staff(staff_id)
);
```

```
-- Medical Records
CREATE TABLE Medical_Records (
    record_id INT PRIMARY KEY,
    patient_id INT NOT NULL,
    staff_id INT NOT NULL,
    diagnosis VARCHAR(200),
    treatment_plan TEXT,
    prescription TEXT,
    notes TEXT,
    visit_date DATE NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (patient_id) REFERENCES Patients(patient_id),
    FOREIGN KEY (staff_id) REFERENCES Medical_Staff(staff_id)
);

-- Insurance Providers
CREATE TABLE Insurance_Providers (
    provider_id INT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    contact_person VARCHAR(100),
    phone VARCHAR(15),
    email VARCHAR(100),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Billing Information
CREATE TABLE Billing (
    bill_id INT PRIMARY KEY,
    patient_id INT NOT NULL,
    appointment_id INT,
    amount DECIMAL(10,2) NOT NULL,
    insurance_claim_id VARCHAR(100),
    status VARCHAR(20) NOT NULL, -- Pending, Paid, Overdue, Insurance-Processing
    bill_date DATE NOT NULL,
    due_date DATE NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (patient_id) REFERENCES Patients(patient_id),
    FOREIGN KEY (appointment_id) REFERENCES Appointments(appointment_id)
);

-- Procedures/Services offered
CREATE TABLE Procedures (
    procedure_id INT PRIMARY KEY,
    name VARCHAR(200) NOT NULL,
    description TEXT,
    base_cost DECIMAL(10,2) NOT NULL,
    department_id INT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
```

```
    FOREIGN KEY (department_id) REFERENCES Departments(department_id)
);

-- Link table for procedures performed during appointments
CREATE TABLE Appointment_Procedures (
    appointment_id INT,
    procedure_id INT,
    notes TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (appointment_id, procedure_id),
    FOREIGN KEY (appointment_id) REFERENCES Appointments(appointment_id),
    FOREIGN KEY (procedure_id) REFERENCES Procedures(procedure_id)
);
```

5. Representative Queries

These queries demonstrate the following database's capabilities for more complex and interesting cases:

1. Complex aggregations and analytics (revenue analysis, age distributions)
2. Window functions for sequential analysis (staff workload)
3. Common Table Expressions (CTEs) for complex multi-step analysis
4. View creation for simplified access to commonly needed data
5. Temporal analysis (follow-up patterns, processing times)
6. Cross-table joins for comprehensive reporting
7. Conditional aggregations for status-based analysis
8. Nested queries and derived tables
9. Statistical analysis (percentiles, averages)
10. Performance monitoring metrics

```
-- 1. Total revenue from last month
SELECT SUM(amount) as total_revenue
FROM Billing
WHERE MONTH(bill_date) = MONTH(CURRENT_DATE - INTERVAL 1 MONTH)
AND YEAR(bill_date) = YEAR(CURRENT_DATE - INTERVAL 1 MONTH);

-- 2. Doctors with highest appointments next week
SELECT
    ms.first_name,
    ms.last_name,
    COUNT(*) as appointment_count
FROM Medical_Staff ms
JOIN Appointments a ON ms.staff_id = a.staff_id
WHERE a.appointment_date BETWEEN CURRENT_DATE AND CURRENT_DATE + INTERVAL 7 DAY
GROUP BY ms.staff_id
ORDER BY appointment_count DESC;

-- 3. Average wait time between scheduling and actual visit
SELECT
    AVG(DATEDIFF(appointment_date, created_at)) as avg_wait_days
```

```
FROM Appointments
WHERE status = 'Completed'
GROUP BY MONTH(appointment_date);

-- 4. Insurance providers with highest rejection rates
SELECT
    ip.name,
    COUNT(CASE WHEN b.status = 'Insurance-Processing' THEN 1 END) * 100.0 / COUNT(*) as rejection_rate
FROM Insurance_Providers ip
JOIN Patients p ON ip.provider_id = p.insurance_provider_id
JOIN Billing b ON p.patient_id = b.patient_id
GROUP BY ip.provider_id
ORDER BY rejection_rate DESC;

-- 5. Most common diagnoses for elderly patients
SELECT
    mr.diagnosis,
    COUNT(*) as diagnosis_count
FROM Medical_Records mr
JOIN Patients p ON mr.patient_id = p.patient_id
WHERE DATEDIFF(CURRENT_DATE, p.date_of_birth) >= 65 * 365
GROUP BY mr.diagnosis
ORDER BY diagnosis_count DESC
LIMIT 10;

-- 6. Patients who missed follow-up appointments in last quarter
WITH FollowUps AS (
    SELECT
        a1.patient_id,
        a1.appointment_date as initial_visit,
        MIN(a2.appointment_date) as follow_up_date
    FROM Appointments a1
    LEFT JOIN Appointments a2 ON a1.patient_id = a2.patient_id
        AND a2.appointment_date > a1.appointment_date
    GROUP BY a1.patient_id, a1.appointment_date
)
SELECT
    p.first_name,
    p.last_name,
    f.initial_visit,
    f.follow_up_date,
    DATEDIFF(f.follow_up_date, f.initial_visit) as days_between_visits
FROM FollowUps f
JOIN Patients p ON f.patient_id = p.patient_id
WHERE f.follow_up_date IS NULL
AND f.initial_visit >= DATE_SUB(CURRENT_DATE, INTERVAL 3 MONTH);

-- 7. Department revenue analysis with procedure breakdown
SELECT
```

```
    d.name as department_name,
    p.name as procedure_name,
    COUNT(*) as procedure_count,
    SUM(b.amount) as total_revenue,
    AVG(b.amount) as avg_revenue_per_procedure
FROM Departments d
JOIN Procedures pr ON d.department_id = pr.department_id
JOIN Appointment_Procedures ap ON pr.procedure_id = ap.procedure_id
JOIN Appointments a ON ap.appointment_id = a.appointment_id
JOIN Billing b ON a.appointment_id = b.appointment_id
GROUP BY d.department_id, pr.procedure_id
WITH ROLLUP;

-- 8. Patient age distribution across departments with percentile analysis
WITH PatientAges AS (
    SELECT
        d.name as department_name,
        FLOOR(DATEDIFF(CURRENT_DATE, p.date_of_birth) / 365) as age,
        COUNT(*) as patient_count
    FROM Patients p
    JOIN Appointments a ON p.patient_id = a.patient_id
    JOIN Medical_Staff ms ON a.staff_id = ms.staff_id
    JOIN Departments d ON ms.department_id = d.department_id
    GROUP BY d.name, FLOOR(DATEDIFF(CURRENT_DATE, p.date_of_birth) / 365)
)
SELECT
    department_name,
    AVG(age) as avg_age,
    MIN(age) as youngest,
    MAX(age) as oldest,
    SUM(patient_count) as total_patients
FROM PatientAges
GROUP BY department_name;

-- 9. Complex analysis of treatment effectiveness
WITH PatientTreatments AS (
    SELECT
        mr1.patient_id,
        mr1.diagnosis,
        mr1.treatment_plan,
        COUNT(mr2.record_id) as follow_up_visits,
        MAX(CASE WHEN mr2.diagnosis = mr1.diagnosis THEN 1 ELSE 0 END) as condition_persisted
    FROM Medical_Records mr1
    LEFT JOIN Medical_Records mr2 ON mr1.patient_id = mr2.patient_id
        AND mr2.visit_date > mr1.visit_date
    GROUP BY mr1.patient_id, mr1.diagnosis, mr1.treatment_plan
)
SELECT
    diagnosis,
```

```
    treatment_plan,
    COUNT(*) as total_patients,
    AVG(follow_up_visits) as avg_follow_ups,
    SUM(condition_persisted) * 100.0 / COUNT(*) as persistence_rate
FROM PatientTreatments
GROUP BY diagnosis, treatment_plan
HAVING COUNT(*) >= 10
ORDER BY persistence_rate ASC;

-- 10. Insurance claim analysis with processing time metrics
SELECT
    ip.name as insurance_provider,
    COUNT(*) as total_claims,
    AVG(DATEDIFF(
        CASE WHEN b.status = 'Paid' THEN b.created_at ELSE CURRENT_TIMESTAMP END,
        b.created_at
    )) as avg_processing_days,
    SUM(CASE WHEN b.status = 'Paid' THEN b.amount ELSE 0 END) as total_paid,
    SUM(CASE WHEN b.status = 'Insurance-Processing' THEN b.amount ELSE 0 END) as total_pending
FROM Billing b
JOIN Patients p ON b.patient_id = p.patient_id
JOIN Insurance_Providers ip ON p.insurance_provider_id = ip.provider_id
WHERE b.insurance_claim_id IS NOT NULL
GROUP BY ip.provider_id
ORDER BY avg_processing_days DESC;

-- 11. Staff workload and efficiency analysis
SELECT
    ms.first_name,
    ms.last_name,
    ms.specialization,
    COUNT(DISTINCT a.appointment_id) as total_appointments,
    COUNT(DISTINCT p.patient_id) as unique_patients,
    AVG(TIMESTAMPDIFF(MINUTE,
        a.appointment_date,
        LEAD(a.appointment_date) OVER (PARTITION BY ms.staff_id ORDER BY a.appointment_date)
    )) as avg_time_between_appointments,
    SUM(b.amount) as total_revenue_generated
FROM Medical_Staff ms
LEFT JOIN Appointments a ON ms.staff_id = a.staff_id
LEFT JOIN Patients p ON a.patient_id = p.patient_id
LEFT JOIN Billing b ON a.appointment_id = b.appointment_id
WHERE a.appointment_date >= DATE_SUB(CURRENT_DATE, INTERVAL 1 MONTH)
GROUP BY ms.staff_id;

-- 12. Comprehensive patient history view
CREATE VIEW Patient_History AS
SELECT
    p.patient_id,
```

```
    p.first_name,
    p.last_name,
    COUNT(DISTINCT a.appointment_id) as total_visits,
    GROUP_CONCAT(DISTINCT mr.diagnosis) as all_diagnoses,
    SUM(b.amount) as total_bills,
    SUM(CASE WHEN b.status = 'Paid' THEN b.amount ELSE 0 END) as total_paid,
    MAX(a.appointment_date) as last_visit,
    COUNT(DISTINCT ms.staff_id) as different_doctors_seen
FROM Patients p
LEFT JOIN Appointments a ON p.patient_id = a.patient_id
LEFT JOIN Medical_Records mr ON p.patient_id = mr.patient_id
LEFT JOIN Billing b ON p.patient_id = b.patient_id
LEFT JOIN Medical_Staff ms ON a.staff_id = ms.staff_id
GROUP BY p.patient_id;
```

6. Data Loading Plan

- Our database will use a combination of pre-approved hospital data and simulated datasets. Online sources like mock healthcare APIs may be used
- We may look into developing a script to clean and transform the data into our required schema format
- We are also deciding if it would be useful to implement a data input feature through a web-based interface.

Challenges: Formatting inconsistencies, data cleaning for missing or incorrect values, and possible mappings of external formats to relational tables.

7. Output Plans

1. Create an interactive web interface using HTML/CSS for data viewing and entry
2. Generate basic reports in CSV format showing:
    a. Monthly appointment summaries
    b. Patient statistics
    c. Billing reports
3. Implement custom SQL views for common data access patterns

8. Specialized Topics

Major Focus: User-friendly and advanced GUI form interface with report generation feature

- Core Features:
    - Interactive patient registration form with validation
    - Basic dashboard showing daily/weekly appointment statistics
    - Automated report generation for:
        - Monthly billing summaries
        - Patient visit history
        - Doctor schedules
    - Possibly a simple data visualization view for:
        - Patient demographics

        - Department workload
        - Revenue trends
    - Appointment scheduling calendar view functionality

Minor Focus: Advanced SQL Topics (Triggers, JDBC, etc.)

- Focused Implementation:
    - Basic triggers for:
        - Logging patient record updates
        - Updating appointment status changes
    - Stored procedures for:
        - Generating common reports
        - Processing new patient registration