1 Abstract

Abstract essentially provides a concise summary of your work Our code and data is available online on Github.

The data and code are available on Github. [1]

2 Introduction

Introduction discusses the motivation and objectives of your work referring to the relevant, up-to-date / most recent / state-of-the-art academic literature and/or products / systems. Concerning the academic resources, you can reach those related articles through Google Scholar / Baidu Scholar.

The proposed Hospital Management System represents a sophisticated, nuanced approach to healthcare information management that distinguishes itself through several innovative features. Unlike traditional hospital databases that often offer monolithic solutions, this system introduces a dynamic and flexible framework for healthcare service tracking, focuses more on the "objects"(which means the service they serve) and more suitable to small-sized hospitals. This approach aligns with recent scholarly insights into digital transformation in healthcare, as highlighted by Zhang et al. (2019) in their seminal work on adaptive healthcare information systems.

1. Individual Service Classification: A standout feature of this system is the sophisticated service management model. The 'Services' table uses 'Passive' and 'Active' service types, which enables more precise tracking of healthcare interventions. This allows for more accurate tracking of hospital faculty and also allows the employees to multitask for the passive service types.

2. Comprehensive Resource Tracking: The database design goes beyond basic patient records by implementing intricate relationships between patients, employees, services, and hospital resources. The inclusion of tables like 'Patient Stay', 'Beds', and 'Billing' creates a connected ecosystem for tracking patient journeys from CheckIn to CheckOut.

3. Flexible Employee-Service Interaction: The 'Employ Serv' table introduces a flexible many-to-many relationship between employees and services, allowing for dynamic service allocation and specialized employee skill mapping that many existing systems do not support.

4. Enhanced Security and Data Integrity: Rigorous data validation is embedded at the schema level. Constraints such as prefix checks (e.g., 'CHECK (PID LIKE 'P')') and strongly typed enums for patient sex, employee type, and payment status ensure data consistency and prevent invalid entries.

5. Modular Medical Record Management: The separation of medical records ('Med Rec') from patient records ('Patient Recs') provides a flexible mechanism for managing medical history while maintaining referential integrity and allowing multiple records per patient. The modular design enables more nuanced access

[1] Github Link: https://github.com/SoumyaLahoti/COMPSC310FinalProject

management. A study by Aziz et al. (2017) in the IEEE Journal of Biomedical and Health Informatics highlights how separated record structures can enhance patient data privacy and selective information sharing. The architecture also allows multiple medical records per patient, addressing the complex medical histories of patients with chronic conditions or multiple treatment episodes. This aligns with recommendations from the International Medical Informatics Association (IMIA) for comprehensive patient data management. [1]

3 Problem

Problem explains the domain you focus on. This part should provide a formal denition, likely using some mathematical terms when possible, besides describing the problem in words

The project focuses on the domain of hospital management, which involves organizing, storing, and retrieving critical data to facilitate efficient healthcare operations. A hospital database system serves as the backbone for managing patient information, staff records, appointment scheduling, billing, and inventory tracking. The key challenge lies in designing a relational database that can handle the complexities of these interrelated data points while ensuring data integrity and accessibility.

Formally, let the hospital database be represented as a relational database schema

$$S = \{R_1, R_2, \ldots, R_n\},$$

where each R_i is a relation (table) defined over a finite set of attributes

$$A_i = \{a_1, a_2, \ldots, a_m\}.$$

Each relation satisfies certain integrity constraints, including:

1. **Entity Integrity**: Ensures every table has a primary key $P_i \subseteq A_i$, where P_i uniquely identifies each tuple $t_i \in R_i$.
2. **Referential Integrity**: Guarantees that any foreign key F_i in R_i corresponds to a primary key in another relation R_j, ensuring consistent relationships across tables.

The hospital database system was implemented using **SQL** for designing the database schema, defining relationships, and writing Data Definition Language (DDL) scripts to create and manage tables. SQL queries were developed to handle essential operations such as retrieving patient records, scheduling appointments, and generating bills. These queries allowed for efficient data manipulation and ensured accurate relationships between patients, doctors, and treatments. The database was connected to a user interface designed with **Python**, using the **Tkinter** library for the front end. This integration enabled users to interact with the database seamlessly, providing functionalities like inputting new data, querying existing records, and displaying results in an intuitive graphical interface.

4 Method

Method / System will explain the ideas to be applied. Similarly to the Problem section, a formal denition is essential especially if you are utilizing some algorithmic approaches. Furthermore, the complete system design with the reasoning behind your choices, the developed sub-components and how they interact with each other, should be reported. This part should be specically supported by ow charts showing how the system works and screenshots illustrating some representative use-cases. More importantly, class diagrams (if applicable) and explicit database design should be provided

The system is designed to manage hospital data efficiently by utilizing a relational database structure to store, retrieve, and manipulate information related to patients, employees, services, billing, and medical records. The system was implemented using **SQL** for the database and **Python** (specifically **Tkinter**) for the front-end user interface. The design focuses on data integrity, scalability, and ease of interaction for hospital staff.

4.1 System Architecture Overview

The overall system can be broken down into two main components: the **database** and the **front-end interface**. The database is responsible for managing the hospital's data, while the front-end interface provides a user-friendly platform for interacting with the database. Below is a more detailed description of each component:

Database Design The database is structured around a set of interconnected tables representing different aspects of the hospital, including patients, employees, services, departments, billing, and medical records. Relationships between entities are represented by foreign keys, ensuring data consistency and integrity. For example, a patient's medical record is linked to the patient's ID, and services availed by a patient are linked to both the patient and the corresponding employee. **Normalization** has been applied to minimize data redundancy while maintaining data integrity and efficiency. The use of primary and foreign keys ensures that all references are valid and consistent. The tables are connected in a way that supports multiple relationships, including one-to-many and many-to-many, such as the relationship between patients and services availed, and between employees and services provided.

Python and Tkinter Interface The front-end interface is built using **Python** with **Tkinter** for graphical user interface (GUI) development. Tkinter was chosen due to its simplicity and integration with Python, allowing for rapid development and deployment. The front-end allows users to interact with the database by providing various options for querying and updating records. It enables hospital staff to add new patients, assign them to beds, track services availed, generate billing information, and maintain medical records. Each operation in the

interface corresponds to an action that directly manipulates the database. For instance, when a new patient is admitted, the system records the patient's information in the `Patients` table and assigns a bed in the `Patient_Stay` table.

4.2 Interaction Between Components

The components interact in a way that ensures smooth operation of the hospital's data management system. When a user interacts with the GUI (e.g., by entering new data), the corresponding SQL queries are executed to update the database. The flow of interactions can be summarized as follows:

1. **Step 1: User Input**: The user inputs data (such as patient information or service details) into the Tkinter interface.
2. **Step 2: Query Execution**: Upon submission, Python executes SQL queries that update or retrieve the data from the database. These queries are constructed based on the user's input and the system's logic.
3. **Step 3: Data Retrieval/Update**: The database returns the requested data, or the system updates the data in the database.
4. **Step 4: Output**: The front-end interface displays the updated or retrieved data to the user, providing feedback in real-time.

4.3 Sub-components

Patient Management The patient management module allows the hospital to track patient records, including personal information, medical history, and services availed. The user can add, update, and view patient details. The data is stored in the `Patients` table and is connected to other tables like `Med_Rec`, `Services_Availed`, and `Patient_Stay`.

Employee Management The employee module helps manage hospital staff, such as doctors and nurses. It includes features for adding employees, updating their information, and assigning them to specific departments and services. Employee records are stored in the `Employees` table, with department associations linked via the `Dept_ID` foreign key.

Service Management The service management component handles the services provided by the hospital, such as consultations, surgeries, and tests. Services are tracked through the `Services` table, and their associations with patients and employees are stored in the `Services_Availed` and `Employ_Serv` tables. The service module allows users to add, update, and view service details, along with cost information and service types.

Billing The billing system manages patient charges, keeping track of the total cost for services availed during their stay. Billing information is stored in the `Billing` table, with connections to patient records and bed assignments through foreign keys. The module allows users to generate bills based on the services availed and patient stays, ensuring that all charges are correctly calculated and displayed.

Medical Records Medical records are stored in the `Med_Rec` table, which contains information about diagnoses, treatments, and ailments. The `Patient_Recs` table links medical records to individual patients. Users can input medical diagnoses and treatment information, which is then linked to the corresponding patient record.

4.4 Database Design and Relationships

The design follows relational database principles, and each table serves a specific role in managing the hospital's operations. The relationships between tables are defined using foreign keys, and normalization techniques have been applied to minimize data redundancy and maintain consistency. Below is an explicit explanation of the table design and key relationships:

- The `Patients` table stores patient details and is linked to the `Services_Availed`, `Patient_Stay`, and `Billing` tables. The `PID` foreign key connects these tables.
- The `Departments` and `Employees` tables are linked via the `Dept_ID` foreign key, allowing for the assignment of employees to specific departments.
- The `Services` table stores information about hospital services, and the `Services_Availed` and `Employ_Serv` tables manage the many-to-many relationships between patients, employees, and services.

4.5 Flowchart

A flowchart can be used to visually represent the main process flow in the system:

1. **Patient Admission Process**:
   - User enters patient information → Store in `Patients` table → Assign a bed in `Beds` table → Generate billing information in `Billing` table.
2. **Service Availment Process**:
   - User enters service details → Store in `Services_Availed` table → Link with patient and employee records → Update billing if necessary.

4.6 Screenshots of Use-Cases

1. **Patient Registration**: A screenshot illustrating how users can input patient information, including the patient's name, date of birth, and medical history.
2. **Billing Overview**: A screenshot showing how the system generates and displays billing information for a patient.

4.7 Billing View

Method and System Design for GetBillingInfo Procedure. The GetBillingInfo procedure is a sophisticated billing information retrieval method designed to generate the billing view for patient services during a specific hospital stay. This procedure integrates multiple database tables to create a detailed, time-sensitive billing report that captures the nuanced aspects of patient service utilization.

Detailed Procedure Workflow *Parameters*

- Patient Identifier (PID): A unique code that identifies the specific patient
- Check-In Date and Time: The timestamp marking the beginning of the patient's hospital stay

Comprehensive Data Integration Process The procedure performs a complex multi-table join that brings together critical information from several database tables. This integration allows for a comprehensive view of the patient's service utilization by connecting:

- Services Availed: Tracking the specific services used
- Patient Stay: Documenting the patient's hospital admission details
- Services: Providing information about each service
- Employees: Identifying the healthcare professionals involved

Dynamic Cost Calculation The procedure implements a sophisticated cost calculation method, particularly for specific service types. For services with IDs 'S017' and 'S019', the cost is dynamically calculated by multiplying the base cost with the actual duration of the service. This approach ensures more accurate and fair billing for time-dependent services.

Procedure First: Call the data existing in the Services Availed Table. Second: Using call function, providing PID and Start. PID is in the form of a six digits string begin with P. and Start is in the format of DATETIME.

Examples

4.8 GetAvailableBeds Procedure

The 'GetAvailableBeds' procedure is designed to check for available beds of a specific type in the hospital. The procedure takes two parameters: the preferred bed type (e.g., "single" or "double") and the check-in time to determine whether the bed is available for a given patient at the requested time.

Detailed Procedure Workflow *Parameters*

- Preferred Bed Type: A string that specifies the type of bed preferred by the patient (e.g., 'single', 'double'). - Check-In Time: The timestamp indicating the patient's check-in time.

Comprehensive Data Integration Process The procedure performs a complex multi-step process that integrates data from multiple tables: - Beds Table: Retrieves information on bed availability and types. - Patient Stay Table: Identifies which beds are currently occupied. - Temporary Tables: Dynamically creates temporary tables to simplify filtering and checking the availability of beds for the given time.

Procedure Steps

1. Create Temporary Bed Types Table: This table is dynamically generated to categorize beds based on their room type (single or double).
2. Create Active Patient Stay Table: This table identifies the beds currently occupied by active patients (patients who have not yet checked out).
3. Filter Available Beds: Using a LEFT JOIN, the procedure checks for available beds of the preferred type by ensuring that no current patient is occupying the bed at the requested check-in time.
4. Drop Temporary Tables: Once the operation is complete, the temporary tables are dropped to clean up.

Example SQL Query Here's the full 'GetAvailableBeds' SQL procedure:

```
DELIMITER //
DROP PROCEDURE IF EXISTS GetAvailableBeds;
CREATE PROCEDURE GetAvailableBeds(
    IN preferred_bed_type VARCHAR(10),
    IN check_in_time DATETIME
)
BEGIN
    -- Step 1: Create Bed Types dynamically
    CREATE TEMPORARY TABLE Temp_Bed_Types AS
    SELECT
        b.Bed_ID,
        b.Room_ID,
        CASE
            WHEN COUNT(b.Bed_ID) = 1 THEN 'single'
            WHEN COUNT(b.Bed_ID) = 2 THEN 'double'
        END AS Bed_Type
    FROM Beds b
    GROUP BY b.Room_ID;

    -- Step 2: Create Active Patient Stay dynamically
    CREATE TEMPORARY TABLE Temp_Active_Patient_Stay AS
    SELECT Bed_ID
    FROM Patient_Stay
    WHERE CheckOut IS NULL;
```

```
    -- Step 3: Filter Available Beds
    SELECT b.Bed_ID, b.Room_ID, b.Bed_Type
    FROM Temp_Bed_Types b
    LEFT JOIN Temp_Active_Patient_Stay aps ON b.Bed_ID = aps.Bed_ID
    WHERE aps.Bed_ID IS NULL  -- Bed is not currently occupied
      AND b.Bed_Type = preferred_bed_type
      AND NOT EXISTS (
          SELECT 1
          FROM Patient_Stay ps
          WHERE ps.Bed_ID = b.Bed_ID
          AND ps.CheckOut IS NULL
          AND (
              ps.CheckIn <= check_in_time AND (ps.CheckOut IS NULL OR ps.CheckOut >= check_i
          )
      );

    -- Clean up
    DROP TEMPORARY TABLE IF EXISTS Temp_Bed_Types;
    DROP TEMPORARY TABLE IF EXISTS Temp_Active_Patient_Stay;
END //
```

The ‘SearchPatients‘ procedure is designed to search for patients in the ‘Patients‘ table based on various search criteria. The procedure uses the parameters ‘search pid‘, ‘search dob‘, ‘search first name‘, and ‘search last name‘ to filter patients by their unique identifiers, date of birth, and names. This allows for flexible searching where parameters can be passed as ‘NULL‘ to ignore certain criteria.

Detailed Procedure Workflow *Parameters*

- search pid: A unique code that identifies the specific patient (e.g., ’P001’). - search dob: The patient’s date of birth (e.g., ’1980-05-15’). - search first-name: The patient’s first name, or a part of it (e.g., ’John’). - search-last-name: The patient’s last name, or a part of it (e.g., ’Doe’).

Comprehensive Data Integration Process The procedure uses conditional logic to apply filters based on the provided parameters. It performs a query on the ‘Patients‘ table where: - The ‘PID‘ is matched if ‘search pid‘ is provided. - The ‘DayOfBirth‘ is matched if ‘search dob‘ is provided. - The ‘Name‘ is filtered using the ‘LIKE‘ operator to match parts of the first and last names.

Procedure Steps

1. Filter by Patient ID: If ‘search pid‘ is provided, the query matches the ‘PID‘ of the patient.

2. Filter by Date of Birth: If ‘search dob‘ is provided, the query matches the ‘DayOfBirth‘.
3. Filter by First Name: If ‘search first name‘ is provided, the query searches for the first name using the ‘LIKE‘ operator with wildcards to match partial names.
4. Filter by Last Name: Similar to first name, if ‘search last name‘ is provided, the query searches for the last name using the ‘LIKE‘ operator.
5. Return the Results: The query returns the rows that match the filters based on the parameters provided.

Example SQL Query Here’s the full ‘SearchPatients‘ SQL procedure:

```
DELIMITER //
DROP PROCEDURE IF EXISTS SearchPatients;
CREATE PROCEDURE SearchPatients(
    IN search_pid VARCHAR(4),
    IN search_dob DATE,
    IN search_first_name VARCHAR(25),
    IN search_last_name VARCHAR(25)
)
BEGIN
    SELECT *
    FROM Patients
    WHERE
        (search_pid IS NULL OR PID = search_pid) AND
        (search_dob IS NULL OR DayOfBirth = search_dob) AND
        (search_first_name IS NULL OR Name LIKE CONCAT('%', search_first_name, '%')) AND
        (search_last_name IS NULL OR Name LIKE CONCAT('%', search_last_name, '%'));
END //
DELIMITER ;
```

Trigger: AfterCheckOutUpdate Once the procedure is called with any combination of ‘search paid‘, ‘search dob‘, ‘search first name‘, and ‘search last name‘, it returns a list of matching patients. This allows flexible searching for patients, even if only partial information is provided.

Result Once the procedure is called with the preferred bed type and check-in time, it returns a list of available beds that match the criteria. This allows for efficient room allocation based on patient preferences and hospital availability.

5 Evaluation

With Computational Results / Evaluation, if your system has empirical aspects, it needs to be tested and the results should be reported while discussing its advantages and disadvantages. When possible, it should be compared against the

existing state-of-the-art / best approaches from the literature. Otherwise, the system should be tested with dierent use-cases to show its performance and capabilities, possibly including screenshots. All should be reported with an in-depth discussion. 3

6 Conclusion

The report should be nalized with the Conclusion section, briey summarizing the developed system alongside with the results / evaluation, and take-home messages. It should additionally oer follow-up ideas on what to do next

7 Appendix

In addition to the main report, an Appendix section is required, outlining the specic contributions made by each team member. A weekly project timeline should also be included, highlighting key accomplishments, obstacles encountered, and the strategies employed to overcome them.

7.1 Soumya

Soumya built the database and did significant research on hospital methods and how hospital systems function. She was critical for peer questions during the presentation and also had significant contributions to the peer review document. She came up with several query ideas but was unable to implement all of them but they way the trigger, views and queries work are clear in her head.

7.2 Pauline

Pauline was largely involved in planning, creating the ER model on paper, as well as a structure and plan for the frontend, making it more clear what queries are necessary for the functioning of the program. Pauline also helped largely with using Github. On the database side, Pauline wrote the trigger for inserting new tuples into Billing, viewing and updating the patient information, showing the bills and medical records for a chosen patient, finding out what employees are permitted to do what services, filtering employees based on specific criteria, and adding entries into the $\text{Services}_{A}vailedtable$.

7.3 Lishi

Individual: Three queriers: GetAvailableBeds, GetBillingInfo, AfterCheckOutUpdate. Collaboration: Help to put down the DDL table. Design some of the front page(but they are not connected to sql yet), join on collaboration of relational design

Acknowledgments. A bold run-in heading in small font size at the end of the paper is used for general acknowledgments, for example: This study was funded by X (grant number Y).

Disclosure of Interests. It is now necessary to declare any competing interests or to specifically state that the authors have no competing interests. Please place the statement with a bold run-in heading in small font size beneath the (optional) acknowledgments[2], for example: The authors have no competing interests to declare that are relevant to the content of this article. Or: Author A has received research grants from Company W. Author B has received a speaker honorarium from Company X and owns stock in Company Y. Author C is a member of committee Z.

References

1. Aziz, N., Srinivasan, S., Kumar, R.: Innovative approaches to electronic health record privacy management. IEEE Journal of Biomedical and Health Informatics **21**(4), 1024–1035 (2017)

[2] If EquinOCS, our proceedings submission system, is used, then the disclaimer can be provided directly in the system.

```sql
-- Create the procedure
CREATE PROCEDURE GetBillingInfo(
    IN input_PID VARCHAR(4),
    IN input_CheckIn DATETIME
)
BEGIN
    SELECT
        ps.Bill_ID,
        ps.PID,
        ps.CheckIn AS CheckIn_DateTime,
        sa.Start AS Start_Service,
        sa.End AS End_Service,
        s.Service_Name,
        -- Calculate the cost based on Service_ID
        CASE
            WHEN sa.Service_ID IN ('S017', 'S019') THEN
                s.Cost * TIMESTAMPDIFF(SECOND, sa.Start, sa.End) / 3600 -- Cost adjusted by dura
            ELSE
                s.Cost
        END AS Adjusted_Cost,
        e.Name AS Employee_Name,
        p.Name AS Patient_Name
    FROM
        Services_Availed sa
    NATURAL JOIN Patient_Stay ps
    NATURAL JOIN Services s
    LEFT JOIN Employees e ON sa.EID = e.EID
    LEFT JOIN Patients p ON ps.PID = p.PID
    WHERE
        ps.PID = input_PID
        AND ps.CheckIn = input_CheckIn
        AND (sa.Start BETWEEN ps.CheckIn AND IFNULL(ps.CheckOut, CURRENT_TIMESTAMP))
        AND (sa.End BETWEEN ps.CheckIn AND IFNULL(ps.CheckOut, CURRENT_TIMESTAMP));
END //
```

Fig. 1. BillView Code

```sql
CALL GetBillingInfo('P001', '2024-01-01 12:00:00');
```

Fig. 2. Calling BillView

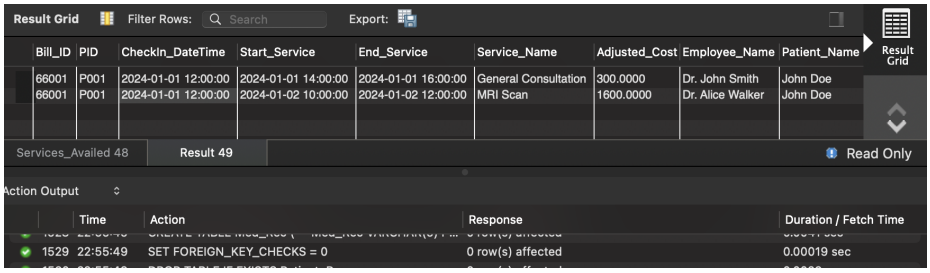

Fig. 3. BillView Result