Healthcare Patient Data Processing System using Python & SQLite

Topics:

Python

SQLite

DDL (Data Definition Language)

DML (Data Manipulation Language)

Error Handling

FIFO (Queue)

LIFO (Stack)

Multithreading

Scenario / Problem Statement:

In a hospital environment, multiple operations such as patient registration, OPD consultation, lab report processing, and billing occur simultaneously.

Handling these processes manually may lead to delays, data inconsistency, and errors.

This project implements an automated healthcare system using Python and SQLite to simulate real-time hospital operations efficiently.

Roles and Responsibilites:

Patient: Provides personal and medical information

Doctor: Consults patients based on FIFO order

Lab Technician: Processes lab reports using LIFO principle

Billing Department: Handles patient billing

System: Manages database, queue, stack, and multithreading

Code Review and Topics Integrated in the Code:

✓ **Database Connection (SQLite**):

The program starts by importing the sqlite3 module to connect with an SQLite database.

A database named hospital.db is created or opened.

A cursor object is used to execute SQL commands.

Purpose:

To store and manage patient data permanently.

✓. DDL – Table Creation:

The CREATE TABLE IF NOT EXISTS patients statement is used to create the patient table structure.

It ensures that the table is created only if it does not already exist.

Purpose:

To define the database schema for storing patient information.

✓. DML – Insert and Update Operations:

The **INSERT** statement is used to add new patient records into the database.

The **UPDATE** statement is used to modify existing patient details.

INSERT OR IGNORE prevents duplicate patient IDs from causing errors.

Purpose:

To manage patient records effectively.

✓. Error Handling:

The try–except block is used to handle runtime errors safely.

If an invalid age (zero or negative) is entered, a custom error is raised.

Errors are displayed without terminating the program.

Purpose:

To prevent program crashes and ensure data validation.

✓. FIFO – OPD Patient Queue:

The FIFO principle (First In, First Out) is implemented using the Queue class.

Patients are added using put() and consulted using get().

The first patient to arrive is consulted first.

Purpose:

To manage OPD patient flow in a fair and orderly manner.

✓. LIFO – Lab Report Processing:

The LIFO principle (Last In, First Out) is implemented using a Python list as a stack.

Lab reports are added using append() and processed using pop().

The most recent report is processed first.

Purpose:

To prioritize emergency and recent lab reports.

✓. Multithreading – Parallel Hospital Operations:

The threading module is used to execute multiple hospital operations simultaneously.

Patient registration, billing, and lab processing run in separate threads.

start() begins thread execution and join() ensures all threads complete before program exit.

Purpose:

To save time and simulate real-time hospital operations.

✓. Final Data Display and Database Closure

All patient records are retrieved from the database and displayed.

The database connection is safely closed using conn.close().

Purpose:

To release resources and avoid memory leaks.

Output:

Patient table is created successfully

Patient records are inserted and updated

Invalid patient entries are handled safely

OPD patients are consulted in FIFO order

Lab reports are processed in LIFO order

Hospital operations run in parallel using multithreading

Interview Questions Discussed:

- What is DDL and DML?
- Why is error handling important?
- What is the difference between FIFO and LIFO?
- What is multithreading and why is it used?
- Why is join() used in threading?
- What are real-time applications of queue and stack?