# Healthcare Organization Database

Dylan Kayyem

3287 - Project Proposal - Part 2

*https://github.com/dylankayyem/3287_project*

## Tools & SQL Interface:

[ 1 ] **Database Management System (DBMS):** MySQL Server will be the chosen DBMS for its robustness, widespread use, and support for complex transactions and security features.

[ 2 ] **Database Interface:** To interact with the MySQL database from the Jupyter Notebook, we'll use the 'mysql.connector' Python module in combination with 'configparser' to read database configuration from a file.

[ 3 ] **Interactive Python Environment:** Jupyter Notebook provides an interactive environment where Python code, SQL queries, explanatory text, and visualizations coexist on one document.

[ 4 ] **Data & Schema Modeling:** I will use the ORM tool, SQLAlchemy, to define the database schema in Python classes, which will provide an abstraction layer for the database interactions.

[ 5 ] **Data Aggregation and Reporting:** Pandas will be used within the Jupyter Notebook for data aggregation tasks. For visualization and further data analysis, we can leverage libraries such as Plotly, Seaborn, Matplotlib within the Notebook.

[ 4 ] **Triggers and Stored Procedures:** Using MySQL allows for the creation of triggers and procedures using its own language, which resembles SQL. However, these can also be executed from the Jupyter Notebook via 'mysql.connector'.

[ 5 ] **Security:** MySQL provides robust security features including encrypted connections, SSL support, and access control based on user roles. This will be configured on the MySQL server, ensuring the data is protected.

[ 6 ] **Demonstration of Concepts:** Jupyter Notebook will serve as a platform to demonstrate database concepts. Code cells will be used to establish and test database connections, create tables, insert and query data.

✓ Each step will be documented with Markdown cells explaining the rationale behind schema design decisions, the execution of SQL queries, the use of indexing, and the demonstration of transactional integrity.

✓ The visualization tools integrated into the Jupyter Notebook will allow for the creation of live graphs and charts to illustrate data relationships and query results.

## Relationships:

To effectively manage the data in the Healthcare Organization Database, my project aims to establish relationships that mirror the logical association between healthcare professionals. This would involve setting primary and foreign keys to link data in a relational database.To relate the tables, I would use the primary key and foreign key relationships as specified [1-6]:

[ 1 ] Each ‘Patient’ is uniquely identified by their ‘Patient_ID’, which is used as a foreign key in the ‘Payment’ and ‘Call_Record’
[ 2 ] tables to link **‘Payments’** and **‘Calls’** to the respective patient.

[ 3 ] The ‘Doctor’ table, with unique ‘Doctor_ID’ for each doctor, is connected to the ‘Patient’ table to indicate which doctor has **‘Diagnosed’** which patient, showcasing a one-to-many relationship since one doctor can diagnose multiple patients.

[ 4 ] For the **‘Admitted_To’** relationship, the ‘Patient’ table would have a foreign key ‘Room_Number’ that references the primary key in the ‘Room’ table, indicating the room to which the patient has been admitted.

[ 5 ] As for the **‘Assigned_To’** relationship, both ‘Nurse’ and ‘Ward_Boy’ entities have a ‘Doctor_ID’ foreign key that references the ‘Doctor’ table, signifying which doctor they are assigned to.

[ 6 ] The **‘Cares’** relationship is many-to-many between ‘Nurse’/’Ward_Boy’ and ‘Patient’, and this necessitates an associative table, ‘Care_Assignment’, which contains ‘CareProvider_ID’ and ‘Patient_ID’ as foreign keys to connect caregivers with their patients.

## Relationships Cont.:

[ 1 ] **Diagnose:**

- ✓ A doctor can diagnose multiple patients, but each diagnosis for a patient is by 1 doctor.
- ✓ One-to-Many (from Doctor to Patient)

[ 2 ] **Admitted_To:**

- ✓ A patient can be admitted to 1 room, but a room can admit multiple patients over time.
- ✓ Many-to-One (from Patient to Room)

[ 3 ] **Assigned_To:**

- ✓ A nurse/ward boy is assigned to 1 doctor, but a doctor can have multiple nurses and ward boys.
- ✓ Many-to-One (from Nurse/Ward_Boy to Doctor)

[ 4 ] **Cares:**

- ✓ A nurse/ward boy can care for multiple patients, and a patient can be cared for by multiple nurses/ward boys.
- ✓ Many-to-Many (from Nurse/Ward_Boy to Patient)

[ 5 ] **Payments:**

- ✓ A patient can make multiple payments, but each payment is made by 1 patient.
- ✓ One-to-Many (from Patient to Payment)

[ 6 ] **Calls:**

- ✓ An existing patient can make multiple calls, but each call record is associated with 1 patient or none if new.
- ✓ One-to-Many (from Patient to Call_Record)

## Entities & Attributes:

[ 1 ] **'Patient' Table:**

- ■ Patient_ID (Primary Key)
- ○ Name
- ○ Address
- ○ Phone
- ○ Email
- ○ Date_Of_Registration
- ○ Disease_Diagnosed
- ○ Prescription
- ○ Room_Number *(if admitted)*
- ○ Payment_Status

[ 2 ] **'Doctor' Table:**

- ■ Doctor_ID (Primary Key)
- ○ Name
- ○ Phone
- ○ Email
- ○ Specialization
- ○ Availability

[ 3 ] **'Room' Table:**

- ■ Room_Number (Primary Key)
- ○ Type *(ICU / General / etc.)*
- ○ Status *(Vacant / Occupied)*
- ○ Date_Of_Status

[ 4 ] **'Payment' Table:**

- ■ Payment_ID (Primary Key)
- ● Patient_ID (Foreign Key)
- ○ Amount
- ○ Date_Of_Payment
- ○ Payment_Method *(Cash / E-banking)*

[ 5 ] **'Nurse' Table:**

- ■ Nurse_ID (Primary Key)
- ● Doctor_ID (Foreign Key)
- ○ Name
- ○ Phone
- ○ Email

[ 6 ] **'Ward_Boy' Table:**

- ■ Ward_ID (Primary Key)
- ● Doctor_ID (Foreign Key)
- ○ Name
- ○ Phone
- ○ Email

[ 7 ] **'Call_Record' Table:**

- ■ Call_ID (Primary Key)
- ● Patient_ID (Foreign Key, if existing patient)
- ○ Caller_Phone
- ○ Date_Time_Of_Call
- ○ Purpose *(Support / Help)*
- ○ Description

[ 8 ] **'Care_Assignment' Table:**

- ■ Care_ID (Primary Key)
- ● CareProvider_ID (Foreign Key, *could be either Nurse_ID / Ward_ID)*
- ○ CareProvider_Type *('Nurse' / 'WardBoy'. This helps identify whether the ID references a nurse or a ward boy)*
- ○ Patient_ID (Foreign Key Referencing Patient))
- ○ Start_Date
- ○ End_Date

## Combination & Aggregation:

- ✓ To demonstrate the application of aggregation techniques within the database, I will utilize SQL queries with JOIN operations. For example, to show which specific ‘Nurse’s are assigned to ‘Patient’s currently residing in the ‘ICU’. This query seamlessly integrates the data of multiple tables by performing JOINs across the ‘Patient Table’, ‘Room Table’, ‘Care_Assignment Table’, and ‘Nurse Table’.
- ✓ The many-to-many ‘Cares’ relationship between ‘Nurse’ / ’Ward_Boy’ and ‘Patient’ is seamlessly handled using a associative table (E.g., ‘Care_Assignment’). The table bridges the combination of ‘CareProvider_ID’ and ‘Patient_ID’ as foreign keys to link caregivers with their patients.
- ✓ Furthermore, aggregation can be extended to generate reports. I can compute large totals, averages, and other statistics, such as the average duration of care assignments (E.g., AVG), the total payments received within a given period (E.g., SUM), or the number of patients diagnosed with a specific disease within a month (E.g., COUNT).
- ✓ Aggregation techniques also enable us to highlight patterns and anomalies. For example, by comparing the number of patients admitted to the number of rooms occupied, we can deduce room turnover rates and plan for future infrastructure investments.
- ✓ Similarly, by assessing the frequency and nature of call records, the hospital can optimize its support and helpline services.
- ✓ Lastly, subqueries and Common Table Expressions (CTEs) will be instrumental in constructing more complex queries. Whether it's deducing the list of patients who've made multiple calls for support or establishing a roster of doctors with their corresponding patients, these advanced SQL techniques will be essential tools
- ✓ In essence, the combination and aggregation of my stored data not only streamline administrative tasks but also fortify the hospital's commitment to delivering quality patient care

# Triggers (1/2)

[ 1 ] **Call Management:**

✓ If a call is from a non-registered number, flag it for potential new patient or emergency scenarios.

✓ Recognize returning patient calls and update the 'Date_Time_Of_Call' in the 'Call_Record' table, and append a new record if necessary.

✓ Set reminders for support staff to ensure follow-ups and patient satisfaction.

[ 2 ] **Staff Allocation:**

✓ Automatically assign a ‘Nurse’ or ‘Ward_Boy’ to new patients based on room assignments and staff availability.

✓ Ensure that when a Nurse or Ward Boy is reassigned, their previous tasks are concluded to prevent double-bookings.

✓ Implement a trigger to update the system and relevant personnel of any changes in doctors' availability.

[ 3 ] **Medical Record Maintenance:**

✓ A trigger to log doctor-patient interactions in the medical records, ensuring historical data is captured.

✓ Alert medical personnel for prescription modifications when a patient's diagnosis changes and set alerts for timely medication delivery.

✓ Schedule reminders for patients requiring periodic check-ups.

[ 4 ] **Care Assignment & Duration:**

✓ Calculate and log the duration of care upon conclusion of assignments.

✓ Trigger a review or continuity of care based on the ‘End_Date’ in the 'Care_Assignment' table.

[ 5 ] **Notification Triggers:**

✓ Set a trigger for sending automated reminders for 'Pending' payment statuses exceeding 30 days.

✓ Notify doctors of new patient assignments via email.

## Triggers (2/2)

[ 6 ] **Data Validation and Integrity:**

- ✓ Validate room occupancy by ensuring a trigger checks for an associated patient ID when 'Status' changes to 'Occupied'.
- ✓ If a patient is discharged, a trigger could automatically change the room status to 'Vacant'.

[ 7 ] **Data Maintenance:**

- ✓ When a new patient is diagnosed with a specific disease, a trigger could check if there are any specific guidelines or treatments that should be followed and inform the assigned doctor.
- ✓ When a nurse or ward boy is no longer assigned to a doctor, a trigger could automatically update their availability status.

[ 8 ] **Audit and Logging:**

- ✓ Maintain a log table through a trigger for changes in critical tables, documenting updates or deletions with details.

[ 9 ] **Safety Checks:**

- ✓ When a new medication is prescribed to a patient, a trigger could cross-check against their current medications and raise a flag if there's a potential harmful interaction.

[ 10 ] **Auto-Assignments:**

- ✓ For efficient staff utilization, if a ward boy or nurse becomes free (i.e., no current patient assignments), a trigger could look for patients without care assignments and automatically assign them.

[ 11 ] **Cascade Updates:**

- ✓ If a doctor updates their contact information (phone or email), a trigger could update all related records, ensuring patients and staff have the latest contact details.
- ✓ When a patient's 'Disease_Diagnosed' field is updated, a trigger could check if the prescription needs to be updated as well.

## Learning Objectives:

[ 1 ] To demonstrate the concepts covered in the course, I will show how normalization prevents redundancy. For example, by creating separate tables for ‘Doctor’, ‘Nurse’, and ‘Ward_Boy’ instead of repeating their information in patient records.

[ 2 ] One of the main features I plan to integrate will be the effective use of JOIN queries.  I'll highlight the process of retrieving interconnected data from diverse tables. For instance, I envision incorporating a section in the project that, at a glance, displays the total payments received over a specific timeframe. This will not only utilize JOIN operations but also aggregate functions to generate this report.

[ 3 ] I aim to design a database where specific actions, perhaps an update in a patient's status, automatically trigger related changes in other parts of the database.

[ 4 ] Furthermore, I aim to incorporate scenarios where the cascading effects of certain operations are evident. For example, if a patient's record is deleted, it might cascade changes such as the removal of their payment history or call records, underscoring the profound interdependencies within the database.

[ 5 ] My project will also delve into complex data fetch operations. Using the tools and techniques I've learned, I'll integrate sub-queries, showcase the use of SELECT statements within WHERE clauses, and effectively blend data from various tables using multiple JOIN operations. The intention is to ensure that the data is not just stored, but also efficiently retrievable and presentable.

[ 6 ] By embedding these elements into my project, I aim to create a comprehensive demonstration of the concepts covered in the course, ensuring that the theoretical knowledge transforms into practical application!