**ST207 Group Project**

Group 10

Candidate Numbers:

**Topic Description:**

This report examines the creation of a General Hospital Management System database using a NOSQL application, MongoDB. It was chosen because it offers a flexible and scalable approach to data storage and retrieval compared to traditional relational databases. We found it to be most suitable to handle diverse and evolving data structures. Our hospital setting explores various essential entities for our chosen environment e.g patients, doctors, appointments, procedures, diagnostic tests, insurance, prescriptions.

We aimed to model several complex relationships in each one of our queries, all considering a specific use case where specific data was required to answer necessary questions. Some relationships included: patient-doctor relations, appointment setting, revenue breakdown, diagnostic results, prescription dosages and health evolution. We aimed to model our data after diverse roles and departments in the medical field as well as myriad medicines and health conditions.

**Data Description: (Requires Brook’s input)**

The data within a hospital environment is typically sensitive and not available publicly. Therefore, to maintain data privacy and confidentiality, we have generated synthetic data that simulates real-world scenarios.

The dataset we have created encompasses various aspects of hospital management. Specifically, we have generated 100 personnel records, including 60 patients, 20 doctors, and 20 nurses. Additionally, we have included 100 appointment records, representing the interactions between patients and doctors. Furthermore, the dataset comprises 60 insurance records, 80 prescription records, 60 procedure records, 100 diagnostic test records, and 60 billing records. To ensure data consistency and integrity, the relationships between these data entities are established by linking them through their unique IDs.

The generated data is stored in CSV files and they are provided alongside this report. To process and manage this data, we have utilized PyMongo, a Python library for MongoDB, to read the CSV files and build relationships among the collections, so that the data is linked appropriately using object IDs. This process allows for efficient querying, analysis, and retrieval of hospital data.

Source of the original dataset:

<https://data.world/ushealthcarepricing/doylestown-hospital-standard-charges>

**Data Modelling:**

Below we will provide the collections and attributes that we used for our database. The relationships have been provided in our **ER diagram** which we attached in our Github file.

1. Personnel Collection:

- `personnelID`: Original ID used to establish relationships.

- `name`: User name.

- `dateOfBirth`: Date of birth.

- `address`: User address.

- `contactInfo`: Contact information.

- `timestamp`: Timestamp.

2. Patients:

- `personnelID`: Related personnel object ID.

- `patientID`: Patient's unique number.

- `insuranceID`: Related patient insurance object ID.

- `prescriptionID`: Related prescription object ID.

- `diagnosticTestID`: Related diagnostic test object ID.

- `appointmentID`: Related appointment object ID.

- `weight`: Patient weight in pounds.

- `height`: Patient height in inches.

- `allergies`: Allergies description.

- `familyHistory`: Disease history.

- `severity`: Severity level.

3. Doctors:

- `personnelID`: Related personnel object ID.

- `doctorID`: Doctor's unique identifier.

- `department`: Doctor's department.

- `medicalLicenseNumber`: Doctor's medical license number.

- `boardCertification`: Doctor's board certification.

- `insuranceAccepted`: Types of insurance accepted by the doctor.

- `prescriptions`: List of prescription object IDs associated with the doctor.

- `diagnosticTests`: List of diagnostic test object IDs associated with the doctor.

4. Nurses:

- `personnelID`: Related personnel object ID.

- `nurseID`: Nurse's unique identifier.

- `department`: Nurse's department.

- `nursingLicenseNumber`: Nurse's nursing license number.

- `certifications`: Nurse's certifications.

5. Insurance:

- `insuranceID`: Related insurance object ID.

- `patientID`: Patient's unique number.

- `insurancePlanName`: Name of the insurance plan.

- `policyholder`: Relationship of the patient to the policyholder.

- `policyNumber`: Policy number associated with the insurance.

- `groupNumber`: Group number associated with the insurance.

- `coverageStartDate`: Start date of coverage.

- `coverageEndDate`: End date of coverage.

6. Appointments:

- `appointmentID`: Appointment unique identifier.

- `patientID`: Patient's unique number.

- `doctorID`: Doctor's object ID.

- `type`: Type of appointment or procedure.

- `date`: Date and time of the appointment.

7. Prescriptions:

- `prescriptionID`: Prescription unique identifier.

- `patientID`: Patient's unique number.

- `prescribingDoctorID`: Doctor's object ID.

- `medication`: Name of the prescribed medication.

- `dosage`: Dosage information for the medication.

- `duration`: Duration for which the medication should be taken.

8. Procedures:

- `procedureID`: Procedure unique identifier.

- `patientID`: Patient's unique number.

- `doctorID`: Doctor's unique identifier.

- `nurseID`: Nurse's unique identifier.

- `type`: Type of procedure.

- `durationOfProcedure`: Duration of the procedure.

- `dateOfProcedure`: Date and time of the procedure.

9. Diagnostic Tests:

- `diagnosticTestID`: Diagnostic test unique identifier.

- `patientID`: Patient's unique number.

- `orderingDoctorID`: Ordering doctor's unique identifier.

- `testingDoctorID`: Testing doctor's unique identifier.

- `testType`: Type of diagnostic test.

- `testDate`: Date and time of the test.

- `testResult`: Result of the diagnostic test.

- `testLocation`: Location where the test was conducted.

10. Billing:

- `billingID`: Billing unique identifier.

- `patientID`: Patient's unique number.

- `procedureID`: Procedure unique identifier.

- `procedurePrice`: Price of the procedure.

- `coPayment`: Co-payment amount.

- `billingDate`: Date of billing.

- `serviceDate`: Date of service.

- `paymentReceivedDate`: Date when payment was received.

**Database Creation and Design Steps:**

The database are created by loading the CSV dataset and are processed for their relationships using pymongo library. The following describes some of the important parts of creating the database.

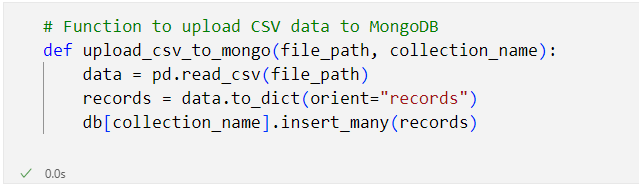
1. Connect to MongoDB cluster
2. We installed the pymongo and pandas library
3. We obtained the ip address of our colab notebook and pasted that into Mongo DB under Network access.
4. We created a Hospital database using the Mongo Client Object
5. We made a function to read the CSV data from the CSV’s of all our data which we uploaded to Github.
6. We made relationships between collections by updating documents with the corresponding object ID’s.

The report contains pictures of our code to demonstrate these steps.

Steps 1-4 (the 5th code block contains both the atlas connection key as well as password)



Step 5: Define read CSV function and import each CSV file





Step 6: Process relationships between collections

The following screen shot is one of the procedure. The other relationships are built in a similar way.



**Database Usage:**

Query 1 rationale: Hospitals like any business, need a detailed breakdown of the effect all their procedures and clients (patients) have on their inflow of cash. This query uses an aggregation pipeline with $groupby, $lookup and $project for a detailed and concise view of the total revenue for each patient and for their respective procedure. The output continues for all other patients but its been cropped for conciseness, the same is true for all the other queries shown.



Query 2 rationale: It’s important for a hospital to accurately understand the duration of its procedures for optimal scheduling and appointment setting. This query needed a $split function to separate the duration string into parts and a $arrayElemat to extract the numeric part of the string.



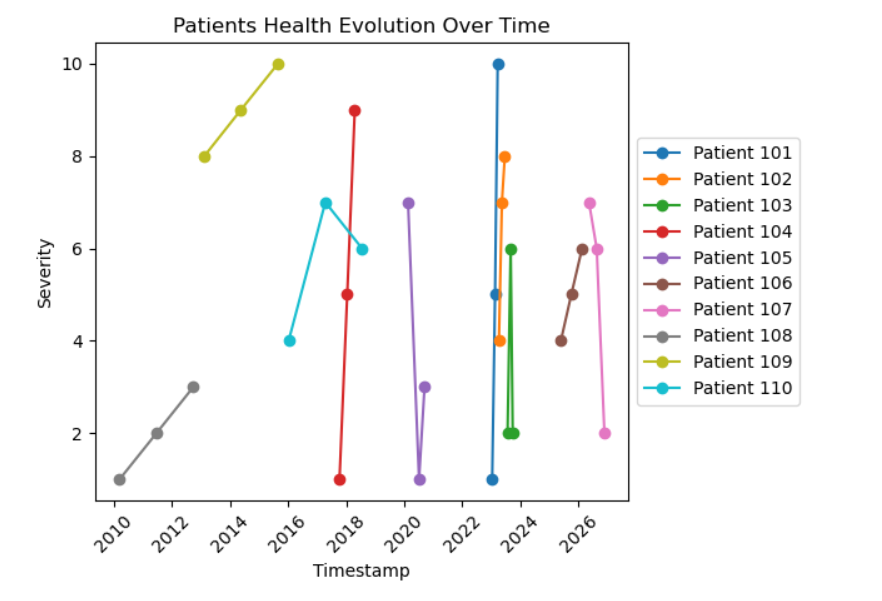
Query 3 rationale: This query focuses on creating a concise view for for hospitals to track patient-doctor interactions as well as relevant procedures. It uses $lookup to get information from relevant collections and then $project to show the fields of interest.

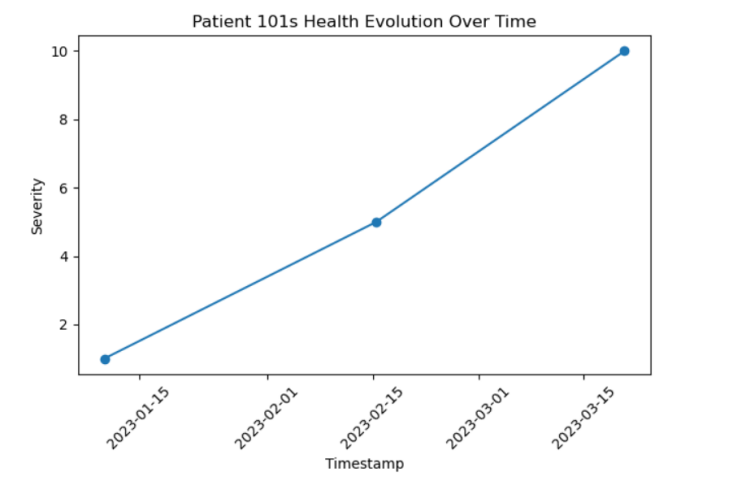


Query 4 rationale: According to our proposal feedback we realised the importance of temporal queries such as patients health evolution over time. Hospitals would employ these temporal tracking to appropriately respond to dynamic changes in a patients condition over time. This query needed $match to get specific patient ID’s, $sort to sort by timestamp in ascending order and $unwind for further processing. It also used matplotlib to plot the changes in patients condition across several years. A similar but unincluded query was made to track patient weights across time.

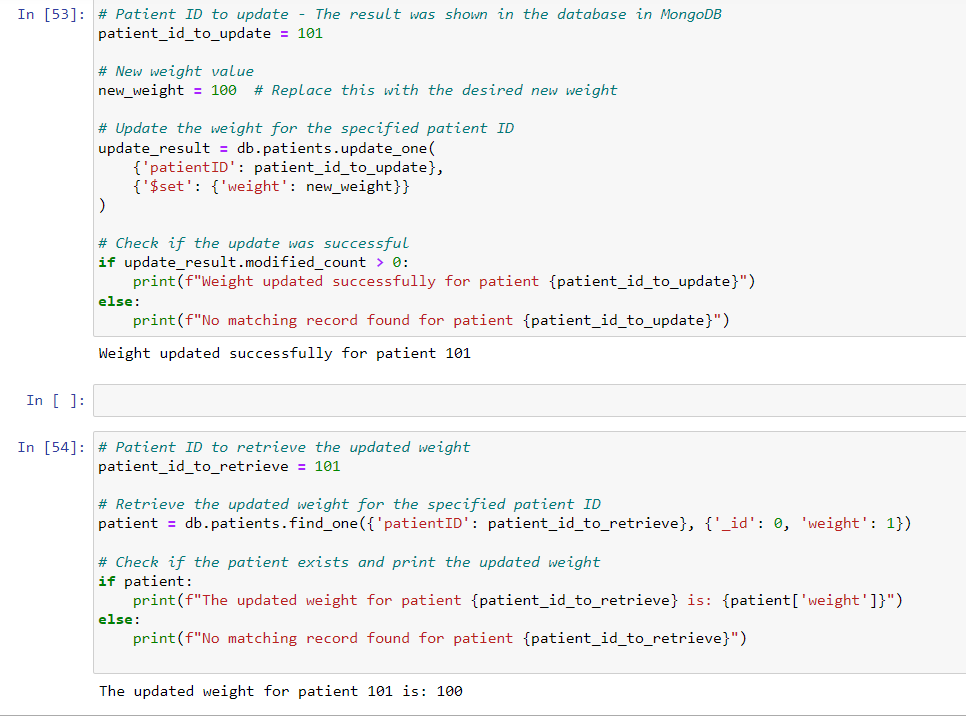






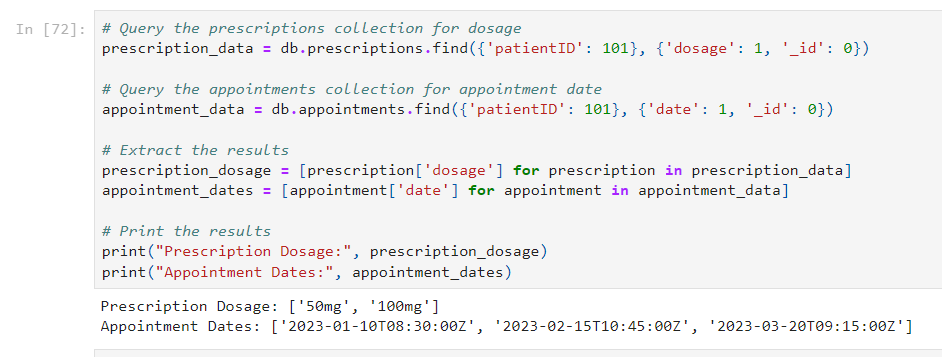


Query 5 rationale: Finally it’s crucial to be able to update or change any information about patientss or anything related to the hospital. So this final query uses update\_one to change a single document, $set to change a single field and importantly modified\_count to make sure the update was actually carried through.

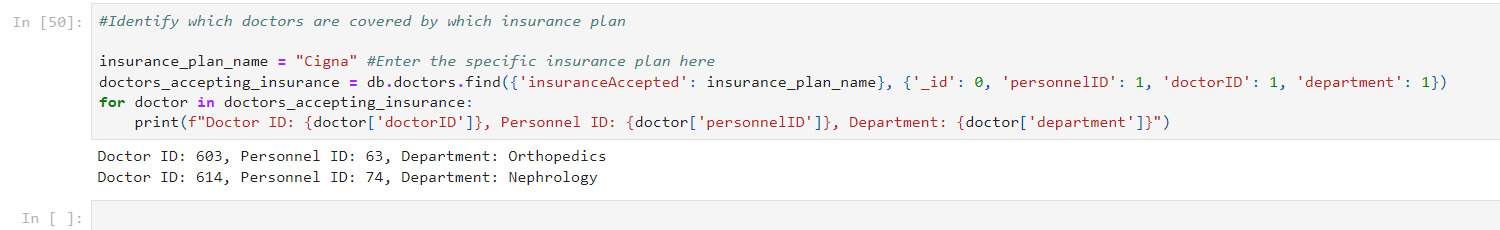
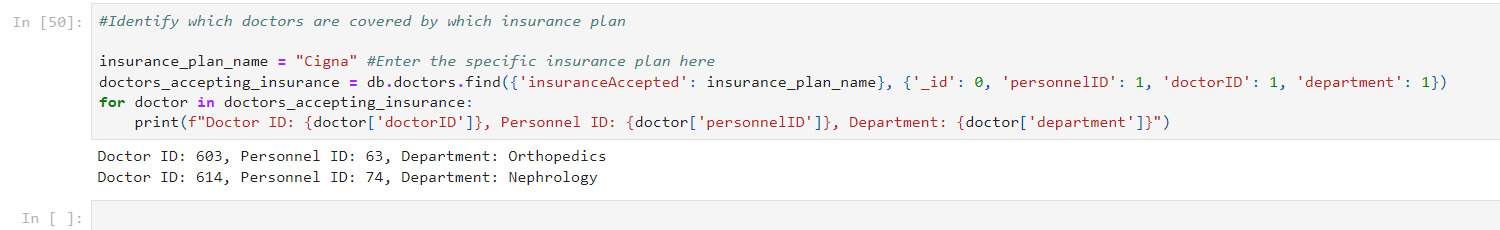


**We performed smaller queries that were not included in the report but which made use of more aspects of our data, such as the following and more:**

Query 6 rationale: Similar to the patient-doctor interactions query, its also important to have a view of all the medical information about a patient. We tried to create one where we can showcase a patients prescription dosages and appointment dates. It uses ‘find’ to get information from the relevant collections.



Query 7 rationale: It’s necessary for patients to know which doctors, departments or procedures are covered by their insurance plan. The following query requires us to input a specific insurance plan, in this example ‘Cigna’ and then it finds the relevant doctors and departments. From there we can dig further to find the procedures that are also covered by then plan.



In conclusion our report showcases the use of MongoDB, python and relevant libraries to create and manage a database designed to model a Hospital Management use case. MongoDB was used because we believed its document-oriented structure would be aligned with handling medical records as well as a data structures that grows horizontally. Using it brought technical complexity due to its scema-less design, which required us to model data carefully once relationships between collections were created. We incorporated data visualisation elements for the temporal queries to better communicate their results with matplotlib and the pandas library.

We believe we were successful in showcasing a variety of operations to model complex data in several realistic hospital scenarios while also using the wide variety of data that we both generated and acquired from mentioned sources.