<u>USE CASE STUDY REPORT</u>

Group No.: Group 16

Student Names: Vipul V Suresh and Sri Sai Subhash Kasireddy

Executive Summary: The following use case study describes the implementation of SQL and NoSQL databases along with connections to front-end applications such as Python to demonstrate the working principles of a typical Database Management system to develop an application that enables the last-mile delivery operations in a supply chain system. The primary goal of this study is to assess the fit, requirements, and constraints that goes into integrating new state of the art technologies and their dependent operational database infrastructure into already existing Database Management Systems or to identify the Database specific design requirements to set up a new last-mile delivery business using the MedsDeliver app. The requirements for this use case involve creating a conceptual data model to understand the relationships between all the entities present in the model. An EER and UML class diagram shows the interrelations between entities such as Drones, Address, Pharmacy, Doctors, Customers, etc., along with its underlying cardinalities and constraints. Following the conceptual modeling, the entities and relations are mapped to a relational database schema which models all the relations with its keys and links to other relations using foreign keys. This schema provides a strong interconnection between multiple relations. MySQL is used in this use case to show the RDBMS implementations. Owing to its strong dependence on enforcing the database schema, SQL limits the database from horizontal scaling when needed. To accommodate this, NoSQL (Non-relational database) database frameworks are being used in major organizations in the current century due to the versatility and the freedom it offers to scale the database horizontally and vertically with containerization methods. MongoDB is one such application/organization which offers NoSQL database infrastructure to host our database. In this use case study, we demonstrate the use of MongoDB to host our database. Although not demonstrated in this case study, due to its containerized clusters of the database, MongoDB aids the scalability both vertically and horizontally with efficient execution of queries. While this describes the back-end infrastructure tied to an application, connections to the front end can be established on various GUI and non-GUI based applications such as Mongo Shell, Python, R, etc., using pre-written libraries such as "MySQL.connector" to connect MySQL to Python and "pymongo" to connect MongoDB server to Python. This offers added functionalities to visualize the data available on the server and further can be used for development purposes in efforts to integrate CI/CD or an ML pipeline. Overall, this case study shows the procedural approaches towards enabling the Database Management System.

I. Introduction

Based on the latest technologies and innovation in recent years around the Last-mile delivery/Supply Chain market, organizations are scrambling their resources to achieve the vision statement of "Convenience is the Future" where things needed by customers are delivered to the customer without the customer spending time and effort to buy the items they want. Enabling this vision needs efficient operations of all infrastructure that is built for a product to function and one of the key elements of this infrastructure is the Database Management system used which hosts and stores all the data surrounding the products' ecosystem.

In this case study, we assess the requirements, capabilities, and constraints of SQL and NoSQL Database systems applied to an application that is used by customers to order medicines and groceries from nearby stores which are in turn delivered by Drones and other autonomous robotic systems.

The current Last-Mile delivery ecosystem employs human drivers to help deliver goods and services to customers upon request which means the current database is versatile enough to include new technology-enabled robotic systems to replace human drivers. And this allows us for a new delivery method to be seamlessly integrated into the current database infrastructure.

- Problem statement: Integrating drone-based medicine delivery into existing delivery supply chain systems enabling efficient utilization of Database Management infrastructure for Last-Mile delivery of medicines using the MedsDeliver App
- The goal of your study: To assess the fit, requirements, and constraints that goes into integrating new state of the art technologies and their dependent operational database infrastructure into already existing Database Management Systems or to identify the Database specific design requirements to set up a new last-mile delivery business using the MedsDeliver app.
- The requirements:
    - To facilitate this integration, a database needs to be created to establish connections within the new network and eventually amalgamate the system into the parent organization's supply chain.
    - Customers can register themselves as patients on the App for prescription medicines or order Over-the-counter medicines from a Pharmacy. The delivery is automated using Machine Learning to find the closest pharmacy to be delivered from. Delivery of prescription medicines can also be scheduled from the store using pre-approved order schedules.
    - To facilitate this, docks meant for drones are installed in strategic locations at specific zip codes with the drone's auto-docking mechanism to secure and recharge the drones
    - The drones fly from the dock to the pharmacy, agents in the pharmacy load the shipment into the drone's lockbox after which the drone flies to the customer's address to deliver the order. Specific passcodes are

provided on the app to the pharmacy agents and the customers to unlock the drone's lockbox to place/retrieve the items.

- The current database design is meant for delivering pharmaceutical orders but, once this is proven to be effective, this can be further expanded to orders of a certain weight and dimensional limits
- On a broader perspective outside the Database Design and Modelling, other Business aspects need to be considered such as the limitations and regulations behind using drones to deliver medicines, fly zones adhering to the Air traffic regulations, Sensitive item checks, Operational security that needs to be translated to business requirements, etc., These can be further assessed, and any such information can be added on to the database design in future iterations.

II. Conceptual Data Modeling

The below-mentioned illustrations describe the data modeling via EER and UML, including domains, cardinalities, constraints, and abstractions.
Figure 2.1a attached in Appendix(VIII) illustrates the EER model for the Database
Figure 2.1b attached here illustrates the UML class diagram (cardinalities are mentioned in the illustration attached in Appendix(VIII) under Figure 2.1.c

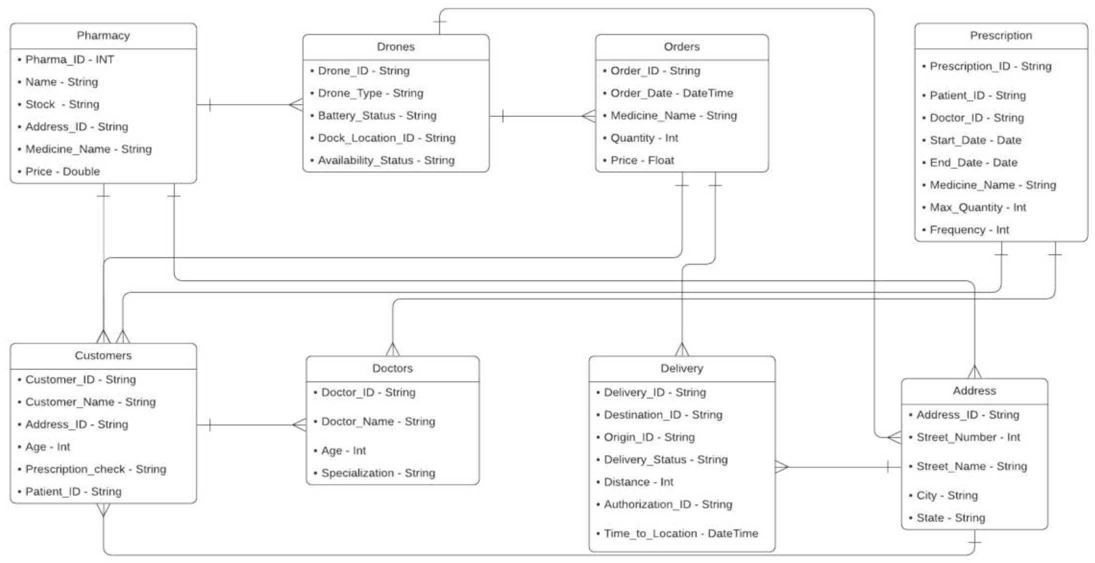


Figure 2.1.b

III. Mapping Conceptual Model to Relational Model

The following schema illustrates the mapping of EER and UML to a relation model, including tables, primary keys, foreign keys, relational constraints, and normalization

Address (Address_ID, Street_Number, Street_Name, City, State)
Customer (Customer_ID, Customer_Name, *Address_ID*, Age, Prescription_Check, Patient_ID)
FOREIGN KEY Address_ID refers to Address_ID in Address; NULL NOT ALLOWED
Delivery (Delivery_ID, *Destination_ID, Origin_ID*, Delivery_Status, Distance, Authorization_ID, Time_to_Location)
FOREIGN KEY Destination_ID, Origin_ID refers to Address_ID in Address; NULL NOT ALLOWED
Doctor (Doctor_ID, Doctor_Name, Age, Specialization)
Drones (Drone_ID, Drone_Type, Battery_status, *Dock_Location_ID*, Availability_Status)
FOREIGN KEY Dock_Location_ID refers to Dock_Location_ID in Address; NULL NOT ALLOWED
Orders (Order_ID, Order_Date, Medicine_Name, Quantity, Price)
Pharmacy (Pharma_ID, Pharma_Name, *Address_ID*, Medicine_Name, In_Stock, Price)
FOREIGN KEY Address_ID refers to Address_ID in Address; NULL NOT ALLOWED
Prescription (Prescription_ID, *Patient_ID, Doctor_ID*, Start_Date, End_Date, Medicine_Name, Max_Quantity, Frequency)
FOREIGN KEY Patient_ID refers to Customer_ID in Customer; NULL NOT ALLOWED
FOREIGN KEY Doctor_ID refers to Doctor_ID in Doctor; NULL NOT ALLOWED

IV. Implementation of Relation Model via MySQL and NoSQL

Implementation of the relational model in MySQL:

The database was created in MySQL under the “ProjectDatabase” name with 8 corresponding relational tables as mapped in the previous section (section III)
The following queries were executed to demonstrate the implementation of MySQL

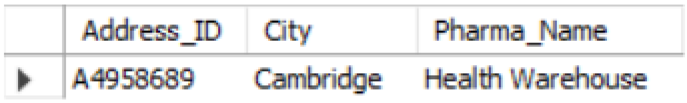
Query 1: Select Address of customers who live in Roxbury

```
SELECT customer.Address_ID
FROM customer inner join address on the address.Address_ID=customer.Address_ID
WHERE address.City='Roxbury';
```

Output return Null since address_ID in City=” Roxbury” corresponds to a Pharmacy

Query 2: Select Address_Id of pharmacies located in the city “Cambridge"

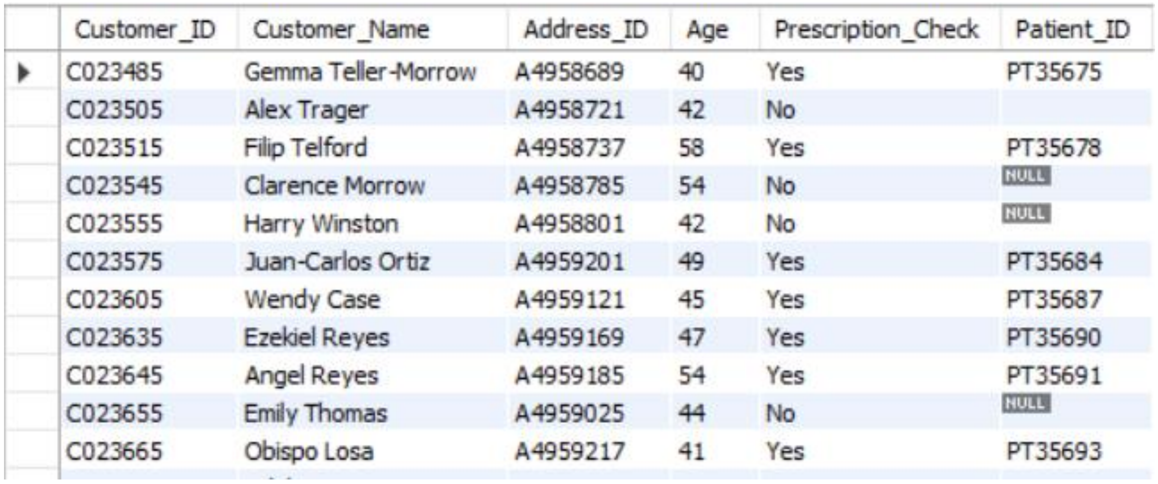
SELECT address.Address_ID, address.City, pharmacy.Pharma_Name
FROM pharmacy inner join address on the address.Address_ID=pharmacy.Address_ID
WHERE address.City='Cambridge';



Address_ID	City	Pharma_Name
A4958689	Cambridge	Health Warehouse

Query 4: Find all Customer_IDs above age 35

SELECT
FROM customer
WHERE Age>35;
 *



Customer_ID	Customer_Name	Address_ID	Age	Prescription_Check	Patient_ID
C023485	Gemma Teller-Morrow	A4958689	40	Yes	PT35675
C023505	Alex Trager	A4958721	42	No	
C023515	Filip Telford	A4958737	58	Yes	PT35678
C023545	Clarence Morrow	A4958785	54	No	NULL
C023555	Harry Winston	A4958801	42	No	NULL
C023575	Juan-Carlos Ortiz	A4959201	49	Yes	PT35684
C023605	Wendy Case	A4959121	45	Yes	PT35687
C023635	Ezekiel Reyes	A4959169	47	Yes	PT35690
C023645	Angel Reyes	A4959185	54	Yes	PT35691
C023655	Emily Thomas	A4959025	44	No	NULL
C023665	Obispo Losa	A4959217	41	Yes	PT35693

Implementation of the relational model in NoSQL:
The following examples and steps show the versatility of MongoDB to host and run NoSQL databases. To draw a comparison between the usage of SQL and NoSQL, we try to run the same queries on our MedsDeliver Database.
The database “ProjectDatabase” contains 8 collections which are the same as SQL tables

```
use('ProjectDatabase');

db.getCollectionNames()
  ["Address","Prescription","Doctors","Delivery",”Customers","Orders",
  "Pharmacy", "Drones"]
```

Query 1: Select Address of customers who live in Roxbury

```
db.Address.find({City : "Roxbury"}, { _id: 0}).pretty()
```

Output:

```
[
  {
    "Address_ID": "A4958513",
    "Street_Number": 772,
    "Street_Name": "Third",
    "City": "Roxbury",
    "State": "Massachusetts"
  }
]
```

Query 2: Select Address_Id of pharmacies located in the city “Cambridge"

```
db.Address.aggregate([{$lookup:{
        from : "Pharmacy",
        localField: "Address_ID",
        foreignField: "Address_ID",
        as: "Pharma_addy"
    }},
    {$match : {City: "Cambridge"}},
    {$unwind: "$Pharma_addy"},
    {$project: {"City": 1, "Address_ID":1, "Pharma_addy.Pharma_ID" : 1}},
]).pretty()
```

Output:

```
[
  {
    "_id": {
      "$oid": "61b3dd532a47e58f27c28d17"
    },
    "Address_ID": "A4958689",
    "City": "Cambridge",
    "Pharma_addy": {
      "Pharma_ID": "PH23851"
    }}]
```

Query 3: Find all Customer_IDs above age 45

```
db.Customers.find({Age : {$gt : 45}}, {Customer_ID: 1, Age: 1,
Customer_Name: 1, _id:0}).pretty()
```

Output:

```
[
  {
    "Customer_ID": "C023515",
    "Customer_Name": "Filip Telford",
    "Age": 58
  },
  {
    "Customer_ID": "C023545",
    "Customer_Name": "Clarence Morrow",
    "Age": 54
  },
  {
    "Customer_ID": "C023575",
    "Customer_Name": "Juan-Carlos Ortiz",
    "Age": 49
  },
  {
    "Customer_ID": "C023635",
    "Customer_Name": "Ezekiel Reyes",
    "Age": 47
  },
  {
    "Customer_ID": "C023645",
    "Customer_Name": "Angel Reyes",
    "Age": 54
  },
  {
    "Customer_ID": "C023675",
    "Customer_Name": "Adelita",
    "Age": 47
  },
  {
    "Customer_ID": "C023695",
    "Customer_Name": "Che Romero",
    "Age": 48
  },
  {
    "Customer_ID": "C023705",
    "Customer_Name": "Michael Ariza",
    "Age": 54
  }
]
```

V. Database Access via R or Python

The following section demonstrates the connection between Python and MySQL and Python and MongoDB which provides a robust analytical tool to find insights from the data available.

The connection to MySQL is established using the MySQL.connector library as shown

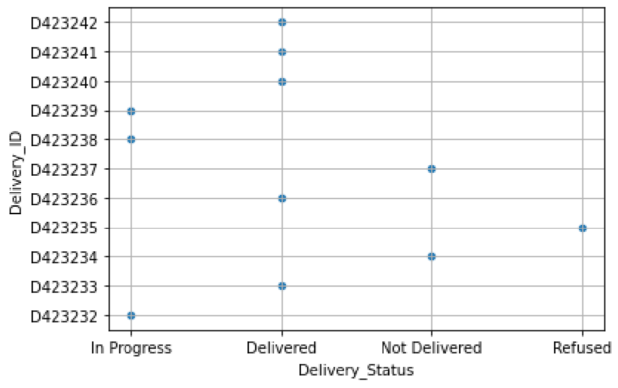
import MySQL.connector
Client = mysql.connector.connect(user=’, password='*password*', host='127.0.0.1', database=' ProjectDatabase')

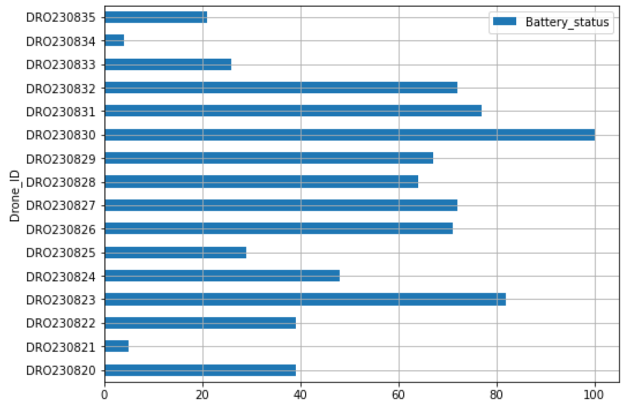
This establishes a link to run analytical tasks on python with libraries like pandas matplotlib enabling the conversion of tabular data onto DataFrames and visualization of data

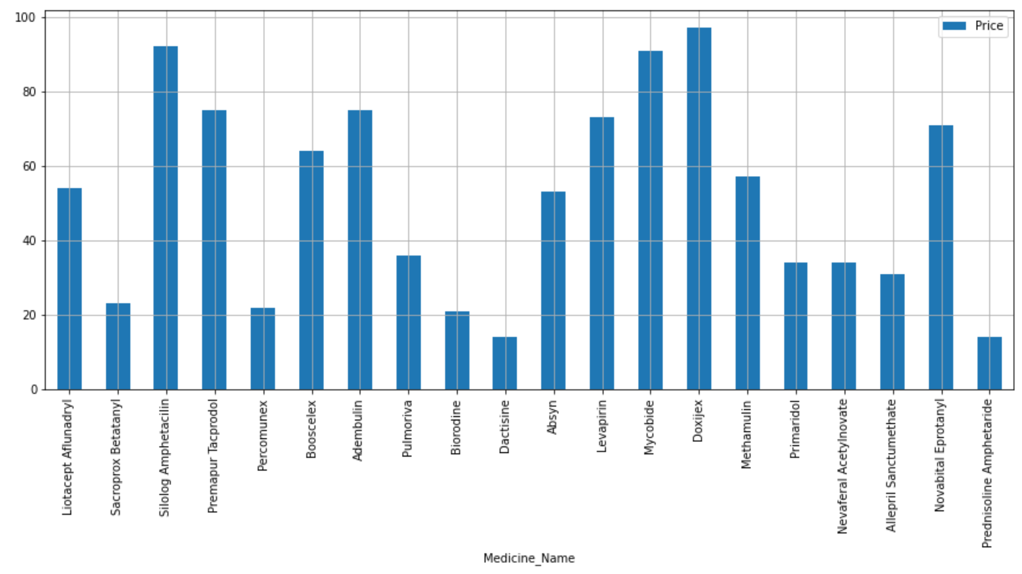
A similar connection was established to MongoDB on python using a connection string available on the MongoDB Atlas platform which provides connections to various front-end applications such as Mongo Shell, Python, R, Scala, Swift, and MongoDB Compass. This is enabled by the libraries “pymongo” and “dnspython”.

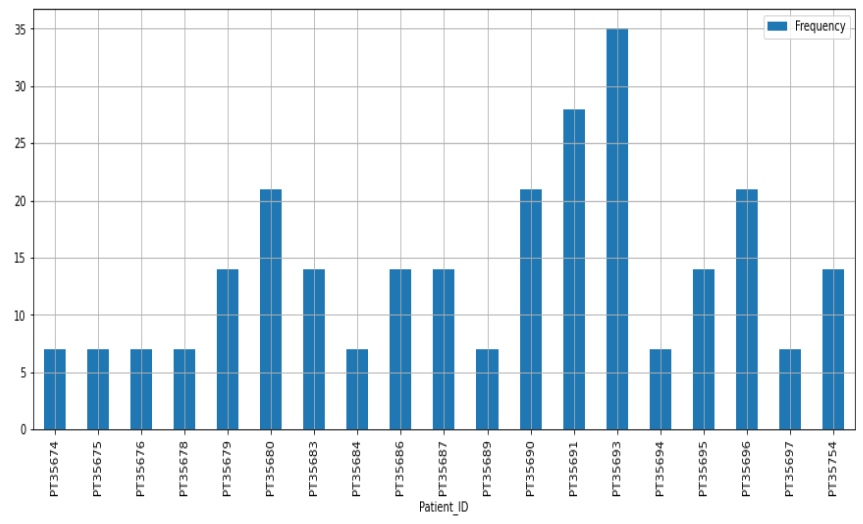
```
import pymongo
import pandas as pd

client =
pymongo.MongoClient("mongodb+srv://DMAproject:DMAproject@cluster0.24cz0.mo
ngodb.net/ProjectDatabase?retryWrites=true&w=majority")

db = client.get_database('ProjectDatabase')
db.list_collection_names()
```

The following visualizations provide some insights to the database on python









VII. Summary and recommendation

The above-described procedure to create and manipulate a database on SQL and NoSQL can be used to develop the MedsDeliver's Database Management infrastructure for real-world implementation. Applications can be connected seamlessly to either of these database management frameworks to deploy scalable operations across the product infrastructure. As mentioned, SQL offers a string dependence between its relations making it more efficient to store operational data which does not require further enhancing of the database structure and scaling of the schema horizontally. Certain drawbacks of NoSQL were also noticed and is described during the demonstration of this database schema which involves duplication of records, Complex MapReduce pipelines, and inconsistent join operations in cases where two or three tables need to be referenced for example referencing Address_ID from Address collection to add address_ID information coherent with the Customer_ID in Orders collection.

To further enhance and develop the database management infrastructure, careful evaluation of business needs, long-term and short-term strategies of the organization need to be considered into account. To enable the autonomous delivery feature where-in the drones fly from and to different address_IDs can be developed over time using advanced analytics and AI. Intensive testing of the current MedsDeliver Database can lead to a powerful infrastructure that can be utilized by existing last-mile delivery and operations companies to integrate into their database systems or can be developed from the ground up as a stand-alone DBMS.

VIII. Appendix

Figure 2.1a- EER model for the MedsDeliver app Database

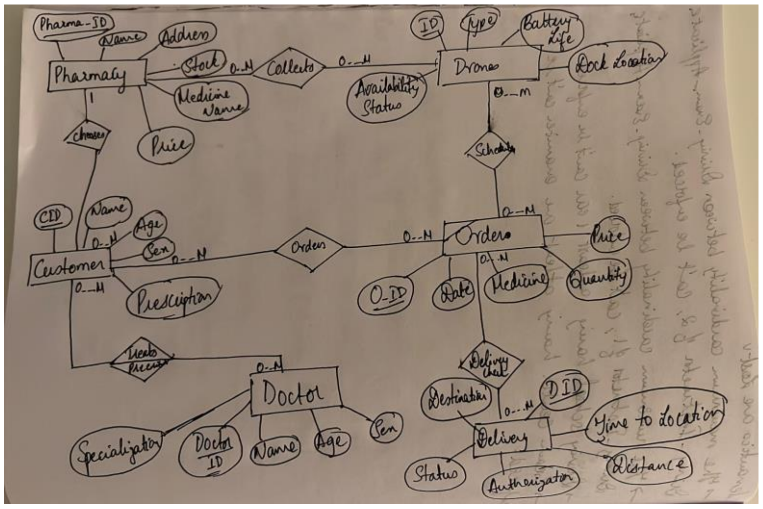
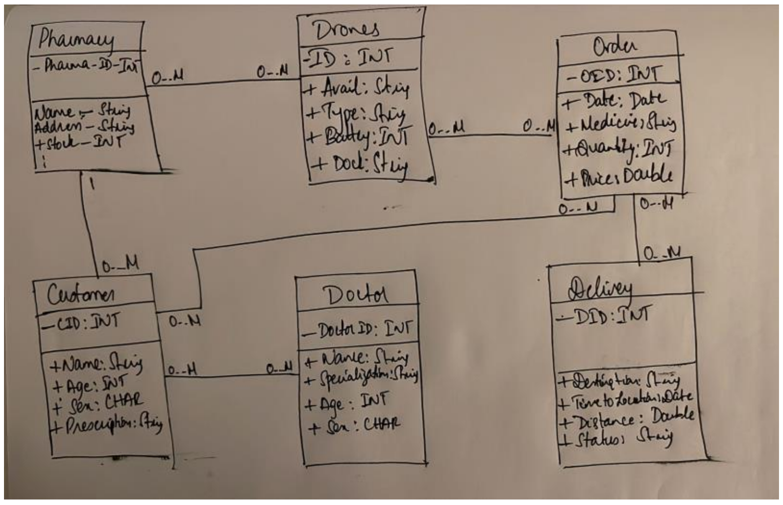


Figure 2.1c- UML class diagram for the MedsDeliver app Database with associated domains and cardinalities


Pharmacy
- Pharma-ID-INT
Name - String
Address - String
+Stock - INT
0..M
0..M
Drones
-ID : INT
+ Avail: String
+ Type: String
+ Battery: INT
+ Dock: String
0..M
0..M
Order
- OID: INT
+ Date: Date
+ Medicine: String
+ Quantity: INT
+ Price: Double
1
0..N
0..M
0..M
0..M
Customer
- CID: INT
+ Name: String
+ Age: INT
+ Sex: CHAR
+ Prescription: String
0..M
0..M
0..M
Doctor
- Doctor ID: INT
+ Name: String
+ Specialization: String
+ Age : INT
+ Sex : CHAR
Delivery
- DID: INT
+ Destination: String
+ Time to Location: Date
+ Distance : Double
+ Status: String