**Lewis University  
CPSC 50900: Database Systems  
Term Project**

# **Pharmacy Management System**

**Pharmacy Management System Design** is basically aimed to provide complete information of the doctors and medicine availability to the patient, according to which the patient can easily book appointment with the doctor for the treatment.

A Doctor can treat based on the illness of patient and can prescribe the medicines to the patient.

A Medical Staff Employee will go through the prescription details and fulfill the request by placing the order for medicines.

Medical Stock contains the information about the available medicines, their expiration and stock quantity details.

**Anil Potru, anilpotru@lewisu.edu**

***https://github.com/anilpotru/Pharmacy\_Management\_DB\_design***

Table of Contents

[Initial Proposal 2](#_Toc85814569)

[Data Sources 2](#_Toc85814570)

[Data Storage Alternatives 2](#_Toc85814571)

[Relational Database Design Process 3](#_Toc85814572)

[Relational Database Design 3](#_Toc85814573)

[Data Definition Language (DDL) Scripts 3](#_Toc85814574)

[Data Manipulation Language Scripts 4](#_Toc85814575)

[Indexes 5](#_Toc85814576)

[Views 5](#_Toc85814577)

[Triggers 5](#_Toc85814578)

[Transactions 5](#_Toc85814579)

[Database Security 6](#_Toc85814580)

[Locking and Concurrent Access 6](#_Toc85814581)

[Backing Up Your Database 6](#_Toc85814582)

[Python Programming 7](#_Toc85814583)

[PHP Programming 7](#_Toc85814584)

[Suggested Future Work 8](#_Toc85814585)

[Activity Log 8](#_Toc85814586)

# Initial Proposal

PMS (Pharmacy Management System) was created to address the challenges of managing all of the paper work for each patient associated with the various departments of hospitalization while maintaining confidentiality. PMS allows staff to manage all patient paperwork in one location, reducing the amount of time they spend organizing and analyzing paperwork. PMS is involved in a variety of projects, including:

Maintain the patient's medical records.

Keep track of the patient's contact information.

Keep a calendar of your appointments.

Keep the insurance information on hand for future need.

Bill payments are being tracked.

Additionally, an admin can update the **webpage** changing the medicines, stock availability, doctors’ information according to the database. And also, can use the **webpage** to check the statistics information from the system.

# Data Sources

**Pharmacy Management System Design** is entities and their attributes:

**Patient:** id(ssn) , firstname, last name ,gender, age, address, email, phonenumber

**Doctor:** id(ssn), firstname, last name, gender, doctortype, active

**DoctorType :** doctortype, description

**MedicineStock** : id, medicine name, batch number, medicine manufacturer, medicine stock quantity, price, expiring date

**DoctorPrescription** : prescription id, patientid, doctorid, precription date

**precription\_details :** id, precriptionid, medicineid, quantity

**Ordersummary** : id, precriptionid, patientid, date, staffid

**MedicalStaffEmployee**: id, firstname, lastname, location, joining date, role

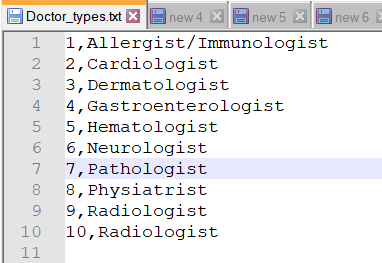
**BillSummary** : billid, orderid, total amount, bill date, patientid, paymenttype, totalpaid

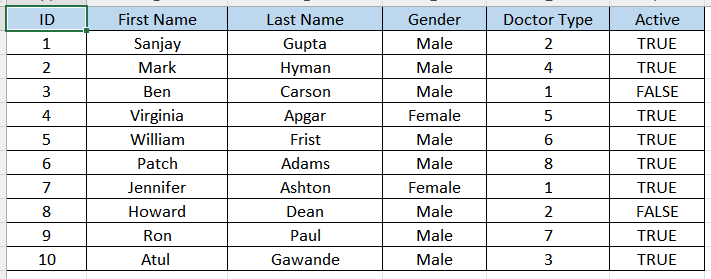
Please find the GitHub link for the sample data sources attachments required for the above entities.

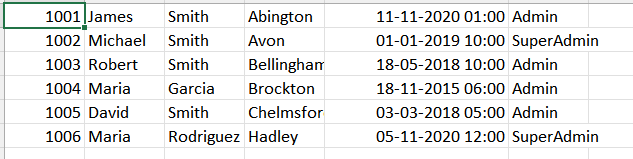
Below is the sample data of the medicine stock entity json

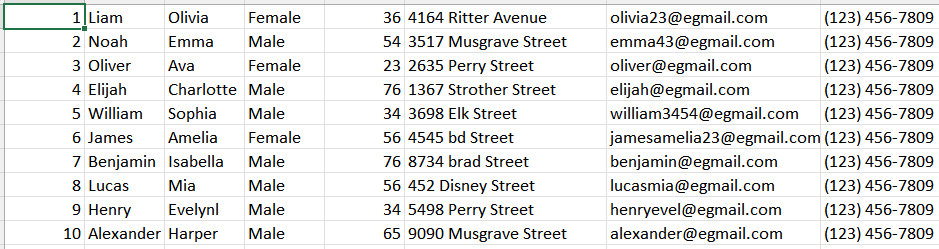
# 

Below are the doctortypes , doctor\_list , medicalStaff and patient information respectively









# Data Sources Alternatives

We have seen that RDBMS is better approach for the above database design. Where it guarantees the granularity and relationship between the tables.

However, as the system grows if we have below requirements the existing RDBMS may not be suitable for future requirements.

1. We have tons of data like medicine, insurance details, notifying the patient

2. We need to have faster search

3. We need to run queries for analytics

So, alternatively we can use NoSQL as the option to acknowledge the above issues. MongoDB, Cassandra, and CouchDB are the examples of NoSQL.

# Relational Database Design Process

**Patient :** id(ssn) , firstname, last name ,gender, age, address, email, phonenumber

**Doctor:** id(ssn), firstname, last name, gender, doctortype, active

**DoctorType :** doctortype, description

**MedicineStock** : id, medicine name, batch number, medicine manufacturer, medicine stock quantity, price, expiring date

**DoctorPrescription** : prescription id, patientid, doctorid, precription date

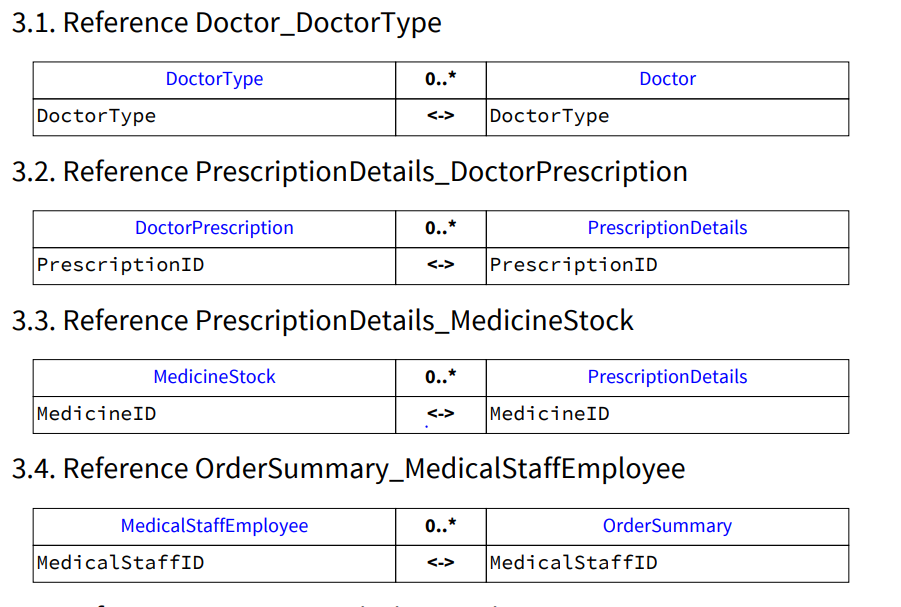
**precription\_details :** id, precriptionid, medicineid, quantity

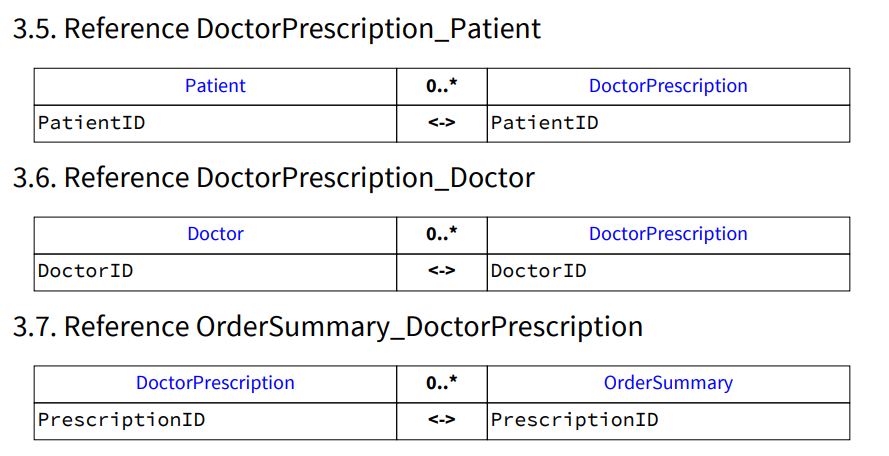
**Ordersummary** : id, precriptionid, patientid, date, staffid

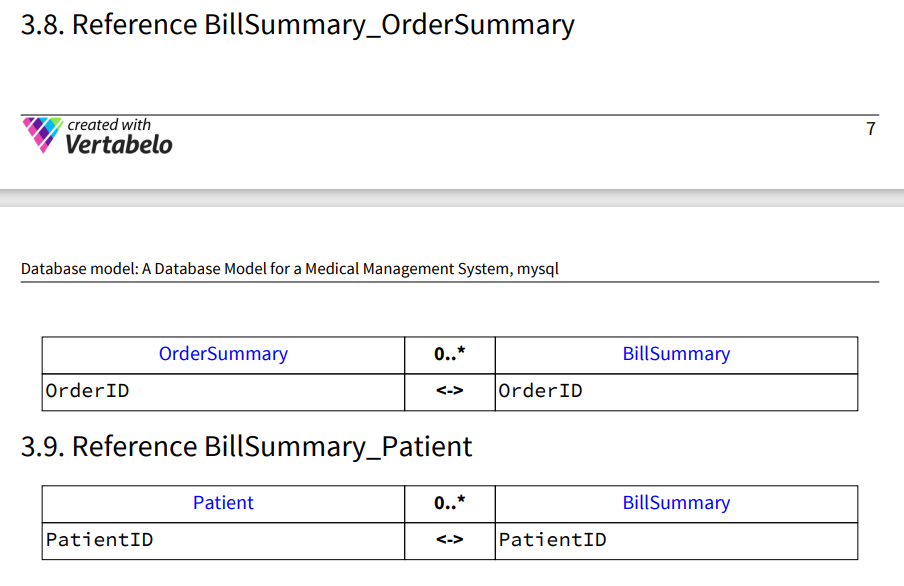
**MedicalStaffEmployee**: id, firstname, lastname, location, joining date, role

**BillSummary** : billid, orderid, total amount, bill date, patientid, paymenttype, totalpaid

**Below is the relationship between the tables**



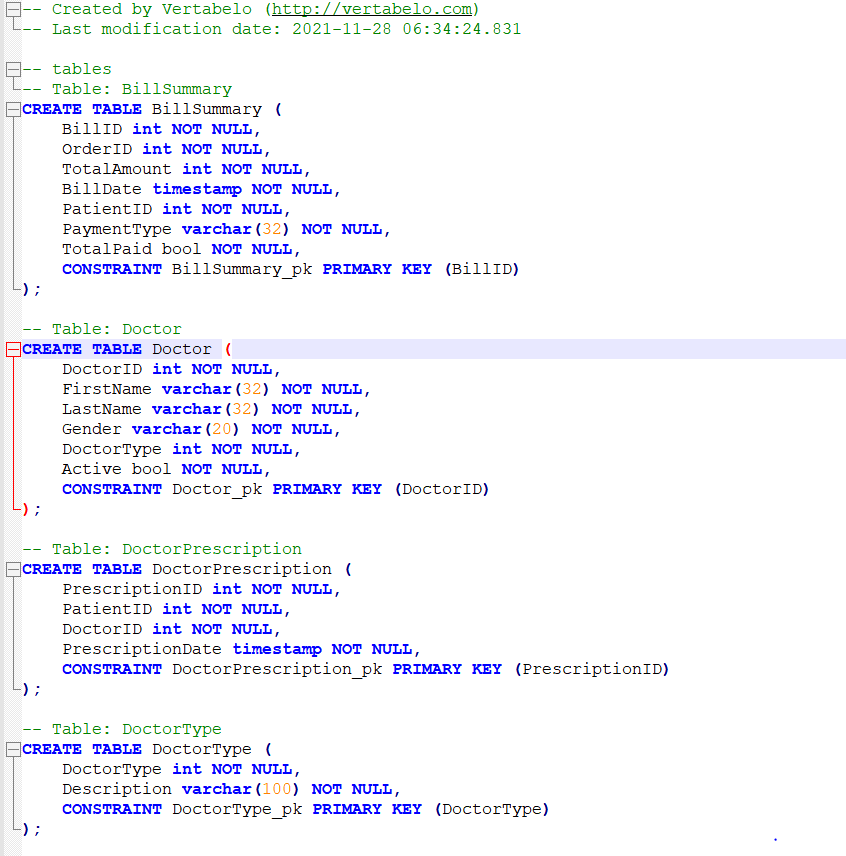


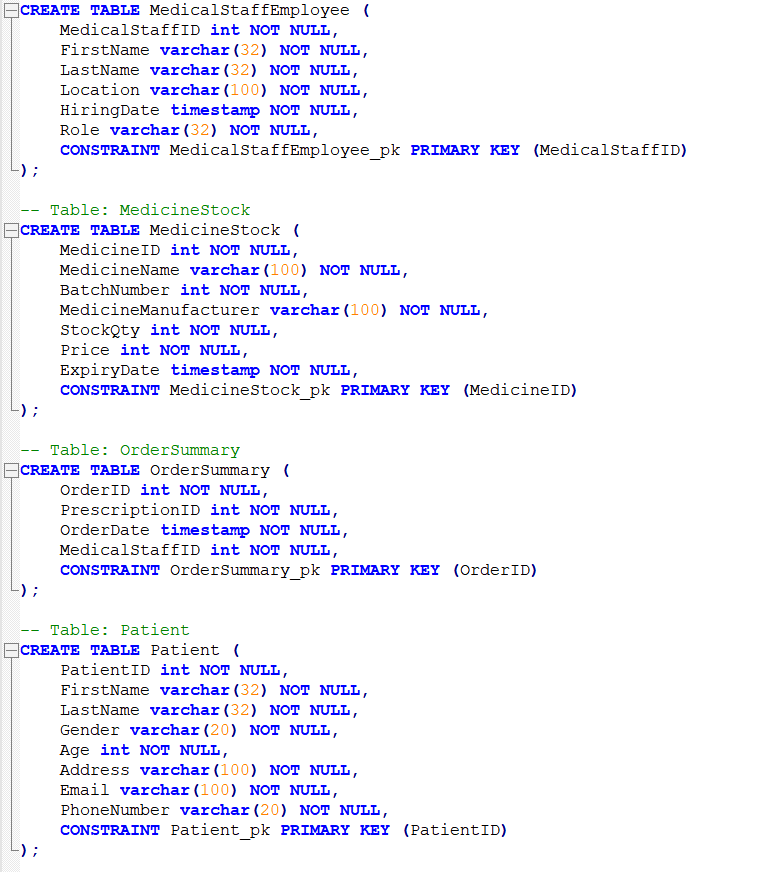


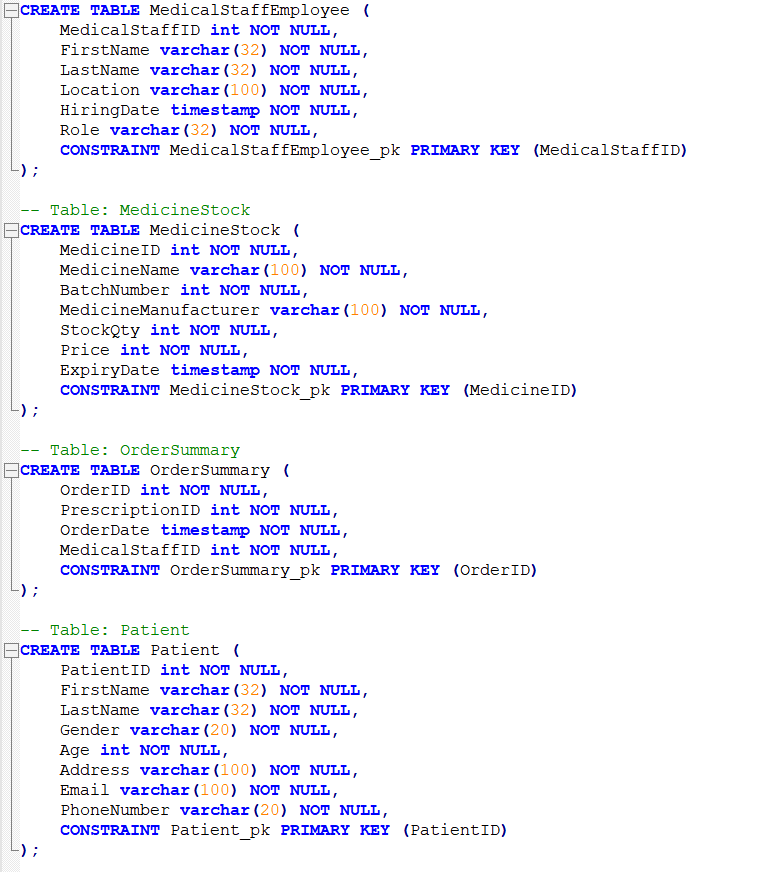
Please find the uploaded DDL script for the relations creation that is created by **Vertabelo**.

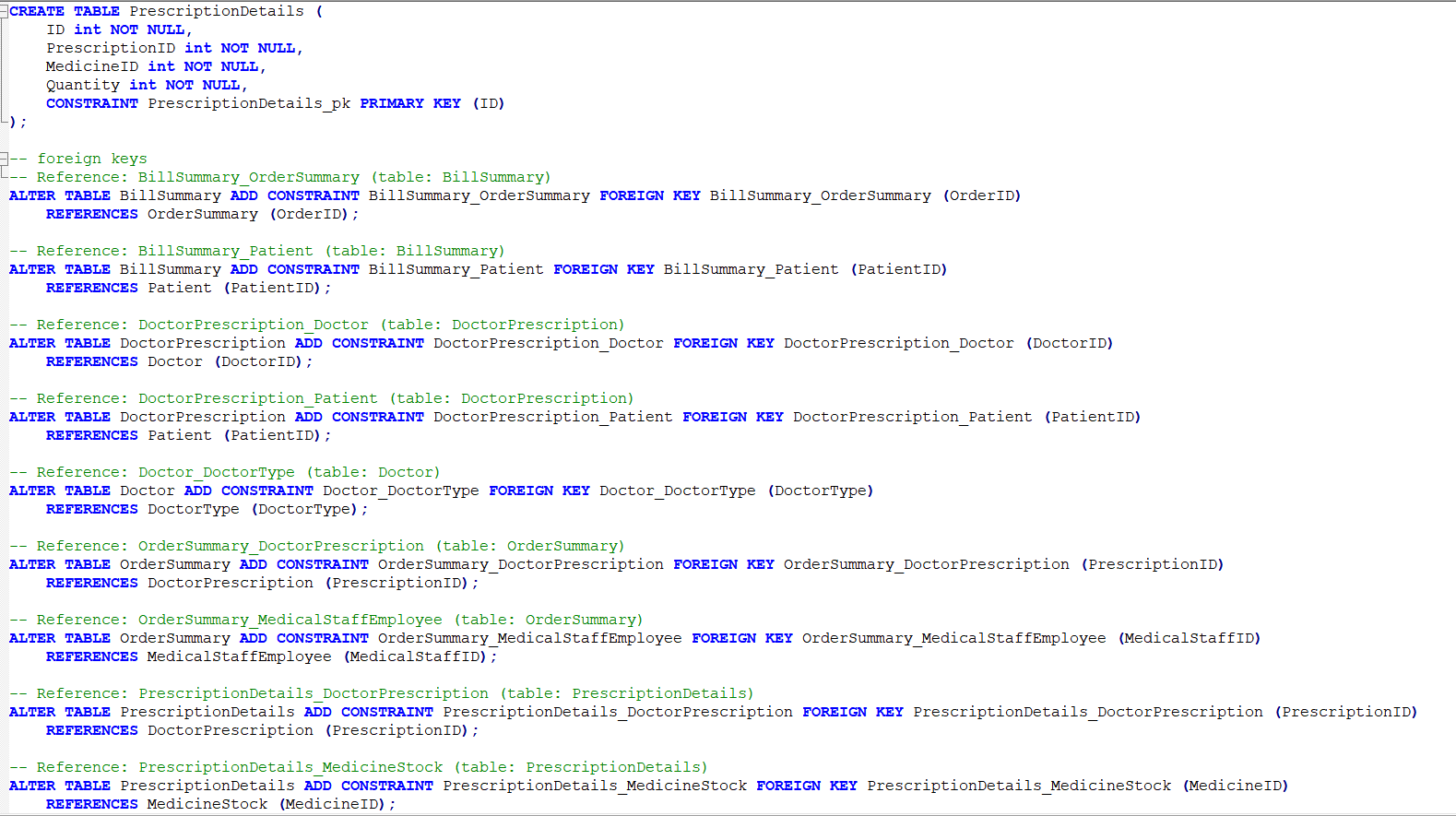








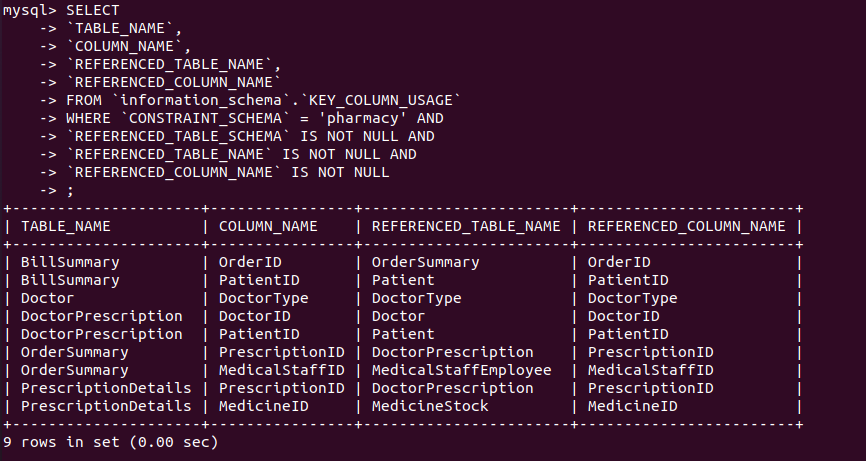




Please find below/or in github the generated relationships document from the Vertabelo



Below is the script to get the relationships between the tables in cinema database;



# Relational Database Design

# Tables

## 1.1. Table Patient

1.1.1. Columns

|  |  |  |  |
| --- | --- | --- | --- |
| **Column name** | **Type** | **Properties** | **Description** |
| PatientID | int | PK |  |
| FirstName | varchar(32) |  |  |
| LastName | varchar(32) |  |  |
| Gender | varchar(20) |  |  |
| Age | int |  |  |
| Address | varchar(100) |  |  |
| Email | varchar(100) |  |  |
| PhoneNumber | varchar(20) |  |  |

## 1.2. Table Doctor

1.2.1. Columns

|  |  |  |  |
| --- | --- | --- | --- |
| **Column name** | **Type** | **Properties** | **Description** |
| DoctorID | int | PK |  |
| FirstName | varchar(32) |  |  |
| LastName | varchar(32) |  |  |
| Gender | varchar(20) |  |  |
| DoctorType | int |  |  |
| Active | bool |  |  |

## 1.3. Table DoctorType

1.3.1. Columns

|  |  |  |  |
| --- | --- | --- | --- |
| **Column name** | **Type** | **Properties** | **Description** |
| DoctorType | int | PK |  |
| Description | varchar(100) |  |  |

## 1.4. Table MedicineStock

1.4.1. Columns

|  |  |  |  |
| --- | --- | --- | --- |
| **Column name** | **Type** | **Properties** | **Description** |
| MedicineID | int | PK |  |
| MedicineName | varchar(100) |  |  |
| BatchNumber | int |  |  |
| MedicineManufacturer | varchar(100) |  |  |
| StockQty | int |  |  |
| Price | int |  |  |
| ExpiryDate | timestamp |  |  |

## 1.5. Table DoctorPrescription

1.5.1. Columns

|  |  |  |  |
| --- | --- | --- | --- |
| **Column name** | **Type** | **Properties** | **Description** |
| PrescriptionID | int | PK |  |
| PatientID | int |  |  |
| DoctorID | int |  |  |
| PrescriptionDate | timestamp |  |  |

## 1.6. Table PrescriptionDetails

1.6.1. Columns

|  |  |  |  |
| --- | --- | --- | --- |
| **Column name** | **Type** | **Properties** | **Description** |
| ID | int | PK |  |
| PrescriptionID | int |  |  |
| MedicineID | int |  |  |
| Quantity | int |  |  |

## 1.7. Table OrderSummary

1.7.1. Columns

|  |  |  |  |
| --- | --- | --- | --- |
| **Column name** | **Type** | **Properties** | **Description** |
| OrderID | int | PK |  |
| PrescriptionID | int |  |  |
| OrderDate | timestamp |  |  |
| MedicalStaffID | int |  |  |

## 1.8. Table MedicalStaffEmployee

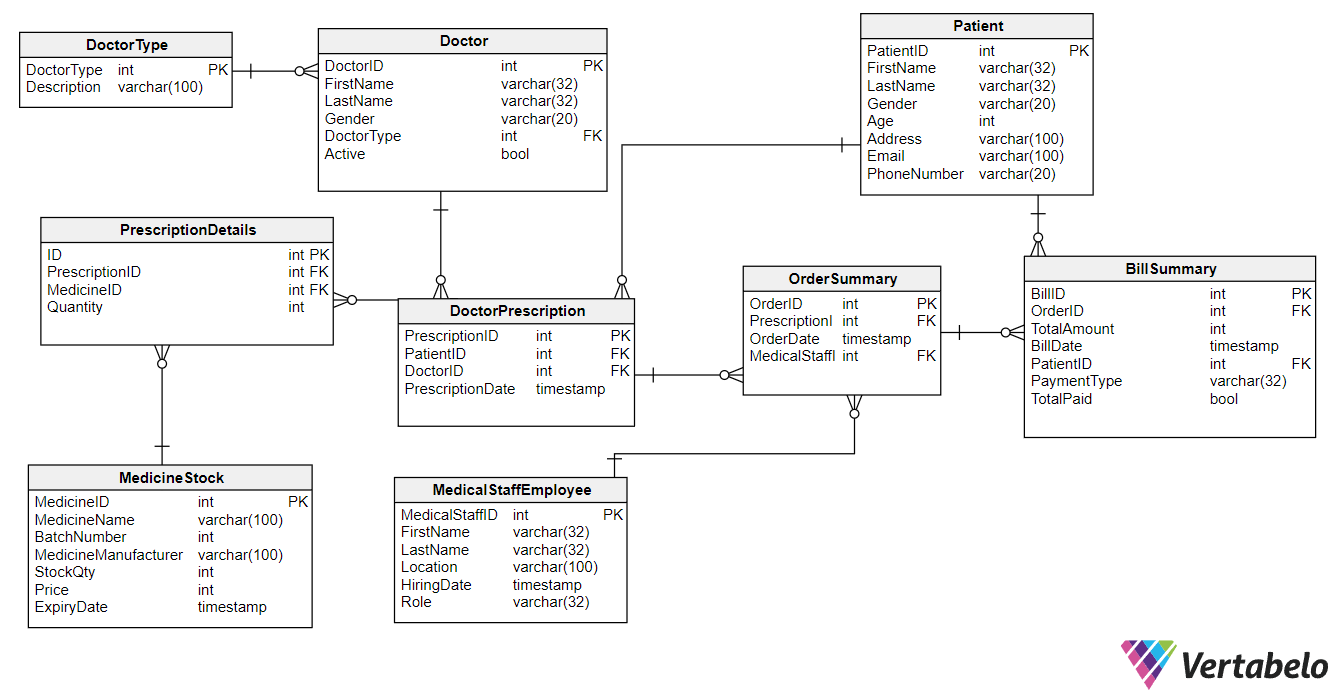
1.8.1. Columns

|  |  |  |  |
| --- | --- | --- | --- |
| **Column name** | **Type** | **Properties** | **Description** |
| MedicalStaffID | int | PK |  |
| FirstName | varchar(32) |  |  |
| LastName | varchar(32) |  |  |
| Location | varchar(100) |  |  |
| HiringDate | timestamp |  |  |
| Role | varchar(32) |  |  |

## 1.9. Table BillSummary

1.9.1. Columns

|  |  |  |  |
| --- | --- | --- | --- |
| **Column name** | **Type** | **Properties** | **Description** |
| BillID | int | PK |  |
| OrderID | int |  |  |
| TotalAmount | int |  |  |
| BillDate | timestamp |  |  |
| PatientID | int |  |  |
| PaymentType | varchar(32) |  |  |
| TotalPaid | bool |  |  |



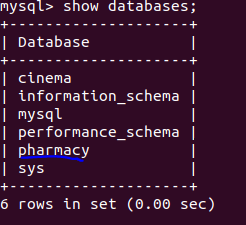
The detailed normalization analysis, relationship is mentioned in the **a) initial proposal** section

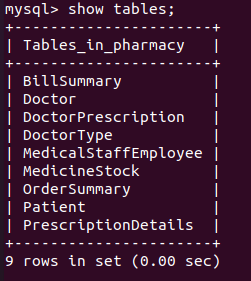
# Data Definition Language (DDL) Scripts

Please find the below DDL scripts generated by Vertabelo

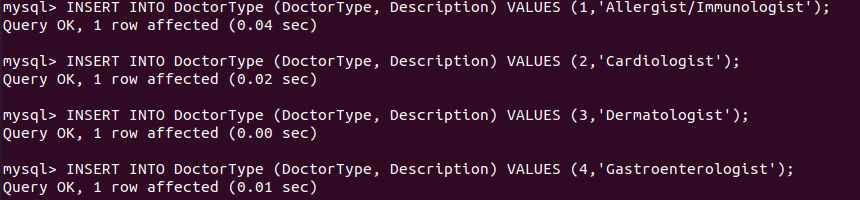


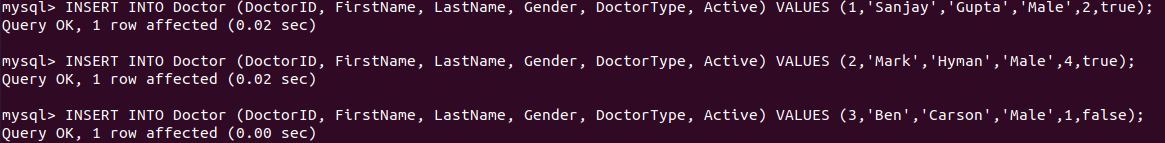


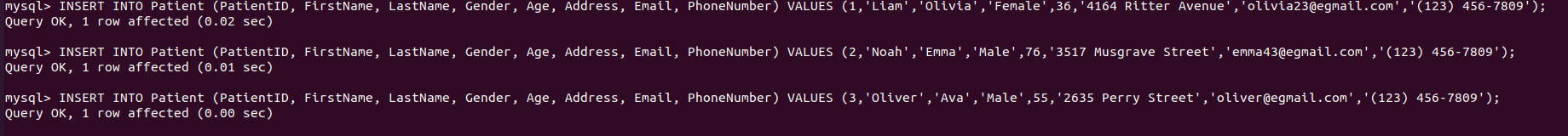


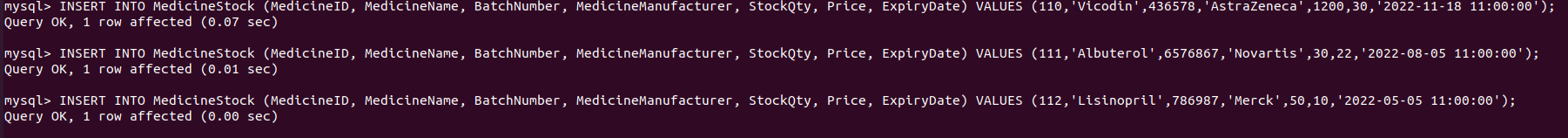


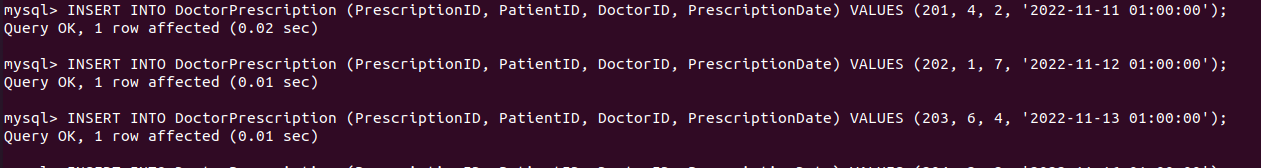
Below are the queries to insert queries

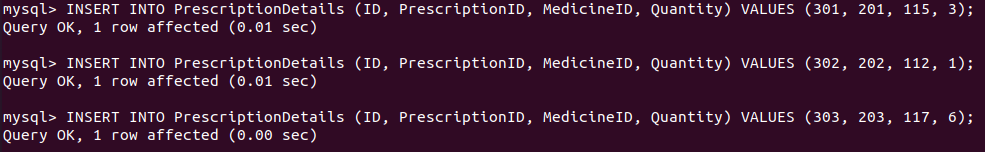


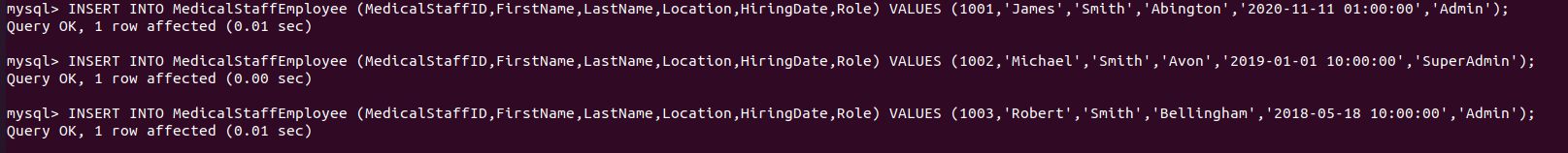


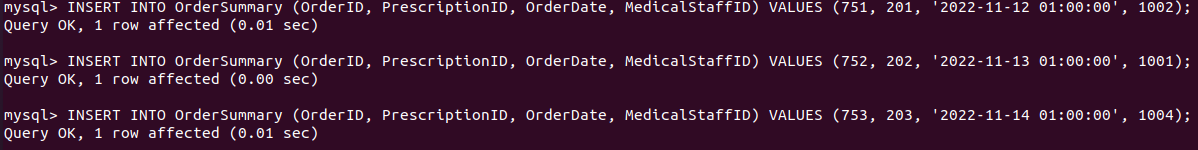


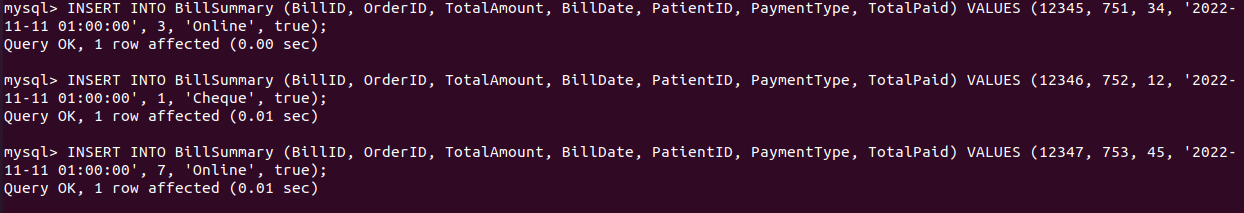










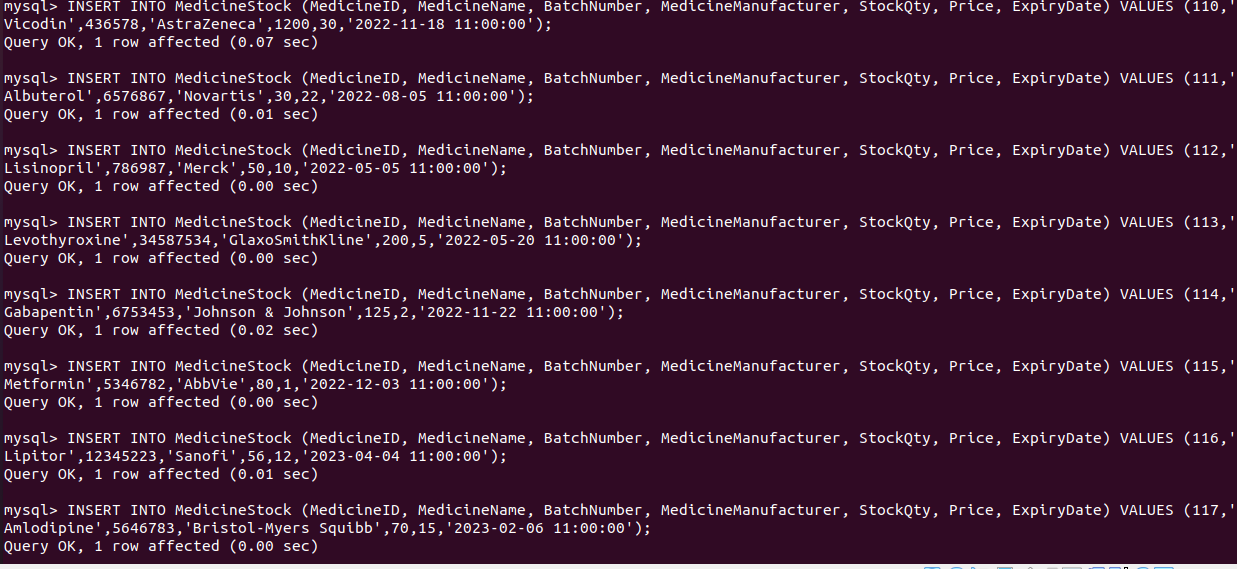


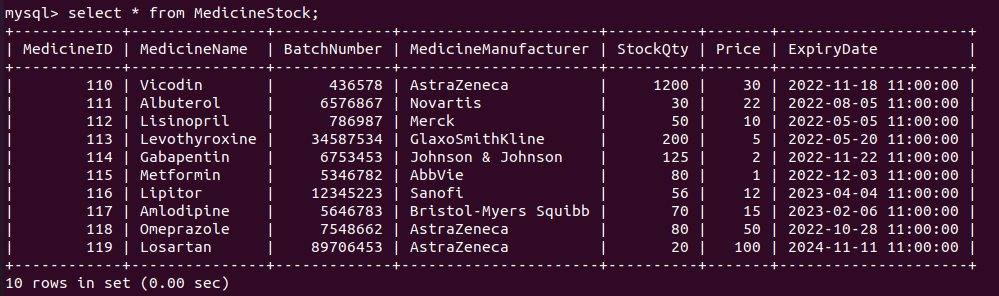
# Data Manipulation Language Scripts

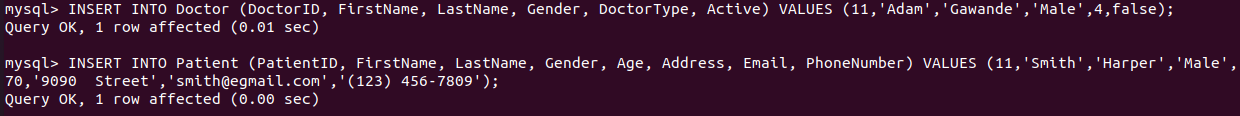
Please find the attached/uploaded in the GitHub DML queries of the above

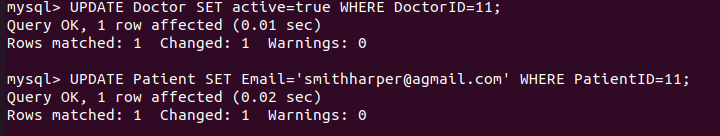


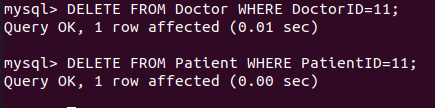
Please find screenshots for the above queries.

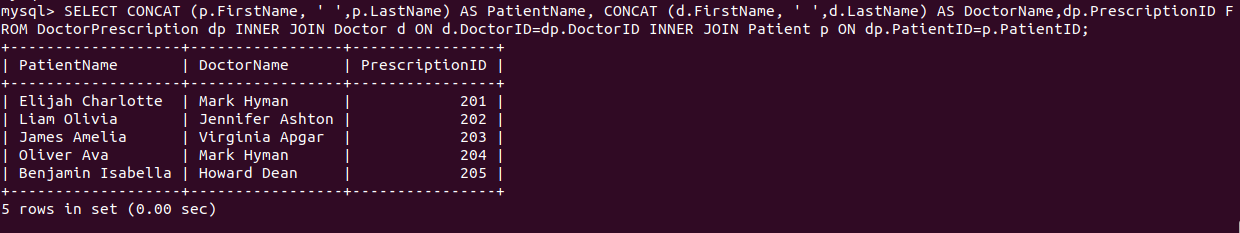


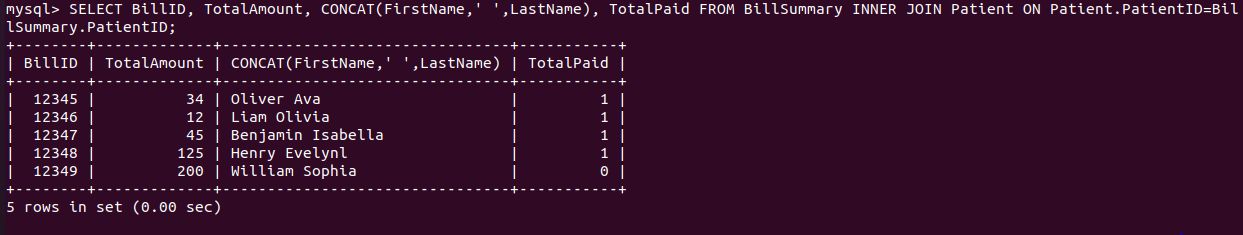


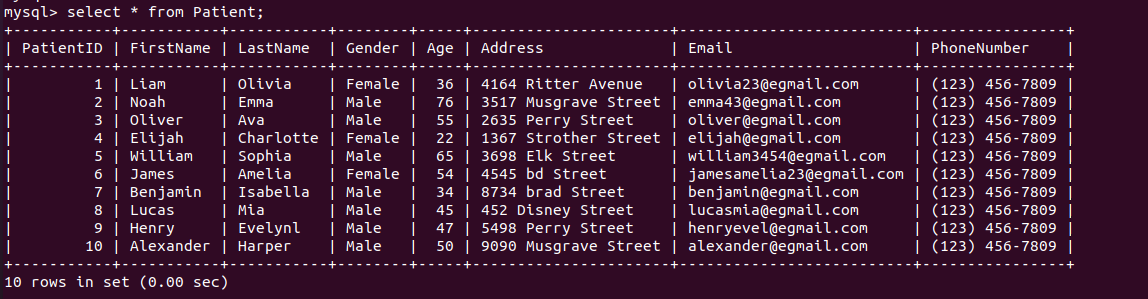


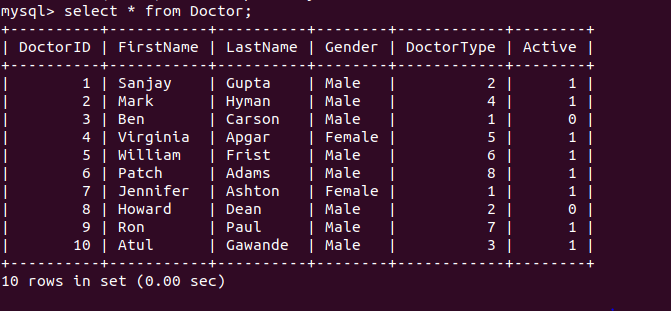












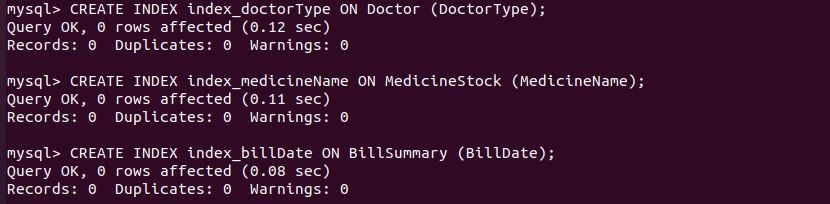
# Indexes

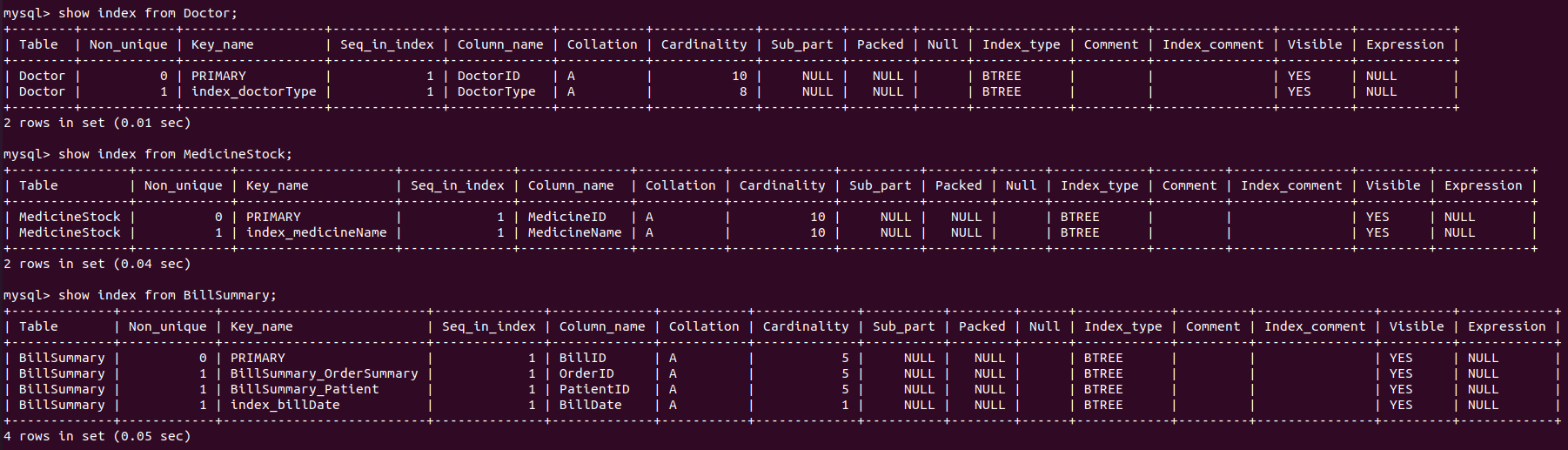
In the Pharmacy Management system, medicines availability, doctor details and the bill details has more dependency and users will frequently ask these details to fulfill the requests.

Below are the 3 possible indexes that can be applied

1. Fetching doctors list by searching the doctor type Physiatrist. Radiologist etc. so we can create the index on doctor table on doctor type.
2. Similarly, medalists, patients frequently used to check the availability of the medicine by name, so there is need to index on medicine name on MedicinceStock table.
3. Since the bill and order information grows automatically, sometimes we need to retrieve the bill details for auditing purpose, for the we can apply index on the bill date column of BillSummary Table.

Below are the SQL needed to generate the indexes. (Also available in DML scripts in GitHub repo)





When a database table is consulted, indexes are utilized to quickly identify data without having to scan every row in the table. Indexes can be built utilizing one or more columns from a database table, allowing for quick random lookups as well as efficient access to ordered items.

From the indexes created when the user performs the where condition on any of the indexed column like DoctorType , MedicineName and BillDate the result will be quicker because index built as Tree . so, the complexity will be O(logN) while searching.

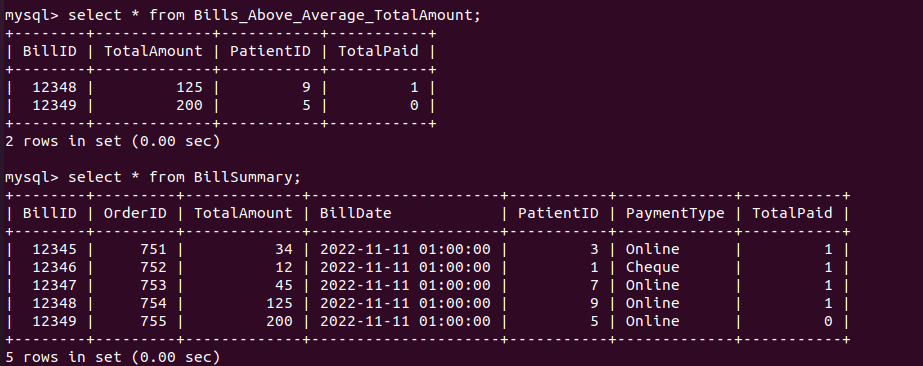
# Views

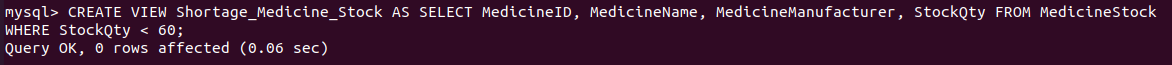
Please find the below sql to create the views

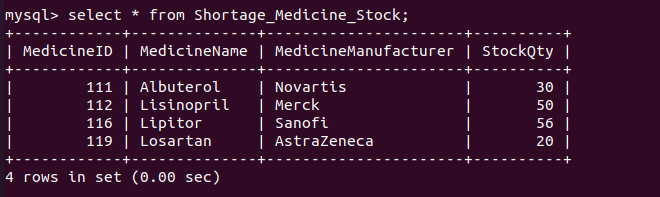
1. View to show the bills whose totalPaid is greater than average total paid
2. View to show the medicines whose stock is less than 60

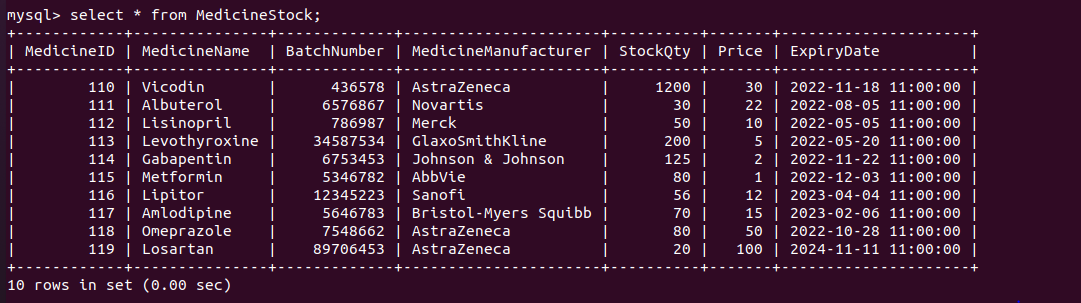
The below queries can be found in the DML scripts in GitHub repo











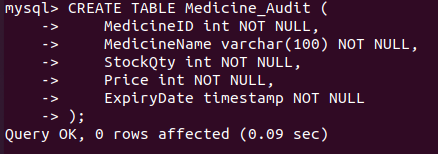
A view is a virtual table in SQL that is created from the result set of a SQL operation.

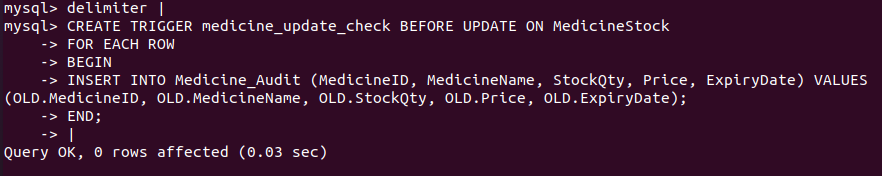
A view is similar to a table in that it has rows and columns. Fields from one or more real tables in the database are used in views.

You can populate a view with SQL statements and functions to portray data as if it came from a single table.

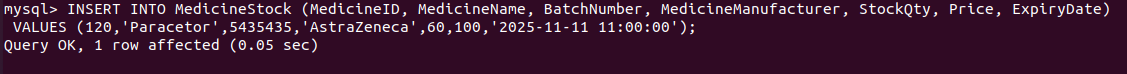
# Triggers

Please find the below SQL in GitHub as well.

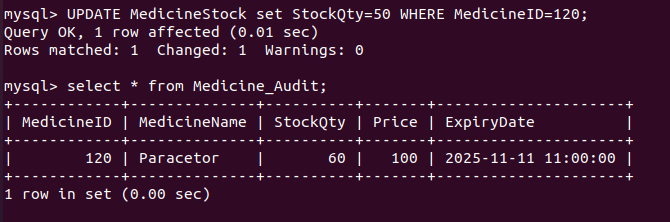


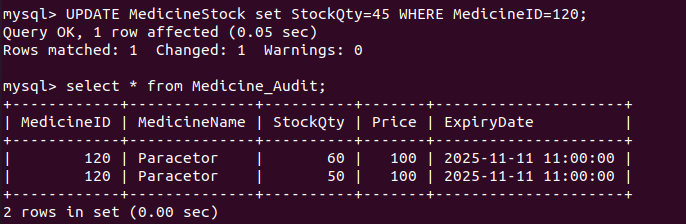


A trigger is a sort of stored procedure that executes automatically when a database server event occurs. When a user attempts to edit data via a data manipulation language (DML) event, DML triggers are triggered. INSERT, UPDATE, and DELETE statements on a table or view are DML events.







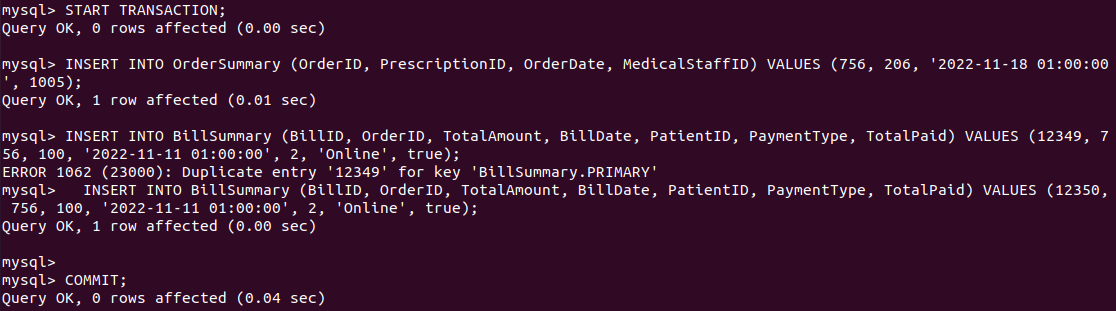


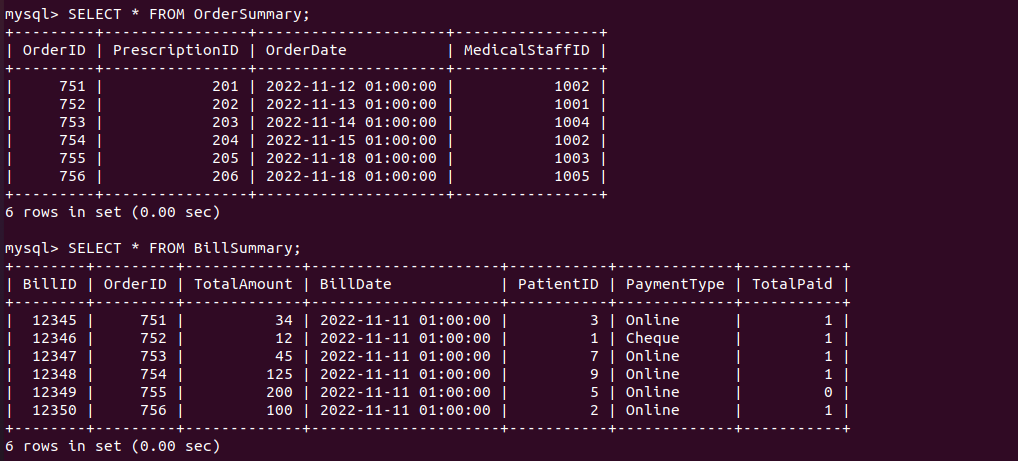
In our use case, we have created the trigger **BEFORE UPDATE** on MedicineStock update, whenever stockQty , price etc.. updates a new record will be inserted into the Medicine\_Audit for auditing purpose.

As you can see on the above screenshots whenever there is a update on MedicineStock table a new record is inserted with old values into the Medicine\_Audit.

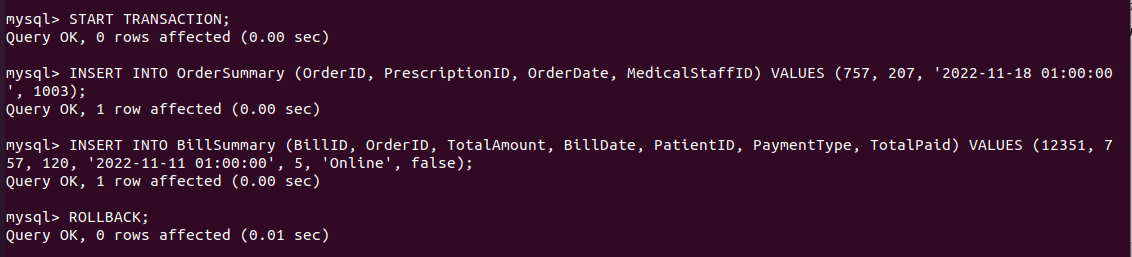
# Transactions

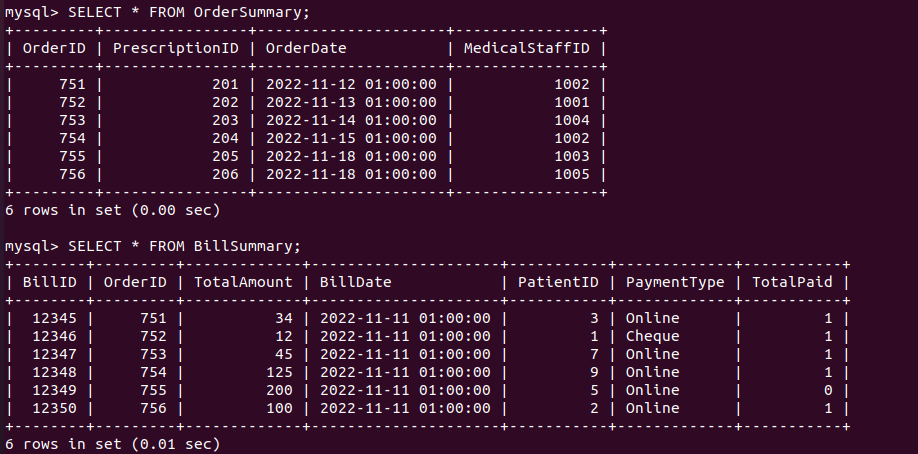
In MySQL, a transaction is a set of statements, queries, or operations, such as select, insert, update, or delete, that are executed sequentially as a single work unit and can be committed or rolled back. When a transaction makes many database modifications, two things happen:





When the transaction is committed, either all modifications are successful.





When a transaction is rolled back, all changes are undone.

As you can see after **rollback** orderid 757 and billid 12351 are not inserted into the respective tables.

# Database Security

**Note:** please find below scripts in uploaded GitHub DML scripts as well.

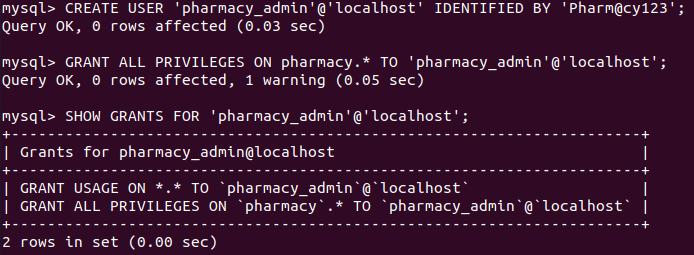
From the pharmacy management system there is need to secure the database and provide different access privileges to different users.

**Pharmacy\_admin :** this user has all the privileges on the pharmacy database, this user can update , add tables, drop tables, etc.

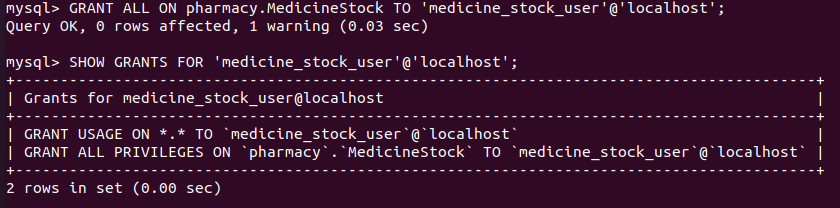
**Medicine\_stock\_user:** this user has the access only on MedicineStock table where this user can update, select, delete and insert the medicine records.

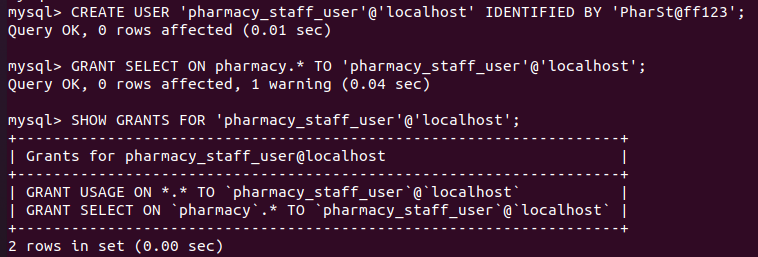
**Pharmacy\_staff\_user:** pharmacy staff can view all the tables in read-only privilege

Below are the screenshots that captured while creating the users and providing the permission.

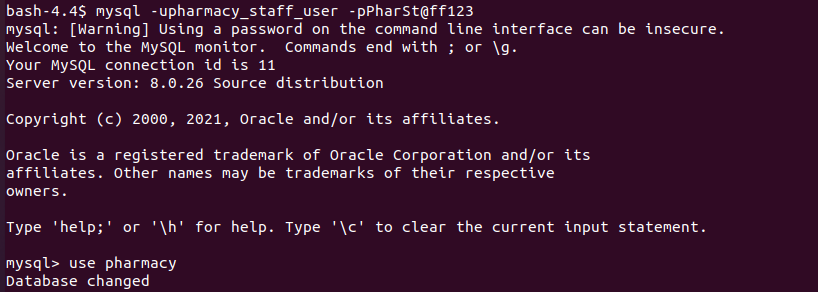


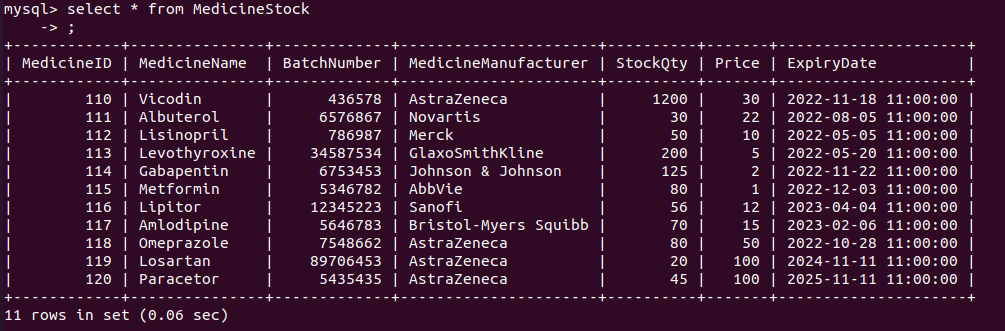


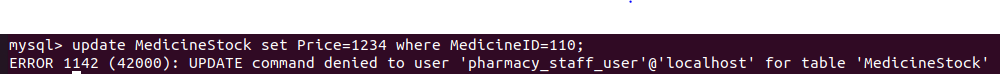




For Example, we can see ‘pharmacy\_staff\_user’ has only read only access on tables

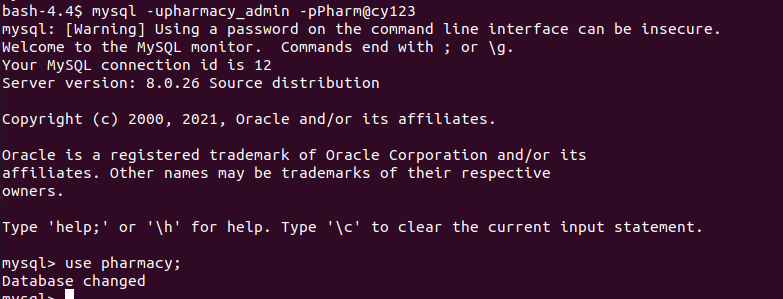


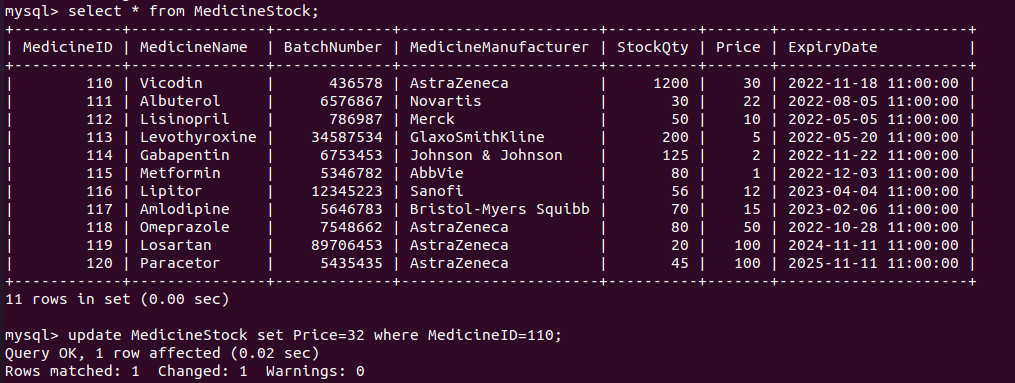




As we can see pharmacy\_staff\_user is denied to update the table.

Similarly, a pharmacy\_admin has all the privileges on pharmacy database.





Pharmacy\_admin has all privileges on pharmacy database. as we could able to update the records successfully.

Here is a short list of other common possible permissions.

* ALL PRIVILEGES- as we saw previously, this would allow a MySQL user full access to a designated database (or if no database is selected, global access across the system)
* CREATE- allows them to create new tables or databases
* DROP- allows them to them to delete tables or databases
* DELETE- allows them to delete rows from tables
* INSERT- allows them to insert rows into tables
* SELECT- allows them to use the SELECT command to read through databases
* UPDATE- allow them to update table rows
* GRANT OPTION- allows them to grant or remove other users’ privileges

# Locking and Concurrent Access

**Note:** please find below scripts in uploaded GitHub DML scripts as well.

A table lock is a mechanism that prevents unauthorized access to the data stored in the table. MySQL allows a client session to expressly acquire a table lock in order to share the table's contents with other sessions. MySQL also supports table locking, which prevents unauthorized changes to the same table within a specified time period.

In MySQL, a session can only acquire or release locks on the table for itself. As a result, one session will not be able to acquire or release table locks for other sessions. It's worth noting that table locking requires TABLE LOCK and SELECT rights.

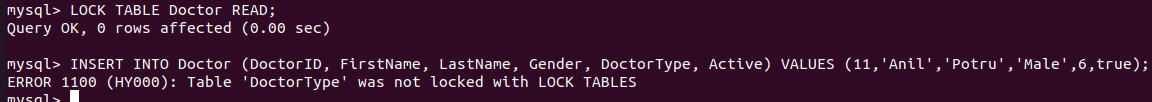
In MySQL, table locking is primarily used to tackle concurrency issues. It will be utilized during the execution of a transaction, that is, reading a value from a table (database) and subsequently writing it to the table (database).

MySQL has two types of table locks: read-only and read-write.

READ LOCK: This lock allows a user to only read the data from a table.

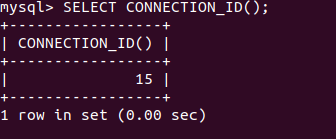
WRITE LOCK: This lock allows a user to do both reading and writing into a table.

1st Session

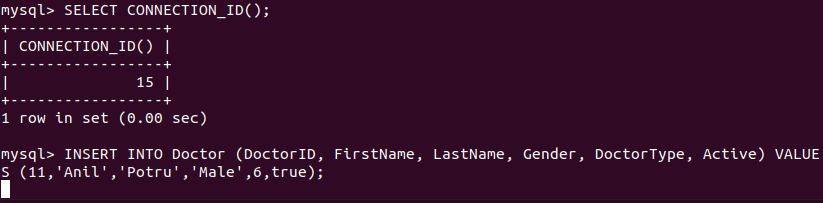


As we can see on successful lock on the Doctor table, we are unable to perform the write operation to the doctor table

2nd Session

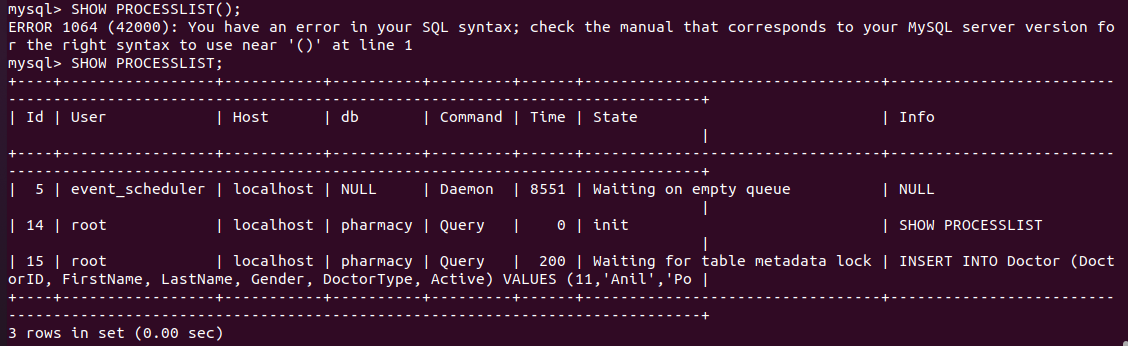


SELECT CONNECTION\_ID() shows the active connectionid list



As we can see in the 2nd session query is in waiting state due to the locking

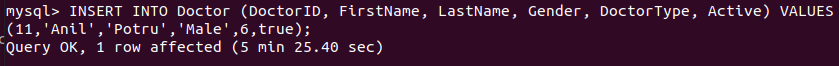
SHOW PROCESSLIST will show the waiting queries



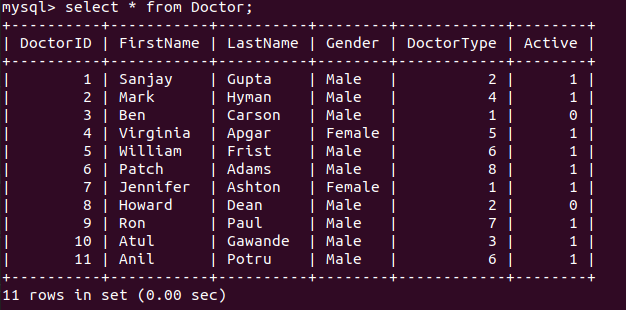
1st Session



2nd Session



So, on unlocking the Doctor table from 1st session will free the Doctor table and which result successful insertion in the second session.



# Backing Up Your Database

Pharmacy Management system is a critical application which involves in patient, doctor, previous prescription and bill summary records

Data is an important part of running any organization, no matter how big or little it is. You could lose a lot of money if your business data is lost. Many businesses rely on online data backup for data protection to ensure that their sensitive data is safe from theft, damage, or disasters.

**Frequency:** Businesses risk losing their credibility and perhaps facing legal action if they lose vital patients or orders data if they do not use fast and adequate backup solutions. If you're in charge of storing information on your patients and doctors, you'll need to upgrade your security system to ensure that sensitive data is kept safe.

For the Pharmacy Management system 24hrs backup will be good enough where we can schedule the job in between 23:00 to 00:00 where the users will be using the application rarely at that time.

**Location:** Setting up and saving the database in on-premise servers will takes lot of time and manual effort, which cloud providing services like aws MySQL etc... we can leverage the cloud advantages. Where cloud providers will provide the autoscaling, reliability and fault tolerance out of box. With cloud autoscaling can be done with matter of minutes to hours.

**Automation:** There are many ways to automate the backup of MySQL. The ideal and simple way to automate the backup is to add the MySQL backup command to the job which runs on timely basis. If we choose cloud provider services of MySQL, backups will be automatically supported by cloud providers like aws, azure etc.

**Security:** An online database backup will save you money by eliminating the need to set up your own system and allowing you to make the most of your existing resources. The easiest method to ensure that your data is safe and secure is to use an online database backup service.

Small and developing businesses are just as vulnerable to data breaches as large corporations. This is why every business owner should think about investing in a safe database backup solution. Small firms are vulnerable to data theft, which can result in the loss of extremely sensitive data.

Factors such as strengthened encryption and [passwords](https://www.businessblogshub.com/2014/08/realistic-password-management-tips/) are critical for securing and backing up data.

Please find below command to take backup of MySQL dump, here we are using **mysqldump** utility to take backup of the MySQL database.



Which will result in dump of pharmacy database

Please find the below (uploaded in to GitHub) backup sql file

Similarly, whenever there is a need to restore the database backup below is the command used to restore the database.

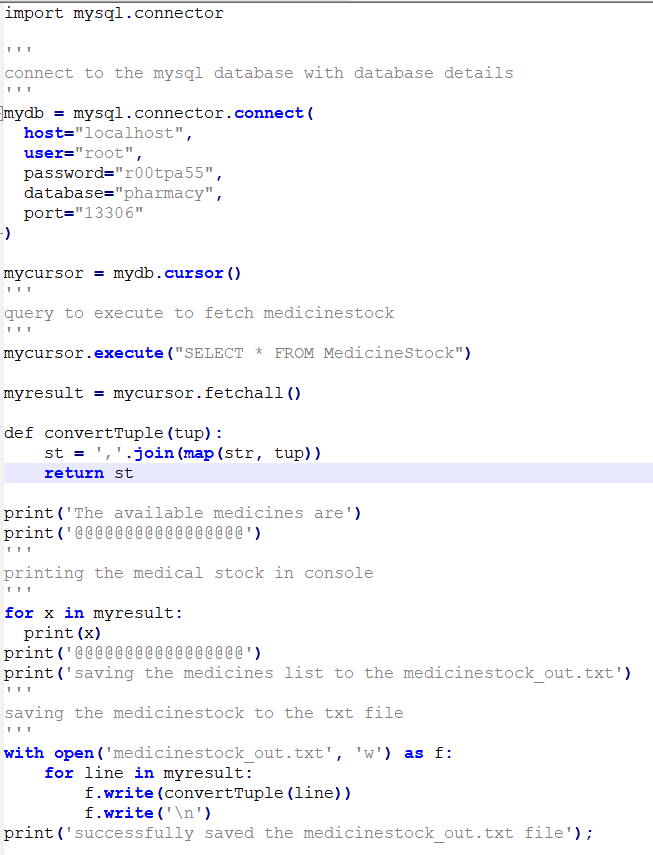


The above commands can be found in GitHub repo.

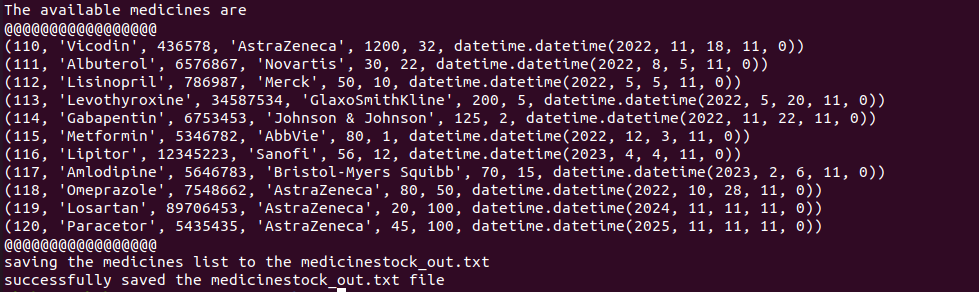
**Note:** If we choose cloud service for our database, the above commands may not be used since most of the above backup and restore features will be provided by cloud provider.

# Python Programming

Below is the python code to connect to the pharmacy database using mysql.connector dependency, and printing the medicine list in the console and saving the medicine list to the medicinestock\_out.txt file.



Please find below the console output and the txt file generated



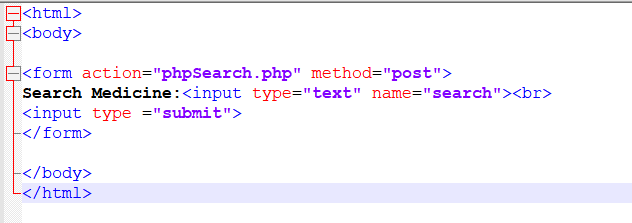


The source code and output found in GitHub as well

# PHP Programming

Please find below html for initial search page

**search.html**

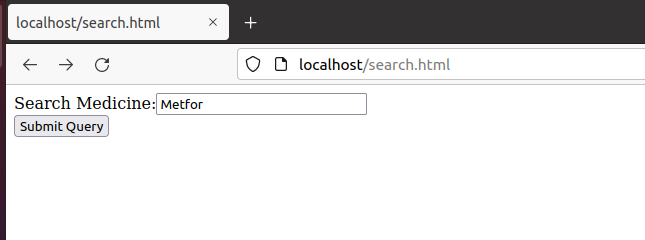


On successful entering the medicine name and clicking on enter below php code will be invoked

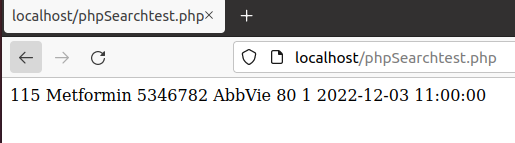
**phpSearch.php**



In php, connection to the mysql database has been established.



On entering the medicine name and submit



The respective results will appear in the php page.

Please find the source in uploaded in GitHub.

# Suggested Future Work

In Pharmacy Management system the data grows rapidly on daily basis. RDBMS is expensive in terms of license and effort. If there is a requirement to accommodate the new columns or new fields, it takes lot of effort to alter and accommodate those changes which results in downtime of the database.

To address the RDBMS issues, NOSQL better fit if there is need to accommodate the new fields without impacting the existing database system. NOSQL has better performance over RDBMS.

Below are the advantages of using the cloud over on-premise for database

* Access from anywhere and at any time - You may use a web browser on any device to access your applications at any time and from anywhere.
* Cloud is cost-effective since there are no upfront charges; instead, you make regular payments, making it an ongoing expense (OpEx). While the monthly cost mounts up over time, maintenance and support services are included, so no annual commitments are required.
* Predictable expenses — Take advantage of monthly fees that include software licenses, upgrades, support, and daily backups.
* IT that is worry-free — You don't have to worry about the maintenance of your software or the hardware it runs on because cloud software is hosted for you; compatibility and upgrades are handled by the cloud service provider.
* High levels of protection - Because data centers use security procedures that are out of reach for most organizations, your data is typically safer in the cloud than on a server in your on-premise.

**Advantages of a NoSQL database**

The scalability and decentralised nature of a NoSQL database are two of its benefits. It allows for the creation of distributed structures.

As databases, they're usually far more open and versatile. They make it considerably easier to adjust to the needs of a project than entity-related models.

It is possible to make changes to the schemes without shutting down the database.

Horizontal scaling: instead of being confined to a single huge device, they can expand in number.

They can be used on devices with limited resources.

Query optimization for databases that store a lot of data.

**NoSQL databases have a number of drawbacks.**

The atomicity of instructions and the integrity of the data are not considered in all NoSQL databases. They have the ability to tolerate what is known as eventual consistency.

SQL instructions have compatibility concerns. The query language of new databases has its own peculiarities, and it is not yet 100 percent compatible with SQL used in relational databases. In a NoSQL database, support for work query concerns is more difficult.

Standardization is lacking. There are various NoSQL databases, but none of them follow the same standards as relational databases. These databases are expected to have an unclear future.

They frequently have access to consoles or management tools that aren't really useful.

# Activity Log

|  |  |  |
| --- | --- | --- |
| *Date* | *Name* | *Comments / Challenges* |
| 3rd Nov 2021 | Anil Potru | Worked on the use case, and created vertabelo account |
| 5th Nov 2021 | Anil Potru | Worked on the Medical Management System ER design |
| 6th Nov 2021 | Anil Potru | Created the tables and relations using vertabelo |
| 12th Nov 2021 | Anil Potru | Installed mysql and exported the DDL and documents from vertabelo |
| 16th Nov 2021 | Anil Potru | Worked on the DML queries and sample data for insertion, downloaded the sample medicines list from internet. |
| 18th Nov 2021 | Anil Potru | Worked on the update, select ,delete and join queries. |
| 20th Nov 2021 | Anil Potru | Worked on the indexes and created indexes |
| 24th Nov 2021 | Anil Potru | Worked on the triggers, created trigger for the medicine stock table. |
| 25th Nov 2021 | Anil Potru | Worked on transactions and rollback |
| 28th Nov 2021 | Anil Potru | Worked on the database security , created users with GRANT permissions |
| 30th Nov 2021 | Anil Potru | Gone through the videos and worked on the locking of tables |
| 1st Dec 2021 | Anil Potru | Gone through the documentation of mysql and found the backup command |
| 3rd Dec 2021 | Anil Potru | Worked on python and php scripts to connect to mysql |