RxPro

“Exceeding expectations in pharmacy care.”

Group 88

Member 1: Manav Mittal, 2021538

Member 2: Garv Makkar, 2021530

Project Scope

The scope of the project includes the design and development of a Data Base Management System (DBMS) for a Pharmacy Store. This includes the creation of a database structure to store and manage customer, employee, supplier and inventory information, as well as the implementation of security features such as user authentication and access control. The DBMS will also be able to track and manage the movement of stock, generate various types of reports and Prepare sales and purchase invoices. Additionally, the DBMS will have the ability to list expired medicines and alert for the medicines that will be expiring soon. The system will be designed to be user-friendly and easy to use for employees of the pharmacy management store.

Functional Requirements

1. Customer Management: The DBMS should be able to store and retrieve customer information, such as name, address, and contact information. It should also be able to track customer purchase history. It will also be able to track any current order by the customer.
2. Supplier management: It will handle details about suppliers, like their names, contacts, items supplied, current orders etc.
3. Employee management: It will handle the details about employees, their name, age, contact, salary, work hours etc.
4. Security management: It will provide security features such as user authentication and access control to protect sensitive information.
5. Inventory Management: Information regarding medicines and other products, like their names, manufacturing company, manufacturing and expiry dates, cost price, selling price, current stock etc. It will be able to track the quantity of stock on hand, as well as the movement of stock in and out of the store.
6. List expired medicines in order to be disposed of, and also give an alert for those who will be expiring soon (within three months).
7. Generate reports: It will be able to generate various types of reports, such as sales reports etc.
8. Prepare sales and purchase invoices.

Technical Requirements

Entities involved

1. Customer: This entity will store information about customers, such as name, address, contact information, and purchase history.
2. Employee: This entity will store information about employees, such as name, age, contact, salary, work hours and role.

3. Supplier: This entity will store information about suppliers, such as name, contact, items supplied, and current orders.
4. Inventory: This entity will store information about the products in the store, such as name, manufacturing company, manufacturing and expiry dates, cost price, selling price, and current stock.
5. Sales: This entity will store information about sales transactions, such as date, time, products sold, quantity and customer.
6. Purchase: This entity will store information about purchase transactions, such as date, time, products purchased, quantity and supplier.
7. Orders: This entity will store information about current orders of customers and suppliers, such as date, products, quantity and status.
8. Expired Medicines: This entity will store information about expired medicines, such as name, expiration date, quantity and status (disposed of or not).
9. Reports: This entity will store information about various types of reports, such as sales reports.
10. Invoices: This entity will store information about the purchase and sales invoices, such as date, products, quantity and amount.

Stakeholders

1. Pharmacy Store Owners: They are the primary stakeholders who will be using the system and will benefit from its implementation. They will be responsible for making decisions regarding the development and implementation of the DBMS.
2. Pharmacy Store Employees: They will be the users of the system and will be responsible for inputting and managing data within the DBMS. They will also be responsible for generating reports and invoices.
3. Customers: They will be indirectly impacted by the DBMS as it will help the store to manage their purchase history and current orders.
4. Suppliers: They will be indirectly impacted by the DBMS as it will help the store to manage their current orders and deliveries.

Access constraints

1. Only authorized employees of the pharmacy store will be granted access to the DBMS.
2. Employees will be assigned different levels of access based on their role and responsibilities within the store. For example, managers may have access to all features of the DBMS, while regular employees may only have access to certain parts of the system.
3. Access to sensitive information, such as customer and employee personal information, will be restricted to only those employees with a valid need to know.
4. Inventory and sales data will be accessible to all employees, but the ability to make changes to this information will be restricted to only certain employees, such as managers.
5. The system will be designed to prevent unauthorized access and protect against data breaches through the use of secure login credentials, such as unique usernames and strong passwords.
6. The DBMS will support setting up and managing user roles and permissions.
7. The system will have the capability of handling multiple concurrent user connections.

Tentative schema for the project

1. Customers: This table would store information about customers, such as name, address, contact information, and purchase history. It would have columns like customer id, name, address, contact, purchase history, etc.

2. Employees: This table would store information about employees, such as name, age, contact, salary, work hours, and role. It would have columns like employee id, name, age, contact, salary, work_hours, role, etc.
3. Suppliers: This table would store information about suppliers, such as name, contact, items supplied, and current orders. It would have columns like supplier id, name, contact, items supplied, current orders, etc.
4. Inventory: This table would store information about the products in the store, such as name, manufacturing company, manufacturing and expiry dates, cost price, selling price, and current stock. It would have columns like product id, name, manufacturer, mfg date, expiry date, cost price, selling price, current stock, etc.
5. Sales: This table would store information about sales transactions, such as date, time, products sold, quantity, and customer. It would have columns like sales id, date, time, product id, quantity, customer id, etc.
6. Purchase: This table would store information about purchase transactions, such as date, time, products purchased, quantity, and supplier. It would have columns like purchase id, date, time, product id, quantity, supplier id, etc.
7. Orders: This table would store information about current orders of customers and suppliers, such as date, products, quantity, and status. It would have columns like order id, date, product id, quantity, status, customer id/supplier id, etc.

Some other technical requirements

1. The DBMS will support ACID (Atomicity, Consistency, Isolation, Durability) properties to ensure data consistency and integrity.
2. It will support Indexing and normalization for better performance.
3. It will have the capability of scalability to handle the increase in data and transactions.
4. The system will be designed to be accessible via web or desktop application.
5. It will have the ability to handle multiple concurrent user connections.
6. It will support the ability to import and export data in various formats, such as CSV, Excel, etc.
7. It will have the ability to handle a large amounts of data with good performance and response time.

Tech Stack

Database: MySQL

Backend: Python

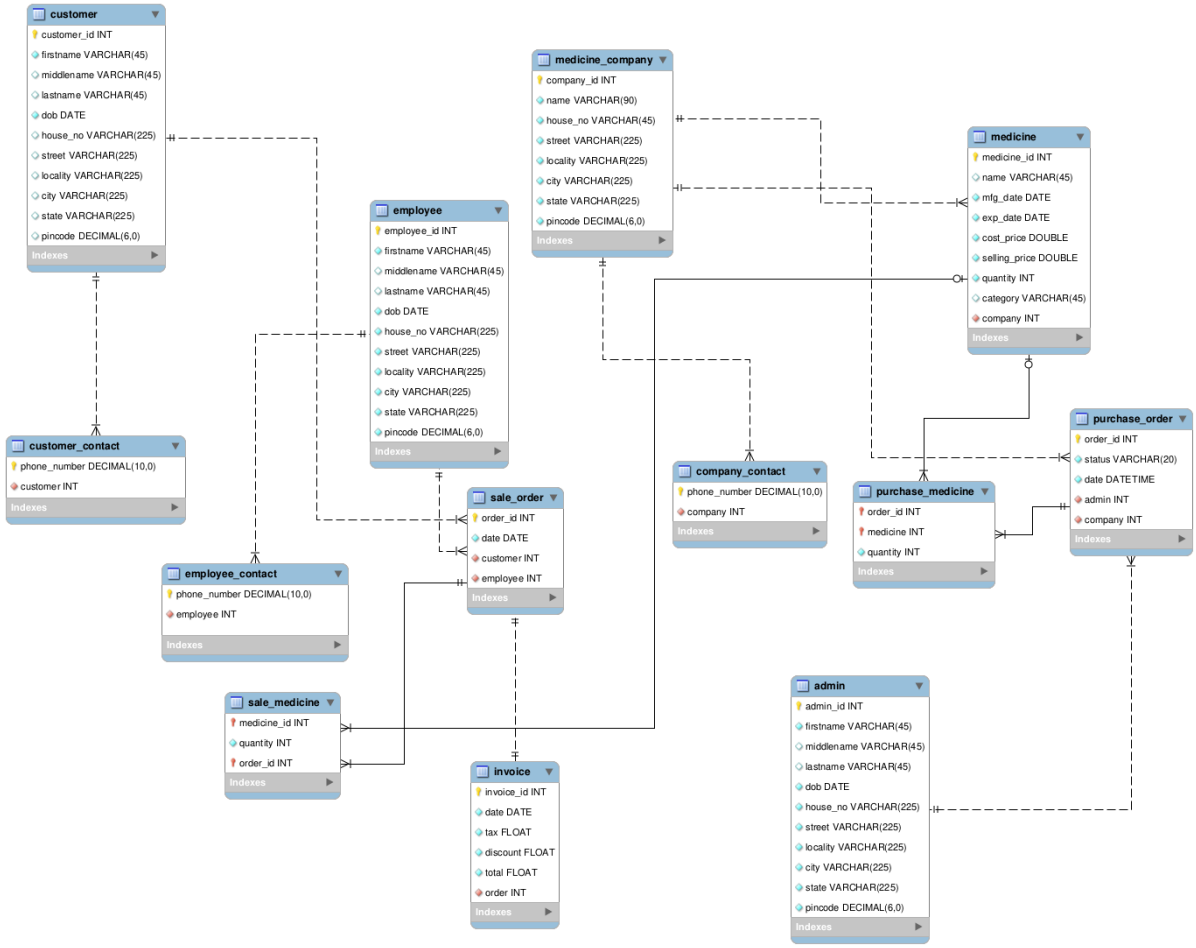
Frontend: Yet to be decided.

Assumptions & brief explanation

1. The application is solely for store managers and employees. The customer will not interact with the application. A customer Entity is created to record who purchased the medicines.
2. Employees will not be able to change the information about the medicine directly. He/She will be only responsible for creating orders after the customer tells the employee which medicines he/she wants to buy. After the order is created, the quantity of the medicines from the stock will be automatically deducted, and an invoice will be generated if the employee wants to do so.
3. Admin will be able to create orders from the suppliers, and after that, depending upon the status of the purchase order, the medicine stock will be updated. Only the admin will be responsible for creating purchase orders of medicines from the suppliers.

4. All the entities' data is stored in their respective tables only and will be fetched from there by using respective ids. This is done to avoid the repetitive storage of data.
5. A customer can have multiple contacts, and it is his/her choice to give contact for the registration. It is not mandatory to give a customer contact number while registering a customer.
6. Admin contact is not stored. It is assumed that the admin is the one who will need the contact of other people.
7. We have not added employee salary into employee table but we will add it later using alter command if we find it necessary. Because the main purpose of out project is to provide a inventory management to a medicine shop. Salary is a attribute which can be directly monitored by the store manager.

ER - traditional



ER - UML Notation and Relational diagram

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/2131a40b-f96e-47f9-aba7-5d508fa6e10e/88_RxPro_ERDiagram_RelationalShema_2021538.pdf

Current number of Entries in each table

1. Admin - 2
2. Customer - 1000
3. Employee - 10
4. Customer_contact - 1000
5. employee_contact - 20
6. invoice - 200
7. medicine - 1000
8. medicine_company - 30
9. purchase_medicine - 346
10. purchase_order - 200
11. sale_medicine - 607
12. sale_order - 200
13. company_contact - 60

TOTAL NUMBER OF ROWS = 4675 (it is the sum total of the number of rows in all the tables)

What we have done in Submission 3

1. We have generated random data from mockaroo.com for the following tables:
    a. Admin
    b. Customer
    c. Employee
    d. Customer_contact
    e. employee_contact
    f. medicine
    g. medicine_company
    h. purchase_order
    i. sale_order
    j. purchase_medicine
    k. sale_medicine
2. Only the data for the invoice table is derived because it included the price total of all medicines for a respective sale_order. So to do so, we have used a python script written by us only. It is attached below. All the queries for inserting data into the invoice table are generated using this python script.

```
import mysql.connector as mcon

connection = mcon.connect(host='localhost',user='root',password='Ma@060304', database='mydb')
mycursor = connection.cursor();

mycursor.execute("use mydb")

mycursor.execute("select * from sale_medicine")
sale_data = mycursor.fetchall()

d_total = {}

for i in sale_data:
    mycursor.execute("select selling_price from medicine where medicine_id=%d"%i[0])
    price = mycursor.fetchone()[0]
    if i[2] not in d_total.keys():
        d_total[i[2]] = price*i[1]
        d_total[i[2]] = round(d_total[i[2]], 2)
    else:
```

```
        d_total[i[2]] += price*i[1]
        d_total[i[2]] = round(d_total[i[2]], 2)

d_date = {}

for i in sale_data:
    mycursor.execute("select date from sale_order where sale_order.order_id=%d"%i[2])
    date = mycursor.fetchone()[0]
    d_date[i[2]] = date

discount = 10.00
tax = 18.00


invoice_id = 1

for i in d_total.keys():
    total = d_total[i] - d_total[i]*discount/100
    total = total*(1+tax/100)
    total = round(total, 2)
    print(f"insert into invoice values ({invoice_id},'{d_date[i]}',{tax},{discount},{total},{i});")
    invoice_id += 1


mycursor.close()
connection.close()
```

Submission 4 - 10 SQL queries

```
-- 1. List details of customers who have more than 1 orders
select customer_id, concat(firstname, " ", middlename, " ", lastname) as name from customer where customer_id in
                                      (select customer from sale_order group by customer having count(customer) > 1) ;
-- ------------------------------------------------------------------------------------------------------------------
-- 2. Calculate sum of prices of all medicine in an order
select order_id,
       date,
       round(sum(total), 2) AS TOTAL
from
    (select so.order_id,
            so.date,
            selling_price*sm.quantity total
     from sale_medicine sm,
          sale_order so,
          medicine m
     where
         sm.order_id=so.order_id
       and
         sm.medicine_id=m.medicine_id
     order by order_id) inv
group by order_id
order by order_id;
-- ------------------------------------------------------------------------------------------------------------------
-- 3. list all purchase orders and their details created by admin 1 and their medicines
select order_id,
       date,
       status,
       group_concat(concat(name, " ", quantity) separator ' | ') as medicines
from
    (select po.order_id,
            po.status,
            po.date,
            m.name ,
            pm.quantity
     from
         purchase_order po,
         purchase_medicine pm,
         medicine m
     where
         m.medicine_id = pm.medicine
       and
         pm.order_id = po.order_id
       and
         po.admin = 1) t
group by
    order_id;
-- ------------------------------------------------------------------------------------------------------------------
-- 4. List all customer details where customer id = 369 - (Can take this input from the Employee through frontend)
select customer_id,
       concat(firstname, " ",middlename, " ", lastname) name,
       dob,
       ci.contact,
```

```
       concat(house_no, " ", street, " ", locality, " ", city) address,
       state,
       pincode
from
    customer c
LEFT JOIN
    (select customer,
       group_concat(phone_number separator ", ") contact
     from
         customer_contact
     group by
         customer) ci
ON c.customer_id = ci.customer
where c.customer_id = 369;
-- ---------------------------------------------------------------------------------------------------------------------
-- 5. List employees in descending order of the profit they made
select employee_id,
       Name,
       round(sum(Profit), 2) Amount
from
    (select employee_id,
            concat(firstname, " ", middlename, " ", lastname) as Name,
            Profit
     from
         sale_order,
         employee,
         (select order_id,
                 round(sum(sellval) - sum(costval), 2) AS Profit
          from
              (select so.order_id,
                      selling_price*sm.quantity*0.9 sellval,
                      cost_price*sm.quantity costval
               from sale_medicine sm,
                    sale_order so,
                    medicine m
               where
                   sm.order_id=so.order_id
                 and
                   sm.medicine_id=m.medicine_id
               order by order_id) inv
          group by order_id
          order by order_id) profitTable
     where sale_order.employee = employee.employee_id
       and sale_order.order_id = profitTable.order_id) t
group by employee_id
order by Amount DESC ;
-- ---------------------------------------------------------------------------------------------------------------------
-- 6. Generate Invoice for a particular order assuming tax is 18% and discount is 10%
select order_id,
       date,
       Total
from
    (select order_id,
            date,
            round(sum(total)*0.9*1.18, 2) AS TOTAL
     from
         (select so.order_id,
                 so.date,
                 selling_price*sm.quantity total
          from sale_medicine sm,
               sale_order so,
               medicine m
          where
              sm.order_id=so.order_id
            and
              sm.medicine_id=m.medicine_id
          order by order_id) inv
     group by order_id
     order by order_id) t
where order_id = 169;
-- ---------------------------------------------------------------------------------------------------------------------
-- 7. Update purchase order status for all orders created before 380 days
update purchase_order
set status = 'placed'
where DATEDIFF(CURDATE(), date) < 380;

select order_id, status from purchase_order where DATEDIFF(curdate(), date) < 380;

-- Update purchase order status for a given order ids
update purchase_order
set status = 'placed'
where order_id=2;

select order_id, status from purchase_order where order_id = 2;
-- ---------------------------------------------------------------------------------------------------------------------
-- 8. Update stock
update medicine
```

```
set quantity = 253
where medicine_id = 26;

select medicine_id, quantity from medicine where medicine_id = 26;
-- ------------------------------------------------------------------------------------------------------------------------
-- 9. Update cost price of all medicines of a particular company by 10%
update medicine
set cost_price = round(cost_price*1.1, 2)
where company=3;

select medicine_id,cost_price from medicine where company = 3;
-- ------------------------------------------------------------------------------------------------------------------------
-- 10. change one of the existing mobile number of an employee
update employee_contact
set phone_number = 9999999998
where employee = 5
  and phone_number = 9999999999;
-- ------------------------------------------------------------------------------------------------------------------------
-- 11. View of all employee and their contacts
create view employee_details_contact as
    select employee_id,
           concat(firstname, " ", middlename, " ", lastname) as name,
           contact
    from employee,
         (select employee,
                 group_concat(phone_number separator ", ") as contact
          from employee_contact
          group by employee) d
where d.employee = employee.employee_id;
```

Project Deadline 5

Triggers

```
-- Trigger 1 ---------------------------------------------------------------------------------
-- Check wether enough stock is available before creating a sale order
delimiter //
CREATE TRIGGER check_quantity
before insert on sale_medicine
for each row
begin
    declare q int;
    select quantity into q from medicine where medicine_id = new.medicine_id;
    if new.quantity > q then
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'ERROR: Not enough stock';
    end if;
end//
delimiter ;
-- ------------------------------------------------------------------------------------------------------------------------
-- Trigger 2 ---------------------------------------------------------------------------------
-- Trigger to automatically delete employee_contact when employee is deleted
delimiter //
create trigger delete_employee
before delete on employee
for each row
begin
  delete from employee_contact where employee=OLD.employee_id;
end//
delimiter ;
```

OLAP Queries

```
-- ---------------OLAP-------------------------------------------------------------------------------------------------
-- ------------------------------------------------------------------------------------------------------------------------
-- Grouping of customers according to their addresses
SELECT
  state,
    city,
    locality,
    COUNT(customer_id) as customer_count
FROM
  customer
GROUP BY
  state,
  city,
    locality
    WITH ROLLUP;
-- ------------------------------------------------------------------------------------------------------------------------
-- odering of employees based on number of oderes they made month and year wise
```

```
SELECT
  employee as employee_id,
    MONTH(date) as month,
    YEAR(date) as year,
    COUNT(order_id) as count
FROM
    sale_order
GROUP BY
  employee,
  YEAR(date),
    MONTH(date)
    WITH ROLLUP;
-- ------------------------------------------------------------------------------------------------------------
-- total sale month and year wise
select
  month(date) as month,
    year(date) as year,
    round(sum(total),2) as total_sale
from
  invoice
group by
    year(date),
    month(date)
    with rollup;
-- ------------------------------------------------------------------------------------------------------------
-- LOSS ACCORDING TO MONTH AND YEAR
SELECT
  YEAR(exp_date) AS year,
  MONTH(exp_date) AS month,
  ROUND(SUM(cost_price * quantity), 2) AS loss
FROM
  medicine
WHERE
  exp_date < CURRENT_DATE()
GROUP BY
  YEAR(exp_date),
  MONTH(exp_date) WITH ROLLUP;
-- ------------------------------------------------------------------------------------------------------------
```

Embedded Queries

```
def query1(cursor):
    # query to create sale order
    entry = 1
    medicine = {}
    customer = int(input("Enter customer ID: "))
    employee = int(input("Enter employee ID: "))
    while entry == 1:
        med_id = int(input("Enter medicine ID: "))
        medicine[med_id] = int(input("Enter the quantity: "))
        entry = int(input("Want to add more press 1 else 0: "))
    date = datetime.now().strftime('%Y-%m-%d')
    cursor.execute(
        f"insert into sale_order(date, customer, employee) values ('{date}', {customer}, {employee})")
    cursor.execute(
        f"select order_id from sale_order where date='{date}' and customer={customer} and employee={employee}")

    order_id = cursor.fetchall()[0][0]

    for i in medicine.keys():
        cursor.execute(
            f"insert into sale_medicine(medicine_id, quantity, order_id) value ({i}, {medicine[i]}, {order_id})")
        cursor.execute(f"update medicine set quantity=quantity-{medicine[i]} where medicine_id={i}")
    cursor.execute(
        f"select order_id,date,Total from (select order_id, date, round(sum(total)*0.9*1.18, 2) AS TOTAL from (select so.order_id, so.
    data = cursor.fetchall()
    cursor.execute(f"insert into invoice (date, tax, discount, total, `order`) value ('{data[0][1]}', 18.0, 10.0, {data[0][2]}, {data[
    print("Order Created and Invoice generated successfully!")
    cursor.execute(f"select * from invoice where `order`={order_id}")
    print(tabulate(cursor.fetchall(), headers=['Invoice ID', 'Date', 'Tax', 'Discount', 'Total', 'Order ID']))


def query2(cursor):
    # query to update stock of the medicines automatically when a purchase order is delivered to the store
    order_id = int(input("Enter Order ID: "))
    cursor.execute(f"select status from purchase_order where order_id={order_id}")
    status = cursor.fetchall()
    if status[0][0] == 'delivered':
        return "Order has been already delivered! No stock is updated."
    cursor.execute(f"select medicine, quantity from purchase_medicine where order_id={order_id}")
    data = cursor.fetchall()
    cursor.execute(f'update purchase_order set status="delivered" where order_id={order_id}')
    for i in data:
```

```
        cursor.execute(f"update medicine set quantity=quantity+{i[1]} where medicine_id={i[0]}")
    return "Status updated successfully and quantities are added to the stock!"
```

Project Deadline 6

Transaction Details

A - Medicine Table

B - Purchase Order Table

C - Sale Order Table

Note:

R(X) - Read on Table X
W(X) - Write on Table X

Transaction 1 - Employee is creating a sale_order. First, the stock is checked for a particular medicine to ensure enough amount is available to create the order. The stock of the medicine is reduced, and the sale order is created. So this includes the first read on A and then write operations on both A and C.

Transaction 2 - When admin marks a purchase order delivered the stock of the medicines in that order is updated that time. So this includes write operations on A and B.

What is the conflict?

A case can for sure arise when an employee is trying to create a sale order for a medicine, and at the same time, the admin is marking a purchase order delivered for the same medicine. So both of them will write on the same row of the medicine table. Thus conflict arises.

Serialized Schedule

T1	T2
R(A)	
W(A)	
W(C)	
	W(A)
	W(B)

Conflict-Serializable Schedule

T1	T2
R(A)	
W(A)	
	W(A)
W(C)	
	W(B)

T1	T2
R(A)	
W(A)	
W(C)	
	W(A)
	W(B)

Non-Conflict-Serializable Schedule

T1	T2
R(A)	
	W(A)
	W(B)
W(A)	
W(C)	

Another example of Non-Conflict-Serializable Schedule

T1	T2
R(A)	
	R(A)
W(A)	
	W(A)

In the above example, two employees are trying to create a sale order for the same medicine. Thus conflict can arise.

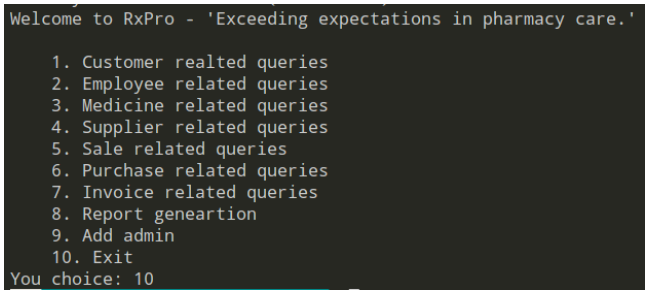
USER GUIDE FOR RxPro

Understanding users: Who are the users of this Database Management System?

Employees and administrators for a pharmacy store can access our Database Management System.

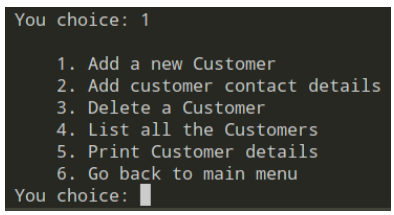
Understanding the usage: How to use this DBMS?

Upon launching the application, you will be asked to log in. Whether you are an employee or an admin, you will be granted access to features. After successful login, you will see a main menu with options numbered 1 to 9. You can enter the number corresponding to the desired option and press Enter to select it.

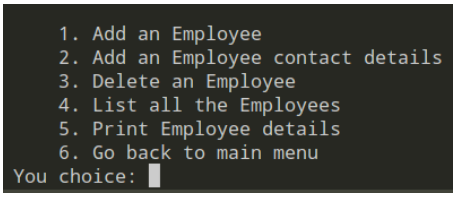


Options:

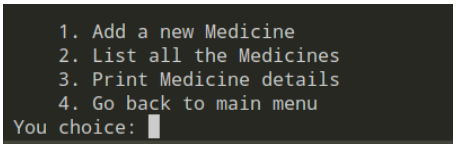
1. Option 1 in the main menu is for customer-related queries. By selecting this option, you can access a sub-menu with options to add a new customer, add customer contact details, delete a customer, list all customers, print customer details, or return to the main menu.



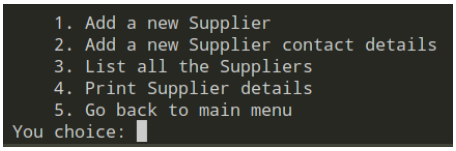
2. Option 2 in the main menu is for employee-related queries. This option allows you to access a sub-menu to add a new employee, add employee contact details, delete an employee, list all employees, print employee details, or return to the main menu.



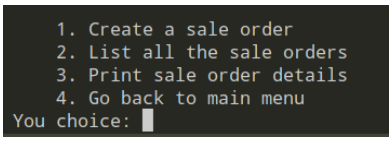
3. Option 3 in the main menu is for medicine-related queries. This option allows you to access a sub-menu to add a new medicine, list all medicines, print medicine details, or return to the main menu.



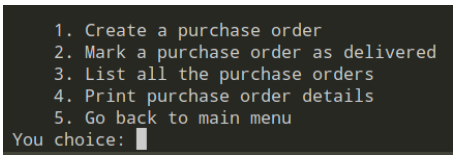
4. Option 4 in the main menu is for supplier-related queries. By selecting this option, you can access a sub-menu with options to add a new supplier, add supplier contact details, list all suppliers, print supplier details, or return to the main menu.



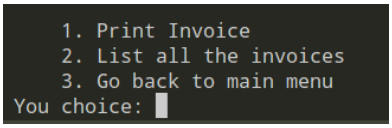
5. Option 5 in the main menu is for sale-related queries. By selecting this option, you can access a sub-menu with options to create a sale order, list all sale orders, print sale order details, or go back to the main menu.



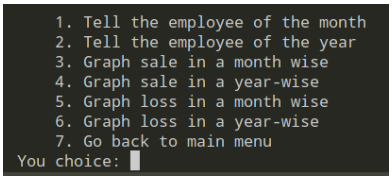
6. Option 6 in the main menu is for purchase-related queries. By selecting this option, you can access a sub-menu with options to create a purchase order, mark a purchase order as delivered, list all purchase orders, print purchase order details, or go back to the main menu.



7. Option 7 in the main menu is for invoice-related queries. By selecting this option, you can access a sub-menu with options to print an invoice, list all invoices, or go back to the main menu.



8. Option 8 in the main menu is for report generation. By selecting this option, you can access a sub-menu with options to tell the employee of the month, tell the employee of the year, graph sales a month-wise, graph sales a year-wise, graph loss a month-wise, graph loss in a year-wise, or go back to the main menu.



9. Option 9 is to add a new admin only an existing admin can add another admin.
10. Option 10 is to save and exit the application.

Things to keep in mind:

- You can navigate through the sub-menus using the same method as in the main menu by entering the number corresponding to the desired option and pressing Enter. Once you have selected an option, the application will execute the corresponding query or generate the requested report.
- It is essential to input valid data and follow the prompts provided by the application to ensure accurate results and proper functioning of the database management system.
- Always make sure to properly exit the application using the provided option to ensure that any changes made are saved adequately before closing the program.
- Since it is a CLI, you must be careful in following the instructions in the menu. Wrong inputs can lead to the program's crash.

This user guide should help you navigate and utilize the primary command-line interface for the pharmacy store database management system. Thank you for using our database management system.

Dependencies

1. mysql-connector-python - pip install mysql-connector-python
2. matplotlib - pip install matplotlib
3. tabulate - pip install tabulate
4. embeddedQueries - provided with the software :)

Contact Information

You can mail us at RxPro@gmail.com for any software functionality queries.