Postmeds

Project Report

April 24, 2023

Scope & Objectives

With this project, we aim to design our own online retail store which includes inventory management and delivery system.In this project we are creating an online retail of medicines and products that can be found in a pharmacy. (For the reference of grader, we plan to make a Database Management System similar to applications like Netmeds,Pharmeasy).

The objective of this project is to record,store & manage the details of various medical products, online orders, shipment and users. We aim to build a project to ease the process of ordering instant medicines with a simplistic UI. This project is an insight into the design and implementation of an online retail Pharmacy Management System.The aim of this project is to develop software for the effective management of a pharmaceutical store with a suitable backend to deal with multiple requests along with proper monitoring.The project provides proper control, storage, security and administration over the online pharmaceutical stores.

Technical Requirements

● **Front-end** HTML,TypeScript/JavaScript,CSS,React

● **Back-end** Django, Python

● **Database** MySQL

Functionalities

1.Procedure

➔ All the supplies of different medicines from different vendors will be recorded and added in the database along with important details like date of purchase, available stock, expiry date and information of the vendors.

➔ The user needs to register his/her basic credentials like Name,Age,Phone Number,Address etc.These details will be added in the database accordingly.

➔ The user can view all the different products under different categories and can add to cart after successfully signing up.

➔ Then the user must choose the mode of payment and redeem any offered code if applicable.

➔ Then the user will get a confirmation message once the order is placed successfully.

➔ The system will find the closest delivery partner and the user will also be able to track the delivery of their order.

➔ Once an order is received the user will have an option to provide feedback or use any return policy in case of damaged, expired or wrong item.

➔ There will also be a premium subscription which will automatically apply 10% discount on cart size value and free delivery charges.The user can order any time of the day with this subscription where the searching radius for closest delivery partner will increase.

2.Stakeholders

- Vendor/Supplier
- Admin (Inventory manager)
- Customer/User
- Delivery partner

3.Requirements

Vendor/Supplier

Sign-up/login

Add product to stock

Add/Remove product supply list

Change Password

Admin

Add/remove product discounts

Change the price of a product

Add product details

Edit payment details

Add/block vendor

Add/block delivery partner

Block existing user

Check Sales of a product

Check Customer reviews

View/Search Customer,Product,Vendor,Deliver partner list

Delivery Partner

Sign-up/login

Accept/decline Delivery requests

Update Delivery status

Return order status

Customer/User

Search/browse Categories=>product

Add/Remove product in Cart

View Cart

Checkout Cart

Buy premium Subscription

Change basic details like phone number,Address etc.

Change Password

View Past Orders

Add/change mode of payment/details

Return/Review product

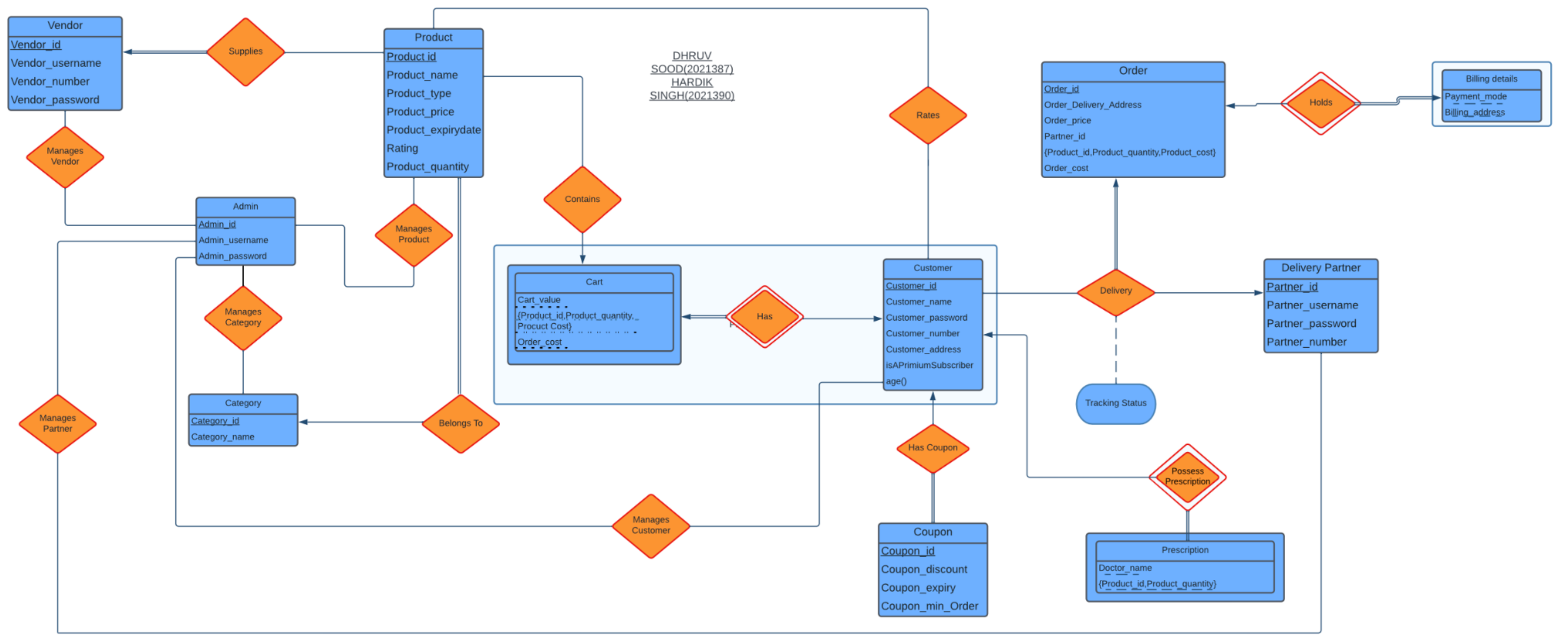
Redeem code

Cancel product

Track Order

Entity Relationship Diagram

https://lucid.app/lucidchart/f7b8dbd5-3f1d-4441-a7e6-62a159007f00/edit?viewport_loc=-184%2C-1376%2C3817%2C1977%2C0_0&invitationId=inv_8b6864ef-a53b-4c10-98b7-08a8b82f4daf



Entities, Attributes & Schemas

1. admin(Admin_id, Admin_username, Admin_password)

Admin_id	INT PRIMARY KEY
Admin_username	VARCHAR(50)
Admin_password	VARCHAR(255)

2. product(Product_id, Product_name, Product_type, Product_price, Product_expirydate, Rating, Product_quantity)

Product_id	INT PRIMARY KEY
Product_name	varchar(50)
Product_type	varchar(20)
Product_price	decimal(10,2)
Product_expirydate	date
Rating	decimal(2,1)
Product_quantity	int

3. user(Customer_id, Customer_name, Customer_password, Customer_number, Customer_address, isPremium_subscriber, Age)

Customer_id	INT PRIMARY KEY
Customer_name	varchar(50)
Customer_password	varchar(255)
Customer_number	varchar(20)
Customer_address	varchar(255)
isPremium_subscriber	tinyint(1)
Age	int

4. vendor(Vendor_id, Vendor_username, Vendor_number, Vendor_password)

Vendor_id	INT AUTO INCREMENT PRIMARY KEY
Vendor_username	varchar(50)
Vendor_number	varchar(20)
Vendor_password	varchar(255)

5. delivery_partner(Partner_id, Partner_username, Partner_password, Partner_number)

Partner_id	INT PRIMARY KEY
Partner_username	varchar(255)
Partner_password	varchar(255)
Partner_number	varchar(255)

6. cart(Customer_id, Product_id, Product_quantity)

Customer_id	INT FOREIGN KEY
Product_id	INT FOREIGN KEY
Product_quantity	INT

7. coupons(Customer_id, Coupon_discount, Coupon_expiry, Coupon_min_Order)

Customer_id	INT PRIMARY KEY FOREIGN KEY
Coupon_discount	decimal(10,2)
Coupon_expiry	date
Coupon_min_Order	decimal(10,2)

Relational Schema

VENDOR(Vendor_id,Vendor_username,Vendor_number,Vendor_password)

PRODUCT(Product_id,Product_name,Product_type,Product_price,Product_expirydate,Rating, Product_quantity)

ADMIN(Admin_id,Admin_username,Admin_password)

CATEGORY(Category_id,Category_name)

CART(Customer_id,Cart_value,{Product_id,Product_quantity,Product_cost})

USER(Customer_id,Customer_name,Customer_password,Customer_number,Customer _address,isPremium _subscriber,Age())

COUPONS(Coupon_id,Coupon_discount,Coupon_expiry,Coupon_miin_Order)

ORDER(Order_id,Order_Delivery_Address,Order_price,Partner_id,Order_cost,{Product _id, Product_quantity,Product_cost})

DELIVERY-PARTNER(Partner_id,Partner_username,Partner_password,Partner_numbe r)

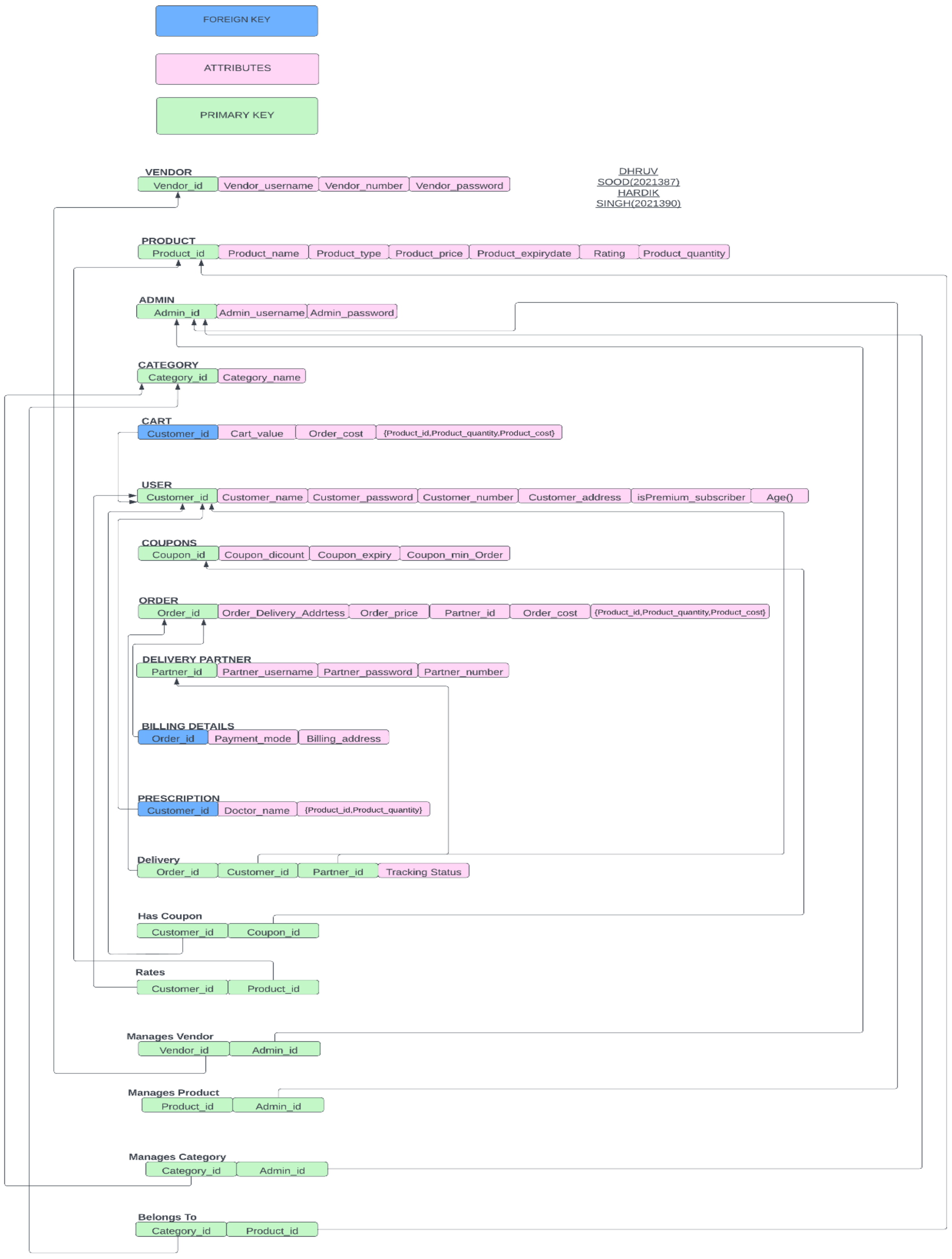
BILLING DETAILS(Order_id,Payment_mode,Billing_address)

PRESCRIPTION(Customer_id,Doctor_name,{Product_id,Product_quantity})

Delivery(Order_id,Customer_id,Partner_id,Tracking_status)

Has Coupon(Customer_id,Coupon_id) Rates(Customer_id,Product_id)

BelongsTo(Category_id,Product_id)


FOREIGN KEY
ATTRIBUTES
PRIMARY KEY
DHRUV
SOOD(2021387)
HARDIK
SINGH(2021390)
VENDOR
Vendor_id
Vendor_username
Vendor_number
Vendor_password
PRODUCT
Product_id
Product_name
Product_type
Product_price
Product_expirydate
Rating
Product_quantity
ADMIN
Admin_id
Admin_username
Admin_password
CATEGORY
Category_id
Category_name
CART
Customer_id
Cart_value
Order_cost
{Product_id,Product_quantity.Product_cost}
USER
Customer_id
Customer_name
Customer_password
Customer_number
Customer_address
isPremium_subscriber
Age()
COUPONS
Coupon_id
Coupon_dicount
Coupon_expiry
Coupon_min_Order
ORDER
Order_id
Order_Delivery_Addrtess
Order_price
Partner_id
Order_cost
{Product_id,Product_quantity,Product_cost}
DELIVERY PARTNER
Partner_id
Partner_username
Partner_password
Partner_number
BILLING DETAILS
Order_id
Payment_mode
Billing_address
PRESCRIPTION
Customer_id
Doctor_name
{Product_id,Product_quantity}
Delivery
Order_id
Customer_id
Partner_id
Tracking Status
Has Coupon
Customer_id
Coupon_id
Rates
Customer_id
Product_id
Manages Vendor
Vendor_id
Admin_id
Manages Product
Product_id
Admin_id
Manages Category
Category_id
Admin_id
Belongs To
Category_id
Product_id

List of SQL Queries

```
Select Product_id, Product_name from product where Product_id  IN

(Select Product_id from belongs_to where Category_id='21' and Product_id NOT IN

(SELECT Product_id from product where Product_name='Amoxicillin' or
Product_name='Paracetamol'));
```

Query 1: "Get Product IDs and names for products in category 21 that are not Amoxicillin or Paracetamol"

This query selects the Product IDs and names from the product table where the Product IDs are in the subquery's result set. The subquery selects Product IDs from the belongs_to table where the Category ID is 21 and the Product ID is not in the subquery's result set. The subquery selects Product IDs from the product table where the Product name is Amoxicillin or Paracetamol.

```
SELECT product_id, SUM(product_quantity) AS Quant

FROM orders

GROUP BY product_id

ORDER BY Quant  DESC

LIMIT 5;
```

Query 2: "Get the top 5 products by order quantity"

This query selects the Product IDs and the sum of their corresponding Product quantities from the orders table. It groups the results by Product ID and sorts the results in descending order based on the sum of Product quantities. The LIMIT clause limits the output to the top 5 products.

```
update product p

set p.product_price=product_price + 20

where p.product_name = 'Eyedrops';
```

Query 3: "Increase the price of Eye Drops by $20"

This query updates the product_price column of the product table for the product with the product_name Eyedrops by adding 20 to its current value.

```
SELECT O.Order_id, U.Customer_name AS "Customer Ordered", O.Order_Delivery_Address,
S.Partner_id

FROM orders O NATURAL JOIN Delivery_Partner S INNER JOIN USER U ON
U.customer_id=O.Order_id

WHERE S.Partner_username="Orpha Altenwerth";
```

Query 4: "Get Order IDs, customer names, delivery addresses, and delivery partner IDs for orders delivered by Orpha Altenwerth"

This query selects the Order IDs, Customer names, and Delivery addresses from the orders table and the Partner IDs from the Delivery_Partner table where the Order IDs match and the Partner_username is "Orpha Altenwerth". It uses the NATURAL JOIN and INNER JOIN clauses to join the orders, delivery_partner, and user tables.

```
UPDATE Belongs_To
```

```
SET Category_id = 21

WHERE Belongs_To.ProductID = 53;
```

Query 5: "Change the category of the product with ID 53 to category 21"

This query updates the Category_id column of the Belongs_To table to 21 for the product with ID 53.

```
SELECT AVG(Cart_value) AS AverageOrderVal FROM cart;
```

Query 6: "Get the average value of a cart"

This query selects the average value of the Cart_value column from the cart table.

```
select count(product_id), category_id

from belongs_to

group by category_id;
```

Query 7: "Get the number of products in each category"

This query selects the count of Product IDs and the Category IDs from the Belongs_To table and groups the results by Category ID.

```
delete from coupons where coupons.Coupon_expiry < CURRENT_DATE;
```

Query 8: "Delete expired coupons"

This query deletes records from the coupons table where the Coupon_expiry date is less than the current date.

```
insert into rates(Customer_id, Product_id) values (6,13);
```

Query 9: "Add a rating for a product purchased by a customer"

This query inserts a record into the rates table with the Customer ID and Product ID specified.

```
SELECT P.Product_name, P.Product_price
FROM PRODUCT P
JOIN belongs_to B ON P.Product_id = B.Product_id
WHERE B.Category_id = '66';
```

Query 10: "Get product names and prices for products in category 66"
This query selects the Product names and prices from the Product table where the Product IDs are in the Belongs_To table and the Category ID is 66.

```
SELECT SUM(O.Order_price)

FROM orders O, delivery D where O.order_id in

(Select D.Order_id where D.Partner_id = '10' and Customer_id is NOT NULL);
```

Query 11: "Get the total price of orders delivered by a specific delivery partner"

This query selects the sum of the Order prices from the orders table where the Order IDs match those in the delivery table where the Partner ID is 10 and the Customer ID is not NULL.

```
SELECT o.order_id, user.customer_id, dp.Partner_username

FROM Orders o, user

CROSS JOIN delivery_partner dp;
```

Query 12: "Get order IDs, customer IDs, and delivery partner usernames for all orders"

This query selects the Order IDs from the orders table, Customer IDs from the user table, and Partner usernames from the delivery_partner table. It performs a CROSS JOIN between the orders and user tables to produce all possible combinations of Order IDs and Customer IDs, then selects the Partner usernames for each combination.

Triggers Implemented + OLAP + Embedded Queries

```
print('''

   ____             __  __  ___          __
  / __ \____  _____/ /_/  |/  /___  ____/ /_____
 / /_/ / __ \/ ___/ __/ /|_/ / __ \/ __  / ___/
/ ____/ /_/ (__  ) /_/ /  / / /_/ / /_/ (__  )
/_/    \____/____/\__/_/  /_/\____/\__,_/____/




''')


print(f"""

   ========================================

           Successfully connected

                to SQL Server

   ========================================

         """)


import mysql.connector
```

```
sql_connector=mysql.connector.connect(

   host="localhost",

   user="root",

   passwd="12345"



)



cursor=sql_connector.cursor()

cursor.execute("USE dbms")



# TRIGGERED QUERIES

# his code creates a trigger that deletes a corresponding row from the belongs_to
table when a row in the product table is updated

# and its Product_quantity value is 0.



cursor.execute('''

CREATE TRIGGER delete_reln

AFTER UPDATE ON product

FOR EACH ROW

BEGIN

 IF NEW.Product_quantity = 0 THEN

   DELETE FROM belongs_to WHERE product_id = NEW.Product_id;
 END IF;

END;

''')


sql_connector.commit()

cursor=sql_connector.cursor()


#  This code creates a trigger that adds a coupon with a discount of 10 to the COUPONS
table

# after a new row is inserted into the user table, using the Customer_id of the
inserted row.


cursor.execute('''

CREATE TRIGGER new_user_bonus

AFTER INSERT ON user

FOR EACH ROW

BEGIN

 DECLARE phone_count INT;

 SET phone_count = (SELECT COUNT(*) FROM user WHERE Customer_number =
NEW.Customer_number);


 IF phone_count = 1 THEN
```

```
   INSERT INTO COUPONS (Customer_id,
Coupon_discount,Coupon_expiry,Coupon_min_Order) VALUES (NEW.Customer_id, 10,
date(CURDATE() + 100), 1000);

 END IF;

 END;''')



sql_connector.commit()



print("Choose an option:")

print("1. Exit")

print("2. Trigger1 Verification")

print("3. Trigger2 Verification")

print("4. Embedded Query 1")

print("5. Embedded Query 2")

print("6. OLAP Query 1")

print("7. OLAP Query 2")

print("8. OLAP Query 3")

print("9. OLAP Query 4")

print("10. OLAP Query 5")

print("11. OLAP Query 6")

while(1):

   query_input=int(input("Enter your choice"))
```

```
if(query_input==1):

    break

if(query_input==2):

        print("BEFORE:\n")

        query = '''

        select * from coupons;

        '''

        cursor.execute(query)

        result_trig33=cursor.fetchall()

        for i in result_trig33:

            print(i)



        q2 = f'''

        INSERT INTO user (`Customer_id`, `Customer_name`, `Customer_password`,
`Customer_number`, `Customer_address`, `isPremium_subscriber`, `Age`) VALUES ({5},
'Dr. Candelario  ', 'dolr', '9812219362', '46334 Buckridge Lodge Suite 189\nWest
Russel, HI 90933', {0}, {33});

        '''

        cursor.execute(q2)

        sql_connector.commit()
        print("AFTER:\n")

        query = '''

        select * from coupons;

        '''

        cursor.execute(query)

        result_trig33=cursor.fetchall()

        for i in result_trig33:

            print(i)


if(query_input==3):

        print("BEFORE:\n")

        query = '''

        select * from belongs_to;

        '''

        cursor.execute(query)

        result_trig33=cursor.fetchall()

        for i in result_trig33:

            print(i)

        # sql_connector.commit()
```

```
          q2 = f'''

          update product set Product_quantity={0} where Product_id={94};

          '''

          cursor.execute(q2)

          sql_connector.commit()



          print("AFTER:\n")

          query = '''

          select * from belongs_to;

          '''

          cursor.execute(query)

          result_trig33=cursor.fetchall()

          for i in result_trig33:

              print(i)



# Embedded Queries



if(query_input==4):

    embdd1="SELECT * FROM Delivery_partner"

    cursor.execute(embdd1)

    result_emb1=cursor.fetchall()
```

```
        for i in result_emb1:

            print(i)

        # sql_connector.commit()

    if(query_input==5):

        embdd2='''

        update product p

        set p.product_price=product_price + 20

        where p.product_name = 'Eyedrops';

        '''

        cursor.execute(embdd2)

        result_emb2=cursor.fetchall()

        for i in result_emb2:

            print(i)

        sql_connector.commit()

    # OLAP queries

    if(query_input==6):

        olap1='''

        SELECT

            Delivery_partner.Partner_id,

            COUNT(*) AS total_deliveries,

            SUM(CASE WHEN Tracking_Status = 'Delivered' THEN 1 ELSE 0 END) AS
deliveries_completed,
            SUM(CASE WHEN Tracking_Status = 'In Transit' THEN 1 ELSE 0 END) AS
 deliveries_in_transit,

            SUM(CASE WHEN Tracking_Status = 'Pending'  THEN 1 ELSE 0 END) AS
 deliveries_cancelled

        FROM

            Delivery

            JOIN orders ON Delivery.Order_id = orders.Order_id

            JOIN Delivery_partner ON Delivery.Partner_id = Delivery_partner.Partner_id

        GROUP BY

            Delivery_partner.Partner_id

        WITH ROLLUP;

        '''

        cursor.execute(olap1)

        res1=cursor.fetchall()

        for i in res1:

            print(i)

        sql_connector.commit()

    if(query_input==7):

        olap2='''SELECT p.Product_type, SUM(o.Order_price) AS total_sales

        FROM

            PRODUCT p

            JOIN ORDERS o ON p.Product_id = o.Product_id
```

```
    GROUP BY

        p.Product_type

    WITH ROLLUP;

    '''

    cursor.execute(olap2)

    res2=cursor.fetchall()

    for i in res2:

        print(i)

    sql_connector.commit()

if(query_input==8):

    olap3='''

    SELECT Payment_mode, COUNT(*) AS Total_customers

    FROM BILLING_DETAILS

    GROUP BY Payment_mode

    WITH ROLLUP;

    '''

    cursor.execute(olap3)

    res3=cursor.fetchall()

    for i in res3:

        print(i)

    sql_connector.commit()
```

```
if(query_input==9):

    olap4='''

    SELECT Product_type, AVG(Rating) AS Average_rating

    FROM PRODUCT

    GROUP BY Product_type;

    '''

    cursor.execute(olap4)

    res4=cursor.fetchall()

    for i in res4:

        print(i)

    sql_connector.commit()

if(query_input==10):

    olap5='''

    SELECT Customer_name,Customer_number,Customer_address,COUNT(*) AS total_users

    FROM USER

    GROUP BY Customer_name,Customer_number,Customer_address;

    '''

    cursor.execute(olap5)

    res5=cursor.fetchall()

    for i in res5:

        print(i)
        sql_connector.commit()

    if(query_input==11):

        olap6='''

        SELECT product_id, SUM(product_quantity) AS Quant

        FROM orders

        GROUP BY product_id

        ORDER BY Quant DESC;'''

        cursor.execute(olap6)

        res6=cursor.fetchall()

        for i in res6:

            print(i)

        sql_connector.commit()
```

1) **<u>Conflicting Transactions</u>:**

   a) Transaction T1:

      i)Reading table product (read(A)):

      ```
      SELECT * FROM product WHERE product_ID = 1;
      ```

      ii)Writing table product (write(A)):

      ```
      UPDATE product SET Product_quantity = Product_quantity - 1
      WHERE product_ID = 1;
      ```

      iii)Reading table user (read(B)):

      ```
      SELECT Customer_id FROM user
      WHERE Customer_id = 1;
      ```

      iv)Writing table user relational table(write(B)):

      ```
      UPDATE user SET Customer_name = ‘ABC’ WHERE
      Customer_id = 1;
      ```

   b) Transaction T2:

      i)Reading table product (read(A)):

      ```
      SELECT * FROM product WHERE product_ID = 1;
      ```

      ii)Writing table product (write(A)):

      ```
      UPDATE product SET Product_quantity = Product_quantity - 1
      WHERE product_ID = 1;
      ```

      iii)Reading table user (read(B)):

      ```
      SELECT Customer_id FROM user
      WHERE Customer_id = 1;
      ```

      iv) Writing table user relational table(write(B)):

      ```
      UPDATE user SET Customer_name = ‘ABC’ WHERE
      Customer_id = 1;
      ```

The two transactions are conflicting because they both involve updating the same row in the 'product' table (product_ID = 1) and the same row in the ‘user’ table (Customer_id = 1).

In transaction T1, after reading the 'product' table, it decreases the Product_quantity of the product_ID = 1 by 1. Then, it reads the ‘user’ table and updates the ‘Customer_name’ to “ABC”.

In transaction T2, after reading the 'product' table, it also decreases the Product_quantity of the product_ID = 1 by 1. Then, it reads the ‘user’ table and updates the ‘Customer_name’ to “ABC”.

SERIAL

T1	T2
read(A)	
write(A)	
write(B)	
Commit T1	
	read(A)
	write(A)
	write(B)
	Commit T2

Conflicting serializable schedule by interchanging non conflicting queries, those being read(A), write(A) with write(B)

CONFLICT SERIALIZABLE

T1	T2
read(A)	
write(A)	
	read(A)
	write(A)
write(B)	
Commit T1	
	write(B)
	Commit T2

Non conflicting serializable schedule by interchanging conflicting queries, those being write(B) of T1 with write(B) with T2

NON CONFLICT SERIALIZABLE

T1	T2
read(A)	
write(A)	
	read(A)
	write(A)
	write(B)
	Commit T2
write(B)	
Commit T1	

2) Non Conflicting Transactions:

The following are 4 Transactions that amongst each other would not produce any conflict, given that they either write different locations in memories or only read from identical memory locations.

a) Transaction T1:
   Select product from the cart of user(read(A))
   select * from Cart where product_ID = 1 and Customer_id = 1
b) Transaction T2:
   Select product quantity from the cart of user(read(A))
   select Product_quantity from Cart where product_ID = 1 and Customer_id = 1

c) Transaction T3:
   Update name of user(write(B))
   update user set Customer_name = “ABC” where Customer_id = 1
d) Transaction T4:
   Insert delivery details for order(write(C))
   insert into delivery values(1, 1, 1, ‘Pending’)

The transactions are non-conflicting because they don't involve any shared or overlapping data items that could potentially cause conflicts if accessed or modified simultaneously.

In the case of T1 and T2, they are both accessing the same "Cart" table but retrieving different information (product and product quantity), so there is no overlap in the data that they are accessing. Therefore, both transactions can be executed concurrently without any issues.

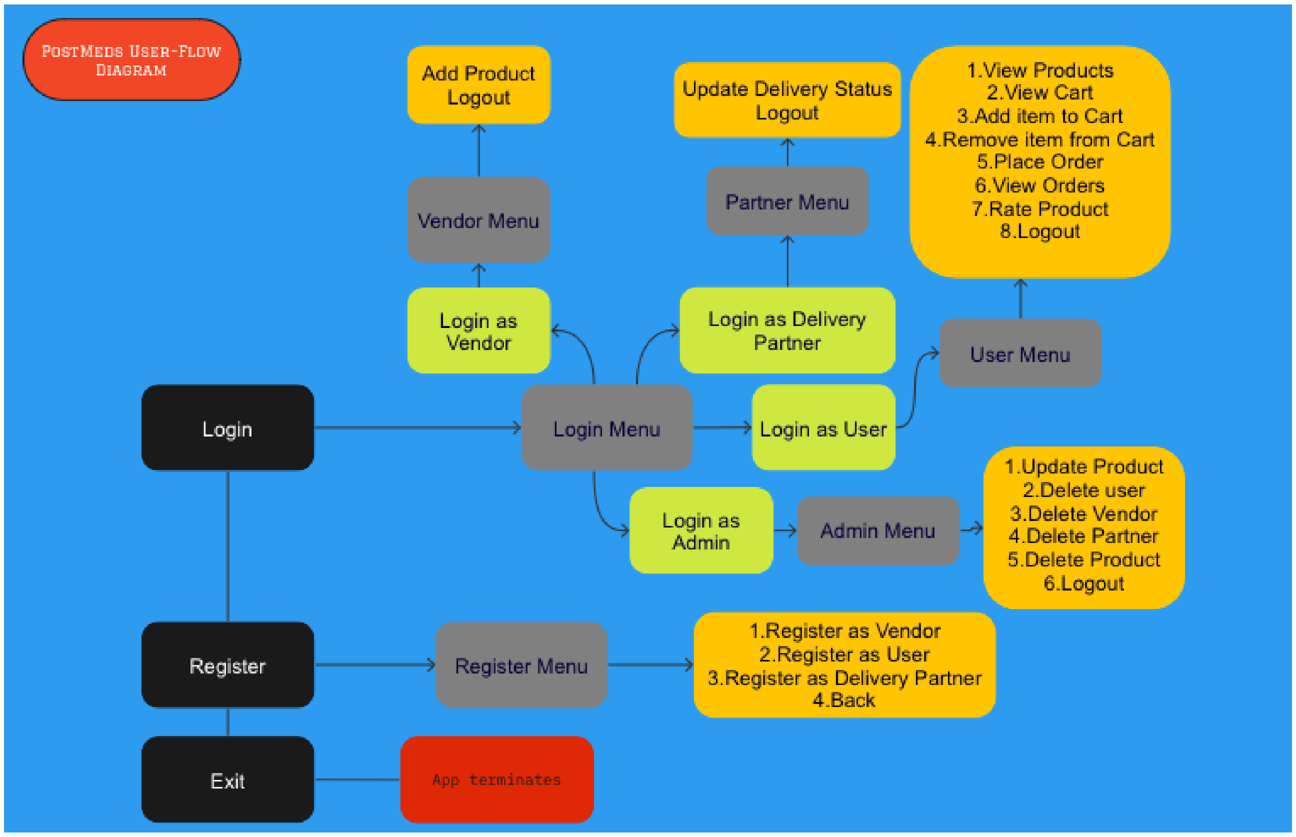
T3 updates the "User" table, which is completely separate from the "Cart" table that T1 and T2 are using. Since T3 doesn't access or modify any data that is relevant to T1 or T2, there is no chance of conflict between these transactions.

Finally, T4 inserts data into the "Delivery" table, which is a completely different table from the "Cart" and "User" tables. This means that there is no overlap in the data that T4 is modifying compared to the data that T1, T2, and T3 are accessing or modifying. Therefore, all four transactions can be executed concurrently without any conflicts.

In general, to ensure non-conflicting transactions, it's important to make sure that each transaction accesses and modifies only the data that it needs and doesn't interfere with the data that other transactions are using. Additionally, proper transaction isolation levels and concurrency control mechanisms can be used to further prevent conflicts and ensure data consistency.

User Flow Diagram

https://miro.com/app/board/uXjVMPibEZw=/?share_link_id=14466495556



User Guide

1. Product Catalogue:

Our users will be able to browse all the products available for purchase on your platform. This could include medicines, medical devices, supplements, and other health-related products. Each product will have a detailed description, including its name, price, dosage, ingredients, and any relevant warnings or instructions. Users can also consider product reviews or ratings from other customers.

2. Add to Cart:

Once a user finds a product they want to buy, they can add it to their cart by specifying the desired quantity. The cart will keep track of all the items the user has selected to purchase.

3. Remove from Cart:

If a user changes their mind or wants to update their order, they can easily remove items from their cart before proceeding to checkout. This feature will ensure that users have complete control over their purchases and can make changes as needed.

4. Place Order:

Once the user has added all the items they wish to purchase to their cart, they can proceed to the checkout to place their order. During the checkout process, they will be asked to provide their delivery address, contact information, and any other relevant details. If the user has an available coupon and the order meets the minimum order value, they can apply the coupon during checkout. You may also consider providing users with multiple payment options, such as credit card, debit card, or online banking.

5. Order History:

Our platform will allow users to view their order history, which will include all the orders they have placed on your website. This feature will help users keep track of their purchases and provide them with an easy way to reorder items they have previously bought. Users will also be able to see the status of each order, including whether it has been shipped, delivered, or cancelled.

6. Product Ratings:

Users will be able to rate the products they have purchased, providing valuable feedback for both you and other potential customers. Users can rate products on a scale of one to ten stars. Providing a credible source for other users before they consider buying a product.

Overall, these features will provide our users with a seamless and convenient way to purchase and receive essential health-related products.

7. Partner Menu

The delivery Partner can login in and update the delivery status of the Order as Pending In transit or Delivered.

8. Admin Menu

The admin has overall access over the stakeholders and has rights to remove any such.The admin can also update the product details such as its price quantity and rating.

The product catalogue feature ensures consistency by providing users with accurate and detailed information about each product. This includes the product name, price, dosage, ingredients, and any relevant warnings or instructions. The descriptions are standardised, which ensures that users can easily compare products and make informed purchasing decisions.

The add to cart and remove from cart features ensure data integrity by accurately tracking the items that the user has selected to purchase. Users can easily remove items from their cart if they change their mind or want to update their order, which ensures that the system remains up-to-date with the user's intent.

The place order feature ensures data integrity by requiring users to provide accurate information during the checkout process. This includes their delivery address, contact information, and any other relevant details. By verifying this information, the system can ensure that orders are delivered to the correct location and that users can be contacted if there are any issues with their order.

The order history feature ensures consistency by providing users with a comprehensive record of all the orders they have placed on the platform. Users can easily access this information and use it to reorder items they have previously bought. The feature also ensures data integrity by accurately tracking the status of each order and providing users with up-to-date information about their purchase.

The product ratings feature ensures consistency by providing users with a standardised rating system that is easy to understand. Users can rate products on a scale of one to ten stars, which ensures that the ratings are consistent across products. The feature also ensures data integrity by accurately tracking the ratings and ensuring that users can access this information when making purchasing decisions.

Overall, these features demonstrate a commitment to consistency, integrity, and other important parameters by providing users with accurate information, tracking user intent, and verifying user information. By implementing these features, the platform can ensure a seamless and convenient purchasing experience for users, while also providing valuable feedback for the platform and other potential customers.