**Conflicting and Non-Conflicting**

**Database Transactions**

Transaction Pair 1 (Conflicting)

Consider the case where an admin updates product quantity to stock (i.e., increases the quantity), and simultaneously, a customer buys the same product (i.e. decreases the quantity).

| **Transaction-1** (T1) | **Transaction-2** (T2) |
| --- | --- |
| Read(Products.Quantity)  Products.Quantity = Products.Quantity + 10  Write(Products.Quantity)  Commit | Read(Products.Quantity)  Products.Quantity = Products.Quantity - 7  Write(Products.Quantity)  Write(Orders)  Commit |

Conflict-Serializable Schedule

| **Transaction-1** (T1) | **Transaction-2** (T2) |
| --- | --- |
| Read(Products.Quantity)  Products.Quantity = Products.Quantity + 10  Write(Products.Quantity) | Read(Products.Quantity)  Products.Quantity = Products.Quantity - 7  Write(Products.Quantity)  Write(Orders) |

Conflict-Serializable Schedule with Locks

| **Transaction-1** (T1) | **Transaction-2** (T2) |
| --- | --- |
| Lock-X(Products)  Read(Products.Quantity)  Products.Quantity = Products.Quantity + 10  Write(Products.Quantity)  Unlock(Products) | Lock-X(Products)  Read(Products.Quantity)  Products.Quantity = Products.Quantity - 7  Write(Products.Quantity)  Unlock(Products)  Lock-X(Orders)  Write(Orders)  Unlock(Orders) |

In this schedule, T1 and T2 are two conflicting transactions. T1 increases the quantity in stock of Product 3 by 10 units, while T2 decreases it by 7 units and simultaneously places an order for 7 units of the product.

Both transactions start by acquiring locks on the Products table to ensure exclusive access. T1 then reads the current quantity in stock, updates it, and releases the lock. Similarly, T2 reads the quantity, updates it, and releases the lock. After updating the quantity, T2 acquires a lock on the Orders table, inserts the order, and then releases the lock.

This schedule ensures that the conflicting operations on the Products table are executed serially, preventing any inconsistency in the quantity update and order placement.

Transaction Pair 2:(Non Conflicting)

Consider the case where the product quantity is updated only when the its quantity is more than zero and simultaneously an insertion in the table Orders is taking place

Non-Conflict-Serializable Schedule

| **Transaction-1** (T1) | **Transaction-2** (T2) |
| --- | --- |
| Read(Products.Quantity)    Products.Quantity := CASE  WHEN Products.Quantity > 0 THEN  Products.Quantity - 1  ELSE Products.Quantity  END    Write(Products.Quantity) | Read(Products.Quantity)  Insert into Orders (user3, 40.00, 5, 1, CURDATE())  WHERE Products.Quantity > 0  Write(Orders) |

Non-Conflict-Serializable Schedule with Locks

| Transaction-1 (T1) | Transaction-2 (T2) |
| --- | --- |
| Lock-X(Products)    Read(Products.Quantity)    Products.Quantity := CASE  WHEN Products.Quantity > 0 THEN  Products.Quantity - 1  ELSE Products.Quantity  END    Write(Products.Quantity)    Unlock(Products) | Lock-X(Products)  Read(Products.Quantity)  Insert into Orders (user3, 40.00, 5, 1, CURDATE())  WHERE Products.Quantity > 0  Unlock(Products) |

T1 attempts to decrease the quantity in stock of Product 5 by 1 unit, ensuring that the quantity doesn't go below 0. T2 tries to insert an order for Product 5 with a quantity of 1, but only if the quantity in stock is greater than 0.

Both transactions acquire exclusive locks on the Products table to ensure exclusive access. T1 reads and updates the quantity in stock, while T2 reads the quantity and attempts to insert an order if the quantity in stock is available.

This schedule is non-conflict-serializable because T2's action depends on the outcome of T1's update, but there's no direct conflict between the operations of T1 and T2.

Transaction Pair 3 (Conflicting)

Consider the case to retrieve the bill amount after order placement and simultaneously a customer checks the quantity available in stock for a specific product

| **Transaction-1** (T1) | **Transaction-2** (T2) |
| --- | --- |
| Read(Billing.bill\_amount)    Write(Billing.bill\_amount)    COMMIT | Read(Products.Quantity)  Write(Products.Quantity)  DO SLEEP(5)  COMMIT |

Conflict-Serializable Schedule

| **Transaction-1** (T1) | **Transaction-2** (T2) |
| --- | --- |
| Read(Billing.bill\_amount)    Write(Billing.bill\_amount)    COMMIT | Read(Products.Quantity)  Write(Products.Quantity)  DO SLEEP(5)  COMMIT |

Conflict-Serializable Schedule With Locks

| **Transaction-1** (T1) | **Transaction-2** (T2) |
| --- | --- |
| LOCK-X(Billing)  Read(Billing.bill\_amount)    Write(Billing.bill\_amount)  Unlock(Billing)    COMMIT | LOCK-X(Products)  Read(Products.Quantity)  Write(Products.Quantity)  Unlock(Products)  DO SLEEP(5)  COMMIT |

In this conflict-serializable schedule with locks, both transactions are executed sequentially with exclusive locks.

Transaction T1 involves processing the payment by selecting the bill amount from the Billing table for the given order ID. It acquires an exclusive lock on the Billing table to ensure exclusive access during the transaction.

Transaction T2 involves checking the quantity available in stock for the product associated with the given order ID from the Products table. It acquires an exclusive lock on the Products table to ensure exclusive access during the transaction.

By using locks, conflicts between concurrent transactions are resolved, ensuring consistency in the database operations.

Online Store CLI User Guide

The CLI provides a simple and interactive interface for users to navigate through different functionalities of the Online Store application.

Users need to log in with appropriate credentials to access specific features based on their roles (admin, user, or seller).

1. Main Menu:

Upon launching the CLI, users are presented with the Main Menu.

Options:

● Admin LogIn: Allows administrative access to the system.

● User LogIn: Permits registered users to log in and interact with the store.

● User SignUp: Enables new users to create accounts.

● Seller LogIn: Grants access for sellers to manage their products.

● Exit: Closes the CLI.

2. Admin Menu:

Upon successful login as an administrator, users access the Admin Menu.

Options:

● View Quarterly Sales of Each Category: Provides insights into sales performance across different product categories.

● View Top 5 Customers: Displays the top five customers based on their purchase amounts.

● Data of Items in the Inventory: Presents information regarding the available inventory.

● Add Category: Allows administrators to add new product categories.

● View All Categories: Displays a list of all existing product categories.

● View All Sellers: Shows a list of all registered sellers.

● Log Out: Logs the administrator out of the system.

3. User Menu:

Upon successful user login, users access the User Menu.

Options:

● View Categories: Allows users to browse available product categories.

● View Cart: Displays the contents of the user's shopping cart.

● Proceed To Checkout: Initiates the checkout process for items in the cart.

● Track Order details: Provides information on the status and details of placed orders.

● Return details: Facilitates the process of initiating product returns.

● Exit to Main Menu: Returns the user to the Main Menu.

4. Seller Menu:

Upon successful login as a seller, users access the Seller Menu.

Options:

● View Seller Details: Displays information related to the seller's products, sales, and earnings.

● Log out: Logs the seller out of the system.

Features Explained

Admin Features:

● View Quarterly Sales of Each Category: Provides a comprehensive overview of sales performance categorized by product categories.

● View Top 5 Customers: Presents a list of the top five customers based on their purchase amounts.

● Data of Items in the Inventory: Offers detailed information regarding the available inventory, including product names, quantities, and prices.

● Add Category: Allows administrators to expand the product catalog by adding new categories.

● View All Categories: Displays a list of all existing product categories within the system.

● View All Sellers: Provides a list of all registered sellers along with their respective details.

User Features

View Categories:

Allows users to explore available product categories and browse items.

● View Cart: Displays the contents of the user's shopping cart, including product names, quantities, and prices.

● Proceed To Checkout: Initiates the checkout process, enabling users to complete their purchases and make payments.

● Track Order details: Provides real-time updates on the status and details of placed orders, enhancing transparency and convenience for users.

● Return details: Facilitates the process of initiating product returns, ensuring a seamless experience for users in case of any issues or concerns.

Seller Features

View Seller Details:

● Provides sellers with comprehensive insights into their products, sales, earnings, and payment details.

● Log out:Enables sellers to securely log out of the system, ensuring data privacy and security.

This guide outlines the functionalities and capabilities of the Online Store CLI, catering to different user roles and providing a seamless shopping experience.

**Command Line Interface**

import mysql.connector

from datetime import datetime, timedelta

from tabulate import tabulate

import time

# Establish cursorection

mydb = mysql.connector.connect(

host='localhost',

user='root',

password='Karora@88',

database='online\_store\_amazon' # Replace 'your\_database\_name' with your actual database name

)

# Check if cursorection is successful

if mydb.is\_connected():

print("connector to MySQL database")

# Create cursor object

cursor = mydb.cursor()

login\_attempts = 0

max\_attempts = 3

def acquire\_lock(lock\_name, timeout):

cursor.execute("SELECT GET\_LOCK(%s, %s)", (lock\_name, timeout))

lock\_status = cursor.fetchone()[0]

if lock\_status == 1:

print(f"Lock '{lock\_name}' acquired successfully.")

else:

print(f"Failed to acquire lock '{lock\_name}'.")

def release\_lock(lock\_name):

cursor.execute("SELECT RELEASE\_LOCK(%s)", (lock\_name,))

lock\_status = cursor.fetchone()[0]

if lock\_status == 1:

print(f"Lock '{lock\_name}' released successfully.")

else:

print(f"Failed to release lock '{lock\_name}'.")

def update\_product\_stock(product\_id, added\_stock):

cursor.execute("START TRANSACTION;")

try:

cursor.execute("SELECT quantity\_in\_stock FROM Products WHERE product\_ID = %s FOR UPDATE;", (product\_id,))

stock = cursor.fetchone()[0]

# Simulate a delay to create a race condition

time.sleep(5)

new\_stock = stock + added\_stock

cursor.execute("UPDATE Products SET quantity\_in\_stock = %s WHERE product\_ID = %s;", (new\_stock, product\_id))

cursor.execute("COMMIT;")

print("Product stock updated successfully.")

except Exception as e:

print("Failed to update product stock:", e)

cursor.execute("ROLLBACK;")

def update\_product\_price():

try:

# Call the stored procedure

cursor.callproc('UpdateProductPrice')

# Fetch the results

for result in cursor.stored\_results():

for row in result.fetchall():

# Check if the first element of the row contains 'Price updated successfully.'

if row[0] == 'Price updated successfully.':

print(row[0], "New Price:", row[1])

else:

print(row[0]) # Print error message

# Commit the transaction

mydb.commit()

except mysql.connector.Error as e:

# Rollback the transaction if an error occurs

print("Update failed, transaction rolled back:", e)

cursor.rollback()

# Example usage of lock and unlock transactions

lock\_name = 'product\_stock'

timeout = 10 # Timeout in seconds

print("\n------------------------")

print("Updating Product Stock")

print("------------------------")

try:

# Acquire the lock

acquire\_lock(lock\_name, timeout)

# Update the stock of product 1

product\_id = 1

added\_stock = 5

update\_product\_stock(product\_id, added\_stock)

finally:

# Release the lock

release\_lock(lock\_name)

print("\n------------------------")

print("Updating Product Price")

print("------------------------")

# lock and unlock transactions

lock\_name = 'product\_price'

timeout = 5 # Timeout in seconds

try:

# Acquire the lock

acquire\_lock(lock\_name, timeout)

update\_product\_price()

finally:

# Release the lock

release\_lock(lock\_name)

def print\_welcome\_message():

print("\033[1m")

print("╔══════════════════════════════════════════╗")

print("║ Welcome to Online Store ║")

print("║ \* Amazon \* ║")

print("╚══════════════════════════════════════════╝")

print("\033[0m")

def print\_main\_menu():

print("\033[1m")

print("╔═══════════════════════════════════════════════════════╗")

print("║ Welcome to Our Store ║")

print("╠═══════════════════════════════════════════════════════╣")

print("║ ║")

print("║ Main Menu ║")

print("║ ║")

print("║ 1. Admin LogIn ║")

print("║ 2. User LogIn ║")

print("║ 3. User SignUp ║")

print("║ 4. Seller LogIn ║")

print("║ 5. Exit ║")

print("║ ║")

print("╚═══════════════════════════════════════════════════════╝")

print("\033[0m")

def print\_admin\_menu(username):

print("\033[1m")

print("╔════════════════════════════════════════════════╗")

print("║ Welcome to the Admin Panel ║")

print("╠════════════════════════════════════════════════╣")

print("║ ║")

print("║ Hello, ", username.center(13), " ║")

print("║ ║")

print("║ Please choose an option from the menu below: ║")

print("║ ║")

print("║ 1. View Quarterly Sales of Each Category ║")

print("║ 2. View Top 5 Customers ║")

print("║ 3. Data of Items in the Inventory ║")

print("║ 4. Add Category ║")

print("║ 5. View All Categories ║")

print("║ 6. View All Sellers ║")

print("║ 7. Log Out ║")

print("║ ║")

print("╚════════════════════════════════════════════════╝")

print("\033[0m")

def print\_user\_menu(username):

print("\033[1m")

print(f"╔══════════════════════════════════════════════╗")

print(f"║ Welcome {username.center(11)} ║")

print(f"╠══════════════════════════════════════════════ ╣")

print("║ ║")

print("║ User Menu Options: ║")

print("║ ║")

print("║ 1. View Categories ║")

print("║ 2. View Cart ║")

print("║ 3. Proceed To Checkout ║")

print("║ 4. Track Order details ║")

print("║ 5. Return details ║")

print("║ 6. Exit to Main Menu ║")

print("║ ║")

print("╚══════════════════════════════════════════════╝")

print("\033[0m")

def print\_seller\_menu(seller\_name):

print("\033[1m")

print(f"╔════════════════════════════════════════════════════╗")

print(f"║ Welcome {seller\_name.center(15)} ║")

print(f"╠════════════════════════════════════════════════════╣")

print("║ ║")

print("║ Seller Menu Options: ║")

print("║ ║")

print("║ 1. View Seller Details ║")

print("║ 2. Log out ║")

print("║ ║")

print("╚════════════════════════════════════════════════════╝")

print("\033[0m")

while(True):

print\_welcome\_message()

print("---------------------------------------------")

print\_main\_menu()

print("Please choose a number from the menu to proceed: ")

#SHOULD WE MAKE A DISTRIBUTOR SIGN-UP?

input\_landing\_page = int(input("Enter the number: "))

#ADMIN LOGIN

if (input\_landing\_page == 1):

print("---------------------------------------------")

count = 0

valid\_admin = 0

while(count<3 and valid\_admin == 0):

query\_auth\_admin = """Select username,password from Admin"""

username = input("Enter your username: ")

password = input("Enter your password: ")

cursor.execute(query\_auth\_admin)

for row in cursor.fetchall():

if (username) == row[0] and (password) == row[1]:

# store = row

# print(row)

valid\_admin = 1

print("Authenticated\n")

break

if valid\_admin == 0:

print("Invalid Username or password\n")

count+=1

print(f"{3-count} tries remaining\n")

while (valid\_admin):

print\_admin\_menu(username)

input\_admin = int(input("Enter the number: "))

if (input\_admin == 1):

query\_report = """SELECT

Category.category\_name AS Category,

SUM(Orders.order\_amount) AS Total\_Sales\_Amount,

SUM(Orders.quantity) AS Total\_Quantity\_Sold

FROM

Category

JOIN Products ON Category.category\_ID = Products.category\_ID

JOIN Orders ON Products.product\_ID = Orders.product\_ID

JOIN Billing ON Orders.order\_ID = Billing.order\_ID

WHERE

Orders.date\_order\_placed >= DATE\_SUB(NOW(), INTERVAL 3 MONTH)

GROUP BY

Category.category\_name WITH ROLLUP

HAVING

Category IS NOT NULL"""

cursor.execute(query\_report)

print("---------------------------------------------")

for row in cursor.fetchall():

print(f"Category: {row[0]}\nSales from the Category (Rs.): {row[1]}\nQuantity sold (units): {row[2]}")

print("---------------------------------------------")

elif (input\_admin == 2):

query\_top\_5\_cust = """SELECT

CASE

WHEN GROUPING(customer\_username) = 1 THEN 'All Customers'

ELSE customer\_username

END AS Customer,

SUM(order\_amount) AS TotalAmount

FROM

Orders

GROUP BY customer\_username with

ROLLUP

HAVING

customer\_username IS NOT NULL

ORDER BY

TotalAmount DESC

LIMIT

5;

"""

cursor.execute(query\_top\_5\_cust)

print("---------------------------------------------")

for row in cursor.fetchall():

print(row)

elif (input\_admin == 3):

query\_inv = """SELECT storage\_type, SUM(quantity\_in\_stock) AS amt FROM Products Group by storage\_type with ROLLUP having storage\_type is not null"""

cursor.execute(query\_inv)

for row in cursor.fetchall():

print(row)

if (input\_admin == 4):

input\_add\_category = input("Enter the name of Category that you want to Add: ")

category\_id = int(input("Enter Category ID: "))

query\_category = """INSERT INTO Category(category\_ID, category\_name) VALUES (%s,%s);"""

val = (category\_id, input\_add\_category)

cursor.execute(query\_category, val)

mydb.commit()

elif (input\_admin == 5):

query = "select \* from Category"

cursor.execute(query)

for row in cursor.fetchall():

print(f"Category ID: {row[0]}, Category Name: {row[1]}")

elif (input\_admin == 6):

query = "select \* from Seller"

cursor.execute(query)

for row in cursor.fetchall():

print(f"Seller ID: {row[0]}, Seller Name: {row[2]}, Product ID sold: {row[3]}, Quantity Sold: {row[4]}, Phone Number: {row[5]}")

elif (input\_admin == 7):

break;

else:

print("Invalid Input!")

#USER LOGIN

elif (input\_landing\_page == 2):

count = 0

valid\_user = 0

while(count<3 and valid\_user == 0):

query\_auth\_user = """Select username,password from Customer"""

username = input("Enter your username: ")

password = input("Enter your password: ")

cursor.execute(query\_auth\_user)

for row in cursor.fetchall():

if (username) == row[0] and (password) == row[1]:

# store = row

# print(row)

valid\_user = 1

print("Authenticated\n")

break

if valid\_user == 0:

print("Invalid Username or password\n")

count+=1

print(f"{3-count} tries remaining\n")

while(valid\_user):

print("---------------------------------------------")

print\_user\_menu(username)

input\_user = int(input("Enter the number from the menu: "))

if (input\_user == 1):

query = "select category\_name from Category"

cursor.execute(query)

i = 1

print("\nSelect the Category from below or type exit: ")

for row in cursor.fetchall():

print(f"{i}: {row[0]}")

i+=1

inp\_customer = input()

if inp\_customer != 'exit':

inp\_category = int(inp\_customer)

query2 = f"""select Products.product\_ID, Products.name, Products.category\_ID, Products.price from Category, Products

where Products.category\_ID = {inp\_category} group by Products.product\_ID order by Products.product\_ID asc

"""

cursor.execute(query2)

for row in cursor.fetchall():

print(f"{row[0]}: {row[1]}")

#accomodating for one quantity inc at a time?

inp\_id\_choice = input("Enter the id of the product you want to add to cart or type exit: ")

if inp\_id\_choice != 'exit':

inp\_id = int(inp\_id\_choice)

query\_3 = f"""select Products.product\_ID, Products.name, Products.price from Products

where Products.product\_ID = {inp\_id}

"""

cursor.execute(query\_3)

query\_cart = """insert into Cart (Billing\_Amount, product\_ID, Quantity, Customer\_username) values (%s, %s, %s, %s)"""

for row in cursor.fetchall():

id = row[0]

quantity = 1

cost = row[2]

val = (cost, id, quantity, username)

cursor.execute(query\_cart,val)

mydb.commit ()

elif (input\_user == 2):

query\_view\_cart = f"""select Cart.Quantity, Cart.Billing\_Amount, Products.product\_ID, Products.name, Cart.product\_ID, Cart.Customer\_Username from Products, Cart where Products.product\_ID = Cart.product\_ID"""

cursor.execute(query\_view\_cart)

bill = 0

for row in cursor.fetchall():

if (row[5] == username):

print(f"item name: {row[3]}, quantity: {row[0]}")

bill = row[1]

print(f"billing amount: {bill}")

elif (input\_user == 3):

#checkout or placing an order

print("Items in your cart: ")

query\_view\_cart = f"""select Cart.quantity, Cart.billing\_amount, Products.product\_ID, Products.name, Cart.product\_ID, Cart.customer\_username from Products, Cart where Products.product\_ID = Cart.product\_ID;"""

cursor.execute(query\_view\_cart)

last\_order\_id = 0

for row in cursor.fetchall():

if (row[5] == username):

print(f"item name: {row[3]}, quantity: {row[0]}")

bill = row[1]

query\_insert = """insert into Orders (order\_ID, Customer\_username, order\_amount, product\_ID, quantity, date\_order\_placed) values (%s, %s, %s, %s, %s, CURRENT\_DATE())"""

#Finding the last order id

last\_order\_id\_query = "SELECT MAX(order\_id) FROM orders"

cursor.execute(last\_order\_id\_query)

last\_order\_id\_result = cursor.fetchone()

last\_order\_id = last\_order\_id\_result[0] if last\_order\_id\_result else 0

val = (last\_order\_id+1, username, round(float(bill),2), row[2], row[0])

cursor.execute(query\_insert,val)

mydb.commit ()

bill\_amount = bill

# bill\_amount = 0

# for row in cursor.fetchall():

# if (row[5] == username):

# print(f"item name: {row[3]}, quantity: {row[0]}")

# bill\_amount = row[1]

# print(f"billing amount: {bill\_amount}")

method\_to\_pay = input("Method to pay (COD/UPI/card/wallet) : ")

query\_insert = """insert into Billing (billing\_ID, payment\_mode, bill\_amount, order\_ID) values ( %s, %s, %s, %s)"""

#Finding the last billing id

last\_billing\_id\_query = "SELECT MAX(billing\_id) FROM billing"

cursor.execute(last\_billing\_id\_query)

last\_billing\_id\_result = cursor.fetchone()

last\_billing\_id = last\_billing\_id\_result[0] if last\_billing\_id\_result else 0

val = (last\_billing\_id+1, method\_to\_pay, round(float(bill\_amount),2), last\_order\_id+1)

cursor.execute(query\_insert,val)

mydb.commit ()

# Track Orders Details.

elif (input\_user == 4):

# Track Orders Query

username = input("Enter username: ")

query\_track\_orders = """SELECT TrackOrder.delivery\_ID, TrackOrder.status, Orders.order\_amount, Products.name, TrackOrder.expected\_arrival\_time

FROM TrackOrder

INNER JOIN Orders ON TrackOrder.order\_ID = Orders.order\_ID

INNER JOIN Products ON Orders.product\_ID = Products.product\_ID

WHERE Orders.customer\_username = %s"""

cursor.execute(query\_track\_orders, (username,))

orders = cursor.fetchall()

# Convert fetched data into a list of lists for tabulate

data = [list(order) for order in orders]

if data:

# Define table headers

headers = ["Delivery ID", "Status", "Amount", "Products", "Expected Arrival Time"]

# Print the tabulated output

print("Your Orders:")

print(tabulate(data, headers=headers, tablefmt="grid")) # You can change "grid" to other table formats

else:

print("You have no orders yet.")

# Return Order Details.

elif(input\_user == 5):

# Function to calculate expected return date

def calculate\_expected\_return\_date(order\_date):

return order\_date + timedelta(days=14)

# Function to check if a return is within the allowed return period

def is\_within\_return\_period(order\_date\_str):

try:

order\_date = datetime.strptime(order\_date\_str, "%Y-%m-%d")

expected\_return\_date = calculate\_expected\_return\_date(order\_date)

return datetime.now() <= expected\_return\_date

except ValueError:

return False

while True:

order\_id = int(input("Enter your Order ID to check return details (Enter 0 to exit): "))

if order\_id == 0:

print("Exiting...")

break

# Check if the order exists

query\_check\_order = "SELECT \* FROM Orders WHERE order\_ID = %s"

cursor.execute(query\_check\_order, (order\_id,))

order\_data = cursor.fetchone()

if order\_data:

order\_date = order\_data[1]

if is\_within\_return\_period(order\_date):

return\_date = input("Enter return date (YYYY-MM-DD): ")

query\_insert\_return = "INSERT INTO Return\_entity (order\_ID, return\_date) VALUES (%s, %s)"

cursor.execute(query\_insert\_return, (order\_id, return\_date))

mydb.commit()

print("Return order successfully added.")

else:

print("Return period has expired. You cannot return this order.")

else:

print("Order not found. Please enter a valid Order ID.")

elif (input\_user == 6):

break

else:

print("Invalid Input!")

#USER SIGN UP

elif (input\_landing\_page == 3):

print("User SignUp")

username = input("Set your username: ")

password = input("Set your password: ")

f\_name = input("Enter First Name: ")

l\_name = input("Enter Last Name: ")

phone = input("Enter phone number: ")

h\_no = int(input("Enter house number: "))

street = input("Enter street name: ")

city = input("Enter city: ")

pin = int(input("Enter 6 digit pin code: "))

query\_insert = """insert into Customer (username, password, first\_name, last\_name, phone\_number, house\_number, street\_name, city, pincode, login\_attempts, is\_blocked) values (%s,%s,%s,%s,%s,%s,%s,%s,%s, 0,0)"""

val = (username, password, f\_name, l\_name, phone, h\_no, street, city, pin)

cursor.execute(query\_insert,val)

mydb.commit ()

print ('Successful SignUp')

print("Redirecting to Home Page")

#print("\nThe SQL cursorection is closed.")

elif (input\_landing\_page == 4):

while(True):

query\_auth\_seller = """SELECT seller\_ID, password, name FROM Seller"""

seller\_name = input("Enter your Seller Name: ")

password = input("Enter your password: ")

cursor.execute(query\_auth\_seller)

valid\_user = False

seller\_data = None

for row in cursor.fetchall():

if seller\_name == row[2] and password == row[1]:

seller\_data = row

valid\_user = True

break

if valid\_user:

print("Authenticated\n")

break

else:

login\_attempts += 1

if login\_attempts < max\_attempts:

print("Invalid Seller Name or Password. Please try again.\n")

print(f"{max\_attempts - login\_attempts} attempts remaining.\n")

else:

print("Maximum login attempts reached. Exiting...\n")

exit()

while valid\_user:

print\_seller\_menu(seller\_name)

input\_seller = int(input("Enter the number: "))

if input\_seller == 1:

print("\nSeller Details: ")

# Fetch all attributes and their values for the particular seller

query\_fetch\_details = """SELECT \* FROM Seller WHERE seller\_ID = %s"""

cursor.execute(query\_fetch\_details, (seller\_data[0],))

seller\_details = cursor.fetchone()

# Display all attributes and their values

print("Seller ID:", seller\_details[0])

print("Product ID:", seller\_details[3])

print("Quantity Sold:", seller\_details[4])

print("Phone Number:", seller\_details[5])

print("Email Address:", seller\_details[6])

print("Earnings:", seller\_details[7])

print("Payment Details:", seller\_details[8])

elif input\_seller == 2:

break

else:

print("Invalid Input!")

elif(input\_landing\_page == 5):

break

else:

print("Sorry, invalid number")

print("Please try again\n")

mydb . close ()

if (mydb):

mydb.close()

CONTRIBUTION:

Kanak ( 2022611) : Transaction, User Interface, User guide CLI

Prakriti Jain (2022362): Transaction + Documentation

Neelu (2022318): Transaction

Sahil Prajapati (2022428) :