DBMS PROJECT

CLOCEAN

OUR TEAM

Harshvardhan Singh

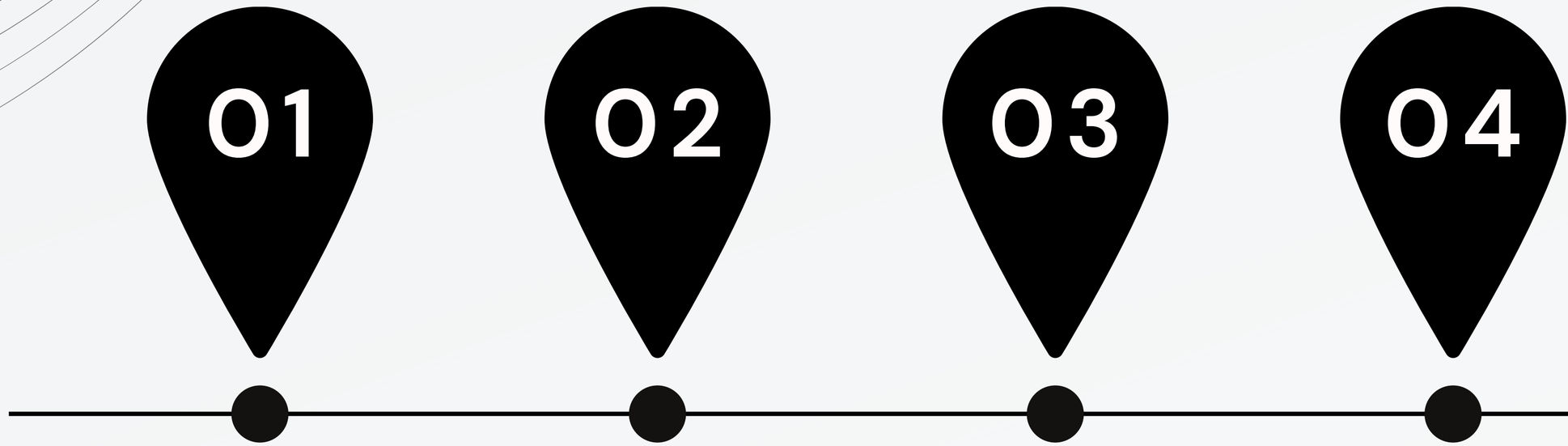
2021052

Himani

2021053

CONTENT

PROJECT TIMELINE



10 FEBURARY

ER Diagram, Relational Schema, Database schema, integrity constraints and data insertion

17 FEBRUARY

Executed various diverse SQL queries

26 MARCH

Executed OLAP queries and triggers, started basic interface, wrote embedded queries

20 APRIL

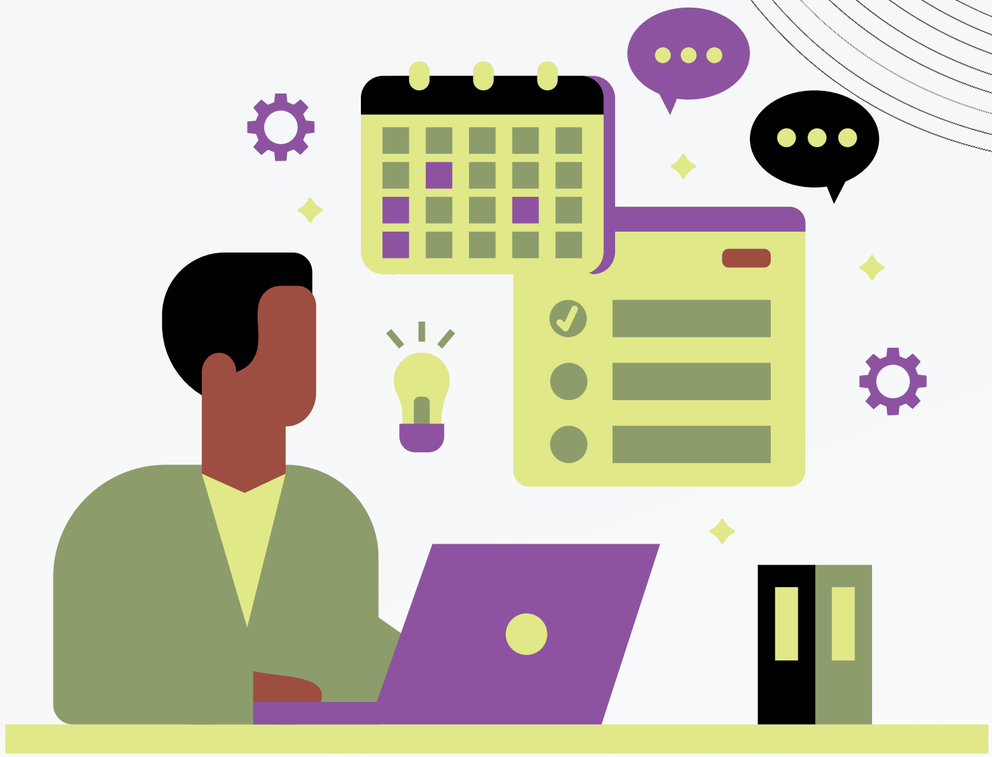
Executed conflicting and non-conflicting transactions, completed user interface

PROJECT SCOPE

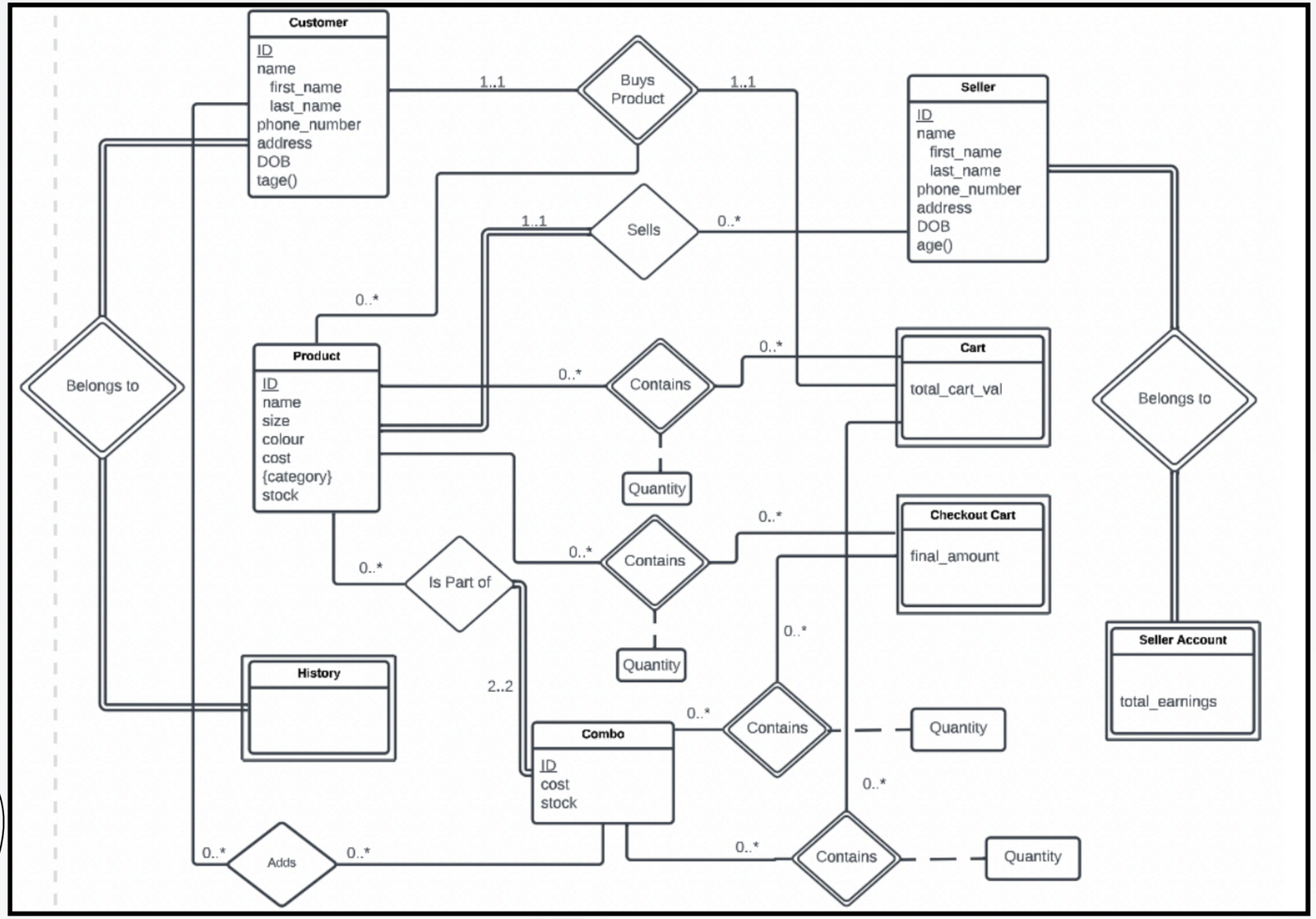
Our application acts as a bridge between customers and cloth sellers. The sellers can add their products to the application, and the customers can choose from all available products. To add their product, a seller will first need to register as a seller on our application, and then he can add products. All the clothes the seller adds will fall into a clothing category with other specifications like size, price, brand and style.

Once logged in, a seller can also update a product's details. Similar to a seller, a customer would also have to first register on the application to buy products. After registering, they will then be able to log in and browse for different products. They can choose the products they want to buy, add them to their cart, and then proceed to checkout. After the payment is confirmed, the updated stock will also reflect in the seller’s account, which he can also check. The customer also has the functionality to see his past orders.

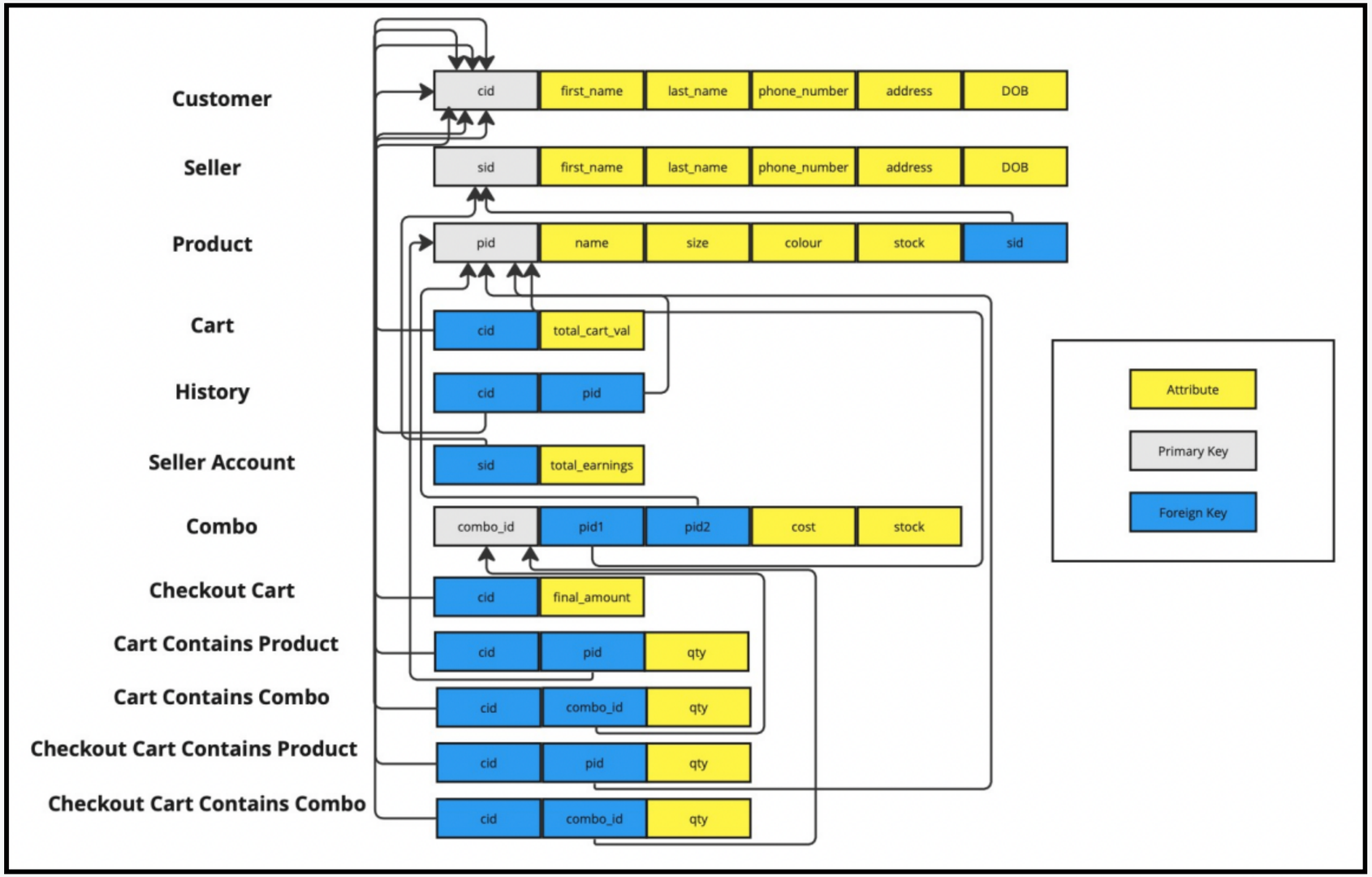
Our application will also have specialized combinations of products at discounted rates.
We will use databases to store and maintain all the relevant information of the seller, customers, and cart details with the history of all the orders.



ER DIAGRAM


Customer
ID
name
first_name
last_name
phone_number
address
DOB
tage()
Buys Product
1..1
1..1
Seller
ID
name
first_name
last_name
phone_number
address
DOB
age()
Sells
1..1
0..*
0..*
Belongs to
Product
ID
name
size
colour
cost
{category}
stock
0..*
Contains
0..*
Cart
total_cart_val
Belongs to
Quantity
Checkout Cart
final_amount
0..*
0..*
Contains
0..*
Is Part of
Quantity
Seller Account
total_earnings
History
2..2
Combo
ID
cost
stock
0..*
Contains
Quantity
0..*
0..*
Adds
0..*
0..*
Contains
Quantity

RELATIONAL SCHEMA



DATA POPULATION

LINK

SQL QUERIES

LINK

TRIGGERS

LINK

OLAP QUERIES

LINK

CONFLICTING TRANSACTIONS

```python
def updateproductstock(pid):
    conn = c.connect(host = "localhost",user = "root", password = "maverick",db = "cloocean", autocommit=True)
    curr = conn.cursor()
    n = input("Enter the new stock for the product: ")
    curr.execute("set autocommit = 0;")
    curr.execute("start transaction;")
    curr.execute("update product set stock = "+ n+" where pid = '"+str(pid)+"';")
    print("Stock update successfully !!!")
    curr.execute("commit;")
    curr.execute("set autocommit = 1;")
    conn.close()
```

T1

```python
def addProductToCart(cid,pid,qty):
    conn = c.connect(host = "localhost",user = "root", password = "maverick",db = "cloocean", autocommit=True)
    curr = conn.cursor()
    curr.execute("set autocommit = 0;")
    curr.execute("start transaction;")
    curr.execute("select stock from product where pid = '"+pid+"';")
    stock = curr.fetchall()
    if(qty>stock[0][0]):
        print("Not enough stock")
        conn.close()
        return
    curr.execute("insert into cart_contains_product values('"+str(cid)+"','"+str(pid)+"','"+str(qty)+"');")
    curr.execute("commit;")
    curr.execute("set autocommit = 1;")
    conn.close()
```

T2

Updating product stock (T1) and adding the product to a cart (T2):

Let's suppose a user wants to add a product to their cart with a quantity that is greater than the stock of the product. Now if T1 is scheduled before T2 then they would be able to add the product with their desired quantity. But if T2 is scheduled before T1 our T2 is running while T1 starts, then he won't be able to do so.

CONFLICTING TRANSACTIONS

```python
def updateproductscost(pid):
    conn = c.connect(host = "localhost",user = "root", password = "maverick",db = "cloocean", autocommit=True)
    curr = conn.cursor()
    n = input("Enter the new cost for the product: ")
    curr.execute("set autocommit = 0;")
    curr.execute("start transaction;")
    curr.execute("update product set cost = "+ n+" where pid = '"+str(pid)+"';")
    print("Stock update successfully !!!")
    curr.execute("commit;")
    curr.execute("set autocommit = 1;")
    conn.close()
```

T1

```python
def checkoutCart(cid):
    conn = c.connect(host = "localhost",user = "root", password = "maverick",db = "cloocean", autocommit=True)
    curr = conn.cursor()
    # Updating the cart, setting final amount to 0
    curr.execute("set autocommit = 0;")
    curr.execute("start transaction;")
    curr.execute("update cart set total_cart_val = 0 where cid = '"+str(cid)+"';")
    # Updating product and combo stock
    curr.execute("update product as p,cart_contains_product as c set p.stock = p.stock - c.qty where p.pid = c.pid and c.cid ='"+str(cid)+"';")
    curr.execute("update combo as co,cart_contains_combo as c set co.stock = co.stock - c.qty where co.combo_id = c.combo_id and c.cid ='"+str(cid)+"';")
    # Updating the history table
    curr.execute("select * from cart_contains_product where cid = '"+str(cid)+"';")
    output1 = curr.fetchall()
    curr.execute("select * from cart_contains_combo where cid = '"+str(cid)+"';")
    output2 = curr.fetchall()
    for i in output1:
        curr.execute("insert into history (cid,pid) values('"+str(cid)+"','"+str(i[1])+"');")
    for i in output2:
        curr.execute("insert into history (cid,combo_id) values('"+str(cid)+"','"+str(i[1])+"');")
    # Clearing the cart
    curr.execute("delete from cart_contains_product where cid = '"+str(cid)+"';")
    curr.execute("delete from cart_contains_combo where cid = '"+str(cid)+"';")
    # Updating seller_acc
    for i in output1:
```

T2

Updating product cost (T1) while a cart is being checked out (T2):
If the seller updates product cost before the cart being checked out, that is T1 is scheduled before T2 then the checkout would be correct but if T2 is scheduled before T1 or seller updates product cost while the cart is being checked out, there would be discrepancy.

NON CONFLICTING TRANSACTIONS

T1

```
def addProductforseller(pid,product_name,size,color,cost,stock,sid,category):
    conn = c.connect(host = "localhost",user = "root", password = "maverick",db = "cloocean", autocommit=True)
    curr = conn.cursor()
    curr.execute("set autocommit = 0;")
    curr.execute("start transaction;")
    curr.execute("insert into product values('"+str(pid)+"','"+str(product_name)+"','"+str(size)+"','"+str(color)+"','"+str(cost)+"','"+str(stock)+"','"+str
    curr.execute("commit;")
```

T2

```
def viewAllCombos():
    conn = c.connect(host = "localhost",user = "root", password = "maverick",db = "cloocean", autocommit=True)
    curr = conn.cursor()
    curr.execute("set autocommit = 0;")
    curr.execute("start transaction;")
    curr.execute("Select * from combo")
    output = curr.fetchall()
    for i in output:
        print(i)
    curr.execute("commit;")
    curr.execute("set autocommit = 1;")
    conn.close()
```

T1 and T2 do not conflict with each other. Although T1 is a write operation but, it does not involve the combo table which is being read by T2.

NON CONFLICTING TRANSACTIONS

```python
def viewbestseller():
    conn = c.connect(host = "localhost",user = "root", password = "maverick",db = "cloocean", autocommit=True)
    curr = conn.cursor()
    curr.execute("set autocommit = 0;")
    curr.execute("start transaction;")
    curr.execute("SELECT p.category, p.product_name, COUNT(*) as frequency FROM history h JOIN product p ON h.pid = p.pid WHERE h.cid IN (SELECT cid FROM cart) GROUP BY
    output = curr.fetchall()
    for i in output:
        print(i)
    curr.execute("commit;")
    curr.execute("set autocommit = 1;")
    conn.close()
```

T1

```python
def viewAllProducts():
    conn = c.connect(host = "localhost",user = "root", password = "maverick",db = "cloocean", autocommit=True)
    curr = conn.cursor()
    curr.execute("set autocommit = 0;")
    curr.execute("start transaction;")
    curr.execute("Select * from product")
    output = curr.fetchall()
    for i in output:
        print(i)
    curr.execute("commit;")
    curr.execute("set autocommit = 1;")
    conn.close()
```

T2

T1 and T2 do not conflict with each other. Although T1 and T2 both, are transactions on the same table "Product", they do not conflict with each other because both of them are read transactions.

FINAL INTERFACE

LINK

USER GUIDE

Welcome to the user guide for Cloocean, an online marketplace management system! This guide will walk you through each and every function of the program and how to use them.
When you run the program, you will see the main menu with four options:

1. Login
2. Register
3. Admin Login
4. Exit

MAIN MENU

Register

If you are a new user, select the second option to register as a customer or seller. You will see two sub-options: Register as customer or Register as seller. Choose one of the options and enter your details to complete the registration process. Once you have registered, you can log in to your account using the Login option.

MENU FOR REGISTRATION

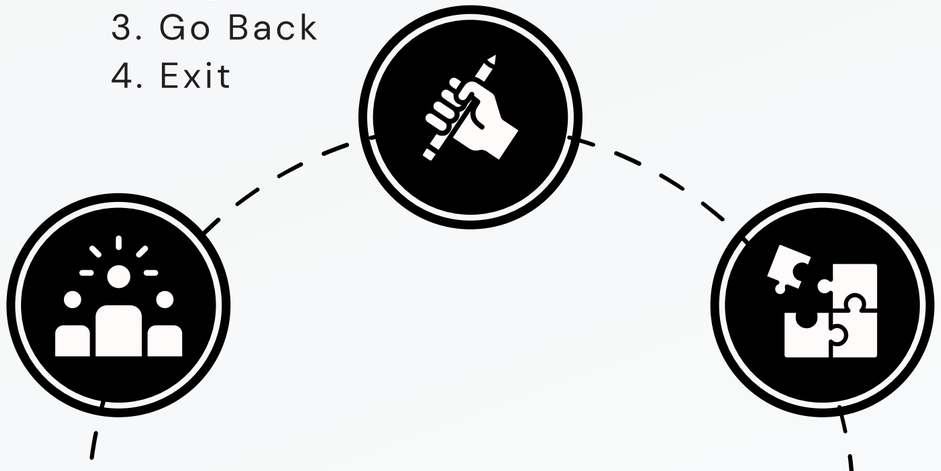
1. Register as customer
2. Register as seller
3. Go Back
4. Exit

Login

If you are an existing customer or seller, select the first option to log in to your account. You will be asked to enter your username and password. After successful authentication, you will be taken to the respective customer or seller dashboard.

MENU FOR LOGIN

1. Login as customer
2. Login as seller
3. Go Back
4. Exit



Admin

The third option is for administrators only. If you are an administrator, select this option and enter the admin username and password to log in. You will be taken to the admin dashboard.

MENU FOR ADMIN

1. Check Top selling products by category
2. Check Customer id with the highest total value
3. Get total earnings of sellers by category
4. Get the total number of products and combos sold by category and size
5. Check all the customer ids for whom the cart is empty
6. Go Back

CUSTOMER DASHBOARD

The customer dashboard is a menu-driven program that allows customers to interact with the online store.

******** MAIN MENU ********

1. View all products
2. View all combos
3. Add a product to cart
4. Add a combo to cart
5. View cart
6. View History
7. My profile
8. Back
9. Exit

****

CUSTOMER

Enter Customer ID - At the beginning of the program, the customer is asked to enter their Customer ID. After entering the Customer ID, they will be prompted to enter their password. If the entered credentials are valid, the customer can access the dashboard.

Main Menu - After successfully logging in, the customer will be presented with a main menu.

1. **View Products/Combos** - The customer can select either option 1 or option 2 from the main menu to view all products or combos respectively.
2. **Add Product/Combo to Cart** - To add a product or combo to the cart. They will be prompted to enter the Product/Combo ID and the quantity they want to add to their cart.
3. **View Cart** - To view the items in their cart. They can also drop any product/combo from the cart or checkout the cart by selecting the respective options from the sub-menu.
4. **View History** - To view the purchase history of the products purchased earlier.
5. **My Profile** - To view the profile information.
6. **Back/Exit** - Customers can go back to the previous menu or exit the program by selecting option 8 or 9 respectively.

SELLER DASHBOARD

The seller dashboard is a menu-driven program that allows sellers to interact with the online store.

******** MAIN MENU ********

1. View my products
2. View my combos
3. Edit product/combo details
4. Add product
5. Add combo
6. View Bestseller
7. My profile
8. Go Back
9. Exit

****

SELLER

Enter Seller ID: At the beginning of the program, the seller is asked to enter their Seller ID. After entering the Seller ID, they will be prompted to enter their password. If the entered credentials are valid, the customer can access the dashboard.

Main menu: After successfully logging in, the customer will be presented with a main menu.

1. **View my products:** This option allows the seller to view all the products they have listed on the platform.
2. **View my combos:** This option allows the seller to view all the combos they have created by combining two products.
3. **Edit product/combo details:** This option allows the seller to edit the details of their products and combos, including stock, cost.
4. **Add product:** This option allows the seller to add a new product to their catalog by entering its details, including the product ID, name, size, color, cost, stock, and category.
5. **Add combo:** This option allows the seller to create a new combo by combining two existing products. The seller needs to enter the IDs of the two products, the combo ID, cost, and stock.
6. **View bestseller:** This option displays the top-selling products on the platform.
7. **My profile:** This option allows the seller to view their profile information, including their contact details and account information.
8. **Go back:** This option takes the seller back to the main menu.
9. **Exit:** This option allows the seller to exit the program.

ADMIN DASHBOARD

The Admin Dashboard is a feature of the program that allows the administrator to perform various functions related to the sales and management of products.

******** MAIN MENU ********

1. Check Top selling products by category
2. Check Customer id with the highest total value
3. Get total earnings of sellers by category
4. Get the total number of products and combos sold by category and size
5. Check all the customer ids for whom the cart is empty
6. Go Back

****

ADMIN

To select an option, the user must enter the corresponding number and press enter. Each option corresponds to a specific function that the administrator can perform.

1. **Check Top selling products by category:** This option allows the administrator to view the top selling products in each category.
2. **Check Customer id with the highest total value:** This option allows the administrator to find the customer with the highest total purchase value.
3. **Get total earnings of sellers by category:** This option allows the administrator to view the total earnings of the sellers in each category.
4. **Get the total number of products and combos sold by category and size**: This option allows the administrator to view the total number of products and combos sold in each category and size.
5. **Check all the customer ids for whom the cart is empty:** This option allows the administrator to find all the customers who have an empty cart.
6. **Go Back:** This option allows the administrator to return to the main menu.
7. **Exit:** This option allows the administrator to exit the program.

REGISTRATION DASHBOARD

The Registration Dashboard is a feature of the program that allows the a customer/seller to get enrolled with CLOOCEAN.

****

REGISTERATION

****

1. Register as customer
2. Register as seller
3. Go Back
4. Exit

****

REGISTER

To select an option, the user must enter the corresponding number and press enter. Each option corresponds to a specific function that the administrator can perform.

1. **Register as customer:** This option allows the person to register as customer with the platform.
2. **Register as seller:** This option allows the person to get registered with the platform as seller.
3. **Go Back:** This option allows the administrator to return to the main menu.
4. **Exit:** This option allows the administrator to exit the program.

HOW TO RUN

Instructions Required to Run

1) Install MySQL Connector, Python 3

2) Install MySQL

3) Compile & Run the user interface