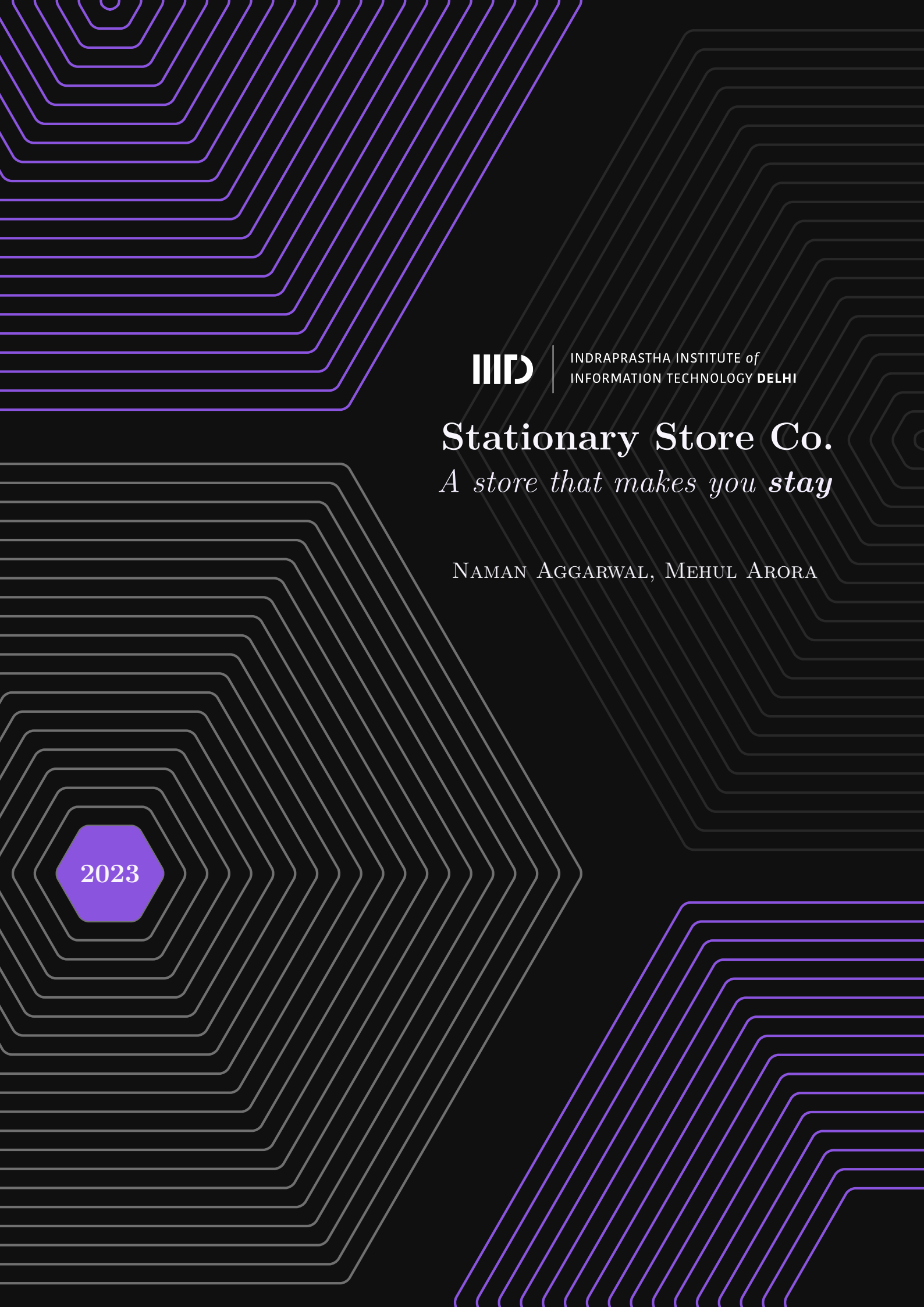


INDRAPRASTHA INSTITUTE *of*
INFORMATION TECHNOLOGY **DELHI**

Stationary Store Co.

*A store that makes you **stay***

Naman Aggarwal, Mehul Arora

2023

Introduction

Stationary Store Co. is an online store for artistic equipment as well as daily-use office/home stationery products.

Tech Stack

- MySQL
- Python
- Django

Functional Requirements

- The app will require the store owners to log in through an admin panel and add products to the database.
- The system will allow employees to add, update, and delete products in the inventory.
- The system will provide a way for employees to view current inventory levels and stock alerts and reorder items when necessary.
- The users of the application will access the online store and will be able to access the entire catalogue of products.
- The users can use various filtering and sorting options available, which are necessary for stationery and art supplies.
- The users can then add supplies to their cart.
- To checkout, the users need to sign up or register. Their credentials are hashed and stored in a table, and the hashes are compared for authentication.
- Whenever users sign up, they get access to a basic level of membership, and further, upon purchases, they will be promoted to a higher level.

- When the user finally buys the items, the inventory is updated, and a bill is generated for the user.
- After purchase, users receive coupons which can be applied to further purchases.

Technical Requirements

- The users need to be uniquely identified. This can be done through their E-mail IDs/Contact numbers.
- The inventory needs to be updated regularly with each order, and the product needs to be shown as "Out of Stock" when the inventory runs out.
- There will be membership levels for users, and after every purchase, it has to be checked if the user can avail higher level of membership.
- The passwords need to be stored securely, and users should be properly authenticated. Web App Security will be a major factor to look at.
- There should be extensive sorting and searching features present. This is because the target demographic of art enthusiasts usually needs to filter products over a lot of criteria such as colour, brand, use case and so on.

ACID Properties

We have devised a set of guidelines to follow throughout the project to ensure that the ACID Properties (Atomicity, Consistency, Isolation, and Durability) are practised, and divided them into three sections: Buyer, Seller and General, depending on the stakeholders.

For Sellers:

- Use transactions when adding, updating, or deleting products from the database.
- Implement proper locking mechanisms to prevent multiple sellers from editing the same product at the same time.
- Use triggers to automatically update the inventory levels whenever a purchase is made, or a product is added/removed.
- Use views to ensure that the data displayed to sellers is consistent and accurate.
- Implement backup and recovery procedures to ensure that data is not lost in the event of a system failure.

For Buyers:

- Use transactions when adding items to the cart or making a purchase.
- Implement proper locking mechanisms to prevent multiple buyers from purchasing the same item at the same time.
- Use triggers to ensure that inventory levels are updated properly after a purchase is made.

- Use views to ensure that the data displayed to buyers (such as product availability and pricing) is consistent and accurate.
- Use hashing and secure storage mechanisms to protect user credentials and ensure proper authentication.

General:

- Use MySQL's built-in features like transactions, triggers, stored procedures, and views to maintain consistency and ensure data integrity.
- Monitor the system regularly to ensure that ACID properties are being followed.
- Train stakeholders on proper usage of the system to ensure that they are not inadvertently compromising data integrity.

Additional Scope

The project can be expanded to include the selling of old used books.

Conceptual Model

Entity Relationship Diagram

‘

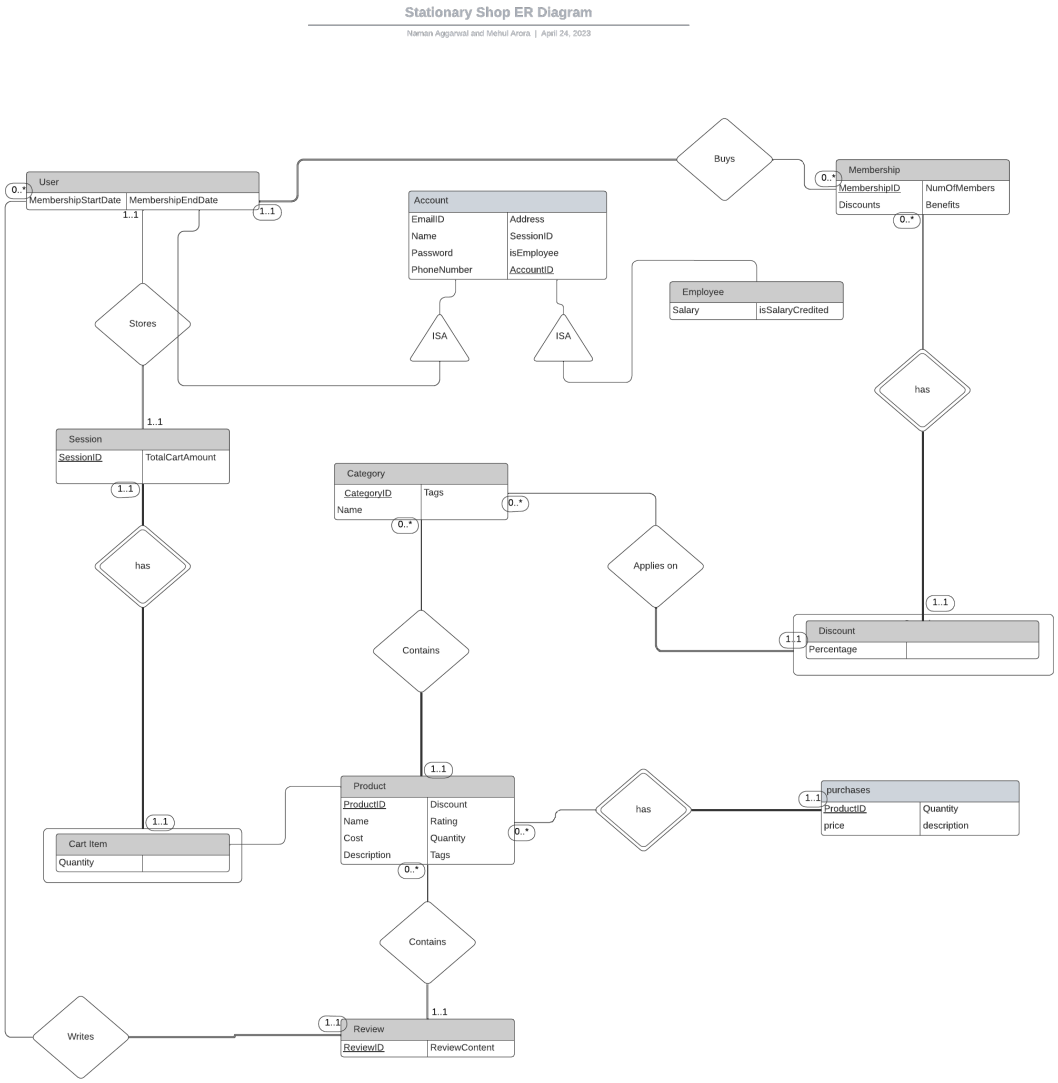


Figure 2.1: Entity Relationship Diagram for the Database.

Relational Schema

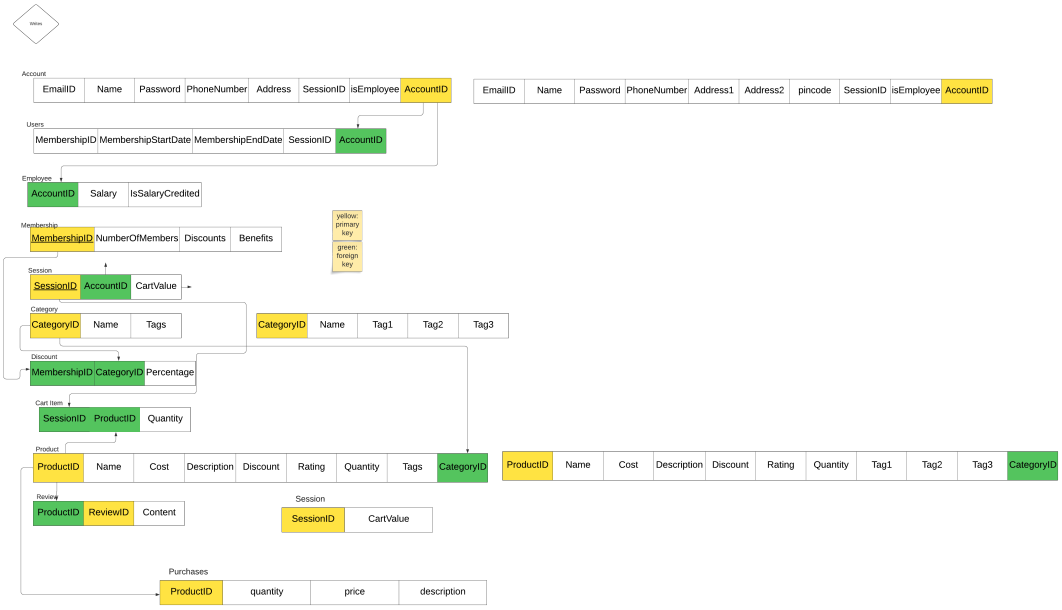


Figure 2.2: Relational Schema for the Database.

SQL Schema and Population

Database Schema

Code:

```
create table Session(
    SessionID int NOT NULL AUTO_INCREMENT,
    CartValue int NOT NULL DEFAULT 0,
    PRIMARY KEY (SessionID)
);
create table Accounts(
    AccountID int NOT NULL AUTO_INCREMENT,
    Name varchar(40) NOT NULL,
```

```
    EmailID varchar(100) NOT NULL UNIQUE,
    Password varchar(30) NOT NULL,
    PhoneNumber BIGINT NOT NULL,
    Address1 varchar(50) NOT NULL,
    Address2 varchar(50),
    Pincode int NOT NULL,
    SessionID int UNIQUE,
    IsEmployee bit NOT NULL DEFAULT 0,
    PRIMARY KEY (AccountID),
    FOREIGN KEY (SessionID) REFERENCES Session(SessionID)
);
create table Membership(
    MembershipID int NOT NULL AUTO_INCREMENT,
    MembershipType varchar(20) NOT NULL UNIQUE,
    NumberOfMembers int NOT NULL DEFAULT 0,
    Discount float NOT NULL DEFAULT 0,
    Benefits varchar(200),
    PRIMARY KEY (MembershipID)
);
create table Users(
    AccountID int NOT NULL UNIQUE,
    MembershipID int NOT NULL,
    MembershipStartDate date NOT NULL,
    MembershipEndDate date NOT NULL,
    PRIMARY KEY (AccountID),
    FOREIGN KEY (MembershipID) REFERENCES Membership(MembershipID),
    FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)
);
create table Employee(
    AccountID int NOT NULL UNIQUE,
    Salary int NOT NULL DEFAULT 0,
    isSalaryCredited bit NOT NULL DEFAULT 0,
    FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)
);
create table Tag(
    TagID int NOT NULL AUTO_INCREMENT,
    TagName varchar(50) NOT NULL,
    PRIMARY KEY (TagID)
);
create table Category(
    CategoryID int NOT NULL AUTO_INCREMENT,
    Name varchar(50) NOT NULL,
    Tag1 int ,
    Tag2 int ,
    Tag3 int,
    PRIMARY KEY (CategoryID),
    -- FOREIGN KEY (Tag1,Tag2,Tag3) REFERENCES Tag(TagID)
```

```
    FOREIGN KEY (Tag1) REFERENCES Tag(TagID),
    FOREIGN KEY (Tag2) REFERENCES Tag(TagID),
    FOREIGN KEY (Tag3) REFERENCES Tag(TagID)
);
create table Discount(
    MembershipID int NOT NULL,
    CategoryID int NOT NULL,
    Percentage float NOT NULL DEFAULT 0,
    PRIMARY KEY (MembershipID,CategoryID),
    FOREIGN KEY (MembershipID) REFERENCES Membership(MembershipID),
    FOREIGN KEY (CategoryID) REFERENCES Category(CategoryID)
);
create table Product(
    ProductID int NOT NULL AUTO_INCREMENT,
    CategoryID int NOT NULL,
    Name varchar(30) NOT NULL,
    Cost int NOT NULL,
    Description varchar(200) NOT NULL,
    Discount float NOT NULL DEFAULT 0,
    Rating float NOT NULL DEFAULT 0,
    Quantity int NOT NULL,
    Tag1 int ,
    Tag2 int ,
    Tag3 int,
    PRIMARY KEY (ProductID),
    FOREIGN KEY (CategoryID) REFERENCES Category(CategoryID),
    -- FOREIGN KEY (Tag1,Tag2,Tag3) REFERENCES Tag(TagID)
    FOREIGN KEY (Tag1) REFERENCES Tag(TagID),
    FOREIGN KEY (Tag2) REFERENCES Tag(TagID),
    FOREIGN KEY (Tag3) REFERENCES Tag(TagID)
);
create table Review(
    ReviewID int NOT NULL AUTO_INCREMENT,
    ProductID int NOT NULL,
    Rating int NOT NULL DEFAULT 0,
    Content varchar(300),
    PRIMARY KEY (ReviewID),
    FOREIGN KEY (ProductID) REFERENCES Product(ProductID)
);
create table CartItem(
    SessionID int NOT NULL,
    ProductID int NOT NULL,
    Quantity int NOT NULL DEFAULT 1,
    PRIMARY KEY (SessionID,ProductID),
    FOREIGN KEY (SessionID) REFERENCES Session(SessionID),
    FOREIGN KEY (ProductID) REFERENCES Product(ProductID)
);
```

Integrity Constraints

The following are the constraints we followed while making the Database Schema:

1. **`NOT NULL` constraint:** This makes sure none of the values for that field is `NULL`.
2. **`UNIQUE` constraint:**This constraint ensures that each of the values for that field is distinct.
3. **`DEFAULT` constraint:** This sets a default value for the field in case no value is provided while populating the table.
4. **`PRIMARY KEY` constraint:** This ensures that each record's next value for that field is a discrete, fixed increment from the previous value.
5. **`PRIMARY KEY` constraint:** This is a combination of the `UNIQUE` and `NOT NULL` constraints. It sets a field as the unique identifying field for each record in the table.
6. **`FOREIGN KEY` constraint:** This ensures that the record's value for that specific field is present in the referred table.

Data Population

Code:

```
import random
import datetime
import names
import secrets

accountNames = [names.get_full_name() for i in range(100)]
accountEmailIDs = ["".join(random.choices("abcdefghijklmnopqrstuvwxyz", k=7)) +
     "@gmail.com" for i in range(100)]
accountPasswords = [secrets.token_hex(8) for i in range(100)]
accountPhoneNumbers = [random.randint(1000000000,9999999999) for i in range
    (100)]
accountAddress1 = [" ".join(random.choices("abcdefghijklmnopqrstuvwxyz",k=10))
    for i in range(100)]
accountAddress2 = [" ".join(random.choices("abcdefghijklmnopqrstuvwxyz",k=10))
    for i in range(100)]
accountPincode = [random.randint(100000,999999) for i in range(100)]
accountSessionID = [i for i in range(1,101)]
isEmployee = [random.randint(0,1) for i in range(100)]


for i in range(100):
    print("insert into Accounts(Name,EmailID,Password,PhoneNumber,Address1,
        Address2,Pincode,SessionID,IsEmployee) values
```

```
        ('{}','{}','{}',{},'{}','{}',{},NULL,{});".format(accountNames[i],
        accountEmailIDs[i],accountPasswords[i],accountPhoneNumbers[i],
        accountAddress1[i],accountAddress2[i],accountPincode[i],isEmployee[i]))

for i in range(100):
    print("insert into Session(CartValue) values(0);")

for i in range(100):
    print("insert into Membership(MembershipType,NumberOfMembers,Discount,
        Benefits) values('{}',0,0,'{}');".format("".join(random.choices("
        abcdefghijklmnopqrstuvwxyz",k=10)),"".join(random.choices("
        abcdefghijklmnopqrstuvwxyz",k=10))))

for i in range(100):
    if isEmployee[i]:
        print("insert into Employee(AccountID,Salary,isSalaryCredited) values
            ({},0,0);".format(i+1))
    else:
        print("insert into Users(AccountID,MembershipID,MembershipStartDate,
            MembershipEndDate) values({},1,'{}','{}');".format(i+1,datetime.date.
            today(),datetime.date.today()+datetime.timedelta(days=365)))

for i in range(100):
    print("insert into Tag(TagName) values('{}');".format("".join(random.
        choices("abcdefghijklmnopqrstuvwxyz",k=10))))

for i in range(100):
    print("insert into Category(Name,Tag1,Tag2,Tag3) values('{}',{},NULL,NULL)
        ;".format("".join(random.choices("abcdefghijklmnopqrstuvwxyz",k=10)),i
        +1))

for i in range(100):
    print("insert into Discount(MembershipID,CategoryID,Percentage) values
        ({},{},{});".format(i+1, random.randint(1,100), random.randint(1,100)))

for i in range(100):
    print("insert into Product(CategoryID,Name,Cost,Description,Discount,Rating,
        Quantity,Tag1,Tag2,Tag3) values(1,'{}',{},'{}',0,0,0,NULL,NULL,NULL);".
        format("".join(random.choices("abcdefghijklmnopqrstuvwxyz",k=10)),random
        .randint(100,1000)," ".join(random.choices("abcdefghijklmnopqrstuvwxyz",
        k=10))))

for i in range(100):
    print("insert into Review(ProductID,Rating,Content) values(1,0,'{}');".
        format(" ".join(random.choices("abcdefghijklmnopqrstuvwxyz",k=10))))

for i in range(100):
```

```
    print("insert into CartItem(SessionID,ProductID,Quantity) values({},1,1);".
        format(i+1))
```


Sample Queries

Complex Selects/Inserts/Updates

Code:

```
-- insert users 11, 16, 19 and 22 as employees
INSERT INTO Employee (AccountID, Salary, isSalaryCredited)
VALUES (11, 100, 0), (16, 100, 0), (19, 100, 0), (22, 100, 0);

-- insert a new user with an expired membership
INSERT INTO Users (AccountID, MembershipID, MembershipStartDate,
    MembershipEndDate)
VALUES (2, 1, '2017-01-01', '2017-01-01');

-- select members whose membership has expired

SELECT Users.AccountID, Users.MembershipID, Users.MembershipStartDate, Users.
    MembershipEndDate
FROM Users
WHERE Users.MembershipEndDate < CURDATE();


-- Nested query to select emails of users who have not renewed their membership
     in the last 3 months
SELECT Accounts.AccountID, Accounts.Name, Accounts.EmailID
```

```
FROM Accounts
WHERE AccountID IN (
    SELECT AccountID
    FROM Users
    WHERE MembershipEndDate < DATE_SUB(CURDATE(), INTERVAL 3 MONTH)
);

-- update the membership of all users that has expired
UPDATE Users
SET MembershipID=1, MembershipStartDate=CURDATE(), MembershipEndDate=DATE_ADD(
    CURDATE(), INTERVAL 1 YEAR)
WHERE Users.MembershipEndDate < CURDATE();

select p.CategoryID,count(c.CategoryID) as count,c.Name from Product as p,
    Category as c where p.CategoryID=c.CategoryID group by c.CategoryID;

--counts the number of products in a particular Category and prints category
    name, CategoryID, Count.(this query demonstrates the usage of natural join)
select * from discount where (percentage,categoryid) in (select max(percentage)
    ,categoryid from discount group by categoryid having count(*) = (select max
    (a) from (select count(*) as a from discount group by categoryid) as n))
    order by categoryid;

-- add table purchases
create table purchases(
    ProductID int,
    quauntity int not null,
    price int not null,
    primary key(ProductID)
);
alter table purchases add description varchar(40) not null;

-- complex insert queries
insert into product(productid,categoryid,name,cost,description,discount,rating,
    quantity,tag1,tag2,tag3) values(101,5,"tryingSomethingNew",99,"random
    ",0,0,89,3,4,5);

insert into cartitem values((select sessionid from session limit 1), 101,10);
delete from cartitem where ProductID in (select ProductID from (select distinct
     product.productid from cartitem,product where cartitem.ProductID=Product.
    ProductID and Product.CategoryID=5) as a);

insert into cartitem values((select sessionid from session limit 1), 2,10);

insert into employee values((select accountid from users limit 1),100,0);
select name from accounts as acc where acc.accountid in (select users.accountid
     from users,employee where employee.accoun
```

```
tid = users.accountid);

-- update the discount of all products with cost > 500
UPDATE Product
SET Discount=0.5
WHERE Product.Cost > 500;

-- start session for user 1
UPDATE Accounts
SET SessionID=1
WHERE AccountID=1;

-- add product 2 to cart
INSERT INTO CartItem (SessionID, ProductID, Quantity)
VALUES (1, 2, 1);

-- a query that violates constraints
INSERT INTO Employee (AccountID, Salary, isSalaryCredited)
VALUES (999, 100, 0);
--- ERROR 1452 (23000): Cannot add or update a child row: a foreign key
    constraint fails (‘stationaryshop‘.‘employee‘, CONSTRAINT ‘employee_ibfk_1‘
     FOREIGN KEY (‘AccountID‘) REFERENCES ‘accounts‘ (‘AccountID‘))
--- no account with id 999
```

Algebraic Operations

Code:

```
-- gives the tuples from table discount where CategoryID which has most
    occurrence in the table and max discount offered in that category.
SELECT * FROM Product WHERE ProductID not in ( SELECT ProductID FROM ( (SELECT
    ProductID, SessionID FROM (select SessionID from cartitem) as s cross join
    (select ProductID from product) as pi) EXCEPT (SELECT product.ProductID,
    SessionID FROM product,cartitem where product.productid=cartitem.productid)
    ) AS r );
--above is a division query to find products used in all the carts

-- Select all users which have purchased a product with a discount
SELECT Users.AccountID, Users.MembershipID, Product.ProductID, Product.Name,
    Product.Cost, Product.Discount
FROM Users
INNER JOIN Accounts ON Accounts.AccountID=Users.AccountID
INNER JOIN CartItem ON CartItem.SessionID=Accounts.SessionID
```

```
INNER JOIN Product ON Product.ProductID=CartItem.ProductID
WHERE Product.Discount > 0;

-- select users which are also employees

SELECT Users.AccountID, Users.MembershipID, Users.MembershipStartDate, Users.
   MembershipEndDate
FROM Users
INNER JOIN Employee ON Employee.AccountID=Users.AccountID;
```

Constraints

Code:

```
-- a query that violates constraints
INSERT INTO Employee (AccountID, Salary, isSalaryCredited)
VALUES (999, 100, 0);
--- ERROR 1452 (23000): Cannot add or update a child row: a foreign key
   constraint fails (‘stationaryshop‘.‘employee‘, CONSTRAINT ‘employee_ibfk_1‘
    FOREIGN KEY (‘AccountID‘) REFERENCES ‘accounts‘ (‘AccountID‘))
--- no account with id 999
```


CLI Production

CLI with Embedded Queries

Code:

```
import mysql.connector
import hashlib
# connect to mysql database
mydb = mysql.connector.connect(
    host = "localhost",
    user = "root",
    password = "naman1234"
)
# create cursor
mycursor = mydb.cursor()
mycursor.execute("use dbmsNew")

def userLogin(mycursor):
    registered = input("Are you registered user? (y/n) : ")
    if registered == 'n':
        name = input("Enter your name : ")
        email = input("Enter your email : ")
        password = input("Enter your password : ")
        # encrypt password
        password = hashlib.md5(password.encode()).hexdigest()[:30]
        phoneNumber = input("Enter your phone number : ")
        address = input("Enter your address line 1: ")
        address2 = input("Enter your address line 2: ")
        pincode = input("Enter your pincode : ")
        admin = input("Are you an admin? (y/n) : ")
        mycursor.execute("insert into accounts(Name,EmailID,Password,PhoneNumber
            ,Address1,Address2,Pincode,isemployee) values
            ('{}','{}','{}','{}','{}','{}','{}',{})".format(name,email,password,
            phoneNumber,address,address2,pincode,admin))
        if int(admin) == 0:
            print("You have been registered successfully! with membership\n")
            mycursor.execute("insert into users(accountid,membershipid,
                MembershipStartDate,MembershipEndDate) values ((select max(
                accountid) from accounts),1,curdate(),date_add(curdate(),
                interval 1 year))")
            mydb.commit()
        mydb.commit()
        print("You have been registered successfully!\n")
        userLogin(mycursor)

    else:
        email = input("Enter your email : ")
        password = input("Enter your password : ")
```

```
        password = hashlib.md5(password.encode()).hexdigest()[:30]
        mycursor.execute("select * from accounts where EmailID = '{}' and
            Password = '{}'".format(email,password))
        account = mycursor.fetchall()
        if len(account) == 0:
            print("Invalid email or password")
            userLogin(mycursor)
            return False
        adminStatus = account[0][9]
        print("Login successful")
        # check if session already exists
        if account[0][8] == None:
            mycursor.execute("insert into session() values() ")
            mycursor.execute("update accounts set SessionID = (select max(
                SessionID) from session) where EmailID = '{}'".format(email))
            mydb.commit()

        if adminStatus == 1:
            print("Admin pannel")
            adminMenu(mycursor)
            return True
        else:
            print("User pannel")
            userMenu(mycursor,sessionID=account[0][8])
            return True

def userMenu(mycursor,sessionID):
    print("1. View all Categories")
    print("2. View all Products")
    print("3. View Cart Items")
    print("4. Add to Cart")
    print("5. Remove from Cart")
    print("6. Checkout")
    print("7. Logout")
    choice = int(input("Enter your choice : "))
    if choice == 1:
        mycursor.execute("select * from category")
        for categories in mycursor.fetchall():
            for elements in categories:
                print(elements,end=" ")
            print()
        print()
        userMenu(mycursor,sessionID)
        return False
    elif choice == 2:
        mycursor.execute("select * from product")
        for products in mycursor.fetchall():
```

```
            for elements in products:
                print(elements,end=" ")
            print()
        print()
        userMenu(mycursor,sessionID)
        return False
    elif choice == 3:
        mycursor.execute("select * from cartItem where SessionID = {}".format(
            sessionID))
        product = mycursor.fetchall()
        if len(product) == 0:
            print("Cart is empty")
            userMenu(mycursor,sessionID)
            return False
        for cart in product:
            mycursor.execute("select Name from product where ProductID = {}".
                format(cart[1]))
            prodName = mycursor.fetchall()[0][0]
            print("product name: ",prodName," quantity: ",cart[2])
            print()
        print()
        userMenu(mycursor,sessionID)
        return False
    elif choice == 4:
        productID = input("Enter product ID : ")
        quantity = int(input("Enter quantity : "))
        mycursor.execute("select * from product where productID = {}".format(
            productID))
        product = mycursor.fetchall()
        if len(product) == 0 or int(quantity) > int(product[0][7]):
            print("Invalid product ID or quantity")
            userMenu(mycursor,sessionID)
            return False
        else:
            # check if product already exists in cart
            mycursor.execute("select * from cartItem where ProductID = {} and
                SessionID = {}".format(productID,sessionID))
            product = mycursor.fetchall()
            if len(product) != 0:
                mycursor.execute("update cartItem set quantity = quantity + {}
                    where ProductID = {} and SessionID = {}".format(quantity,
                    productID,sessionID))
                mydb.commit()
                print("Product quantity updated successfully!\n")
                userMenu(mycursor,sessionID)
                return False
            mycursor.execute("insert into cartItem(SessionID,ProductID,quantity)
```

```
                values ({},{},{})".format(sessionID,productID,quantity))
            mydb.commit()
            print("Product added to cart successfully!\n")
            userMenu(mycursor,sessionID)
            return False
    elif choice == 5:
        productID = input("Enter product ID : ")
        mycursor.execute("select * from cartItem where ProductID = {} and
            SessionID = {}".format(productID,sessionID))
        product = mycursor.fetchall()
        if len(product) == 0:
            print("Invalid product ID")
            userMenu(mycursor,sessionID)
            return False
        else:
            mycursor.execute("delete from cartItem where ProductID = {} and
                SessionID = {}".format(productID,sessionID))
            mydb.commit()
            print("Product removed from cart successfully!\n")
            userMenu(mycursor,sessionID)
            return False
    elif choice == 6:
        mycursor.execute("select * from cartItem where SessionID = {}".format(
            sessionID))
        cartItems = mycursor.fetchall()
        if len(cartItems) == 0:
            print("Cart is empty")
            userMenu(mycursor,sessionID)
        else:
            mycursor.execute("START Transaction")
            priceTotal = 0
            for cartItem in cartItems:
                mycursor.execute("select cost from product where ProductID = {}".
                    format(cartItem[1]))
                price = mycursor.fetchall()[0][0]
                priceTotal += price*cartItem[2]
                # mycursor.execute("delete from cartItem where SessionID = {}
                    and ProductID = {}".format(sessionID,cartItem[1]))
                mycursor.execute("update cartitem set quantity = 0 where
                    ProductID = {} and sessionid = {}".format(cartItem[1],
                    sessionID))
            print("Total price : ",priceTotal)
            mydb.commit()
            mycursor.execute("COMMIT")

def adminMenu(mycursor):
    print("1. Add new Category")
```

```
    print("2. View all Categories")
    print("3. Add new Product")
    print("4. Delete product")
    print("5. View all products")
    print("6. Logout")
    print("7. view all products which are not in all cart")
    print("8. view total number of products in each category")
    print("9. view membership ending quaterwise")
    print("10. view membership ending quarterwise for active customers")
    # what could be example of rollup query

    choice = int(input("Enter your choice : "))
    if choice == 1:
        categoryName = input("Enter category name : ")
        mycursor.execute("insert into category(Name) values ('{}')".format(
            categoryName))
        mydb.commit()
        print("Category added successfully!\n")
        adminMenu(mycursor)
    elif choice == 2:
        mycursor.execute("select * from category")
        categories = mycursor.fetchall()
        for category in categories:
            print("Category ID: ",category[0],end=" ")
            print("Category Name: ",category[1])
        print()
        adminMenu(mycursor)
    elif choice == 3:
        categoryName = input("Enter category name : ")
        mycursor.execute("select categoryID from category where Name = '{}'".
            format(categoryName))
        category = mycursor.fetchall()
        if len(category) == 0:
            print("Invalid category name")
            adminMenu(mycursor)
        else:
            productName = input("Enter product name : ")
            productPrice = input("Enter product price : ")
            productQuantity = input("Enter product quantity : ")
            productDescription = input("Enter product description : ")
            mycursor.execute("insert into product(Name,Cost,Quantity,Description
                ,CategoryID) values ('{}','{}','{}','{}',{})".format(productName,
                productPrice,productQuantity,productDescription,category[0][0]))
            mydb.commit()
            print("Product added successfully!\n")
            adminMenu(mycursor)
    elif choice == 4:
```

```
        productName = input("Enter product name : ")
        mycursor.execute("select * from product where Name = '{}'".format(
            productName))
        product = mycursor.fetchall()
        if len(product) == 0:
            print("Invalid product name")
            adminMenu(mycursor)
        mycursor.execute("delete from product where Name = '{}'".format(
            productName))
        mydb.commit()
        print("Product deleted successfully!\n")
        adminMenu(mycursor)
    elif choice == 5:
        mycursor.execute("select * from product")
        products = mycursor.fetchall()
        for product in products:
            print("Product ID: ",product[0],end=" ")
            print("Product Name: ",product[2],end=" ")
            print("Product Price: ",product[3],end=" ")
            print("Product Quantity: ",product[7],end=" ")
            print("Category ID: ",product[1])
            # for elements in product:
            # print(elements,end=" ")
            # print()
        print()
        adminMenu(mycursor)
    elif choice == 6:
        # mycursor.execute("delete from session where SessionID = (select max(
            SessionID) from session)")
        # mydb.commit()
        print("Logout successful")
        return True
    elif choice == 7:
        mycursor.execute("select distinct(productid) from (SELECT ProductID,
            SessionID FROM (( select distinct(SessionID) from cartitem) as s
            cross join (select ProductID from product) as pi) EXCEPT (SELECT
            ProductID,SessionID FROM cartitem)) as a")
        products = mycursor.fetchall()
        for product in products:
            mycursor.execute("select * from product where ProductID = {}".format
                (product[0]))
            product = mycursor.fetchall()
            print(product[0])
        print()
        adminMenu(mycursor)
    elif choice == 8:
        mycursor.execute("select category.Name, count(product.ProductID) from
```

```
            category,product where category.categoryID = product.categoryID
            group by category.Name with rollup")
        products = mycursor.fetchall()
        for product in products:
            print(product)
        print()
        adminMenu(mycursor)
    elif choice == 9:
        mycursor.execute("SELECT CONCAT('Q', QUARTER(MembershipEndDate)) AS
            Quarter, COUNT(*) AS MembershipCount FROM Users GROUP BY Quarter
            WITH ROLLUP")
        products = mycursor.fetchall()
        for product in products:
            print(product)
        print()
        adminMenu(mycursor)
    elif choice == 10:
        mycursor.execute("SELECT CONCAT('Q', QUARTER(Users.MembershipEndDate))
            AS Quarter, COUNT(*) AS NumUsers FROM Users JOIN Accounts ON Users.
            AccountID = Accounts.AccountID WHERE Accounts.SessionID IS NOT NULL
            GROUP BY Quarter WITH ROLLUP")
        products = mycursor.fetchall()
        for product in products:
            print(product)
        print()
        adminMenu(mycursor)

userLogin(mycursor)
```

OLAP Queries

Code:

```
select productid from (SELECT ProductID, SessionID
FROM ((select distinct(SessionID) from cartitem) as s cross join (select
   ProductID from product) as pi)
EXCEPT (SELECT ProductID,SessionID FROM cartitem)) as a

select category.Name,count(product.ProductID) from category,product
where category.categoryID = product.categoryID
group by category.Name with rollup

SELECT
```

```
CONCAT('Q', QUARTER(MembershipEndDate)) AS Quarter,
COUNT(*) AS MembershipCount FROM Users
GROUP BY Quarter WITH ROLLUP

SELECT
CONCAT('Q', QUARTER(Users.MembershipEndDate))
AS Quarter, COUNT(*)
AS NumUsers
FROM Users JOIN Accounts ON Users.AccountID = Accounts.AccountID
WHERE Accounts.SessionID IS NOT NULL GROUP BY Quarter WITH ROLLUP;
```

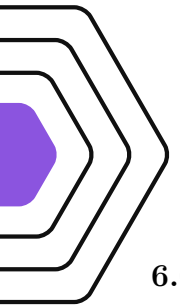
Triggers

Code:

```
create trigger updateQuantity
after insert
on cartItem
for each row
begin
update product set Quantity = Quantity - new.quantity where productID = new.
   productID;
end

create trigger deleteCartItem
after delete
on cartItem
for each row
begin
update product set Quantity = Quantity + old.quantity where productID = old.
   productID;
end
```

Transactions and Final UI



Transactions

6.0.1 Sample Transactions:

We made transactions T1 and T2 to demonstrate the various kinds of transactions, their conflicts and locking mechanisms.

T1
Select quantity from products where productid = 1;
//read(quantity)
Quantity: if quantity>2:continue ? abort
Quantity:= quantity - 2
Update products set quantity = Quantity where productid=1;
//write(quantity)
Insert into order values(productid, quantity)
//write(order)
commit

T2
Select quantity from products where productid = 1;
//read(quantity)
Quantity: if quantity>2:continue ? abort
Quantity:= quantity - 2
Update products set quantity = Quantity where productid=1;
//write(quantity)
Insert into order values(productid, quantity)
//write(order)
commit

6.0.2 Non-Conflicting Serialisable Transactions

T1	**T2**
Select quantity from products where productid = 1; //read(quantity)	
	Select quantity from products where productid = 1; //read(quantity)
	Quantity:= quantity + 2
	Update products set quantity = Quantity where productid=1; //write(quantity)
	commit
Quantity: if quantity>2:continue ? abort	
Quantity:= quantity - 2	
Update products set quantity = Quantity where productid=1; //write(quantity)	
Insert into order values(productid, quantity) //write(order)	
commit	

6.0.3 Non-Conflicting Locking Transactions

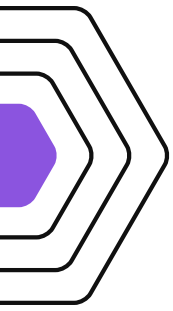
T1	**T2**
LOCK-S(quantity)	
read(quantity)	
unlock(quantity)	
	LOCK-X(quantity)
	read(quantity)
	Quantity:= quantity + 2
	write(quantity)
	unlock(quantity)
LOCK-X(quantity)	
Quantity: if quantity>2:continue ? abort	
Quantity:= quantity - 2	
write(quantity)	
unlock(quantity)	
LOCK-X(order)	
write(order)	
unlock(order)	

6.0.4 Conflicting Serialisable Transactions

T1	T2
Select quantity from products where productid = 1; //read(quantity)	
Quantity: if quantity>2:continue ? abort	
Quantity:= quantity - 2	
Update products set quantity = Quantity where productid=1; //write(quantity)	
	Select quantity from products where productid = 1; //read(quantity)
	Quantity:= quantity + 2
	Update products set quantity = Quantity where productid=1; //write(quantity)
	commit
Insert into order values(productid, quantity) //write(order)	
commit	

6.0.5 Conflicting Locking Transactions

T1	T2
LOCK-X(quantity)	
read(quantity)	
Quantity: if quantity>2:continue ? abort	
Quantity:= quantity - 2	
write(quantity)	
unlock(quantity)	
	LOCK-X(quantity)
	read(quantity)
	Quantity:= quantity + 2
	write(quantity)
	unlock(quantity)
LOCK-X(order)	
write(order)	
unlock(order)	



User Interface

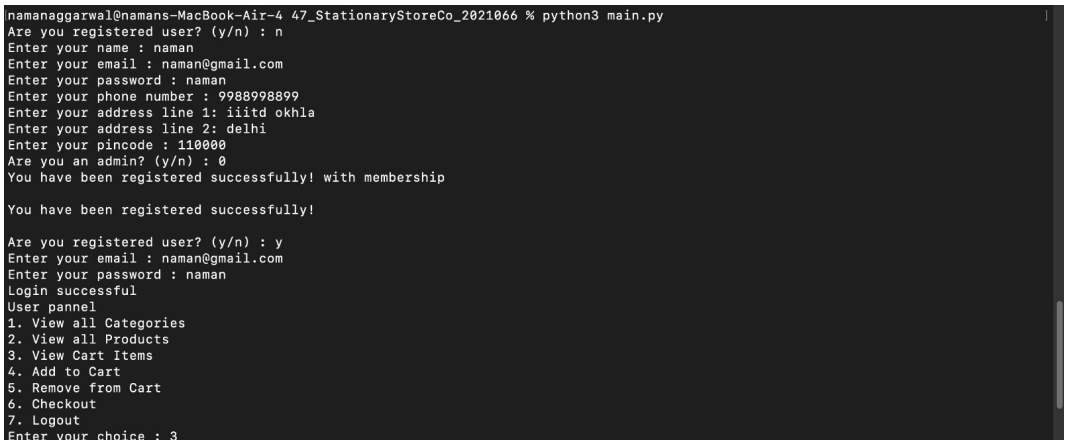


Figure 6.1: User registration.

Figure 6.2: User panel.

How to use?

For seller:

1. Configure `main.py` with the username and password to your MySQL instance
2. Run `python3 main.py`
3. Type 'y' and press Enter.
4. Login with the username password combination: `admin:admin`. This can be changed in `main.py` according to your needs.
5. You will be presented with a menu through which you can select options to add, modify, view and delete products.
6. You can use options 7-10 to generate reports of OLAP Queries
7. Type `6` and press Enter to logout.

For buyer:

Figure 6.3: Admin Login and Admin Panel.

Figure 6.4: User order placed.

1. Netcat into the server using `nc <ip address>`
2. If you haven't registered, type '`n`' and press Enter. You will be asked foir yout details for registration
3. Otherwise, type '`y`' and press Enter to login with your credentials.
4. Once registered, you will be presented with a menu through which you can select options to view prodcuts and add them to your cart or remove them from your cart and process your order
5. Type `6` and press Enter to checkout your order. Total price will be displayed
6. Type `7` and press Enter to logout.

In an alternate world, there exists a mobile shop.