**PROJECT REPORT**

**Submitted for :**

**DATABASE MANAGEMENT SYSTEM (UCS310)**

|  |  |
| --- | --- |
| **SUBMITTED BY** | **ROLL NUMBER** |
| **VASHISTHA** | **102203447** |
| **SAMARTH SINGH** | **102203487** |
| **GAUTAM** | **102203498** |
| **VIDHI GOYAL** | **102203659** |

**TOPIC: VIRTUAL TRADING PLATFORM**

**Submitted to-**

**Mr. Shashank Singh**



**Computer Science and Engineering Department TIET, Patiala**

**(JAN-MAY, 2024)**

**ABSTRACT**

**Development and Implementation of a Virtual Trading Platform using MySQL and Python**

**The abstract of this report encapsulates the development and implementation of a virtual trading platform leveraging MySQL and Python. In an increasingly digitalized world, virtual trading platforms offer individuals a risk-free environment to hone their investment skills, test strategies, and explore financial markets. This report outlines the architectural design, functionality, and implementation details of the virtual trading platform.**

**The platform's foundation rests upon the integration of MySQL, a robust relational database management system, and Python, a versatile programming language renowned for its simplicity and efficiency. MySQL serves as the backbone for storing and managing crucial data such as user information, stock market data, transaction history, and portfolio details.**

**Throughout the development process, emphasis was placed on scalability, security, and user experience. MySQL's scalability features ensure the platform can accommodate a growing user base and increasing volumes of data. Security measures, including encryption protocols and secure authentication mechanisms, safeguard users' sensitive information and transactions. User experience enhancements, such as intuitive navigation, responsive design, and real-time updates, contribute to a seamless and engaging trading environment.**

**INDEX**

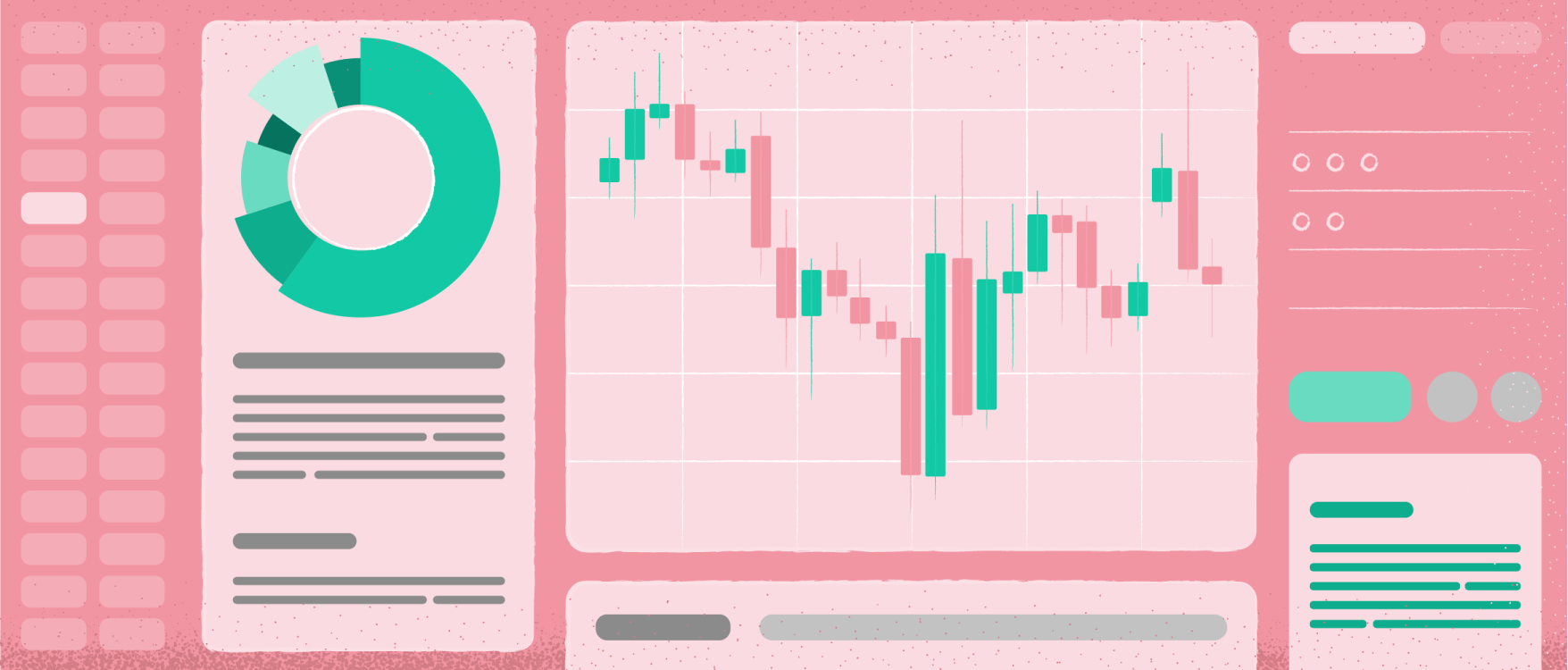
|  |  |  |
| --- | --- | --- |
| **SR NO** | **CONTENT** | **PAGE** |
| **1** | **INTRODUCTION** | **4** |
| **2** | **ER DIAGRAM** | **5** |
| **3** | **ER TO TABLE** | **6** |
| **4** | **NORMALIZATION** | **7** |
| **5** | **PROJECT CODE** | **8-23** |
| **6** | **OUTPUT VS CODE** | **24-29** |
| **7** | **OUTPUT DATABASE** | **30-34** |
| **8** | **CONCLUSION** | **35** |

**INTRODUCTION**

**A trading platform is a software system used to trade securities. It allows investors to open, close, and manage market positions online through a financial intermediary, such as an online broker.**

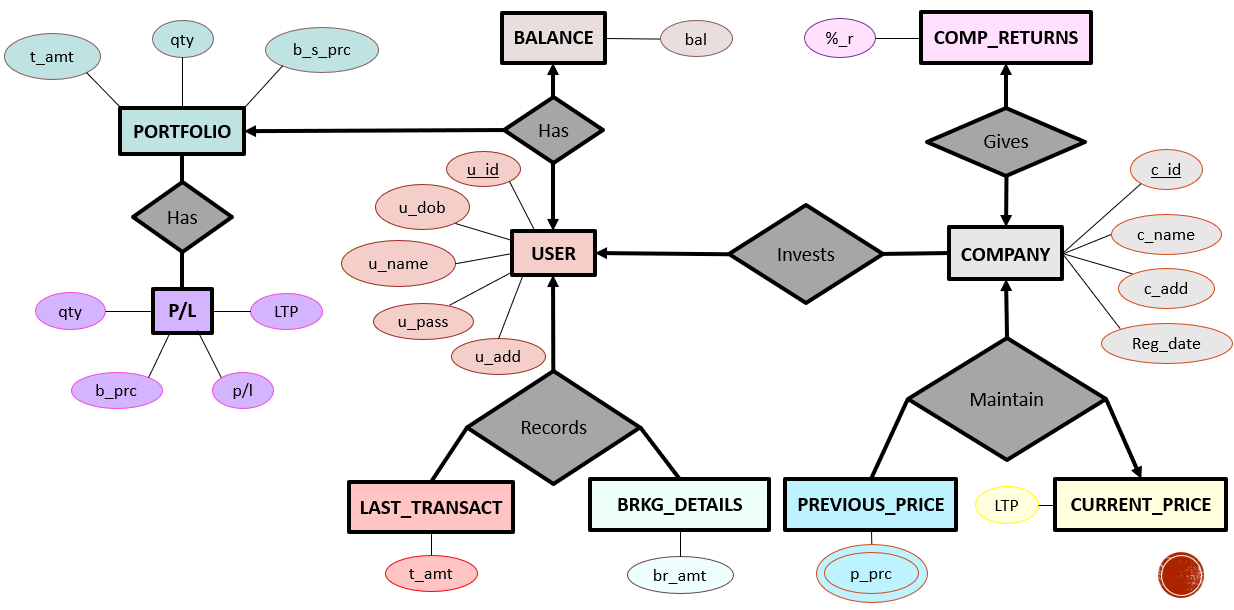
**Online trading platforms are frequently offered by brokers either for free or at a discount in exchange for maintaining a funded account and/or making a specified number of trades per month. The best trading platforms offer a mix of robust features and low fees.**

**Online Trading is a method that facilitates buying and selling of financial instruments such as mutual funds, equities, bonds, Sovereign gold bonds, derivatives, stocks, ETFs and commodities through an electronic interface. Online Trading has simplified a complex process into a few clicks.**

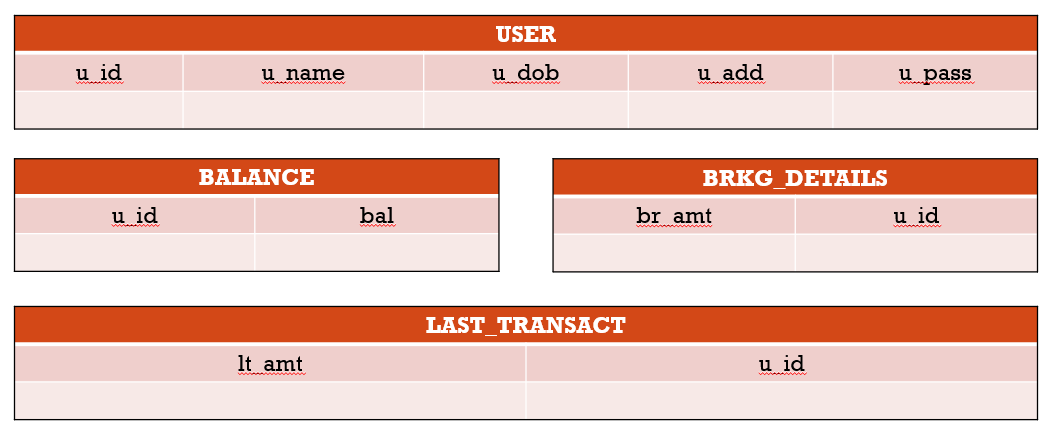


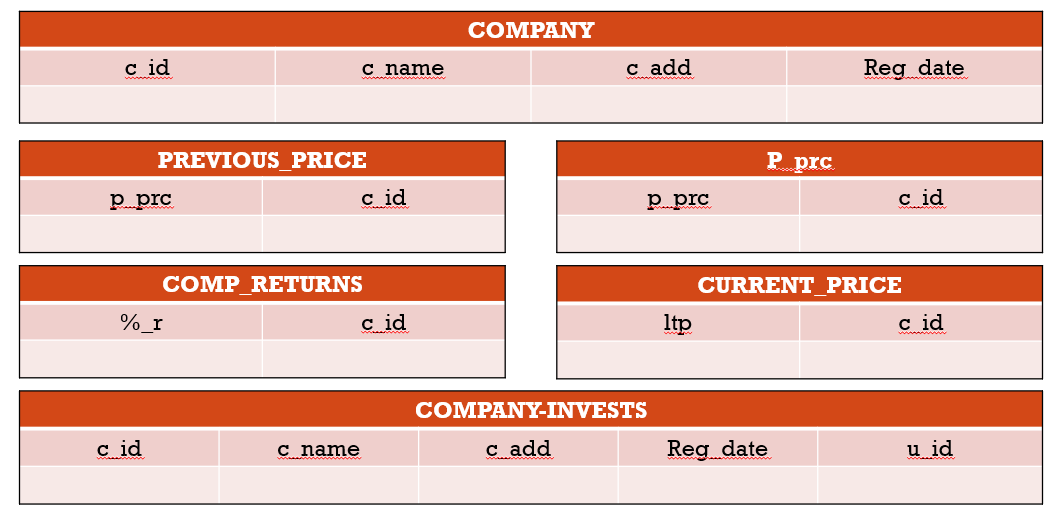
**ER Diagram**

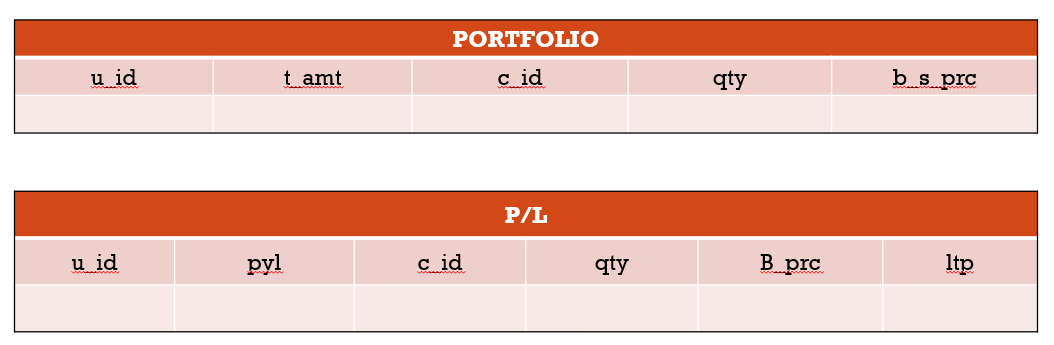
**An Entity-Relationship (ER) diagram is a visual representation of the data model that describes how entities are related to each other within a system. It's a conceptual modeling technique used in software engineering to design databases in a graphical format.**



**ER TO TABLES**







**NORMALIZATION**

**Normalization is a database design technique used to organize tables and their relationships in a way that reduces redundancy and dependency. In this code, normalization principles are applied in several places:**

1. **User Information (user\_info) Table:**

**The user\_info table is in the third normal form. There are no transitive dependencies present. Each non-key attribute (columns other than the primary key) is dependent only on the primary key U\_id.**

1. **Company Information (company\_info) Table:**

**Similarly, the company\_info table is in the third normal form. All non-key attributes are directly dependent on the primary key C\_id.**

1. **Portfolio Table:**

**The portfolio table, which maintains users' stock holdings, appears to be in 3NF. The attributes such as B\_S\_price, Qty, and Total\_amt are directly related to the primary key (C\_id, U\_id), and there are no transitive dependencies.**

**Normalization to 3NF ensures that there are no non-prime attributes dependent on other non-prime attributes, i.e., all attributes are functionally dependent only on the primary key. This reduces data redundancy and helps maintain data integrity.**

**PROJECT CODE**

import mysql.connector

import random

import matplotlib.pyplot as plt

p\_count=0

mydb = mysql.connector.connect(

  host="localhost",

  user="root",

  password="GAUTAM",

  database="TRADING\_PLATFORM"    )

mycursor = mydb.cursor()

price\_list=[]

mycursor.execute("CREATE DATABASE TRADING\_PLATFORM")

mycursor.execute("CREATE TABLE user\_info (U\_name VARCHAR(255),U\_id int primary key, U\_address VARCHAR(255),U\_dob DATE,U\_pass VARCHAR(255))")

mycursor.execute("CREATE TABLE Brkg\_details (U\_id int, Br\_amt int , FOREIGN KEY (U\_id) REFERENCES user\_info(U\_id))")

mycursor.execute("CREATE TABLE Comp\_returns (C\_id int, Prec\_return int , FOREIGN KEY (C\_id) REFERENCES company\_info(C\_id))")

mycursor.execute("CREATE TABLE PL (U\_id int,C\_id int , Qty int , P\_L int , LTP\_buy\_price int , Sell\_Price int)")

mycursor.execute("CREATE TABLE Last\_Transact (U\_id int ,T\_amt int , FOREIGN KEY (U\_id) REFERENCES user\_info(U\_id))")

mycursor.execute("CREATE TABLE Prev\_Price (C\_id int, Pre\_Price int , FOREIGN KEY (C\_id) REFERENCES company\_info(C\_id))")

mycursor.execute("CREATE TABLE company\_info (C\_name VARCHAR(255),C\_id int primary key , C\_address VARCHAR(255),Reg\_date DATE)")

mycursor.execute("CREATE TABLE Balance (U\_id int, Balance int , FOREIGN KEY (U\_id) REFERENCES user\_info(U\_id))")

mycursor.execute("CREATE TABLE Stock\_price\_current (C\_id int, LTP int , FOREIGN KEY (C\_id) REFERENCES company\_info(C\_id))")

mycursor.execute("CREATE TABLE portfolio (C\_id int,U\_id int , B\_S\_price int ,Qty int,Total\_amt int, FOREIGN KEY (C\_id) REFERENCES company\_info(C\_id),FOREIGN KEY (U\_id) REFERENCES user\_info(U\_id))")

def add\_user():

    print("Welcome to the PLATFORM")

    uname = input("Enter the User Name: ")

    uid = int(input("Enter the User Id: "))

    udob = input("Enter the Dob: ")

    uadd = input("Enter the address: ")

    upass = input("Enter the password: ")

    sql = "INSERT INTO user\_info (U\_name, U\_id, U\_address, U\_dob, U\_pass) VALUES (%s, %s, %s, %s, %s)"

    val = (uname, uid, uadd, udob, upass)

    mycursor.execute(sql, val)

    mydb.commit()

    print("User added successfully!")

    print()

    sql = "INSERT INTO Balance (U\_id,Balance) VALUES (%s, %s)"

    val = (uid,0)

    mycursor.execute(sql, val)

    mydb.commit()

def mod\_user():

    uid=int(input("Enter the uid to modify : "))

    uname = input("Enter the New User Name: ")

    udob = input("Enter the New Dob: ")

    uadd = input("Enter the New address: ")

    upass = input("Enter the New password: ")

    sql="update user\_info set U\_name=%s ,U\_dob=%s,U\_address=%s,U\_pass=%s where U\_id=%s"

    val=(uname,udob,uadd,upass,uid)

    mycursor.execute(sql, val)

    mydb.commit()

    print("User modified successfully!")

    print()

def add\_comapny():

    print("Welcome to the PLATFORM")

    cname = input("Enter the Comapny Name: ")

    cid = int(input("Enter the Comapny Id: "))

    cdate = input("Enter the Reg Date : ")

    cadd = input("Enter the Company address: ")

    price=int(input("Enter the listed(initial) price : "))

    sql = "INSERT INTO company\_info (C\_name, C\_id, C\_address, Reg\_date) VALUES (%s, %s, %s, %s)"

    val = (cname, cid, cadd, cdate)

    mycursor.execute(sql, val)

    mydb.commit()

    print("Company added successfully!")

    print()

    sql = "INSERT INTO Stock\_price\_current (C\_id,LTP) VALUES (%s, %s)"

    val = (cid,price)

    mycursor.execute(sql, val)

    mydb.commit()

def mod\_company():

    cid=int(input("Enter the cid to modify : "))

    cname = input("Enter the New Company Name: ")

    cdate = input("Enter the New Reg date: ")

    cadd = input("Enter the New address: ")

    sql="update company\_info set C\_name=%s ,Reg\_date=%s,C\_address=%s where C\_id=%s"

    val=(cname,cdate,cadd,cid)

    mycursor.execute(sql, val)

    mydb.commit()

    print("Company modified successfully!")

    print()

def add\_balance():

    uid=input("Enter the User id to add money : ")

    amt=int(input("Enter the amount to be added : "))

    sql="select Balance from Balance where U\_id=%s"

    mycursor.execute(sql, (uid,))

    prev=mycursor.fetchone()

    prev\_bal=int(prev[0])

    new\_bal=prev\_bal+amt

    sql="update Balance set Balance=%s where U\_id=%s"

    val=(new\_bal,uid)

    mycursor.execute(sql, val)

    mydb.commit()

    print("Money Added successfully!")

    print()

    sql = "INSERT INTO last\_transact (U\_id,T\_amt) VALUES (%s, %s)"

    val = (uid, amt)

    mycursor.execute(sql, val)

    mydb.commit()

def withdraw\_bal():

    uid=input("Enter the User id  : ")

    amt=int(input("Enter the amount to withdraw : "))

    sql="select Balance from Balance where U\_id=%s"

    mycursor.execute(sql, (uid,))

    cur=mycursor.fetchone()

    cur\_bal=int(cur[0])

    if cur\_bal > amt:

        new\_bal=cur\_bal-amt

        sql="update Balance set Balance=%s where U\_id=%s"

        val=(new\_bal,uid)

        mycursor.execute(sql, val)

        mydb.commit()

        print("Money Withdrawed successfully!")

        sql = "INSERT INTO last\_transact (U\_id,T\_amt) VALUES (%s, %s)"

        val = (uid, -amt)

        mycursor.execute(sql,val)

        mydb.commit()

    else:

        print("Insufficient Money in account ")

def update\_ltp():

    cid=int(input("enter the Cid : "))

    price\_move = random.randint(1, 15) \* 5

    price\_trend=random.randint(0,1)

    arr=["+","-"]

    char=arr[price\_trend]

    sql="select LTP from Stock\_price\_current where C\_id=%s"

    mycursor.execute(sql, (cid,))

    cur=mycursor.fetchone()

    prev\_price=int(cur[0])

    if(char == '+'):

        new\_p=prev\_price+price\_move

    else:

        new\_p=prev\_price-price\_move

    sql="update Stock\_price\_current set LTP=%s where C\_id=%s"

    val=(new\_p,cid)

    mycursor.execute(sql, val)

    mydb.commit()

    price\_list.append(new\_p)

    global p\_count;

    p\_count=p\_count+1

    sql="select C\_name from company\_info where C\_id=%s"

    mycursor.execute(sql, (cid,))

    cur=mycursor.fetchone()

    print("Current stock price of company :",cur[0], " : ",new\_p)

    sql = "INSERT INTO prev\_price (C\_id,Pre\_Price) VALUES (%s, %s)"

    val = ( cid,new\_p)

    mycursor.execute(sql, val)

    mydb.commit()

def Chart():

    update\_ltp()

    update\_ltp()

    update\_ltp()

    update\_ltp()

    update\_ltp()

    y=price\_list

    x=[]

    for i in range(0,p\_count,1):

        x.append(i\*5)

    plt.plot(x, y)

    plt.xlabel('TIME')

    plt.ylabel('PRICE')

    plt.title('PRICE CHART')

    plt.show()

def buy\_share():

    uid=int(input("Enter the user id :"))

    cid=input("Enter the company id : ")

    qty=int(input("Enter the qty :"))

    sql="select LTP  from Stock\_price\_current where C\_id=%s"

    mycursor.execute(sql, (cid,))

    cur=mycursor.fetchone()

    bs\_price=cur[0]

    f\_amt=bs\_price\*qty

    sql="select Balance from Balance where U\_id=%s"

    mycursor.execute(sql, (uid,))

    cur=mycursor.fetchone()

    ava\_val=int(cur[0])

    print("Brokerage : 5 %")

    bro\_amt=0.05\*f\_amt

    if(ava\_val > f\_amt+bro\_amt):

        sql = "INSERT INTO portfolio (C\_id,U\_id,B\_S\_price,Qty,Total\_amt) VALUES (%s, %s, %s, %s,%s)"

        val = ( cid,uid,bs\_price,qty,f\_amt)

        mycursor.execute(sql, val)

        mydb.commit()

        print("Share purchased successfully!")

        print("")

        sql="update Balance set Balance=%s where U\_id=%s"

        r\_bal=ava\_val-(f\_amt+bro\_amt)

        val=(r\_bal,uid)

        mycursor.execute(sql, val)

        mydb.commit()

        print("Balance updated successfully!")

        print("")

        sql = "INSERT INTO Brkg\_details (U\_id,Br\_amt) VALUES (%s, %s)"

        val = ( uid,bro\_amt)

        mycursor.execute(sql, val)

        mydb.commit()

        print("Brokerage Received")

        pl\_amt=0

        sp=0

        sql = "INSERT INTO pl (U\_id,C\_id,Qty,P\_L,LTP\_buy\_price,Sell\_price) VALUES (%s,%s, %s, %s, %s,%s)"

        val = ( uid,cid,qty,pl\_amt,bs\_price,sp)

        mycursor.execute(sql, val)

        mydb.commit()

        sql = "INSERT INTO last\_transact (U\_id,T\_amt) VALUES (%s, %s)"

        val = (uid, -(f\_amt+bro\_amt))

        mycursor.execute(sql, val)

        mydb.commit()

    else :

        print("Insufficient Money")

def sell\_share():

    uid=int(input("Enter the user id :"))

    cid=input("Enter the company id : ")

    qty=int(input("Enter the qty to sell :"))

    sql="select Qty from portfolio where U\_id=%s and C\_id=%s "

    val=(uid,cid)

    mycursor.execute(sql, val)

    cur=mycursor.fetchone()

    ava\_qty=int(cur[0])

    print("Brokerage : 2.5 %")

    update\_ltp()

    sql="select LTP from Stock\_price\_current where C\_id=%s"

    mycursor.execute(sql, (cid,))

    cur=mycursor.fetchone()

    c\_price=int(cur[0])

    if(ava\_qty >= qty):

        amt\_with=c\_price\*qty

        bro\_amt=0.025\*amt\_with

        sql="select Balance from Balance where U\_id=%s "

        value=(uid,)

        mycursor.execute(sql, value)

        cur=mycursor.fetchone()

        ava\_val=int(cur[0])

        sql="update Balance set Balance=%s where U\_id=%s"

        r\_bal=ava\_val+amt\_with-bro\_amt

        val=(r\_bal,uid)

        mycursor.execute(sql, val)

        mydb.commit()

        print("Balance updated successfully!")

        print("")

        sql="update portfolio set  Qty=%s where U\_id=%s and C\_id=%s "

        new\_qty=ava\_qty-qty

        val=(new\_qty,uid,cid)

        mycursor.execute(sql, val)

        print("Qty updated successfully !!")

        mydb.commit()

        sql="select  LTP\_buy\_price from pl where U\_id=%s and C\_id=%s"

        value=(uid,cid)

        mycursor.execute(sql, value)

        cur=mycursor.fetchone()

        bp\_amt=int(cur[0])

        plamt=(c\_price-bp\_amt)\*qty

        sql="update pl set sell\_price=%s, p\_l=%s,qty=%s where U\_id=%s and C\_id=%s "

        val=(c\_price,plamt,new\_qty,uid,cid)

        mycursor.execute(sql, val)

        print("P/L Statement updated successfully !!")

        mydb.commit()

        sql = "INSERT INTO last\_transact (U\_id,T\_amt) VALUES (%s, %s)"

        val = (uid, (amt\_with-bro\_amt))

        mycursor.execute(sql, val)

        mydb.commit()

    else:

        print("Insufficient Shares")

def Comapny\_return():

    cid=int(input("Enter the company id : "))

    sql="select  LTP\_buy\_price from pl where C\_id=%s"

    value=(cid,)

    mycursor.execute(sql, value)

    cur=mycursor.fetchone()

    buy\_amt=int(cur[0])

    sql="select  Sell\_Price from pl where C\_id=%s"

    value=(cid,)

    mycursor.execute(sql, value)

    cur=mycursor.fetchone()

    sell\_amt=int(cur[0])

    per=((sell\_amt-buy\_amt)/buy\_amt)\*100

    sql = "INSERT INTO comp\_returns (C\_id,Prec\_return) VALUES (%s, %s)"

    val = (cid,per)

    mycursor.execute(sql, val)

    mydb.commit()

    print("Percentage return is : ", per , "%")

def Display\_Previous\_price():

    cid=int(input("Enter the cid : "))

    sql="select  C\_name from company\_info where C\_id=%s "

    value=(cid,)

    mycursor.execute(sql, value)

    cur=mycursor.fetchone()

    print("Company :  ",cur[0])

    print("Company id : ",cid )

    sql="select  Pre\_Price from prev\_price where C\_id=%s"

    value=(cid,)

    mycursor.execute(sql, value)

    cur=mycursor.fetchall()

    for i in cur:

        print("Price ",i[0])

def login():

    print("Enter the following details for login : ")

    ui=int(input("Enter the use id :"))

    pas=input("Enter the pass :")

    sql="select U\_pass from user\_info where U\_id=%s"

    value=(ui,)

    mycursor.execute(sql, value)

    oe\_pass=mycursor.fetchone()

    l\_pass=oe\_pass[0]

    if(l\_pass == pas):

        print("Logged in successfully")

        return 1

    else:

        print("INVALID DETAILS")

        return 0

print(" $$ WELCOME TO THE VIRTUAL TRADING PLATFORM  $$ ")

print("Making the DEMO USER : ")

add\_user()

a=login()

while(a):

    print("Enter the option to proceed with the platform :- ")

    print("Option 1 : TO ADD NEW USER" )

    print("")

    print("Option 2 : TO MODIFY USER" )

    print("")

    print("Option 3 : TO ADD NEW COMPANY" )

    print("")

    print("Option 4 : TO MODIFY COMPANY" )

    print("")

    print("Option 5 : TO ADD BALANCE" )

    print("")

    print("Option 6 : TO WITHDRAW BALANCE" )

    print("")

    print("Option 7 : TO SHOW LTP OF A STOCK" )

    print("")

    print("Option 8 : TO DISPLAY PRICE CHART" )

    print("")

    print("Option 9 : TO BUY SHARE" )

    print("")

    print("Option 10 : TO SELL SHARE" )

    print("")

    print("Option 11 : TO DISPLAY COMPANY RETURNS" )

    print("")

    print("Option 12 : TO DISPLAY PREVIOUS PRICE OF A STOCK" )

    print("")

    print("Option 13 : TO LOGOUT" )

    print("")

    op=int(input("Enter Desired option : "))

    match op:

        case 1:

            add\_user()

        case 2:

            mod\_user()

        case 3:

            add\_comapny()

        case 4 :

            mod\_company()

        case 5 :

            add\_balance()

        case 6 :

            withdraw\_bal()

        case 7:

            update\_ltp()

        case 8:

            Chart()

        case 9:

            buy\_share()

        case 10 :

            sell\_share()

        case 11:

            Comapny\_return()

        case 12:

            Display\_Previous\_price()

        case 13 :

            a=0

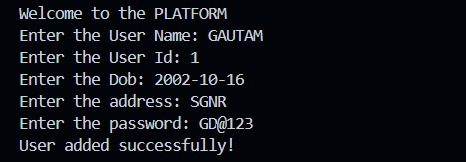
            print("Logged out successfully")

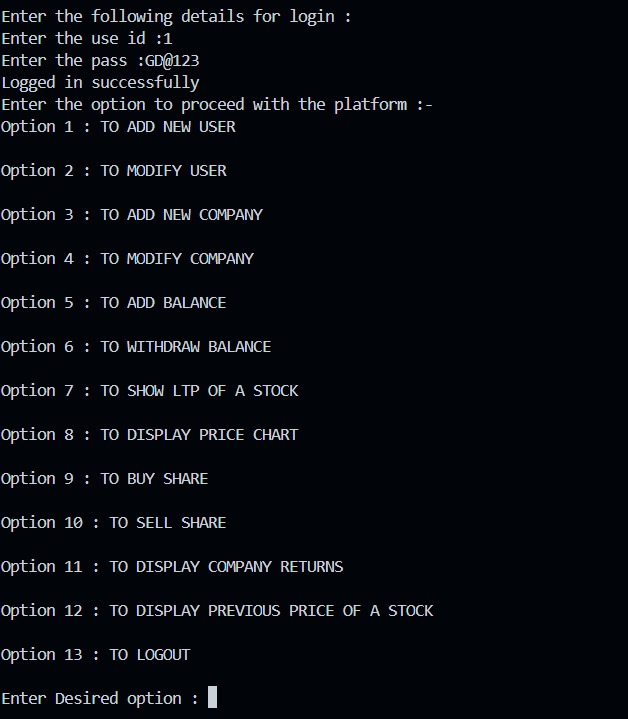
        case \_ :

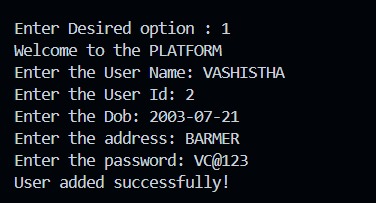
            print("Invalid option selected :")

**OUTPUT (VS CODE)**

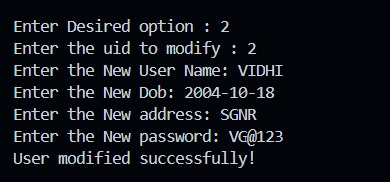
1. **ADD USER**



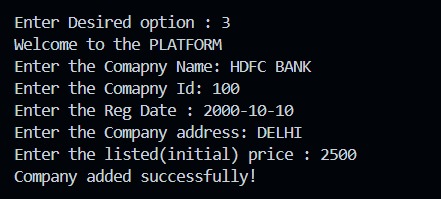




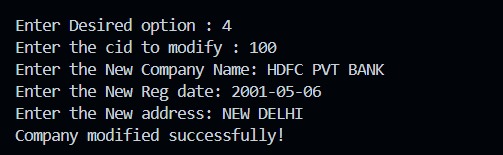
1. **MODIFY USER**



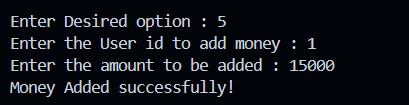
1. **ADD COMPANY**



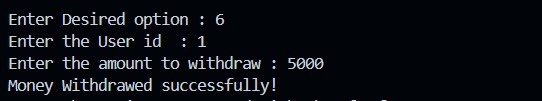
1. **MODIFY COMPANY**



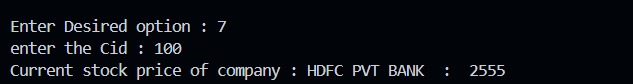
1. **ADD BALANCE**



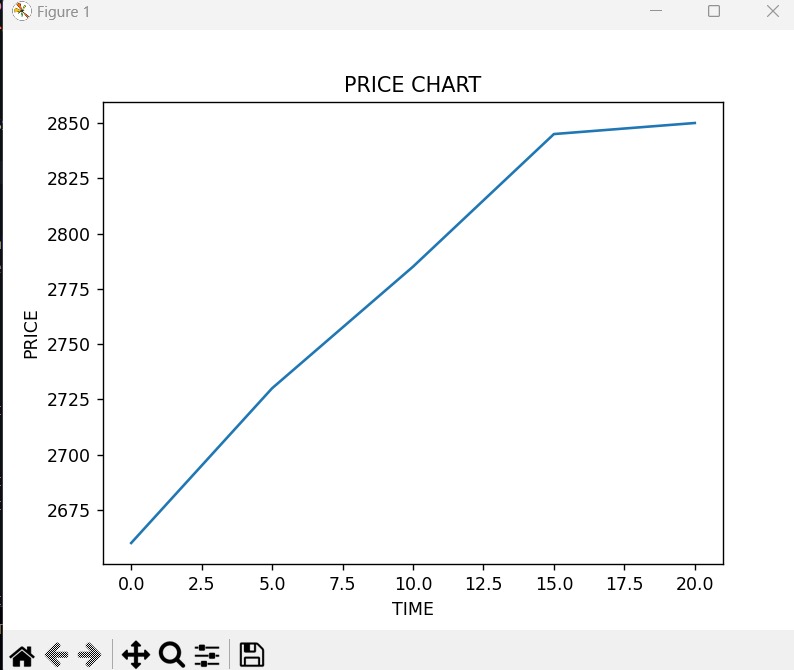
1. **WITHDRAW BALANCE**

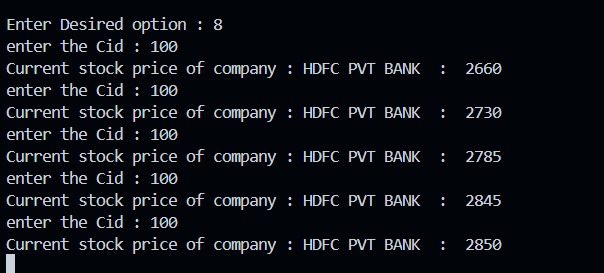


1. **DISPLAY LTP**

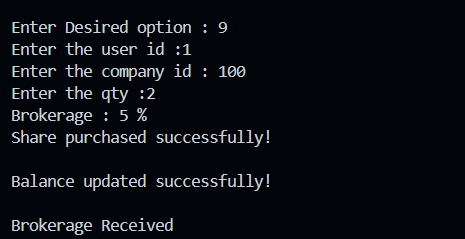


1. **CHART**

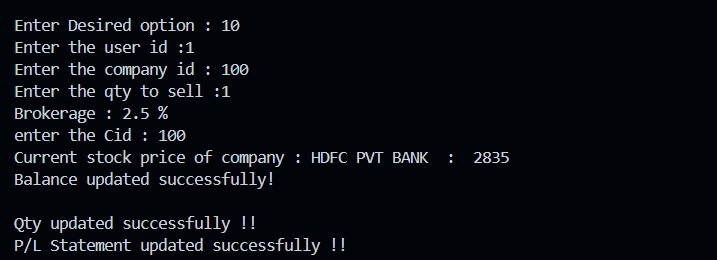




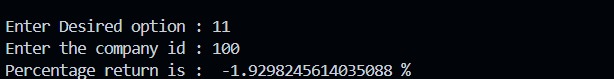
1. **BUY SHARE**



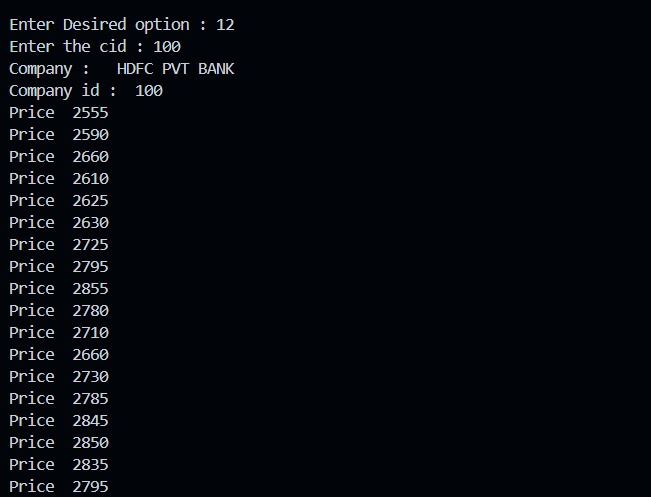
1. **SELL SHARE**



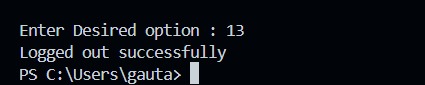
1. **COMPANY RETURN**



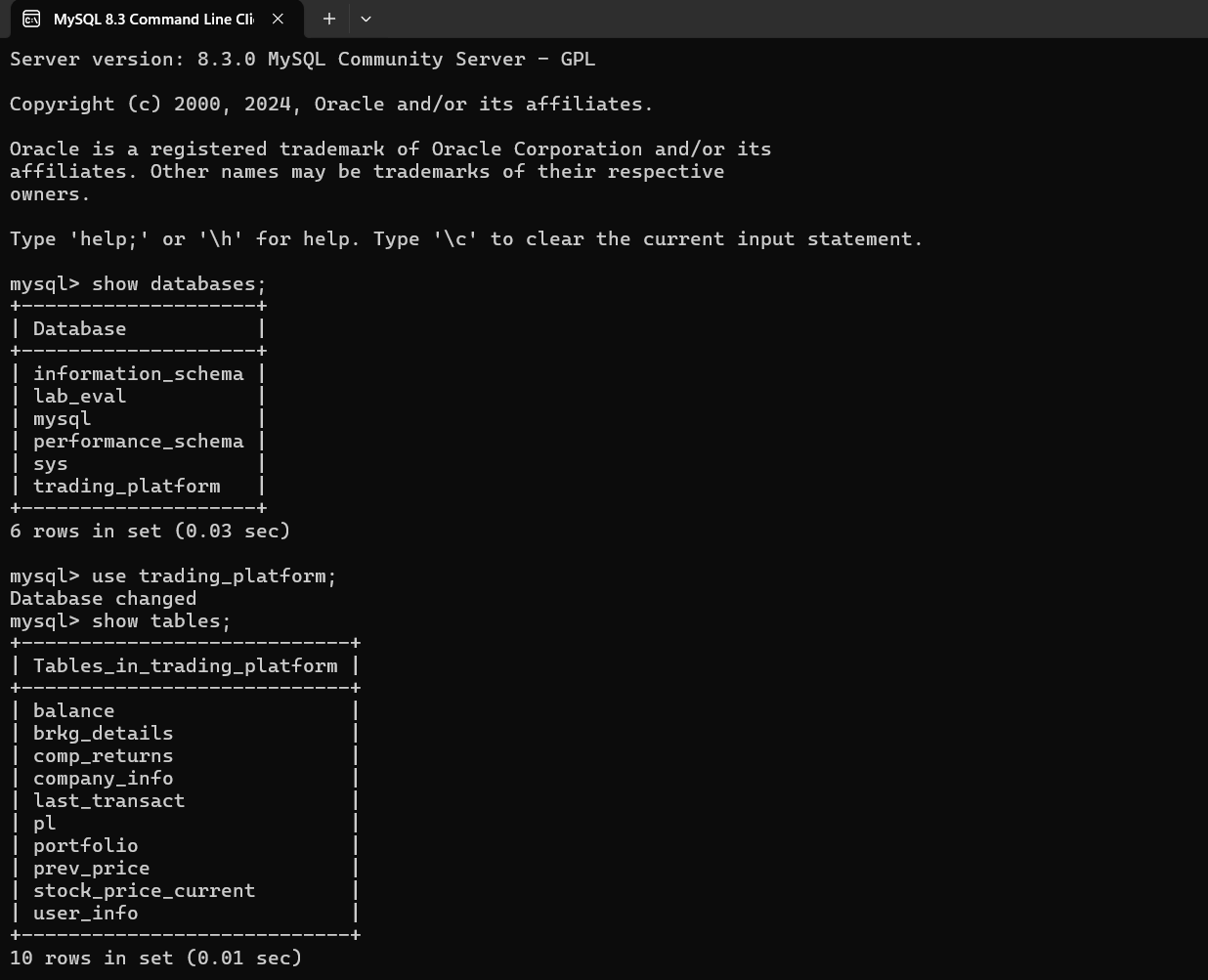
1. **DISPLAY PREVIOUS PRICES**

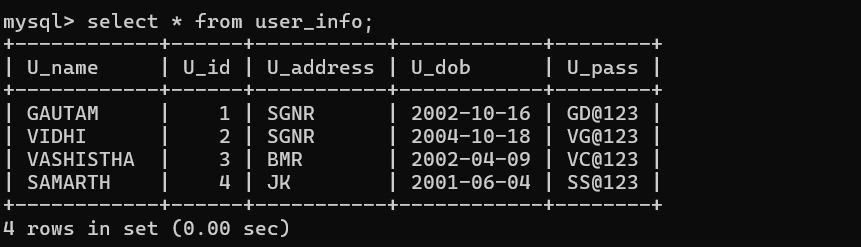


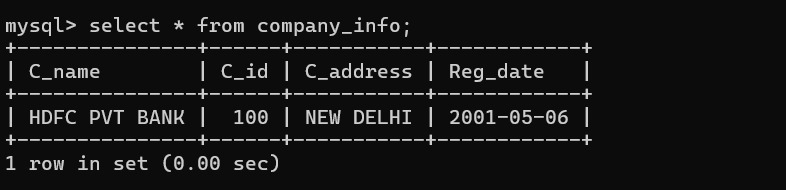
1. **LOGOUT**

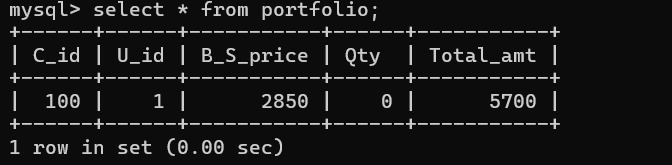


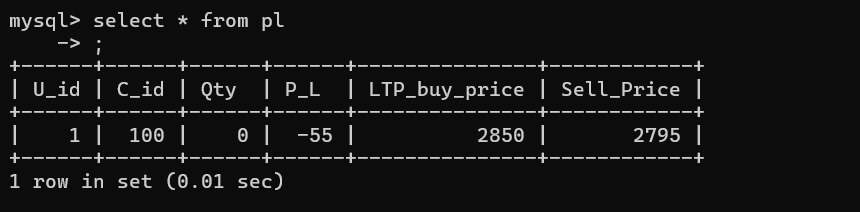
**OUTPUT (DATABASE)**

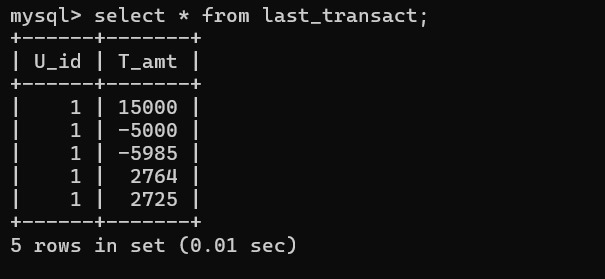


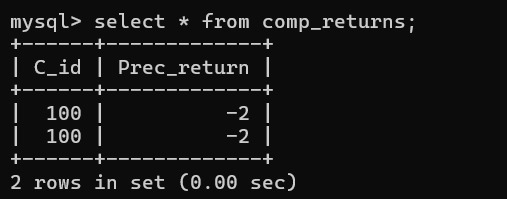


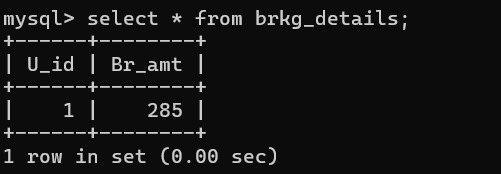


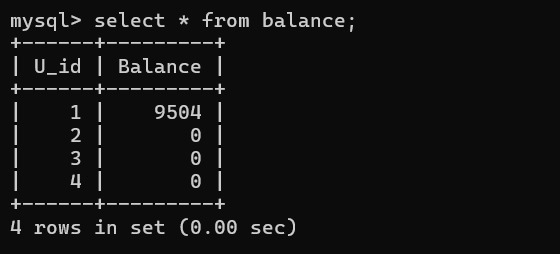


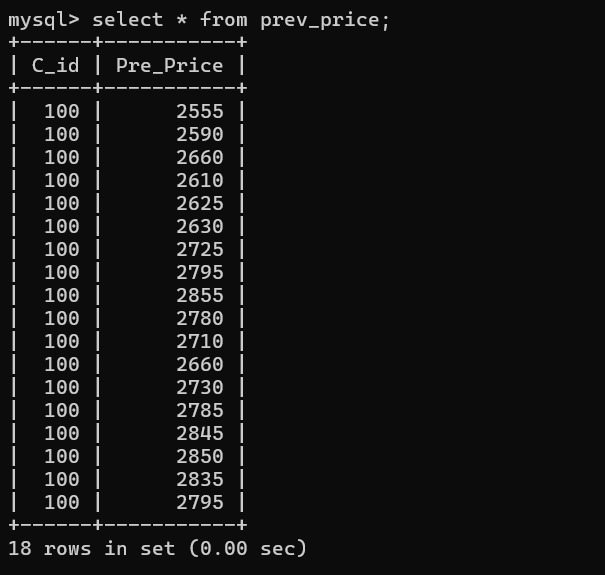


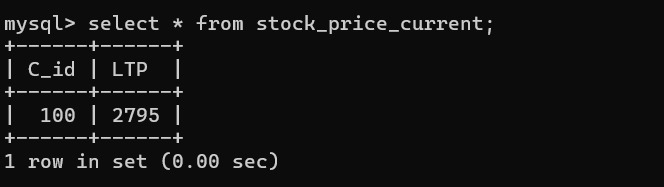












**CONCLUSION**

**In conclusion, the development and implementation of this virtual trading platform underscore the fusion of MySQL and Python as powerful tools for creating innovative financial applications. By providing users with a realistic yet risk-free avenue for exploring financial markets, the platform aims to democratize access to investment education and empower individuals to make informed financial decisions.**

