ASSIGNMENT – 4

DBMS

ACCESSI–SPHERE

PALAK BHARDWAJ (2022344)-1 Trigger, embedded queries, admin login for CLI
SWARNIMA PRASAD (2022525)- 1 Trigger, embedded queries, user login for CLI

1. Triggers

Trigger 1

```
DELIMITER //

-- Create the trigger

CREATE TRIGGER UpdateOrderStatus

BEFORE INSERT ON OrderTable

FOR EACH ROW

BEGIN

   -- Set the OrderStatus to 'Pending' for the newly inserted order

   SET NEW.OrderStatus = 'Pending';

   -- UPDATE OrderTable

--    SET OrderStatus = 'Pending'

--    WHERE OrderID = NEW.OrderID;

END;

//
```

```
DELIMITER ;
```

Trigger 2

```
DROP TRIGGER IF EXISTS block_user;

DELIMITER //

CREATE TRIGGER block_user

AFTER INSERT ON login_attempts

FOR EACH ROW

BEGIN

   DECLARE attempts INT;

   SELECT COUNT(*) INTO attempts FROM login_attempts WHERE UserID = NEW.UserID;

   IF attempts >= 3 THEN

      UPDATE users SET account_status = 'Blocked' WHERE UserID = NEW.UserID;

   END IF;

END;

//

DELIMITER ;
```

2. Command Line Interface code

```
import mysql.connector

import re

from datetime import date


usernumber = 22
```

```
transactionno=2002

cnx = mysql.connector.connect(

    user="root", host="localhost", password="$Uvy1201&5!",
database="accessisphere"

)

mycursor = cnx.cursor()



def validate_email(email):

    pattern = re.compile(r"[^@]+@[^@]+\.[^@]+")

    return pattern.match(email)



def login():

    print("^" * 50)

    print("Welcome Admin")

    print("^" * 50)

    print("Please Enter Valid Username and Password")

    admin_id = str(input("enter username:"))

    password = str(input("enter password:"))

    try:

        query = "Select * From Admin where AdminID='{}' and
Password='{}'".format(
```

```
            admin_id, password
        )
        mycursor.execute(query)
        rec = mycursor.fetchall()
        cnx.commit()

        if rec:
            print("\tACCESS GRANTED")


    except mysql.connector.Error as err:
        print("Error:", err)
    # else:
    #     print("-" * 50)
    #     print("\tWrong credentials")
    #     print("-" * 50)
    #     return False


def after_login(userid):
    global transactionno

    # print("3.See all product categories")
```

```
    # print("4.See all products under a productcategory")
    # print("5.See products in cart")
    # print("6.Order products from your cart")
    # print("7.Exit")
    while True:
        print("1.View products")
        print("2.View Cart")
        # print("3.Search Product")
        cart_id = userid
        print("3.Logout")
        inuu = int(input())
        if inuu == 1:
            mycursor.execute("select distinct(category) from
product")
            index = 1
            print("*" * 20)
            for row in mycursor:
                print(index, " → ", row[0])
                index += 1
            print("*" * 20)
            ch = int(input("enter choice number for category : "))
            if ch == 1:
                cat = "Mobility"
```

```
            elif ch == 2:
                cat = "Hearing"
            elif ch == 3:
                cat = "Vision"
            elif ch == 4:
                cat = "Speech"

            sql = "SELECT Name, Product_id FROM product WHERE
product.Category = %s"
            val = (cat,)
            mycursor.execute(sql, val)

            print("These are the Availible products for this
category:")
            for row in mycursor:
                print(row[1], " ", row[0])

            pro_id = int(input("Enter product id of the required
product : "))
            query3 = "select  * from product where product_id= %s"
            val = (pro_id,)
            mycursor.execute(query3, val)

            for row in mycursor:
```

```
                print("Description  ",row[2])
                print("Price - ",row[5]  )
                print("Company - ",row[6]  )

            cart_ch = int(input("enter 1 to add to cart / else 2 :
"))
            print("-"*50)
            if cart_ch == 1:
                qty = int(input("enter quantity"))

                print("Your cartID is ",cart_id)
                sql_q = "insert into Cart (cart_id, productID,
quantity) values(%s,%s,%s)"
                val2 = (
                    cart_id,
                    pro_id,
                    qty,
                )
                mycursor.execute(sql_q, val2)
                cnx.commit()

                print("Added to cart")
            else:
                print("not added. Retry")
```

```
                continue
        elif inuu == 2:
            print("Products in cart are ")
            # query2 = "Select * From logincredentials where
username='{}' ".format(
            #         username
            #     )

            sql = "SELECT Product_id,Name FROM product,cart WHERE
product.product_ID = cart.productID and cart_id=%s"
            val = (cart_id,)
            mycursor.execute(sql, val)
            result_set = mycursor.fetchall()
            for row in result_set:
                print(row)
            cnx.commit()
            sql =  "SELECT SUM(Price) FROM product INNER JOIN cart ON
product.Product_ID = cart.productID WHERE cart.cart_id = %s GROUP BY
cart.cart_id"
            val = (cart_id,)
            mycursor.execute(sql, val)
            res=mycursor.fetchone()

            if res is not None:
```

```
    totalprice = res[0]
    print("Total price is", totalprice)
else:
    print("No items in the cart")
    continue

cnx.commit()



while True:
    print("1. to place order")
    print("2. exit")
    order_ch=int(input("enter choice "))

    if order_ch==1:



        sql = "SELECT * from Users WHERE Users.UserID=%s"
        val = (userid,)
        mycursor.execute(sql, val)
        re = mycursor.fetchone()
        State=re[6]
        City=re[7]
```

```
                    Street=re[8]
                    Pincode=re[10]
                    cnx.commit()

                    # sql = "SELECT SUM(Quantity) FROM Cart WHERE
cart_id=%s"
                    # val=(cart_id,)
                    # re = mycursor.fetchall()
                    # if re:  # Check if any rows are returned
                    #     qty = re[0][0]  # Access the first element
of the first row
                    #     print("Total Quantity in Cart:", qty)
                    # else:
                    #     print("No items in the cart")




                    pc=int(input("enter your payment mode \n 1- UPI\n
2- Cash \n 3-Card\n enter: "))
                    if pc==1:
                        PaymentMode="UPI"
                    elif pc==2:
                        PaymentMode="Cash"
                    elif pc==3:
```

```
                    PaymentMode="Card"
                transactionno+=1
                transactionid=transactionno
                date_today = date.today()
                # Orderstatus="Pending"
                Deliveryworkerid=3



                sql="INSERT INTO OrderTable ( Cart_id, Discount,
State, City, PaymentMode, Street, Pincode, TransactionID,
ProductAmount, DeliveryworkerID,  UserID, OrderDate) VALUES
(%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s)"

val=(cart_id,0,State,City,PaymentMode,Street,Pincode,transactionid,to
talprice,Deliveryworkerid,userid,date_today)
                mycursor.execute(sql, val)
                cnx.commit()


                # sql="DELETE FROM Cart WHERE cart_id=%s"
                # val=(cart_id,)
                # mycursor.execute(sql, val)
                # cnx.commit()
                print("Order Placed")
```

```
                    break
                else :
                    print("Exited")
                    break


        elif inuu == 3:
            print("Logged out")
            break


# baadne
def signup():
    global usernumber
    while True:

        print("^" * 50)
        userID = usernumber + 1
        usernumber += 1
        username = str(input("enter username:"))
        password = str(input("enter password:"))
```

```
print("^" * 50)


name = input("Enter Name: ")
age = input("Enter Age: ")
phonenumber = input("Enter Phone Number: ")
emailaddress1 = input("Enter Email Address1: ")
emailaddress2 = input("Enter Email Address2 (optional): ")
if not emailaddress2:
    emailaddress2 = None
state = input("Enter State: ")
city = input("Enter City: ")
street = input("Enter Street: ")
apartmentno = input("Enter Apartment No: ")
pincode = input("Enter Pincode: ")
landmark = input("Enter Landmark: ")


# Validations
if len(phonenumber) != 10:
    print("Error: Only 10 digits allowed for Phone Number")
    continue
if not validate_email(emailaddress1):
    print("Error: Email Address1 isn't valid")
    continue
```

```
        if emailaddress2 and not validate_email(emailaddress2):
            print("Error: Email Address2 isn't valid")
            continue

        # Insert into database
        try:
            sql1 = "INSERT INTO Users (UserID, name, Age,
PhoneNumber1, EmailAddress1, EmailAddress2, State, City, Street,
ApartmentNo, Pincode, Landmark) VALUES (%s, %s, %s, %s, %s, %s, %s,
%s, %s, %s, %s, %s)"
            val1 = (
                userID,
                name,
                age,
                phonenumber,
                emailaddress1,
                emailaddress2,
                state,
                city,
                street,
                apartmentno,
                pincode,
                landmark,
            )
```

```
            sql2 = "INSERT INTO logincredentials (UserID,
username,password) VALUES (  %s,  %s, %s)"
            val2 = (
                userID,
                username,
                password,
            )
            mycursor.execute(sql1, val1)
            mycursor.execute(sql2, val2)
            cnx.commit()  # Commit the changes to the database
            print("User signed up successfully!")
            break
        except mysql.connector.Error as err:
            if err.errno == 1062:
                print("Error: Duplicate entry. UserID or Email
Address already exists.")
            elif err.errno == 1406:
                print("Error: Input value too long. Please provide
shorter values.")
            elif err.errno == 1264:
                print("Error: Age value is too big. Please provide a
smaller value.")
            elif err.errno == 3819:
                print(
```

```
                    "Error: Invalid input format for Email Address/
Age incorrect/phone number/ pincode. Please provide a valid email
address."
                )
            else:
                print(
                    "Error, try with valid entries. try entering all
essential details:"
                )
        finally:
            mycursor.close()  # Close the cursor
            cnx.close()  # Close the connection



# def signup():
#     while True:
#         userID = input("Enter UserID: ")
#         name = input("Enter Name: ")
#         age = input("Enter Age: ")
#         phonenumber = input("Enter Phone Number: ")
#         emailaddress1 = input("Enter Email Address1: ")
#         emailaddress2 = input("Enter Email Address2 (optional): ")
#         state = input("Enter State: ")
#         city = input("Enter City: ")
```

```
#         street = input("Enter Street: ")
#         apartmentno = input("Enter Apartment No: ")
#         pincode = input("Enter Pincode: ")
#         landmark = input("Enter Landmark: ")
#         cnx.commit

#         # Validations
#         if len(phonenumber) != 10:
#             print("Error: Only 10 digits allowed for Phone Number")
#             continue
#         if not validate_email(emailaddress1):
#             print("Error: Email Address1 isn't valid")
#             continue
#         if emailaddress2 and not validate_email(emailaddress2):
#             print("Error: Email Address2 isn't valid")
#             continue

#         # Insert into database
#         try:

#             sql = "INSERT INTO Users (UserID, name, Age,
PhoneNumber1, EmailAddress1, EmailAddress2, State, City, Street,
ApartmentNo, Pincode, Landmark) VALUES (%s, %s, %s, %s, %s, %s, %s,
%s, %s, %s, %s, %s)"
```

```
#             val = (userID, name, age, phonenumber, emailaddress1,
emailaddress2, state, city, street, apartmentno, pincode, landmark)
#             mycursor.execute(sql, val)


#             print("User signed up successfully!")
#             break
#         except mysql.connector.Error as err:
#             if err.errno == 1062:
#                 print("Error: Duplicate entry. UserID or Email
Address already exists.")
#             elif err.errno == 1406:
#                 print("Error: Input value too long. Please provide
shorter values.")
#             elif err.errno == 1264:
#                 print("Error: Age value is too big. Please provide
a smaller value.")
#             elif err.errno == 3819:
#                 print("Error: Invalid input format for Email
Address. Please provide a valid email address.")
#             else:
#                 print("Error, try with valid entries:", err.msg)



def add_product():
    print("Add a Product")
```

```
    name = input("Enter Product Name: ")
    description = input("Enter Product Description: ")
    company_name = input("Enter Company Name: ")
    category = input("Enter Product Category: ")
    status = input("Enter Product Status (True/False): ")
    supplier_id = input("Enter Supplier ID: ")
    quantity = input("Enter Quantity: ")

    # Insert into database
    try:
        sql = "INSERT INTO Product (Name, Description, Company_name,
Category, Status, SupplierID, Quantity) VALUES (%s, %s, %s, %s, %s,
%s, %s)"
        val = (name, description, company_name, category, status,
supplier_id, quantity)
        mycursor.execute(sql, val)
        cnx.commit()
        print("Product added successfully!")
    except mysql.connector.Error as err:
        print("Error:", err)

def display_delivery_workers():
    try:
        # Establish connection to the database
```

```
        # cursor = cnx.cursor()


        # Execute SQL query to retrieve delivery workers data
        query = ("SELECT ID, workerName, Age, City FROM
DeliveryWorker")
        mycursor.execute(query)


        # Print column headers
        print("{:<5} {:<15} {:<5} {:<15} ".format(
            "ID", "Name", "Age", "City"))
        print("="*50)


        # Iterate over the rows and print each delivery worker's
information
        for (ID, workerName, Age, City) in mycursor:
            print("{:<5} {:<15} {:<5} {:<15}".format(
                ID, workerName, Age, City))


        # Close cursor and connection


    except mysql.connector.Error as err:
        print("Error:", err)


def add_ngo():
```

```
    print("Add an NGO")
    name = input("Enter NGO Name: ")
    # company_id = input("Enter Company ID: ")
    description = input("Enter NGO Description: ")
    email = input("Enter Email Address: ")
    address = input("Enter Address: ")
    contact_number = input("Enter Contact Number: ")

    try:
        sql = "INSERT INTO NGOs (Name, Description, Email, Address,
ContactNumber) VALUES (%s, %s, %s, %s, %s)"
        val = (name, description, email, address, contact_number)
        mycursor.execute(sql, val)
        cnx.commit()
        print("NGO added successfully!")
    except mysql.connector.Error as err:
        print("Error:", err)

def get_active_users_count():
    try:
        # Execute SQL query to count the number of active users
        query = "SELECT COUNT(*) AS ActiveUsersCount FROM Users WHERE
account_status = 'Active';"
        mycursor.execute(query)
```

```
        result = mycursor.fetchone()
        active_users_count = result[0]
        print(f"The number of active users are :
{active_users_count}")
    except mysql.connector.Error as err:
        print(f"An error occurred: {err}")



def display_user():
    try:
        # Establish connection to the database

        # Execute SQL query to fetch user information
        query = ("""
            SELECT u.UserID, u.name, COUNT(o.OrderID) AS
TotalPurchases
            FROM Users u
            LEFT JOIN OrderTable o ON u.UserID = o.UserID
            GROUP BY u.UserID, u.name
            HAVING COUNT(o.OrderID) >= 1
        """)
        mycursor.execute(query)

        # Print column headers
```

```
        print("{:<10} {:<30} {:<15}".format(
            "User ID", "User Name",  "Total Purchases"))
        print("="*60)

        # Iterate over the rows and print user information
        for (UserID, name, TotalPurchases) in mycursor:
            print("{:<10} {:<30} {:<15}".format(
                UserID, name, TotalPurchases))

        # Close cursor and connection


    except mysql.connector.Error as err:
        print("Error:", err)

def inventory_analysis():
    try:
        # Establish connection to the database

        # Execute SQL query for inventory analysis
        query = ("""
            SELECT p.Product_ID, p.Name, p.Quantity, s.Name AS
Supplier, s.City AS SupplierCity, s.State AS SupplierState
```

```
            FROM Product p
            JOIN Supplier s ON p.SupplierID = s.SupplierID;

        """)
        mycursor.execute(query)

        # Print column headers
        print("{:<10} {:<30} {:<10} {:<20} {:<15} {:<15}".format(
            "Product ID", "Product Name", "Quantity", "Supplier",
"Supplier City", "Supplier State"))
        print("="*100)

        # Iterate over the rows and print product information
        for (Product_ID, Name, Quantity, Supplier, SupplierCity,
SupplierState) in mycursor:
            print("{:<10} {:<30} {:<10} {:<20} {:<15} {:<15}".format(
                Product_ID, Name, Quantity, Supplier, SupplierCity,
SupplierState))

        # Close cursor and connection

    except mysql.connector.Error as err:
```

```
        print("Error:", err)


def view_product_sales():


    try:
        # Establish connection to the database




        # Execute SQL query to fetch product sale statistics from the
view
        query = ("SELECT Product_Name, Supplier_Name,
Total_Units_Sold, Total_Sales_Amount FROM ProductSalesSummary")
        mycursor.execute(query)


        # Print column headers
        print("="*110)
        print("{:<30} {:<30} {:<20} {:<20}".format(
             "Product Name", "Supplier Name", "Total Units Sold",
"Total Sales Amount"))
        print("="*110)


        # Iterate over the rows and print product sale statistics
        for (Product_Name, Supplier_Name, Total_Units_Sold,
Total_Sales_Amount) in mycursor:
```

```
            print("{:<30} {:<30} {:<30} {:<20} ".format(
                Product_Name, Supplier_Name, Total_Units_Sold,
Total_Sales_Amount))



    except mysql.connector.Error as err:
        print("Error:", err)



while True:

    # print("3.See all product categories")
    # print("4.See all products under a productcategory")
    # print("5.See products in cart")
    # print("6.Order products from your cart")
    # print("7.Exit")
    print("1. Admin Login")
    print("2. User Portal")
    login_input = int(input("Enter 1 for Admin Login / Enter 2 for
User Activities \n "))

    if login_input == 1:

        log_attempt = login()
```

```
        while True:
            print(" Admin Menu : Enter digit accordingly")
            if log_attempt:
                print("*" * 50)
                print("1. Add Product")
                print("2. Add NGO")
                print("3. Show Delivery workers general information
")
                print("4. See the number of Active users ")
                print("5. Show Analysis of the current inventory")
                print("6. Display Users with alteast one purchase")
                print("7. View Product Sale Stats")
                print("8. Logout")
                print("*" * 50)
                chk = int(input())
                if chk == 1:
                    add_product()
                if chk == 2:
                    add_ngo()
                if chk==3:
                    display_delivery_workers()
                if chk == 4:
```

```
                get_active_users_count()
            if chk==5:
                inventory_analysis()
            if chk==6:
                display_user()
            if chk==7:
                view_product_sales()
            if chk == 8:
                print("Exiting")
                break
    continue


# admin_login_count = 0
# if login_input == 1:
#     while(True):
#         log_attempt = login()
#         print(" Admin Menu : Enter digit accordingly")
#         if log_attempt:
#             print("*"*50)
#             print("1. Add Product")
#             print("2. Add NGO")
#             print("3. Show Delhivery workers")
#             print("4. See the number of Active users ")
```

```
#             print("5 Exit")
#             print("*"*50)
#             chk = int(input())
#             if chk == 1:
#                 add_product()
#             if chk == 2:
#                 add_ngo()
#             if chk == 4:
#                 get_active_users_count()
#             if chk==5:
#                 break



    while login() != True:
        admin_login_count += 1
        if admin_login_count == 3:
            break
        login()

elif login_input == 2:
    print("1.Sign up")
    print("2.Log In")
```

```
        print("3.Exit")
        inuu = int(input("enter your choice"))
        if inuu == 1:
            signup()
        elif inuu == 2:


            print("^" * 50)
            print("Login as USER")
            print("^" * 50)
            logincount = 0
            while True:


                if logincount == 4:
                    break
                print("^" * 50)
                print("Please Enter your username")
                username = str(input("enter username:"))
                query2 = "Select * From logincredentials where
username='{}' ".format(
                    username
                )
                mycursor.execute(query2)
```

```
                rec_recheck = mycursor.fetchone()


                if rec_recheck:
                    userid = rec_recheck[0]
                    print("your user id-",userid)
                    status_query = (
                        "Select account_status From users where
userid='{}' ".format(
                            userid
                        )
                    )
                    mycursor.execute(status_query)


                    bl = mycursor.fetchone()
                    b = bl[0]
                    if b == "Blocked":
                        print("^" * 50)
                        print("Blocked userid")
                        print(" Can't Enter ")
                        break
                    password = str(input("enter password:"))
                    pass_query = (
                        "Select * From logincredentials where
Password='{}' ".format(
```

```
                    password
                )
            )
            mycursor.execute(pass_query)
            pass_recheck = mycursor.fetchall()

            if pass_recheck:
                print("^" * 50)
                print("ACCESS GRANTED")

                after_login(userid)
                #! to include rest of the code
                break
            else:
                logincount += 1
                print("Password Incorrect")
                print(" Restarting ")
                insert_attempt = (
                    "INSERT INTO login_attempts (UserID)
VALUES (  %s)"
                )
                val2 = (userid,)
                mycursor.execute(insert_attempt, val2)
```

```
                        cnx.commit()
                else:
                    print("wrong username")
        elif inuu == 3:
            print("You have logged out as user")
            break
```

3. Sql Database

```
Drop database accessisphere;
CREATE DATABASE if not exists ACCESSISPHERE;
USE ACCESSISPHERE;



CREATE TABLE IF NOT EXISTS Users (
    UserID int unique PRIMARY KEY NOT NULL,
    name varchar(255) NOT NULL,
    Age int CHECK (Age > 0), -- can be null because  the site doesnt use user age anywhere
    PhoneNumber1 bigint unique NOT NULL CHECK (PhoneNumber1 >= 1000000000 AND PhoneNumber1 <= 9999999999),
    EmailAddress1 varchar(255) unique NOT NULL CHECK (EmailAddress1 LIKE '%@%.%'),
```

```
    EmailAddress2 varchar(255) unique CHECK (EmailAddress2 LIKE
'%@%.%'),
    State varchar(255) NOT NULL,
    City varchar(255) NOT NULL,
    Street varchar(255) NOT NULL,
    ApartmentNo varchar(255) NOT NULL,
    Pincode varchar(255) NOT NULL check (length(pincode)=6),
    Landmark varchar(255)
);
ALTER TABLE users
ADD account_status VARCHAR(20) DEFAULT 'Active';



-- DROP TABLE IF EXISTS premiummember;
CREATE TABLE if not exists premiummember (
    -- for one member only one premium memebrship exists
    MembershipID INT PRIMARY KEY not null,
    Duration INT not null, -- taken to be in months
    TransactionID INT not null unique,
    UserID int not null unique  ,
    foreign key (UserID) references Users(UserID)
    -- a permium member must be a user too
);
```

```
-- DROP TABLE IF EXISTS admin;

CREATE TABLE IF NOT EXISTS Admin (

    FirstName VARCHAR(255) NOT NULL,

    LastName VARCHAR(255),

    AdminID INT PRIMARY KEY,

    Password VARCHAR(255) NOT NULL CHECK (

        LENGTH(Password) >= 8 AND           -- Password should be at
least 8 characters long

        REGEXP_LIKE(Password, '[A-Z]') AND  -- Password should
contain at least one uppercase letter

        REGEXP_LIKE(Password, '[0-9]')     -- Password should contain
at least one digit

    )

);
```

```
-- DROP TABLE IF EXISTS ngos;--

CREATE TABLE if not exists NGOs (

    Name VARCHAR(255) not null,
```

```
    CompanyID INT auto_increment PRIMARY KEY,

    Description TEXT not null,

    Email VARCHAR(255) not null unique check (Email LIKE '%@%._%'),

    Address TEXT not null,

    ContactNumber VARCHAR(20) not null unique CHECK (ContactNumber >=
1000000000 AND ContactNumber <= 9999999999)

);


 CREATE Table if not exists Supplier(

 Name Text not null,

 SupplierID int Primary Key Not Null,

 State varchar(255) Not Null ,

 City varchar(255) Not Null,

 Street varchar(255) Not Null,

 Appartmentnumber Int not null ,

Phone_number bigint not null CHECK (Phone_number >= 1000000000 AND
Phone_number <= 9999999999),

AccountNumber bigint  unique not null ,

 -- one supplier can have only one registered account for business,
one phone number associated for e-commerce

IFSCcode int not null ,

UPI_id varchar(100) not null


 );
```

```
 ALTER TABLE Supplier DROP PRIMARY KEY;


ALTER TABLE Supplier
ADD PRIMARY KEY (SupplierID, Street);


  CREATE TABLE if not exists DeliveryWorker(
 ID int primary key not null ,
 workerName varchar (50) not null ,
 Age int not null ,
 City varchar (50) not null ,
 State varchar (50) not null ,
 Street varchar(255) not null ,
 Pincode int check (length(pincode)=6),
 Product varchar(255) not null ,
 Email varchar (100) check (Email LIKE '%@%.%') not null unique,
 Phone_number varchar (15) not null unique ,
 License_code text not null ,
 UPI_ID varchar(100) unique not null ,
 -- one delivery worker must have one working UPI_ID registered for
this e-commerce business
 Salary float not null
 );
 ALTER TABLE DeliveryWorker DROP PRIMARY KEY;
```

```
 ALTER TABLE DeliveryWorker
ADD PRIMARY KEY (ID, product);
-- one delivery worker can deliver multiple products


--  SET GLOBAL sql_mode = '';


 -- DROP TABLE IF EXISTS product;
CREATE TABLE if not exists Product(
   Product_ID int auto_increment ,
   Name text not null  ,
   Description text not null ,
   SupplierID int not null,
   Quantity int,
   Price FLOAT not null,
   primary key (Product_ID),
   foreign key (SupplierID) references Supplier(SupplierID),
   Company_name text not null ,
   Category text not null ,
   Status boolean not null
   -- here product_ID is sufficient as primary key as it doesn't refer
to product name or model it refers to each unique product
 );
```

```
CREATE TABLE if not exists UserProductSearch (
    UserID int not null,
    ProductID int not null,
    SearchDateTime DATETIME,
    FOREIGN KEY (UserID) REFERENCES Users(UserID),
    FOREIGN KEY (ProductID) REFERENCES Product(Product_ID),
    PRIMARY KEY (UserID, ProductID, SearchDateTime)
);


-- DROP TABLE IF EXISTS cart;
CREATE TABLE if not exists Cart(
 cart_id int not null   ,
 productID int not null,
 Quantity int not null,
 primary key(cart_id,productID)
 -- engine = InnoDB  Add this line--
 );


CREATE TABLE if not exists OrderTable (
```

```
    OrderID INT Not null unique auto_increment,
    Cart_id int not null unique,
    foreign key (Cart_id) references Cart(cart_ID),
    Discount FLOAT,
    -- only one discount type applicable on one order
    -- Price FLOAT not null,
    State VARCHAR(50) not null,
    City VARCHAR(50) not null,
    PaymentMode VARCHAR(50) not null,
    -- paymentMode represnts not the avaible modes but the
paymentmode via which payment was done
    Street TEXT not null,
    Pincode INT not null,
    TransactionID INT unique not null,
    ProductAmount FLOAT not null,
    DeliveryworkerID int not null ,
    -- one order can have deliveryworker
    UserID int not null unique,
    -- one
    foreign key (DeliveryworkerID) references DeliveryWorker(ID),
    foreign key (UserID) references Users(UserID) ,
    OrderDate date NOT NULL,
    OrderStatus varchar(255),
```

```
    primary key(UserID,OrderID)
     -- Add this line
 );
```

```
-- DROP TABLE IF EXISTS logincredentials;
 CREATE TABLE if not exists LoginCredentials(
 UserID int not null unique,
 Username varchar (50) primary key not null , -- username means login
credentials ka user name
 foreign key (UserID) references Users(UserID) ,
    Password VARCHAR(255) NOT NULL CHECK (
        LENGTH(Password) >= 8 AND           -- Password should be at
least 8 characters long
        REGEXP_LIKE(Password, '[A-Z]') AND  -- Password should
contain at least one uppercase letter
        REGEXP_LIKE(Password, '[0-9]')     -- Password should contain
at least one digit
    )
 );
```

```
-- DROP TABLE IF EXISTS productreview;
 CREATE TABLE if not exists ProductReview(
   UserID int not null,
   foreign key (UserID) references Users(UserID) ,
   Rating float not null ,
   Review_ID int primary key
 );

-- CREATE TABLE if not exists Cart(
-- cart_id int primary key  ,
--  foreign key (productID) references product(Product_ID) ,
--  Quantity int
--  );

CREATE TABLE IF NOT EXISTS ReviewReply (
    ReviewReplyID INT not null,
    Review_ID INT not null,
    UserID INT not null,
    PRIMARY KEY (ReviewReplyID, Review_ID),  -- Combined primary key
with ReviewReplyID and Review_ID
    FOREIGN KEY (Review_ID) REFERENCES ProductReview(Review_ID),
```

```
    FOREIGN KEY (UserID) REFERENCES Users(UserID),
    ReplyText VARCHAR(255) NOT NULL,
    ReplyDate VARCHAR(50)
);

CREATE TABLE IF NOT EXISTS checksavailability (
    ProductID INT not null,
    SupplierID INT not null,
    StockQuantity INT not null,
    PRIMARY KEY (ProductID,SupplierID),
    -- here product-id is unique for a product, hence for one
productID there can be only one ProuductID
    FOREIGN KEY (ProductID) REFERENCES Product(Product_ID),
    FOREIGN KEY (SupplierID) REFERENCES Supplier(SupplierID)

);

CREATE TABLE login_attempts (
    attempt_id INT AUTO_INCREMENT PRIMARY KEY,
    UserID INT,
    attempt_timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (UserID) REFERENCES users(UserID)
);
```

```
-- DELIMITER //

-- CREATE TRIGGER update_stock_and_check_availability AFTER INSERT ON
OrderTable FOR EACH ROW
-- BEGIN
--     DECLARE done INT DEFAULT FALSE;
--     DECLARE product_id INT;
--     DECLARE ordered_quantity INT;
--     DECLARE available_stock INT;

--     DECLARE cur CURSOR FOR
--         SELECT c.ProductID, c.Quantity, ca.StockQuantity
--         FROM Cart c
--         JOIN checksavailability ca ON c.ProductID = ca.ProductID
--         WHERE c.cart_id = NEW.Cart_id;

--     DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

--     OPEN cur;
```

```
--     product_loop: LOOP
--         FETCH cur INTO product_id, ordered_quantity,
available_stock;

--         IF done THEN
--             LEAVE product_loop;
--         END IF;

--         SET available_stock = available_stock - ordered_quantity;

--         IF available_stock < 0 THEN
--             SET available_stock = 0;
--         END IF;

--         UPDATE checksavailability
--         SET StockQuantity = available_stock
--         WHERE ProductID = product_id;

--         IF available_stock < 0 THEN
--             SET @error_message = CONCAT('Product "', product_id,
'" is out of stock. You can only order up to ', available_stock, '
units.');
--             SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT =
@error_message;
```

```
--             LEAVE product_loop;
--         END IF;
--     END LOOP;


--     CLOSE cur;
-- END;
-- //


-- DELIMITER ;



-- DELETE FROM Users WHERE UserID BETWEEN 1 AND 20;


INSERT INTO Users (UserID, name, Age, PhoneNumber1, EmailAddress1,
EmailAddress2, State, City, Street, ApartmentNo, Pincode, Landmark)
VALUES
(1, 'Amit Patel', 30, 9876543210, 'amit@gmail.com',
'amit2@yahoo.com', 'Gujarat', 'Ahmedabad', '123 ABC Road', '101',
'380081', 'Near Park'),
(2, 'Priya Singh', 25, 8765432109, 'priya@gmail.com',
'priya3@hotmail.com', 'Uttar Pradesh', 'Lucknow', '456 XYZ Street',
'102', '226001', 'Opposite Mall'),
(3, 'Rahul Sharma', 28, 7654321098, 'rahul@gmail.com',
'rahul2@outlook.com', 'Maharashtra', 'Mumbai', '789 MNO Avenue',
'103', '400001', 'Next to Cinema'),
```

```
(4, 'Sneha Gupta', 23, 6543210987, 'sneha@gmail.com', null, 'Delhi',
'New Delhi', '101 DEF Lane', '104', '110001', 'Near School'),

(5, 'Ravi Verma', 27, 5432109996, 'ravi@gmail.com', null, 'Tamil
Nadu', 'Chennai', '102 GHI Road', '105', '600001', 'Beside Temple'),

(6, 'Neha Shah', 22, 4321098765, 'neha@gmail.com', 'neha2@gmail.com',
'Karnataka', 'Bangalore', '103 JKL Colony', '106', '560001', 'Behind
Market'),

(7, 'Aarav Reddy', 26, 3210987654, 'aarav@gmail.com', null,
'Telangana', 'Hyderabad', '104 MNO Street', '107', '500001', 'Near
Lake'),

(8, 'Ishaan Kapoor', 29, 2109876543, 'ishaan@gmail.com', null, 'West
Bengal', 'Kolkata', '105 PQR Avenue', '108', '700001', 'Inside
Garden'),

(9, 'Kriti Mishra', 24, 1098765432, 'kriti@gmail.com', null, 'Madhya
Pradesh', 'Bhopal', '106 STU Lane', '109', '462001', 'On Top Floor'),

(10, 'Rajat Singhania', 20, 9876543211, 'rajat@gmail.com',
'rajat2@yahoo.com', 'Rajasthan', 'Jaipur', '107 VWX Road', '110',
'302001', 'In Front of Mall'),

(11, 'Manisha Patil', 31, 8765432108, 'manisha@gmail.com',
'manisha2@hotmail.com', 'Andhra Pradesh', 'Vijayawada', '108 YZA
Street', '111', '520001', 'Near Bank'),

(12, 'Deepak Singh', 32, 7654321097, 'deepak@gmail.com',
'deepak@outlook.com', 'Punjab', 'Chandigarh', '109 BCD Colony',
'112', '160001', 'Beside Stadium'),

(13, 'Nisha Yadav', 28, 6543210986, 'nisha@gmail.com',
'nisha2@gmail.com', 'Haryana', 'Gurgaon', '110 EFG Road', '113',
'122001', 'Next to Hospital'),

(14, 'Rajesh Tiwari', 33, 5432109995, 'rajesh@gmail.com',
'rajesh2@gmail.com', 'Bihar', 'Patna', '111 HIJ Lane', '114',
'800001', 'Behind School'),
```

```
(15, 'Pooja Mehta', 22, 4321098764, 'pooja@gmail.com',
'pooja2@gmail.com', 'Uttarakhand', 'Dehradun', '112 JKL Street',
'115', '248001', 'Near Market'),

(16, 'Karan Malhotra', 27, 3210987653, 'karan@gmail.com', null,
'Jharkhand', 'Ranchi', '113 LMN Avenue', '116', '834001', 'Beside
Park'),

(17, 'Sakshi Sharma', 26, 2109876542, 'sakshi@gmail.com', null,
'Assam', 'Guwahati', '114 OPQ Road', '117', '781001', 'Next to
Airport'),

(18, 'Rahul Gupta', 25, 1098765431, 'rahul1@gmail.com', null,
'Odisha', 'Bhubaneswar', '115 RST Colony', '118', '751001', 'Inside
College'),

(19, 'Khushi Singh', 24, 9876543220, 'khushi@gmail.com', null, 'Goa',
'Panaji', '116 UVW Street', '119', '403001', 'Near Beach'),

(20, 'Vivek Kumar', 29, 8765432110, 'vivek@gmail.com', null,
'Kerala', 'Thiruvananthapuram', '117 XYZ Road', '120', '695001', 'In
Front of Temple');



-- Insert values into PremiumMember table

INSERT INTO PremiumMember (MembershipID, Duration, TransactionID,
UserID)

VALUES

(1, 12, 1001, 1),

(2, 6, 1002, 2),

(3, 3, 1003, 3),
```

```
(4, 12, 1004, 4),
(5, 6, 1005, 5),
(6, 3, 1006, 6),
(7, 12, 1007, 7),
(8, 6, 1008, 8),
(9, 3, 1009, 9),
(10, 12, 1010, 10);

INSERT INTO Admin (FirstName, LastName, AdminID, Password)
VALUES
('Swarnima', 'Prasad', 1, 'password1'),
('Palak', 'Bhardwaj', 2, 'password2');

-- Insert values into NGOs table
INSERT INTO NGOs (Name, CompanyID, Description, Email, Address,
ContactNumber)
VALUES
('Access for All', 1, 'A non-profit organization that promotes
accessibility and inclusion for people with disabilities',
'accessforall@gmail.com', '123 Main Street, New Delhi, 110001',
'1123456789'),
('Enable India', 2, 'A social enterprise that provides employment
opportunities and training for people with disabilities',
'enableindia@gmail.com', '456 Park Avenue, New Delhi, 110002',
'1134567890'),
```

```
('Samarthya', 3, 'A charity that provides education and
rehabilitation for children with disabilities',
'samarthya@gmail.com', '789 Market Road, New Delhi, 110003',
'1145678901'),

('Nayi Disha', 4, 'A platform that connects families of people with
intellectual and developmental disabilities with resources and
support', 'nayidisha@gmail.com', '101 Hill Street, New Delhi,
110004', '1156789012'),

('Saksham', 5, 'An organization that empowers people with visual
impairment through technology and advocacy', 'saksham@gmail.com',
'102 Lake View, New Delhi, 110005', '1167890123'),

('Muskaan', 6, 'A foundation that works for the welfare and
integration of people with Down syndrome', 'muskaan@gmail.com', '103
Garden Lane, New Delhi, 110006', '1178901234'),

('AADI', 7, 'An institute that provides comprehensive services for
people with physical and multiple disabilities', 'aadi@gmail.com',
'104 River Side, New Delhi, 110007', '1189012345'),

('Shishu Sarothi', 8, 'A centre that offers education, therapy, and
legal aid for children with disabilities', 'shishusarothi@gmail.com',
'105 Forest Road, New Delhi, 110008', '1190123456'),

('Vidya Sagar', 9, 'A network that supports the rights and dignity of
people with neurological and developmental disabilities',
'vidyasagar@gmail.com', '106 Sky Tower, New Delhi, 110009',
'1101234567'),

('Ummeed', 10, 'A team that provides early intervention and therapy
for children with developmental disabilities', 'ummeed@gmail.com',
'107 Sunshine Plaza, New Delhi, 110010', '1112345678');


-- DELETE FROM Cart WHERE cart_id BETWEEN 1 AND 10;

INSERT INTO Cart (cart_id, productID, quantity)
```

```
VALUES
(1, 1, 1),
(2, 2, 2),
(3, 3, 3),
(4, 4, 4),
(5, 5, 5),
(6, 6, 6),
(7, 7, 7),
(8, 8, 8),
(9, 9, 9),
(10, 10, 10);


INSERT INTO Supplier (Name, SupplierID, State, City, Street,
Appartmentnumber, Phone_number, AccountNumber, IFSCcode, UPI_id)
VALUES
('WheelCo', 1, 'Delhi', 'New Delhi', '123 Main Street', 101,
9876543210, 111, 111111, 'wheelco@upi'),
('CrutchCo', 2, 'Delhi', 'New Delhi', '456 Park Avenue', 102,
8765432109, 223, 222222, 'crutchco@upi'),
('HearCo', 3, 'Delhi', 'New Delhi', '789 Market Road', 103,
7654321098, 344, 333333, 'hearco@upi'),
('BrailleCo', 4, 'Delhi', 'New Delhi', '101 Hill Street', 104,
6543210987, 433, 444444, 'braillecо@upi'),
('StickCo', 5, 'Delhi', 'New Delhi', '102 Lake View', 105,
5432109876, 598, 555555, 'stickco@upi'),
```

```
('MagnifyCo', 6, 'Delhi', 'New Delhi', '103 Garden Lane', 106,
4321098765, 698, 666666, 'magnifyco@upi'),

('SpeakCo', 7, 'Delhi', 'New Delhi', '104 River Side', 107,
3210987654, 789, 777777, 'speakco@upi'),

('CaneCo', 8, 'Delhi', 'New Delhi', '105 Forest Road', 108,
2109876543, 889, 888888, 'caneco@upi'),

('ProstheticCo', 9, 'Delhi', 'New Delhi', '106 Sky Tower', 109,
1098765432, 919, 999999, 'prostheticco@upi'),

('GlassCo', 10, 'Delhi', 'New Delhi', '107 Sunshine Plaza', 110,
9987654321, 122, 101010, 'glassco@upi');


INSERT INTO DeliveryWorker (ID, workerName, Age, City, State, Street,
Pincode, Product, Email, Phone_number, License_code, UPI_ID, Salary)

VALUES

(1, 'Rajesh Kumar', 25, 'New Delhi', 'Delhi', '123 Main Street',
110001, 'Wheelchair', 'rajesh@gmail.com', '9876543210', 'DL-1234',
'rajesh@upi', 20000),

(2, 'Priya Sharma', 24, 'New Delhi', 'Delhi', '456 Park Avenue',
110002, 'Crutches', 'priya@gmail.com', '8765432109', 'DL-2345',
'priya@upi', 18000),

(3, 'Vikram Singh', 26, 'New Delhi', 'Delhi', '789 Market Road',
110003, 'Hearing Aid', 'vikram@gmail.com', '7654321098', 'DL-3456',
'vikram@upi', 21000),

(4, 'Anjali Gupta', 23, 'New Delhi', 'Delhi', '101 Hill Street',
110004, 'Braille Keyboard', 'anjali@gmail.com', '6543210987',
'DL-4567', 'anjali@upi', 17000),

(5, 'Ravi Verma', 27, 'New Delhi', 'Delhi', '102 Lake View', 110005,
'Walking Stick', 'ravi@gmail.com', '5432109876', 'DL-5678',
'ravi@upi', 19000),
```

```
(6, 'Pooja Patel', 22, 'New Delhi', 'Delhi', '103 Garden Lane',
110006, 'Magnifier', 'pooja@gmail.com', '4321098765', 'DL-6789',
'pooja@upi', 16000),

(7, 'Amit Shah', 28, 'New Delhi', 'Delhi', '104 River Side', 110007,
'Speech Synthesizer', 'amit@gmail.com', '3210987654', 'DL-7890',
'amit@upi', 22000),

(8, 'Neha Jain', 21, 'New Delhi', 'Delhi', '105 Forest Road', 110008,
'Cane', 'neha@gmail.com', '2109876543', 'DL-8901', 'neha@upi',
15000),

(9, 'Suresh Mehta', 29, 'New Delhi', 'Delhi', '106 Sky Tower',
110009, 'Prosthetic Limb', 'suresh@gmail.com', '1098765432',
'DL-9012', 'suresh@upi', 23000),

(10, 'Reena Roy', 20, 'New Delhi', 'Delhi', '107 Sunshine Plaza',
110010, 'Glasses', 'reena@gmail.com', '0987654321', 'DL-0123',
'reena@upi', 14000);
```

```
INSERT INTO Product (Product_ID, Name, Description, Company_name,
Category, Status,SupplierID,Quantity,Price)

VALUES

(1, 'Wheelchair', 'A chair with wheels that can be propelled by the
user or an attendant', 'WheelCo', 'Mobility', true,1,1,1000),

(2, 'Crutches', 'A pair of supports that help a person to walk with
an injured leg or foot', 'CrutchCo', 'Mobility', true,1,1,2000),

(3, 'Hearing Aid', 'A device that amplifies sound for a person with
hearing loss', 'HearCo', 'Hearing', true,3,400,5000),
```

```
(4, 'Braille Keyboard', 'A keyboard that allows a person with visual
impairment to type in braille', 'BrailleCo', 'Vision',
true,1,50,2000),

(5, 'Walking Stick', 'A stick that helps a person to balance or
walk', 'StickCo', 'Mobility', false,2,40,5000),

(6, 'Magnifier', 'A device that enlarges the appearance of an object
for a person with low vision', 'MagnifyCo', 'Vision',
true,3,70,2000),

(7, 'Speech Synthesizer', 'A device that converts text to speech for
a person with speech impairment', 'SpeakCo', 'Speech',
true,3,90,2000),

(8, 'Cane', 'A long stick with a curved handle that helps a person to
walk', 'CaneCo', 'Mobility', false,3,5,5000),

(9, 'Prosthetic Limb', 'An artificial limb that replaces a missing
one', 'ProstheticCo', 'Mobility', true,3,0,5000),

(10, 'Glasses', 'A pair of lenses that correct a person vision',
'GlassCo', 'Vision', true,3,8,5000);
```

```
INSERT INTO OrderTable (OrderID, Cart_id, Discount,  State, City,
PaymentMode, Street, Pincode, TransactionID, ProductAmount,
DeliveryworkerID,  UserID, OrderDate, OrderStatus)

VALUES

(1, 1, 0.1, 'Delhi', 'New Delhi', 'Cash', '123 Main Street', 110001,
1001, 9000, 1,  1, '2024-02-11', 'Delivered'),

(2, 2, 0.2,  'Delhi', 'New Delhi', 'Card', '456 Park Avenue', 110002,
1002, 8000, 2,  2, '2024-02-11', 'Delivered'),
```

```
(3, 3, 0.15,  'Delhi', 'New Delhi', 'UPI', '789 Market Road', 110003,
1003, 7650, 3,  3, '2024-02-11', 'Delivered'),

(4, 4, 0.05,  'Delhi', 'New Delhi', 'Cash', '101 Hill Street',
110004, 1004, 7600, 4,  4, '2024-02-11', 'Delivered'),

(5, 5, 0.1,  'Delhi', 'New Delhi', 'Card', '102 Lake View', 110005,
1005, 4500, 5,  5, '2023-02-11', 'Delivered'),

(6, 6, 0.2,  'Delhi', 'New Delhi', 'UPI', '103 Garden Lane', 110006,
1006, 2400, 6,  6, '2024-05-11', 'Delivered'),

(7, 7, 0.15,  'Delhi', 'New Delhi', 'Cash', '104 River Side', 110007,
1007, 595, 7,  7, '2024-02-11', 'Delivered'),

(8, 8, 0.05,  'Delhi', 'New Delhi', 'Card', '105 Forest Road',
110008, 1008, 380, 8,  8, '2024-02-11', 'Delivered');
```

```
INSERT INTO LoginCredentials (Username, UserID, Password)

VALUES

('amit', 1, 'Amit4045'),

('priya', 2, 'Priya456'),

('rahul', 3, 'Raaaa789'),

('ravi', 5, 'Ravi2024'),

('neha', 6, 'Neha3033'),

('ishan',8,'Ishan202'),

('Kriti',9,'Kriti202'),

('Sneha', 4, 'SNeha5054'),
```

```
('Rajat', 10, 'Rajat606'),
('Manisha', 11, 'Manisha707'),
('Deepak', 12, 'Deepak707'),
('Nisha', 13, 'Nisha707'),
('Rajesh',14,'Rajesh38938'),
('Pooja', 15, 'Pooja707'),
('Karan', 16, 'Karan707'),
('Sakshi', 17, 'Sakshi707'),
('Rahu', 18, 'Rahul707'),
('Aarav',7,'Aarav9090'),
('Khushi', 19, 'Khushi707'),
('Vivek', 20, 'Vivek77707');

INSERT INTO ProductReview (UserID, Rating, Review_ID)
VALUES
(1, 4.5, 1),
(2, 3.5, 2),
(3, 4.0, 3),
(4, 2.5, 4),
(5, 3.0, 5),
(6, 4.5, 6),
(7, 3.5, 7),
(8, 4.0, 8),
```

```
(9, 2.5, 9),

(10, 3.0, 10);


INSERT INTO ReviewReply (ReviewReplyID, Review_ID, UserID, ReplyText,
ReplyDate)

VALUES

(1, 1, 1, 'Great product!', '2024-02-01'),

(2, 2, 2, 'Good but could be better', '2024-02-02'),

(3, 3, 3, 'Satisfactory', '2024-02-03'),

(4, 4, 4, 'Not as expected', '2024-02-04'),

(5, 5, 5, 'Disappointing', '2024-02-05'),

(6, 6, 6, 'Excellent service!', '2024-02-06'),

(7, 7, 7, 'Could improve', '2024-02-07'),

(8, 8, 8, 'Average', '2024-02-08'),

(9, 9, 9, 'Poor quality', '2024-02-09'),

(10, 10, 10, 'Exceptional!', '2024-02-10');



-- Example values for UserProductSearch table

INSERT INTO UserProductSearch (UserID, ProductID, SearchDateTime)
VALUES

(1, 1, '2024-02-12 08:30:00'),

(1, 2, '2024-02-12 08:35:00'),

(2, 3, '2024-02-12 09:00:00'),
```

```
(3, 1, '2024-02-12 10:15:00'),
(3, 4, '2024-02-12 10:20:00'),
(4, 2, '2024-02-12 11:45:00'),
(4, 3, '2024-02-12 11:50:00');

INSERT INTO checksavailability (ProductID, SupplierID, StockQuantity)
VALUES
(1, 1, 1),  -- ProductID 1 from SupplierID 1 has 100 items in stock
(2, 1, 1),   -- ProductID 2 from SupplierID 1 has 50 items in stock
(3, 1, 150),  -- ProductID 3 from SupplierID 1 has 150 items in stock
(1, 2, 200),  -- ProductID 1 from SupplierID 2 has 200 items in stock
(2, 2, 75),   -- ProductID 2 from SupplierID 2 has 75 items in stock
(3, 2, 100);  -- ProductID 3 from SupplierID 2 has 100 items in stock

-- UPDATE Product
-- SET Status = 'Out of Stock'
-- WHERE Product_ID = 123 AND Quantity = 0;

-- SET SQL_SAFE_UPDATES = 0;

-- Query1
```

```
-- stakeholders such as admins can easily create a summary of product
sale


CREATE VIEW ProductSalesSummary AS

SELECT

    p.Product_ID,

    p.Name AS Product_Name,

    p.Description AS Product_Description,

    s.Name AS Supplier_Name,

    SUM(c.Quantity) AS Total_Units_Sold,

    SUM(ot.ProductAmount) AS Total_Sales_Amount

FROM

    Product p

JOIN

    Cart c ON p.Product_ID = c.ProductID

JOIN

    Supplier s ON p.SupplierID = s.SupplierID

JOIN

    ordertable ot ON ot.Cart_id = c.cart_id

GROUP BY

    p.Product_ID, p.Name, p.Description, s.Name;
```

```
-- stakeholders such as admins can easily view a summary of product
sales they created


-- Illustration of the query

-- SELECT * FROM ProductSalesSummary;


-- Query2


-- There is a large volume of data in Product Table,which contains
various accessibility products.

-- Users can search for products based on their names.

-- The search queries can take large amount of time to execute due to
the lack of indexing.


-- reference statement


CREATE INDEX idx_product_name ON Product (Name(255));


-- Illustration of the Query


-- SELECT *

-- FROM Product

-- WHERE Name LIKE 'Wheel%';
```

```
-- Query3


-- the admin may want to maintain up-to-date

-- information about product availability for customers and internal
operations.

UPDATE Product

SET Status = CASE

                WHEN Quantity > 0 THEN true

                ELSE false

            END;


-- select * from product;


-- Query4


SELECT DISTINCT dw.ID, dw.workerName

FROM deliveryworker dw

JOIN OrderTable o ON dw.ID = o.DeliveryworkerID

WHERE o.OrderDate BETWEEN '2024-01-01' AND '2024-03-01';


-- Query5


-- The transaction involves deducting the ordered quantity from the
product's inventory and
```

```
-- recording the order details in the OrderTable.

-- This scenario showcases the use of a transaction to ensure data
integrity and consistency in the database



UPDATE checksavailability

SET StockQuantity = StockQuantity - (

    SELECT Quantity

    FROM Cart

    WHERE Cart.cart_id = 9 AND checksavailability.ProductID =
Cart.productID

)

WHERE ProductID IN (

    SELECT productID

    FROM Cart

    WHERE Cart.cart_id = 9

);


-- Select * from supplier;


-- Query 6


-- Scenario : The admin might want to view all suppliers with a
certain account number for tracking
```

```
SELECT Product_ID,Name, SupplierID

FROM Product

WHERE SupplierID IN (SELECT SupplierID FROM Supplier WHERE
AccountNumber = 111);


-- Query 7


-- The admin may want to see all the products even those that are
discountinued,

-- or have no suppliers at that moment


SELECT p.Product_ID, p.Name, p.Company_name, s.Name

FROM Product p

LEFT OUTER JOIN Supplier s ON p.SupplierID = s.SupplierID;


-- Query 8


-- This query helps in identifing suppliers who do not supply a
specific category of

-- products, which could be useful for identifying potential new
suppliers for certain product categories.


SELECT s.SupplierID, s.Name
```

```
FROM Supplier s
WHERE NOT EXISTS (
    SELECT 1
    FROM Product p
    WHERE p.SupplierID = s.SupplierID
    AND p.Category = 'Mobility'
);

-- Query 9

SELECT DISTINCT s.*
FROM Supplier s
INNER JOIN Product p ON s.SupplierID = p.SupplierID
INNER JOIN ProductReview pr ON p.Product_ID = pr.UserID
WHERE pr.Rating >= 4;

-- Query 10

-- seeing users with atleast 1 purchase

SELECT u.UserID, u.name, COUNT(o.OrderID) AS TotalPurchases
FROM Users u
JOIN OrderTable o ON u.UserID = o.UserID
```

```
GROUP BY u.UserID, u.name

HAVING COUNT(o.OrderID) >= 1;


-- INSERT INTO Users (UserID, name, Age, PhoneNumber1, EmailAddress1,
EmailAddress2, State, City, Street, ApartmentNo, Pincode, Landmark)

-- VALUES (21, 'John Doe', 30, 1234567890, 'invalid_email',
'john@yahoo.com', 'Delhi', 'New Delhi', '123 Elm Street', '201',
'110011', 'Near School');




-- -- Updating User's Phone Number:

-- UPDATE Users

-- SET PhoneNumber1 = 9999

-- WHERE UserID = 1;




-- -- This will fail because PhoneNumber1 and EmailAddress1 are
required fields.

-- INSERT INTO Users (UserID, name, Age, State, City, Street,
Pincode)

-- VALUES (1001, 'John Doe', 30, 'New York', 'NYC', '123 Main St',
10001);


-- -- This will fail due to the primary key constraint violation on
UserID.
```

```
-- INSERT INTO Users (UserID, name, Age, PhoneNumber1, EmailAddress1,
State, City, Street, Pincode)

-- VALUES (1, 'Alice', 25, '9998887777', 'alice@example.com',
'Delhi', 'Delhi', '123 Main St', 110001);


-- -- This will fail due to the check constraint on Age, ensuring
it's a positive value.

-- UPDATE Users

-- SET Age = -5

-- WHERE UserID = 1;


-- TRIGGER1
```

```
DROP TRIGGER IF EXISTS block_user;

DELIMITER //

CREATE TRIGGER block_user

AFTER INSERT ON login_attempts

FOR EACH ROW

BEGIN

    DECLARE attempts INT;
```

```
    SELECT COUNT(*) INTO attempts FROM login_attempts WHERE UserID =
NEW.UserID;

    IF attempts >= 3 THEN

        UPDATE users SET account_status = 'Blocked' WHERE UserID =
NEW.UserID;

    END IF;

END;

//

DELIMITER ;


-- DELIMITER //

-- -- Create trigger

-- CREATE TRIGGER update_product_quantity

-- AFTER INSERT ON OrderTable

-- FOR EACH ROW

-- BEGIN

--     -- Update product quantity in checksavailability table

--     UPDATE checksavailability

--     SET StockQuantity = StockQuantity - (

--         SELECT Quantity

--         FROM Cart

--         WHERE cart_id = NEW.Cart_id

--     )

--     WHERE ProductID = (
```

```
--         SELECT productID
--         FROM Cart
--         WHERE cart_id = NEW.Cart_id
--     );
-- END;


-- //
-- DELIMITER ;



-- INSERT INTO login_attempts (UserID) VALUES (1);
-- INSERT INTO login_attempts (UserID) VALUES (2);
-- INSERT INTO login_attempts (UserID) VALUES (3);
-- INSERT INTO login_attempts (UserID) VALUES (1);
-- INSERT INTO login_attempts (UserID) VALUES (1);
-- INSERT INTO login_attempts (UserID) VALUES (1);




-- TRIGGER 2
-- ALTER TABLE deliveryworker
-- ADD availability BOOLEAN DEFAULT TRUE;
```

```
-- DROP TRIGGER IF EXISTS da_unavailable;

-- DELIMITER $$

-- CREATE TRIGGER da_unavailable AFTER INSERT ON ordertable

-- FOR EACH ROW

-- BEGIN

--     UPDATE deliveryworker SET availability = FALSE WHERE ID=
NEW.DeliveryworkerID;

-- END;

-- $$

-- DELIMITER ;


-- -- Check the availability of a specific delivery agent

-- SELECT ID, availability FROM deliveryworker WHERE ID = 20;



-- INSERT INTO DeliveryWorker (ID, workerName, Age, City, State,
Street, Pincode, Product, Email, Phone_number, License_code, UPI_ID,
Salary,availability)

-- VALUES

-- (20, 'Raghu Kumar', 25, 'New Delhi', 'Delhi', '123 Main Street',
110001, 'Wheelchair', 'raghu@gmail.com', '9876643210', 'DL-1234',
'raghu@upi', 20000,1);

-- INSERT INTO Users (UserID, name, Age, PhoneNumber1, EmailAddress1,
EmailAddress2, State, City, Street, ApartmentNo, Pincode, Landmark)

-- VALUES
```

```
-- (21, 'Anju Patel', 30, 9879543210, 'anju@gmail.com',
'anju2@yahoo.com', 'Gujarat', 'Ahmedabad', '123 ABC Road', '101',
'380081', 'Near Park');


-- delete from DeliveryWorker where ID=20;

-- delete from Users where UserID=21;

-- delete from OrderTable where OrderID=20;



DELIMITER //

-- Create the trigger

CREATE TRIGGER UpdateOrderStatus

BEFORE INSERT ON OrderTable

FOR EACH ROW

BEGIN

    -- Set the OrderStatus to 'Pending' for the newly inserted order

    SET NEW.OrderStatus = 'Pending';

    -- UPDATE OrderTable

--     SET OrderStatus = 'Pending'

--     WHERE OrderID = NEW.OrderID;

END;

//

DELIMITER ;
```

```
INSERT INTO OrderTable (OrderID, Cart_id, Discount,  State, City,
PaymentMode, Street, Pincode, TransactionID, ProductAmount,
DeliveryworkerID,  UserID, OrderDate)

VALUES

(20, 10, 0.1, 'Delhi', 'New Delhi', 'Cash', '123 Main Street',
110001, 2001, 9000, 7,  20, '2024-02-11');
```

```
SELECT SUM(Price) FROM product INNER JOIN cart ON product.Product_ID
= cart.productID WHERE cart.cart_id =1 GROUP BY cart.cart_id;
```