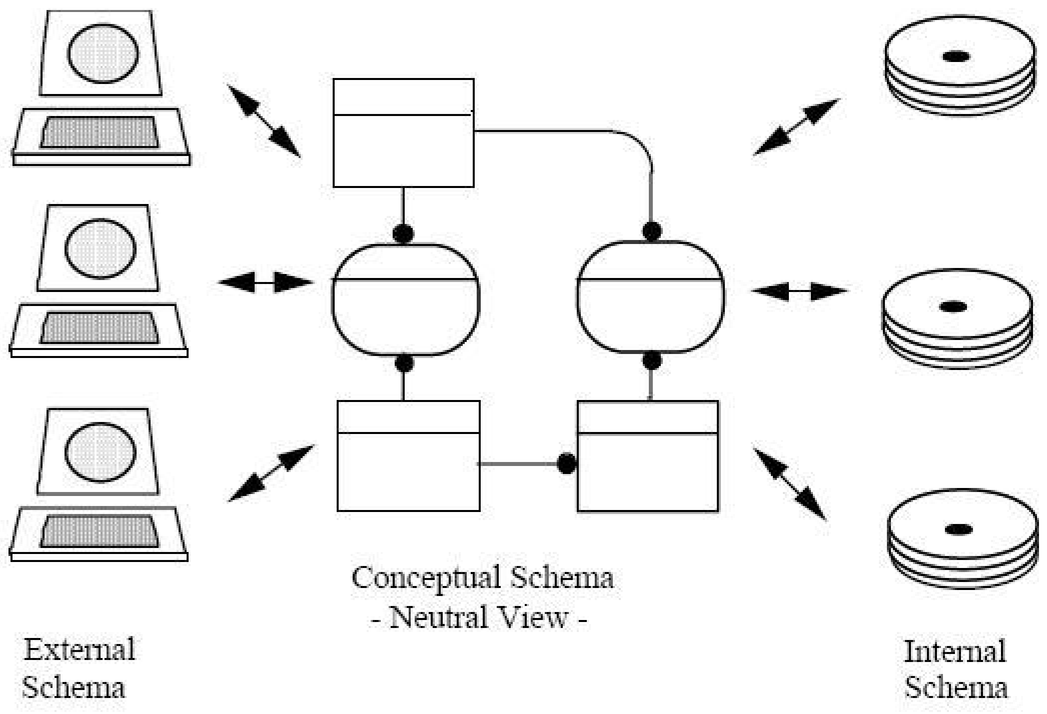

Documentation

Date: 29/03/2023

Embedded SQL Queries

- Sign up for User :

```
def user_Signup(name, gender, age, address, phone_number, password):
    cursor = connection.cursor()
    query = "INSERT INTO customer (Name, Gender, Age, Address,
PhoneNumber, Password) VALUES ('{}', '{}', '{}', '{}', '{}',
'{}')".format(name, gender, age, address, phone_number, password)
    cursor.execute(query)
    connection.commit()
    cursor.close()
```

- Login for User :

```
def user_Login(name, password):

   global current_user

   try:

       cursor = connection.cursor()

       cursor.execute("select user_id, name from customer where name = '{}' and
password = '{}'".format(name, password))

       data = cursor.fetchall()

       [id,  name] = data[0]

       current_user = id

       print("Successfully Logged In as {} (USER ID: '{}')".format(name,id))

       cursor.close()

       return True

   except:

       print("Incorrect username or password. Please try again.")

       return False
```

- Sign up for Admin :

```
def admin_Signup(name, gender, age, username, password):

       cursor = connection.cursor()

       cursor.execute("INSERT INTO Admins (Name, Gender, Age, Username, Password)
VALUES ('{}', '{}', '{}', '{}', '{}')".format(name, gender, age, username, password))

       connection.commit()

       cursor.close()
```

- Login for Admin :

```
def admin_Login(username, password):


   global current_admin

   try:

       cursor = connection.cursor()

       cursor.execute("select admin_id, name from Admins where username = '{}' and
password = '{}'".format(username, password))

       data = cursor.fetchall()

       [id,  name] = data[0]

       current_admin = id

       print("Succesfully Loged In as {} (Admin ID: '{}')".format(name,id))

       cursor.close()

       return True


   except:

       print("Incorrect username or password. Please try again.")

       return False
```

- Sign up for Delivery Agent :

```
def delivery_Agent_Signup(name, gender, age, phone_number, password):

   cursor = connection.cursor()

   cursor.execute("INSERT INTO DeliveryAgent (Name, Gender, Age, PhoneNumber, Rating,
Password) VALUES ('{}', '{}', '{}', '{}', {}, '{}')".format(name, gender, age,
phone_number, 0, password))

   connection.commit()
   cursor.close()
```

- Login for Delivery Agent :

```
def delivery_Agent_Login(name, password):

   global current_delivery_agent

   try:

       cursor = connection.cursor()

       cursor.execute("SELECT DeliveryAgent_ID, Name FROM DeliveryAgent WHERE 
Name = '{}' AND Password = '{}'".format(name, password))

       data = cursor.fetchall()

       [id, name] = data[0]

       current_delivery_agent = id

       print("Successfully logged in as {} (Delivery Agent ID: 
'{}')".format(name, id))

       cursor.close()

       return True


   except:

       print("Incorrect username or password. Please try again.")

       return False
```

- Add Category :

```
def add_category(name, discount, description):

   cursor = connection.cursor()

   cursor.execute("insert into category values ('{}','{}','{}','{}')".format(name, 
discount, description, current_admin))
```

```
   cursor.commit()

   cursor.close()
```

- Delete Category :

```
def delete_category(name):

   try:

       cursor = connection.cursor

       cursor.execute("DELETE FROM Category WHERE Name = '{}' ".format(name))

       cursor.commit()

       cursor.close()

       print("Succesfully deleted")

   except Exception as e:

       print("There was an error in deleting the category")

       print(e)
```

- View Cart for User :

```
def view_cart():

    global current_user

   try:

       cursor = connection.cursor()

       cursor.execute("SELECT Product.Name, Product.Price, Cart.Quantity FROM Product
INNER JOIN Cart ON Product.Product_ID = Cart.Product_ID WHERE Cart.User_ID = %s",
(current_user,))

       data = cursor.fetchall()

       print(data)
```

```
except Exception as e:

    print(e)
```

OLAP Queries

- Top 3 categories that were viewed most and added most to the cart by users :

```
SELECT c.Name AS Category_Name, SUM(c2.Quantity) AS Total_Quantity

FROM Category c

JOIN Product p ON p.Category_ID = c.Category_ID

JOIN Cart c2 ON c2.Product_ID = p.Product_ID

GROUP BY c.Category_ID ORDER BY Total_Quantity

DESC LIMIT 3;
```

  - This is an OLAP query because it involves aggregating and summarizing data from multiple tables (Category, Product, and Cart) to provide insights into the total quantity of products sold by category.
  - It is doing the following things :
    - Joining tables: The query joins three tables, Category, Product, and Cart, to extract data from each of them.
    - Filtering: The query filters the data to select only the records where the Product_ID in the Cart table matches the Product_ID in the Product table.
    - Grouping: The query groups the data by Category_ID to obtain the total quantity of products sold for each category.
    - Aggregation: The query sums up the quantity of products sold for each category.

        - Sorting: The query sorts the results by Total_Quantity in descending order to identify the top three categories by the quantity of products sold.

- Total Sales made by each retailer in the current month:

```
SELECT R.Name as RetailerName, SUM(O.Order_Amount) as TotalSales

FROM Orders O JOIN Cart C ON O.Cart_ID = C.Cart_ID

JOIN Product P ON C.Product_ID = P.Product_ID

JOIN Retailer R ON P.Retailer_ID = R.Retailer_ID

WHERE MONTH(O.Order_Date) = MONTH(CURRENT_DATE())

AND YEAR(O.Order_Date) = YEAR(CURRENT_DATE()) GROUP BY R.Retailer_ID;
```

    - This is an OLAP query because it is analyzing sales data for a specific month and year. The query is joining multiple tables - Orders, Cart, Product, and Retailer - to retrieve information about the retailer's name and the total sales amount for each retailer. The query is using the GROUP BY clause to group the results by the retailer's ID.
    - It is doing the following things :
        - Joining tables: The query joins four tables, Orders, Cart, Product, and Retailer, using the JOIN keyword to extract data from each of them.
        - Filtering: The query filters the data to select only the records where the Order_Date is in the current month and year, using the MONTH() and YEAR() functions. The query also filters the data to select only the records where the Product_ID in the Cart table matches the Product_ID in the Product table.
        - Grouping: The query groups the data by the Retailer_ID to obtain the total sales amount for each retailer.

        - Aggregation: The query uses the SUM() function to aggregate the Order_Amount column and obtain the total sales amount for each retailer.
        - Sorting: The query sorts the results by the TotalSales column in descending order to identify the top retailers by the total sales amount.

- Total Sales made by each category by each retailer.

```
SELECT Category.Name AS Category, Retailer.Name As Retailer,

SUM(Orders.Order_Amount) AS TotalSales

FROM Orders INNER JOIN Cart ON Orders.Cart_ID = Cart.Cart_ID

INNER JOIN Product ON Cart.Product_ID = Product.Product_ID

INNER JOIN Category ON Product.Category_ID = Category.Category_ID

INNER JOIN Retailer ON Product.Retailer_ID = Retailer.Retailer_ID

GROUP BY Category.Category_ID, Retailer.Retailer_ID
```

    - This is an OLAP query because it is analyzing sales data for different categories of products and different retailers. The query is joining multiple tables - Orders, Cart, Product, Category, and Retailer - to retrieve information about the category name, retailer name, and the total sales amount for each combination of category and retailer. The query is using the GROUP BY clause to group the results by both the Category_ID and Retailer_ID.
    - It is doing the following things :
        - Joining tables: The query joins five tables, Orders, Cart, Product, Category, and Retailer, using the INNER JOIN keyword to extract data from each of them.
        - Filtering: The query does not have any filters.

        - Grouping: The query groups the data by both Category_ID and Retailer_ID to obtain the total sales amount for each combination of category and retailer.
        - Aggregation: The query uses the SUM() function to aggregate the Order_Amount column and obtain the total sales amount for each combination of category and retailer.
        - Sorting: The query does not have any sorting.

- Top 5 products sold :

```
SELECT Product.Name, SUM(Orders.Order_Amount) as Total_Sales

FROM Orders

INNER JOIN Cart ON Orders.Cart_ID = Cart.Cart_ID

INNER JOIN Product ON Cart.Product_ID = Product.Product_ID

GROUP BY Product.Name

WITH ROLLUP

HAVING Product.Name IS NOT NULL

ORDER BY Total_Sales DESC

LIMIT 5;
```

    - This is an OLAP query because it is analyzing sales data for different products. The query is joining multiple tables - Orders, Cart, and Product - to retrieve information about the product name and the total sales amount for each product. The query is using the GROUP BY clause to group the results by the Product_ID column and the SUM() function to aggregate the Order_Amount column and obtain the total sales amount for each product. The query is also using the ORDER BY clause to sort the results by the TotalSales column in descending order and the LIMIT clause to limit the results to the top 5 products by sales amount.
    - It is doing the following things :

- Joining tables: The query joins two tables, Orders and Cart, using the INNER JOIN keyword to extract data from each of them. The query also joins the Product table using the INNER JOIN keyword to retrieve information about the products sold.
- Filtering: The query filters the data to select only the records where the Product_ID in the Cart table matches the Product_ID in the Product table. This ensures that we are only looking at sales data for products that have been sold.
- Grouping: The query groups the data by the Product_ID column to obtain the total sales amount for each product.
- Aggregation: The query uses the SUM() function to aggregate the Order_Amount column and obtain the total sales amount for each product.
- Sorting: The query uses the ORDER BY clause to sort the results by the TotalSales column in descending order to identify the top 5 products by sales amount.

Triggers

- Checking the quantity in inventory before adding to the cart :

```
delimiter //

CREATE TRIGGER check_cart_quantity

BEFORE INSERT ON Cart

FOR EACH ROW

BEGIN
```

```
   DECLARE product_quantity INT;

   SELECT Quantity INTO product_quantity FROM Product WHERE Product_ID =
NEW.Product_ID;

   IF NEW.Quantity > product_quantity THEN

       SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'ERROR: The requested
quantity exceeds the available quantity for this product.';

   END IF;

END;//
```

- The trigger checks whether the requested quantity of a product in the cart exceeds the available quantity of that product in the "Product" table. It does so by retrieving the current quantity of the product from the "Product" table based on the "Product_ID" of the new row to be inserted into the "Cart" table.If the requested quantity is greater than the available quantity, the trigger generates an error by using the SIGNAL statement with SQLSTATE '45000' and sets the error message to "ERROR: The requested quantity exceeds the available quantity for this product."In summary, this trigger ensures that the quantity of products requested to be added to the "Cart" table does not exceed the available quantity in the "Product" table to prevent overselling and stockouts.

- Updating the quantity in the inventory whenever a cart is converted to an Order :

```
DELIMITER //

CREATE TRIGGER reduce_product_quantity AFTER INSERT ON Orders

FOR EACH ROW
```

```
BEGIN

   UPDATE Product

   JOIN Cart ON Product.Product_ID = Cart.Product_ID

   SET Product.Quantity = Product.Quantity - Cart.Quantity

   WHERE Cart.Cart_ID = NEW.Cart_ID;

END //



DELIMITER ;
```

- The trigger updates the quantity of products in the "Product" table based on the products in the corresponding order. It does so by joining the "Product" and "Cart" tables on the "Product_ID" column and checking which products were present in the order. The quantity of each product in the "Product" table is then reduced by the quantity that was present in the "Cart" table for that particular order.The WHERE clause limits the update to only the products that were present in the order that was just inserted into the "Orders" table. It does this by matching the "Cart_ID" of the order being inserted with the "Cart_ID" in the "Cart" table.In summary, this trigger reduces the quantity of products in the "Product" table by the amount that was sold in each order, ensuring that the available product quantity is updated accordingly after each sale.