Problem definition:

We have to design an online retail store system

Project scope:

For our project the stakeholders are;

1. Customer
2. Supplier
3. Customer service
4. Administrator

Relational Schema

- CUSTOMER(C_ NAME, C_PHNO, C_MAIL, C_ADDR(houseno, street/locality, city/town, state, pincode), C_PASSWORD)

- QUERIES(EMPLOYEE_ID, C_PHNO, QUERY_ID)
- CUSTOMER_SERVICE(EMPLOYEE_ID, STATUS, PHONE, EMAIL, ADDRESS(houseno, street/locality, city/town, state, pincode), DATE_OF_HIRING, FULL_NAME, PASSWORD)

- SUPPLIER(S_NAME, S_PHNO, S_MAIL, S_ADDR, S_PASSWORD)
- PRODUCT(P_NAME, P_ID, PRICE, AVAILABLE_QUANTITY)
- CATEGORY(CAT_NAME)
- CATEGORY(CAT_NAME)
- HAVE(O_ID, C_PHNO(FK))
- MAKES(CART_ID, C_PHNO(FK))
- SUPPLIES(S_PHNO, P_ID)
- CATEGORIZING(P_ID, CAT_NAME(FK))
- CONSISTS OF(P_ID, QUANTITY,cart_id,dateadded)
- ORDER(O_ID, TOTAL_COST, O_STATUS,date added,p_quantity,p_id(fk))
- PAYMENT(INVOICE_ID, CART_ID, PAYMENT_MODE)
- OWNS(C_PHNO(FK), INVOICE_ID(FK),ordercount)
- BILL(INVOICE_ID, TIMESTAMP, TOTAL_COST)

To Solve the project we have broke the project into simpler problems
First we define our layout on ER Diagram to understand the database, flow and logistics of the online retail store system .
For ER diagram:

1. identify the entity
2. Attributes of entities
3. Relationship among the entities
4. Attributes of the relationship
5. Defining the partial and total participation in the relationship.
6. Format the ER Diagram to beautify it.
7. We haven't added the admin in the Er diagram because the admin has only one job: he can run any sql query in the database.

IMPLEMENTATION IN THE SQL WORKBENCH

1. Created the table of the entities
2. Then created the table for the relationship
3. First populate the data in the entity table and then in relationship
4. Recheck the data if there is a mistake in it.
5. Defined the views and grant
6. Created 2 views first from the perspective of the customer to see the products and then from the supplier perspective to see the availability of products.
7. Grant is implemented to grant permission to the admin customer and supplier
8. Then there are sql queries and embedded sql queries.
9. Trigger are implemented to show a workflow there are 4 trigger which are working simultaneously first to check the customer count increase and decrease then dec the product quantity on any purchase then increase the product quantity by supplier
10. We use .csv file and sql queries to populate the data in tables

GUI implementation for the working application

1. We use java language to implement the GUI for the online retail store system
2. Javafx enhances the visuals of the application.

TRIGGERS

There are 4 triggers in total.

1. Ph_gap —> The trigger ph_gap executes after insertion on customer table. It increments the customer count by 1 after insertion of a new customer.
2. Ph_gap —> The trigger ph_gap1 executes after deletion on the customer table. It decrements the customer count by 1 after deletion of a customer.

The first two triggers help us in recording the count of customers i.e., the number of customers who have their account active at the present time.

3. Quantity_update —> The trigger quantity_update executes after insertion in orders table. When a customer orders something then he/she selects some quantity of a product so the quantity of that product should be decremented in the product table by the quantity that customer orders. So, this trigger updates the quantity of that product in the product table.
4. Quantity_update1 —> The trigger quantity_update1 executes after insertion in supplies table. I.e., when a supplier supplies a product then, this trigger increments the quantity of that product in the product table by the quantity that supplier supplies.

The last two triggers help us in maintaining the exact amount of available products we have in our database.

```
drop table customercount;
DROP TRIGGER ph_gap;
DROP TRIGGER ph_gap1;

create table customercount(
count int
);
insert into customercount(count)
select sum(0);

#Trigger 1
CREATE TRIGGER ph_gap
after insert on customer
for each row
update customercount
SET count=count+1;

#Trigger 2
CREATE TRIGGER ph_gap1
after delete on customer
for each row
update customercount
SET count=count-1;

INSERT INTO Customer VALUES ('shobhit',1234567890,'a@gmail.com','djhf ijhfgwe
iuegfg','fgiuh','1986-04-07');
INSERT INTO Customer VALUES ('shobhit1',1234567891,'a@gmail.com','djhf ijhf454we
iuegfg','dsjsuhf','2001-09-10');
INSERT INTO Customer VALUES ('shobhit2',1234567892,'a@gmail.com','djhf ij78454we
iuegfg','kjhgb','2002-09-10');
INSERT INTO Customer VALUES ('shobhit2',1234578892,'a@gmail.com','djhf ij78454we
iuegfg','kjhgb','2002-09-10');

select * from customercount;
```

```
DELETE FROM Customer WHERE c_phone=1234567890;
DELETE FROM Customer WHERE c_phone=1234567891;
DELETE FROM Customer WHERE c_phone=1234567892;
DELETE FROM Customer WHERE c_phone=1234578892;

select * from customercount;



#TRIGGER 3
DROP TRIGGER QUANTITYUPDATE;
CREATE TRIGGER  QuantityUpdate
AFTER INSERT ON Orders FOR EACH ROW
UPDATE product
        SET p_available_quantity = p_available_quantity - NEW.p_quantity
        WHERE  p_id=NEW.p_ID;

UPDATE  PRODUCT SET p_available_quantity = 34 where p_id=1;
UPDATE  PRODUCT SET p_available_quantity = 43 where p_id=2;
UPDATE  PRODUCT SET p_available_quantity = 40 where p_id=3;
UPDATE  PRODUCT SET p_available_quantity = 22 where p_id=4;
select * from product;
DELETE FROM ORDERS WHERE O_ID=9774;
DELETE FROM ORDERS WHERE O_ID=9754;
DELETE FROM ORDERS WHERE O_ID=1022;
DELETE FROM ORDERS WHERE O_ID=1078;

INSERT INTO `orders`VALUES (9774,2456,'Dispatched','2021-07-16',20,1);
INSERT INTO `orders`VALUES (9754,15345,'Dispatched','2021-07-16',20,2);
INSERT INTO `orders`VALUES (1022,4545,'Dispatched','2021-07-16',10,3);
INSERT INTO `orders`VALUES (1078,4542,'Dispatched','2021-07-16',11,4);

select * from product;

#TRIGGER 4
DROP TRIGGER QuantityUpdate1;
CREATE TRIGGER  QuantityUpdate1
AFTER INSERT ON supplies FOR EACH ROW UPDATE product
SET p_available_quantity = p_available_quantity + NEW.p_quantity
WHERE  p_id=NEW.p_id;
DELETE FROM supplies WHERE p_id=1;
UPDATE  PRODUCT SET p_available_quantity = 50 where p_id=5;
UPDATE  PRODUCT SET p_available_quantity = 60 where p_id=6;
UPDATE  PRODUCT SET p_available_quantity = 70 where p_id=7;
UPDATE  PRODUCT SET p_available_quantity = 80 where p_id=8;

select * from product;
DELETE FROM supplies WHERE p_id=5;
DELETE FROM supplies WHERE p_id=6;
DELETE FROM supplies WHERE p_id=7;
DELETE FROM supplies WHERE p_id=8;

INSERT INTO  `supplies` VALUES (9317519976,5,100);
INSERT INTO  `supplies` VALUES (8503052892,6,100);
INSERT INTO  `supplies` VALUES (7675513907,7,100);
INSERT INTO  `supplies` VALUES (6581683191,8,100);
select * from product;
DELETE FROM ORDERS WHERE O_ID=9775;
DELETE FROM ORDERS WHERE O_ID=9755;
DELETE FROM ORDERS WHERE O_ID=1023;
DELETE FROM ORDERS WHERE O_ID=1079;

INSERT INTO `orders`VALUES (9775,2456,'Dispatched','2021-07-16',20,5);
INSERT INTO `orders`VALUES (9755,15345,'Dispatched','2021-07-16',20,6);
INSERT INTO `orders`VALUES (1023,4545,'Dispatched','2021-07-16',10,7);
INSERT INTO `orders`VALUES (1079,4542,'Dispatched','2021-07-16',11,8);

select * from product;
```

Grant

User's accounts were created and given specific permissions. We create a user admin providing all the permissions to all tables. Then we created a customer granting them all control on their orders and display permission on the products. Suppliers have permission to update the quantity so given the update control over the product table.

```
CREATE USER 'ADMIN'@'localhost' IDENTIFIED BY 'password';
GRANT ALL ON *.* to 'admin'@'localhost' WITH GRANT OPTION;
CREATE USER 'customer'@'localhost' IDENTIFIED BY 'password';
GRANT SELECT on project.product to 'customer'@'localhost';
GRANT ALL ON project.Orders to 'customer'@'localhost';
CREATE USER 'supplier'@'localhost' IDENTIFIED BY 'password';
GRANT UPDATE on project.product to 'customer'@'localhost';
```

Views

Then two views, the customer_view and the supplier_view were created. The customer's view was made with attributes P_name,p_price and p_id. The quantity remaining in the stock was not included. The supplier_view was created with attributes P_name, quantity and p_id. The p_price was not included as the supplier need not know about it.

```
#view 1
CREATE OR REPLACE VIEW customer_view AS
SELECT P_name,p_price,p_id
FROM Product;
SELECT * FROM CUSTOMER_VIEW;
```

```
#view view2
CREATE OR REPLACE VIEW supplier_view AS
SELECT P_name,p_available_quantity,p_id
FROM Product;
SELECT * FROM SUPPLIER_VIEW;
```

ATTRIBUTES FOR INDEXING

1. Category_name → Index all the products by their category name so it's become easy to find a product. For example, if we are searching for a mobile phone then we see the electronics category from the index and search in that category only. We do not need to search in all the categories to find a product.

Create index cat_i on categorizing(cat_name);

We can find the id of the product by their category using the above index on table categorizing and once we have the product id then we can find all the details of that product from the product table.

2. Customer_phno → First sort all the customers in order of their phone numbers then Index all the customers by their phone numbers using primary indexing as the key i.e., phone number is the primary key(unique) and sorted. so that it's easy to find a customer.
create index ci on customer(c_phone);

3. Employee_id → Index all the employees by their hiring date using primary indexing after sorting or by secondary indexing without sorting. So that we can find the most experienced employees and use them to train the new employees.
Create index ei on employee(hiring_date);

4. Supplier_id → Index all the suppliers by their phone numbers.
Create index si on supplier(s_phno);

5. order _id → sort the customers on the basis of their order count so that we can find the active customers, the customers who ordered repeatedly.
Create index oi on owns(o_count);
We can find the customer's phone numbers which is the primary key of the customer according to their order count using the above index and after finding phone numbers, we can see all the data of the customer from the customer table.

TO VIEW THE INDEX TABLES USE THE BELOW QUERIES:

1. **show index from categorising from project;**
2. **show index from customer from project;**
3. **show index from employee from project;**
4. **show index from supplier from project;**
5. **show index from owns from project;**

SQL QUERIES

```
#Display the details of all the products of category electronics whose price<1000.
Select * from product where product.p_price<10000 and product.p_id in
                    (select p_id from categorising where cat_name= 'electronics');


#Display the details of customer who have some items in their cart. To notify and remind them
that their cart have something
Select * from customer where c_phone in
          (select c_phone from makes where cart_id in
 (select cart_id from consists_of where quantity>0)) ;


#Display the order_id and customer-phone of the order which are delivered successfully.
(select c_phone, o_id from have where o_id in
                    (Select o_id from orders where o_status= 'Delivered'));


#Display the details of the customers who have ordered more than 10 times.
Select * from customer where c_phone in
          (select c_phone from owns where o_count>10);

#customers who never queried ever
SELECT C.c_Name, C.c_email
 from Customer C
Where c_phone NOT IN(
SELECT C.c_phone
from Customer C, queries Q
WHERE C.c_Phone=Q.C_Phone
);


#supply produxts with price>30
Select s_phone
From Supplies
Where s_phone in(
Select S.s_phone
 From supplies S, Product P
Where P.p_price>30 and S.P_id =P.P_id
);



#Fetch all products who are having the same  price as product with id=10.
```

```
SELECT P2.p_id, P2.p_price FROM
PRODUCT P1, PRODUCT P2
WHERE P1.p_id= 10 ;



#suppliers of products ie out of stock
SELECT S_phone FROM supplies
WHERE p_id in(
SELECT P_id FROM product
WHERE p_available_quantity <50
);

#PRODUCT WHICH HAVE PRICE MORE THAN 1000 AND AVAILABLE QUANTITY >10
SELECT * FROM project.Product WHERE p_price > 1000 AND p_available_quantity  > 10;



#SELECT THE CUSTOMERS WHO HAVE PAID ONLINE
select * from customer where c_phone in
(select c_phone from have where o_id in
(select o_id from payment where payment_mode='online'));


#select the specific cateogory then show the products in it
SELECT * FROM Product WHERE p_id IN(
        SELECT P_id from categorising where cat_name = 'toys'
  );
```

<u>EMBEDDED SQL QUERIES</u>

Embedded sql queries are implemented in the java and the whole code is self explanatory.

```
package application;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.SQLException;
import java.sql.Statement;




public class querries
{

        static int cartids = 130;
        static Connection con;
        public static void connect(String args[])throws SQLException
        {
                String url = "jdbc:mysql://localhost:3306/project";
                String username = "root";
                String password = "x1x2x3x4";
                //String querry = "SELECT * FROM project.Product WHERE p_id<10;";
                try
                {
                        Class.forName("com.mysql.cj.jdbc.Driver");
                }catch(ClassNotFoundException e )
                {
                        e.printStackTrace();    }
                try
                {       con = DriverManager.getConnection(url, username, password);
                        System.out.println("connected successfully");
                }catch(SQLException e)
                {       e.printStackTrace();    }

        }
        public static void displayresult(ResultSet result) throws SQLException
        {
                ResultSetMetaData rsmd = result.getMetaData();
                while(result.next())
                {
                        String u = "";
                        for(int i=1;i<=rsmd.getColumnCount();i++)
                        {
```

```
                u+=result.getString(i)+" :";
            }
            System.out.println(u);
        }
    }
    public static ResultSet executequerry(String querry)
    {
        ResultSet result = null;
        try
        {
            Statement statement = con.createStatement();
            result = statement.executeQuery(querry);

        }catch(SQLException e)
        {
            e.printStackTrace();

        }
        return result;
    }
    public static void executequerry2(String querry)
    {
        try
        {
            Statement statement = con.createStatement();
            statement.executeUpdate(querry);
            System.out.println("Executed successfully");

        }catch(SQLException e)
        {
            e.printStackTrace();

        }
    }

    public static ResultSet search(String pname)
    {
        String querry = "SELECT * FROM Product WHERE p_name = '"+pname+"'";
        return executequerry(querry);
    }
    public static ResultSet searchcat(String pname)
    {
        System.out.print("procedding "+pname);
        String querry = "SELECT * FROM Product WHERE p_id IN( SELECT P_id from
categorising where cat_name = " + ("'"+pname+"'")+ " );";
```

```
        return executequerry(querry);
}
public static ResultSet getproductdetails(String name) throws SQLException
{
        String querry = "SELECT * FROM Product WHERE p_name = "+"'"+name+"'";
        Statement statement = con.createStatement();
        ResultSet result = statement.executeQuery(querry);
        //displayresult(result);
        return result;
}
public static void displayallproduct() throws SQLException
{
        String querry = "SELECT * FROM Product ";
        Statement statement = con.createStatement();
        ResultSet result = statement.executeQuery(querry);
        displayresult(result);
}
public static ResultSet displayallcategories() throws SQLException {
        String querry = "SELECT * FROM Category";
        Statement statement = con.createStatement();
        ResultSet result = statement.executeQuery(querry);
        return result;
}

public static int getcartid(long cid)
{
        try {
        String querry = "SELECT * FROM project.makes WHERE c_phone = "+cid+";";
        ResultSet result = executequerry(querry);
        if(result.next())
        {
                int cart;
                        cart = Integer.parseInt(result.getString(1));
                        return cart;
        }
                } catch (NumberFormatException e) {
                        e.printStackTrace();
                } catch (SQLException e) {
                        e.printStackTrace();
                }

        return 0;
}
public static ResultSet getcart(long cart) throws SQLException
{
			String querry2 = "SELECT * FROM project.Cart WHERE c_id = "+cart+";";
			ResultSet result2 = executequerry(querry2);
			return result2;
		}
		public static ResultSet getprofiledata(long cid)
		{
			System.out.println("cid-> "+cid);
			String querry = "SELECT * From project.Customer WHERE c_phone = "+cid;
			ResultSet result = executequerry(querry);
			return result;
		}
		public static boolean changeaddres(long cid ,String add)
		{
			try {
				String querry = "UPDATE Customer SET c_address = '"+add+"'  WHERE
	c_phone = "+cid+" ;";
				System.out.println(querry);
				Statement statement = con.createStatement();
				boolean b;
				b = statement.execute(querry);
				System.out.println(b);
				return true;
			} catch (SQLException e) {e.printStackTrace(); return false;}
		}
		public static boolean changemail(long cid ,String mail)
		{
			try {
				String querry = "UPDATE Customer SET c_email = '"+mail+"'  WHERE
	c_phone = "+cid+" ;";
				System.out.println(querry);
				Statement statement = con.createStatement();
				boolean b;
				b = statement.execute(querry);
				System.out.println(b);
				return true;
			} catch (SQLException e) {e.printStackTrace(); return false;}
		}
		public static boolean changename(long cid ,String name)
		{
			try {
				String querry = "UPDATE Customer SET c_name = '"+name+"'  WHERE
	c_phone = "+cid+" ;";
				System.out.println(querry);
				Statement statement = con.createStatement();
				boolean b;
				b = statement.execute(querry);
				System.out.println(b);
				return true;
			} catch (SQLException e) {e.printStackTrace(); return false;}
		}
		public static boolean changedob(long cid ,String dob)
		{
			try {
				String querry = "UPDATE Customer SET c_DOB = '"+dob+"'  WHERE
c_phone = "+cid+" ;";
				System.out.println(querry);
				Statement statement = con.createStatement();
				boolean b;
				b = statement.execute(querry);
				System.out.println(b);
				return true;
			} catch (SQLException e) {e.printStackTrace(); return false;}
		}
		public static int newcart(long uid)
		{
			cartids+=1;
			String querry  = "INSERT INTO  `Cart` VALUES ( '"+(cartids)+"', 0 );";

			String querry2  = "INSERT INTO `makes` VALUES ( "+cartids+", "+uid+" );";

			System.out.println(querry);
			System.out.println(querry2);
			executequerry2(querry);
			executequerry2(querry2);
			return (cartids);
		}
		public static boolean addtocart(int cid ,int pid,int q)
		{
			String querry  = "INSERT INTO `consists_of` VALUES ( "+pid+", "+q+",
\"2022/04/28\","+cid+");";
			System.out.println(querry);
			executequerry2(querry);
			return false;
		}
		public static boolean checkcart(int pid ,int cid)
		{
			String querry = "SELECT * FROM `consists_of` WHERE cart_ID = "+cid +"
AND p_id = "+pid+";" ;
			ResultSet result = executequerry(querry);
			try {
```

```
                        if(result.next())
                        {
                                return true;
                        }
                        else
                        {
                                return false;
                        }
                } catch (SQLException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
                return false;

        }
        public static boolean updatequantity(int cid,int pid,int q)
        {

                String querry  = "";
                System.out.println(querry);
                executequerry2(querry);
                return false;
        }

        public static String getname(long uid)
        {
                try {
                String querry = "SELECt * FROM `Customer` WHERE c_phone = "+uid+" ;";
                ResultSet result = executequerry(querry);
                        if(result.next())
                        {
                                return result.getString(1);
                        }
                } catch (SQLException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
                return null;
        }

}
```

CONTRIBUTION

Yash vardhan singh	❖ Create tables using ddl in mysql. ❖ find all the attributes and composite attributes for each entity. ❖ Fill all the data in the tables. ❖ Add attributes in E-R diagram ❖ Find the relationships between the entities. ❖ Find the primary keys for each entity. ❖ Find the types of all relationships(i.e., one-one, one-many, many-one, many-many). ❖ Created all the embedded sql queries ❖ Write java code to connect to the mysql server ❖ Create low level UI for few queries using c as host language ❖ Create low level UI for few queries using java as host language ❖ Drop the c code. ❖ Wrote sql queries. ❖ Check for the query optimization on sarthak queries ❖ Check for queries optimization on shobhit's queries by RA

Shobhit verma	❖ identifying all the entities-: customer, customer service, category, supplier, product, order, cart, and bill. ❖ find all the attributes and composite attributes for each entity. ❖ Conversion of er diagram to relational schema ❖ Find the relationships between the entities. ❖ Find the primary keys for each entity. ❖ Find the types of all relationships(i.e., one-one, one-many, many-one, many-many). ❖ Find the primary and foreign keys for the relationships, weak entities, total and partial participation ❖ Write 5 sql queries. ❖ Identify the attribute(s) to create Index tables required for the queries. ❖ Implementing appropriate triggers that

	support the data management in the application. 1st trigger is ph_gap which keeps the data of newly added customers i.e., if a new customer is added then the count of number of customers is incremented to 1. Similarly the 2nd trigger is ph_gap1 which decrements the count of the number of customers to 1. Third trigger is product_refill which works on updation of product. When a customer buys products then the quantity of that product will decrease. Hence, when the quantity of a product becomes less than or equal to 5 then, the above trigger will add the id of that product in the table less products so that we can know which product is less in quantity and we can order it from the suppliers of that respective product.

Sarthak dixit	❖ Write sql queries.(thinking of the various application of the project) ❖ find all the attributes and composite attributes for each entity.(after referencing to various sources) ❖ Find the primary keys for each entity**.** ❖ Find the types of all relationships(i.e., one-one, one-many, many-one, many-many). ❖ Handled the views and grants ❖ Find the relationships between the entities. ❖ Grants: user account created and grants given ❖ Then two views, the customer_view, supplier_view were created with everything that they need to access.

Aaryan s verma	❖ identifying all the entities-: customer, customer service, category, supplier, product, order, cart, and bill. By reiterating over the project and ❖ Collect and create data to populate and implement it in tables**.** ❖ find all the attributes and composite attributes for each entity.

	❖ Find the relationships between the entities. ❖ Find the types of all relationships(i.e., one-one, one-many, many-one, many-many). ❖ Worked on the feedback and updated the sql database, added some new attributes in the relationships, revised the previous entities and updated it. ❖ Data population in the updated sql database i have made the csv file the import in the sql in the respective table.