Online Retail Store System

Ayush Kumar Singh
Jatin Kumar Sharma
Prachi
Sourav Goyal

Problem Statement:

In this question, we are addressing the issue of creating an effective database management system example for an online retail store. The store should sell a variety of products, and should be well managed in terms of storage and delivery. The staff should follow a certain hierarchy, along with responsibilities given to special employee positions, such as managers, category heads, etc. There needs to be a system that manages inflow and outflow of products, the former coming in to the storage spaces of the store using various vendors, and the latter being means of users to digitally store their desired products until checkout followed by actually delivering those products physically to their homes. Multiple forms of payment needs to be addressed, like in real life. Finally, there needs to be an adequate amount of data to populate all the tables so queries can be tested and a foolproof system can be achieved.

Scope:

The scope considers payments, delivery, and management. Our project aims at having multiple inventories across multiple cities, for effective storage. There is also a proper hierarchy of people working for categories: from employees being supervised by supervisors, products being overseen by category heads, and managers looking over inventories. Payments of different types are also being considered, with both cash on delivery and credit card payment being accepted. Products have adequate basic descriptions, along with scope for additional description and specifications, should they be necessary.

Assumptions:

One assumption we have kept is in cart. We have reserved the first 1000 places in our cart for users with items they haven't yet purchased but saved in cart. Our database only has 500 customers but we kept 1000 spaces open for new customers who may join.
Another assumption is for login. Category head and branch managers can add new employees to work under them, but we believe that physical verification is necessary, along with various deliverables. So we don't activate the employee login until a later time.

Stakeholders:

Manager
Category Heads
Workers
Customers

Delivery persons
Bank
Vendor

Relational Schema:

- **Vendor:**

Vendor(**venid# PK**, fname,lname, email,phone, gender, hno, street, district, city, state, pincode)

```
create table Vendor(
venid int primary key AUTO_INCREMENT,
fname varchar(20) not null,
lname varchar(20) default "-",
email varchar(50) not null, check(email like '%_@___%.___%'),
phone char(12) not null,
gender varchar(9) not null, check(gender in ('Male', 'Female', 'others')),
hno int default -1, check(hno >= -1),
street varchar(150) not null,
district varchar(50) not null,
city varchar(20) not null,
state varchar(30) not null,
pincode int not null, check(pincode between 000000 and 999999)
);
```

- **Invoice:**

Invoice(**invid# PK**, invenid# FK, statusof, receivedDate, fulfilledDate)

```
create table Invoice(
        invid int primary key AUTO_INCREMENT,
   invenid int,
   statusof boolean not null,
   receivedDate date not null,
   fulfilledDate date not null, check(date(fulfilledDate) >= date(receivedDate)),
   foreign key(invenid) references Inventory(invenid)
   on delete cascade
   on update cascade
);
```

- **Morders:**

Morders(venid# FK, **invid# FK**)

```
create table Morders(
        invid int primary key,
   venid int,
        foreign key(invid) references Invoice(invid),
   foreign key (venid) references Vendor(venid)
   on delete cascade
   on update cascade
);
```

- **Batch:**

Batch(**bid# PK**, pdid# FK, invid# FK, quantity)

```
create table Batch(
        bid int primary key AUTO_INCREMENT,
   pdid int,
   invid int,
   quantity int default 0, check(quantity >= 0),
   foreign key (pdid) references Products(pdid),
   foreign key(invid) references Invoice(invid)
   on delete cascade
   on update cascade
);
```

- **Manager:**

Manager(**meid# PK,** invenid# FK, fname, lname, phone, email, dob, age(), gender, hno, street, district, city, state, pincode, salary, experience, doj)

```
create table Manager(
meid int primary key AUTO_INCREMENT,
invenid int,
fname varchar(20)not null,
lname varchar(20) default "-",
phone char(12) not null,
email varchar(50) not null, check(email like '%_@__%.__%'),
dob date not null, check(year(dob) < 2022),
age int generated always as(2022 -  year(dob)),
gender varchar(9) not null, check(gender in ('Male', 'Female', 'Others')),
hno int default -1, check(hno >= 0),
street varchar(150) not null,
district varchar(50) not null,
city varchar(20) not null,
state varchar(30) not null,
pincode int not null, check(pincode between 000000 and 999999),
salary int default 10000, check(salary >= 8000),
experience int not null, check(experience >= 0),
doj date not null, check(year(doj) > year(dob)),

foreign key(invenid) references Inventory(invenid)
   on update cascade
);
```

- **Inventory:**

Inventory(**invenid# PK,** hno, street, district, city, state, pincode)

```
create table Inventory(
invenid int primary key AUTO_INCREMENT,
hno int not null, check(hno >= 0),
street varchar(255) not null,
district varchar(20) not null,
city varchar(20) not null,
state varchar(50) not null,
pincode int not null, check(pincode between 000000 and 999999)
);
```

- **InInventory:**

InInventory(**invenid# FK, pdid# FK,** quantity**)**

```
create table InInventory(
        invenid int,
        pdid int,
        quantity int default 0, check(quantity >= 0),
        primary key(invenid,pdid),
        foreign key (invenid) references Inventory(invenid),
        foreign key(pdid) references Products(pdid)
        on delete cascade
   on update cascade
);
```

- **Distributes:**

Distributes(**invenid# PK**, **venid# FK**)

```
create table Distributes(
invenid int,
venid int,
primary key(invenid,venid),
foreign key(invenid) references Inventory(invenid),
foreign key(venid) references Vendor(venid)
on delete cascade
   on update cascade
);
```

- **Employee:**

Employee(**eid# PK,** fname, lname, phone, email, dob, age(), gender, hno, street, district, city, state, pincode, salary, experience, speciality, doj)

```
create table Employee(
eid int primary key,check(eid >= 0),
fname varchar(20)not null,
lname varchar(20) default "-",
phone char(12) not null,
email varchar(50) not null check(email like '%_@__%.__%'),
dob date not null,
age int generated always as ((2022-year(dob))) virtual,
gender varchar(9) not null check(gender in ('Male', 'Female', 'Others')),
hno int default -1,check(hno >= 0),
street varchar(150) not null,
district varchar(50) not null,
city varchar(20) not null,
state varchar(30) not null,
pincode int not null, check(pincode between 000000 and 999999),
salary int default 10000,check(salary >= 8000),
experience int not null,check(experience >= 0),
speciality varchar(20) not null,
doj date not null,check(year(dob) < year(doj))
);
```

- **Supervision**:

Supervision(**meid# FK**, **eid# FK**)

```
create table Supervision(
eid int,
meid int,
primary key(eid, meid),
foreign key(meid) references Manager(meid),
foreign key(eid) references Employee(eid)
        on delete cascade
   on update cascade
);
;
```

- **Supervisor**:

Supervisor(supervisor_eid# FK, **supervisee_eid# FK**)

```
create table Supervisor(
        superviser_eid int,
   supervisee_eid int,
   primary key(supervisee_eid),
   foreign key (superviser_eid) references Employee(eid),
   foreign key (supervisee_eid) references Employee(eid)
   on delete cascade
   on update cascade
);
```

- **Category**:

Category(**catid# PK**, catname, noofworkers)

```
create table Category(
catid int primary key AUTO_INCREMENT,
catname varchar(20) not null,
noofworkers int not null, check(noofworkers >= 0)
);
```

- **Cathead**:

Cathead(**cheid# FK**, catid FK)

```
create table CatHead(
cheid int primary key,
catid int not null,
foreign key (cheid) references Employee(eid),
foreign key (catid) references Category(catid)
on delete cascade
on update cascade
);
```

- **Worker**:

Worker(**weid FK**, catid FK)

```
create table Worker(
weid int primary key,
catid int not null,
foreign key (weid) references Employee(eid),
foreign key (catid) references Category(catid)
on delete cascade
on update cascade
);
```

- **Customer**:

Customer(**uid# PK**, fname, lname, phone, email, gender, dob, age(), hno, street, district, city, state, pincode)

```
create table Customer(
uid int primary key AUTO_INCREMENT,
fname varchar(20)not null,
lname varchar(20) default "-",
phone char(12) not null,
email varchar(50) not null, check(email like '%_@__%.__%'),
gender varchar(9) not null, check(gender in ('Male', 'Female', 'Others')),
dob date not null,  check(year(dob) < 2022),
age int generated always as(2022-year(dob)) virtual,
hno int default -1, check(hno >= 0),
street varchar(150) not null,
district varchar(50) not null,
city varchar(20) not null,
state varchar(30) not null,
pincode int not null, check(pincode between 000000 and 999999)
);
```

- **Views**:

Views(**timstamp PK**, **uid# FK**, **catid# FK**)

```
create table Views(
timstamp timestamp not null DEFAULT current_timestamp(),
uid int,
catid int,
primary key(timstamp,uid,catid),
foreign key (uid) references Customer(uid),
foreign key (catid) references Category(catid)
        on delete cascade
        on update cascade
);
```

- **Products**:

Products(**pdid# PK**, catid# FK,name, brand, images, oftype, costPrice, sellingPrice,discount, rating)

```
create table Products(
        pdid int primary key AUTO_INCREMENT,
        catid int,
   name varchar(100) not null,
   brand varchar(50) not null,
   images varchar(255) not null,
   oftype varchar(50) not null,
   costPrice int not null, check(costPrice >= 0),
   sellingPrice int not null, check(sellingPrice >= 0),
   discount int not null, check(discount >= 0),
   rating int not null, check (rating between 1 and 5),
   foreign key(catid) references Category(catid)
   on update cascade
);
```

- **<u>EAppliances</u>**:

EAppliances(**<u>pdid# FK</u>**, **<u>mfgdate PK</u>**, quantity)

```
create table Eappliances(
        pdid int,
   mfgDate date not null,
   quantity int not null, check(quantity >= 0),
   primary key(pdid, mfgdate),
   foreign key(pdid) references Products(pdid)
   on delete cascade
   on update cascade
);
```

- **<u>Footwears</u>**:

Footwears(**<u>pdid# FK, color PK, gender PK, size PK</u>**, quantity)

```
create table Footwears(
        pdid int,
        color varchar(20) not null,
        gender varchar(20) not null,
        size int not null, check(size >= 0),
   quantity int not null, check(quantity >= 0),
        primary key(pdid, color, gender, size),
        foreign key(pdid) references Products(pdid)
        on delete cascade
   on update cascade
);
```

- **<u>Clothes</u>**:

Clothes(**<u>pdid# FK, color PK, gender PK, size PK</u>**, quantity)

```
create table Clothes(
        pdid int,
        color varchar(20) not null,
        gender varchar(20) not null,
        size varchar(20) not null, check(size in('M', 'L', 'S', 'XL', 'XXL', 'XXXL')),
    quantity int not null, check(quantity >= 0),
        primary key(pdid, color, gender, size),
        foreign key(pdid) references Products(pdid)
        on delete cascade
    on update cascade
);
```

- **<u>Corders</u>**:

Corders(**<u>cordid# PK</u>,** uid# FK, dateoforderplaced, dateoforderdelivery, orderstatus, totalCost)

```
create table Corders(
cordid int primary key AUTO_INCREMENT,
uid int not null,
dateoforderplaced date not null,
dateoforderdelivery date not null, check(date(dateoforderdelivery) >= date(dateoforderplaced)),
orderstatus boolean default false,
totalCost int default 0, check(totalCost >= 0),
foreign key(uid) references Customer(uid)
on delete cascade
on update cascade
);
```

- **<u>Transactions</u>**:

Transactions(**<u>tid# PK</u>**, cordid# FK,ofstatus, timstamp, paymentmethod)

```
create table Transactions(
tid int primary key AUTO_INCREMENT,
cordid int,
ofstatus boolean not null,
timstamp timestamp not null DEFAULT current_timestamp(),
paymentmethod varchar(10) not null, check(paymentmethod in('COD', 'UPI')),
foreign key(cordid) references Corders(cordid)
on delete cascade
on update cascade
);
```

- **<u>Bank</u>**:

Bank(------)

- **<u>Cart</u>**:

Cart(**<u>ordid# FK</u>**, **<u>pdid# FK</u>**, **<u>quantity</u>**, subtotal, **<u>attr, placed</u>**)

```
create table Cart(
ordid int,
pdid int,
quantity int default 0, check(quantity >= 0),
subtotal int default 0,  check(subtotal >= 0),
attr varchar(80) not null,
placed boolean default false,
primary key(ordid, pdid, quantity, attr, placed),
foreign key(ordid) references Corders(cordid),
foreign key (pdid) references Products(pdid)
        on delete cascade
   on update cascade
);
```

- **<u>Login</u>**:

login(**<u>loginID# FK</u>**, **<u>usertype# PK</u>**, password**)**

```
create table login(
loginID int not null, check(loginID >= 0),
usertype varchar(20) not null, check ( usertype in('customer','manager','employee')),
pasword varchar(12) not null, check (LENGTH(pasword) >= 6 and LENGTH(pasword) <= 12),
primary key (loginID,usertype)
);
```

- **DeliveryPerson**

DeliveryPerson(**<u>cordid#FK,</u>** deid FK)

```
create table DeliveryPerson(
cordid int primary key,
deid int,
foreign key(cordid) references Corders(cordid),
foreign key(deid) references Employee(eid)
on delete cascade
on update cascade
);
```

Weak Entity

InInventory: Weak entity because it is a means of storing products in our inventory. It has no primary key.
Cart: Cannot exist without products, needs products to exist. Only has foreign keys as its primary key

Ternary relationship:

Transaction, Customer, Corders: Customer needs to place order, Customer also needs to fulfil transaction by paying. Also, Corders generates a transaction.

Entity Relationship Table

ENTITY 1	**ENTITY 2**	**RELATIONSHIP**	**TYPE**
Vendor	Invoice	orders	One to Many
Invoice	Batch	belongsto	One to Many
MOrders-Aggregate	products	contains	Many to Many
Inventory	InInventory	stores	One to Many
InInventory	Products	contains	Many to One
MOrders-Aggregate	Inventory	distributes	Many to Many
Manager	Inventory	manages	One to one
Manager	Employee	manages	One to Many
Manager	MOrders-Aggregate	ordered	One to Many
Employee	Category	Works	Many to one
Employee	Category	cathead	One to one
Employee	COrders	Delivers	One to Many
Employee	Employee	Supervisor	One to Many
Category	Products	Contains	Many to One
Products	Clothes	isA	One to Many

Products	Footwear	isA	One to Many
Products	EAppliances	isA	One to Many
Customer	Category	views	Many to Many
Cart	Product	Contains	Many to One
Customers	COrders	Generates	One to Many
Customers	Transaction	Generates	One to Many
COrders	Transaction	Generates	One to One
Orders	Cart	from	Many to One
Bank	Transaction	verifies	One to Many
Customer	User	Is A	Many to One
Employee	User	Is A	Many to One
Manager	User	Is A	Many to One
User	Login	login_details	One to Many

Triggers:

Discount:

```
create  trigger doDiscount
before insert on Products
for each row
set new.discount=new.costPrice*0.1,
new.sellingPrice=new.costPrice-new.discount;


create  trigger discountDu
before update on Products
for each row
set new.discount=new.costPrice*0.1,
new.sellingPrice=new.costPrice-new.discount;
```

Inserting item in cart:

```
create trigger jbcartmeadd1
after insert on Cart
for each row
```

```
update Eappliances
set quantity=quantity-new.quantity
where Eappliances.pdid=new.pdid;


create trigger jbcartmeadd2
after insert on Cart
for each row
update Footwears
set quantity=quantity-new.quantity
where Footwears.pdid=new.pdid;


create trigger jbcartmeadd3
after insert on Cart
for each row
update Clothes
set Clothes.quantity=Clothes.quantity-new.quantity
where Clothes.pdid=new.pdid;
```

OnInventory:

```
create trigger onIninventory1
after insert on Cart
for each row
update Footwears,InInventory
set Footwears.quantity=10+Footwears.quantity,
InInventory.quantity=InInventory.quantity-10
where Footwears.quantity<5 and Footwears.pdid=new.pdid and InInventory.pdid=new.pdid;


create trigger onIninventory2
after insert on Cart
for each row
update Clothes,InInventory
set Clothes.quantity=10+Clothes.quantity,
InInventory.quantity=InInventory.quantity-10
where Clothes.quantity<5 and Clothes.pdid=new.pdid and InInventory.pdid=new.pdid;


create trigger onIninventory3
after insert on Cart
for each row
update Eappliances,InInventory
set Eappliances.quantity=10+Eappliances.quantity,
InInventory.quantity=InInventory.quantity-10
where Eappliances.quantity<5 and Eappliances.pdid=new.pdid and InInventory.pdid=new.pdid;

DELIMITER $$
CREATE TRIGGER InInventorypr
BEFORE Update
ON InInventory
FOR EACH ROW
BEGIN
  if (NEW.quantity < 10) THEN
        set NEW.quantity = NEW.quantity + 15;
  END IF;
END$$
```

Views:

```
--views
--Manager
create view InInventoryViewMan as
SELECT InInventory.pdid, Products.name, Products.brand, InInventory.quantity,
InInventory.invenid
FROM InInventory JOIN Products ON InInventory.pdid = Products.pdid;

create view VendorViewMan as
SELECT Vendor.venid, Vendor.fname, Vendor.lname, Vendor.email, Vendor.phone,
Vendor.gender, Vendor.hno, Vendor.street, Vendor.district, Vendor.city,
Vendor.state, Vendor.pincode, Distributes.invenid FROM Vendor JOIN Distributes
ON Vendor.venid = Distributes.venid;

create view InvoiceUnderMan as
SELECT Invoice.invid, Invoice.invenid, Invoice.statusof, Invoice.receivedDate,
Invoice.fulfilledDate, Morders.venid FROM Invoice JOIN Morders
ON Invoice.invid = Morders.invid;

create view BatchView as
SELECT Products.name, Products.brand, Products.oftype, Batch.quantity, Batch.invid,
Morders.venid, Invoice.invenid FROM Invoice JOIN Products JOIN Batch JOIN Morders
ON Invoice.invid = Morders.invid AND Invoice.invid = Batch.invid AND Products.pdid =
Batch.pdid;


--Category Head
create view InventoryViewCat as
SELECT Inventory.invenid, Inventory.hno, Inventory.street, Inventory.district,
Inventory.city, Inventory.state, Inventory.pincode, Employee.eid FROM Inventory JOIN
```

```
Manager JOIN Supervision JOIN Employee ON Inventory.invenid = Manager.invenid AND
Manager.meid = Supervision.meid AND
Supervision.eid = Employee.eid;

create view InInventoryViewCat as
SELECT DISTINCT InInventory.pdid, Products.name, Products.brand, InInventory.quantity,
Employee.eid
FROM Products JOIN InInventory JOIN Inventory JOIN Manager JOIN Supervision JOIN
Employee JOIN CatHead ON
InInventory.pdid = Products.pdid AND Inventory.invenid = InInventory.invenid AND
Manager.invenid = Inventory.invenid
AND Manager.meid = Supervision.meid AND Supervision.eid = Employee.eid AND
Employee.eid = CatHead.cheid AND
Products.catid = CatHead.catid;

create view EmployeeUnderCat as
SELECT E1.eid, E1.fname, E1.lname, E1.phone, E1.age, E1.salary, S1.superviser_eid
FROM Employee E1 JOIN Supervisor S1 ON E1.eid = S1.supervisee_eid;

create view ProductsUnderCat as
SELECT DISTINCT Products.pdid, Products.catid, Products.name, Products.brand,
Products.images,
Products.oftype, Products.costPrice, Products.sellingPrice, Products.discount, Products.rating,
CatHead.cheid FROM Products JOIN CatHead ON Products.catid = CatHead.catid;

--DeliveryPerson
create view Shippings as
SELECT D.cordid, C.dateoforderplaced, C.dateoforderdelivery, C.orderstatus, C.totalCost,
U.uid,
U.fname, U.lname, U.phone, U.email, U.hno, U.street, U.district, U.city, U.state, U.pincode,
D.deid
FROM DeliveryPerson D JOIN Corders C JOIN Customer U ON D.cordid = C.cordid AND C.uid
= U.uid
ORDER BY C.orderstatus=0;


--Worker
create view InventoryViewWork as
SELECT DISTINCT Inventory.invenid, Inventory.hno, Inventory.street, Inventory.district,
Inventory.city,
Inventory.state, Inventory.pincode, Employee.eid FROM Inventory JOIN Manager JOIN
Supervision JOIN Employee JOIN Supervisor JOIN
CatHead ON Employee.eid = Supervisor.supervisee_eid AND Supervisor.superviser_eid =
CatHead.cheid AND
```

CatHead.cheid = Supervision.eid AND Supervision.meid = Manager.meid AND Inventory.invenid = Manager.invenid;

create view InInventoryViewWork as
SELECT DISTINCT InInventory.pdid, Products.name, Products.brand, InInventory.quantity, Employee.eid
FROM Products JOIN InInventory JOIN Inventory JOIN Manager JOIN Supervision JOIN Employee JOIN
CatHead JOIN Supervisor ON InInventory.pdid = Products.pdid AND Inventory.invenid = InInventory.invenid AND
Manager.invenid = Inventory.invenid AND Manager.meid = Supervision.meid AND
Supervision.eid = CatHead.cheid AND CatHead.cheid = Supervisor.superviser_eid AND
Supervisor.supervisee_eid = Employee.eid AND Products.catid = CatHead.catid;

create view ProductsUnderWork as
SELECT DISTINCT Products.pdid, Products.catid, Products.name, Products.brand, Products.images,
Products.oftype, Products.costPrice, Products.sellingPrice, Products.discount, Products.rating,
Employee.eid FROM Products JOIN CatHead JOIN Supervisor JOIN Employee ON
Products.catid = CatHead.catid AND CatHead.cheid = Supervisor.superviser_eid AND
Supervisor.supervisee_eid = Employee.eid;

--Store
create view CartView as
SELECT Cart.pdid, Cart.quantity, Cart.subtotal, Cart.attr, Cart.placed, Products.name, Products.sellingPrice, Products.catid, Cart.ordid FROM Cart JOIN Products ON Cart.pdid = Products.pdid
WHERE Cart.placed = 0;

--**user**
create view InCartView as
SELECT Cart.pdid, Cart.quantity, Cart.subtotal, Products.catid,
Products.name, Products.sellingPrice, Cart.ordid
FROM Cart JOIN Products ON Cart.pdid = Products.pdid
WHERE Cart.placed = 1;

create view DeliveryPersonView as
select DIStINCT eid
FROM Supervision
WHERE eid NOT IN (SELECT DISTINCT cheid FROM CatHead);

```
create view OrderView as
SELECT Cart.quantity, Cart.subtotal, Products.name,
Products.sellingPrice, Cart.ordid
FROM Cart JOIN Products ON Cart.pdid = Products.pdid
WHERE Cart.placed = 0;
```

GRANTS:

```
--Grants
CREATE USER 'Customer'@'localhost' IDENTIFIED BY 'Root#1234';
CREATE USER 'Manager'@'localhost' IDENTIFIED BY 'Root#1234';
CREATE USER 'CatHead'@'localhost' IDENTIFIED BY 'Root#1234';
CREATE USER 'DeliveryPerson'@'localhost' IDENTIFIED BY 'Root#1234';
CREATE USER 'Worker'@'localhost' IDENTIFIED BY 'Root#1234';

--Inventory
GRANT SELECT
ON Store.Inventory TO 'Manager'@'localhost';

--Inventory
GRANT ALL PRIVILEGES
ON Store.InInventory TO 'Manager'@'localhost';

--Products
GRANT UPDATE, SELECT
ON Store.Products TO 'Worker'@'localhost';

GRANT ALL
ON Store.Products TO 'CatHead'@'localhost';

GRANT SELECT
ON Store.Products TO 'Customer'@'localhost';

--Clothes
GRANT ALL
ON Store.Clothes TO 'CatHead'@'localhost';
```

```
GRANT SELECT
ON Store.Clothes TO 'Customer'@'localhost';

--Footwear
GRANT ALL
ON Store.Footwears TO 'CatHead'@'localhost';

GRANT SELECT
ON Store.Footwears TO 'Customer'@'localhost';

--Eappliances
GRANT ALL
ON Store.Eappliances TO 'CatHead'@'localhost';

GRANT SELECT
ON Store.Eappliances TO 'Customer'@'localhost';


--Customer
GRANT SELECT, UPDATE
ON Store.Customer TO 'Customer'@'localhost';

--Manager
GRANT SELECT, UPDATE
ON Store.Manager TO 'Manager'@'localhost';

--Emploeee
GRANT ALL
ON Store.Employee TO 'Manager'@'localhost';

GRANT ALL
ON Store.Employee TO 'CatHead'@'localhost';

GRANT SELECT, UPDATE
ON Store.Employee TO 'DeliveryPerson'@'localhost';

GRANT SELECT, UPDATE
ON Store.Employee TO 'Worker'@'localhost';

--DeliveryPerson
GRANT SELECT
ON Store.DeliveryPerson TO 'DeliveryPerson'@'localhost';

GRANT SELECT, INSERT
```

```
ON Store.DeliveryPerson TO 'Customer'@'localhost';

--Worker
GRANT SELECT, INSERT, DELETE
ON Store.Worker TO 'CatHead'@'localhost';

--Corders
GRANT SELECT, UPDATE
ON Store.Corders TO 'DeliveryPerson'@'localhost';

GRANT SELECT, INSERT
ON Store.Corders TO 'Customer'@'localhost';


--Morders
GRANT SELECT, INSERT
ON Store.Morders TO 'Manager'@'localhost';

--Vendor
GRANT SELECT
ON Store.Vendor TO 'Manager'@'localhost';

--Transactions
GRANT SELECT, INSERT, DELETE
ON Store.Transactions TO 'Customer'@'localhost';

--Cart
GRANT ALL
ON Store.Cart TO 'Customer'@'localhost';

--Views
GRANT SELECT, INSERT
ON Store.Views TO 'Customer'@'localhost';

--Distributes
GRANT SELECT, INSERT
ON Store.Distributes TO 'Manager'@'localhost';

--Batch
GRANT SELECT, INSERT
ON Store.Batch TO 'Manager'@'localhost';
```

```
--Invoice
GRANT SELECT, INSERT
ON Store.Invoice TO 'Manager'@'localhost';

--login
GRANT SELECT
ON Store.login TO 'Customer'@'localhost', 'Manager'@'localhost', 'DeliveryPerson'@'localhost',
'Worker'@'localhost', 'CatHead'@'localhost';

--Supervisor
GRANT SELECT, INSERT
ON Store.Supervisor TO 'CatHead'@'localhost';

--Supervision
GRANT SELECT, INSERT
ON Store.Supervision TO 'Manager'@'localhost';


--CatHead
GRANT SELECT, INSERT, DELETE
ON Store.CatHead TO 'Manager'@'localhost';

GRANT SELECT
ON Store.CatHead TO 'CatHead'@'localhost';

--Category
GRANT SELECT
ON Store.Category TO 'CatHead'@'localhost', 'Worker'@'localhost', 'Customer'@'localhost';

-- GRANTS ON VIEWS
--BatchView
GRANT SELECT
ON Store.BatchView TO 'Manager'@'localhost';

--CartView
GRANT SELECT
ON Store.CartView TO 'Customer'@'localhost';

--EmployeeUnderCat
GRANT SELECT
ON Store.EmployeeUnderCat TO 'CatHead'@'localhost';
```

```
--InCartView
GRANT SELECT
ON Store.InCartView TO 'Customer'@'localhost';

--InInventoryViewCat
GRANT SELECT
ON Store.InInventoryViewCat TO 'CatHead'@'localhost';

--InInventoryViewMan
GRANT SELECT
ON Store.InInventoryViewMan TO 'Manager'@'localhost';

--InInventoryViewWork
GRANT SELECT
ON Store.InInventoryViewWork TO 'Worker'@'localhost';

--InventoryViewCat
GRANT SELECT
ON Store.InventoryViewCat TO 'CatHead'@'localhost';

--InventoryViewWork
GRANT SELECT
ON Store.InventoryViewWork TO 'Worker'@'localhost';

--InvoiceUnderMan
GRANT SELECT
ON Store.InvoiceUnderMan TO 'Manager'@'localhost';

--OrderView
GRANT SELECT
ON Store.OrderView TO 'Customer'@'localhost';

--ProductsUnderCat
GRANT SELECT
ON Store.ProductsUnderCat TO 'CatHead'@'localhost';

--ProductsUnderWork
GRANT SELECT
ON Store.ProductsUnderWork TO 'Worker'@'localhost';

--Shippings
GRANT SELECT
ON Store.Shippings TO 'DeliveryPerson'@'localhost';
```

```
--VendorViewMan
GRANT SELECT
ON Store.VendorViewMan TO 'Manager'@'localhost';

--DeliveryPersonView
GRANT SELECT
ON Store.DeliveryPersonView TO 'Customer'@'localhost';
```

Queries:

New Queries:

1. List the inventory id details which contains the product which is most bought by customer with id 2

```
create index onq1
on Cart(pdid);

select *
from InInventory
where pdid in (
  select pdid
  from Cart
  where pdid in (
    select max(quantity)
    from Cart
    where ordid in (
      select ordid
      from Corders
      where uid = 2
    )
  )
);
```

2. Find all products that were purchased atleast twice in the year 2019.

```
create index onq2
on Cart(pdid)

Select Cart.pdid
from Cart,Corders
where Cart.ordid in (
  select cordid
  from Corders
  where Year(Corders.dateoforderplaced) = 2019
```

```
   )
Group by Cart.pdid
having (Count(*) > 1);
```

3.Find all the delivery person who have delivered more than 1 orders

```
create index onq3
on DeliveryPerson(deid);

Select deid from DeliveryPerson
group by deid
having count(*)>2;
```

4.Increment the salary of all the employees by 10 percent who have been working in the store for more than 10 years.

```
create index onq4
on Employee(experience);

update Employee
set salary=salary+0.1*salary
where experience>=10;
```

5. Find the total sales in each category.

```
create index onq5
on Products(pdid);

select Products.catid, sum(Cart.quantity)
from Products, Cart
where Products.pdid = Cart.pdid
group by Products.catid;
```

6.Find the name of the customers who have ordered any cloth of color blue.

```
create index onq6
on Clothes(pdid, color);

select fname, lname from Customer
where uid in (
  select uid
  from Corders
        where cordid in (
    select cordid
    from Cart
                where pdid in (
      select Products.pdid
      from Products, Clothes
      where Clothes.pdid = Products.pdid and Clothes.color = 'blue'
            )
        )
    );
```

7.Find the name and phone number of the customers who have ever chosen the payment method as Cash on Delivery.

```
create index onq7
on Corders(cordid);

select fname, lname, phone
from Customer
where uid in(select uid from Corders
                    where cordid in (
          select  cordid
          from Transactions
                                  where paymentmethod = 'COD'
                                )
                );
```

8. List all the Employees whose salary is greater than the average salary of the all the specialities.

```
create index onq8
on Employee(speciality);

select *
from Employee
where salary>all(
  select avg(salary)
  from Employee
  group by speciality
  );
```

9.List all the workers who work in the clothes category

```
create index onq9
on Employee(eid);

Select *
from Employee
where eid in(Select weid
        from Worker
        where catid=1
        );
```

10.List all Delivery persons whose city is same as that of the order delivered by him.

```
create index onq10
on DeliveryPerson(cordid);

Select E.eid from Employee as E, DeliveryPerson as D
where E.eid=D.deid and D.cordid in(select cordid
                      from Corders
                      where uid in(Select uid
                              from Customer
                              where city=E.city
                              )
);
```

Instruction to operate the project:

First extract the zip file at the desired location then create a virtual environment in the Ecommerce folder of the extracted zip. Now activate the venv which will be followed by installation of the Flask and Python connector in the venv. Now open the sql workbench and run the provided sql file named as RetailStore.sql this will create the full database in the system with the required grants and views over the different tables of the same database.

Now open adAuth.py, admin.py, auth.py and change the database password to the root password of your sql server. Now run the init.py file with the way that suits you. Then open the link in your browser.

The browser will open the Customer login page, which is the gateway of the Ecommerce website. Now one can explore the Retail Store and can add a product to the cart and view the products category viz. and order them. The generated checkout can be viewed by accessing the my account tab. My account have a lot of information and some rights to change the personal information of the customer.

ADMIN: To access the admin page we have to go for /adAuth. Where an admin can go for Manager and Employee using the login.

Admin will redirect the admin part to their respective sections and where an admin can access and explore the allotted rights.

E-R Diagram :- https://app.diagrams.net/#G1q_4-IFjgyI9kh-LwkLhI4BAmG-dMaXk9

GITHUB REPO :- https://github.com/ShrJatin/RetailStore.git

Contributors:

Ayush Kumar Singh-
Jatin Sharma
Prachi
Sourav Goyal