**Database Project Report**

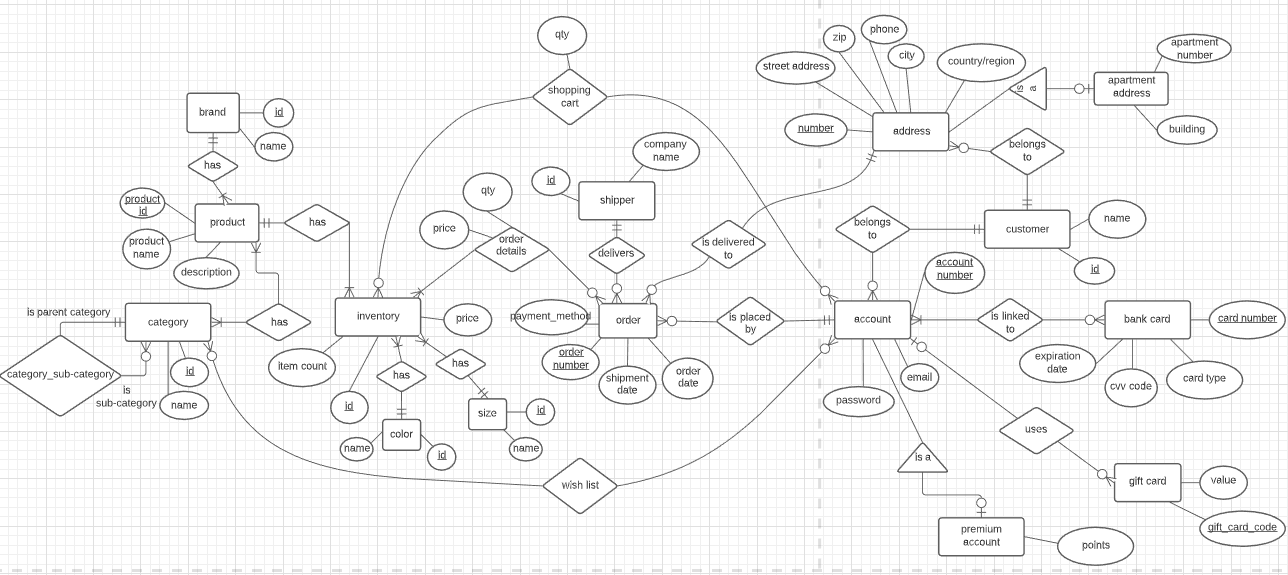
Hieu Dao Le Duc #192489,

Tyler Messina-Katunar #1955092

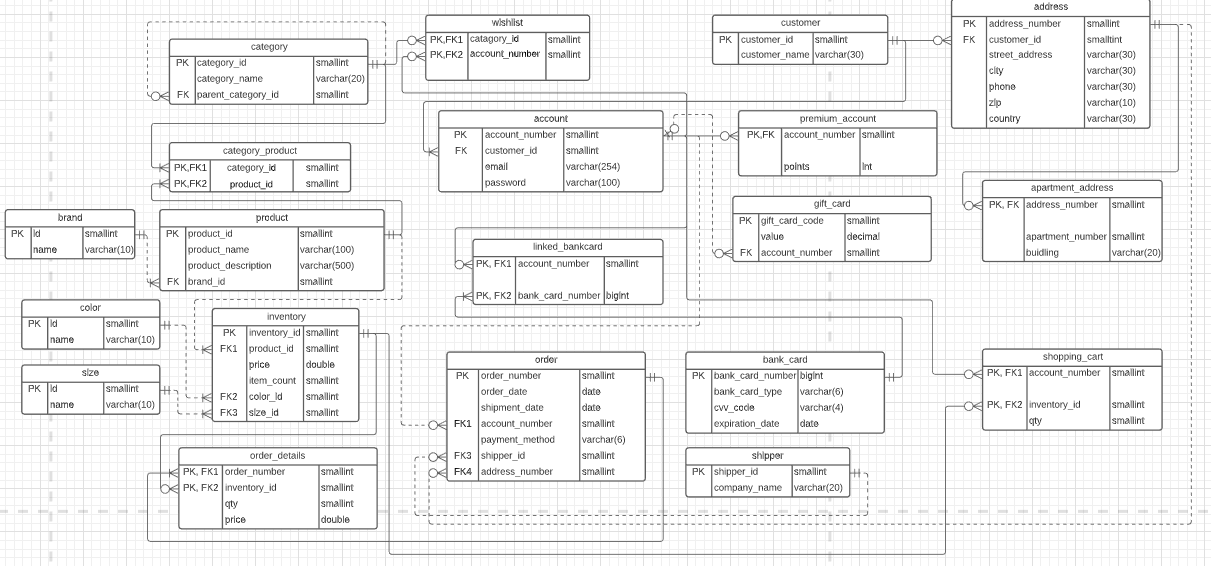
JIANJUAN GAO #1831989

YAOYU WANG #1423615

**Conceptual Design**

****

**Physical Design**

****

**Link to lucidchart:**

[**https://app.lucidchart.com/lucidchart/invitations/accept/50f1094c-f3f8-43a7-a2ed-191cd3bae321**](https://app.lucidchart.com/lucidchart/invitations/accept/50f1094c-f3f8-43a7-a2ed-191cd3bae321)

**1. SQL**

drop database if exists online\_shopping;

create database online\_shopping;

use online\_shopping;

# Customer

create table customer

(customer\_id smallint primary key auto\_increment,

customer\_name varchar(30) not null

);

# Address

create table address

(address\_number smallint primary key auto\_increment,

customer\_id smallint,

street\_address varchar(30) not null,

city varchar(30) not null,

phone varchar(20) not null,

zip varchar(10) not null,

country varchar(30) not null,

foreign key(customer\_id) references customer(customer\_id)

);

# Apartment address

create table apt\_address

(address\_number smallint primary key,

apt\_number smallint not null,

building varchar(20) not null,

foreign key(address\_number) references address(address\_number)

);

# Account

create table `account`

(account\_number smallint primary key auto\_increment,

customer\_id smallint,

email varchar(254) not null,

password varchar(100) not null,

foreign key(customer\_id) references customer(customer\_id)

);

# Premium account

create table premium\_account

(account\_number smallint primary key,

points int not null,

foreign key(account\_number) references `account`(account\_number)

);

# Bank card

create table bankcard

(bankcard\_number bigint,

card\_type varchar(6) not null,

cvv\_code smallint not null,

expiration\_date date not null,

primary key(bankcard\_number)

);

# Account & Bankcard

create table linked\_bankcard

(account\_number smallint,

bankcard\_number bigint,

primary key(account\_number, bankcard\_number),

foreign key(account\_number) references `account`(account\_number),

foreign key(bankcard\_number) references bankcard(bankcard\_number)

);

# Gift card

create table gift\_card

(gift\_card\_code smallint primary key,

value decimal not null,

account\_number smallint,

foreign key(account\_number) references `account`(account\_number)

);

# Category

create table category(

category\_id smallint primary key auto\_increment,

category\_name varchar(20) not null unique,

parent\_category\_id smallint,

foreign key(parent\_category\_id) references category(category\_id)

);

# Wishlist

create table wishlist

(category\_id smallint,

account\_number smallint,

primary key(category\_id, account\_number),

foreign key(category\_id) references category(category\_id),

foreign key(account\_number) references `account`(account\_number)

);

# Brand

create table brand

(brand\_id smallint primary key auto\_increment,

brand\_name varchar(20) not null unique

);

# Product

create table product

(product\_id smallint primary key auto\_increment,

product\_name varchar(100) not null unique,

product\_description varchar(100) not null,

brand\_id smallint not null,

foreign key(brand\_id) references brand(brand\_id));

# Category & Product

create table category\_product

(category\_id smallint,

product\_id smallint,

primary key(category\_id, product\_id),

foreign key(category\_id) references category(category\_id),

foreign key(product\_id) references product(product\_id)

);

# Color

create table color

(color\_id smallint primary key auto\_increment,

color\_name varchar(30) not null unique

);

# Size

create table size

(size\_id smallint primary key auto\_increment,

size\_name varchar(30) not null unique

);

# Inventory

create table inventory(

inventory\_id smallint primary key auto\_increment,

product\_id smallint not null,

price double not null,

item\_count smallint not null,

color\_id smallint,

size\_id smallint,

foreign key(product\_id) references product(product\_id),

foreign key(color\_id) references color(color\_id),

foreign key(size\_id) references size(size\_id)

);

# Shopping cart

create table shopping\_cart

(account\_number smallint,

inventory\_id smallint,

qty smallint not null,

primary key(account\_number, inventory\_id),

foreign key(account\_number) references `account`(account\_number),

foreign key(inventory\_id) references inventory(inventory\_id)

);

# Shipper

create table shipper(

shipper\_id smallint primary key auto\_increment,

company\_name varchar(20) not null

);

# Order

create table `order`

(order\_number smallint primary key auto\_increment,

order\_date date not null,

shipment\_date date,

account\_number smallint not null,

payment\_method varchar(6) not null,

shipper\_id smallint,

address\_number smallint not null,

foreign key(account\_number) references `account`(account\_number),

foreign key(shipper\_id) references shipper(shipper\_id),

foreign key(address\_number) references address(address\_number)

);

# Order details

create table order\_details

(order\_number smallint,

inventory\_id smallint,

qty smallint not null,

price double not null,

primary key(order\_number, inventory\_id),

foreign key(order\_number) references `order`(order\_number),

foreign key(inventory\_id) references inventory(inventory\_id)

);

# Display all orders and order details of all customers, including

# the quantity and price paid for every item in the order and the total price paid for each order:

select \*, (select sum(od1.qty \* od1.price) from order\_details od1 where od.`Order No.` = od1.order\_number group by od1.order\_number) 'Total Order Price' from

(select c.customer\_name 'Customer', a.account\_number from customer c, account a where c.customer\_id = a.customer\_id) c,

(select `Order No.`, `Product Name`, `Color`, `Size`, `Unit Price`, `Qty`, `Account No.`, `Order Date`, `Shipment Date`, `Shipper ID`, `Company`, `Apt.`, `Building`, `Address`, `Phone`, `Unit Price` \* `Qty` 'Total Unit Price'

from (select i.inventory\_id, p.product\_name 'Product Name', c.color\_name 'Color', s.size\_name 'Size'

from product p, inventory i, color c, size s

where p.product\_id = i.product\_id and i.color\_id = c.color\_id and i.size\_id = s.size\_id) i,

(select o.\*, od.inventory\_id, od.qty 'Qty', od.price 'Unit Price' from

(select o.order\_number 'Order No.', o.account\_number 'Account No.', o.order\_date 'Order Date', o.shipment\_date 'Shipment Date',

sh.shipper\_id 'Shipper ID', sh.company\_name 'Company',

a.apt\_number 'Apt.', a.building 'Building', concat(a.street\_address, ', ', a.city, ', ', a.country, ', ', a.zip) 'Address', a.phone 'Phone'

from `order` o, shipper sh, (select a1.address\_number, a2.apt\_number, a2.building, a1.street\_address, a1.city, a1.country, a1.zip, a1.phone from address a1 left join apt\_address a2 on a1.address\_number = a2.address\_number) a

where o.shipper\_id = sh.shipper\_id and o.address\_number = a.address\_number) o,

order\_details od

where o.`Order No.` = od.order\_number) od

where i.inventory\_id = od.inventory\_id) od

where c.account\_number = od.`Account No.` order by `Order No.`;

# Display the frequency distribution of how many orders were

# placed by how many customers. Name your columns as 'number of orders' and 'number of customers'.

select `No. of orders`, count(noOfOrders.customer\_id) 'No. of customers' from

(select c.customer\_id, count(o.order\_number) 'No. of orders' from (select c.customer\_id, a.account\_number from customer c left join account a on c.customer\_id = a.customer\_id) c left join

`order` o on c.account\_number = o.account\_number group by c.account\_number) noOfOrders group by `No. of orders`;

**2. Data**

# ADD A NEW CUSTOMER

drop procedure if exists add\_customer;

delimiter //

create procedure add\_customer(in customerName varchar(30))

begin

insert into customer(customer\_name) value(customerName);

end //

delimiter ;

# call add\_customer('New customer');

# select \* from customer where customer\_name = 'New customer';

# CREATE A NEW ACCOUNT

drop procedure if exists create\_account;

delimiter //

create procedure create\_account(in customerId smallint,

in email varchar(254),

in `password` varchar(100))

begin

# # if this customer hasn't used this email address

if not exists (select 1 from account a where a.customer\_id = customerId and a.email = email) then

insert into account(customer\_id, email, password) value(customerId, email, `password`);

else signal sqlstate '43095' set message\_text = 'You already had an account using this email address';

end if;

end //

delimiter ;

# call create\_account(22, 'abc@gmail.com', '123');

# select \* from account where customer\_id = 22;

# SELECT user FROM mysql. user;

# show grants for 'abc@gmail.com';

# INSERT A NEW PRODUCT

drop procedure if exists insert\_product;

delimiter //

create procedure insert\_product(in productName varchar(100),

in description varchar(100),

in brandName varchar(20))

begin

# if this brand does not exist

if not exists (select 1 from brand where brand\_name = brandName) then

# create new brand

insert into brand (brand\_name) value (brandName);

end if;

insert into product (product\_name, product\_description, brand\_id) value(productName, description, (select brand\_id from brand where brand\_name = brandName));

end //

delimiter ;

# call insert\_product('New product', 'abc', 'brand');

# select \* from brand where brand\_name = 'brand';

# select \* from product where product\_name = 'New product';

# ADD A PRODUCT TO INVENTORY

drop procedure if exists add\_to\_inventory;

delimiter //

create procedure add\_to\_inventory(in productName varchar(100),

in color varchar(30),

in size varchar(30),

in price double,

in availableQty smallint)

begin

set @productId = (select product\_id from product where product\_name = productName);

if @productId is null then

signal sqlstate '43096' set message\_text = 'This product does not exist. You need to create this product first';

end if;

if not exists (select 1 from color where color\_name = color) then

# create new color

insert into color (color\_name) value (color);

end if;

if not exists (select 1 from size where size\_name = size) then

# Insert new size

insert into size (size\_name) value (size);

end if;

if price < 0 then

signal sqlstate '43096' set message\_text = 'Price cannot be negative';

end if;

if availableQty < 0 then

signal sqlstate '43096' set message\_text = 'Available quantity cannot be negative';

end if;

insert into inventory (product\_id, price, item\_count, color\_id, size\_id) value(@productId, price, availableQty, (select color\_id from color where color\_name = color), (select size\_id from size where size\_name = size));

end //

delimiter ;

# call add\_to\_inventory('New product', 'color', 'size', 1, 1);

# SET UP PAYMENT METHOD

drop procedure if exists setup\_payment\_method;

delimiter //

create procedure setup\_payment\_method(in accountNumber smallint, in paymentMethod varchar(6), out result tinyint)

begin

# if at least a suitable card is found

if exists (select 1 from linked\_bankcard lk, bankcard b where account\_number = accountNumber and lk.bankcard\_number = b.bankcard\_number and card\_type = paymentMethod) then

set result = 1;

end if;

end //

delimiter ;

# set @result = 0;

# call setup\_payment\_method(1, 'debit', @result);

# call setup\_payment\_method(1, 'credit', @result);

# select @result;

# BEGIN AN ORDER

drop procedure if exists begin\_order;

delimiter //

create procedure begin\_order(in accountNumber smallint,

in paymentMethod varchar(6),

in orderDate date,

in addressNumber smallint)

begin

set @result = 0;

call setup\_payment\_method(accountNumber, paymentMethod, @result);

if @result is null then

signal sqlstate '43097' set message\_text = 'No suitable card is found for this payment method';

elseif @result = 1 then

set @customerIdOfThisAccount = (select customer\_id from account where account\_number = accountNumber);

set @customerIdOfThisAddress = (select customer\_id from address where address\_number = addressNumber);

if (@customerIdOfThisAccount != @customerIdOfThisAddress) then

signal sqlstate '43097' set message\_text = 'Account and Address do not match';

end if;

insert into `order`(order\_date, account\_number, payment\_method, address\_number) value(orderDate, accountNumber, paymentMethod, addressNumber);

end if;

end //

delimiter ;

# call begin\_order(1, 'debit', curdate(), 1);

# call begin\_order(1, 'credit', curdate(), 4);

# ADD ITEM TO ORDER

drop procedure if exists add\_to\_order;

delimiter //

create procedure add\_to\_order(in orderNumber smallint,

in inventoryId smallint,

in quantity smallint)

begin

set @originalPrice = (select price from inventory where inventory\_id = inventoryId);

set @recalculatedPrice = @originalPrice + 0.15 \* @originalPrice;

insert into order\_details value(orderNumber, inventoryId, quantity, @recalculatedPrice);

end //

delimiter ;

# CHECKOUT SHOPPING CART

drop procedure if exists checkout\_shopping\_cart;

delimiter //

create procedure checkout\_shopping\_cart(in orderNumber smallint,

in accountNumber smallint)

checkout:begin

declare i int default 0;

set @lastRow = (select count(\*) from shopping\_cart where account\_number = accountNumber);

if @lastRow = 0 then

leave checkout;

end if;

while i < @lastRow do

set @inventoryId = (select inventory\_id from shopping\_cart where account\_number = accountNumber limit i, 1);

set @quantity = (select qty from shopping\_cart where account\_number = accountNumber limit i, 1);

call add\_to\_order(orderNumber, @inventoryId, @quantity);

set i = i + 1;

end while;

end //

delimiter ;

# select \* from order\_details;

# begin;

# call begin\_order(1, 'credit', curdate(), 4);

# get the latest order

# set @orderNumber = (select max(order\_number) from `order`);

# call checkout\_shopping\_cart(@orderNumber, 1);

# commit;

# select \* from order\_details;

**3. Normalization**

Our order, order\_detail, customer, and product tables are in 3NF. because all attributes depend on the primary key, no partial dependencies,no transitive dependencies. Details are as following:

order table:

Step 1: What is the primary key of the table? order\_number

Step 2: Check for partial dependencies. Write your functional dependencies.

order\_number -> order\_date,shipment\_date,account\_number,payment\_method,

shipper\_id,address\_number

No partial dependencies. In 2NF

For the reason that one customer can have many accounts and addresses, so one account can have many addresses.so there are no transitive dependencies between account\_number and address\_number.it's in 3NF

order\_detail table:

Step 1: What is the primary key of the table?order\_number+inventory\_id

Step 2: Check for partial dependencies. Write your functional dependencies.

No partial dependencies

order\_number+inventory\_id -> qty, price

so, order\_detail in 2NF and there’s no transitive dependencies, in 3NF

Table customer:

Step 1: What is the primary key of the table?customer\_id

Step 2: Check for partial dependencies. Write your functional dependencies.

No partial dependencies

customer\_id -> customer\_name

so, customer in 2NF. No transitive dependencies, it is in 3NF

product table:

Step 1: What is the primary key of the table?product\_id

Step 2: Check for partial dependencies. Write your functional dependencies.

product\_id -> product\_name,product\_description,brand\_id

No partial dependencies, in 2NF. No transitive dependencies, in 3NF

**4. Indexes**

The 2 indexes being built are ***idx\_inventory\_price\_productId\_inventoryId*** and ***idx\_order\_orderDate\_orderNum***.

create index idx\_inventory\_price\_productId\_inventoryId on inventory(price, product\_id, inventory\_id);

***idx\_inventory\_price\_productId\_inventoryId*** was chosen to fit a query that searches for items in inventory between having a price in a specified range which is used very often in online stores in which we are trying to replicate. The index is organised by price, product\_id and then inventory\_id because it first looks through the WHERE clause (contains price) then it looks through the ORDER BY (product\_id then inventory\_id).

Ex: select inventory\_id, product\_id

from inventory

where price between 5 and 10

order by product\_id, inventory\_id;

create index idx\_order\_orderDate\_orderNum ON `order` (order\_date, order\_number);

***idx\_order\_orderDate\_orderNum*** was chosen to fit the query that searches for orders in between 2 dates and then returns the order\_number and its order\_date. This would be useful for getting orders between a time period so they can be managed by suppliers or to be organised and displayed to those that manage orders in the company. The index organises order\_date then order\_number in the order table since order\_date is looked up first in WHERE and then order\_number is looked for in SELECT.

Ex: select order\_number, order\_date

from `order`

where order\_date between '2020-11-20' and '2020-11-22';

**5. Database Users, Roles, and Permissions**

DROP ROLE IF EXISTS registered\_customer, administrator;

The two roles created (registered\_customer and administrator)’s permissions and why they were given is shown in the following.

**registered\_customer:**

registered\_customer represents a customer of the online shopping retailer. As such data referring to themselves like their password, email, location and orders should be visible to them (as it is their own) but other customers should not be visible as it would give them other people’s info and would lead to security breaches.

GRANT SELECT ON brand TO registered\_customer;

GRANT SELECT ON color TO registered\_customer;

GRANT SELECT ON size TO registered\_customer;

GRANT SELECT ON category TO registered\_customer;

They are able to SELECT (see) any details such as **color**, **brand**, **size** and **category** as they are bits of data often able to be selected or at least viewed by customers (like where you are able to pick from colors to search things by).

GRANT SELECT ON category\_product TO registered\_customer;

GRANT SELECT ON product TO registered\_customer;

GRANT SELECT ON inventory TO registered\_customer;

They are also able to SELECT (see) **inventory**, **product** and **category\_product** as customers are shown a selection of goods in online retailers but can’t usually add their own or change their properties meaning they can only select and can’t UPDATE or INSERT.

GRANT SELECT, UPDATE(email, password), INSERT ON account TO registered\_customer;

GRANT SELECT, UPDATE(customer\_name) ON customer TO registered\_customer;

GRANT SELECT ON premium\_account TO registered\_customer;

They are also able to SELECT **account**, **customer** and **premium\_account** with the stipulation that they are only able to see their own. As it would be dangerous to see other people’s passwords. This is because customers can usually see their own account details like email and password or name but can’t see other’s accounts They can also UPDATE their **account’**s password and email and their **customer** name as it is a standard function of sites like these to allow the changing of personal information. They can also be able to INSERT new accounts as long as their account belongs to them.

GRANT SELECT ON gift\_card TO registered\_customer;

GRANT SELECT ON linked\_bankcard TO registered\_customer;

The same concept is with the **gift\_card** and **linked\_bankcard** which they are only able to SELECT (see) their info as customers should be able to see their cards but can’t change them. This is again with the stipulation that they can only see their own cards as if they can see others; it would allow other people to use each other's bank cards.

GRANT SELECT, UPDATE(apt\_number,building), INSERT, DELETE ON apt\_address TO registered\_customer;

GRANT SELECT, UPDATE(street\_address,city,phone,zip, country), INSERT, DELETE ON address TO registered\_customer;

They can SELECT their own **apt\_address** and **address** as they should be able to see where they registered their own location so they can know where their ordered package will be sent to. They can also UPDATE **apt\_address**’ apt\_number ,building and they can UPDATE **address**’ street\_address, city, phone, zip and country in case the customer moves and needs to change where they live and need to change these. They can INSERT and DELETE **apt\_address** and **address** that are their own with the assumption that they cannot DELETE if there is only 1 address left and they can only see their own.

GRANT SELECT, INSERT, DELETE ON wishlist TO registered\_customer;

GRANT SELECT, UPDATE(inventory\_id, qty), INSERT, DELETE ON shopping\_cart TO registered\_customer;

For **wishlist** and **shopping\_cart,** registered\_customer is able to SELECT as customers are able to see their own wishlist and what's inside their shopping\_cart so they can buy things. They can also UPDATE the inventory\_id and qty of **shopping\_cart** as so they can change what resides inside their shopping cart and how much they want to buy. (Same stipulation that they can only see their own). They can also INSERT and DELETE a row into both as each row represents a single item and a shopping cart and wishlist can contain multiple items meaning they can INSERT and DELETE them.

GRANT SELECT ON order\_details TO registered\_customer;

GRANT SELECT(order\_number, order\_date, shipment\_date, account\_number, payment\_method, address\_number) ON `order` TO registered\_customer;

The tables **order** and **order\_details** both visible (SELECT) to customers. This is except for the shipper\_id in **order** because who ships the product is more internal affairs to an extent not meant to be seen by regular people who just want to buy things. Both tables are unable to DELETE, INSERT or UPDATE because the information within an order is based on company data such as shipment\_date which is highly based on internal affairs which should not be known or chosen by the customer. It also should not be possible for registered customers to change data at will as it is not their call for how long it should take to arrive.

registered\_customer doesn’t have access to things related to suppliers as suppliers are decided by workers for the company not the customer and as such they do not need to see it.

**administrator:**

GRANT ALL ON \* TO administrator;

An administrator represents the highest form of power within a system. As such they should have rights to do just about anything in a system and see everything and their innerworkings. To give administrators full power all permissions to SELECT, UPDATE, INSERT and DELETE were given to them for every table.

**6.Views**

**Registered Customer has the right to know every information about the their order such as: order number,product name,price,quantity and shipping details**

DROP VIEW IF EXISTS RegisteredCustomer;

CREATE VIEW RegisteredCustomer AS

SELECT

AC.account\_number,

O.order\_number,

P.product\_name,

B.brand\_name,

C.color\_name,

S.size\_name,

I.price 'Price (before tax)',

OD.qty,

SP.company\_name 'Shipping Company',

O.shipment\_date,

concat(

A.street\_address,',',

A.city,',',

A.country,' ',

A.zip) 'Shipping Address',

A.phone

FROM `account` AC,`order` O,product P,inventory I,brand B,color C,size S,order\_details OD,shipper SP,address A

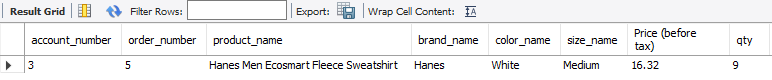
WHERE AC.account\_number=O.account\_number AND O.address\_number=A.address\_number AND O.order\_number=OD.order\_number AND I.inventory\_id=OD.inventory\_id AND I.product\_id=P.product\_id AND

O.shipper\_id=SP.shipper\_id AND I.color\_id=C.color\_id AND I.size\_id=S.size\_id and p.brand\_id=B.brand\_id

WITH CHECK OPTION;

**For example: one customer with account number 3,he or she can get any details about his or her order.**

select \* from RegisteredCustomer where account\_number =3;

****

****

# A fedex\_manager should only see shippers that work for FedEx

create role fedex\_manager;

drop view if exists fedex\_manager\_on\_shipper;

create view fedex\_manager\_on\_shipper as

select \* from shipper where company\_name = 'FedEx'

with check option;

grant all on fedex\_manager\_on\_shipper to fedex\_manager;

drop user if exists freddy;

create user freddy identified by 'fedex';

grant fedex\_manager to freddy;

set default role fedex\_manager to freddy;

# select \* from fedex\_manager\_on\_shipper;

**7. Triggers**

# check item availability before inserting into order\_details

drop trigger if exists check\_inventory\_availabity;

delimiter //

Create trigger check\_inventory\_availabity

before insert on order\_details

for each row

begin

if (select item\_count from inventory where inventory\_id = new.inventory\_id) < new.qty then

signal sqlstate '43098' set message\_text = 'Not enough items';

end if;

end //

delimiter ;

# Decrease item count after inserting into order\_details

DROP TRIGGER IF EXISTS decrease\_qty\_inventory;

delimiter //

CREATE TRIGGER decrease\_qty\_inventory AFTER INSERT ON order\_details

FOR EACH ROW

BEGIN

UPDATE inventory SET item\_count = item\_count - new.qty WHERE

inventory\_id=(SELECT inventory\_id FROM order\_details WHERE inventory\_id = NEW.inventory\_id);

END //

delimiter ;

# select \* from order\_details;

# select \* from inventory where inventory\_id = 155;

# insert into order\_details value(1, 155, 1, 20);

# insert into order\_details value(2, 155, 677, 20);