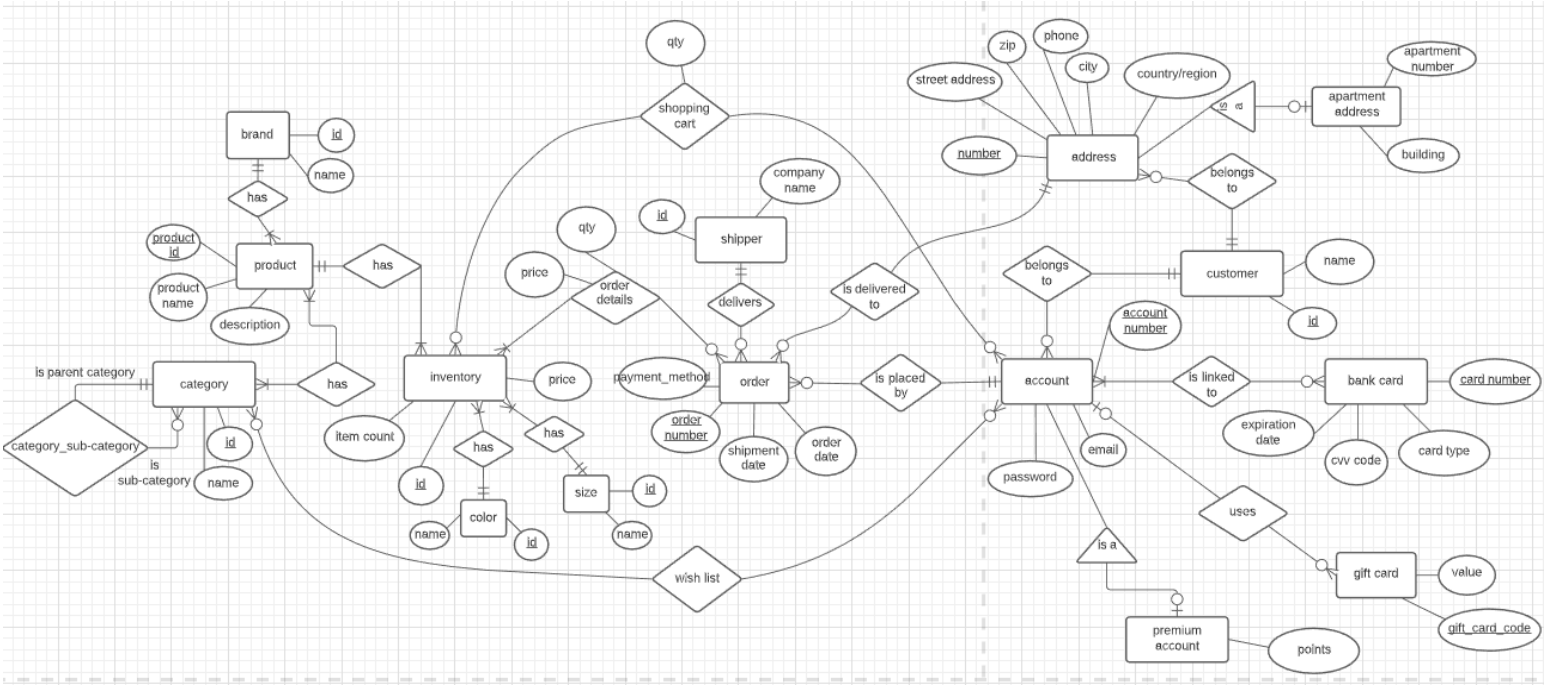
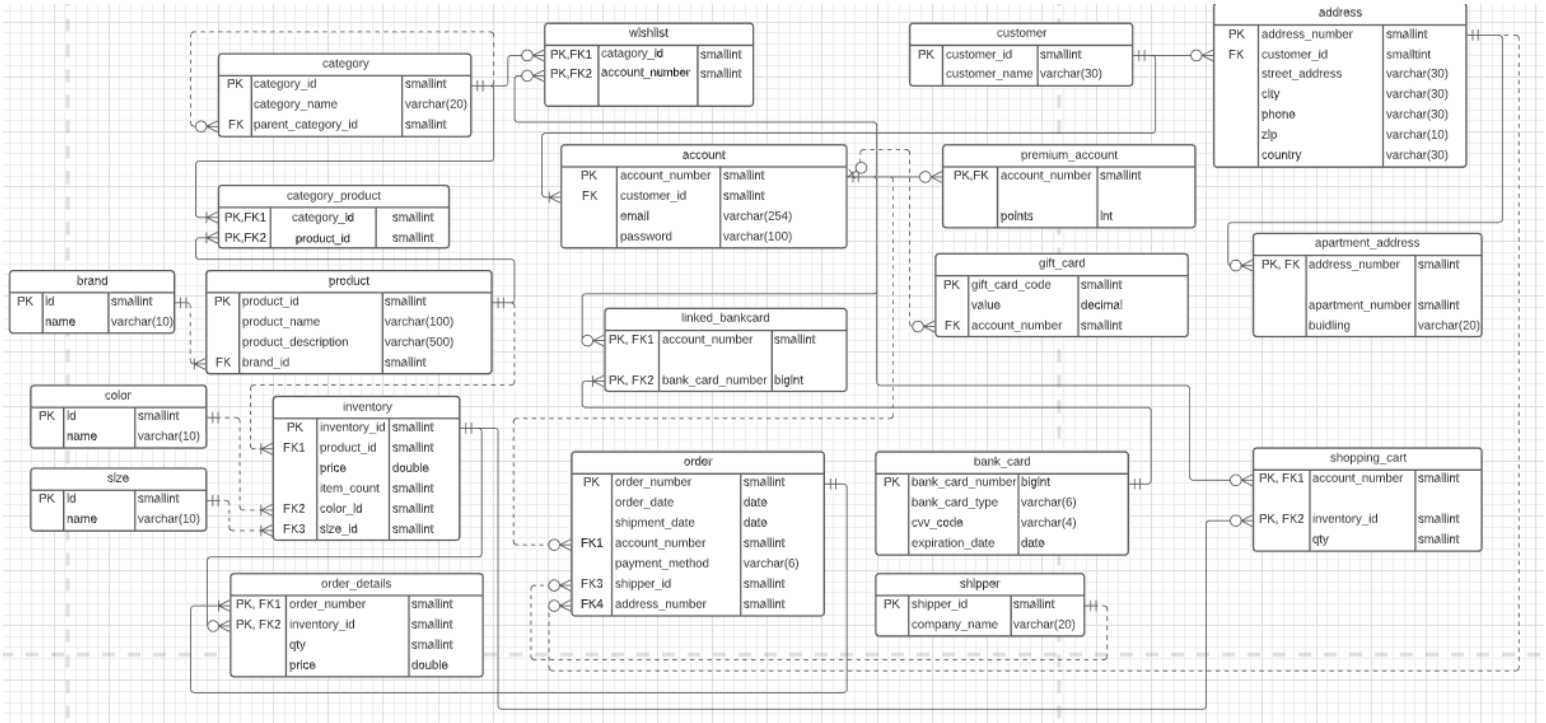
Database Project Report

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

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;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

account_number	order_number	product_name	brand_name	color_name	size_name	Price (before tax)	qty
3	5	Hanes Men Ecosmart Fleece Sweatshirt	Hanes	White	Medium	16.32	9

Shipping Company	shipment_date	Shipping Address	phone
DHL	2020-12-03	124 W Miller,New York,USA 14513	531322

```
# 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);
```