CSC-634 Database Project

by Sihyuan Han

2021 Summer

Database: Online Clothing Store

1. Define the information content of your database

a) Define a set of entities and appropriate attributes for each entity. Minimum 10 entities

- Customer: People who purchased from the store, including the customer's name, email, and other basic information, also stored whether he or she is a member of the store.
- Member: If a customer has registered to become a member, they will get a member id and a password. This table includes member's liked product list id and shopping cart id.
- Shopping_Cart: Each member has a shopping cart id, which stores the product he or she adds inside, and the last update (edit) time.
- Liked_Product: Each member has a liked product id, which stores the product he or she adds as "liked", and the last update time.
- Order: When a customer places an order it will generate an order id. This table includes basic order information of each order id, such as purchase date, price, transaction id, warehouse id, shipping address, order status and so on.
- Transaction: Contains customer id which indicates who is processing the payment, payment method, status, bank and billing address are also included in this table.
- Warehouse: Each order has a designated warehouse where products are going to be shipped out. Warehouse table includes the location of the warehouse and its contact information.
- Order_Detail: Each order's information, order id and product id are both primary keys. Including order id, products, quantity, unit price and discount voucher may be applied.
- Product: Online store products. Stored product's basic information, containing brand, category, price, url, stock, manufacturer id and other details of each product.
- Brand: Product's brand identified. Including each brand's name, logo, and types of the products it sells.
- Manufacturer: Identified by manufacturer id, including manufacturer name, logo and the country it is from.

b) Define a set of relationships that might exist between/among entities and attributes. Such relationships may include one-to-one, one-to-many and many-to-many associations

Customer to Member: one to one (or zero), because a customer may not be a member but a member must be a customer

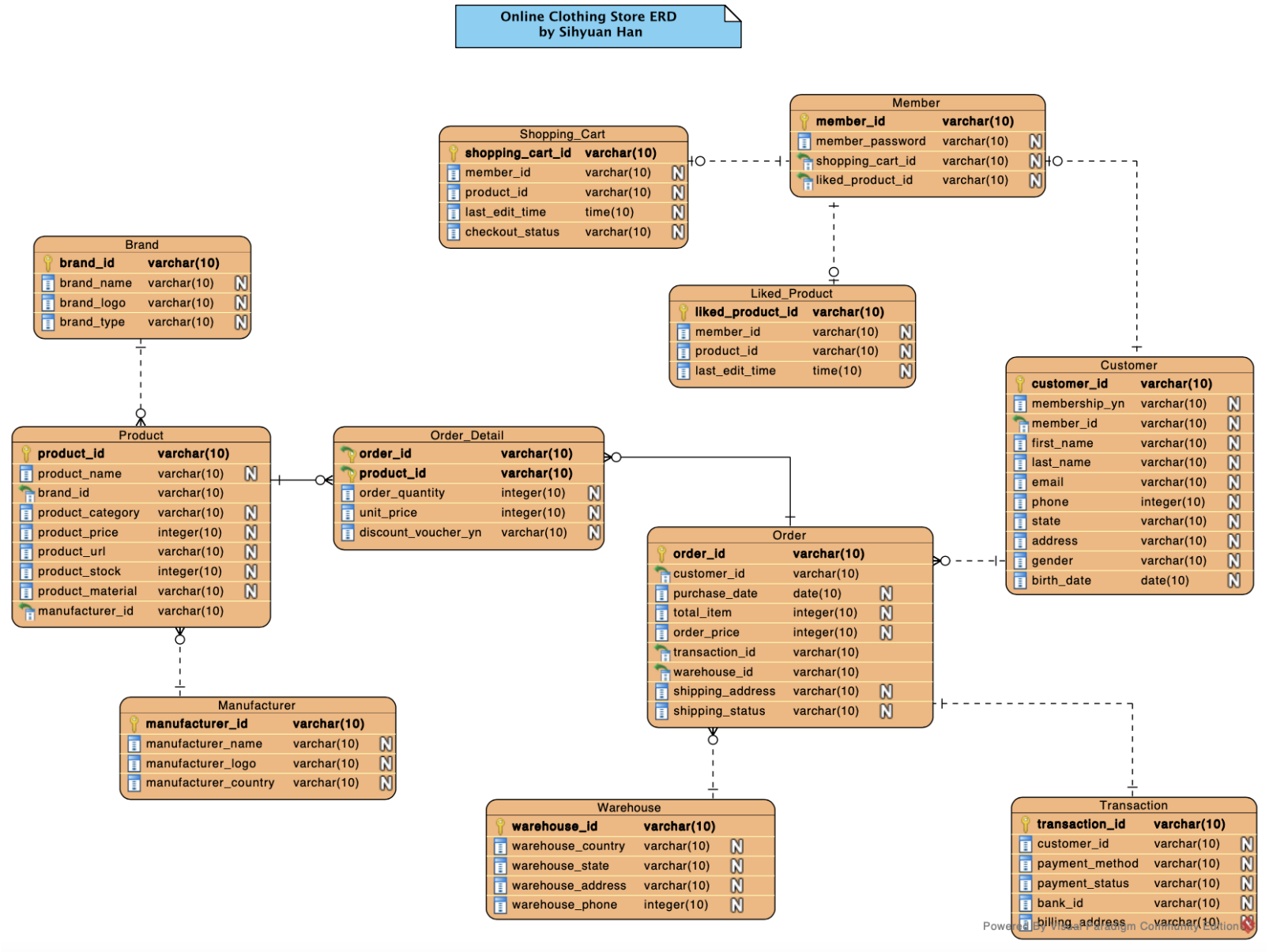
Member to Shopping_Cart (and Liked_Product): one to one (or zero), a member may put interested product to his or her shopping cart or favorite list

Customer to Order: one to many, one customer can purchase many orders, but a order must be purchased by only one customer

Order and Transaction: one to one, there is only going to be one payment per order, and an order is only going to be paid once
Warehouse to Order: one to many, an order can only be shipped out by one warehouse but a warehouse could ship out many orders
Order and Product: many to many, linked by Order_Deatail table which take both order_id and product_id as primary key
Brand (and Manufacturer) to Product: one to many, a brand produces many products while one product can only be manufactured by one brand

c) Define a set of constraints that may be imposed on data
For the main tables such as Customer, Order and Product, the cutomer_id, order_id and product_id will be unique key constraints which are not null, setting them as each table's primary key to avoid possible duplicated records for the whole relational database. In addition, define related columns as foreign keys to connect each table. For instance, to connect order and product table, which is a many to many relationship, it would be more specific to create another table order_detail and set order_id and product_id as composite primary keys to connect these tables.

2. Define an E-R Diagram for your database design



3. Define a relational schema for your database design
Make sure that you have both one-to-many and many-to-many associations
a) Define one or more realistic key(s) for every relation scheme. Use both simple and composite keys
- Customer: (customer_id(PK), member_id(FK), name, email, phone, address)
- Member: (member_id(PK), password, shopping_cart_id(FK), liked_product_id(FK))
- Shopping_Cart: (shopping_cart_id(PK), edit_time, checkout_status))
- Liked_Product: (liked_product_id(PK), edit_time)
- Order: (order_id(PK), purchase_date, shipping_address, customer_id(FK), transaction_id (FK), warehouse_id(FK))
- Transaction: (transaction_id(PK), payment_method, bank_id, billing_address)
- Warehouse: (warehouse_id(PK), country, address, phone)
- Order_Detail: (order_id, product_id(Composite PK), unit_price, quantity, price) composite primary keys
- Product: (product_id(PK), category, price, url, stock, brand_id (FK), manufacturer_id (FK), material)
- Brand: (brand_id(PK), name, logo)
- Manufacturer: (manufacturer_id(PK), name, logo, country)

The Order_Detail table uses composite primary keys, other tables use simple primary keys.

b) Define a realistic set of Functional / Multivalued Dependencies (when appropriate) for every relation scheme
For the Order table, order_id → purchase_date, shipping_address, customer_id, for Customer table, customer_id → name, email, phone, address, also for Product table, product_id → product's name, brand's id , price. If the brand's name goes into the Product table, then it would be 2NF since the brand's name is determined by brand's id. So for table Manufacturer, Brand and Warehouse, it only has their id as foreign key in the table and details in the separate table to achieve the relation scheme in the highest normal form.

c) Check whether your relational schema is in 2NF, 3NF, BCNF, 4NF
4NF, as shown in the ERD, all columns can be determined only by the primary key in the table and no multivalued dependencies.

d) Put your relational schema in the highest normal form that is possible
It is in the highest normal form.

4. Implementation: Create your database using MySQL, or... to Perform the following operations

(A)

- Create tables: (just for creating 4 tables, not all)

```
-- create db
create database if not exists clothdb;
use clothdb;

-- create table
use clothdb;

create table if not exists customer
(
customer_id char(5) not null primary key,
first_name varchar(10),
gender varchar(10),
state varchar(5)
);

create table if not exists orders
(
order_id char(5) not null primary key,
customer_id char(5),
purchase_date date,
shipping_status varchar(10)
);

create table if not exists order_detail
(
order_id char(5),
product_id char(5),
order_quantity double,
unit_price integer,
discount_voucher_yn varchar(5),
primary key (order_id, product_id)
);

create table if not exists product
(
product_id char(5) not null primary key,
```

```
product_name varchar(15),
product_category varchar(10),
price integer,
stock double,
material varchar(10)
);
```

- Select

(1) select involving one/more conditions in Where Clause

```
-- Display product's id and name with products that made by cotton and remaining stock over
100
SELECT
   product_id, product_name
FROM
   product
WHERE
   material = 'cotton' AND stock > 100;

# product_id, product_name
'P-502', 'cat_tshirt'
'P-506', 'kid_socks'
'P-508', 'letter_shorts'
```

(2) select with aggregate functions (i.e., SUM,MIN,MAX,AVG,COUNT)

```
-- Show each order's total price
SELECT
   order_id, SUM(unit_price) AS total_price
FROM
   order_detail
GROUP BY order_id;

# order_id, total_price
'O-001', '10'
'O-002', '31'
'O-003', '58'
'O-004', '30'
'O-005', '23'
'O-006', '25'
'O-007', '38'
'O-008', '9'
```

(3) select with Having, Group By, Order By clause

```
-- Find each product category's average price that higher than $10 and listed as an order high to low by price
SELECT
   product_category, AVG(price) AS avg_price
FROM
   product
GROUP BY product_category
HAVING avg_price > 10
ORDER BY avg_price DESC;

# product_category, avg_price
'bottom', '28.3333'
'dress', '24.0000'
'top', '14.0000'
```

(4) Nested Select

```
-- Find customer's first name with orders that haven't be shipped out yet
SELECT
   first_name
FROM
   customer
WHERE
   customer_id IN (SELECT
         customer_id
      FROM
         orders
      WHERE
         shipping_status = 'processing');

# first_name
'David'
'Kate'
'Grant'
'Jack'
'Leo'
```

(5) select involving the Union operation

```
-- Find all customer with their orders whether they have it or not
SELECT
```

```
    *
FROM
    customer c
        LEFT JOIN
    orders os ON c.customer_id = os.customer_id
UNION SELECT
    *
FROM
    customer c
        RIGHT JOIN
    orders os ON c.customer_id = os.customer_id;

# customer_id, first_name, gender, state, order_id, customer_id, purchase_date, shipping_status
'C-100', 'Abby', 'female', 'AZ', NULL, NULL, NULL, NULL
'C-101', 'Betty', 'female', 'NY', NULL, NULL, NULL, NULL
'C-102', 'Calvin', 'male', 'CA', NULL, NULL, NULL, NULL
'C-103', 'David', 'male', 'NJ', 'O-001', 'C-103', '2020-06-27', 'processing'
'C-103', 'David', 'male', 'NJ', 'O-005', 'C-103', '2020-01-28', 'shipped'
'C-104', 'Emma', 'female', 'IL', 'O-002', 'C-104', '2020-02-13', 'shipped'
'C-105', 'Fiona', 'female', 'TX', NULL, NULL, NULL, NULL
'C-106', 'Grant', 'male', 'UT', 'O-006', 'C-106', '2020-05-03', 'processing'
'C-107', 'Jack', 'male', 'AZ', 'O-003', 'C-107', '2020-03-04', 'shipped'
'C-107', 'Jack', 'male', 'AZ', 'O-007', 'C-107', '2020-11-01', 'processing'
'C-108', 'Kate', 'female', 'CA', 'O-004', 'C-108', '2020-07-15', 'processing'
'C-109', 'Leo', 'male', 'MN', 'O-008', 'C-109', '2020-12-21', 'processing'
```

- Insert

(1-1) insert one tuple into a table

insert into customer values('C-120', 'Mary', 'female', 'WA');

customer_id, first_name, gender, state
'C-100', 'Abby', 'female', 'AZ'
'C-101', 'Betty', 'female', 'NY'
'C-102', 'Calvin', 'male', 'CA'
'C-103', 'David', 'male', 'NJ'
'C-104', 'Emma', 'female', 'IL'
'C-105', 'Fiona', 'female', 'TX'
'C-106', 'Grant', 'male', 'UT'
'C-107', 'Jack', 'male', 'AZ'
'C-108', 'Kate', 'female', 'CA'
'C-109', 'Leo', 'male', 'MN'
'C-120', 'Mary', 'female', 'WA'

(1-2) for 2 tables, just add 3 records for each table

insert into product values('P-801', 'stripe_shirt', 'top', 17, 100, 'polyester'),
('P-802', 'ckeck_tshit', 'top', 13, 200, 'polyester'),
('P-803', 'cat_socks', 'socks', 9, 300, 'polyester');

product_id, product_name, product_category, price, stock, material
'P-500', 'floral_dress', 'dress', '20', '100', 'linen'
'P-501', 'tank_top', 'top', '10', '50', 'cotton'
'P-502', 'cat_tshirt', 'top', '17', '200', 'cotton'
'P-503', 'denim_shorts', 'bottom', '25', '100', 'denim'
'P-504', 'dog_pants', 'bottom', '30', '60', 'polyester'
'P-505', 'plain_dress', 'dress', '28', '70', 'linen'
'P-506', 'kid_socks', 'socks', '6', '200', 'cotton'
'P-507', 'pocket_tshirt', 'top', '15', '110', 'polyester'
'P-508', 'letter_shorts', 'bottom', '30', '200', 'cotton'
'P-509', 'adult_socks', 'socks', '9', '200', 'polyester'
'P-801', 'stripe_shirt', 'top', '17', '100', 'polyester'
'P-802', 'ckeck_tshit', 'top', '13', '200', 'polyester'
'P-803', 'cat_socks', 'socks', '9', '300', 'polyester'

(2) insert a set of tuples (by using another select statement)

-- insert female customers from customer into new table CustomerFemale
create table if not exists CustomerFemale

```
(
cf_id varchar(10) not null primary key,
cf_state varchar(5)
);

insert into CustomerFemale (cf_id, cf_state) select customer_id, state from customer where
gender = 'female';

# CustomerFemale
# cf_id, cf_state
'C-100', 'AZ'
'C-101', 'NY'
'C-104', 'IL'
'C-105', 'TX'
'C-108', 'CA'
```

(3) insert involving two tables

```
-- insert customer id purchase between Nov and Dec to CustomerFemale with FL state
insert into CustomerFemale (cf_id, cf_state) select customer_id, 'FL' from orders where
purchase_date >= '2020-11-01' and purchase_date <= '2020-12-31';

# CustomerFemale
# cf_id, cf_state
'C-100', 'AZ'
'C-101', 'NY'
'C-104', 'IL'
'C-105', 'TX'
'C-107', 'FL'
'C-108', 'CA'
'C-109', 'FL'
```

- Delete

(1) delete one tuple or a set of tuples: from one table

```
-- cancel orders that order quantity is equal to or more than 3
DELETE FROM order_detail
WHERE
   order_quantity >= 3;
```

before

```
# order_id, product_id, order_quantity, unit_price, discount_voucher_yn
'O-001', 'P-501', '1', '10', 'no'
'O-002', 'P-500', '3', '18', 'yes'
'O-002', 'P-506', '3', '5', 'yes'
'O-002', 'P-509', '3', '8', 'yes'
'O-003', 'P-504', '2', '30', 'no'
'O-003', 'P-505', '2', '28', 'no'
'O-004', 'P-508', '1', '30', 'no'
'O-005', 'P-502', '2', '17', 'no'
'O-005', 'P-506', '2', '6', 'no'
'O-006', 'P-503', '1', '25', 'no'
'O-007', 'P-501', '3', '9', 'yes'
'O-007', 'P-502', '3', '15', 'yes'
'O-007', 'P-507', '3', '14', 'yes'
'O-008', 'P-509', '1', '9', 'no'
```

after

```
# order_id, product_id, order_quantity, unit_price, discount_voucher_yn
'O-001', 'P-501', '1', '10', 'no'
'O-003', 'P-504', '2', '30', 'no'
'O-003', 'P-505', '2', '28', 'no'
'O-004', 'P-508', '1', '30', 'no'
'O-005', 'P-502', '2', '17', 'no'
'O-005', 'P-506', '2', '6', 'no'
'O-006', 'P-503', '1', '25', 'no'
'O-008', 'P-509', '1', '9', 'no'
```

(2) from multiple tables

```
-- delete orders data from MN or NJ
DELETE FROM orders
WHERE
   customer_id IN (SELECT
```

```
    customer_id
  FROM
    customer
  WHERE
    state = 'MN' or state = 'NJ');
```

before

```
# order_id, customer_id, purchase_date, shipping_status
'O-001', 'C-103', '2020-06-27', 'processing'
'O-002', 'C-104', '2020-02-13', 'shipped'
'O-003', 'C-107', '2020-03-04', 'shipped'
'O-004', 'C-108', '2020-07-15', 'processing'
'O-005', 'C-103', '2020-01-28', 'shipped'
'O-006', 'C-106', '2020-05-03', 'processing'
'O-007', 'C-107', '2020-11-01', 'processing'
'O-008', 'C-109', '2020-12-21', 'processing'
```

after

```
# order_id, customer_id, purchase_date, shipping_status
'O-002', 'C-104', '2020-02-13', 'shipped'
'O-003', 'C-107', '2020-03-04', 'shipped'
'O-004', 'C-108', '2020-07-15', 'processing'
'O-006', 'C-106', '2020-05-03', 'processing'
'O-007', 'C-107', '2020-11-01', 'processing'
```

- Update

(1) update one tuple or a set of tuples: from one table

```
-- update price if stock >= 100 take 10% off, if not, increase 10%
UPDATE product
SET
   price = CASE
      WHEN stock >= 100 THEN price* 0.9
      WHEN stock <100 THEN price* 1.1
   END;
```

before

```
# product_id, product_name, product_category, price, stock, material
'P-500', 'floral_dress', 'dress', '20', '100', 'linen'
'P-501', 'tank_top', 'top', '10', '50', 'cotton'
'P-502', 'cat_tshirt', 'top', '19', '200', 'cotton'
'P-503', 'denim_shorts', 'bottom', '25', '100', 'denim'
'P-504', 'dog_pants', 'bottom', '30', '60', 'polyester'
'P-505', 'plain_dress', 'dress', '28', '70', 'linen'
'P-506', 'kid_socks', 'socks', '7', '200', 'cotton'
'P-507', 'pocket_tshirt', 'top', '17', '110', 'polyester'
'P-508', 'letter_shorts', 'bottom', '33', '200', 'cotton'
'P-509', 'adult_socks', 'socks', '10', '200', 'polyester'
```

after

```
# product_id, product_name, product_category, price, stock, material
'P-500', 'floral_dress', 'dress', '18', '100', 'linen'
'P-501', 'tank_top', 'top', '11', '50', 'cotton'
'P-502', 'cat_tshirt', 'top', '17', '200', 'cotton'
'P-503', 'denim_shorts', 'bottom', '23', '100', 'denim'
'P-504', 'dog_pants', 'bottom', '33', '60', 'polyester'
'P-505', 'plain_dress', 'dress', '31', '70', 'linen'
'P-506', 'kid_socks', 'socks', '6', '200', 'cotton'
'P-507', 'pocket_tshirt', 'top', '15', '110', 'polyester'
'P-508', 'letter_shorts', 'bottom', '30', '200', 'cotton'
'P-509', 'adult_socks', 'socks', '9', '200', 'polyester'
```

(2) from multiple tables

```
-- set shipping status as shipped where customer state is NJ or CA
UPDATE orders
SET
```

```
   shipping_status = 'shipped'
WHERE
   customer_id IN (SELECT
         customer_id
      FROM
         customer
      WHERE
         state = 'NJ' OR state = 'CA');
```

before

```
# first_name, state, order_id, customer_id, purchase_date, shipping_status
'David', 'NJ', 'O-001', 'C-103', '2020-06-27', 'processing'
'Emma', 'IL', 'O-002', 'C-104', '2020-02-13', 'shipped'
'Jack', 'AZ', 'O-003', 'C-107', '2020-03-04', 'shipped'
'Kate', 'CA', 'O-004', 'C-108', '2020-07-15', 'processing'
'David', 'NJ', 'O-005', 'C-103', '2020-01-28', 'shipped'
'Grant', 'UT', 'O-006', 'C-106', '2020-05-03', 'processing'
'Jack', 'AZ', 'O-007', 'C-107', '2020-11-01', 'processing'
'Leo', 'MN', 'O-008', 'C-109', '2020-12-21', 'processing'
```

after

```
# first_name, state, order_id, customer_id, purchase_date, shipping_status
'David', 'NJ', 'O-001', 'C-103', '2020-06-27', 'shipped'
'Emma', 'IL', 'O-002', 'C-104', '2020-02-13', 'shipped'
'Jack', 'AZ', 'O-003', 'C-107', '2020-03-04', 'shipped'
'Kate', 'CA', 'O-004', 'C-108', '2020-07-15', 'shipped'
'David', 'NJ', 'O-005', 'C-103', '2020-01-28', 'shipped'
'Grant', 'UT', 'O-006', 'C-106', '2020-05-03', 'processing'
'Jack', 'AZ', 'O-007', 'C-107', '2020-11-01', 'processing'
'Leo', 'MN', 'O-008', 'C-109', '2020-12-21', 'processing'
```

- Create View

(1) based on one relation

```
-- create view as products with stock more than 100
CREATE VIEW PStocks100 AS
   SELECT
      *
   FROM
      product
   WHERE
      stock > 100;
```

```
# product_id, product_name, product_category, price, stock, material
'P-502', 'cat_tshirt', 'top', '17', '200', 'cotton'
'P-506', 'kid_socks', 'socks', '6', '200', 'cotton'
'P-507', 'pocket_tshirt', 'top', '15', '110', 'polyester'
'P-508', 'letter_shorts', 'bottom', '30', '200', 'cotton'
'P-509', 'adult_socks', 'socks', '9', '200', 'polyester'
```

(2) more than one relation

```
-- create a view with customer's name, gender, purchase date, shipping status and order quantity
CREATE VIEW CustomersOrder AS
   SELECT DISTINCT
      c.first_name,
      os.purchase_date,
      os.shipping_status,
      od.order_quantity
   FROM
      customer c
         JOIN
      orders os ON c.customer_id = os.customer_id
         JOIN
      order_detail od ON os.order_id = od.order_id;
```

```
# first_name, purchase_date, shipping_status, order_quantity
'David', '2020-06-27', 'processing', '1'
'Emma', '2020-02-13', 'shipped', '3'
'Jack', '2020-03-04', 'shipped', '2'
'Kate', '2020-07-15', 'processing', '1'
'David', '2020-01-28', 'shipped', '2'
'Grant', '2020-05-03', 'processing', '1'
```

'Jack', '2020-11-01', 'processing', '3'
'Leo', '2020-12-21', 'processing', '1'

(3) operate on View (i.e., select, insert, delete, update)
insert
insert into PStocks100 (product_id, product_name, product_category, price, stock, material)
values ('P-520', 'dot_dress', 'dress', 25, 300, 'cotton');

product_id, product_name, product_category, price, stock, material
'P-502', 'cat_tshirt', 'top', '17', '200', 'cotton'
'P-506', 'kid_socks', 'socks', '6', '200', 'cotton'
'P-507', 'pocket_tshirt', 'top', '15', '110', 'polyester'
'P-508', 'letter_shorts', 'bottom', '30', '200', 'cotton'
'P-509', 'adult_socks', 'socks', '9', '200', 'polyester'
'P-520', 'dot_dress', 'dress', 25, 300, 'cotton'

delete
-- delete 'top' category
DELETE FROM PStocks100
WHERE
  product_category = 'top';

product_id, product_name, product_category, price, stock, material
'P-506', 'kid_socks', 'socks', '6', '200', 'cotton'
'P-508', 'letter_shorts', 'bottom', '30', '200', 'cotton'
'P-509', 'adult_socks', 'socks', '9', '200', 'polyester'

update
-- update price increase 10%
UPDATE PStocks100
SET
  price = price *1.1;

<u>before</u>
product_id, product_name, product_category, price, stock, material
'P-502', 'cat_tshirt', 'top', '17', '200', 'cotton'
'P-506', 'kid_socks', 'socks', '6', '200', 'cotton'
'P-507', 'pocket_tshirt', 'top', '15', '110', 'polyester'
'P-508', 'letter_shorts', 'bottom', '30', '200', 'cotton'
'P-509', 'adult_socks', 'socks', '9', '200', 'polyester'

after
product_id, product_name, product_category, price, stock, material
'P-502', 'cat_tshirt', 'top', '19', '200', 'cotton'
'P-506', 'kid_socks', 'socks', '7', '200', 'cotton'
'P-507', 'pocket_tshirt', 'top', '17', '110', 'polyester'
'P-508', 'letter_shorts', 'bottom', '33', '200', 'cotton'
'P-509', 'adult_socks', 'socks', '10', '200', 'polyester'

select

```
-- show polyester mateiral's avg price
SELECT
   ROUND(AVG(price)) AS avg_price
FROM
   PStocks100
WHERE
   material = 'polyester';
```

avg_price
'12'

(B)

Also, create at least 4 different practical/useful triggers (written in MySQL) for your database to perform the following tasks:

- Creating Database Log

```
-- create CustomerData from customer table
CREATE TABLE CustomerData AS SELECT * FROM
   customer;

-- create theLog table
CREATE TABLE theLog (
   message VARCHAR(100)
);

DELIMITER $$
CREATE TRIGGER Add_Customer AFTER INSERT ON CustomerData
FOR EACH ROW
BEGIN
INSERT INTO theLog VALUES(CONCAT(current_date(), ': customer has been added by
',current_user()));
END$$
DELIMITER ;

-- insert to activate trigger
insert into CustomerData values('C-500', 'Sean', 'male', 'FL');

# theLog
# message
'2021-06-16: customer has been added by root@localhost'
```

- Gathering Statistics

-- create summary table ProductData: calculating each category's avg price and stock

```
CREATE TABLE ProductData (
   category VARCHAR(10),
   avg_price DOUBLE,
   avg_stock INTEGER
);

INSERT INTO ProductData
SELECT
   product_category AS category,
   ROUND(AVG(price), 2) AS avg_price,
   AVG(stock) AS avg_stock
FROM
   product
GROUP BY product_category;

-- trigger: update ProductData whenever insert new data into product table
DELIMITER $$
CREATE TRIGGER ProductData AFTER INSERT
ON product
FOR EACH ROW
BEGIN
DELETE FROM ProductData;
INSERT INTO ProductData
(SELECT
   product_category,
   ROUND(AVG(price), 2),
   AVG(stock)
FROM
   product
GROUP BY product_category);
END$$
DELIMITER ;

-- insert to activate trigger
insert into product values ('P-601', 'string_pants', 'bottom', 40, 350, 'cotton');
```

<u>before</u>

```
# product
```

product_id, product_name, product_category, price, stock, material
'P-500', 'floral_dress', 'dress', '20', '100', 'linen'
'P-501', 'tank_top', 'top', '10', '50', 'cotton'
'P-502', 'cat_tshirt', 'top', '17', '200', 'cotton'
'P-503', 'denim_shorts', 'bottom', '25', '100', 'denim'
'P-504', 'dog_pants', 'bottom', '30', '60', 'polyester'
'P-505', 'plain_dress', 'dress', '28', '70', 'linen'
'P-506', 'kid_socks', 'socks', '6', '200', 'cotton'
'P-507', 'pocket_tshirt', 'top', '15', '110', 'polyester'
'P-508', 'letter_shorts', 'bottom', '30', '200', 'cotton'
'P-509', 'adult_socks', 'socks', '9', '200', 'polyester'

ProductData
category, avg_price, avg_stock
'bottom', '28.33', '120'
'dress', '24', '85'
'socks', '7.5', '200'
'top', '14', '120'

<u>after</u>
product
product_id, product_name, product_category, price, stock, material
'P-500', 'floral_dress', 'dress', '20', '100', 'linen'
'P-501', 'tank_top', 'top', '10', '50', 'cotton'
'P-502', 'cat_tshirt', 'top', '17', '200', 'cotton'
'P-503', 'denim_shorts', 'bottom', '25', '100', 'denim'
'P-504', 'dog_pants', 'bottom', '30', '60', 'polyester'
'P-505', 'plain_dress', 'dress', '28', '70', 'linen'
'P-506', 'kid_socks', 'socks', '6', '200', 'cotton'
'P-507', 'pocket_tshirt', 'top', '15', '110', 'polyester'
'P-508', 'letter_shorts', 'bottom', '30', '200', 'cotton'
'P-509', 'adult_socks', 'socks', '9', '200', 'polyester'
'P-601', 'string_pants', 'bottom', 40, 350, 'cotton'

ProductData
category, avg_price, avg_stock
'dress', '24', '85'
'top', '14', '120'
'bottom', '31.25', '178'
'socks', '7.5', '200'

- Enforcing Referential Integrity

```
-- Before adding a new order data, it has to make sure that the customer exists! Otherwise, a
message will show up in theLog table!
DELIMITER $$
CREATE TRIGGER AddOrders BEFORE INSERT ON orders
FOR EACH ROW
BEGIN
DECLARE temp INT; SET temp = 0;
SELECT COUNT(*) INTO temp FROM orders, customer WHERE
orders.customer_id = customer.customer_id
AND customer.customer_id = new.customer_id;
IF temp = 0 THEN
INSERT INTO theLog VALUES(CONCAT('customer id: ', new.customer_id, ' does not exist!'));
END IF;
END$$
DELIMITER ;

-- insert to activate trigger
-- exist data
insert into orders values('O-500', 'C-103', '2020-03-13', 'shipped');
```

<u>after</u>

```
# theLog
# message
('C-103' exists in the customer table, so there is no error message showing in theLog table!)

-- non-exist data
insert into orders values('O-600', 'C-112', '2020-09-09', 'shipped');
```

<u>after</u>

```
# theLog
# message
'customer id: C-112 does not exist!'
('C-112' does not exist in the customer table, so there is an error message showing in theLog
table!)
```

```
-- Before adding new data into the order detail table, order id has to be in the orders table system!
Else, a message will show up in theLog table!

-- 'orders' pk: order_id
CREATE TABLE Order_ID AS SELECT order_id FROM
    orders;

DELIMITER $$
CREATE TRIGGER Add_ODetail BEFORE INSERT ON order_detail
FOR EACH ROW
BEGIN
DECLARE temp INT; SET temp = 0;
SELECT COUNT(*) INTO temp FROM order_detail, Order_ID WHERE
order_detail.order_id = Order_ID.order_id
AND Order_ID.order_id = new.order_id;
IF temp = 0 THEN
INSERT INTO theLog VALUES(CONCAT('order id: ', new.order_id, ' does not exist in the
system!'));
END IF;
END$$
DELIMITER ;

-- insert to activate trigger
-- non-exist data
insert into order_detail values('O-100', 'P-503', 1, 25, 'no');

# theLog
# message
'order id: O-100 does not exist in the system!'
```

- Enforcing Attribute Domain Constraints

```
-- Make sure to add the product with the existing category!

-- create table Product_Type
CREATE TABLE Product_Type AS SELECT DISTINCT product_category FROM
    product;

DELIMITER $$
CREATE TRIGGER Add_Product BEFORE INSERT ON product
FOR EACH ROW
BEGIN
DECLARE temp INT; SET temp = 0;
SELECT COUNT(*) INTO temp FROM Product_Type WHERE
product_category = new.product_category;
IF temp = 0 THEN
INSERT INTO theLog VALUES(CONCAT('product category: ', new.product_category, ' does
not exist in the system!'));
END IF;
END$$
DELIMITER ;

-- insert to activate trigger
insert into product values('P-700', 'face mask', 'cosmetics', 50, 10, 'cotton');

-- for check
SELECT
    *
FROM
    product;

SELECT
    *
FROM
    theLog;

# theLog
# message
'product category: cosmetics does not exist in the system!'
```