VIET NAM NATIONAL UNIVERSITY, HO CHI MINH CITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

DATABASE SYSTEM (LAB)
(CO2014)

Management system for online sales application

Instructor: Trần Thị Quế Nguyệt

Full name	Student ID	Class	Major
Đổng Hoàng Sơn	2110507	L03	KH - KT Máy Tính
Lương Thế Tổng	2115040	L03	KH - KT Máy Tính
Đoàn Duy Tùng	2112609	L03	KH - KT Máy Tính
Bùi Hoàng Phi	2011799	L03	KH - KT Máy Tính
Nguyễn Trịnh Ngọc Huân	2211144	L03	KH - KT Máy Tính

HO CHI MINH CITY, 31/10/2023

Contents

List of figures

List of Tables

1 Group members and Percentage of work

FullName	Student ID	Problems	Percentage of work
Đổng Hoàng Sơn	2110507	Create table + trigger + App	100%
Nguyễn Trịnh Ngọc Huân	2211144	Insert data + UI + report + testing	100%
Đoàn Duy Tùng	2112609	Functions + App	100%
Lương Thế Tổng	2115040	Procedures + App	100%
Bùi Hoàng Phi	2011799	Insert data + UI + report + testing	100%

2 Create table and insert sample data

The implementation will be carried out using the SQL-Server language with the management interface being SQL Server Management Studio (SSMS).

2.1 Implement the table

Write the code to implement the designed tables, including primary key constraints, foreign key constraints, data constraints, and semantic constraints as specified in the assignment 1 (using check or trigger)
Create tables:

- User

```
create table [User] (
  user_id   varchar(9)    ,
  user_name varchar(15)   not null,
  nickname  varchar(15)   not null,
  linking_phone_number  char(10) not null,
  linking_email varchar(100)  not null,
  password  varchar(100)  not null,
  user_id_contact varchar(9)  not null,
  contact_id  varchar(9)  not null
  constraint PK_user
    primary key (user_id),
  constraint Domain_phone_user
    check(linking_phone_number not like '%[^0-9]%'
    and linking_phone_number like '0%' and len(linking_phone_number) = 10),
  constraint Domain_email_user
    check(linking_email like '%@%'),
  constraint UQ_user_name
    unique (user_name),
  constraint UQ_linking_phone_number
    unique (linking_phone_number),
  constraint UQ_linking_email
    unique (linking_email),
  constraint user_id_format
    check (user_id like 'UID%' and len(user_id) = 9),
)

```

- Contact Information

```
create table Contact_info (
  user_id   varchar(9),
  contact_id  varchar(9),
  full_name varchar(50) not null,
  phone_number char(10) not null,
  email   varchar(100) default null,
  number    int   not null,
  street    varchar(30) not null,
  ward    varchar(30) not null,
  district  varchar(30) not null,
  city    varchar(30) not null,
  country   varchar(30) not null,
  constraint PK_Contact_info
    primary key (user_id,contact_id),
  constraint CheckNumber
    check(number > 0),
  constraint Domain_phone_contact
    check(phone_number not like '%[^0-9]%'
    and phone_number like '0%' and len(phone_number) = 10),
  constraint Domain_email_contact
    check(email like '%@%'),
  constraint contact_id_format
    check(contact_id like 'CID%' and len(contact_id) = 9)
)

```

- Follow

```
create table Follow (
  follower_id varchar(9),
  following_id  varchar(9),
  constraint PK_Follow
    primary key(follower_id,following_id)
)

```

- Seller

```
create table Seller (
  user_id   varchar(9),
  shop_id   varchar(9)  not null,
  constraint PK_seller
    primary key (user_id)
)

```

- Buyer

```
create table Buyer (
  user_id   varchar(9),
  constraint PK_buyer
    primary key (user_id)
)

```

- Shopping Cart

```
create table Shopping_cart (
  user_id   varchar(9),
  number    int,
  no_product  int not null default 0,
  date_created  Date  not null default convert(date,getdate()),
  constraint PK_cart
    primary key (user_id, number),
  constraint CheckNumber_No
    check(number >0 and no_product >= 0)
)

```

- Places

```
create table Places (
  order_id  varchar(14),
  user_id   varchar(9) default null,
  user_id_cart  varchar(9) default null,
  number  int default null,
  constraint notNull
    check((user_id is not null or (user_id_cart is not null and number is not null))
    and (user_id is null or (user_id_cart is null and number is null))
    ),
  constraint PK_places
    primary key (order_id),
)

```

- Shop's phone number

```
create table Shop_phone_number (
  shop_id   varchar(9),
  phone_number  char(10),
  constraint PK_shop_phone_number
    primary key (shop_id, phone_number),
  constraint Domain_shop_phone
    check(phone_number not like '%[^0-9]%' and phone_number like '0%' and len(phone_
    number) = 10)
)

```

- Shop's address

```
create table Shop_address (
  shop_id   varchar(9),
  number    int,
  street    varchar(30),
  ward    varchar(30),
  district  varchar(30),
  city    varchar(30),
  country   varchar(30),
  constraint PK_shop_address
    primary key (shop_id,number,street,ward,district,city,country),
  constraint CheckNumber_shop
    check(number > 0)
)

```

- Add

```
create table [Add] (
  product_id  varchar(9),
  user_id   varchar(9) not null,
  number    int     not null,
  quantity  int   not null default 1,
  constraint PK_add
    primary key (product_id),
  constraint Quantity_domain
    check(quantity > 0)
)

```

- Shop

```
create table Shop (
  shop_id   varchar(9),
  bio     varchar(1000) not null default 'No description',
  no_following  int   not null default 0,
  no_follower   int   not null default 0,
  url_link  varchar(100)  not null check(url_link like 'https://%.%'),
  rating    decimal(2,1)  not null default 0,
  [name]    varchar(30) not null,
  date_joined date  not null default convert(date,getdate()),
  no_product  int not null default 0 , --trigger here
  constraint rating_domain
    check(rating between 0 and 5),
  constraint PK_shop
    primary key (shop_id),
  constraint DomainShop
    check (no_following >= 0 and no_follower >= 0 and (rating between 0 and 5)),
  constraint shop_id_format
    check(shop_id like 'SID%' and len(shop_id) = 9),
  constraint no_product_domain
    check(no_product >= 0),
  constraint UQ_url
    unique(url_link),
  constraint UQ_name
    unique([name]),
)

```

- Review

```
create table Review (
  review_id varchar(9),
  no_stars  int   not null default 0,
  content   varchar(1000) not null default 'No description',
  time_created  datetime  not null default getdate(),
  product_id  varchar(9)  not null,
  user_id   varchar(9)  not null,
  constraint stars_domain
    check(no_stars between 0 and 5),
```

```
  constraint PK_review
    primary key (review_id),
  constraint review_id_format
    check (review_id like 'RID%' and len(review_id) = 9)
)

```

- Product instance

```
create table Product_instance (
  instance_id varchar(9),
  current_price decimal(10,1) not null,
  product_id  varchar(9)  not null
  constraint PK_product_instance
    primary key (instance_id),
  constraint instance_id_domain
    check(instance_id like 'IID%' and len(instance_id) = 9),
  constraint cur_price_domain
    check (current_price >= 0)
)

```

- Belong to

```
create table Belong_to (
  instance_id varchar(9),
  product_id  varchar(9)  not null,
  variant_name  varchar(120)  not null,
  constraint PK_belongto
    primary key (instance_id)
)


```

- Variant

```
create table Variant (
  product_id  varchar(9),
  variant_name  varchar(120),
  price decimal(10,1) not null, -- trigger here
  remaining_amount  int not null default 0 check(remaining_amount >= 0),
  details   varchar(1000) not null default 'No description',
  img   varchar(1000),
  constraint PK_Variant
    primary key (product_id, variant_name),
  constraint price_ver_domain
    check(price >= 0)
)

```

- Product

```
create table Product (
  product_id  varchar(9),
  description   varchar(1000) not null default 'No description',
  name  varchar(120) not null,
  total_remaining int not null default 0 check(total_remaining >= 0),
  no_sales  int not null default 0 check (no_sales >= 0),
  minimum_price decimal(10,1) not null, --trigger here
  maximum_price decimal(10,1) not null,
  category_id   varchar(9),
  shop_id   varchar(9)  not null,
  img     varchar(1000),
  constraint PK_product
    primary key (product_id),
  constraint CheckMinMax
    check(minimum_price <= maximum_price),
  constraint product_id_format
    check(product_id like 'PID%' and len(product_id) = 9),
  constraint min_domain
```

```
    check(minimum_price >= 0),
)
```

- Category

```
create table Category(
  category_id varchar(9),
  category_name varchar(50),
  [desciption] varchar(1000) not null default 'No description',
  constraint PK_category
    primary key(category_id),
  constraint cat_format
    check(category_id like 'CAT%' and len(category_id) = 9)
)
```

- Applies

```
create table Applies (
  order_id  varchar(14),
  voucher_id  varchar(9),
  constraint PK_applies
    primary key (order_id, voucher_id),
)
```

- Can apply

```
create table Can_apply (
  category_id varchar(9) default 'CATffffff', -- applies all category
  voucher_id  varchar(9),
  shop_id   varchar(9)  default 'SIDffffff', -- applies all shop
  constraint PK_can_apply
    primary key(category_id, voucher_id, shop_id)
)
```

- Order

```
create table [Order] (
  order_id  varchar(14),
  status    varchar(15) not null,
  no_product  int not null check(no_product >= 1),
  time_order  datetime  not null default getdate(),
  ID_payment  varchar(9)  not null,
  delivery_id varchar(9)  not null,
  constraint order_id_format
    check(order_id like 'ORD%' and len(order_id) = 14),
  constraint PK_order
    primary key (order_id),
  constraint Status_domain
    check (
      status = 'Confirming' or
      status = 'Waiting pickup' or
      status = 'Delivering' or
      status = 'Done'or
      status = 'Cancelled' or
      status = 'Refund'
    )
)
```

- Voucher

```
create table Voucher (
  voucher_id  varchar(9),
  quantity  int   not null default 0 check(quantity >= 0),
  discount  decimal(10,1) check(discount >= 0),
  lowest_applied_price  decimal(10,1) not null default 0 check(lowest_applied_price >=0),
```

```
  start_time  datetime  not null default getdate(),
  end_time  datetime,
  [type]    varchar(15) not null,
  maximum_cash_discount decimal(10,1),
  [percent]   decimal(2,2),
  constraint vch_format
    check(voucher_id like 'VCH%' and len(voucher_id) = 9),
  constraint PK_voucher
    primary key (voucher_id),
  constraint Check_dis_per
    check(
      discount is not null or
      (maximum_cash_discount is not null and [percent] is not null)),
  constraint Type_domain
    check(
      [type] = 'For delivery' or
      [type] = 'For products'
    ),
  constraint start_end_time
    check(end_time > start_time),
)

```

- Delivery service

```
create table Delivery_service(
  delivery_id   varchar(9),
  name    varchar(50) not null,
  price decimal(10,1) not null check (price >= 0),
  estimated_time  int not null check(estimated_time >= 0),
  constraint PK_delivery
    primary key (delivery_id),
  constraint del_format
    check(delivery_id like 'DEL%' and len(delivery_id) = 9)
)

```

- Is contained

```
create table Is_contained(
  instance_id   varchar(9),
  order_id    varchar(14) not null,
  constraint PK_contained
    primary key (instance_id)
)

```

- Payment

```
create table Payment(
  ID_payment  varchar(9),
  owner_name  varchar(50) not null,
  user_id   varchar(9) not null,
  constraint PK_payment
    primary key (ID_payment),
  constraint pay_format
    check(ID_payment like 'PAY%' and len(ID_payment) = 9)
)

```

- E-wallet

```
create table E_wallet (
  ID_payment  varchar(9)  primary key,
  wallet_number varchar(16) not null,
  constraint wallet_format
    check(wallet_number not like '%[^0-9]%')

)

```

- Card payment

```
create table Card_payment (
  ID_payment  varchar(9)  primary key,
  card_number char(16) not null,
  [type]    varchar(15) not null check([type] = 'Debit' or [type] = 'Credit' or [type] =
    'Napas'),
  constraint card_format
    check(card_number not like '%[^0-9]%' and len(card_number) = 16)
)

```

- Internet banking

```
create table Internet_banking (
  ID_payment  varchar(9)  primary key,
  account_number  varchar(16) not null,
  constraint account_format
    check(account_number not like '%[^0-9]%')
)

```

- COD

```
create table COD(
  ID_payment  varchar(9)  primary key,
)

```

Add constraint:

- Add foreign key constraints

```
alter table [User] add constraint FK_UID_contact foreign key (user_id_contact, contact_id
    ) references Contact_info(user_id,contact_id)

alter table Contact_info add constraint FK_user_owner foreign key (user_id) references [
    User](user_id)

alter table Follow add constraint FK_followed_id foreign key (follower_id) references [
    User](user_id)
alter table Follow add constraint FK_following_id foreign key (following_id) references [
    User](user_id)

alter table Seller add constraint FK_seller_uid foreign key (user_id) references [User](
    user_id)
alter table Seller add constraint FK_seller_sid foreign key (shop_id) references Shop(
    shop_id)

alter table Buyer add constraint FK_buyer_uid foreign key (user_id) references [User](
    user_id)

alter table Shopping_cart add constraint FK_cart_uid foreign key (user_id) references
    Buyer(user_id)

alter table Places add constraint FK_places_uid foreign key (user_id) references Buyer(
    user_id)
alter table Places add constraint FK_places_cart foreign key (user_id_cart,number)
    references Shopping_cart(user_id,number)
alter table Places add constraint FK_places_ord foreign key (order_id) references [Order
    ](order_id)

alter table [Add] add constraint  FK_add_pid foreign key (product_id) references Product(
    product_id) on update cascade
alter table [Add] add constraint FK_add_uid foreign key (user_id,number) references
    Shopping_cart(user_id,number)

alter table Shop_phone_number add constraint FK_phone_sid foreign key (shop_id)
    references Shop(shop_id)

```

```
alter table Shop_address add constraint FK_addr_sid foreign key(shop_id) references Shop(shop_id)

alter table Review add constraint FK_review_pni foreign key (product_id) references Product(product_id) on update cascade
alter table Review add constraint FK_review_uid foreign key (user_id) references Buyer(user_id)

alter table Product_instance add constraint FK_instance_pid foreign key (product_id) references Product(product_id) on update cascade

alter table belong_to add constraint FK_belong_iid foreign key (instance_id) references Product_instance(instance_id)
alter table belong_to add constraint FK_belong_pid foreign key (product_id,variant_name) references Variant(product_id,variant_name) on update cascade

alter table Variant add constraint FK_Variant_pid foreign key (product_id) references Product(product_id) on update cascade

alter table Product add constraint FK_pn_cat foreign key (category_id) references Category(category_id)
alter table Product add constraint FK_pn_sid foreign key (shop_id) references Shop(shop_id)

alter table applies add constraint FK_applies_ord foreign key (order_id) references [Order](order_id)
alter table applies add constraint FK_applies_vch foreign key (voucher_id) references Voucher(voucher_id)

```

```
alter table Can_apply add constraint FK_canapply_cat foreign key (category_id) references Category(category_id)
alter table Can_apply add constraint FK_canapply_vch foreign key (voucher_id) references Voucher(voucher_id)
alter table Can_apply add constraint FK_canapply_sid foreign key (shop_id) references Shop(shop_id)

alter table [Order] add constraint FK_ord_pay foreign key (ID_payment) references Payment(ID_payment)
alter table [Order] add constraint FK_ord_del foreign key (delivery_id) references Delivery_service(delivery_id)

alter table Is_contained add constraint FK_contained_iid foreign key (instance_id) references Product_instance(instance_id)
alter table Is_contained add constraint FK_contained_ord foreign key (order_id) references [Order](order_id)

alter table Payment add constraint FK_payment_uid foreign key (user_id) references [User](user_id)
alter table E_wallet add constraint FK_wallet_pay foreign key (ID_payment) references Payment(ID_payment)
alter table Internet_banking add constraint FK_bank_pay foreign key (ID_payment) references Payment(ID_payment)
alter table Card_payment add constraint FK_card_pay foreign key (ID_payment) references Payment(ID_payment)
alter table COD add constraint FK_COD_pay foreign key (ID_payment) references Payment(ID_payment)

```

2.2 Insert sample data

Insert meaningful sample data for all tables.

Here are screenshots of the data tables created by our team.

product_id	user_id	number	quantity
PID000001	UID000001	1	1
PID000002	UID100052	1	2
PID000007	UID100013	1	1
PID000008	UID100046	1	1
PID000009	UID100049	1	1
PID000010	UID100052	1	1
PID000011	UID000002	2	1
PID000012	UID000003	3	1
PID000013	UID000004	4	1
PID000014	UID000005	5	1

Figure 1: Add to shopping cart

order_id	voucher_id
ORD00000000001	VCH000006
ORD00000000002	VCH000007
ORD00000000003	VCH000008
ORD00000000004	VCH000009
ORD00000000005	VCH000010
ORD00000000031	VCH000010
ORD00000000032	VCH000010

Figure 2: Apply the Voucher to the Order

instance_id	product_id	variant_name
IID000001	PID000001	Standard
IID000002	PID000001	Premium
IID000003	PID000002	Classic Fit
IID000004	PID000002	Slim Fit
IID000005	PID000003	32GB
IID000006	PID000003	64GB
IID000007	PID000004	Copper
IID000008	PID000004	Silver
IID000009	PID000005	Basic Kit
IID000010	PID000005	Deluxe Kit
IID000011	PID000006	Statement Piece
IID000012	PID000006	Classic
IID000013	PID000007	Bifold
IID000014	PID000007	Trifold
IID000015	PID000008	Basic Model
IID000016	PID000008	Pro Model
IID000017	PID000009	Modern Design
IID000018	PID000009	Vintage Style
IID000019	PID000010	Classic Blue
IID000020	PID000010	Stainless Steel
IID000021	PID000011	Athletic Style
IID000022	PID000011	Casual Comfort
IID000023	PID000012	Relaxed Fit
IID000024	PID000012	Skinny Fit
IID000025	PID000013	Pro Camera
IID000026	PID000013	Compact Camera
IID000027	PID000014	Cozy Plaid
IID000028	PID000014	Luxury Velvet

Figure 3: Variant and Product instance belongs to which Product

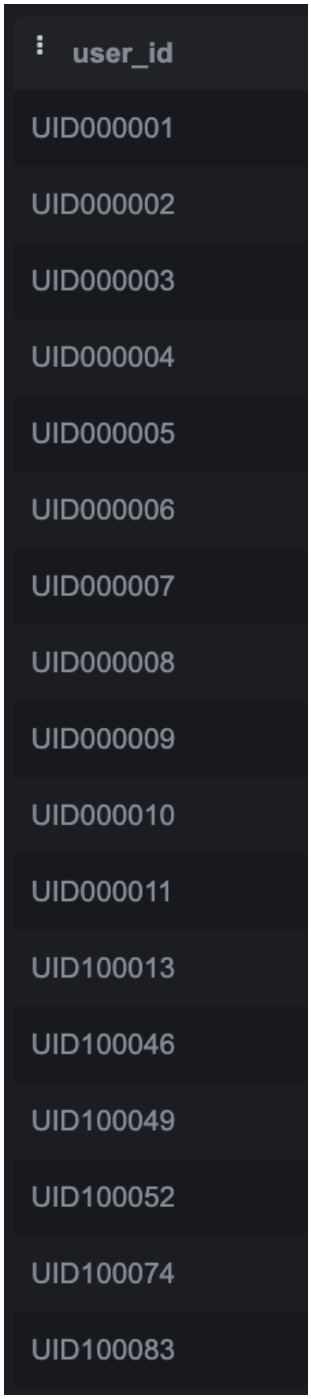

Figure 4: Buyer

category_id	voucher_id	shop_id
CAT000001	VCH000006	SID000001
CAT000004	VCH000007	SIDffffff
CAT000005	VCH000008	SID000005
CAT000006	VCH000009	SID000001
CATffffff	VCH000010	SID000004
CATffffff	VCH000010	SID100052

Figure 5: Category and Shop can apply the Voucher

ID_payment	card_number	type
PAY001113	0987654321123456	Debit
PAY002011	0987654321123477	Napas

Figure 6: Card Payment

category_id	category_name	desciption
CAT000001	Footwear	No description
CAT000002	Apparel	No description
CAT000003	Electronics	No description
CAT000004	Home Decor	No description
CAT000005	Outdoor Gear	No description
CAT000006	Jewelry	No description
CAT000010	Household	No description
CAT000011	Market	No description
CAT000012	Beauty	No description
CAT000013	Women Clothes	No description
CAT000014	Men Clothes	No description
CATffffff	ALL CATEGORIES	No description

Figure 7: Category

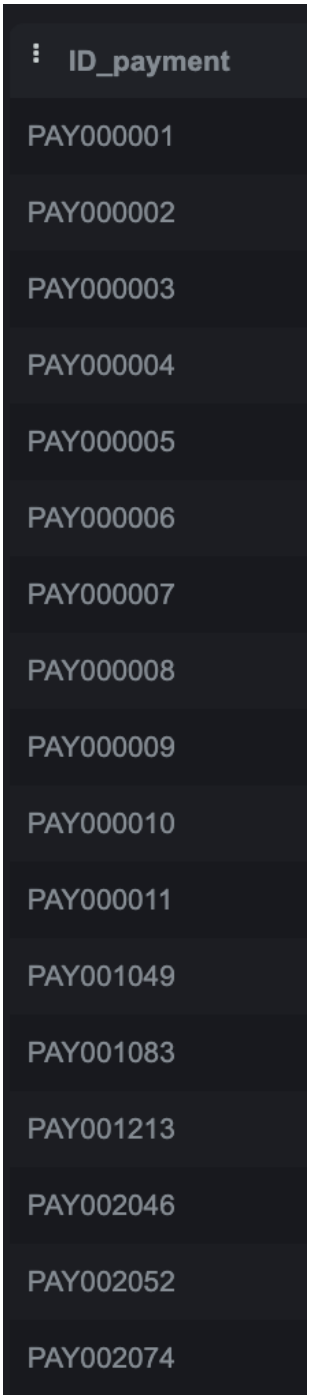
ID_payment
PAY000001
PAY000002
PAY000003
PAY000004
PAY000005
PAY000006
PAY000007
PAY000008
PAY000009
PAY000010
PAY000011
PAY001049
PAY001083
PAY001213
PAY002046
PAY002052
PAY002074

Figure 8: COD

delivery_id	name	price	estimated_time
DEL000001	Standard Deli	5000.0	3
DEL000002	Express Deli	10000.0	2
DEL000003	Next-Day Deli	15000.0	1
DEL000004	Free Pickup	0.0	0
DEL000005	Two-Day Ship	8000.0	2
DEL000011	GHN	27000.0	2
DEL000012	GHTK	15000.0	4

Figure 10: Delivery Service

ID_payment	wallet_number
PAY001013	01231231232323
PAY001052	01231231232324
PAY002083	01231231232325

Figure 11: E-wallet

user_id	contact...	full_name	phone_...	email	number	street	ward	district	city	country
UID000001	CID000001	Nguyen ...	03563...	huan112...	20	Ta Quang Buu	Dong Hoa	Di An	Binh Duong	Vietnam
UID000002	CID000002	John Doe	02345...	john.doe...	25	Main St	West	Central	Cityville	USA
UID000003	CID000003	Alice Smith	08765...	alice.smi...	30	Oak Ave	East	Suburbia	Townsville	USA
UID000004	CID000004	Bob Joh...	05123...	bob.j@g...	15	Elm St	North	Downtown	Metropolis	USA
UID000005	CID000005	Emily Jo...	03678...	emily.jon...	40	Maple St	South	Village	Ruraltown	USA
UID000006	CID000006	David Mil...	08901...	david.mil...	10	Cedar St	West	Central	Cityville	USA
UID000007	CID000007	Sophia B...	03456...	sophia.br...	35	Pine St	East	Suburbia	Townsville	USA
UID000008	CID000008	Daniel W...	07654...	daniel.w...	5	Willow St	North	Downtown	Metropolis	USA
UID000009	CID000009	Olivia Da...	02109...	olivia.da...	45	Birch St	South	Village	Ruraltown	USA
UID000010	CID000010	William ...	00876...	william.w...	15	Spruce St	West	Central	Cityville	USA
UID000011	CID000011	Ava Taylor	03579...	ava.taylo...	50	Cherry St	East	Suburbia	Townsville	USA
UID100013	CID100013	Dong Ho...	01675...	son_017...	1	Ly Thuong Kiet	phuong 15	quan 10	thanh pho H...	Viet Nam
UID100013	CID100113	Dong Ho...	02733...	son_017...	1	ki tuc xa khu A, khu pho 6	Linh Trung	Thu Duc	thanh pho H...	Viet Nam
UID100013	CID100213	Dong Ho...	03750...	son_017...	12	Dong Nai	phuong 10	quan 10	thanh pho H...	Viet Nam
UID100046	CID100046	Dinh Tong	08543...	tong_01...	1	Ly Thuong Kiet	phuong 15	quan 10	thanh pho H...	Viet Nam
UID100049	CID100049	Minh Ho...	07617...	hoangaa...	1	Ly Thuong Kiet	phuong 15	quan 10	thanh pho H...	Viet Nam
UID100052	CID100052	Duy Tung	00759...	tungbbb...	248	Ly Thuong Kiet	phuong 15	quan 10	thanh pho H...	Viet Nam
UID100052	CID100152	Duy Tung	00759...	tungbbb...	1	Ly Thuong Kiet	phuong 15	quan 10	thanh pho H...	Viet Nam
UID100074	CID100074	Ngoc Huan	03451...	huanccc...	13	Tan Lap, khu pho 6	Linh Trung	Thu Duc	thanh pho H...	Viet Nam
UID100083	CID100083	Dinh Ho...	04293...	sonddd...	1	Ly Thuong Kiet	phuong 15	quan 10	thanh pho H...	Viet Nam

Figure 9: Contact information

follower_id	following_id
UID000002	UID000003
UID000002	UID000004
UID000003	UID000002
UID000003	UID000005
UID000004	UID000002
UID000004	UID000006
UID000005	UID000002
UID000005	UID000007
UID000006	UID000002
UID000006	UID000008
UID000007	UID000002
UID000007	UID000009
UID000008	UID000002
UID000008	UID000010
UID000009	UID000002
UID000009	UID000011
UID000010	UID000002
UID000011	UID000002
UID000011	UID000004
UID100013	UID100049
UID100049	UID100013
UID100049	UID100052
UID100049	UID100074
UID100049	UID100083

Figure 12: Followers of users

ID_payment	account_number
PAY001002	0924483533
PAY001003	28243483533
PAY001004	06244832524
PAY001005	9524483545
PAY001006	89224583533
PAY001046	03551234123321
PAY001074	0824483533

Figure 13: Internet banking

instance_id	order_id
IID000001	ORD00000000001
IID000002	ORD00000000001
IID000003	ORD00000000002
IID000004	ORD00000000002
IID000005	ORD00000000003
IID000006	ORD00000000001
IID000021	ORD00000000005
IID000026	ORD00000000004
IID000027	ORD00000000004
IID000028	ORD00000000004
IID000029	ORD00000000004
IID000030	ORD00000000005
IID080466	ORD00000000031
IID080467	ORD00000000032
IID080468	ORD00000000033

Figure 14: Order is contained of what Product instance

order_id	status	no_product	time_order	ID_payment	delivery_id
ORD00000000001	Confirming	1	2023-12-12 18:19:15	PAY000001	DEL000001
ORD00000000002	Waiting pickup	1	2023-12-12 18:19:15	PAY000002	DEL000002
ORD00000000003	Delivering	1	2023-12-12 18:19:15	PAY000003	DEL000003
ORD00000000004	Done	1	2023-12-12 18:19:15	PAY000004	DEL000004
ORD00000000005	Cancelled	1	2023-12-12 18:19:15	PAY000005	DEL000005
ORD00000000031	Done	1	2023-11-14 00:00:00	PAY001013	DEL000011
ORD00000000032	Done	1	2023-11-14 00:00:00	PAY001074	DEL000011
ORD00000000033	Confirming	1	2023-11-14 00:00:00	PAY001074	DEL000011

Figure 15: Order

ID_payment	owner_name	user_id
PAY000001	Nguyen Huan	UID000001
PAY000002	John Doe	UID000002
PAY000003	Alice Smith	UID000003
PAY000004	Alice Smith	UID000004
PAY000005	Emily Jones	UID000005
PAY000006	David Miller	UID000006
PAY000007	Sophia Brown	UID000007
PAY000008	Daniel White	UID000008
PAY000009	Olivia Davis	UID000009
PAY000010	William Wilson	UID000010
PAY000011	Ava Taylor	UID000011
PAY001002	John Doe	UID000002
PAY001003	Alice Smith	UID000003
PAY001004	Alice Smith	UID000004
PAY001005	Emily Jones	UID000005
PAY001006	David Miller	UID000006
PAY001013	Dong Hoang Son	UID100013
PAY001046	Dinh Tong	UID100046
PAY001049	Minh Hoang	UID100049
PAY001052	Duy Tung	UID100052
PAY001074	Ngoc Huan	UID100074
PAY001083	Dinh Hong Son	UID100083
PAY001113	Dong Hoang Son	UID100013
PAY001213	Dong Hoang Son	UID100013
PAY002011	Ava Taylor	UID000011
PAY002046	Dinh Tong	UID100046
PAY002052	Duy Tung	UID100052
PAY002074	Ngoc Huan	UID100074
PAY002083	Dinh Hong Son	UID100083

Figure 16: Payment

order_id	user_id	user_id_cart	number
ORD00000000001	UID000001	NULL	NULL
ORD00000000002	NULL	UID000002	2
ORD00000000003	UID000003	NULL	NULL
ORD00000000004	NULL	UID000004	4
ORD00000000005	UID000005	NULL	NULL
ORD00000000031	UID100013	NULL	NULL
ORD00000000032	UID100074	NULL	NULL
ORD00000000033	UID100074	NULL	NULL

Figure 17: Place Order

product_id	description	name	total_remaining	no_sales	minimum_price	maximum_price	category_id	shop_id	img
PID000001	Stylish and com…	Comfort Shoes	100	50	29001.0	69001.0	CAT000001	SID000001	NULL
PID000002	A classic additio…	Casual Shirt	80	30	19001.0	49001.0	CAT000002	SID000002	NULL
PID000003	High-performan…	Smartphone	150	100	89001.0	199001.0	CAT000003	SID000003	NULL
PID000004	Perfect for cozy…	Candle Holder	50	20	9001.0	24001.0	CAT000004	SID000004	NULL
PID000005	Outdoor advent…	Camping Gear	120	70	49001.0	99001.0	CAT000005	SID000005	NULL
PID000006	Unique and styli…	Necklace	60	40	34001.0	74001.0	CAT000006	SID000001	NULL
PID000007	A modern twist …	Leather Wallet	90	60	24001.0	59001.0	CAT000002	SID000002	NULL
PID000008	Efficient and reli…	Laptop	110	80	19001.0	1449001.0	CAT000003	SID000003	NULL
PID000009	Elegant and fun…	Table Lamp	40	15	44001.0	89001.0	CAT000004	SID000004	NULL
PID000010	Stay hydrated o…	Water Bottle	70	25	9001.0	29001.0	CAT000005	SID000005	NULL
PID000011	Perfect for ever…	Sneakers	90	60	39001.0	79001.0	CAT000001	SID000001	NULL
PID000012	A versatile addit…	Denim Jeans	120	80	29001.0	69001.0	CAT000002	SID000002	NULL
PID000013	Capture every …	Digital Camera	30	15	149001.0	499001.0	CAT000003	SID000003	NULL
PID000014	Create a cozy a…	Throw Blanket	65	35	14001.0	34001.0	CAT000004	SID000004	NULL
PID000015	Gear up for you…	Hiking Boots	100	50	59001.0	119001.0	CAT000005	SID000005	NULL
PID110464	No description	Nuoc xa Comfo	0	0	52000.0	69000.0	CAT000010	SID100052	NULL
PID138894	No description	Bot giat Omo	0	0	14000.0	14000.0	CAT000010	SID100052	NULL
PID144103	No description	Nuoc xa Domy	0	0	47000.0	47000.0	CAT000010	SID100052	NULL
PID163853	No description	Bo beo	0	0	5000.0	20000.0	CAT000011	SID100074	NULL
PID186682	No description	Banh trang	0	0	24000.0	112000.0	CAT000011	SID100074	NULL
PIDffffff	This product na…	Unknown	0	0	0.0	0.0	NULL	SIDffffff	NULL

Figure 18: Product

IID000001	49001.0	PID000001
IID000002	29001.0	PID000002
IID000003	99001.0	PID000003
IID000004	14001.0	PID000004
IID000005	79001.0	PID000005
IID000006	39001.0	PID000006
IID000007	59001.0	PID000007
IID000008	19001.0	PID000008
IID000009	44001.0	PID000009
IID000010	69001.0	PID000010
IID000011	34001.0	PID000011
IID000012	24001.0	PID000012
IID000013	89001.0	PID000013
IID000014	54001.0	PID000014
IID000015	49001.0	PID000015
IID000016	69001.0	PID000001
IID000017	29001.0	PID000002
IID000018	49001.0	PID000002
IID000019	89001.0	PID000003
IID000020	119001.0	PID000003
IID000021	14001.0	PID000004
IID000022	24001.0	PID000004
IID000023	49001.0	PID000005
IID000024	99001.0	PID000005

Figure 19: Product instance (1)

IID000025	34001.0	PID000006
IID000026	74001.0	PID000006
IID000027	24001.0	PID000007
IID000028	59001.0	PID000007
IID000029	19001.0	PID000008
IID000030	1449001.0	PID000008
IID080460	47000.0	PID144103
IID080466	47000.0	PID144103
IID080467	47000.0	PID144103
IID080468	47000.0	PID144103
IID080469	47000.0	PID144103
IID270747	112000.0	PID186682
IID339757	24000.0	PID186682
IID684325	52000.0	PID110464
IID691489	14000.0	PID138894
IID694971	69000.0	PID110464
IID757288	20000.0	PID163853
IID774582	5000.0	PID163853

Figure 20: Product instance (2)

review_id	no_stars	content	time_created	product_id	user_id
RID000001	4	Great product! Very comfortab…	2023-01-15 08:30:00	PID000001	UID000001
RID000002	5	Excellent service and fast deli…	2023-02-02 12:45:00	PID000002	UID000002
RID000003	3	Average quality, expected more.	2023-03-10 15:20:00	PID000003	UID000003
RID000004	5	Love it! Best purchase ever.	2023-04-05 09:10:00	PID000004	UID000004
RID000005	2	Not satisfied with the product. …	2023-05-20 14:55:00	PID000005	UID000005
RID000006	4	Impressive product quality. W…	2023-06-12 11:25:00	PID000006	UID000006
RID000007	3	Decent price for the features p…	2023-07-18 08:40:00	PID000007	UID000007
RID000008	5	Absolutely love the design. A+…	2023-08-23 13:15:00	PID000008	UID000008
RID000009	2	Disappointed with the shippin…	2023-09-30 17:30:00	PID000009	UID000009
RID000010	4	Good value for the money. Ha…	2023-10-05 09:55:00	PID000010	UID000010
RID000011	1	Terrible experience. The prod…	2023-11-11 14:20:00	PID000011	UID000011
RID000012	5	Exactly what I was looking for.…	2023-12-15 10:45:00	PID000012	UID100013
RID000013	3	Average product. Nothing spe…	2023-01-20 16:10:00	PID000013	UID100013
RID000014	5	Amazing customer service. Q…	2023-02-25 12:35:00	PID000014	UID000001
RID000015	2	Not as described. Misleading …	2023-03-03 08:00:00	PID000015	UID000002
RID000016	4	Sturdy construction. Happy wi…	2023-04-08 13:25:00	PID000001	UID000003
RID000017	1	Waste of money. Poor quality …	2023-05-15 17:50:00	PID000002	UID000004
RID000018	5	Excellent packaging. Product …	2023-06-20 11:15:00	PID000003	UID000005
RID000019	3	Satisfactory purchase. Could …	2023-07-25 16:40:00	PID000004	UID000006
RID000020	4	Good product overall. Met my …	2023-08-30 12:05:00	PID000005	UID000007
RID000021	2	Unpleasant odor from the pro…	2023-09-05 08:30:00	PID000006	UID000011
RID000022	5	Outstanding performance. Wo…	2023-10-10 13:55:00	PID000007	UID100083
RID000023	3	Not user-friendly. Complicated…	2023-11-15 17:20:00	PID000008	UID000006
RID000024	5	Exceptional design. Adds eleg…	2023-12-20 11:45:00	PID000009	UID100049
RID000025	4	Reliable and efficient. Happy …	2023-01-25 16:10:00	PID000010	UID100074
RID399786	4	Good. Sau khi phoi con co mu…	2023-11-14 00:00:00	PID144103	UID100013
RID411285	5	Binh luan mang tinh chat nha…	2023-11-14 00:00:00	PID144103	UID100046

Figure 21: Review

user_id	shop_id
UID000002	SID000001
UID000003	SID000002
UID000004	SID000003
UID000005	SID000004
UID000006	SID000005
UID100052	SID100052
UID100074	SID100074
UID100083	SID100083

Figure 22: Seller

shop_id	bio	no_following	no_follower	url_link	rating	name	date_joined	no_product
SID000001	Fashion paradise ...	100	500	https://ChicBoutiq...	4.5	Chic Boutique	2023-01-01	0
SID000002	Handcrafted treas...	150	300	https://ArtisanHav...	3.8	Artisan Haven	2023-02-15	0
SID000003	Explore tech innov...	200	700	https://GadgetGal...	4.2	Gadget Galaxy	2023-03-20	0
SID000004	Books, coffee, an...	50	100	https://BookNook....	4.0	Book Nook	2023-04-10	0
SID000005	Outdoor gear and ...	80	200	https://Adventure...	4.8	Adventure Out	2023-05-05	0
SID100052	Sell something to ...	0	0	https://Tiemgiatui....	0.0	Tiem giat ui	2023-02-15	0
SID100074	Super yummy thin...	0	0	https://Banhtrang....	0.0	Banh trang	2023-03-20	0
SID100083	Moniso official sec...	0	0	https://secondHan...	0.0	Cua hang second ...	2023-05-05	0
SIDffffff	No description	0	0	https://ignore.com	0.0	ALL SHOPS	2023-12-12	0

Figure 23: Shop

user_id	number	no_product	date_created
UID000001	1	1	2023-12-14
UID000002	2	1	2023-12-14
UID000003	3	1	2023-12-14
UID000004	4	1	2023-12-14
UID000005	5	1	2023-12-14
UID100013	1	1	2023-11-14
UID100046	1	1	2023-11-15
UID100049	1	1	2023-11-16
UID100052	1	2	2023-11-18
UID100074	1	0	2023-11-19
UID100083	1	0	2023-06-14

Figure 24: Shopping cart

shop_id	number	street	ward	district	city	country
SID000001	20	Main St	Downtown	Central	Cityville	USA
SID000002	30	Oak Ave	Suburbia	East	Townsville	USA
SID000003	15	Elm St	Downtown	North	Metropolis	USA
SID000004	40	Maple St	Village	South	Ruraltown	USA
SID000005	10	Cedar St	Central	West	Cityville	USA
SID100052	248	Ly Thuong Kiet	phuong 15	quan 10	thanh pho Ho Chi Minh	Viet Nam
SID100074	13	Tan Lap, khu pho 6	Linh Trung	Thu Duc	thanh pho Ho Chi Minh	Viet Nam
SID100083	1	Ly Thuong Kiet	phuong 15	quan 10	thanh pho Ho Chi Minh	Viet Nam

Figure 25: Shop's address

shop_id	phone_number
SID000001	0234567890
SID000002	0876543210
SID000003	0512345675
SID000004	0512345678
SID000005	0890123456
SID100052	0075948675
SID100074	0345138094
SID100083	0429363132

Figure 26: Shop's phone number

user_id	user_name	nickname	linking_phone_nu...	linking_email	password	user_id_contact	contact_id
UID000001	NguyenHuan	huan2114	0356382638	huan1122@gmail.com	nothing@@ink	UID000001	CID000001
UID000002	JohnDoe	johndoe123	0234567890	john.doe@gmail.com	securePass123	UID000002	CID000002
UID000003	AliceSmith	ali_smith	0876543210	alice.smith@gmail.com	strongPassword	UID000003	CID000003
UID000004	BobJohnson	bobJ	0512345678	bob.j@gmail.com	pass1234	UID000004	CID000004
UID000005	EmilyJones	emily_j	0367890123	emily.jones@gmail.com	mySecurePassword	UID000005	CID000005
UID000006	DavidMiller	david_m	0890123456	david.miller@gmail.c...	p@ssw0rd	UID000006	CID000006
UID000007	SophiaBrown	sophia_b	0345678901	sophia.brown@gmail...	brownie123	UID000007	CID000007
UID000008	DanielWhite	danW	0765432109	daniel.white@gmail.c...	danielPass	UID000008	CID000008
UID000009	OliviaDavis	olivia_d	0210987654	olivia.davis@gmail.com	oliviaPassword	UID000009	CID000009
UID000010	WilliamWilson	willW	0087654321	william.w@gmail.com	wilsonPass	UID000010	CID000010
UID000011	AvaTaylor	ava_t	0357924680	ava.taylor@gmail.com	taylor123	UID000011	CID000011
UID100013	son_0175	son_0175	0167502421	son_0175@gmail.com	12345678	UID100013	CID100013
UID100046	tong_01	tong_01	0854342531	tong_01@gmail.com	12345678	UID100046	CID100046
UID100049	hoangaaa	hoangaaa	0761712741	hoangaaa@gmail.com	12345678	UID100049	CID100049
UID100052	tungbbb	tiem_giat_ui	0075948675	tungbbb@gmail.com	12345678	UID100052	CID100052
UID100074	huanccc	banh_trang_00	0345138094	huanccc@gmail.com	12345678	UID100074	CID100074
UID100083	sonddd	secondhand_gau	0429363132	sonddd@gmail.com	12345678	UID100083	CID100083

Figure 27: User

product_id	variant_name	price	remaining_amount	details	img
PID000001	Premium	69001.0	30	Premium Variant with extra co…	NULL
PID000001	Standard	49001.0	50	Standard Variant of Comfort S…	NULL
PID000002	Classic Fit	29001.0	40	Classic fit Casual Shirt for eve…	NULL
PID000002	Slim Fit	49001.0	20	Slim fit Casual Shirt for a mod…	NULL
PID000003	32GB	89001.0	100	32GB Variant of Smartphone …	NULL
PID000003	64GB	119001.0	50	64GB Variant with extended st…	NULL
PID000004	Copper	14001.0	30	Copper Candle Holder for a ru…	NULL
PID000004	Silver	24001.0	20	Silver Candle Holder for a sle…	NULL
PID000005	Basic Kit	49001.0	80	Basic Camping Gear for outdo…	NULL
PID000005	Deluxe Kit	99001.0	40	Deluxe Camping Gear with ad…	NULL
PID000006	Classic	74001.0	15	Classic Elegance Necklace fo…	NULL
PID000006	Statement Piece	34001.0	25	Statement Necklace with uniq…	NULL
PID000007	Bifold	24001.0	50	Bifold Leather Wallet for every…	NULL
PID000007	Trifold	59001.0	30	Trifold Leather Wallet with mul…	NULL
PID000008	Basic Model	19001.0	70	Basic Laptop for everyday tasks.	NULL
PID000008	Pro Model	1449001.0	20	Pro Laptop with advanced spe…	NULL
PID000009	Modern Design	44001.0	15	Modern Table Lamp for conte…	NULL
PID000009	Vintage Style	89001.0	25	Vintage Style Lamp for a touc…	NULL
PID000010	Classic Blue	9001.0	45	Classic Blue Water Bottle for …	NULL
PID000010	Stainless Steel	29001.0	25	Stainless Steel Water Bottle f…	NULL
PID000011	Athletic Style	39001.0	30	Athletic Style Sneakers for act…	NULL
PID000011	Casual Comfort	79001.0	20	Casual Comfort Sneakers for …	NULL

Figure 28: Variant (1)

PID000012	Relaxed Fit	29001.0	35	Relaxed Fit Denim Jeans for c…	NULL
PID000012	Skinny Fit	69001.0	15	Skinny Fit Denim Jeans for a …	NULL
PID000013	Compact Camera	499001.0	20	Compact Digital Camera for o…	NULL
PID000013	Pro Camera	149001.0	10	Pro Digital Camera for profess…	NULL
PID000014	Cozy Plaid	14001.0	15	Cozy Plaid Throw Blanket for …	NULL
PID000014	Luxury Velvet	34001.0	10	Luxury Velvet Throw Blanket f…	NULL
PID000015	Explorer	119001.0	15	Explorer Hiking Boots for the …	NULL
PID000015	Trailblazer	59001.0	25	Trailblazer Hiking Boots for ou…	NULL
PID110464	1kg	69000.0	3	No description	NULL
PID110464	500g	52000.0	2	No description	NULL
PID163853	100g	5000.0	5	No description	NULL
PID163853	500g	20000.0	8	No description	NULL
PID186682	Combo1	24000.0	4	No description	NULL
PID186682	Combo2	112000.0	5	No description	NULL

Figure 29: Variant (2)

voucher_id	quantity	discount	lowest_applied_...	start_time	end_time	type	maximum_cash...	percent
VCH000001	50	NULL	10000.0	2023-01-01 00:00:...	2023-02-01 00:00:...	For delivery	10000.0	0.05
VCH000002	100	20000.0	50000.0	2023-02-15 00:00:...	2023-03-15 00:00:...	For products	12000.5	0.10
VCH000003	30	NULL	15000.0	2023-03-01 00:00:...	2023-04-01 00:00:...	For products	5000.0	0.30
VCH000004	80	25000.0	100000.0	2023-04-15 00:00:...	2023-05-15 00:00:...	For delivery	NULL	NULL
VCH000005	20	NULL	30000.0	2023-05-01 00:00:...	2023-06-01 00:00:...	For delivery	8000.0	0.60
VCH000006	40	NULL	25000.0	2023-06-15 00:00:...	2023-07-15 00:00:...	For products	10000.0	0.20
VCH000007	60	15000.0	70000.0	2023-07-01 00:00:...	2023-08-01 00:00:...	For delivery	13000.0	0.08
VCH000008	25	NULL	20000.0	2023-08-15 00:00:...	2023-09-15 00:00:...	For products	5000.0	0.50
VCH000009	70	30000.0	120000.0	2023-09-01 00:00:...	2023-10-01 00:00:...	For delivery	NULL	NULL
VCH000010	20	8000.0	0.0	2023-11-14 00:00:00	NULL	For products	NULL	NULL
VCH000011	15	NULL	40000.0	2023-10-15 00:00:...	2023-11-15 00:00:00	For products	12000.0	0.19

Figure 30: Voucher

3 Application implementation

	user_id	user_name	nickname	linking_phone_number	linking_email	password	user_id_contact	contact_id
1	UID123456	Jence	Yatta	0911112222	son.dongtech@hcmut.edu.vn	123	UID123456	CID123456

Figure 31: New seller to demo

	product_id	description	name	total_remaining	no_sales	minimum_price	maximum_price	category_id	shop_id	img
1	PID000000	A timeless classic, "To Kill a Mocki...	"To Kill a Mockingbird" by Harper Lee	70	0	150000.0	200000.0	CAT123456	SID123456	To-Kill-A-Mockingbi
2	PID000016	Orwell's dystopian masterpiece, "1...	"1984" by George Orwell	50	20	200000.0	300000.0	CAT123456	SID123456	1984.jpg
3	PID000017	Set in the extravagant 1920s, "The ...	"One Hundred Years of Solitude" by Gabriel Garcia	140	30	180000.0	300000.0	CAT123456	SID123456	One-Hundred-Year
4	PID000018	Marquez's magnum opus, "One H...	"The Catcher in the Rye" by J.D. Salinger	40	50	250000.0	400000.0	CAT123456	SID123456	The-Catcher-In-The
5	PID000019	J.D. Salinger's classic follows the ...	"Sapiens: A Brief History of Humankind" by Yuval N	110	69	400000.0	600000.0	CAT123456	SID123456	Sapiens.jpg
6	PID000020	Harari's insightful exploration, "Sa...	"The Hobbit" by J.R.R. Tolkien	290	100	300000.0	300000.0	CAT123456	SID123456	Camping-Gear-Bas
7	PID000021	Paulo Coelho's philosophical nove...	"The Alchemist" by Paulo Coelho	120	24	220000.0	220000.0	CAT123456	SID123456	The-Alchemist.jpg
8	PID000022	Huxley's futuristic novel, "Brave Ne...	"Brave New World" by Aldous Huxley	140	50	300000.0	300000.0	CAT123456	SID123456	Brave-New-World.jp
9	PID000023	McCarthy's haunting novel, "The R...	"The Road" by Cormac McCarthy	90	35	160000.0	160000.0	CAT123456	SID123456	The-Road.jpg
10	PID000024	Set in the extravagant 1920s, "The ...	"The Great Gatsby" by F. Scott Fitzgerald	190	20	190000.0	190000.0	CAT123456	SID123456	NULL

Figure 32: More data in Product to demo

	product_id	variant_name	price	remaining_amount	details
1	PID000000	normal	150000.0	50	Normal
2	PID000000	special	200000.0	30	Special
3	PID000016	normal	200000.0	40	Normal
4	PID000016	special	300000.0	20	Special
5	PID000017	normal	180000.0	100	Normal
6	PID000017	special	300000.0	50	Special
7	PID000018	normal	250000.0	40	Normal
8	PID000018	special	400000.0	10	Special
9	PID000019	normal	400000.0	80	Normal
10	PID000019	special	600000.0	40	Special

Figure 33: More Variant

	review_id	no_stars	content	time_created	product_id	user_id
1	RID300000	1	This book is such a mess.	2023-12-13 21:58:45.743	PID000000	UID000001
2	RID300001	2	The topic should be considered again.	2023-12-13 21:58:45.743	PID000016	UID000001
3	RID300002	3	No offence but this book gives no inspiration.	2023-12-13 21:58:45.743	PID000017	UID000001
4	RID300003	4	Reading this book recalls my memory vaguely abo...	2023-12-13 21:58:45.743	PID000018	UID000001
5	RID300004	5	This book is outstanding. The publisher should pri...	2023-12-13 21:58:45.743	PID000019	UID000001

Figure 34: More Review

	order_id	status	no_product	time_order	ID_payment	delivery_id
1	ORD30000000001	Confirming	2	2023-12-13 21:59:25.920	PAY000001	DEL000001
2	ORD30000000002	Confirming	2	2023-12-13 21:59:25.920	PAY000001	DEL000001
3	ORD30000000003	Waiting pickup	2	2023-12-13 21:59:25.920	PAY000001	DEL000002
4	ORD30000000004	Waiting pickup	2	2023-12-13 21:59:25.920	PAY000001	DEL000002
5	ORD30000000005	Delivering	2	2023-12-13 21:59:25.920	PAY000001	DEL000004
6	ORD30000000006	Delivering	2	2023-12-13 21:59:25.920	PAY000001	DEL000004
7	ORD30000000007	Done	2	2023-12-13 21:59:25.920	PAY000001	DEL000003
8	ORD30000000008	Done	2	2023-12-13 21:59:25.920	PAY000001	DEL000003

	product_id	instance_id	order_id
1	PID000000	IID000031	ORD30000000005
2	PID000001	IID000031	ORD30000000005
3	PID000002	IID000031	ORD30000000005
4	PID000003	IID000031	ORD30000000005
5	PID000004	IID000031	ORD30000000005
6	PID000005	IID000031	ORD30000000005
7	PID000006	IID000031	ORD30000000005
8	PID000007	IID000031	ORD30000000005

	order_id	user_id	user_id_cart	number
1	ORD30000000001	NULL	UID000001	1
2	ORD30000000002	NULL	UID000001	1
3	ORD30000000003	NULL	UID000001	1
4	ORD30000000004	NULL	UID000001	1
5	ORD30000000005	NULL	UID000001	1
6	ORD30000000006	NULL	UID000001	1
7	ORD30000000007	NULL	UID000001	1
8	ORD30000000008	NULL	UID000001	1
9	ORD30000000009	NULL	UID000001	1
10	ORD30000000010	NULL	UID000001	1
11	ORD30000000011	NULL	UID000001	1
12	ORD30000000012	NULL	UID000001	1

Figure 35: Data demo in Order, Is_contained and Places

3.1 Trigger

Write 8 triggers to control INSERT, UPDATE, DELETE actions on some created tables. Meet the following requirements:

- *Have at least 1 trigger that calculates updates on data in a table other than the one the trigger is set on. (Trigger related to computing derived attributes)*
- *Prepare codes and illustrative data to test the triggers when reporting.*

3.1.1 Trigger to update no_product

- **Trigger description**: This trigger is designed to update the `no_product` column in the `Shop` table based on changes made to the `Product` table. It increases or decreases the `no_product` count for the associated `shop_id` depending on whether a record is inserted or deleted in the `Product` table:
  - The trigger fires after an INSERT or DELETE operation on the `Product` table.
  - This checks if there are rows in the inserted table, indicating an INSERT operation. If true, it sets @exceed = 1 and declares a cursor to iterate over the inserted rows. If the condition is false, it means a DELETE operation, so it sets @exceed = -1 and declares a cursor to iterate over the deleted rows.
  - Update the `Shop` table, increase or decrease the `no_product` count based on the value of @exceed for the corresponding `shop_id`.
- **The trigger creation code**:

```
create trigger update_no_product
on Product
after insert, delete
as
begin
  set nocount on;

  ----------------
  declare @exceed int;
  declare @shop_id varchar(9);


  if exists (select 0 from inserted) --insert
  begin
```

```
    declare cur Cursor for select shop_id from inserted;
    set @exceed = 1
  end
  else
  begin
    declare cur Cursor for select shop_id from deleted;
    set @exceed = -1
  end
  open cur;
  fetch next from cur into @shop_id;
  ----------------
  while @@FETCH_STATUS = 0
  begin
    update Shop
    set no_product = no_product  + @exceed
    where shop_id = @shop_id
    fetch next from cur into @shop_id
  end

  close cur;
  deallocate cur;
end;

```

- **Trigger testing**:

```
insert into Product values('PID331122', 'Ao phong gia re', 'Ao phong', 100, 0,
    200000,200000, 'CAT000005', 'SID000005',NULL);

```

- **Result**

shop_id	bio	no_following	no_follower	url_link	rating	name	date_joined	no_product
SID000005	Outdoor gear and...	80	200	https://Adventure...	4.8	Adventure Out	2023-05-05	0

Figure 36: Shop table before inserting new product in Product table

shop_id	bio	no_following	no_follower	url_link	rating	name	date_joined	no_product
SID000005	Outdoor gear and ...	80	200	https://Adventure...	4.8	Adventure Out	2023-05-05	1

Figure 37: no_product in Shop table updated after inserting new product in Product table

3.1.2 Trigger to update minimum_price, maximum_price, total_remaining

- **Trigger description**: This trigger is designed to keep the `Product` table updated with aggregated values (`minimum_price`, `maximum_price`, and `total_remaining`) based on changes in the `Variant` table:
  - The trigger fires after an INSERT, DELETE or UPDATE operation on the `Variant` table.
  - Check if there are rows in the inserted table, indicating an INSERT or UPDATE operation. If true, it declares a cursor to iterate over distinct `product_id` values from the inserted table. Otherwise, it does the same for the deleted table.
  - Calculate the minimum price, maximum price, and total remaining amount for each product based on its variants.
  - Update the corresponding row in the `Product` table with the calculated values.

  The trigger creation code:

```
create trigger update_min_max_totalremain
on Variant
after insert, delete,update
as
begin
```

```
  set nocount on;

  --------------------
  if exists (select 0 from inserted) --insert or update
  begin
    declare cur Cursor for select distinct product_id from inserted
  end
  else
  begin
    declare cur Cursor for select distinct product_id from deleted
  end

  declare @product_id varchar(9);
  declare @min decimal(10,1) = 0;
  declare @max decimal(10,1) = 0;
  declare @totalremain int = 0;
  open cur
  fetch next from cur into @product_id
  --------------------
  while @@FETCH_STATUS = 0
  begin
    set @min = (
      select min(price)
      from Variant
      where product_id = @product_id)
    set @max = (
      select max(price)
      from Variant
      where product_id = @product_id)
    set @totalremain = (
      select sum(remaining_amount)
      from Variant
      where product_id = @product_id)


    if @min is null set @min = 0
    if @max is null set @max = 0
    if @totalremain is null set @totalremain = 0


    update Product
    set minimum_price = @min ,
      maximum_price = @max,
      total_remaining = @totalremain
    where product_id = @product_id
    fetch next from cur into @product_id
  end
  close cur
  deallocate cur
  --------------------
end

```

- **Trigger testing**:

```
update Variant set price = 100000, remaining_amount = 10 where product_id = 'PID000000'
    and variant_name = 'normal'

```

- **Result**:

	product_id	variant_name	price	remaining_amount	details	img
1	PID000000	normal	150000.0	50	Normal	NULL
2	PID000000	special	200000.0	30	Special	NULL

	product_id	description	name	total_remaining	no_sales	minimum_price	maximum_price	category_id	shop_id	img
1	PID000000	A timeless classic, "To Kill a Mockingbird" delv...	"To Kill a Mockingbird" by Harper Lee	80	0	150000.0	200000.0	CAT123456	SID123456	To-Kill-A-Mockingbirc

Figure 38: Product table before inserting new variant

	product_id	variant_name	price	remaining_amount	details	img
1	PID000000	normal	100000.0	10	Normal	NULL
2	PID000000	special	200000.0	30	Special	NULL

	product_id	description	name	total_remaining	no_sales	minimum_price	maximum_price	category_id	shop_id	img
1	PID000000	A timeless classic, "To Kill a Mockingbird" delv...	"To Kill a Mockingbird" by Harper Lee	40	0	100000.0	200000.0	CAT123456	SID123456	To-Kill-A-Mockingbir

Figure 39: Result in Product table after inserting

3.1.3 Trigger to update no_product in Order

- **Trigger description**: The trigger maintains the `no_product` column in the `Order` table by updating it based on changes in the `Is_contained` table:
  - The trigger fires after an INSERT or DELETE operation on the `Is_contained` table.
  - Check if there are rows in the inserted table, indicating an INSERT operation. If true, it declares a cursor to iterate over distinct `order_id` values from the inserted table. Otherwise, it does the same for the deleted table.
  - Count the number of products.
  - Update the `Order` table with the calculated `no_product` value.

  The trigger creation code:

```
create trigger update_no_product_of_order
on Is_contained
after insert, delete
as
begin
  set nocount on;

  --------------------
  if exists (select 0 from inserted)
  begin
    declare cur Cursor for (select distinct order_id from inserted);
  end
  else
  begin
    declare cur cursor for (select distinct order_id from deleted);
  end
  declare @order_id varchar(14);
  declare @no_product int;
  declare @list_product table(product_id varchar(9));
  open cur
  fetch next from cur into @order_id
  --------------------
  while @@FETCH_STATUS = 0
  begin
    insert into @list_product
    select distinct p.product_id
    from Is_contained i, Product_instance p
    where i.order_id = @order_id and p.instance_id = i.instance_id;

    set @no_product = (select count(*) from @list_product)
    update [Order]
    set no_product = @no_product
    where order_id = @order_id
    fetch next from cur into @order_id
    delete @list_product
  end
  close cur
  deallocate cur
end

```

- **Trigger testing**:

```
Insert into Is_contained VALUES ('IID774582','ORD00000000001');

```

- **Result**

order_id	status	no_product	time_order	ID_payment	delivery_id
ORD00000000001	Confirming	3	2023-12-13 23:45:17	PAY000001	DEL000001

Figure 40: Order table before inserting

order_id	status	no_product	time_order	ID_payment	delivery_id
ORD00000000001	Confirming	4	2023-12-12 18:19:15	PAY000001	DEL000001

Figure 41: Result in Order table after inserting

3.1.4 Trigger to update no_product in Shopping cart

- **Trigger description**: The trigger named `update_no_product_of_cart` maintains the `no_product` column in the `Shopping_cart` table by updating it based on changes in the `[Add]` table:
  - The trigger fires after an INSERT or DELETE operation on the `[Add]` table.
  - Check if there are rows in the inserted table, indicating an INSERT operation. If true, it declares a cursor to iterate over distinct `user_id` and `number` values from the inserted table. Otherwise, it does the same for the deleted table.
  - Update the `Shopping_cart` table with the calculated `no_product` value.

  The trigger creation code:

```
create trigger update_no_product_of_cart
on [Add]
after insert,delete
as
begin
  set nocount on;

  --------------------
  if exists (select 0 from inserted)
  begin
    declare cur Cursor for (select distinct user_id,number  from inserted);
  end
  else
  begin
    declare cur cursor for (select distinct user_id,number from deleted);
  end
  declare @user_id varchar(9);
  declare @number   int;
  open cur;
  fetch next from cur into @user_id, @number;
  ----------------------
  while @@FETCH_STATUS = 0
  begin
    update Shopping_cart
    set no_product = (select count(*) from [Add] where user_id = @user_id and number =
    @number )
    where user_id = @user_id and number = @number;

    fetch next from cur into @user_id, @number;
  end
  close cur
  deallocate cur

end
```

- **Trigger testing**:

```
delete from [Add] where product_id = 'PID000001' AND user_id = 'PID000001';

```

- **Result**

user_id	number	no_product	date_created
UID000001	1	0	2023-12-12

Figure 42: Result in Shopping_cart table after deleting

3.1.5 Trigger to check the quantity of Voucher

- **Trigger description**: The trigger named `check_n_update_quantity_Voucher` is designed for the `Applies` table. It triggers after an `INSERT` operation and performs the following tasks:
  - Checks the availability of the voucher based on certain conditions.
  - If the conditions are not met or the voucher quantity is less than or equal to 0, it prints an error message, rolls back the transaction, and returns.
  - Updates the quantity of the voucher by decreasing it by 1 for each voucher applied.

  The trigger creation code:

```
create trigger check_n_update_quantity_Voucher
on Applies
after Insert
as
begin
  --check available
  declare cur Cursor for (select * from inserted)
  declare @ord varchar(14)
  declare @vch varchar(9)

  open cur
  fetch next from cur into @ord, @vch

  while @@FETCH_STATUS = 0
  begin
    if not exists (
      select * from Can_apply
      where voucher_id = @vch and
      (category_id = 'CATffffff' or category_id in (
        select distinct p.category_id from Is_contained i, Product_instance pin, Product p
        where i.order_id = @ord and i.instance_id = pin.instance_id and pin.product_id = p.product_id)) and
      (shop_id = 'SIDffffff' or shop_id in (
        select distinct p.shop_id from Is_contained i, Product_instance pin, Product p
        where i.order_id = @ord and i.instance_id = pin.instance_id and pin.product_id = p.product_id))
    ) or (select quantity from Voucher where voucher_id = @vch) <= 0
    begin
      print 'Can not apply voucher due to its conditions'
      close cur
      deallocate cur
      rollback transaction
      return
    end

    fetch next from cur into @ord, @vch
  end
  close cur
  deallocate cur

  -- update quantity
  declare cur Cursor for (select voucher_id from inserted);

  open cur
  fetch next from cur into @vch;

  while @@FETCH_STATUS = 0
```

```
  begin
    update Voucher
    set quantity = quantity - 1
    where voucher_id = @vch

    fetch next from cur into @vch
  end
  close cur
  deallocate cur
end

```

- **Trigger testing**:

```
Delete from Applies where order_id = 'ORD00000000032';
insert into Applies VALUES ('ORD00000000032','VCH000011');
insert into Applies values ('ORD30000000012','VCH000006');

```

- **Result**

order_id	voucher_id
ORD00000000001	VCH000006
ORD00000000002	VCH000007
ORD00000000003	VCH000008
ORD00000000004	VCH000009
ORD00000000005	VCH000010
ORD00000000031	VCH000010
ORD00000000032	VCH000011

Figure 43: Result in Applies table after reinserting the suitable record for voucher

```
Can not apply voucher due to its conditions
Msg 3609, Level 16, State 1, Line 247
The transaction ended in the trigger. The batch has been aborted.

Completion time: 2023-12-14T21:50:04.9327607+07:00
```

Figure 44: Trigger rollback at last statement

3.1.6 Trigger to calculate the avg_rating of Shop

- **Trigger description**: The trigger named `cal_avg_rating` is designed for the `Review` table. It triggers after an `INSERT` or `UPDATE` operation and performs the following tasks:
  - Declares a cursor to iterate over distinct `shop_id` values from the `inserted` table.
  - Calculates the average rating for each shop based on the reviews of its associated products.
  - If there are no reviews, sets the average rating to 0.
  - Updates the `rating` column in the `Shop` table with the calculated average rating.

  The trigger creation code:

```sql
create trigger cal_avg_rating
on Review
after insert,update
as
begin
  declare cur Cursor for (
  select distinct s.shop_id from inserted i,Shop s, Product p
  where i.product_id = p.product_id and p.shop_id = s.shop_id)

  declare @shop_id varchar(9)
  open cur
  fetch next from cur into @shop_id
  while @@FETCH_STATUS = 0
  begin
    declare @avgRating decimal(2,1)
    set @avgRating = (
      select AVG(cast(r.no_stars as decimal(2,1))) as avgRate
      from Review r, Product p
      where r.product_id = p.product_id and p.shop_id = @shop_id)
    if @avgRating is null
      set @avgRating = 0

    update Shop
    set rating = @avgRating
    where shop_id = @shop_id;

    fetch next from cur into @shop_id

  end
  close cur
  deallocate cur
end

```

- **Trigger testing**:

```sql
UPDATE Review SET no_stars = 2 WHERE review_id = 'RID000001';

```

- **Result**

shop_id	bio	no_following	no_follower	url_link	rating	name	date_joined	no_product
SID000001	Fashion paradise ...	100	500	https://ChicBoutiq...	4.5	Chic Boutique	2023-01-01	0

Figure 45: Shop table before changing

shop_id	bio	no_following	no_follower	url_link	rating	name	date_joined	no_product
SID000001	Fashion paradise ...	100	500	https://ChicBoutiq...	2.6	Chic Boutique	2023-01-01	0

Figure 46: Result in Shop table after changing the no_star in Review

3.1.7 Trigger to check instance_id before update

- **Trigger description**: The trigger named `check_before_update` is designed for the `Product_instance` table. It triggers for an `UPDATE` operation and performs the following task:
  - Declares a cursor to iterate over `instance_id` values from the `inserted` table.
  - Checks if the product instance exists in the `Is_contained` table, indicating that it is already part of an order.
  - If the instance is found in an order, prints an error message, rolls back the transaction, and returns.

  The trigger creation code:

```sql
create trigger check_before_update
on Product_instance
for update
as
begin
  declare cur Cursor for (select instance_id, current_price from inserted)

  declare @instance_id varchar(9)
  declare @new_price decimal(10,1)

  open cur
  fetch next from cur into @instance_id,@new_price

  while @@FETCH_STATUS = 0
  begin
    if exists (select * from Is_contained where instance_id = @instance_id) -- exists in order
    begin
      declare @old_price decimal(10,1) = (select current_price from Product_instance where instance_id = @instance_id)
      if(@old_price != @new_price)
      begin
        print 'Error in product instance ' + @instance_id
        print 'Can not change current price of a instance that is in order'
        close cur
        deallocate cur
        rollback transaction
        return
      end
    end
    fetch next from cur into @instance_id,@new_price
  end
  close cur
  deallocate cur
end

```

- **Trigger testing**:

```sql
update Product_instance set current_price = 100000 where instance_id = 'IID000001';

```

- **Result**

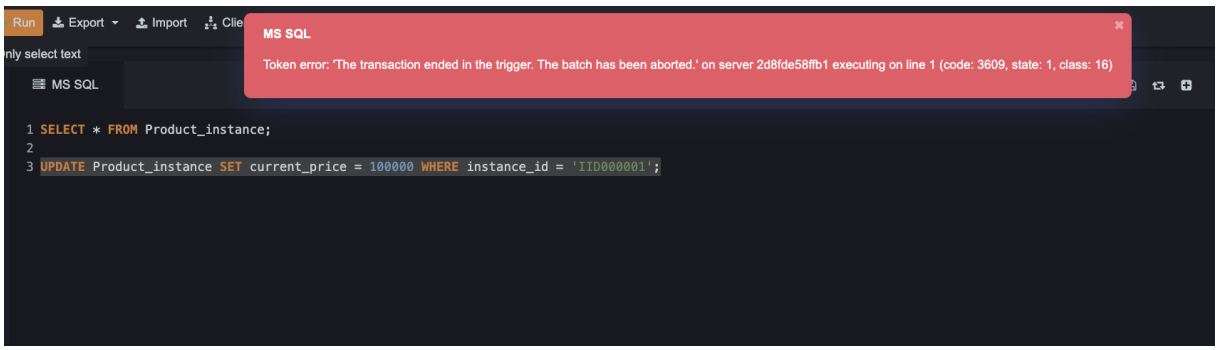

Figure 47: Message showing after changing the price of product already in someone's order

3.1.8 Trigger to update total_remaining after adding instance

- **Trigger description**: The trigger named `update_remain_after_add_instance` is designed for the `Product_instance` table. It triggers after an `INSERT` operation and performs the following tasks:
  - Declares a cursor to iterate over `product_id` and the count of instances added (`numAdd`) from the `inserted` table.

- Checks if the total remaining of the product would be exceeded after adding the instances. If so, prints an error message, rolls back the transaction, and returns.
- Updates the `total_remaining` column in the `Product` table by decrementing it based on the number of instances added for each product.

The trigger creation code:

```
create trigger update_remain_after_add_instance
on Product_instance
after insert
as
begin
  declare cur Cursor for (select product_id, count(*) as numAdd from inserted group by
    product_id)
  declare @product_id varchar(9)
  declare @numAdd int

  open cur
  fetch next from cur into @product_id, @numAdd

  while @@FETCH_STATUS = 0
  begin
    if @numAdd > (select total_remaining from Product where product_id = @product_id)
    begin
      print 'Exceed total remaining of product'
      close cur
      deallocate cur
      rollback transaction
      return
    end
    fetch next from cur into @product_id, @numAdd
  end

  close cur
  deallocate cur

  --------------------------------
  declare cur Cursor for (select product_id, count(*) as numAdd from inserted group by
    product_id)
  open cur
  fetch next from cur into @product_id, @numAdd

  while @@FETCH_STATUS = 0
  begin
    update Product
    set total_remaining = total_remaining - @numAdd
    where product_id = @product_id;

    fetch next from cur into @product_id, @numAdd
  end

  close cur
  deallocate cur
end

```

- **Trigger testing**:

```
insert into Product_instance VALUES ('IIDffffff',40000,'PID000001');

```

- **Result**

product_id	description	name	total_remaining	no_sales	minimum_price	maximum_price	category_id	shop_id	img
PID000001	Stylish and co...	Comfort Shoes	100	50	29001.0	69001.0	CAT000001	SID000001	NULL

Figure 48: Product table before inserting

product_id	description	name	total_remaining	no_sales	minimum_price	maximum_price	category_id	shop_id	img
PID000001	Stylish and co…	Comfort Shoes	99	50	29001.0	69001.0	CAT000001	SID000001	NULL

Figure 49: Result in Product table

3.2 Procedure

Write 5 procedures containing only queries to display data, with input parameters for WHERE and/or HAVING clauses (if any), including:

- *1 query from 2 or more tables with WHERE and ORDER BY clauses.*
- *1 query with aggregate function, GROUP BY, HAVING, WHERE, and ORDER BY clauses involving 2 or more tables.*
- *Have at least 1 procedure related to retrieving data from the table in query 1.2.1.*
- *Prepare statements and illustrative data for calling the procedures in a report.*

3.2.1 Insert procedure

Procedure to insert product

- **Stored procedure description**: The stored procedure named `insert_product` is designed to insert a new product into the `Product` table. It takes several parameters, some of which are optional, and performs validation checks before insertion. The procedure also outputs a result message indicating the success or failure of the insertion.

  The stored procedure code:

```
create procedure insert_product
@product_id varchar(9), --mendatory
@des varchar(1000) = null, --optional
@name varchar(50),  --mendatory
@total_remaining int, --mendatory
@price  decimal(10,1), --mendatory
@cat varchar(9) = null, --optional
@sid varchar(9), --mendatory
@img varchar(1000) = null,
@result varchar(1000) = '' Output --default = ''
as
begin
  if exists (select product_id from Product where product_id = @product_id)
    set @result = @result + 'Product ID has already existed_'
  if not exists (select shop_id from Shop where shop_id = @sid)
    set @result = @result + 'Invalid Shop ID_'
  if @cat is not null and not exists (select category_id from Category where category_id
    = @cat)
    set @result = @result + 'Invalid Catgory ID_'
  if @total_remaining < 0
    set @result = @result + 'Total remaining can not be a negative number_'
  if @price < 0
    set @result = @result + 'Price can not be a negative number_'
  if @product_id not like 'PID%' or LEN(@product_id) != 9
    set @result = @result + 'Invalid product id format_'
  if @result = ''
  begin
    if @des is not null
      insert into Product values (
        @product_id,
        @des,
        @name,
        @total_remaining,
        default,
        @price,
        @price,
        @cat,
```

```
        @sid,
        @img)
    else
      insert into Product values (
        @product_id,
        default,
        @name,
        @total_remaining,
        default,
        @price,
        @price,
        @cat,
        @sid,
        @img)
    set @result = 'Successfully adding'
  end
  select @result
end

```

- **Procedure testing**:

```
EXEC insert_product @product_id = 'PIDxxxxxx' , @des='Tai nghe hay'
, @name ='Tai nghe' , @total_remaining=50 , @price=20000 , @cat='CAT000011', @sid= '
    SID100074';

```

- **Result**

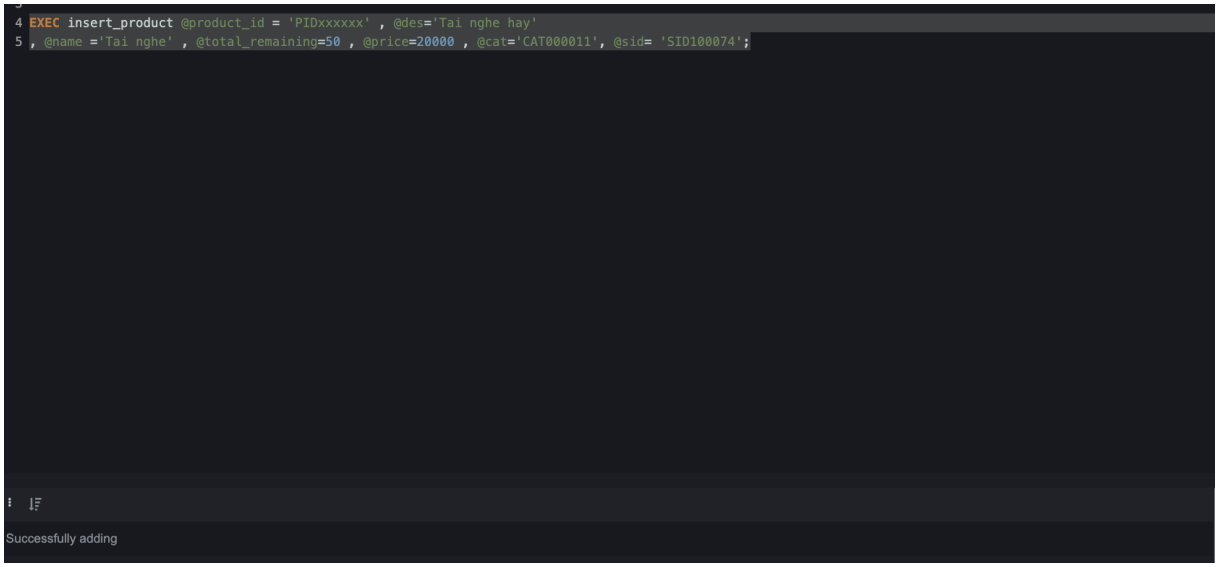

Figure 50: Message after Successfully adding product

product_id	description	name	total_remaining	no_sales	minimum_price	maximum_price	category_id	shop_id	img
PIDxxxxxx	Tai nghe hay	Tai nghe	50	0	20000.0	20000.0	CAT000011	SID100074	NULL

Figure 51: New product in Product table

3.2.2 Delete procedure

- **Stored procedure description**: The stored procedure named `delete_product` is designed to delete a product from the `Product` table. It takes a product ID as a parameter and performs validation checks before deletion. The procedure also outputs a result message indicating the success or failure of the deletion.

  The stored procedure code:

```
create procedure delete_product
@product_id varchar(9),
@result varchar(100) = '' output
as
begin
  if not exists (select * from Product where product_id = @product_id)
    set @result = @result + 'Invalid product name'
  if @result = ''
  begin
    --delete belong and variant fisrt
    delete Belong_to where product_id = @product_id
    delete Variant where product_id = @product_id

    --not exist in is contained (mean is not in order) => delete
    delete Product_instance
    where product_id = @product_id and instance_id not in
    (select pin.instance_id from Is_contained i, Product_instance pin where i.instance_id
     = pin.instance_id);

    --exist in is contained => ref to null product_id pniffffff)
    update Product_instance
    set product_id = 'PIDffffff'
    where product_id = @product_id;

    --update review
    update Review
    set product_id = 'PIDffffff'
    where product_id = @product_id;

    --update Add
    update [Add]
    set product_id = 'PIDffffff'
    where product_id = @product_id

    --delete Product
    delete Product where product_id = @product_id
    set @result = 'Successfully deleting'
  end
  select @result
end

```

- **Procedure testing**:

```
EXEC delete_product @product_id = 'PIDxxxxxx';

```

- **Result**

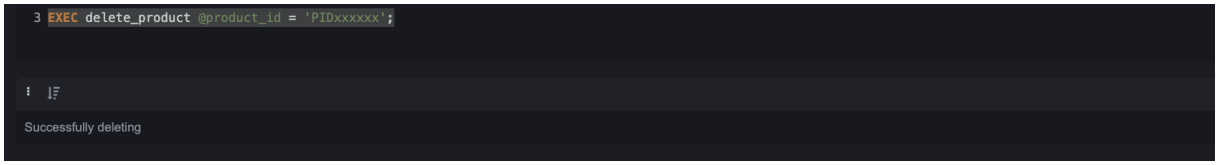

Figure 52: Message after successfully deleting product

product_id	description	name	total_remaining	no_sales	minimum_price	maximum_price	category_id	shop_id	img
PID000001	Stylish and com…	Comfort Shoes	89	50	49001.0	70000.0	CAT000001	SID000001	NULL
PID000002	A classic additio…	Casual Shirt	80	30	19001.0	49001.0	CAT000002	SID000002	NULL
PID000003	High-performan…	Smartphone	150	100	89001.0	199001.0	CAT000003	SID000003	NULL
PID000004	Perfect for cozy…	Candle Holder	50	20	9001.0	24001.0	CAT000004	SID000004	NULL
PID000005	Outdoor advent…	Camping Gear	120	70	49001.0	99001.0	CAT000005	SID000005	NULL
PID000006	Unique and styli…	Necklace	60	40	34001.0	74001.0	CAT000006	SID000001	NULL
PID000007	A modern twist …	Leather Wallet	90	60	24001.0	59001.0	CAT000002	SID000002	NULL
PID000008	Efficient and reli…	Laptop	110	80	19001.0	1449001.0	CAT000003	SID000003	NULL
PID000009	Elegant and fun…	Table Lamp	40	15	44001.0	89001.0	CAT000004	SID000004	NULL
PID000010	Stay hydrated o…	Water Bottle	70	25	9001.0	29001.0	CAT000005	SID000005	NULL
PID000011	Perfect for ever…	Sneakers	90	60	39001.0	79001.0	CAT000001	SID000001	NULL
PID000012	A versatile addit…	Denim Jeans	120	80	29001.0	69001.0	CAT000002	SID000002	NULL
PID000013	Capture every …	Digital Camera	30	15	149001.0	499001.0	CAT000003	SID000003	NULL
PID000014	Create a cozy a…	Throw Blanket	65	35	14001.0	34001.0	CAT000004	SID000004	NULL
PID000015	Gear up for you…	Hiking Boots	100	50	59001.0	119001.0	CAT000005	SID000005	NULL
PID110464	No description	Nuoc xa Comfo	0	0	52000.0	69000.0	CAT000010	SID100052	NULL
PID138894	No description	Bot giat Omo	0	0	14000.0	14000.0	CAT000010	SID100052	NULL
PID144103	No description	Nuoc xa Domy	0	0	47000.0	47000.0	CAT000010	SID100052	NULL
PID163853	No description	Bo beo	0	0	5000.0	20000.0	CAT000011	SID100074	NULL
PID186682	No description	Banh trang	0	0	24000.0	112000.0	CAT000011	SID100074	NULL
PIDffffff	This product na…	Unknown	0	0	0.0	0.0	NULL	SIDffffff	NULL

Figure 53: Product table after deleting

3.2.3 Update procedure

Procedure to update product

- **Stored procedure description**: The stored procedure named `update_product` is designed to update product information. It takes several parameters, including the current product ID (`pid_current`) and the new product ID (`pid_new`). The procedure performs validation checks and updates the product information accordingly. The procedure also outputs a result message indicating the success or failure of the update.

  The stored procedure code:

```
create procedure update_product --when update, get all information from edit panel
@pid_current varchar(9),
@pid_new  varchar(9),
@des varchar(1000),
@name varchar(50),
@total_remaining int,
@min_price  decimal(10,1),
@max_price  decimal(10,1),
@cat varchar(9),
@sid varchar(9),
@img varchar(1000),
@result varchar(1000) = '' Output --default = ''
as
begin
  declare @min_price_cur decimal(10,1) = (select minimum_price from Product where product
    _id = @pid_current)
  declare @max_price_cur decimal(10,1) = (select maximum_price from Product where product
    _id = @pid_current)

  if @pid_current != @pid_new and exists(select * from Product where product_id=@pid_new)
    set @result = @result + 'Product ID has already existed_'
  if @pid_current != @pid_new and (@pid_new not like 'PID%' or len(@pid_new) != 9)
    set @result = @result + 'Invalid product id format_'
  if @total_remaining < 0
    set @result = @result + 'Total remaining can not be a negative number_'
```

```
  if @cat is not null and not exists (select * from Category where category_id = @cat)
    set @result = @result + 'Invalid category_'
  if not exists (select * from Shop where shop_id = @sid)
    set @result = @result + 'Invalid shop_'
  if exists (select * from Variant where product_id = @pid_current) -- has Variant => can
    't modify total remaining or price
  begin
    if @max_price_cur != @max_price or @min_price_cur != @min_price
      set @result = @result + 'This product name has 1 or more Variants,so the price
    cannot be modified here_'
    if @total_remaining != (select total_remaining from Product where product_id = @pid_
    current)
      set @result = @result + 'This product name has 1 or more Variants, so total
    remaining cannot be modified here_'
  end
  if not exists (select * from Variant where product_id = @pid_current) -- no Variant =>
    max = min and >= 0
  begin
    if @min_price != @max_price
      set @result = @result + 'There are not any Variants in this product, so the minimum
     price and maximum price must have the same value_'
    if @min_price <0 or @max_price < 0
      set @result = @result + 'The price can not be a negative number_'
  end

  if @result = ''
  begin
    --update des first
    if(@des is not null)
      update Product set [description] = @des where product_id = @pid_current;
    else
      update Product set [description] = default where product_id = @pid_current;

    update Product
    set
    [name] = @name,
    total_remaining = @total_remaining,
    minimum_price = @min_price,
    maximum_price = @max_price,
    category_id = @cat,
    shop_id = @sid,
    product_id = @pid_new, --update cascade, no need to update on belong, Variant,
    product, review, add
    img = @img
    where product_id = @pid_current;

    set @result = 'Successfully updating'
  end
  select @result
end

```

- **Procedure testing**:

```
EXEC update_product @pid_current = 'PIDffffff', @pid_new='PID221144', @des='Limited
    Pokemon card', @name='Pokemon legend card',
@total_remaining= 1,@min_price = 200000, @max_price=200000, @cat='CAT000004',@sid= '
    SID000004', @img = NULL;

```

- **Result**

product_id	description	name	total_remaining	no_sales	minimum_price	maximum_price	category_id	shop_id	img
PIDffffff	This product na...	Unknown	0	0	0.0	0.0	NULL	SIDffffff	NULL

Figure 54: Before updating

```
EXEC update_product @pid_current = 'PIDffffff', @pid_new='PID221144', @des='Limited Pokemon card', @name='Pokemon legend card',
@total_remaining= 1,@min_price = 200000, @max_price=200000, @cat='CAT000004',@sid= 'SID000004', @img = NULL;

Successfully updating
```

Figure 55: Successfully updating

product_id	description	name	total_remaining	no_sales	minimum_price	maximum_price	category_id	shop_id	img
PID221144	Limited Pokem...	Pokemon legen...	1	0	200000.0	200000.0	CAT000004	SID000004	NULL

Figure 56: After updating

Procedure to filter product

- **Stored procedure description**: The stored procedure named `filter_product` is designed to filter products based on various criteria such as total remaining, number of sales, price range, category, and shop. It returns a result set containing product information that meets the specified criteria.

  The stored procedure code:

```
create procedure filter_product
@total_remaining_min_filter int = -1,
@total_remaining_max_filter int = -1,
@no_sales_min_filter int = -1,
@no_sales_max_filter int = -1,
@min_price_filter decimal(10,1) = -1,
@max_price_filter decimal(10,1) = -1,
@cat_filter varchar(9) = '',
@sid_filter varchar(9) = ''
as
begin
  select
  s.[name],
  c.category_name,
  p.product_id,
  p.[name], p.[description], p.no_sales,
  p.total_remaining,
  p.minimum_price,
  p.maximum_price,
  p.img
  from Product p, Shop s, Category c
  where
    (@total_remaining_min_filter = -1 or p.total_remaining >= @total_remaining_min_filter
    ) and
    (@total_remaining_max_filter = -1 or p.total_remaining <= @total_remaining_max_filter
    ) and
    (@no_sales_min_filter = -1 or p.no_sales >= @no_sales_min_filter) and
    (@no_sales_max_filter = -1 or p.no_sales <= @no_sales_max_filter) and
    (@min_price_filter = -1 or p.minimum_price >= @min_price_filter) and
    (@max_price_filter = -1 or p.maximum_price <= @max_price_filter) and
    (@cat_filter = '' or p.category_id = @cat_filter) and
    (@sid_filter = '' or p.shop_id = @sid_filter) and
    p.shop_id = s.shop_id and p.category_id = c.category_id
  order by p.minimum_price asc;
end
```

- **Procedure testing**:

```
EXEC filter_product @total_remaining_min_filter = 10, @total_remaining_max_filter=100,
    @no_sales_min_filter=10,
@no_sales_max_filter = 100, @min_price_filter =2000, @max_price_filter=100000,@cat_filter
    = 'CAT000001', @sid_filter='SID000001';
```

- **Result**

```
3 EXEC filter_product @total_remaining_min_filter = 10, @total_remaining_max_filter=100, @no_sales_min_filter=10,
4 @no_sales_max_filter = 100, @min_price_filter =2000, @max_price_filter=100000,@cat_filter= 'CAT000001', @sid_filter='SID000001';
5
```

name	category_name	product_id	name	description	no_sales	total_remaining	minimum_price	maximum_price	img
Chic Boutique	Footwear	PID000011	Sneakers	Perfect for ever...	60	90	39001.0	79001.0	NULL
Chic Boutique	Footwear	PID000001	Comfort Shoes	Stylish and com...	50	89	49001.0	70000.0	NULL

Figure 57: After filtering

Procedure to find best-seller

- **Stored procedure description**: The stored procedure named `best_selling_store` is designed to find the best-selling stores based on the minimum number of sales (`no_sales_min`) and the minimum number of products sold (`no_product_min`). It returns a result set containing the shop ID, shop name, and the count of most-selling products for each qualifying store.

  The stored procedure code:

```
create procedure best_selling_store
@no_sales_min int,
@no_product_min int
as
begin
  select s.shop_id, s.[name], count(*) as no_most_selling_product
  from Shop s, Product p
  where s.shop_id = p.shop_id and p.no_sales >= @no_sales_min
  group by s.shop_id, s.[name]
  having count(*) >= @no_product_min
  order by no_most_selling_product desc
end

```

- **Procedure testing**:

```
Exec best_selling_store @no_sales_min=50, @no_product_min= 2;

```

- **Result**

```
2
3 EXEC best_selling_store @no_sales_min=50, @no_product_min= 2;
```

shop_id	name	no_most_selling_product
SID000005	Adventure Out	2
SID000002	Artisan Haven	2
SID000001	Chic Boutique	2
SID000003	Gadget Galaxy	2

Figure 58: Best seller after filtering

Note : this test used the data in the section 2 (not inlucding the additional data of section 3)

3.3 Function

Write 4 functions that meet the following requirements:

- *Contain IF and/or LOOP statements to compute stored data.*
- *Include data query statements, retrieve data from the queries to verify calculations.*
- *Have input parameters and validate the input parameters.*
- *Prepare statements and data to illustrate calling the functions in a report.*

3.3.1 Function to check login

- **Function description**: The function named `check_login_seller` is designed to check seller login credentials. It takes parameters such as username (`user_name`), email (`email`), and password (`pwd`). The function returns the shop ID (`varchar(9)`) if the login is successful; otherwise, it returns 'Deny'.

  The function code:

```
create function check_login_seller(
@user_name varchar(15),
@email  varchar(100),
@pwd  varchar(100)
)
returns varchar(9)
as
begin
  declare @sid varchar(9) = 'Deny'
  if not exists(select * from [User] where ((@user_name is not null and @user_name = user
    _name)
  or (@email is not null and @email = linking_email)) and [password] = @pwd)
    return @sid
  set @sid = (select shop_id from Seller s, [User] u where s.user_id = u.user_id and u.
    user_name = @user_name)
  return @sid
end

```

- **Function testing**:

```
select dbo.check_login_seller('JohnDoe','john.doe@gmail.com', 'securePass123') AS Shop_ID
    ;

```

- **Result**

```
SELECT dbo.check_login_seller('JohnDoe','john.doe@gmail.com', 'securePass123') AS Shop_ID;

Shop_ID
SID000001
```

Figure 59: Successfully login as seller

3.3.2 Function calculate sum_revenue

- **Function description**: The table-valued function named `sum_revenue` is designed to calculate the total revenue for each product in a given shop within a specified date range. It takes parameters such as shop ID (`shopID`), start date (`startDate`), and end date (`endDate`). The function returns a table with columns including product ID, product name, and total revenue.

  The function code:

```
create function sum_revenue(@shopID varchar(9),@startDate date,@endDate date)
returns @res table(product_id varchar(9), [name] varchar(50), total_revenue decimal(11,1)
    )
as
begin
  if @startDate > @endDate or not exists (select * from Shop where shop_id = @shopID)
    insert into @res values('PIDffffff','No product', 0)
  else
  begin
  declare @listProduct table (product_id varchar(9), [name] varchar(50))
  insert into @listProduct select product_id, [name] from Product where shop_id = @shopID
  declare @product_id varchar(9)
  declare @name varchar(50)

  declare cur Cursor for select * from @listProduct
```

```
15
16    open cur
17    fetch next from cur into @product_id,@name
18
19    while @@FETCH_STATUS = 0
20    begin
21      declare @total_revenue decimal(11,1) =
22      (select sum(pin.current_price)
23      from Product_instance pin, Is_contained i, [Order] o
24      where pin.instance_id = i.instance_id and o.order_id = i.order_id and o.status = '
        Done' and pin.product_id = @product_id
25      and convert(date,o.time_order) between @startDate and @endDate)
26
27      if @total_revenue is null
28        set @total_revenue = 0
29      insert into @res  values(@product_id,@name,@total_revenue)
30
31      fetch next from cur into @product_id,@name
32    end
33
34    close cur
35    deallocate cur
36    end
37    return
38  end
39
```

- **Function testing**:

```
1  select * from  dbo.sum_revenue('SID000001', '2023-01-01', '2023-12-31');
2
```

- **Result**

product_id	name	total_revenue
PID000001	Comfort Shoes	0.0
PID000006	Necklace	74001.0
PID000011	Sneakers	0.0

Figure 60: Product, name and revenue of a shop

3.3.3 Function to list order

- **Function description**: The table-valued function named `list_order` is designed to retrieve a summary of order status counts for a given buyer within a specified date range. It takes parameters such as buyer ID (`buyerID`), start date (`startDate`), and end date (`endDate`). The function returns a table with columns including order status and the corresponding count.

  The function code:

```
1  create function list_order(@buyerID varchar(9),@startDate date,@endDate date)
2  returns @res table (Confirming int, Waiting_pickup int, Delivering int, Done int,
       Cancelled int, Refund int) as
3  begin
4    if(@startDate > @endDate) and not exists (select user_id from Buyer where user_id =
       @buyerID)
5      insert into @res values (null,null,null,null,null,null)
6    else
7    begin
8      insert into @res values (0,0,0,0,0,0)
9      declare @listOrder table (order_id varchar(14), [status] varchar(15))
10     insert into @listOrder
11       select o.order_id, o.[status]
12       from [Order] o, Places p
```

```
      where o.order_id = p.order_id and (p.user_id = @buyerID or p.user_id_cart =
    @buyerID)
      and CONVERT(date,o.time_order) between @startDate and @endDate

    declare cur cursor for select [status] from @listOrder
    declare @status varchar(15)

    open cur
    fetch next from cur into @status

    while @@FETCH_STATUS = 0
    begin
      if @status = 'Done'
        update @res set Done = Done + 1
      else if @status = 'Confirming'
        update @res set Confirming = Confirming + 1
      else if @status = 'Waiting pickup'
        update @res set Waiting_pickup = Waiting_pickup + 1
      else if @status = 'Delivering'
        update @res set Delivering = Delivering + 1
      else if @status = 'Cancelled'
        update @res set Cancelled = Cancelled + 1
      else if @status = 'Refund'
        update @res set Refund = Refund + 1

      fetch next from cur into @status
    end
    close cur
    deallocate cur
  end
  return
end

```

- **Function testing**:

```
select * from dbo.list_order('UID000001','2023-01-01', '2023-12-31');

```

```
select * from dbo.list_order('UID000001','2022-01-01','2023-12-13')

select * from dbo.sum_revenue('SID123456','2022-01-01','2023-12-13')
```

101 %

Results | Messages

	Confirming	Waiting_pickup	Delivering	Done	Cancelled	Refund
1	3	2	2	2	2	2

Figure 61: Listing order from buyer

3.3.4 Other function/Procedure

- generate_PID() : use to generate new ID for Product that has not existed yet in database

```
create function generate_PID() -- use in application
returns varchar(9)
as
begin
  if not exists (select * from Product)
    return 'PID000000'

  declare @newPID varchar(9)
  declare @listPID table (postfix int)

```

```
  insert into @listPID
  select Convert(int,REPLACE(p.product_id,'PID','1')) as pid_trans
  from Product p
  where p.product_id != 'PIDffffff'
  order by pid_trans asc

  declare cur Cursor for (select * from @listPID)
  declare @pre int
  declare @current int

  open cur
  fetch next from cur into @pre

  if @pre > 1000000 return 'PID000000'

  fetch next from cur into @current

  while @@FETCH_STATUS = 0
  begin
    if @current != (@pre + 1)
      break

    set @pre = @current
    fetch next from cur into @current
  end

  if @pre = 1999999
  begin
    return null
  end
  set @newPID = 'PID' + substring(convert(varchar(7),@pre + 1),2,6)
  return @newPID
end

```

test :

```
declare @newPID varchar(9) = (select dbo.generate_PID())
if @newPID not in (select product_id from Product)
    print @newPID + ' is new product_id'
```

```
.01 %
Messages
PID000026 is new product_id
```

Figure 62: test generate new Product_id

- generate_IID() : to generate new instance_id that has not existed yet in database

```
create function generate_IID()
returns varchar(9)
as
begin
  if not exists (select * from Product_instance)
    return 'IID000000'

  declare @newIID varchar(9)
  declare @listIID table (postfix int)

  insert into @listIID
  select Convert(int,REPLACE(p.instance_id,'IID','1')) as iid_trans
  from Product_instance p
  order by iid_trans asc

  declare cur Cursor for (select * from @listIID)
  declare @pre int
```

```
  declare @current int

  open cur
  fetch next from cur into @pre

  if @pre > 1000000 return 'IID000000'

  fetch next from cur into @current

  while @@FETCH_STATUS = 0
  begin
    if @current != (@pre + 1)
      break

    set @pre = @current
    fetch next from cur into @current
  end

  if @pre = 1999999
  begin
    return null
  end
  set @newIID = 'IID' + substring(convert(varchar(7),@pre + 1),2,6)
  return @newIID
end


```

test :

```
declare @newIID varchar(9) = (select dbo.generate_IID())
if @newIID not in (select instance_id from Product_instance)
    print @newIID + ' is new instance_id'
```

```
%
Messages
IID000139 is new instance_id

Completion time: 2023-12-14T23:35:09.7442986+07:00
```

Figure 63: Test generate Instance_id

- add_instance(@no_add int,@product_id, varchar(9),@price decimal(10,1)) : to add muiltiple instance base on product_id

```
create procedure add_instance
@no_add int,
@product_id varchar(9),
@price decimal(10,1)
as
begin
  while @no_add > 0
  begin
    declare @instance_id varchar(9) = (select dbo.generate_IID() as newIID)
    insert into Product_instance values(@instance_id,@price,@product_id)
    set @no_add = @no_add -1
  end
end

```

- no_instance_on_sale(@product_id varchar(9)) : return the number of instance that has not been sold yet (mean not in Is_contained table)

```
create function no_instance_on_sale
```

```
(
@product_id varchar(9)
)
returns int
as
begin
  declare @rs int
  set @rs = (select count(*) as num from Product_instance where product_id = @product_id
    and instance_id not in(select instance_id from Is_contained))
  return @rs
end

```

test

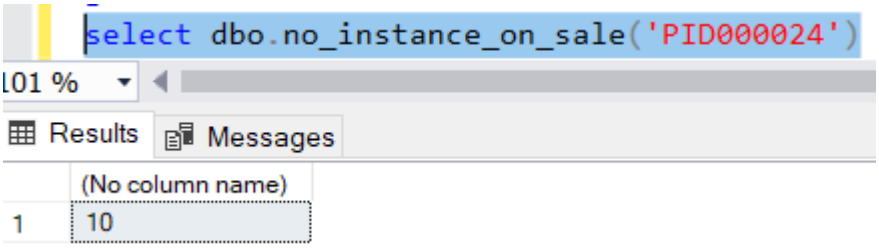

Figure 64: Test function

4 Program (Winform) illustrating the connection of the application to the database

The result involves integrating the database into a simple application developed by our team. The application is implemented using Winform written by the C programming language and utilizes the SQL Server Management Studio. The team's application is collectively developed by the team members, and the individual responsibilities are outlined in the task assignment section.

4.1 Login

In order to get access to all the features of the application, users (sellers) are required to proceed with the login process.

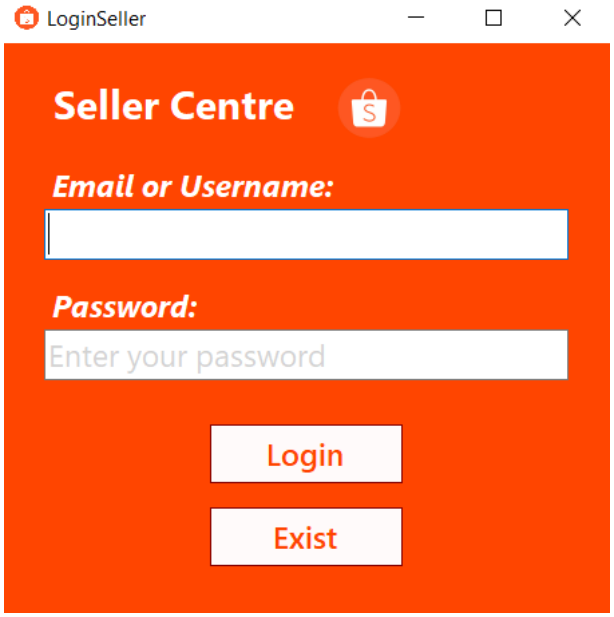

Figure 65: Login page

4.2 Shop's information page

After users have successfully logged into the website, they will see the first page of the Shop's information, displaying information about all Shops and features in the left bar.

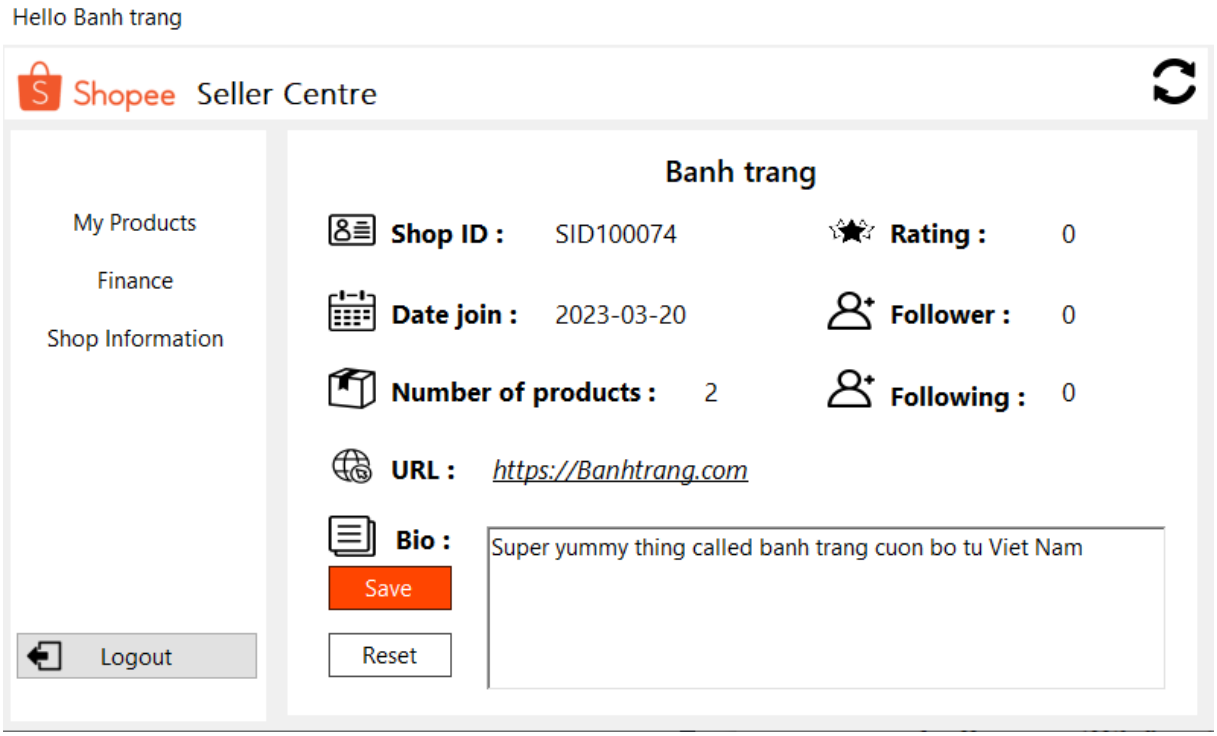

Figure 66: Shop's information page

4.3 Shop's finance page

Shop's finance page is where the sellers can control the revenues with the UI like following:

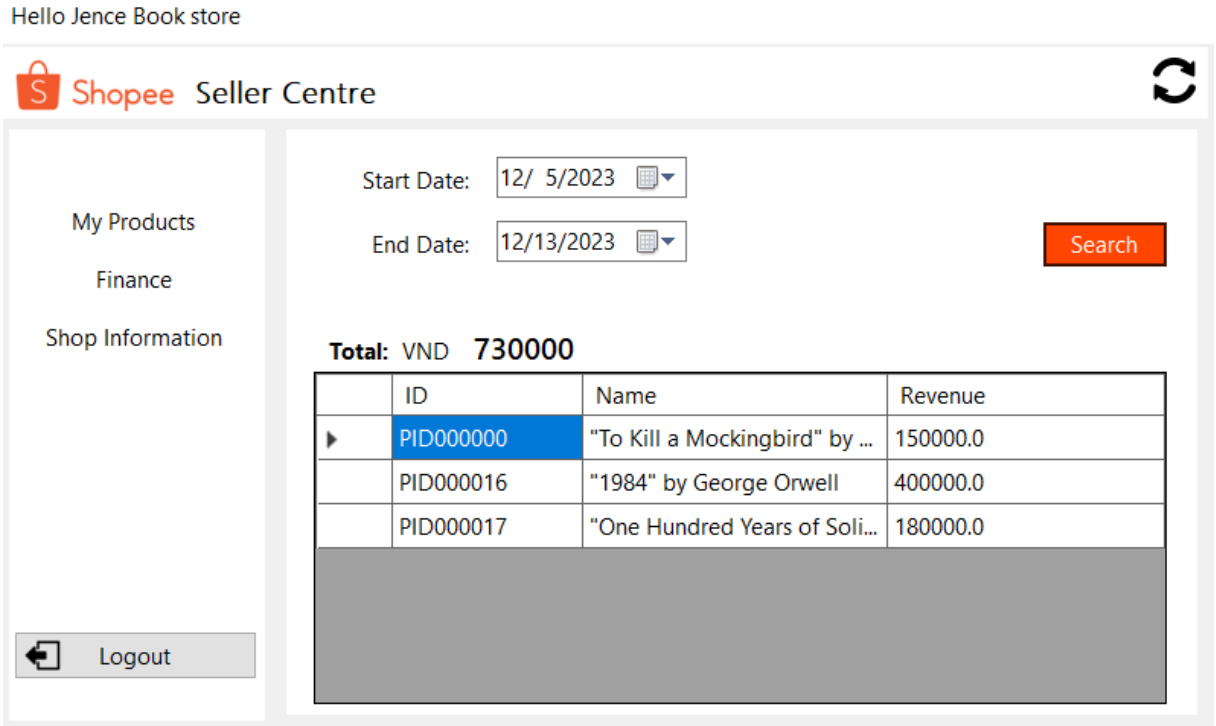

Figure 67: Shop's finance page

4.4 Shop's product page

Shop's product page is where sellers can control their product. In the first view, they can get the general information of their product with the small table showing all information.

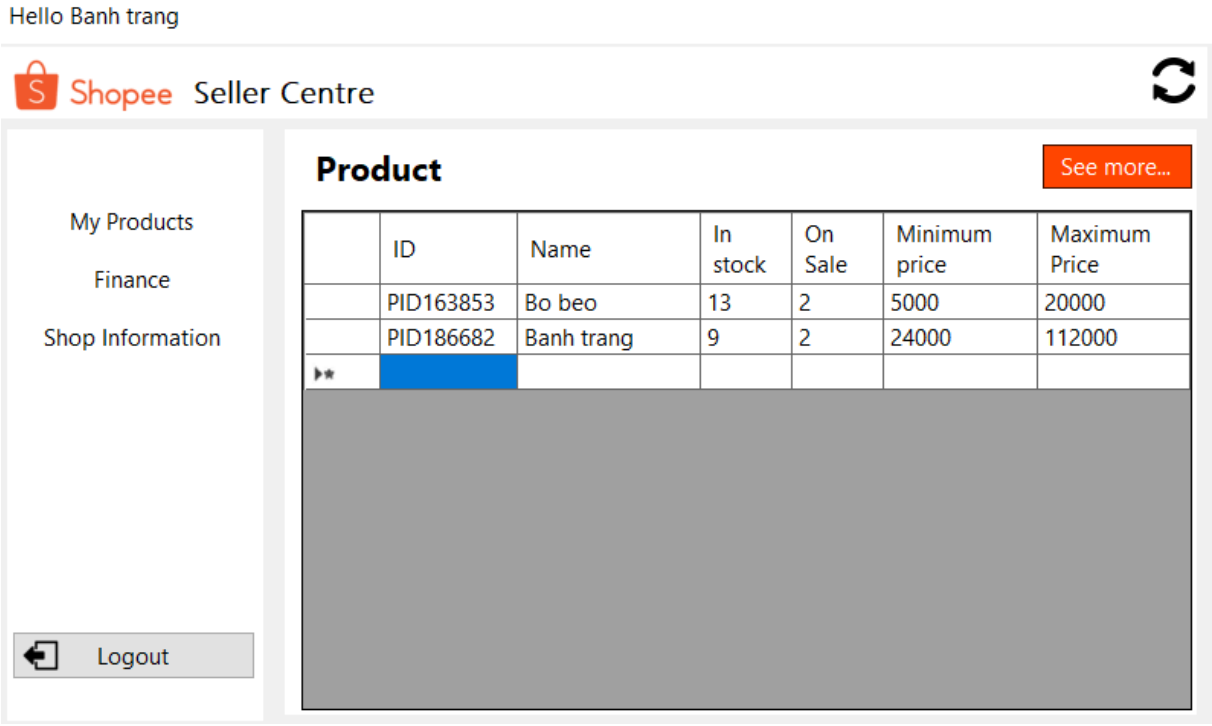

Figure 68: Shop's product page

After clicking `See more ...` button, sellers can get more specific information about their product as well as control them.

Here, sellers can add new product with `Add Product ...` button.

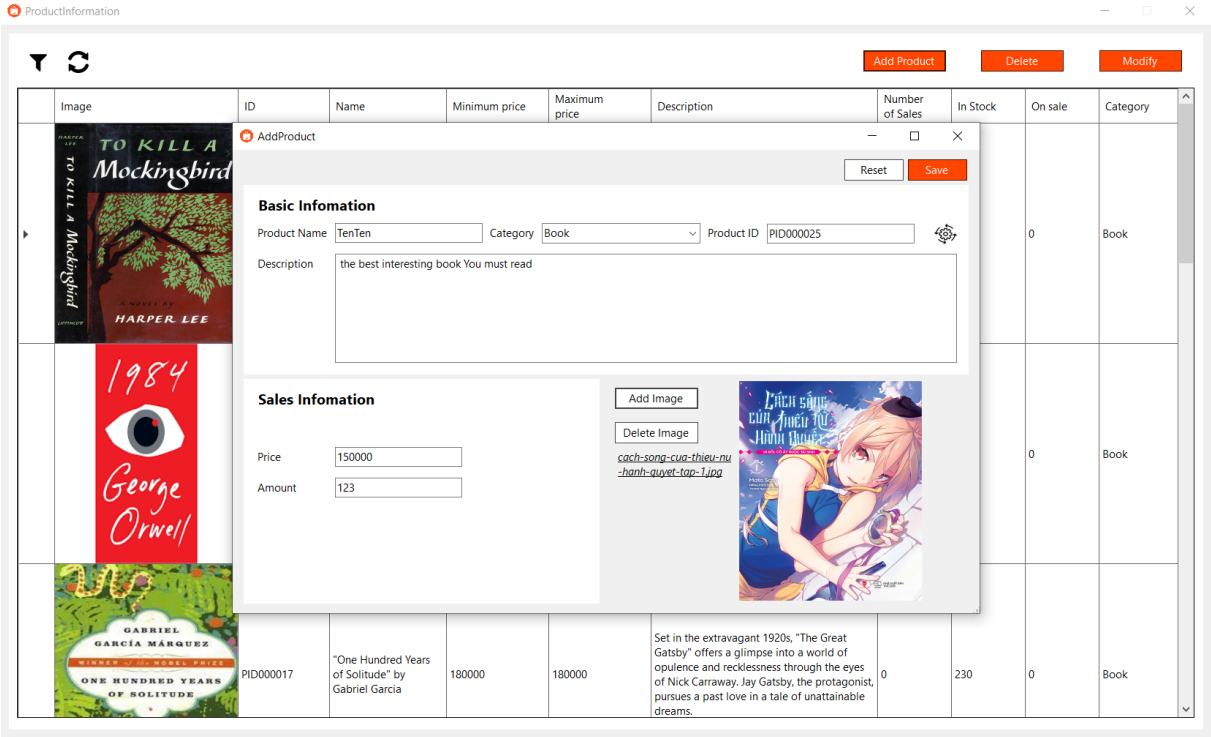

Figure 69: Add product view

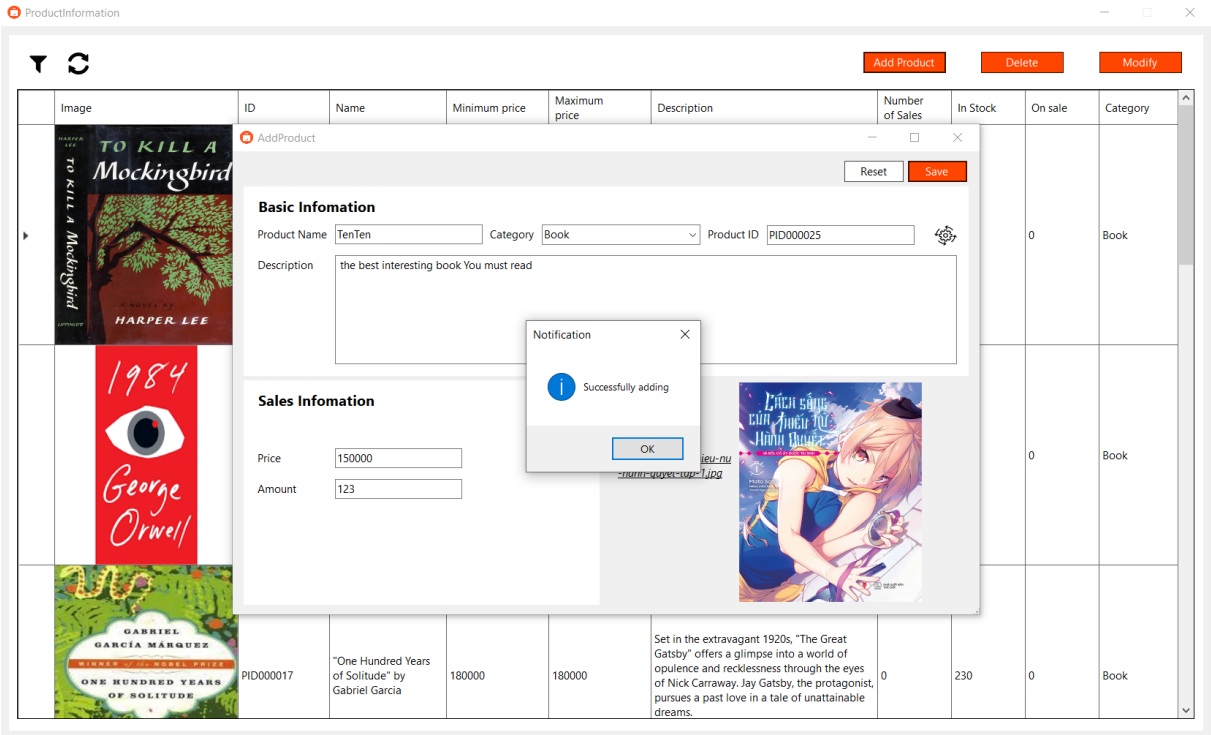

Figure 70: Successfully adding product

The sellers can also modify the information of their product with `Modify` button.

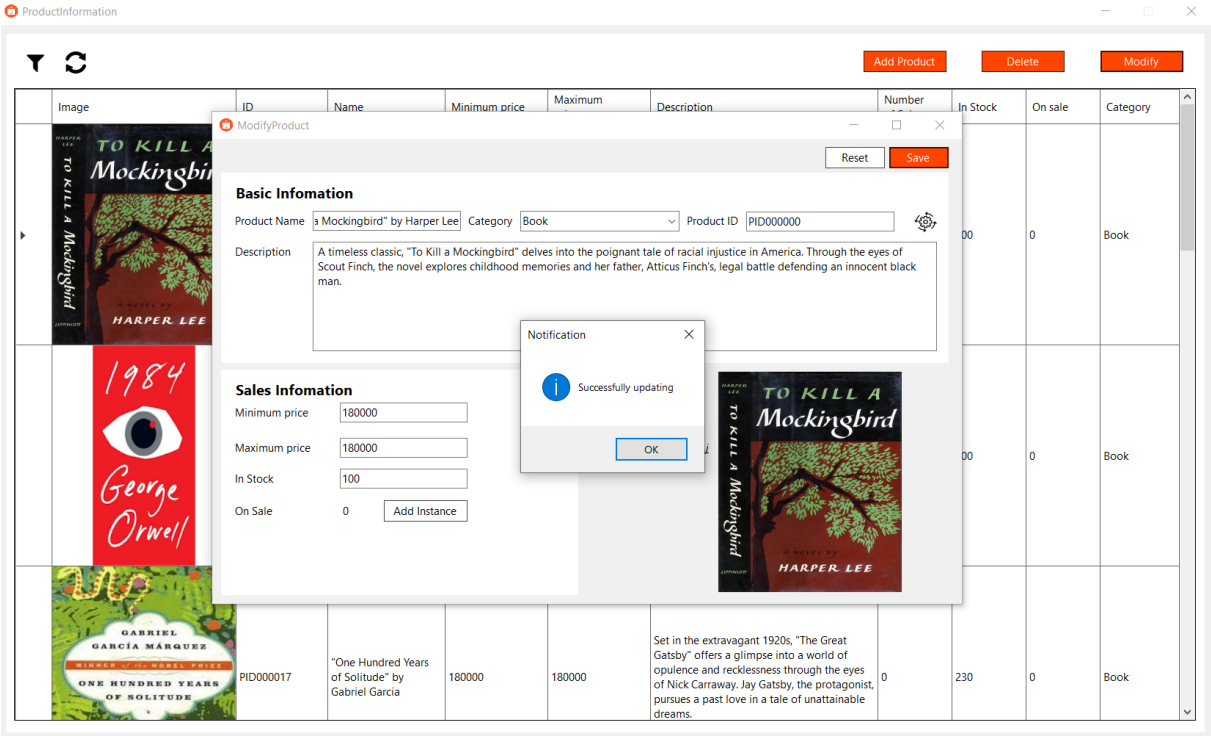

Figure 71: Successfully modified

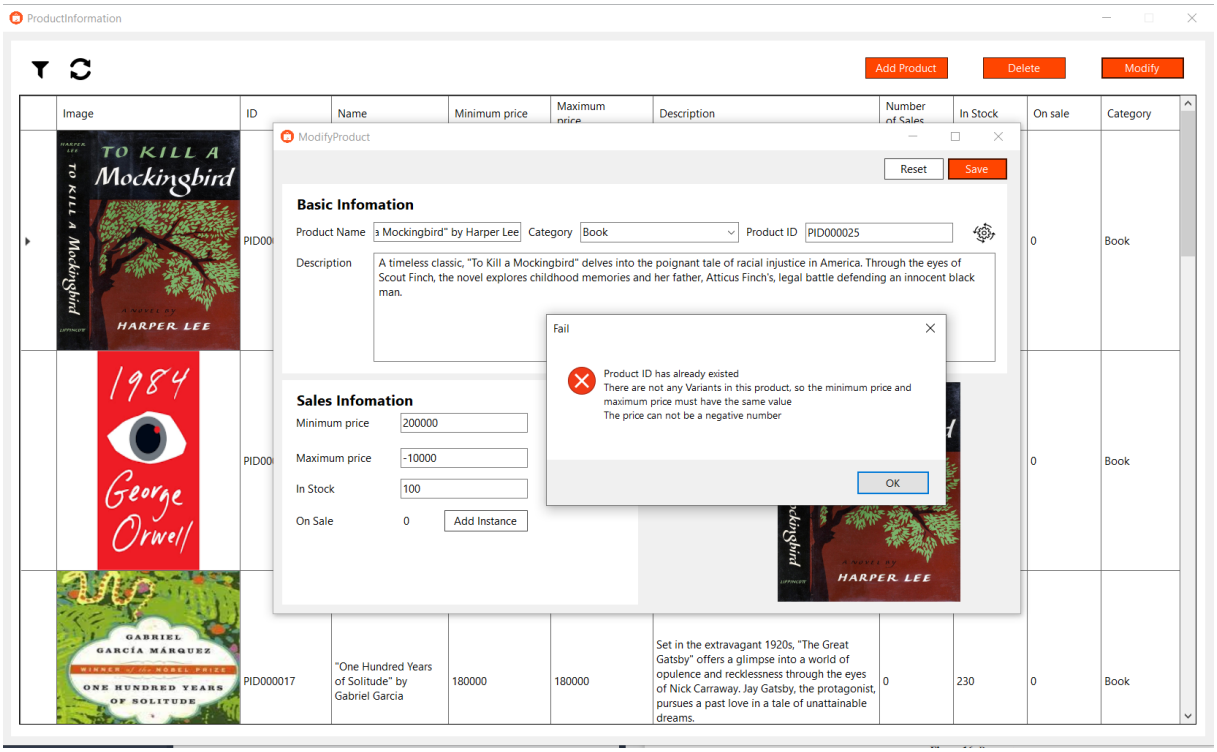

Figure 72: Error occurs when modifying

They can also delete product.

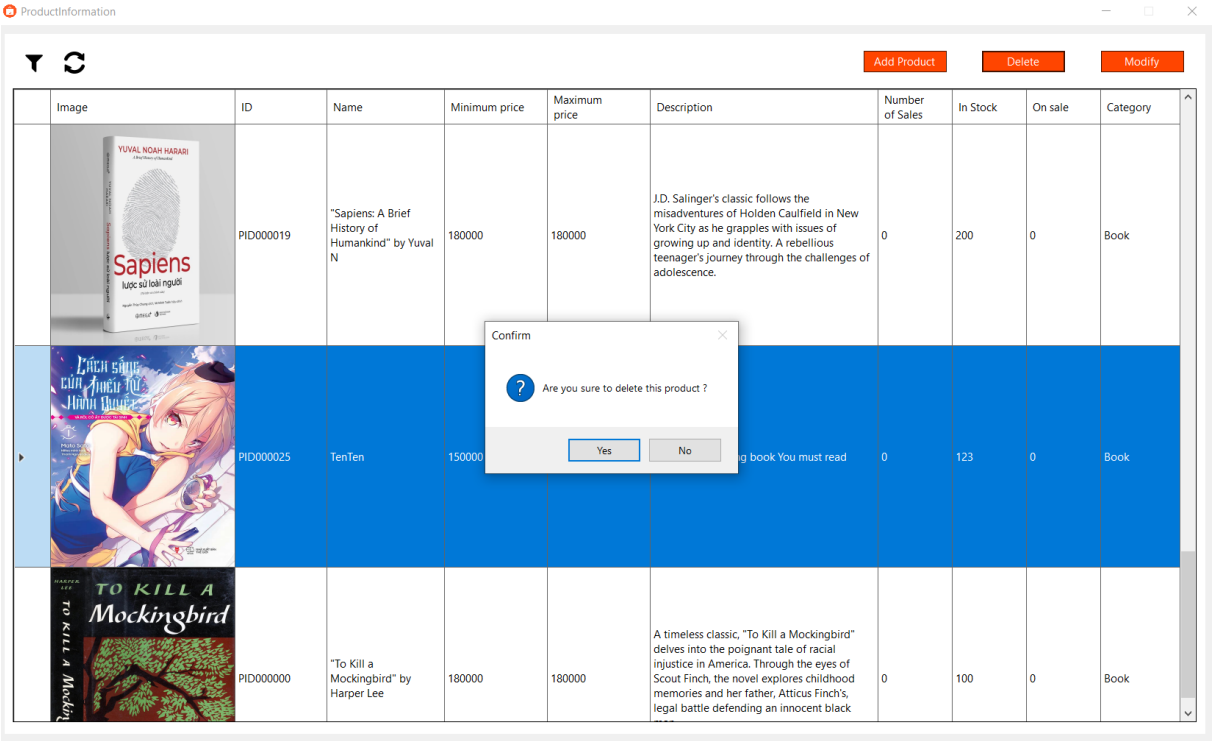

Figure 73: Confirm deletion

For the management purpose, sellers can filter their product.

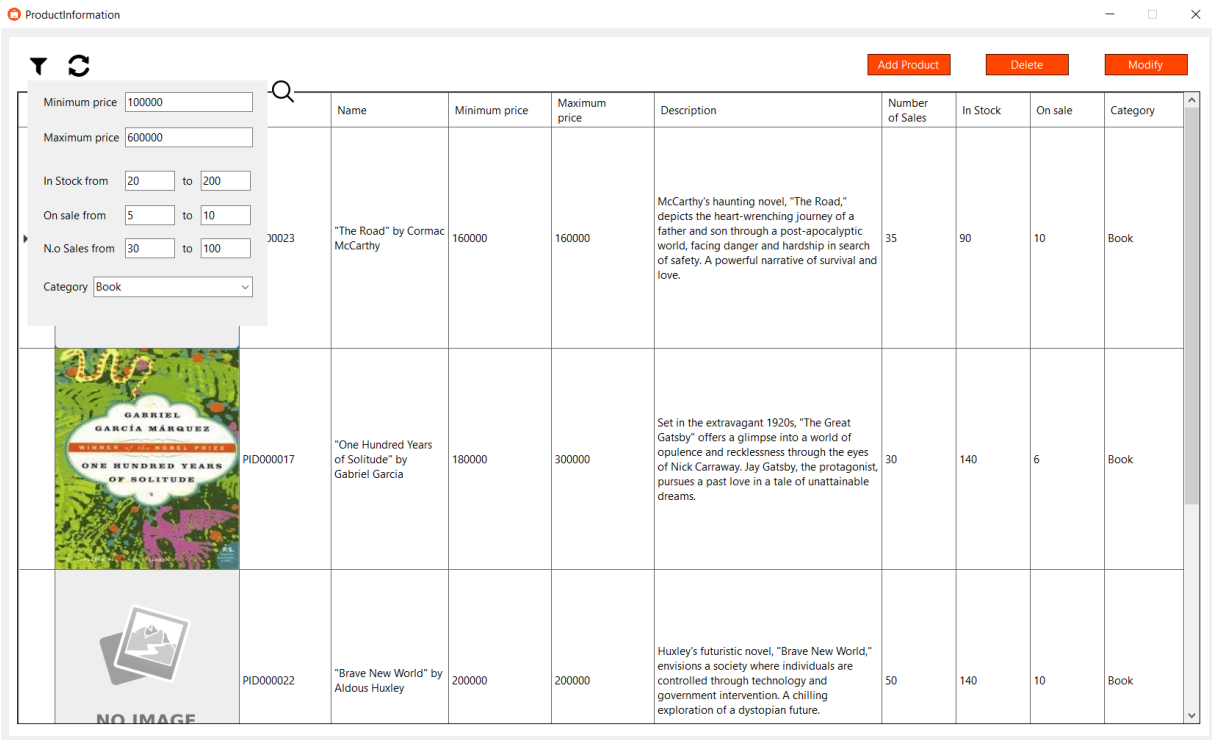

Figure 74: Filtering the product

5 Updating ERD and Mapping

In this assignment, we decided to modify some of our database design.

- Changing the table's name of `Product` to `Product_instance`.
- Changing the tables' name of `Product_name` to `Product`.
- Changing the relationship between `Add` and `Product`.

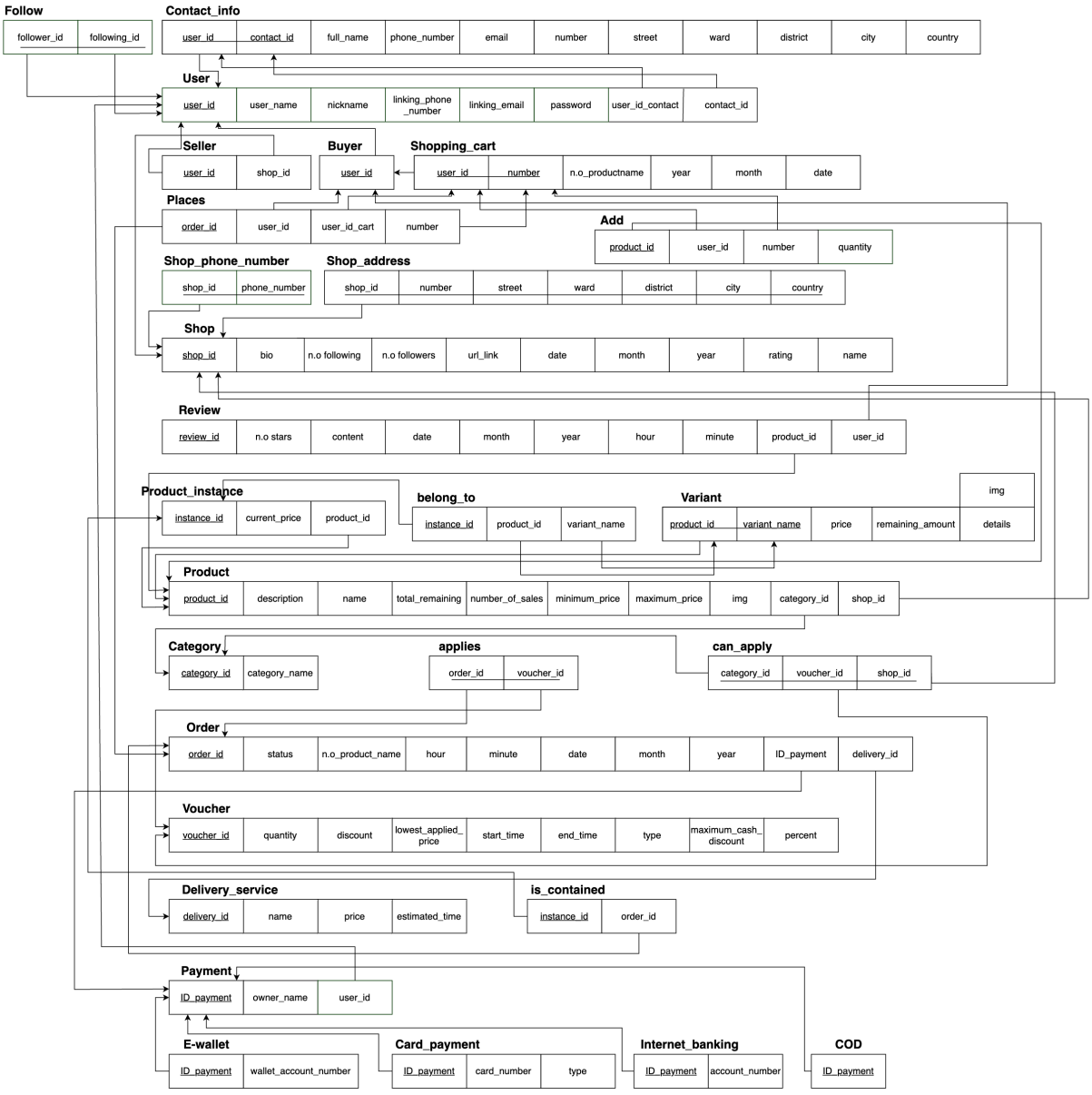

Figure 75: Mapping

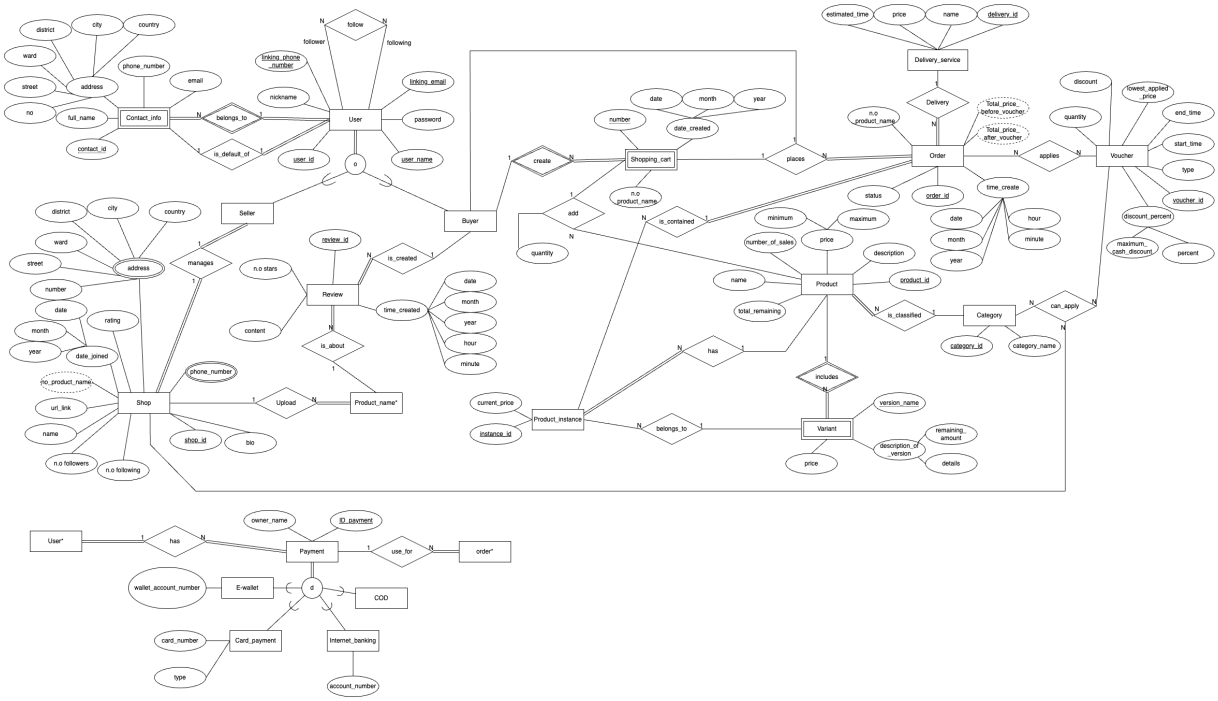

Figure 76: New ERD