Database System Project 2

Electronic Vendor Company

20201564 김성현

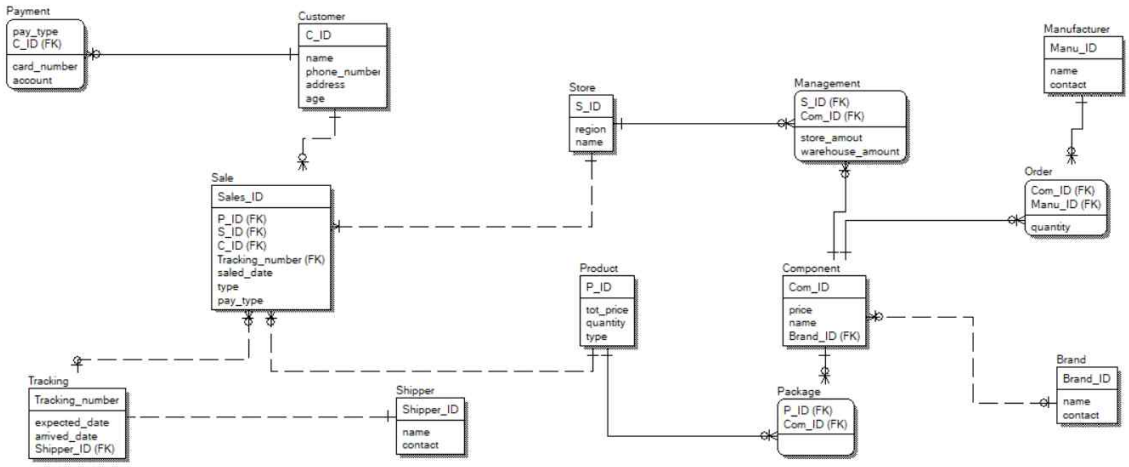
목차

문제 내용

지난 프로젝트에서 구현한 electronic vendor company의 데이터베이스를 배운 내용을 토대로 보다 좋은 design으로 변형하고, BCNF 정규화를 통해 decompose한다. 그 다음 데이터 타입, 도메인, 제약 조건을 추가하여 logical diagram을 physical diagram으로 변환한다. 마지막으로 ODBC를 이용해 MySQL과 C코드를 구현하여 제시된 쿼리문을 수행하는 프로그램을 구현한다.

Relation Schema Diagram 수정 + BCNF Decompose

BCNF 정규화를 하기에 앞서, 기존의 Diagram에서 불필요한 정보의 중복이 많이 발생하는 것을 줄이기 위해 다음과 같이 수정하였다.



변경 내용은 다음과 같다.

1. Pay, Deal, Shipping, Inven_Brand에 대한 Relation Table 삭제 (정보의 중복이 다량 발생)
2. Sales_Data, Deal, Sales Table 통폐합 (굳이 여러 Table에 정보를 분산하여 저장하지 않았다.)
3. Inventory Relation 이름 변경 (인벤토리에 대한 내용은 Management에 저장되어 있으므로 적절하지 않다고 판단, 부품, 구성요소의 뜻을 담고 있는 Component로 변경)

또한 이 Relation들은 전부 BCNF임이 확인된다. 이하는 그 과정을 보여준다.

Payment

Pay_type, C_ID->card_number, account로 단 하나 존재하고, pay_type, C_ID는 super key이므로 BCNF이다.

Customer

C_ID->name, phone_number, address, age로 단 하나 존재하고, C_ID가 super key이므로 BCNF이다.

Sales
Sales_ID -> C_ID, P_ID, S_ID, Tracking_number, saled_date, saled_type, pay_type이고, Sales_ID가 super key이므로 BCNF이다.
C_ID, P_ID, S_ID->Tracking_number의 FD는 존재하지 않는데, 이는 하나의 고객이 한 상품을 한 매장에서 여러 번 구매할 수 있기 때문이다.

Tracking
Traking_number -> Shipper_ID, expected_date, arrived_date의 하나이고, Tracking_number가 Super key이므로 BCNF이다.

Shipper
Shipper_ID -> name, contact의 하나이고 Shipper_ID가 super key이므로 BCNF이다.

Store
Store_ID -> name, region의 하나이고 Store_ID가 super key이므로 BCNF이다.

Product
P_ID -> tot_price, quantity, type의 하나이고 P_ID가 super key이므로 BCNF이다.

Management
S_ID, Com_ID -> store_amount, warehouse_amount의 하나이고 S_ID, Com_ID가 super key이므로 BCNF이다.

Component
Com_ID -> Brand_ID, price, name의 하나이고 Com_ID가 super key이므로 BCNF이다.

Package
Com_ID, P_ID -> Com_ID, P_ID의 trivial한 FD 하나만 존재하므로 BCNF이다.

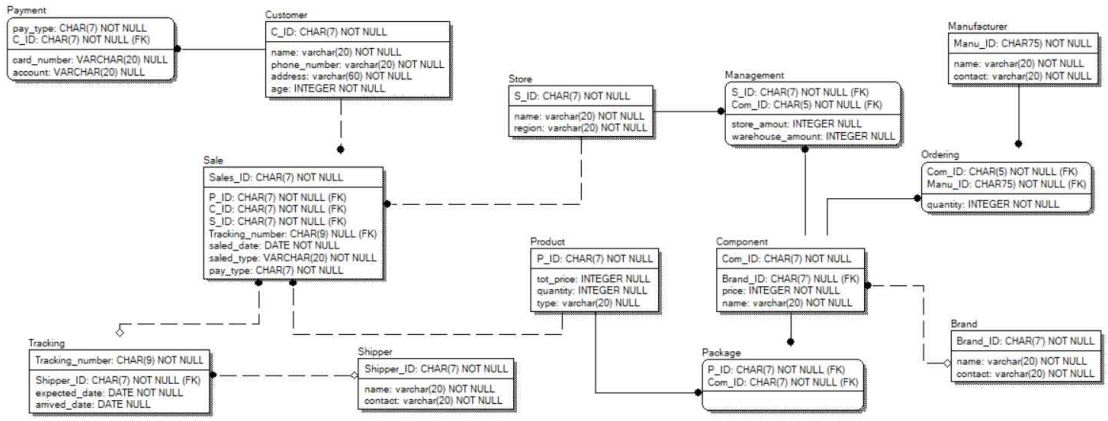
Manufacturer
Manu_ID -> name, contact의 하나이고 Manu_ID가 super key이므로 BCNF이다.

Ordering
Manu_ID, Com_ID -> quantity의 하나이고 Manu_ID, Com_ID가 super key이므로 BCNF이다.

Brand
Brand_ID -> name, contact의 하나이고 Brand_ID가 super key이므로 BCNF이다.

Physical Schema Diagram으로 변환

다음은 앞의 Schema Diagram을 Physical Schema Diagram으로 변환한 것이다.



대부분의 값은 char, varchar이고, PK의 경우 7자리의 char로 구분한다. 예외적으로 Tracking_number만 9자리로 구분한다. 구분의 편의를 위해, customer의 PK는 앞자리가 0, Store의 PK는 앞자리가 10~14, Brand의 PK는 앞자리가 15~19, Component의 PK는 앞자리가 20~24, Manufacturer의 PK는 앞자리가 25~29, Product의 PK는 앞자리가 3, Shipper의 PK는 앞자리가 40~44, Sale의 PK는 앞자리가 7~9로 고정하였다. 금액, 개수, 양은 모두 integer 속성을 부여하였다. 날짜는 DATE의 속성을 부여하였다. 대부분의 값은 NULL이 허용되지 않고, Management, Payment와 Tracking의 일부 attribute만 NULL이 허용된다.
또, Tracking_number가 ZZZZ99999는 offline상에서 거래가 이루어진 경우이고, Shipper_ID가 4499999인 경우는 offline상에서 거래가 이루어진 것의 가상의 운송자 ID이다. (offline과 online상의 매출 비교를 원활하게 해준다.)

Create table 및 insert tuple 과정

테이블을 생성하는 create, tuble을 삽입하는 insert를 위에 설명한 조건대로 구현한 코드는 다음과 같다. insert까지 모두 포함하면 지나치게 길어지기 때문에, 보고서에는 각 Table에서 대표로 하나씩만 첨부하였다.

```
create table Customer
(C_ID    char(7) not null,
name    varchar(20) not null,
phone_number  varchar(20) not null,
address varchar(20) not null,
age       integer not null,
primary key(C_ID),
check(C_ID >='0000000' and C_ID <'1000000'));

create table Store
(S_ID    char(7) not null,
```

```
name varchar(20) not null,
region varchar(20) not null,
primary key(S_ID),
check(S_ID >='1000000' and S_ID <'1500000'));

create table Brand
(Brand_ID                char(7) not null,
name varchar(20) not null,
contact varchar(20) not null,
primary key(Brand_ID),
check(Brand_ID>='1500000' and Brand_ID<'2000000'));

create table Component
(Com_ID          char(7) not null,
Brand_ID         char(7) not null,
foreign key (Brand_ID) references Brand(Brand_ID) on delete cascade,
price integer not null,
name varchar(20) not null,
primary key(Com_ID),
check(Com_ID>='2000000' and Com_ID<'2500000'));


create table Manufacturer
(Manu_ID  char(7) not null,
name varchar(20) not null,
contact varchar(20) not null,
primary key(Manu_ID),
check(Manu_ID>='2500000' and Manu_ID<'3000000'));

create table Ordering
(Com_ID char(7) not null,
Manu_ID  char(7) not null,
quantity integer not null,
primary key (Com_ID,Manu_ID),
foreign key (Com_ID) references Component(Com_ID) on delete cascade,
foreign key (Manu_ID) references Manufacturer(Manu_ID) on delete cascade);

create table Management
(S_ID char(7) not null,
Com_ID char(7) not null,
store_amount integer,
```

```
warehouse_amount integer,
primary key (S_ID,Com_ID),
foreign key (S_ID) references Store(S_ID) on delete cascade,
foreign key (Com_ID) references Component(Com_ID) on delete cascade);

create table Product
(P_ID char(7) not null,
tot_price integer not null,
quantity integer not null,
type varchar(20) not null,
primary key (P_ID),
check(P_ID>='3000000' and P_ID<'4000000'));

create table Package
(P_ID char(7) not null,
Com_ID char(7) not null,
primary key(P_ID,Com_ID),
foreign key (P_ID) references Product(P_ID) on delete cascade,
foreign key (Com_ID) references Component(Com_ID) on delete cascade);

create table Payment
(pay_type char(7) not null,
C_ID char(7) not null,
card_number varchar(20),
account varchar(20),
primary key (pay_type,C_ID),
foreign key (C_ID) references Customer(C_ID) on delete cascade);

create table Shipper
(Shipper_ID char(7) not null,
name varchar(20) not null,
contact varchar(20) not null,
primary key(Shipper_ID),
check(Shipper_ID>='4000000' and Shipper_ID<'4500000'));

create table Tracking
(Tracking_number char(9) not null,
Shipper_ID char(7) not null,
expected_date date not null,
arrived_date date,
primary key(Tracking_number),
```

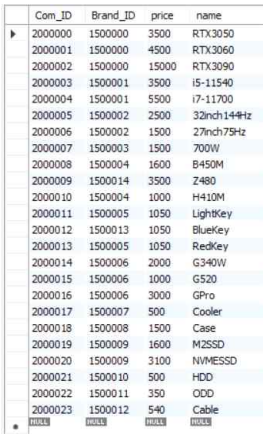
```
foreign key (Shipper_ID) references Shipper(Shipper_ID) on delete cascade);

create table Sale
(Sales_ID char(7) not null,
P_ID char(7) not null,
C_ID char(7) not null,
S_ID char(7) not null,
Tracking_number char(9) not null,
saled_date date not null,
saled_type varchar(20) not null,
pay_type char(7) not null,
primary key(Sales_ID),
check (Sales_ID>='7000000' and Sales_ID<='9999999'),
foreign key (C_ID) references Customer(C_ID) on delete cascade,
foreign key (P_ID) references Product(P_ID) on delete cascade,
foreign key (S_ID) references Store(S_ID) on delete cascade,
foreign key (Tracking_number) references Tracking(Tracking_number));

insert into Customer values('0000001','Kim','010-1111-3333','Gyeongi Gwangmyeong','22');
insert into Store values('1000001','Yonex','California');
insert into Brand values('1500011','Nexon','031-333-5565');
insert into Component values('2000002','1500000','15000','RTX3090');
insert into Manufacturer values('2500007','Yongsan','1521-6666');
insert into Ordering values('2000008','2500007','10');
insert into Management values('1000005','2000010','0','0');
insert into Product values('3000002','30000','5','Graphic');
insert into Package values('3000008','2000013');
insert into Payment values('online','0000005','12395678',null);
insert into Shipper values('4000007','USPS','111-2122');
insert into Tracking values('ABCH12345','4000007','20220306',null);
insert into Sale values('7000002','3000001','0000002','1000000','ABCH12345','20210301','online','online');
```

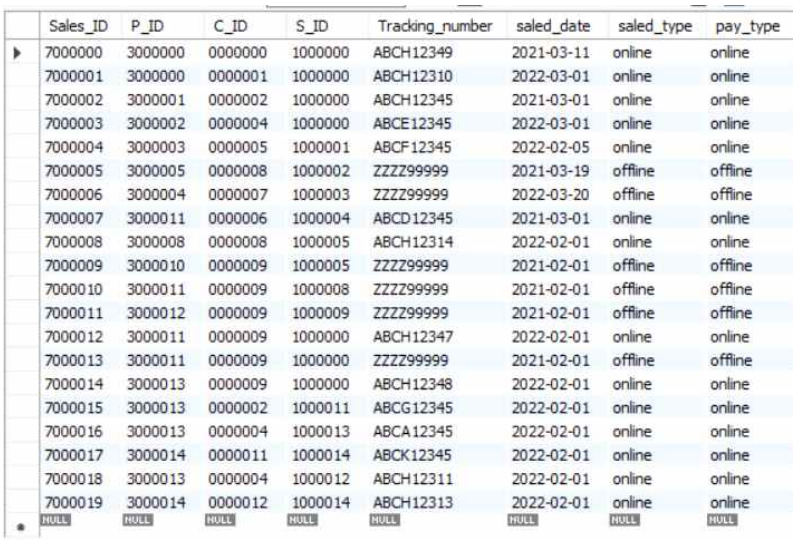
insert가 모두 마무리되고 MySQL상에 insert된 데이터를 확인하면 다음과 같다.

Brand_ID	name	contact
1500000	Hyundai	02-460-3819
1500001	Kia	02-470-5869
1500002	Samsung	031-560-5719
1500003	LG	051-460-3819
1500004	Kakao	031-679-5119
1500005	Naver	02-750-0619
1500006	Line	031-560-6869
1500007	Coupang	02-680-9567
1500008	Uahan	02-466-7777
1500009	Yogiyo	02-012-3465
1500010	SK	031-512-3665
1500011	NC	031-666-7465
1500012	Netmarble	031-976-7565
1500013	Smilegate	031-663-9654
1500014	Sinsegye	02-865-9545

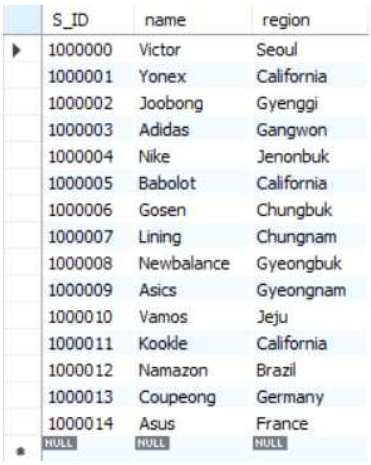


Com_ID	Brand_ID	price	name
2000000	1500000	3500	RTX3050
2000001	1500000	4500	RTX3060
2000002	1500000	15000	RTX3090
2000003	1500001	3500	i5-11540
2000004	1500001	5500	i7-11700
2000005	1500002	2500	32inch144Hz
2000006	1500002	1500	27inch75Hz
2000007	1500003	1500	700W
2000008	1500004	1600	B450M
2000009	1500014	3500	Z480
2000010	1500004	1000	H410M
2000011	1500005	1050	LightKey
2000012	1500013	1050	BlueKey
2000013	1500005	1050	RedKey
2000014	1500006	2000	G340W
2000015	1500006	1000	G520
2000016	1500006	3000	GPro
2000017	1500007	500	Cooler
2000018	1500008	1500	Case
2000019	1500009	1600	M2SSD
2000020	1500009	3100	NVMESSD
2000021	1500010	500	HDD
2000022	1500011	350	ODD
2000023	1500012	540	Cable

C_ID	name	phone_number	address	age
0000000	Jung	010-1111-2222	Seoul Mapo	57
0000001	Kim	010-1111-3333	Gyeongi Gwangmyeong	22
0000002	Chang	010-1111-4444	Seoul Gangdong	53
0000003	Nang	010-5555-2222	Seoul Songpa	55
0000004	Lim	010-6666-2222	Seoul Eunpyeong	60
0000005	Lee	010-6666-3333	Seoul Gangbuk	32
0000006	Choi	010-6666-4443	Gyeongi Bundang	34
0000007	Yang	010-6656-3333	Gangwon Wonju	49
0000008	Seo	010-7666-3333	Busan Haewondaegu	66
0000009	Park	010-6666-3443	Gyeongbuk Munkyung	63
0000010	Oh	010-6866-3333	Incheon Gyeyanggu	68
0000011	Koo	010-8666-6433	Seoul Gangnam	57
0000012	Min	010-9666-3333	Chungbuk Jeungpyeong	22
0000013	Kang	010-9766-3483	Jeju Jeju	29
0000014	Hong	010-7666-4563	Oceania Australia	22



Sales_ID	P_ID	C_ID	S_ID	Tracking_number	saled_date	saled_type	pay_type
7000000	3000000	0000000	1000000	ABCH12349	2021-03-11	online	online
7000001	3000000	0000001	1000000	ABCH12310	2022-03-01	online	online
7000002	3000001	0000002	1000000	ABCH12345	2021-03-01	online	online
7000003	3000002	0000004	1000000	ABCE12345	2022-03-01	online	online
7000004	3000003	0000005	1000001	ABCF12345	2022-02-05	online	online
7000005	3000005	0000008	1000002	ZZZZ99999	2021-03-19	offline	offline
7000006	3000004	0000007	1000003	ZZZZ99999	2022-03-20	offline	offline
7000007	3000011	0000006	1000004	ABCD12345	2021-03-01	online	online
7000008	3000008	0000008	1000005	ABCH12314	2022-02-01	online	online
7000009	3000010	0000009	1000005	ZZZZ99999	2021-02-01	offline	offline
7000010	3000011	0000009	1000008	ZZZZ99999	2021-02-01	offline	offline
7000011	3000012	0000009	1000009	ZZZZ99999	2021-02-01	offline	offline
7000012	3000011	0000009	1000000	ABCH12347	2022-02-01	online	online
7000013	3000011	0000009	1000000	ZZZZ99999	2021-02-01	offline	offline
7000014	3000013	0000009	1000000	ABCH12348	2022-02-01	online	online
7000015	3000013	0000002	1000011	ABCG12345	2022-02-01	online	online
7000016	3000013	0000004	1000013	ABCA12345	2022-02-01	online	online
7000017	3000014	0000011	1000014	ABCK12345	2022-02-01	online	online
7000018	3000013	0000004	1000012	ABCH12311	2022-02-01	online	online
7000019	3000014	0000012	1000014	ABCH12313	2022-02-01	online	online



S_ID	name	region
1000000	Victor	Seoul
1000001	Yonex	California
1000002	Joobong	Gyenggi
1000003	Adidas	Gangwon
1000004	Nike	Jeonbuk
1000005	Babolot	California
1000006	Gosen	Chungbuk
1000007	Lining	Chungnam
1000008	Newbalance	Gyeongbuk
1000009	Asics	Gyeongnam
1000010	Vamos	Jeju
1000011	Kookle	California
1000012	Namazon	Brazil
1000013	Coupeong	Germany
1000014	Asus	France

Tracking_number	Shipper_ID	expected_date	arrived_date
ABCA12345	4000004	2022-03-06	2022-03-15
ABCD12345	4000000	2022-03-04	
ABCE12345	4000001	2022-03-04	2022-03-04
ABCF12345	4000002	2022-03-04	2022-03-02
ABCG12345	4000003	2022-03-05	
ABCH12310	4000008	2022-03-04	2022-03-11
ABCH12311	4000009	2022-03-16	2022-03-11
ABCH12312	4000010	2022-03-04	2022-03-11
ABCH12313	4000010	2021-06-17	2021-05-11
ABCH12314	4000012	2021-03-05	2021-04-10
ABCH12345	4000007	2022-03-06	
ABCH12347	4000006	2022-03-06	2022-03-15
ABCH12348	4000007	2022-03-11	2022-03-15
ABCH12349	4000007	2022-03-15	2022-03-15
ABCK12345	4000005	2022-03-07	
ZZZZ99999	4499999	9999-12-31	

S_ID	Com_ID	store_amount	warehouse_amount
1000000	2000000	5	5
1000000	2000001	4	5
1000001	2000002	0	0
1000001	2000003	4	2
1000002	2000004	7	5
1000003	2000005	5	8
1000004	2000006	5	10
1000004	2000007	6	11
1000005	2000008	0	0
1000005	2000009	5	4
1000005	2000010	0	0
1000006	2000011	5	4
1000006	2000012	5	4
1000007	2000013	5	4
1000007	2000014	5	4
1000008	2000015	0	0
1000009	2000016	5	4
1000009	2000017	5	4
1000010	2000018	0	0
1000011	2000019	5	4
1000012	2000020	5	4
1000013	2000021	5	4
1000013	2000022	5	4
1000014	2000023	3	4
NULL	NULL	NULL	NULL

Manu_ID	name	contact
2500000	Dongsan	1111-6666
2500001	Yangsan	1171-6666
2500002	Nangsan	1151-6666
2500003	Nongsan	1231-6666
2500004	Namsan	1181-6666
2500005	Buksan	1311-6666
2500006	Seosan	1211-6666
2500007	Yongsan	1521-6666
2500008	Bingsan	1621-6666
2500009	Visang	1571-6666
2500010	Sangsan	6521-6666
2500011	Bangsan	2521-6666
2500012	Asan	7521-6666
2500013	Gwangsan	9531-6666
2500014	Mangsang	1591-6666
NULL	NULL	NULL

Com_ID	Manu_ID	quantity
2000000	2500000	2
2000001	2500000	1
2000002	2500000	4
2000003	2500001	2
2000004	2500001	3
2000005	2500004	5
2000006	2500004	8
2000007	2500006	3
2000008	2500007	10
2000009	2500007	12
2000010	2500008	5
2000011	2500009	5
2000012	2500010	5
2000013	2500011	1
2000014	2500012	5
2000015	2500012	4
2000016	2500012	9
2000017	2500005	3
2000018	2500005	8
2000019	2500013	5
2000020	2500013	8
2000021	2500013	7
2000022	2500014	3
2000023	2500014	5
NULL	NULL	NULL

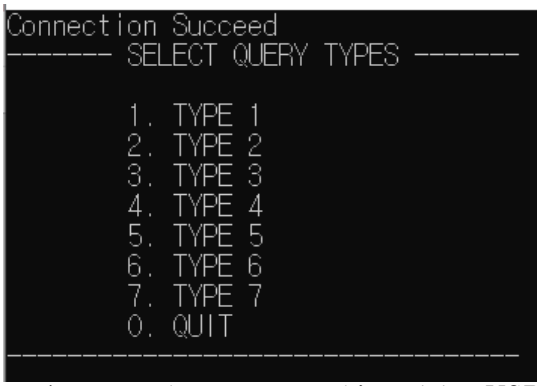
P_ID	Com_ID
3000000	2000000
3000000	2000001
3000001	2000002
3000002	2000003
3000002	2000004
3000003	2000005
3000003	2000006
3000004	2000007
3000004	2000008
3000005	2000009
3000005	2000010
3000006	2000011
3000007	2000012
3000008	2000013
3000009	2000014
3000010	2000015
3000011	2000016
3000011	2000017
3000012	2000018
3000012	2000019
3000013	2000020
3000013	2000021
3000014	2000022
3000014	2000023
NULL	NULL

pay_type	C_ID	card_number	account
offline	0000007	NULL	8888-4444
offline	0000008	NULL	3333-4444
offline	0000009	NULL	1111-2222
offline	0000013	NULL	8488-4444
offline	0000014	NULL	3343-4424
online	0000000	12345678	NULL
online	0000001	12355678	NULL
online	0000002	12365678	NULL
online	0000003	12375878	NULL
online	0000004	12325678	NULL
online	0000005	12395678	NULL
online	0000006	12355678	NULL
online	0000010	12325676	NULL
online	0000011	12395672	NULL
online	0000012	12355673	NULL
NULL	NULL	NULL	NULL

P_ID	tot_price	quantity	type
3000000	50000	4	Com
3000001	40000	1	Monitor
3000002	30000	5	Graphic
3000003	20000	3	Keyboard
3000004	22000	7	Mouse
3000005	56000	8	Item
3000006	70000	1	Bonche
3000007	90000	2	CD
3000008	120000	5	Hard
3000009	130000	6	Solid
3000010	130000	6	Card
3000011	140000	4	Soft
3000012	150000	2	Ware
3000013	160000	1	QWERTY
3000014	120000	6	Water
NULL	NULL	NULL	NULL

Shipper_ID	name	contact
4000000	CJ	111-2222
4000001	QJ	141-2222
4000002	USPC	121-2222
4000003	RJ	114-2222
4000004	GJ	117-2222
4000005	HJ	111-8222
4000006	SJ	111-9222
4000007	USPS	111-2122
4000008	KJ	111-6222
4000009	PJ	111-2229
4000010	HJ	101-2229
4000011	MJ	711-2229
4000012	NJ	211-2229
4000013	BJ	221-2229
4000014	TJ	781-2229
4499999	None	None
NULL	NULL	NULL

SQL 구현

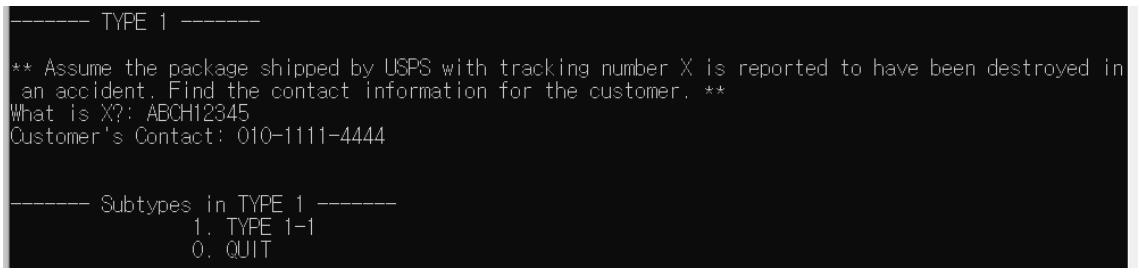


1. Assume the package shipped by USPS with tracking number X is reported to have been destroyed in an accident. Find the contact information for the customer.

실행되는 쿼리

```
select phone_number
from Customer as C, Tracking as T, Sale as S, Shipper
where T.Tracking_number = S.Tracking_number a
nd S.C_ID = C.C_ID
and T.Tracking_number = '%s' and Shipper.name='USPS'
```

%s에는 입력한 X가 들어간다.



미리 insert한 data에서 이에 해당하는 데이터의 Tracking number를 ABCH12345로 설정해두었다. (이외에는 USPS사가 운송하는 배송 상품이 없다.) 원하는 데이터가 제대로 도출됨이 확인된다.

1-1. Then find the contents of that shipment and create a new shipment of replacement items.

실행되는 쿼리

```
select S.C_ID, S.P_ID, S.S_ID, S.saled_date, S.saled_type, S.pay_type, Shipper.Shipper_ID,
S.Sales_ID
from Sale as S, Tracking as T, Shipper
where S.Tracking_number=T.Tracking_number
and T.Tracking_number = '%s' and Shipper.name='USPS'
delete from Sale where Sale.Tracking_number = '%s'
delete from Tracking where Tracking.Tracking_number = '%s'
insert into Tracking values('%s','%s','%s',NULL)
insert into Sale values('%s','%s','%s','%s','%s','%s','%s','%s')
```

각각의 %s는 Tracking, Sale에 해당하는 알맞은 값이 들어간다. 단, Tracking_number는 10000을 더한 새로운 코드로 부여된다. Traking과 Sale table은 서로 reference하는 Key가 있어서 단독으로 삭제가 불가능하다. 따라서 delete를 두 번 해주고, insert 또한 두 번 해주었다.

```
------- Subtypes in TYPE 1 -------
                1. TYPE 1-1
                0. QUIT

Input(SUBTYPE) : 1
 ** Then find the contents of that shipment and create a new shipment of replacement items. **
Customer ID: 0000002
Product ID: 3000001
Store ID: 1000000
Saled date: 2021-03-01

Inserted NEW Tracking Data, Tracking_number is ABCH22345
```

2. Find the customer who has bought the most (by price) in the past year.
실행되는 쿼리

```
select ss.c_id, sum(pp.tot_price * pp.quantity) as s
from sale as ss, product as pp
where year(ss.saled_date) = '%d' and pp.p_id = ss.p_id
group by ss.c_ID order by s desc
limit 1
```

즉, 고객의 구매 기록을 전부 뽑고, 거기서 product의 가격과 수량을 곱한 것의 합을 내림차순으로 정리하면, 맨 첫 번째 row에 가장 많이 소비한 고객이 나옴을 이용했다.

```
------- SELECT QUERY TYPES -------

        1. TYPE 1
        2. TYPE 2
        3. TYPE 3
        4. TYPE 4
        5. TYPE 5
        6. TYPE 6
        7. TYPE 7
        0. QUIT
---------------------------------

Input(TYPE) : 2

------- TYPE 2 -------

** Find the customer who has bought the most (by price) in the past year. **
Which year? : 2021
Customer's ID: 0000009


------- Subtypes in TYPE 2 -------
                1. TYPE 2-1
                0. QUIT
```

2-1. Then find the product that the customer bought the most.
실행되는 쿼리

```
select ss.p_ID, ss.c_id, sum(p.quantity * p.tot_price) as cnt
from customer as CC, Sale as ss, product as p
where cc.c_id = ss.c_id and ss.p_id = p.p_id and ss.c_id = '%s'
group by ss.p_ID order by cnt desc
Limit 1
```

즉, 2에서 찾은 고객이 구매한 상품을 가격으로 내림차순 정렬한 것을 맨 처음 row만을 출력한다.

```
------- Subtypes in TYPE 2 -------
                1. TYPE 2-1
                0. QUIT

Input(SUBTYPE) : 1
 ** Then find the product that the customer bought the most. **
Product ID: 3000011
```

3. Find all products sold in the past year.

실행되는 쿼리

```
select distinct s.p_id
from sale as s
where year(s.saled_date)='%d'
```

sale table에서 saled_date의 year가 입력된 년도인 것만을 distinct하게 출력한다.

```
------- SELECT QUERY TYPES -------

        1. TYPE 1
        2. TYPE 2
        3. TYPE 3
        4. TYPE 4
        5. TYPE 5
        6. TYPE 6
        7. TYPE 7
        0. QUIT
---------------------------------

Input(TYPE) : 3

------- TYPE 3 -------

** Find all products sold in the past year. **
Which year? : 2021
Product ID
3000000
3000001
3000005
3000010
3000011
3000012
```

3-1. Then find the top k products by dollar-amount sold.

실행되는 쿼리

```
select ss.p_ID, sum(p.quantity * p.tot_price) as cnt
from customer as CC, Sale as ss, product as p
where cc.c_id = ss.c_id and ss.p_id = p.p_id and year(ss.saled_date) = '%d'
group by ss.p_ID order by cnt desc
limit %d
```

구매 기록에 있는 product의 가격과 수량을 곱한 것을 전부 더한 순서대로 내림차순 정렬해주었다. 이 table의 Limit를 K로 제한한다.

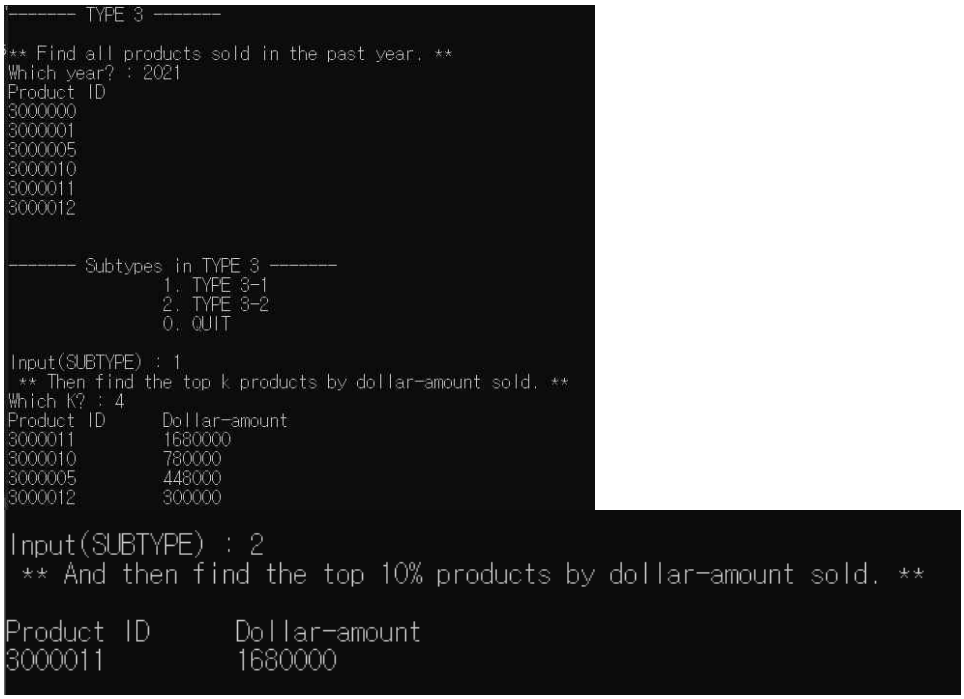
3-2. And then find the top 10% percent products by dollar-amount sold.
실행되는 쿼리

```
select ss.p_ID, sum(p.quantity * p.tot_price) as cnt
from customer as CC, Sale as ss, product as p
where cc.c_id = ss.c_id and ss.p_id = p.p_id and year(ss.saled_date) = '%d'
group by ss.p_ID order by cnt desc
```

에서 select 되는 row의 개수를 idx라 하면
idx/=10을 해준다. (idx==0이면 1로 설정)
이 후

```
select ss.p_ID, sum(p.quantity * p.tot_price) as cnt
from customer as CC, Sale as ss, product as p
where cc.c_id = ss.c_id and ss.p_id = p.p_id and year(ss.saled_date) = '%d'
group by ss.p_ID order by cnt desc
limit idx
```

를 실행한다.
3-1과 원리는 동일하다.



4. Find all products by unit sales in the past year.
실행되는 쿼리
```
select s.sales_id, s.p_id, s.saled_date, p.quantity
from product as p, sale as s
where p.p_id=s.p_id and year(s.saled_date)='%d' order by s.sales_id
```
sale 목록에서 입력된 년도에 팔린 product의 quantity 숫자를 전부 출력해준다.

```
Input(TYPE) : 4

------- TYPE 4 -------

** Find all products by unit sales in the past year. **
Which year? : 2021

Saled ID        Product ID      Saled_date      Quantity
7000000         3000000         2021-03-11             4
7000002         3000001         2021-03-01             1
7000005         3000005         2021-03-19             8
7000007         3000011         2021-03-01             4
7000009         3000010         2021-02-01             6
7000010         3000011         2021-02-01             4
7000011         3000012         2021-02-01             2
7000013         3000011         2021-02-01             4
```

4-1. Then find the top k products by unit sales.
실행되는 쿼리
```
select ss.p_ID, sum(p.quantity) as cnt
from customer as CC, Sale as ss, product as p
where cc.c_id = ss.c_id and ss.p_id = p.p_id and year(ss.saled_date) = '%d'
group by ss.p_ID order by cnt desc
limit K
```
팔린 product의 quantity를 모두 더한 것을 내림차순 정렬하여 K의 limit 만큼만 select 해준다.

4-2. And then find the top 10% products by unit sales.
실행되는 쿼리
```
select ss.p_ID, sum(p.quantity) as cnt
from customer as CC, Sale as ss, product as p
where cc.c_id = ss.c_id and ss.p_id = p.p_id and year(ss.saled_date) = '%d'
group by ss.p_ID order by cnt desc
```
에서 select 되는 row의 개수를 idx라 하면
idx/=10을 해준다. (idx==0이면 1로 설정)
이 후
```
select ss.p_ID, sum(p.quantity) as cnt
from customer as CC, Sale as ss, product as p
where cc.c_id = ss.c_id and ss.p_id = p.p_id and year(ss.saled_date) = '%d'
group by ss.p_ID order by cnt desc
limit idx
```
를 실행한다.
4-1과 기본 원리는 동일하다. 내림차순 정렬 후 상위 10%만큼만 select 해준다.

```
** Find all products by unit sales in the past year. **
Which year? : 2021

Saled ID        Product ID      Saled_date      Quantity
7000000         3000000         2021-03-11              4
7000002         3000001         2021-03-01              1
7000005         3000005         2021-03-19              8
7000007         3000011         2021-03-01              4
7000009         3000010         2021-02-01              6
7000010         3000011         2021-02-01              4
7000011         3000012         2021-02-01              2
7000013         3000011         2021-02-01              4


------- Subtypes in TYPE 4 -------
                1. TYPE 4-1
                2. TYPE 4-2
                0. QUIT

Input(SUBTYPE) : 1
 ** Then find the top k products by unit sales. **
Which K? : 5
Product ID      Quantity
3000011         12
3000005         8
3000010         6
3000000         4
3000012         2
```

```
------- Subtypes in TYPE 4 -------
                1. TYPE 4-1
                2. TYPE 4-2
                0. QUIT

Input(SUBTYPE) : 2
 ** And then find the top 10% products by unit sales. **

Product ID      Quantity
3000011         12
```

5. Find those products that are out-of-stock at every store in California.

실행되는 쿼리

```
select p.p_id, p.type, p.tot_price, c.com_id
from product as p, component as c, package as pa, management as m, store as s
where pa.p_id = p.p_id and pa.com_id = c.com_id and m.com_id = c.com_id
and s.s_id = m.s_id and s.region = 'California' and
(m.store_amount = '0' AND m.warehouse_amount = '0')
```

California의 위치한 store와 같은 ID의 management의 store_amount와 warehouse_amout의 값이 모두 0인 component가 있는 product를 전부 출력해준다. 또한 어떤 부품이 품절인지도 보여준다.

```
Input(TYPE) : 5

------- TYPE 5 -------

** Find those products that are out-of-stock at every store in California. **
Product ID      Product type    Product price   Component ID
3000001         Monitor         40000           2000002
3000004         Mouse           22000           2000008
3000005         Item            56000           2000010
```

6. Find those packages that were not delivered within the promised time.
실행되는 쿼리

```
select s.s_id, t.tracking_number, t.expected_date, t.arrived_date
from sale as s, tracking as t
where t.tracking_number = s.tracking_number and t.expected_date < t.arrived_date
```

sale 목록에서, tracking_number를 통해 약속 도착 날짜가 실제 도착 날짜보다 빠른 경우만을 전부 찾아준다. 단, 아직 도착하지 않았거나 offline 경우에는 자동으로 제외된다.

```
------- TYPE 6 -------

** Find those packages that were not delivered within the promised time. **
Sales ID        Tracking Number         Expected date           Arrived date
1000013         ABCA12345               2022-03-06              2022-03-15
1000000         ABCH12310               2022-03-04              2022-03-11
1000005         ABCH12314               2021-03-05              2021-04-10
1000000         ABCH12347               2022-03-06              2022-03-15
1000000         ABCH12348               2022-03-11              2022-03-15
```

7. Generate the bill for each customer for the past month.
실행되는 쿼리

```
create table bill as
select c.c_id, c.name, c.phone_number, sum(p.tot_price * p.quantity) as tot_bill
from customer as c, sale as s , product as p
where s.c_id = c.c_id and p.p_id = s.p_id and year(s.saled_date)='%d'
and month(s.saled_date)='%d'
group by c.c_id
```

입력한 년도와 월에 각 고객이 구매한 모든 sale 데이터에서 product의 가격 * 수량을 전부 더한 값, 즉 고객의 소비 액수를 담은 하나의 table을 새로 만들어준다. 또한 create와 동시에 select도 해주어서 콘솔에도 출력된다. 이후 drop table bill로 중복 생성을 피했다.

```
Input(TYPE) : 7

------- TYPE 7 -------

** Generate the bill for each customer for the past month. **
Which year? : 2022
Which month? : 2
Customer ID     Customer name   Customer Contact        Total bill
0000002         Chang           010-1111-4444           160000
0000004         Lim             010-6666-2222           320000
0000005         Lee             010-6666-3333           60000
0000008         Seo             010-7666-3333           600000
0000009         Park            010-6666-3443           720000
0000011         Koo             010-8666-6433           720000
0000012         Min             010-9666-3333           720000
```

코드 흐름

ODBC를 통해 C언어로 구현할 수 있다. MySQL과 Visual Studio를 서로 연동한다. 호스트, 사용자, 비밀번호, 스키마 이름을 mysql_real_connet를 통해 연결된다. 그리고 기본 CRUD 정보가 있는 20201564_1.txt를 읽어들여 테이블과 데이터를 생성한다. 쿼리는 사용자에 입력에 맞춰서 mysql_query를 통해 전달된다. 이때 실행 결과는 mysql_store_result를 통해 반환받고 mysql_free_result를 통해 free시킨다. 모든 과정은 sprintf를 이용해 구현했다.
0번이 입력되어 모든 과정이 종료될 때는 20201564_2.txt를 읽어들여 delete와 drop을 실시한다. 그 코드는 다음과 같다. create한 순서의 반대의 순서로 삭제하면 FK에 의한 충돌을 피할 수 있다.

```
SET SQL_SAFE_UPDATES = 0;

delete from Sale;
delete from Tracking;
delete from Shipper;
delete from Payment;
delete from Package;
delete from Product;
delete from Management;
delete from Ordering;
delete from Manufacturer;
delete from Component;
delete from Brand;
delete from Store;
delete from Customer;

drop table Sale;
drop table Tracking;
drop table Shipper;
drop table Payment;
drop table Package;
drop table Product;
drop table Management;
drop table Ordering;
drop table Manufacturer;
drop table Component;
drop table Brand;
drop table Store;
drop table Customer;
```