Database System

CSE4110-01

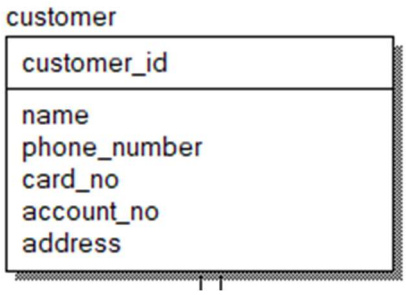
Project 2

학번: 20160169

이름: 박성환

1. BCNF test

A. Customer



i. customer_id → name, phone_number, card_no, account_no, addresss

- customer_id가 super key 이므로 BCNF를 만족한다.

ii. phone_number → name, card_no, account_no, address

- phone_number가 super key 이므로 BCNF를 만족한다.

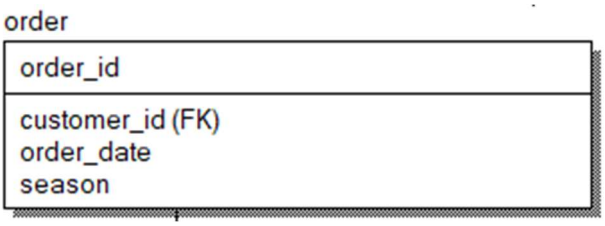
iii. card_no → name, phone_number, account_no, addresss

- card_no가 super key 이므로 BCNF를 만족한다.

iv. account_no → name, phone_number, card_no, address

- account_no가 super key 이므로 BCNF를 만족한다.

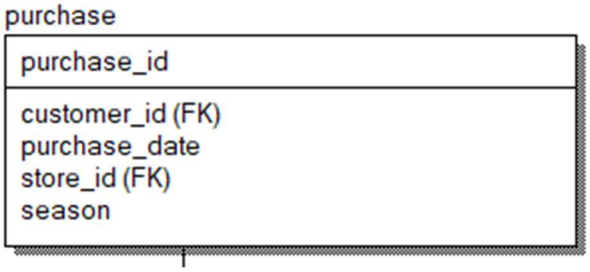
B. Order



i. order_id → customer_id, order_date, season

- order_id가 super key 이므로 BCNF를 만족한다.

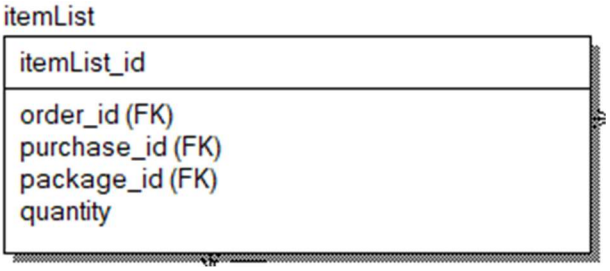
C. Purchase



i. purchase_id → customer_id, store_id, purchase_date, season

- purchase_id가 super key 이므로 BCNF를 만족한다.

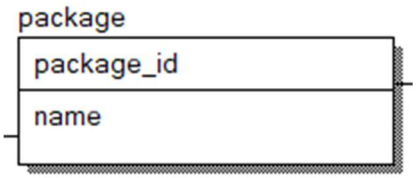
D. itemList



i. itemList_id → purchase_id, order_id, package_id, quantity

- itemList_id가 super key이므로 BCNF를 만족한다.

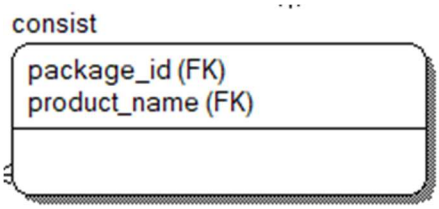
E. package



i. package_id → name

- package_id가 super key이므로 BCNF를 만족한다. 모든 package_id에 대하여 package의 이름이 다르기 때문이다.

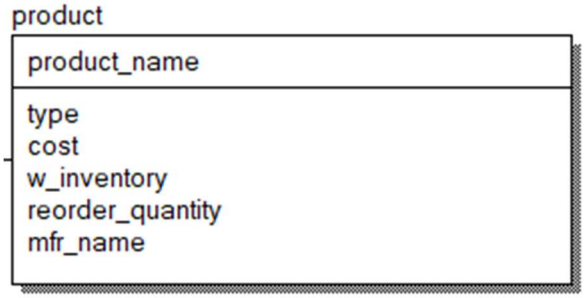
F. consist



i. package_id, product_name → packaged_id, product_name

- trivial한 functional dependency이므로 BCNF를 만족한다.

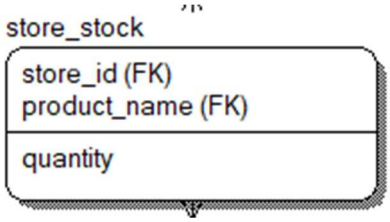
G. product



i. product_name → mfr_name, type, cost, w_inventory, reorder_quantity

- product_name이 super key이므로 BCNF를 만족한다.

H. store_stock



i. store_id, product_name → quantity

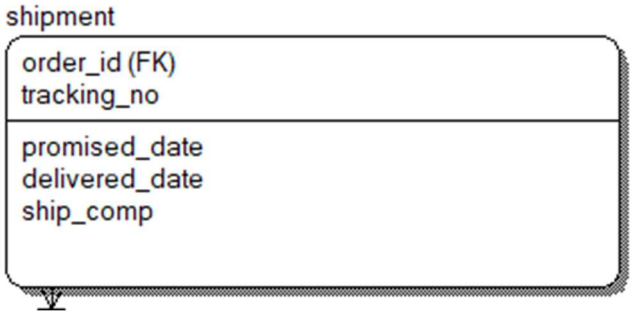
- store_id, product_name 조합 super key 이므로 BCNF를 만족한다.

I. store

i. store_id → region

- store_id가 super key 이므로 BCNF를 만족한다.

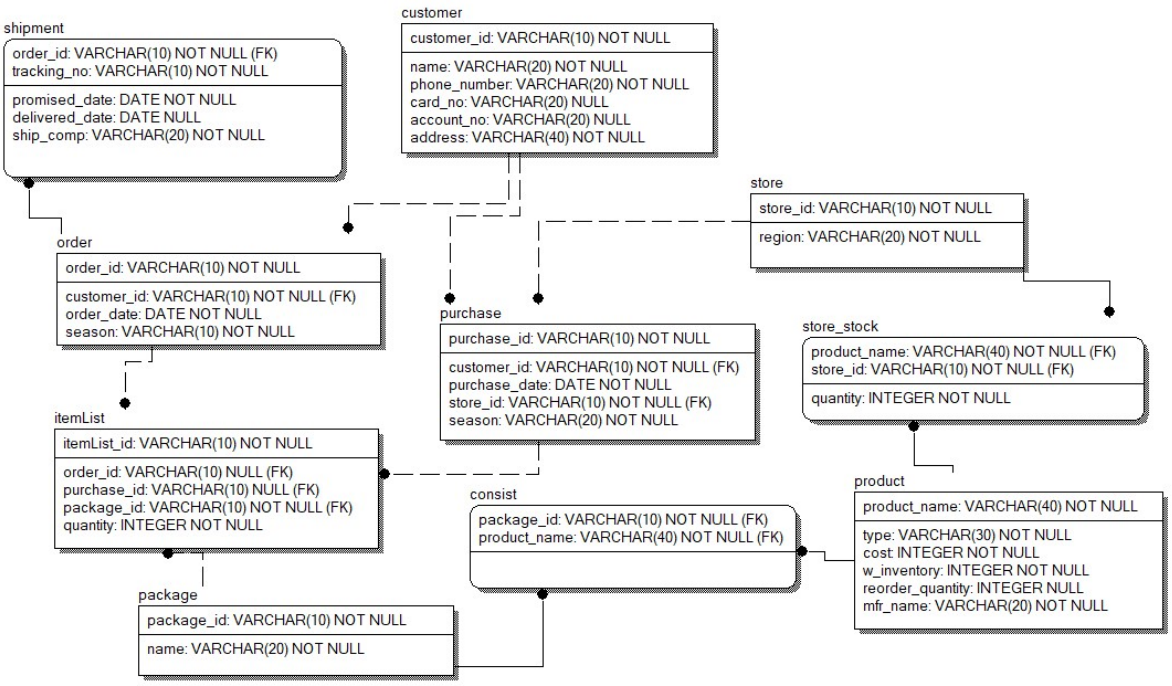
J. Shipment



i. order_id, tracking_no → promised_date, delivered_date, ship_comp

- order_id, tracking_no가 super key이므로 BCNF를 만족한다.

2. Physical Diagram

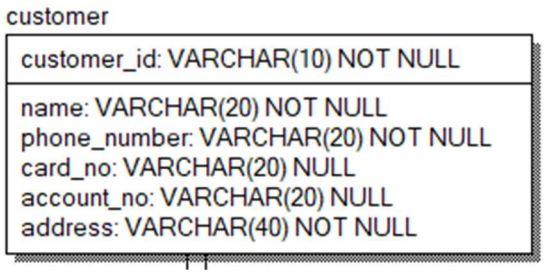


A. 모든 ID를 8자리로 설정하였다.

B. 프로젝트 2에서 처리해야 하는 SQL문들을 고려하여 프로젝트 1의 logical diagram에서 relation 두 개를 줄여 간소화하였다.

3. 각 Relation의 Physical Diagram & SQL CREATE 구문

A. customer




```
CREATE TABLE customer (
    customer_id VARCHAR(10) unique,
    name VARCHAR(20)  NOT NULL ,
    phone_number VARCHAR(20)  NOT NULL unique,
    card_no VARCHAR(20),
    account_no VARCHAR(20),
    address VARCHAR(40)  NOT NULL,
    primary key(customer_id));
```

i. customer_id: id는 8자리이므로 여유를 두어 최대 10자리 문자열로 설정하였다. Primary key이므로 NOT NULL로 설정을 하였으며 SQL 문에서는 unique로 설정을 하였다.

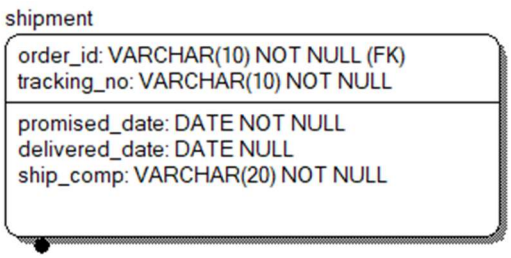
ii. phone_number: 최대 20자리 문자열로 설정하였고 고객별 연락처이기 때문에 NOT NULL 및 SQL 문에서는 unique로 설정을 하였다.

iii. card_no: 최대 20자리 문자열로 설정하였다. 모든 고객들의 카드 번호를 저장하지 않기 때문에 NULL 값도 가능하다.

iv. account_no: 최대 20자리 문자열로 설정하였다. Account_no의 유무로 contract 유무를 판단하기 때문에 NULL 값도 가능하도록 하였다.

v. address: 최대 40자리 문자열로 설정하였다. Shipment relation의 destination으로 사용되기 때문에 NOT NULL로 설정하였다.

B. shipment




```
CREATE TABLE shipment (
    order_id VARCHAR(10) unique,
    tracking_no  VARCHAR(10) unique,
    ship_comp VARCHAR(20)
        NOT NULL check(ship_comp in('USPS','DHL','FedEx')),
    promised_date DATE,
    delivered_date DATE,
    primary key(order_id, tracking_no),
    foreign key (order_id) references orders(order_id)
        on delete cascade );
```

i. order_id: Order Relation을 참조하는 foreign key이다.

ii. tracking_no: order_id와 함께 super key 역할을 한다. id들과 마찬가지로 최대 10자리 문자열로 설정하였다. Shipment의 필수적인 정보이므로 NOT NULL로 설정

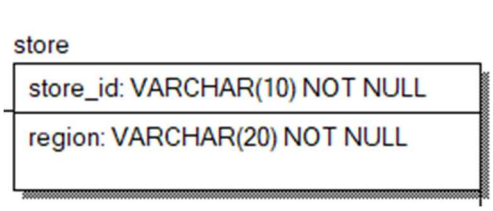
하였고 SQL 구문에서는 unique로 설정하였다.

iii. promised_date: 날짜이므로 date 자료형으로 설정하였고 배송 예정 날짜는 shipment에 필수적이므로 NOT NULL로 설정하였다.

iv. delivered_date: 날짜이므로 date 자료형으로 설정하였다. 배송 여부를 NULL 값인지 아닌지를 통해 구분하므로 NULL 값을 허용한다.

v. ship_comp: 배송 회사를 의미한다. 최대 20자리 문자열로 설정하였다. SQL 구문을 통해 회사를 USPS, DHL, FedEx로 제한하였다.

C. store

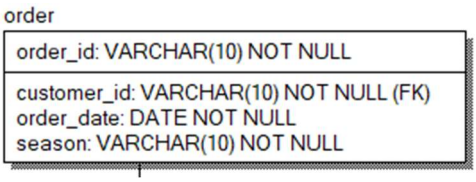



```
CREATE TABLE store (
    store_id              VARCHAR(10) unique,
    region                VARCHAR(20)  NOT NULL,
    primary key(store_id)
);
```

i. store_id: 다른 id들과 마찬가지로 최대 10자리 문자열로 설정하였다. Store의 고유한 id 값이므로 SQL 구문에서 unique 설정을 하였다.

ii. region: store가 위치한 지역을 주 단위로 나타낸다. 최대 20자리 문자열로 설정하였다. Store는 항상 특정한 주에 위치하므로 NOT NULL로 설정하였다.

D. order




```
CREATE TABLE orders (
    order_id              VARCHAR(10) unique,
    customer_id           VARCHAR(10),
    order_date                 DATE,
    season                VARCHAR(10)
        check(season in ('SPRING', 'SUMMER','FALL','WINTER')),
    primary key(order_id),
    foreign key (customer_id)
        references customer(customer_id) on delete set null );
```

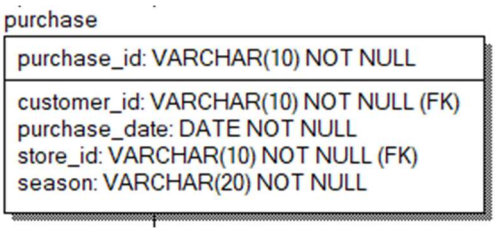
i. order_id : 다른 id들과 마찬가지로 최대 10자리 문자열로 설정하였다. 각 온라인 주문들의 고유한 id 값이므로 SQL 구문에서 unique 설정을 하였다.

ii. customer_id: Customer relation을 참조하는 foreign key이다.

iii. order_date: 날짜이므로 date 자료형을 사용하였다. 주문 날짜는 항상 존재하므로 NOT NULL로 설정하였다.

iv. season: 최대 10자리 문자열로 설정하였고 SQL 구문을 통해 SPRING, SUMMER, FALL, WINTER 중 한 가지 값을 갖도록 하였다.

E. purchase




```
CREATE TABLE purchase (
    purchase_id           VARCHAR(10) unique,
    customer_id           VARCHAR(10),
    store_id              VARCHAR(10),
    purchase_date         DATE,
    season                VARCHAR(10)
        check(season in ('SPRING', 'SUMMER','FALL','WINTER')),
    primary key(purchase_id),
    foreign key (customer_id)
        references customer(customer_id) on delete set null,
    foreign key (store_id)
        references store(store_id) on delete set null );
```

i. purchase_id: 다른 id들과 마찬가지로 최대 10자리 문자열로 설정하였다. 오프라인에서 이루어진 각각의 구매의 고유한 id 값이므로 SQL 구문에서 unique 설정을 하였다.

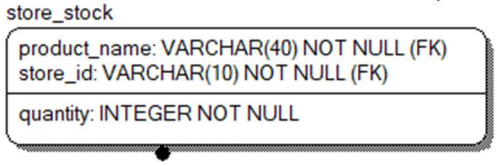
ii. customer_id: Customer relation을 참조하는 foreign key이다.

iii. store_id: Store relation을 참조하는 foreign key이다.

iv. purchase_date: 날짜이므로 date 자료형을 사용하였다. 구매 날짜는 항상 존재하므로 NOT NULL로 설정하였다.

v. season: 최대 10자리 문자열로 설정하였고 SQL 구문을 통해 SPRING, SUMMER, FALL, WINTER 중 한 가지 값을 갖도록 하였다.

F. store_stock



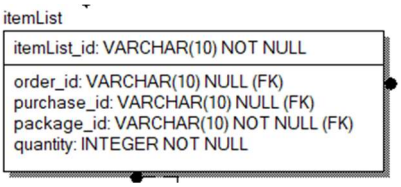

```
CREATE TABLE store_stock (
    store_id              VARCHAR(10),
    product_name          VARCHAR(40),
    quantity              NUMERIC(10,0)  check(quantity >= 0),
    primary key(product_name, store_id),
    foreign key (product_name)
        references product(product_name) on delete cascade,
    foreign key (store_id)
        references store(store_id) on delete cascade
);
```

i. product_name: Product relation을 참조하는 foreign key이다.

ii. store_id: Store relation을 참조하는 foreign key이다.

iii. quantity: 각 매장의 특정 상품의 재고 수를 표현하므로 INTEGER 자료형으로 설정하였고 SQL 문에서 NOT NULL로 하였다. 재고 수는 항상 0 이상이므로 SQL 문에서 0 이상의 값을 가지도록 조건을 추가하였다.

G. itemList




```
CREATE TABLE itemList (
    itemList_id          VARCHAR(10) unique,
    order_id             VARCHAR(10),
    purchase_id          VARCHAR(10),
    package_id           VARCHAR(10),
    quantity             NUMERIC(10,0)  check(quantity > 0),
    primary key(itemList_id),
    foreign key (order_id) references orders(order_id) on delete cascade,
    foreign key (purchase_id) references purchase(purchase_id) on delete cascade,
    foreign key (package_id) references package(package_id) on delete cascade );
```

i. itemList_ID: itemList relation의 각 tuple의 고유한 ID 값이고 SQL 구문에서 unique 설정을 하였다.

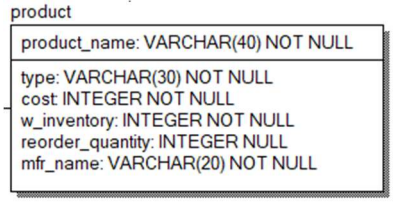
ii. order_ID: order relation을 참조하는 foreign key이다.

iii. purchase_ID: purchase relation을 참조하는 foreign key이다.

iv. package_ID: package relation을 참조하는 foreign key이다. 언제나 구매한 상품이 존재하므로 NOT NULL 설정을 하였다.

v. quantity: 특정 상품의 구매 수량을 나타내므로 INTEGER 자료형으로 설정하였고 SQL 문에서 NOT NULL로 하였다. 0개를 구매하는 경우는 없으므로 SQL 문에서 항상 양수이도록 조건을 추가하였다.

H. product




```
CREATE TABLE product (
    product_name          VARCHAR(40) unique,
    product_type          VARCHAR(30)  NOT NULL ,
    cost                  NUMERIC(10,0)  check(cost > 0),
    w_inventory           NUMERIC(10,0)  check(w_inventory >= 0),
    mfr_name              VARCHAR(20) NOT NULL,
    reorder_quantity      NUMERIC(10,0)  check(reorder_quantity >= 0),
    primary key(product_name)
);
```

i. product_name: 최대 40자리 문자열로 설정하였다. Product relation의 super key 이므로 SQL 구문에서 unique 조건을 추가하였다.

ii. product_type: 최대 30자리 문자열로 설정하였다. 하나의 상품은 항상 특정한 상품 분류에 속하므로 NOT NULL 조건을 추가하였다.

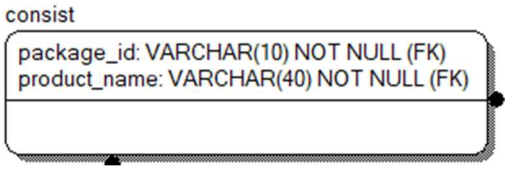
iii. cost: 상품의 가격을 나타내므로 INTEGER 자료형으로 설정하였고 하나의 상품은 항상 가격을 가지므로 NOT NULL 설정을 하였다. SQL 구문에서는 항상 양수 값을 가지도록 조건을 추가하였다.

iv. w_inventory: warehouse의 재고 수를 나타내므로 INTEGER 자료형으로 설정하였고 NOT NULL 설정을 하였다. SQL 구문에서는 항상 0 이상의 값을 가지도록 조건을 추가하였다.

v. reorder_quantity: manufacturer에게 신청한 재고 주문량을 의미한다. INTEGER 자료형으로 설정하였다. 항상 재고 주문을 해놓는 것이 아니므로 NULL 값을 허용하였다. SQL 구문에서는 항상 0 이상의 값을 가지도록 설정하였다.

vi. mfr_name: manufacturer를 의미한다. 최대 20자리 문자열로 설정하였고 모든 상품은 manufacturer를 가지므로 NOT NULL로 설정하였다.

l. consist

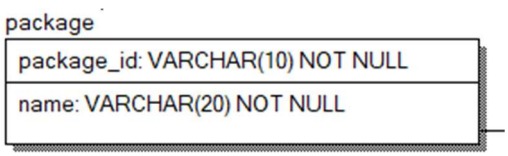



```
CREATE TABLE consist (
    package_id          VARCHAR(10),
    product_name        VARCHAR(40),
    primary key(package_id, product_name),
    foreign key (package_id)
        references package(package_id) on delete cascade,
    foreign key (product_name)
        references product(product_name) on delete cascade
);
```

i. package_id: package relation을 참조하는 foreign key이다. 모든 package에 대하여 해당 package를 구성하는 상품을 표기해야 하므로 NOT NULL로 설정하였다

ii. product_name: Product relation을 참조하는 foreign key이다. Product를 갖고 있지 않는 package는 없으므로 NOT NULL로 설정하였다.

J. package




```
CREATE TABLE package (
    package_id          VARCHAR(10) unique,
    name                VARCHAR(20)  NOT NULL,
    primary key(package_id)
);
```

i. package_id: package relation의 super key이므로 NOT NULL 설정을 하였고 SQL 구문에서는 unique 조건을 추가하였다.

ii. name: 최대 20자리 문자열로 설정하였고 해당 package의 이름을 나타내므로 NOT NULL 조건을 추가하였다.

4. Query

- **(TYPE 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.**

```
select C.name, C.phone_number, O.order_id, S.ship_comp, S.tracking_no
from shipment as S join orders as O on S.order_id = O.order_id
join customer as C on O.customer_id = C.customer_id
where S.tracking_no = '70000016' and S.ship_comp = 'USPS' and S.delivered_date is null
```

- 마지막 줄 where 구문의 tracking_no에 해당하는 값은 실제 프로그램에서 delivered_date이 NULL인 tuple 중에서 랜덤으로 선택하도록 하였다.

```
select C.name, C.phone_number, O.order_id, S.ship_comp, S.tracking_no
from shipment as S join orders as O on S.order_id = O.order_id
join customer as C on O.customer_id = C.customer_id
where S.tracking_no = '%s' and S.ship_comp = '%s' and S.delivered_date is null;
```

```
---------------------------- TYPE 1 -----------------------------

** Assume the package shipped by 'USPS' with tracking number '70000016' is reported to have been destroyed in an accident. **
** Find the contact information for the customer. **

        ================================================================================
        | Name          Phone no.           Order_ID      Shipper      Tracking No.|
        ================================================================================
        | John          010-2745-6613       20000016      USPS           70000016 |
        ================================================================================
```

- 위는 TYPE 1의 실행 결과이다.

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

```
select C.product_name, sum(I.quantity)
from shipment as S join itemList as I on S.order_id = I.order_id
    join package as P on P.package_id = I.package_id
    join consist as C on C.package_id = P.package_id
where S.order_id = '20000016' and S.ship_comp = 'USPS' and S.delivered_date is null
group by C.product_name;
```

- 위 SQL 구문에서 where 구문의 order_id에 해당하는 값은 type 1에서 얻은 order_id를 그대로 사용하도록 하였다. Shipment는 order와 별개로 관리되므로 우선 type1에서 찾은 shipment는 삭제한 후 order_id는 기존의 것을 사용하고 tracking_no를 갱신하여 새로운 shipment가 추가되도록 하였다.

```
delete from shipment where shipment.order_id = '%d';
insert into shipment values('%d','%d','%s',curdate(),null);
```

```
sprintf(delQuery, "delete from shipment where shipment.order_id = '%d'", order_id);
sprintf(insertQuery, "insert into shipment values('%d','%d','%s',curdate(),null)", order_id, new_track_no, shipper);
```

- Delete와 insert는 위의 SQL query를 이용해 처리하였다.

```
---------------------------- TYPE 1-1 ----------------------------

** Then find the contents of the shipment for tracking number '70000016' **

        ================================
        | Product             Quantity |
        ================================
        | ASUS PG32UQX               1 |
        | MX Ergo                    1 |
        | Sagaris                    1 |
        | ThinkPad                   1 |
        | ThinkStation               1 |
        ================================

        << New Shipment with Order_ID: 20000016, Tracking_No: 70000021 inserted. >>
```

- 위는 TYPE 1-1의 실행결과이다.

- **(TYPE 2) Find the customer who has bought the most (by price) in the past year.**

```sql
with X as (
    with A as (
        select CS1.customer_id as 'acid', CS1.name as 'acus', sum(I1.quantity * PD1.cost) as 'asum'
        from customer as CS1 join orders as O on CS1.customer_id = O.customer_id
        join itemList as I1 on O.order_id = I1.order_id
        join consist as C1 on I1.package_id = C1.package_id
        join product as PD1 on PD1.product_name = C1.product_name
        where O.order_date >= DATE_SUB(NOW(),INTERVAL 1 YEAR)
        group by acid
    ),
    B as (
        select CS2.customer_id as 'bcid', CS2.name as 'bcus', sum(I2.quantity * PD2.cost) as 'bsum'
        from customer as CS2 join purchase as P on CS2.customer_id = P.customer_id
        join itemList as I2 on P.purchase_id = I2.purchase_id
        join consist as C2 on I2.package_id = C2.package_id
        join product as PD2 on PD2.product_name = C2.product_name
        where P.purchase_date >= DATE_SUB(NOW(),INTERVAL 1 YEAR)
        group by bcid
    )
    select *
        from
        (select * from A left join B on A.acid = B.bcid) as T
        union
        (select * from A right join B on A.acid = B.bcid)
) select coalesce(acid,bcid), coalesce(acus, bcus), coalesce(asum,0)+coalesce(bsum,0) as 'total' from X
  order by total desc
  limit 1 ;
```

- Order relation과 purchase relation이 itemList relation만 공유하면서 나뉘어 있으므로 SQL 구문의 with 구문에서 각각 join한 후 필요한 정보만 얻도록 하였다.
- A relation은 customer, itemList, consist, product과 order relation을 join하고 B relation은 customer, itemList, consist, product과 purchase relation을 join한다.
- With 구문의 각각의 relation A, B에서 고객 별로 총 사용한 금액을 확인할 수 있도록 customer_id로 그룹화하고 sum()을 통해 고객 별로 사용한 금액을 모두 합산하였다.
- With 구문의 A와 B relation을 21 ~ 24번 줄에서 Full Outer Join한 후 총 금액으로 내림차순 정렬하였다. 마지막 줄의 "limit 1" 구문을 통해 가장 많은 금액을 사용한 고객을 출력한다.
- DATE_SUB(NOW(), INTERVAL 1 YEAR) 구문을 통해 지난 1년으로 기간을 제한하였다.

```
---------------------------- TYPE 2 -----------------------------
** Find the customer who has bought the most (by price) in the past year. **

        ========================================
        | Customer_ID     Name          Amount($)|
        ========================================
        | 00000002        Jack             18610 |
        ========================================
```

- 위는 TYPE 2의 실행 결과이다. 1년 동안 가장 많은 금액을 구매한 고객과 그 금액을 표시한다.

- **(TYPE 2-1) Then find the product that the customer bought the most.**

```
1  with X as (
2      with A as (
3          select C1.product_name as 'aprod', sum(I1.quantity * PD1.cost) as 'asum'
4          from orders as O join itemList as I1 on O.order_id = I1.order_id
5          join consist as C1 on I1.package_id = C1.package_id
6          join product as PD1 on C1.product_name = PD1.product_name
7          where O.order_date >= DATE_SUB(NOW(),INTERVAL 1 YEAR)
8          and O.customer_id = '%s'
9          group by C1.product_name
10     ),
11     B as (
12         select C2.product_name as 'bprod', sum(I2.quantity * PD2.cost) as 'bsum'
13         from purchase as P join itemList as I2 on P.purchase_id = I2.purchase_id
14         join consist as C2 on I2.package_id = C2.package_id
15         join product as PD2 on C2.product_name = PD2.product_name
16         where P.purchase_date >= DATE_SUB(NOW(),INTERVAL 1 YEAR)
17         and p.customer_id = '%s'
18         group by C2.product_name
19     )
20     select *
21     from
22         (select * from A left join B on A.aprod = B.bprod) as T
23         union
24         (select * from A right join B on A.aprod = B.bprod)
25 )
26 select coalesce(aprod,bprod), coalesce(asum,0)+coalesce(bsum,0) as 'total' from X
27 order by total desc
28 limit 1;
```

- 위 SQL 구문에서 8번 줄과 17번 줄의 customer_id는 type2에서 찾은 customer_id를 사용하도록 하였다.
- With relation의 A, B를 구할 때 TYPE 2의 join에서 특정 customer_id만 탐색하는 구문만 추가하였다.
- Order relation과 purchase relation이 itemList relation만 공유하면서 나뉘어 있으므로 SQL 구문의 with 구문에서 각각 join한 후 필요한 정보만 얻도록 하였다.
- DATE_SUB(NOW(), INTERVAL 1 YEAR) 구문을 통해 지난 1년으로 기간을

제한하였다.

```
---------------------------- TYPE 2-1 ----------------------------
** The product that Jack had bought the most in the past year. **

        ================================
        | Product             Amount($)|
        ================================
        | iMac                     7500 |
        ================================
```

- 위는 TYPE 2-1의 실행 결과이다. TYPE2에서 찾은 고객이 가장 많이 산 상품을 출력한다.

- **(TYPE 3) Find all products sold in the past year.**

```
1  select *
2  from
3  (
4      select distinct C1.product_name
5          from orders as O
6          join itemList as I1 on O.order_id = I1.order_id
7          join consist as C1 on I1.package_id = C1.package_id
8          where O.order_date >= DATE_SUB(NOW(),INTERVAL 1 YEAR)
9  ) as X union distinct (
10     select distinct C2.product_name
11     from purchase as P
12     join itemList as I2 on P.purchase_id = I2.purchase_id
13     join consist as C2 on I2.package_id = C2.package_id
14     where P.purchase_date >= DATE_SUB(NOW(),INTERVAL 1 YEAR)
15 ) ;
```

- Order와 Purchase relation이 나뉘어 있으므로 itemList와 각각을 product_name만 선별하여 join한 후 두 join된 relation을 9번 줄의 union distinct를 통해 product_name이 중복되지 않게 합친다.

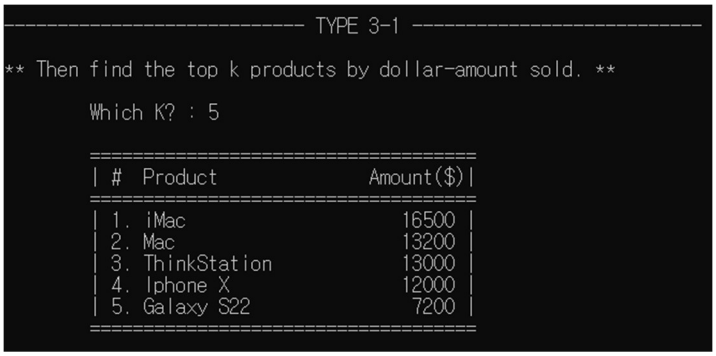
```
---------------------------- TYPE 3 ----------------------------
** Find all products sold in the past year. **

        ======================
        | Product            |
        ======================
        | Iphone X           |
        | Magic Keyboard     |
        | Magic Mouse        |
        | iMac               |
        | iPad Pro           |
        | Mac                |
        | ThinkStation       |
        | Gram               |
        | Pebble             |
        | ASUS PG27UQX       |
        | MX Ergo            |
        | Pavilion           |
        | ThinkPad           |
        | ASUS PG32UQX       |
        | Sagaris            |
        | Frontier           |
        | Lenovo Legion      |
        | ThinkVision        |
        | Neo G7             |
        | Galaxy S22         |
        | Galaxy S8          |
        ======================
```

- 위는 TYPE 3의 실행결과이다.

- **(TYPE 3-1) Then find the top k products by dollar-amount sold.**

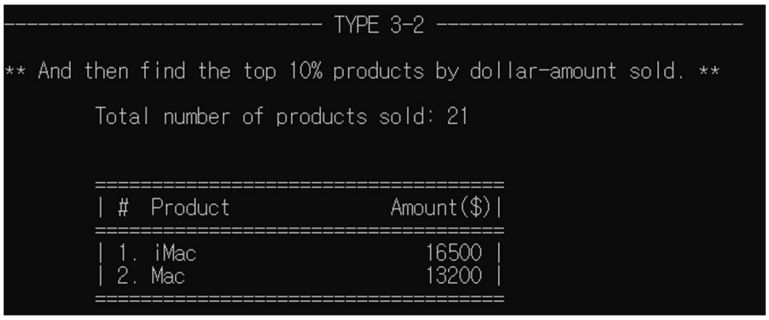
```sql
with X as (
    with A as (
        select C1.product_name as 'aprod', sum(I1.quantity * PD1.cost) as 'asum'
            from orders as O
            join itemList as I1 on O.order_id = I1.order_id
            join consist as C1 on I1.package_id = C1.package_id
            join product as PD1 on PD1.product_name = C1.product_name
            where O.order_date >= DATE_SUB(NOW(),INTERVAL 1 YEAR)
            group by PD1.product_name
    ),
     B as (
        select C2.product_name as 'bprod', sum(I2.quantity * PD2.cost) as 'bsum'
            from purchase as P
            join itemList as I2 on P.purchase_id = I2.purchase_id
            join consist as C2 on I2.package_id = C2.package_id
            join product as PD2 on PD2.product_name = C2.product_name
            where P.purchase_date >= DATE_SUB(NOW(),INTERVAL 1 YEAR)
            group by C2.product_name
    )
    select *
        from
        (select * from A left join B on A.aprod = B.bprod) as T
        union
        (select * from A right join B on A.aprod = B.bprod)
)
select coalesce(aprod,bprod), coalesce(asum,0)+coalesce(bsum,0) as total from X
order by total desc
;
```

- Order relation과 purchase relation이 itemList relation만 공유하면서 나뉘어 있으므로 SQL 구문의 with 구문에서 각각 join한 후 필요한 정보만 얻도록 하였다.
- A relation은 itemList, consist, product과 order relation들을 join하고 B relation은 itemList, consist, product과 purchase relation들을 join한다.
- With 구문의 각각의 relation A, B에서 상품 별로 판매된 총 금액을 확인할 수 있도록 product_name으로 그룹화하고 sum()을 통해 상품 별 판매된 금액을 모두 합산하였다.
- 이후 relation A, B를 FULL OUTER JOIN 한 후 상품을 판매된 금액을 기준으로 내림차순 정렬하였다.



- 위는 TYPE 3-1 실행결과이다.

- **(TYPE 3-2) And then find the top 10% products by dollar-amount sold.**
  - TYPE 3-2 같은 경우는 TYPE 3-1과 같은 Query를 사용하였다. 상품 별 총 판매 금액 기준으로 내림차순으로 정렬된 값을 받아와 전체 tuple 개수의 10%에 해당하는 수만큼 출력하였다.



- 위는 TYPE 3-2의 실행결과이다.

- **(TYPE 4) Find all products by unit sales in the past year.**

```sql
with X as (
    with A as (
        select C1.product_name as 'aprod', sum(I1.quantity) as 'asum'
            from orders as O
            join itemList as I1 on O.order_id = I1.order_id
            join consist as C1 on I1.package_id = C1.package_id
            where O.order_date >= DATE_SUB(NOW(),INTERVAL 1 YEAR)
            group by C1.product_name
    ),
     B as (
        select C2.product_name as 'bprod', sum(I2.quantity) as 'bsum'
            from purchase as P
            join itemList as I2 on P.purchase_id = I2.purchase_id
            join consist as C2 on I2.package_id = C2.package_id
            where P.purchase_date >= DATE_SUB(NOW(),INTERVAL 1 YEAR)
            group by C2.product_name
    )
    select *
        from
        (select * from A left join B on A.aprod = B.bprod) as T
        union
        (select * from A right join B on A.aprod = B.bprod)
)
select coalesce(aprod,bprod), coalesce(asum,0)+coalesce(bsum,0) from X
;
```

- Order relation과 purchase relation이 itemList relation만 공유하면서 나뉘어 있으므로 SQL 구문의 with 구문에서 각각 join한 후 필요한 정보만 얻도록 하였다.
- A relation은 itemList, consist와 order relation들을 join하고 B relation은 itemList, consist와 purchase relation들을 join한다.
- A, B relation은 Consist relation의 product_name 속성으로 그룹화한 후 itemList의 quantity 속성으로 합산하여 상품 별 구매 개수를 나타내도록 하였다.
- A와 B relation을 FULL OUTER JOIN하여 최종 relation을 구한다.

```
---------------------------- TYPE 4 -----------------------------

** Find all products by unit sales in the past year. **

        ==============================
        | Product             Quantity |
        ==============================
        | Iphone X                  12 |
        | Magic Keyboard            17 |
        | Magic Mouse               17 |
        | iMac                      11 |
        | iPad Pro                  19 |
        | Mac                       11 |
        | ThinkStation              13 |
        | Gram                       6 |
        | Pebble                     4 |
        | ASUS PG27UQX               5 |
        | MX Ergo                   14 |
        | Pavilion                   7 |
        | ThinkPad                  17 |
        | ASUS PG32UQX              14 |
        | Sagaris                   12 |
        | Frontier                   4 |
        | Lenovo Legion              7 |
        | ThinkVision                7 |
        | Neo G7                     4 |
        | Galaxy S22                 9 |
        | Galaxy S8                  8 |
        ==============================
```

- 위는 TYPE 4의 실행결과이다.

- **(TYPE 4-1) Then find the top k products by unit sales.**

```
1  with X as (
2      with A as (
3          select C1.product_name as 'aprod', sum(I1.quantity) as 'asum'
4              from orders as O
5              join itemList as I1 on O.order_id = I1.order_id
6              join consist as C1 on I1.package_id = C1.package_id
7              where O.order_date >= DATE_SUB(NOW(),INTERVAL 1 YEAR)
8              group by C1.product_name
9      ),
10      B as (
11         select C2.product_name as 'bprod', sum(I2.quantity) as 'bsum'
12             from purchase as P
13             join itemList as I2 on P.purchase_id = I2.purchase_id
14             join consist as C2 on I2.package_id = C2.package_id
15             where P.purchase_date >= DATE_SUB(NOW(),INTERVAL 1 YEAR)
16             group by C2.product_name
17     )
18     select *
19         from
20         (select * from A left join B on A.aprod = B.bprod) as T
21         union
22         (select * from A right join B on A.aprod = B.bprod)
23 )
24 select coalesce(aprod,bprod), coalesce(asum,0)+coalesce(bsum,0) as 'total' from X
25 order by total desc
26 ;
```

- TYPE 4의 Query 구문에서 상품 별 판매 개수로 내림차순 정렬하는 구문만 24, 25번 줄에 추가하였다.

```
--------------------------- TYPE 4-1 ---------------------------

** Then find the top k products by unit sales. **

        Which k? : 5

        ====================================
        | #  Product               Quantity |
        ====================================
        | 1. iPad Pro                    19 |
        | 2. Magic Keyboard              17 |
        | 3. Magic Mouse                 17 |
        | 4. ThinkPad                    17 |
        | 5. MX Ergo                     14 |
        ====================================
```

- 위는 TYPE 4-1의 실행결과이다.

- **(TYPE 4-2) And then find the top 10% products by unit sales.**

  - TYPE 4-2 같은 경우는 TYPE 4-1과 같은 Query를 사용하였다. 상품 별 총 판매 개수 기준으로 내림차순으로 정렬된 값을 받아와 전체 tuple 개수의 10%에 해당하는 수만큼 출력하였다.

```
--------------------------- TYPE 4-2 ---------------------------

** And then find the top 10% products by unit sales. **

        Number of products sold: 21

        ====================================
        | #  Product               Quantity |
        ====================================
        | 1. iPad Pro                    19 |
        | 2. Magic Keyboard              17 |
        ====================================
```

- 위는 TYPE 4-2의 실행결과이다.

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

```sql
select *
from
store as A natural join store_stock as B
where A.region = 'California'
and B.quantity = 0
;
```

- Store와 Store_stock relation을 natural join한 후 지역과 재고 수로 선별하였다.

```
--------------------------- TYPE 5 ---------------------------

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

        ================================================
        | Store_ID    State         Product         Stock |
        ================================================
        | 50000001    California    ASUS PG32UQX        0 |
        | 50000001    California    Frontier            0 |
        | 50000001    California    ThinkStation        0 |
        | 50000013    California    Galaxy S22          0 |
        | 50000013    California    MX Ergo             0 |
        ================================================
```

- 위는 TYPE 5의 실행결과이다.

- **(TYPE 6) Find those packages that were not delivered within the promised time.**

```
select S.order_id, S.tracking_no, S.ship_comp, S.promised_date, S.delivered_date, P.name
from shipment as S join
itemList as I on I.order_id = S.order_id
join package as P on I.package_id = P.package_id
where  S.delivered_date is not null and
S.delivered_date > S.promised_date
;
```

- Shipment relation에서 delivered_date가 NULL이 아닌 tuple 중에서 delivered_date가 promised_date보다 늦은 tuple을 선별한다.

```
---------------------------- TYPE 6 -----------------------------
** Find those packages that were not delivered within the promised time. **
```

Order_ID	Tracking no.	Shipper	Promised_Date	Delivered_Date	Package
20000002	70000002	USPS	2021-08-09	2021-08-12	Lenovo desktop
20000004	70000004	FedEx	2022-01-02	2022-01-05	Desktop Set2
20000004	70000004	FedEx	2022-01-02	2022-01-05	Iphone single
20000007	70000007	USPS	2022-01-22	2022-01-24	Apple Home
20000012	70000012	USPS	2022-01-04	2022-01-06	Apple Edu
20000012	70000012	USPS	2022-01-04	2022-01-06	Gaming Package
20000004	70000004	FedEx	2022-01-02	2022-01-05	Lenovo monitor

- 위는 TYPE 6의 실행결과이다.

- **(TYPE 7) Generate the bill for each customer for the past month.**

```
with X as (
    with A as (
        select CS1.customer_id as 'acid', CS1.name as 'acus', sum(I1.quantity * PD1.cost) as 'asum', CS1.account_no as 'Aacc'
        from
        customer as CS1 join orders as O on CS1.customer_id = O.customer_id
        join itemList as I1 on O.order_id = I1.order_id
        join consist as C1 on I1.package_id = C1.package_id
        join product as PD1 on PD1.product_name = C1.product_name
        where O.order_date >= DATE_SUB(NOW(),INTERVAL 1 MONTH)
        and CS1.account_no is not null
        group by acid
    ),
    B as (
        select CS2.customer_id as 'bcid', CS2.name as 'bcus', sum(I2.quantity * PD2.cost) as 'bsum', CS2.account_no as 'Bacc'
        from
        customer as CS2 join purchase as P on CS2.customer_id = P.customer_id
        join itemList as I2 on P.purchase_id = I2.purchase_id
        join consist as C2 on I2.package_id = C2.package_id
        join product as PD2 on PD2.product_name = C2.product_name
        where P.purchase_date >= DATE_SUB(NOW(),INTERVAL 1 MONTH)
        and CS2.account_no is not null
        group by bcid
    )
    select *
        from
        (select * from A left join B on A.acid = B.bcid) as T
        union
        (select * from A right join B on A.acid = B.bcid)
) select coalesce(acid,bcid), coalesce(acus, bcus), coalesce(Aacc, Bacc) as acc_no, coalesce(asum,0)+coalesce(bsum,0) as 'total' from X ;
```

- TYPE 2의 Query문에서 customer relation에서 account_no가 NULL이 아닌 tuple을 선별하는 구문만 추가하였다.

```
---------------------------- TYPE 7 -----------------------------

** Generate the bill for each customer for the past month. **

        ================================================================
        |                          <<BILL>>                            |
        ================================================================
        | Customer_ID       Name           Account NO.          Price |
        ================================================================
        | 00000002          Jack           649-2567-2456-91     12550 |
        | 00000012          Liam           855-9075-2601-86      2700 |
        | 00000004          John           636-1996-3154-02      4245 |
        | 00000010          Armstrong      319-5315-0899-80      4060 |
        ================================================================
```

- 위는 TYPE 7의 실행 결과이다.

5. C Code

A. main

```c
int main(void) {

    if (!init_DB()) return 1;
    srand(time(NULL));

    while (1) {
        showTypes();    // print 7 types to choose
        int option = getInput();

        if (option == 0) break;

        switch (option) {
        case 1:
            type1(); break;
        case 2:
            type2(); break;
        case 3:
            type3(); break;
        case 4:
            type4(); break;
        case 5:
            type5(); break;
        case 6:
            type6(); break;
        case 7:
            type7(); break;
        }
    }
    deinit_DB();

    return 0;
}
```

i. init_DB 함수를 호출하여 데이터베이스 및 기본 세팅을 초기화한다. While 문을 돌려 계속하여 option을 입력 받을 수 있도록 한다.

ii. 마지막으로 deinit_DB() 함수를 호출하여 데이터베이스 테이블들을 DROP 한다.

B. init_DB

```
bool init_DB() {

    if (mysql_init(&conn) == NULL)
        printf("mysql_init() error!");

    connection = mysql_real_connect(&conn, host, user, pw, db, 3306, (const char*)NULL, 0);
    if (connection == NULL)
    {
        printf("%d ERROR : %s\n", mysql_errno(&conn), mysql_error(&conn));
        return false;
    }

    printf("Connection Succeed\n");

    if (mysql_select_db(&conn, db)) {
        printf("%d ERROR : %s\n", mysql_errno(&conn), mysql_error(&conn));
        return false;
    }

    fp = fopen("20160169.txt", "r");

    char buffer[BUF_LEN];

    fgets(buffer, BUF_LEN, fp);
    int lineNum = atoi(buffer);

    for (int i = 0; i < lineNum; i++) {
        memset(buffer, 0, sizeof(buffer));
        fgets(buffer, BUF_LEN, fp);
        int state = mysql_query(connection, buffer);

        if (state) {
            printf("ERROR: %s \n", buffer);
            fclose(fp);
            return false;
        }
    }

    return true;
}
```

i. mysql과 connection을 설정하고 20160169.txt 파일에서 CREATE와 INSERT SQL 구문들을 읽고 데이터베이스 테이블을 초기화한다.

C. deinit_DB

```
void deinit_DB() {

    while (feof(fp) == 0) {
        char buffer[BUF_LEN];
        fgets(buffer, BUF_LEN, fp);
        int state = mysql_query(connection, buffer);

        if (state) {
            printf("ERROR: %s \n", buffer);
            fclose(fp);
            return;
        }
    }

    fclose(fp);

    mysql_close(connection);
    printf("BYE! \n");
}
```

i. 20160169.txt 파일에서 DELETE 및 DROP SQL 구문들을 읽고 데이터베이스 테이블들을 모두 삭제한다.

ii. mysql과의 connection을 종료한다.

D. TYPE 관련 함수

```
void type1();
void type1_1(const char* shipper, const int order_id);
void type2();
void type2_1(char* c_id, char* c_name);
void type3();
void type3_1();
void type3_2();
void type4();
void type4_1();
void type4_2();
void type5();
void type6();
void type7();
```

i. 각 QUERY들에 대응하는 함수들을 따로 구현하여 13개의 쿼리를 처리하였다.

ii. 해당 함수들에 대해서는 C code에 주석으로 설명을 하였다.