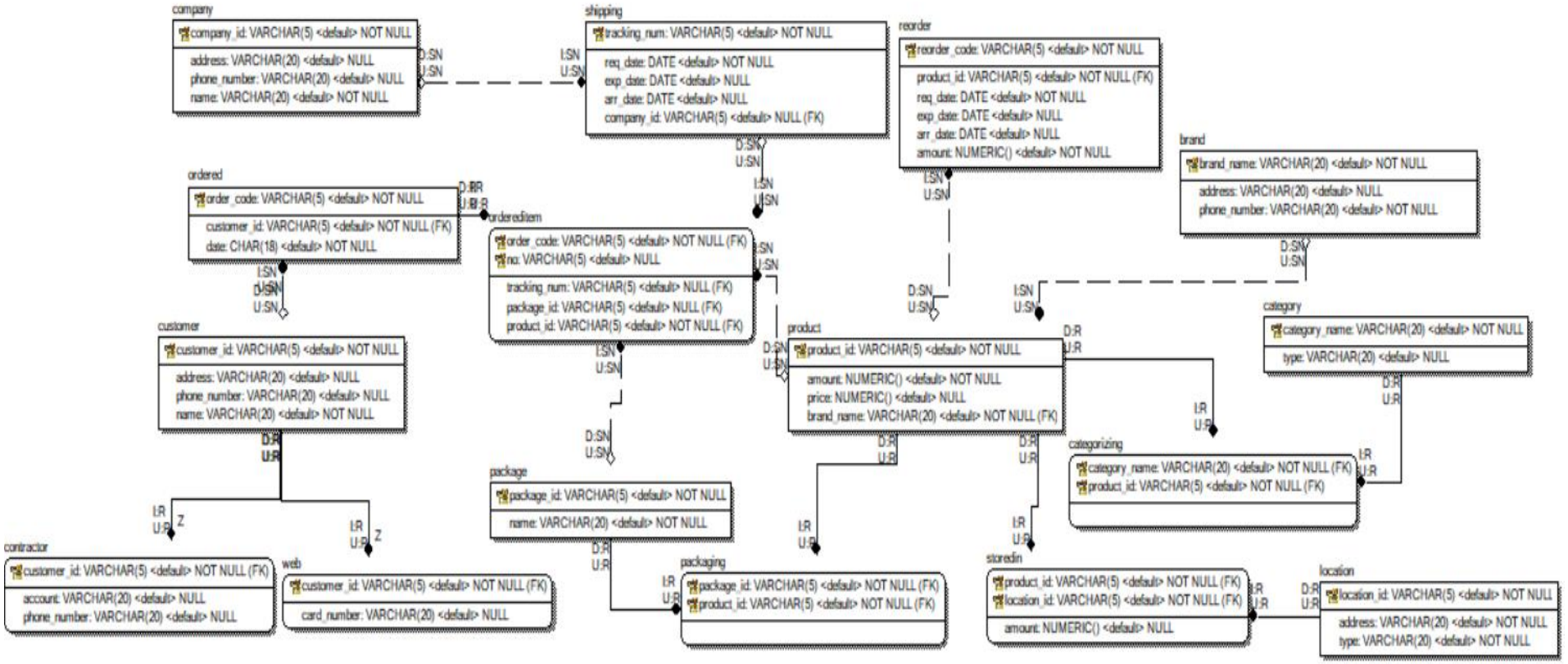
DATABASE SYSTEM

PROJECT 2

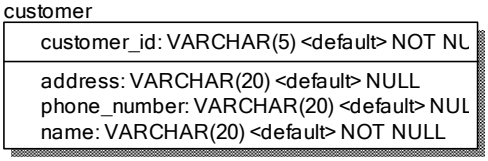
CSE4110-02

20160608 이지훈

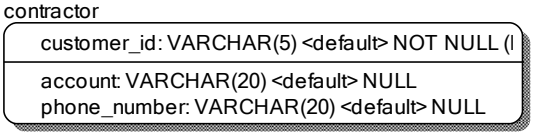

company
company_id: VARCHAR(5) <default> NOT NULL
address: VARCHAR(20) <default> NULL
phone_number: VARCHAR(20) <default> NULL
name: VARCHAR(20) <default> NOT NULL
shipping
tracking_num: VARCHAR(5) <default> NOT NULL
req_date: DATE <default> NOT NULL
exp_date: DATE <default> NULL
arr_date: DATE <default> NULL
company_id: VARCHAR(5) <default> NULL (FK)
reorder
reorder_code: VARCHAR(5) <default> NOT NULL
product_id: VARCHAR(5) <default> NOT NULL (FK)
req_date: DATE <default> NOT NULL
exp_date: DATE <default> NULL
arr_date: DATE <default> NULL
amount: NUMERIC() <default> NOT NULL
brand
brand_name: VARCHAR(20) <default> NOT NULL
address: VARCHAR(20) <default> NULL
phone_number: VARCHAR(20) <default> NOT NULL
ordered
order_code: VARCHAR(5) <default> NOT NULL
customer_id: VARCHAR(5) <default> NOT NULL (FK)
date: CHAR(18) <default> NOT NULL
orderedItem
order_code: VARCHAR(5) <default> NOT NULL (FK)
no: VARCHAR(5) <default> NULL
tracking_num: VARCHAR(5) <default> NULL (FK)
package_id: VARCHAR(5) <default> NULL (FK)
product_id: VARCHAR(5) <default> NOT NULL (FK)
customer
customer_id: VARCHAR(5) <default> NOT NULL
address: VARCHAR(20) <default> NULL
phone_number: VARCHAR(20) <default> NULL
name: VARCHAR(20) <default> NOT NULL
product
product_id: VARCHAR(5) <default> NOT NULL
amount: NUMERIC() <default> NOT NULL
price: NUMERIC() <default> NULL
brand_name: VARCHAR(20) <default> NOT NULL (FK)
category
category_name: VARCHAR(20) <default> NOT NULL
type: VARCHAR(20) <default> NULL
categorizing
category_name: VARCHAR(20) <default> NOT NULL (FK)
product_id: VARCHAR(5) <default> NOT NULL (FK)
package
package_id: VARCHAR(5) <default> NOT NULL
name: VARCHAR(20) <default> NOT NULL
packaging
package_id: VARCHAR(5) <default> NOT NULL (FK)
product_id: VARCHAR(5) <default> NOT NULL (FK)
storedin
product_id: VARCHAR(5) <default> NOT NULL (FK)
location_id: VARCHAR(5) <default> NOT NULL (FK)
amount: NUMERIC() <default> NULL
location
location_id: VARCHAR(5) <default> NOT NULL
address: VARCHAR(20) <default> NOT NULL
type: VARCHAR(20) <default> NOT NULL
contractor
customer_id: VARCHAR(5) <default> NOT NULL (FK)
account: VARCHAR(20) <default> NULL
phone_number: VARCHAR(20) <default> NULL
web
customer_id: VARCHAR(5) <default> NOT NULL (FK)
card_number: VARCHAR(20) <default> NULL

1. 데이터베이스 구성

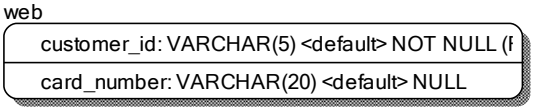
데이터 베이스를 구성하기 위해 사용한 tool 은 C++과 mysql 이다. 총 16 개의 Table 을 생성했으며 각 테이블별 구조는 다음과 같다.

<table>
<tr>
<td>customer
 

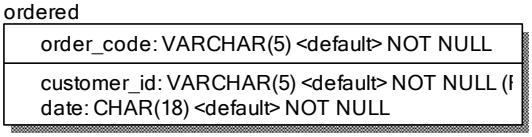
create table customer (customer_id varchar(5), name varchar(20) not null, address varchar(20), phone_number varchar(20), primary key (customer_id));

contractor
 

create table contractor (customer_id varchar(5), account varchar(20) not null, payment date, primary key (customer_id), foreign key (customer_id) references customer (customer_id) on delete cascade);

web
 

create table web (customer_id varchar(5), card_number varchar(20) not null, primary key (customer_id), foreign key (customer_id) references customer (customer_id) on delete cascade);</td>
<td>Description:
Customer 와 하위 수준 엔티티인 web 과 contractor 이다. 해당 엔티티는 고객 정보를 담고 있으며 web 과 contractor 를 구분한 이유는 고객 유형에 따라 Card Number 가 필요한지, Account(정기 계약 고객)가 필요한지 다르기 때문이다. 이 테이블을 통해 우리는 customer 이름 및 연락처와 주소를 알 수 있다.

Explanation about Physical Schema:
해당 스키마에서 Primary key 는 모두 customer_id 이다. 고객 정보는 customer_id 를 바탕으로 구분된다. 그리고 기본적으로 id 를 등록할 때, name 은 필수 요소로 설정해 Not null 로 지정했고 이외에는 주문정보가 없다면 등록하지 않아도 되도록 하기 위해 null 값을 허용했다.</td>
</tr>
<tr>
<td>ordered
 

create table ordered (order_code varchar(5), customer_id varchar(5) not null, date date, primary key (order_code), foreign key (customer_id) references customer (customer_id) on delete cascade);</td>
<td>Description:
주문이 발생할 때마다, 생성되는 ordered table 이다. 해당 테이블을 통해 우리는 주문에 해당하는 customer_id 와 날짜를 구할 수 있다.

Explanation about Physical Schema:
PK 는 ordered_code 로 설정했다. 그리고 주문 시 고객정보와 주문일이 없는 것은 가능하지 않기 때문에 Not null 로 설정했다.</td>
</tr>
</table>

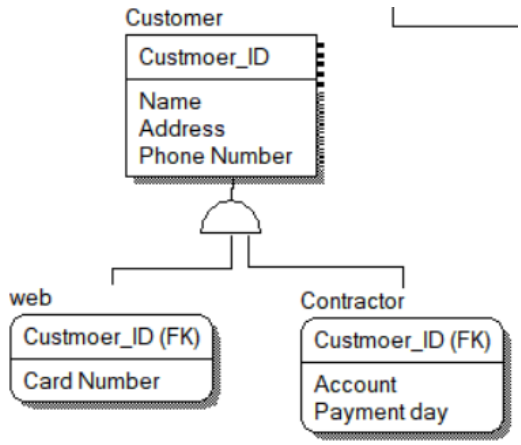
ordereditem order_code: VARCHAR(5) <default> NOT NULL (FK order_no: VARCHAR(5) <default> NULL tracking_num: VARCHAR(5) <default> NULL (FK) package_id: VARCHAR(5) <default> NULL (FK) product_id: VARCHAR(5) <default> NOT NULL (FK) create table ordereditem (order_code varchar(5), no varchar(5), product_id varchar(5) not null, package_id varchar(5), tracking_num varchar(5), primary key (order_code, no), foreign key (order_code) references ordered (order_code) on delete cascade, foreign key (product_id) references product (product_id) on delete cascade, foreign key (package_id) references package (package_id) on delete cascade, foreign key (tracking_num) references shipping (tracking_num) on delete cascade);	**Description:** 주문별 상품 정보를 담은 테이블이다. Order code 와 no 에 따라 product 와 package 여부, tracking number 가 달라진다. 일종의 장바구니 개념으로 보면 된다. **Explanation about Physical Schema:** PK 는 Order code 와 no 이고 product 없이는 주문이 발생하지 않으므로 Not null 로 설정했다. 운송장 번호의 경우 주문이 발생한 뒤 바로 배송 이 가능하지 않을수도 있기 때문에 위와 같이 설정했다.
shipping tracking_num: VARCHAR(5) <default> NOT NULL req_date: DATE <default> NOT NULL exp_date: DATE <default> NULL arr_date: DATE <default> NULL company_id: VARCHAR(5) <default> NULL (FK) create table shipping (tracking_num varchar(5), company_id varchar(5), req_date date not null, exp_date date, arr_date date, primary key (tracking_num), foreign key (company_id) references company (company_id) on delete cascade);	**Description:** 배송 정보를 담은 테이블이다. 각각 배송 요청일, 배송 도착 예정일, 실제 도착일이 표시되어 있다. 이 테이블을 이용해 우리는 각 주문의 배송현황을 알 수 있다. **Explanation about Physical Schema:** PK 는 Tracking number, 운송장 번호이다. 그리고 요청일은 배송을 등록하는 순간 발생하기 때문에 not null 로 설정했고 나머지의 값들은 배송 지연, 미입력 등으로 null 값이 발생할 수 있도록 했다.
product product_id: VARCHAR(5) <default> NOT NULL amount: NUMERIC() <default> NOT NULL price: NUMERIC() <default> NULL brand_name: VARCHAR(20) <default> NOT NULL (create table product (product_id varchar(5), brand_name varchar(20), amount numeric(10) not null, price numeric(10) not null, primary key (product_id), foreign key (brand_name) references brand (brand_name) on delete cascade);	**Description:** Product 의 전체 재고를 파악할 수 있는 product table 이다. 해당 테이블은 brand table 의 브랜드명을 FK 로 갖고 있으며 Amount 는 product 의 수량, price 는 가격을 표시한다. **Explanation about Physical Schema:** 가격은 미정일 수 있기 때문에 null 이 허용되도록 설정했다.

location location_id: VARCHAR(5) <default> NOT NULL address: VARCHAR(20) <default> NOT NULL type: VARCHAR(20) <default> NOT NULL create table location (location_id varchar(5), address varchar(20), type varchar(10), primary key (location_id));	**Description:** 상품을 보관하는 warehouse 와 판매하는 store 의 위치 및 정보가 담긴 table 이다 **Explanation about Physical Schema:** PK 는 location ID 로 설정했다. 장소가 존재하면 address 또한 필수적으로 입력되어야 하고, store 인지 warehouser 인지 구분이 가능한 type 또한 입력되어야 한다.
reorder reorder_code: VARCHAR(5) <default> NOT NUI product_id: VARCHAR(5) <default> NOT NULL (req_date: DATE <default> NOT NULL exp_date: DATE <default> NULL arr_date: DATE <default> NULL amount: NUMERIC() <default> NOT NULL create table reorder (reorder_code varchar(5), product_id varchar(5) not null, req_date date, exp_date date, arr_date date, amount numeric(10) not null, primary key (reorder_code), foreign key (product_id) references product (product_id) on delete cascade);	**Description:** 재고 주문 정보를 담은 테이블이다. 각각 주문 요청일, 배송 도착 예정일, 실제 도착일, 수량, 주문 상품 등이 담겨있다. 해당 테이블을 통해 우리는 재고 주문이 일어났는지, 재고 보충일 등을 알 수 있다. **Explanation about Physical Schema:** 재고 주문이므로 product 와 수량은 필수적으로 입력하게 했으며 요청일 또한 null 값을 허용하지 않게 했다.
brand brand_name: VARCHAR(20) <default> NOT NULL address: VARCHAR(20) <default> NULL phone_number: VARCHAR(20) <default> NULL create table brand (brand_name varchar(20), address varchar(20), phone_number varchar(20), primary key (brand_name));	**Description:** 상품의 브랜드를 담은 테이블이다. 상품을 만든 회사 관련 정보가 담겨있다. **Explanation about Physical Schema:** 상품별 브랜드 정보에서 address 와 phone_number 는 필수적인 정보가 아니기 때문에 null 을 허용했다.
package package_id: VARCHAR(5) <default> NOT N name: VARCHAR(20) <default> NOT NULL	**Description:** 묶음 판매가 가능한 패키지의 정보를 담은 테이블이다. **Explanation about Physical Schema:** 패키지명은 null 값을 허용하지 않도록 했다.
brand brand_name: VARCHAR(20) <default> NOT NULL address: VARCHAR(20) <default> NULL phone_number: VARCHAR(20) <default> NOT NULL	.**Description:** 택배 회사 정보를 담은 테이블이다. Tracking number 가 해당 테이블의 Company_ID 를 FK 로 갖는다

create table brand (brand_name varchar(20), address varchar(20), phone_number varchar(20), primary key (brand_name));	**Explanation about Physical Schema:** 상품별 브랜드 정보에서 재고 주문을 위한 phone_numbe 가 필수적인 정보이기 때문에 null 을 허용하지 않았다.
packaging package_id: VARCHAR(5) <default> NOT NULL product_id: VARCHAR(5) <default> NOT NULL (F create table packaging (package_id varchar(5), product_id varchar(5), primary key (package_id ,product_id), foreign key (product_id) references product (product_id) on delete cascade, foreign key (package_id) references package (package_id) on delete cascade);	**Description:** 패키지와 product 의 관계를 표현한 릴레이션이다. 이 테이블을 통해 우리는 패키지의 구성 상품이 무엇인지 알 수 있다. **Explanation about Physical Schema:** 다 대 다 관계이므로 Foreign Key 를 모두 Primary Key 로 갖는다.
category category_name: VARCHAR(20) <default> NOT N type: VARCHAR(20) <default> NULL create table category (category_name varchar(20), type varchar(20) not null, primary key (category_name));	**Description:** 카테고리별 분류 정보가 담겨 있다. 분류 정보는 monitor, TV 등 제품 종류에 관련된 데이터이다. **Explanation about Physical Schema:** 상품은 도착했으나 재고 처리는 실시하지 않은 경우가 있을 수 있어 Null 을 허용했다.
categorizing category_name: VARCHAR(20) <default> NOT NULL product_id: VARCHAR(5) <default> NOT NULL (FK) create table categorizing (category_name varchar(20), product_id varchar(5), primary key (category_name, product_id), foreign key (category_name) references category (category_name) on delete cascade, foreign key (product_id) references product (product_id) on delete cascade);	**Description:** category 와 product 의 관계를 표현한 릴레이션이다. 따라서 이를 통해 카테고리별 어떠한 상품이 포함되는 지 알 수 있다. **Explanation about Physical Schema:** 다 대 다 관계이므로 Foreign Key 를 모두 Primary Key 로 갖는다.
storedin product_id: VARCHAR(5) <default> NOT NULL (FK) location_id: VARCHAR(5) <default> NOT NULL (FK) amount: NUMERIC() <default> NULL create table storedin (location_id varchar(5), product_id varchar(5), amount numeric(4), primary key	**Description:** Location 과 product 의 관계를 표현한 릴레이션이다. 각 장소 별 보관하는 제품 정보와 그 수량이 담겨있다. **Explanation about Physical Schema:** 다 대 다 관계이므로 Foreign Key 를 모두 Primary Key 로 갖는다.

(location_id ,product_id), foreign key (product_id) references product (product_id) on delete cascade, foreign key (location_id) references location (location_id) on delete cascade);	

2. BCNF decomposition.



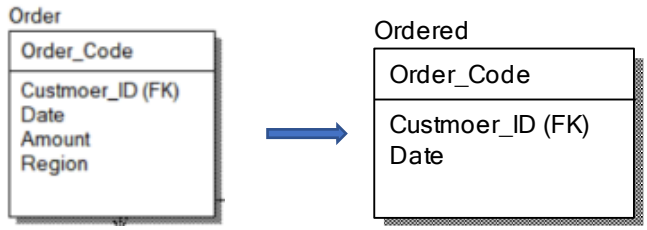
Functional Dependency:

customer_ID -> Name, address, phone number, card_number, account, payment day

해당 종속성이 있는 이유는 고객 고유의 ID 마다 이름, 주소, 그리고 연락처가 모두 다르기 때문이다. 따라서 ID 가 동일하다면 다른 정보 또한 동일해야 한다.

BCNF 과정

Customer_ID 는 primary key, 즉 super key 이므로 BCNF 에 위반하지 않는다고 판단하여 분해를 실시하지 않았다.



Functional Dependency:

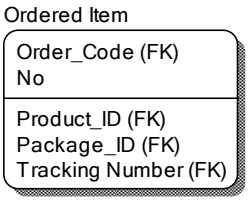
Order_code -> Date, amount, region, customer_ID

Customer_ID -> region

주문 코드가 동일하다면 주문일 및 수량, 지역이 같아야 하므로 첫 번째 종속성을 만족한다. 또한 고객이 같으면 지역이 동일하기 때문에 두 번째 종속성을 만족한다.

BCNF 과정

여기서 Order_code 는 super key 이기 때문에 따로 BCNF 를 실시하지 않았다. 다만 customer_ID 의 경우 super key 가 아니므로 분해가 필요했으나 해당 속성은 이미 customer 테이블에서 같은 정보를 추출할 수 있기 때문에 제거했다. 또한 amount 의 경우 ordered_item 에서 해당 정보를 추출할 수 있기 때문에 무결성을 해칠 우려가 있어 제거했다.



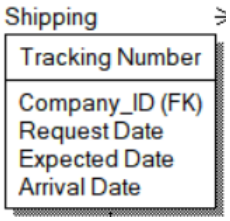
Functional Dependency:

Order_code, NO -> product_ID, package_ID, Tracking number

여기서 Ordered_code -> Tracking Number 로 표현할 수 없는 이유는 주문량이 많을 경우 배송이 여러 번에 걸쳐 일어날 수 있기 때문이다. 반면 Order code 와 no 가 같다면 상품 정보, 패키지 정보, 배송 정보가 동일해야 하기 때문에 첫 번째 종속성은 만족한다.

BCNF 과정

Order_code, ID 는 primary key 이므로 BCNF 를 위반하지 않는다.



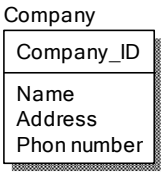
Functional Dependency:

Tracking number-> Company ID, Request Date, Expected Date, Arrival Date

운송장 번호에 따라 택배회사, 배송 관련 date 가 결정된다.

BCNF 과정

Tracking number 는 primary key 이므로 BCNF 를 위반하지 않는다



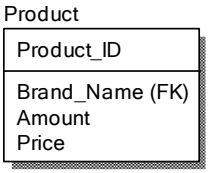
Functional Dependency:

Company ID -> Name, address, phone number

택배 회사 관련 정보는 ID 가 같다면 동일해야 한다.

BCNF 과정

ID 는 primary key 이므로 BCNF 를 위반하지 않는다



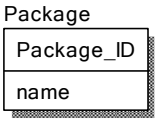
Functional Dependency:

Product_ID -> brand Name, price, amount

product ID 는 판매하는 상품마다 모두 다르고 이런 product_ID 가 같다면 동일한 상품이라는 의미이므로 brand 와 가격, 수량이 동일해야 한다.

BCNF 과정

ID 는 primary key 이므로 BCNF 를 위반하지 않는다



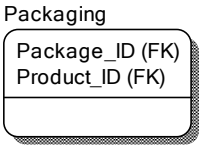
Functional Dependency:

Package ID -> Name

패키지 id 가 같다면 패키지명이 동일해야 한다.

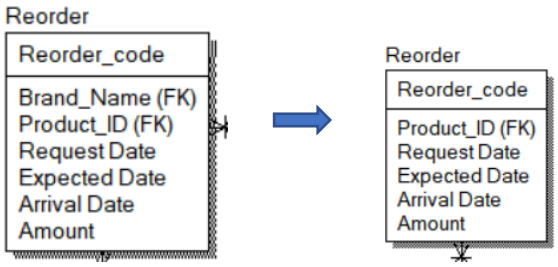
BCNF 과정

ID 는 PK 이므로 BCNF 를 위반하지 않는다.



Functional Dependency:

패키지 ID 가 같아도 product 는 다를 수 있고(여러 개로 구성), product ID 가 같아도 package ID 는 다를 수 있다(여러 개의 패키지에 속할 수 있음)



Functional Dependency:

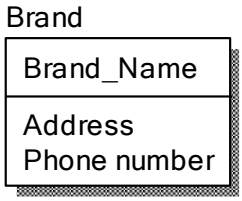
Reorder code-> Request Date, Expected Date, Arrival date, amount, product_ID

Product_ID -> Brand_name

Code 에 따라 재고 주문일, 재고 도착일, 예정일, 수량, 상품 번호 등이 결정된다. 그리고 product id 가 같다면 제조사도 동일해야 하므로 두 번째 종속성 또한 만족한다.

BCNF 과정

Reorder code 는 super key 이므로 BCNF 를 위반하지 않지만 product_ID 는 super key 나 trivial 한 depedency 가 아니므로 정규화를 실시했다. 그리고 해당 정보는 이미 product table 에 있어 테이블을 제거했다.



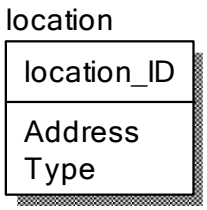
Functional Dependency:

Brand_name -> address, phone_number

제조사명이 같다면 제조사 관련 정보는 동일해야 한다.

BCNF 과정

Brand name 은 super key 이므로 위반하지 않는다.



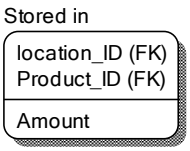
Functional Dependency:

Location_id-> address, Type

장소 코드가 동일하다면 관련 정보 또한 동일해야 한다.

BCNF 과정

Location_id 는 super key 이므로 위반하지 않는다.



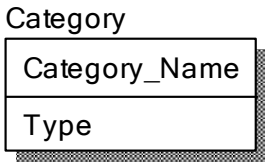
Functional Dependency

Location_id, product_id-> amount

장소마다 제품의 보관 수량이 다르고, 장소에는 여러 제품이 보관되어 있다. 하지만 한 장소에 보관되는 특정 제품은 수량 값이 여러 개일 수 없다.

BCNF 과정

Location_id, product_id 는 super key 이므로 위반하지 않는다.



Functional Dependency:

Category_name-> Type

Type 이 같아도 category name 은 다를 수 있지만 역은 성립하지 않는다. 즉, 카테고리명이 동일하면 type 또한 동일해야한다.

BCNF 과정

Category_name 은 super key 이므로 위반하지 않는다.

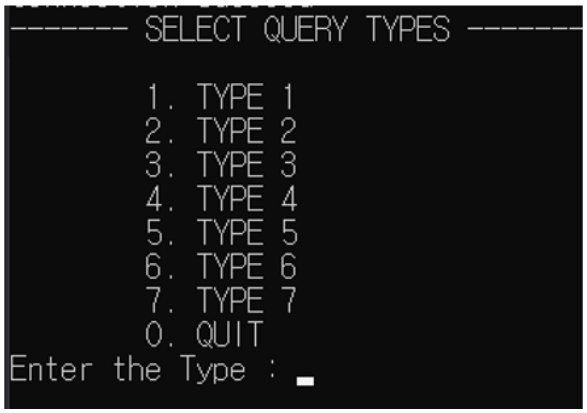
따라서 BCNF 를 위반하여 수정한 테이블은 다음과 같다.

Ordered, Reorder

이 외의 테이블은 모두 super key 에 해당하여 종속성을 위반하지 않았다.

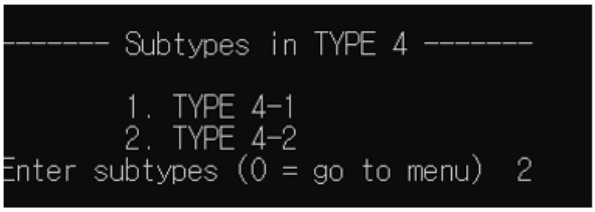
3. 기능 구현 및 설명

- 해당 목차에서는 각 화면 별 기능과 구현 방식에 대해 설명하겠다.



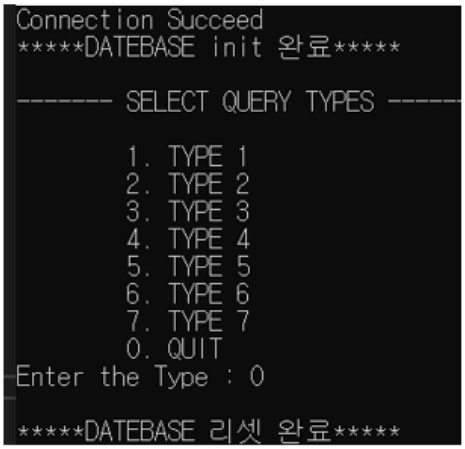
초기화면
Input type : number

초기화면으로, 0~7 이외의 값이 입력되면 같은 화면이 반복되도록 구성하였다.



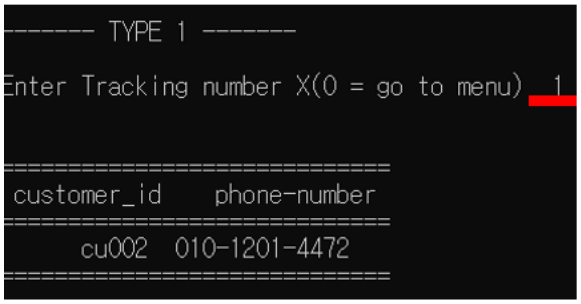
각 Type별 subtype

특정 기능에는 subtype이 존재한다. Subtype은 int 값으로 표시된 내용 이외의 값을 입력하면 같은 화면을 다시 출력한다.
또한 0을 입력할 경우 main menu로 돌아갈 수 있다.



시작 및 종료 화면

처음 시작 시 mysql ODBC를 통해 DB를 연결하고 query 문을 실행하여 DB를 구성한다.
그리고 Main menu에서 0을 입력할 경우, 입력한 데이터들을 모두 초기화한다.

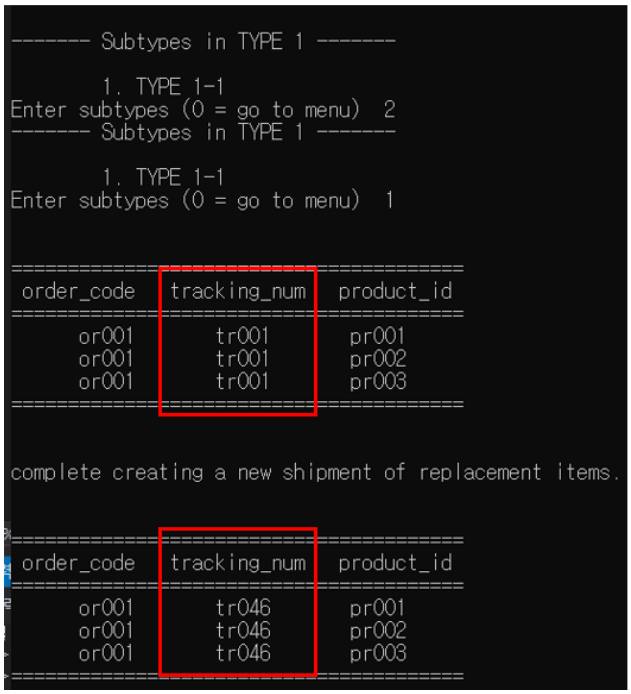


TYPE 1
요구사항 :
Assume the package shipped by USPS with tracking number X is reported to have been destroyed in an accident. Find the contact information for the customer.

Input type : (number)

기능 설명:
특정 사용자가 입력한 X에 따라 해당하는 customer 정보를 출력한다.

상세 설명:
X tracking number가 파손되었을 때, 파손 사실을 알리기 위해 고객 정보가 필요하다. 따라서 이용자에게 X를 입력 받고(해당 예에서는 1로 가정), tr001을 주문한 cu002 이용자의 phone_number를 표시했다.



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

Input type : (number)

기능 설명:
사용자가 서브 type을 입력하면 기존 주문서를 보여주고 shipping에 새로운 tuple을 추가한 뒤 변경사항을 알려준다.

상세 설명:

이제 파손된 상품의 정보를 확인하고 해당 상품을 다시 재배송해야 한다. 이를 위해 ordereditem table에서 입력 받은 X를 조건으로 하여 이용자에게 파손된 상품의 정보를 표시했다. 그리고 shipping table에 새로운 튜플을 이전까지의 tuple 개수를 카운트하여 새로운 id 값을 추가하고, 기존 주문의 tracking_num을 update한 뒤 업데이트 뒤 변경된 내용을 표시했다.
.

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

============================
 customer_id       amount
============================
      cu002          45000
============================
```

customer_id	amount
cu002	45000
cu001	14000
cu003	10000

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

Input type : 없음

기능 설명:
작년 기준 가장 구매 금액이 큰 고객을 표시한다.
Amount는 구매 금액이다.

상세 설명:
먼저 order를 2021년의 조건을 걸어 추출했다. 그 뒤 ordered item table에 product table을 join하여 제품별 금액을 추출한 다음 이를 고객 기준으로 그룹화했다.

```
       1. TYPE 2-1
Enter subtypes (0 = go to menu)  1



============================
 product_id       amount
============================
      pr003             4
============================
```

TYPE 2-1
요구사항 :
find the product that the customer bought the most.

Input type : number

기능 설명:
구매 금액이 가장 큰 고객의 최다 구매 상품을 표시한다.
Amount는 구매 금액이다.

상세 설명:
앞서 구한 customer_id를 이용해 해당 고객의 주문상품을 ordered item에서 추출한 뒤 그룹화하였다. 그 뒤 내림차순 정렬하여 최다 구매상품을 추출했다.

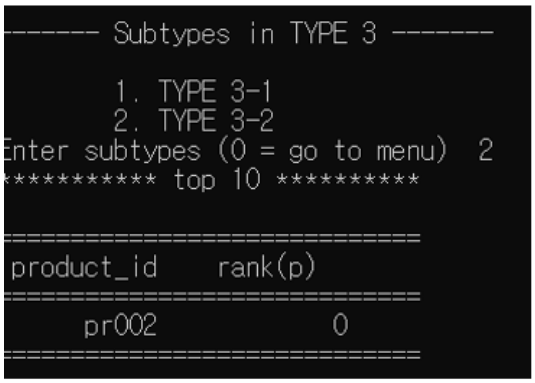
```
------- TYPE 3 -------


====================
 product_id
====================
      pr001
      pr002
      pr003
      pr004
====================
```

TYPE 3
요구사항 :
Find all products sold in the past year

Input type : 없음

기능 설명:
작년 기준 판매 상품 목록을 표시한다.

상세 설명:
Ordered item 테이블과 ordered table을 order code 기준으로 join한 뒤 작년 기준 조건을 걸었다. 그리고 distinct를 사용해 중복을 제거했다.

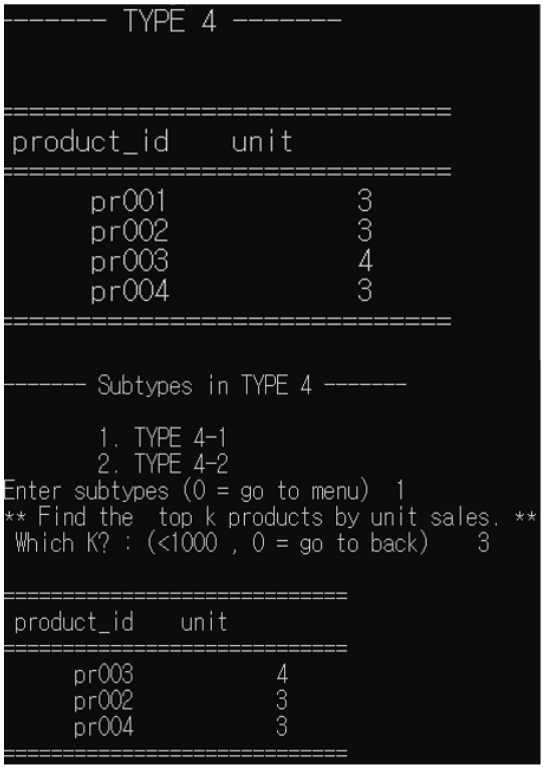


TYPE 3-2
요구사항 :
find the top 10% products by dollar-amount sold

Input type : 없음

기능 설명:
작년 기준 판매 상품을 금액 기준으로 상위 10%만큼 출력한다. Rank는 상위 기준 비율이다.

상세 설명:
ordered item 테이블에 order를 통해 날짜를 join했고 작년이라는 조건을 건 뒤 product를 기준으로 grouping 했다. 그리고 여기에 rank를 사용해 각 product별 비율을 구한 뒤 조건문을 이용했다.
해당 예에서 tuple 개수가 20개가 넘지 않아 1개의 tuple만이 결과로 나오게 되었다.



TYPE 4
요구사항 :
Find all products by unit sales in the past year.

Input type : 없음

기능 설명:
작년 기준 판매된 상품 id와 판매량을 표시했다.

상세 설명:
Ordered item에 ordered 테이블을 join해 날짜를 추출했고 이를 그룹화하여 product_id별로 count했다.

TYPE 4-1
요구사항 :find the top k products by unit sales

Input type : number

기능 설명:
작년 판매량 기준 상위 k개만큼 표시했다.

상세 설명:
해당 예시에서는 k는 3이다. Type 4에서 구한 테이블에서 내림차순 정렬을 한 뒤 limt을 이용해 k만큼 추출했다.

```
-------- Subtypes in TYPE 4 --------

        1. TYPE 4-1
        2. TYPE 4-2
Enter subtypes (0 = go to menu)  2

=========================================
 product_id          unit       rank
=========================================
      pr003            4            0
=========================================
```

TYPE 4-2
요구사항 :
Find the top 10% products by unit sales

Input type : 없음

기능 설명:
작년 판매량 기준 상위 10%를 표시한다.

상세 설명:
Type 4에서 구한 테이블에서 내림차순 정렬을 한 뒤 limt을 percent_rank()를 사용해 판매량 별 상위 퍼센트 비율을 구했다. 그리고 비율이 0.1보다 작거나 같을 경우 표시할 수 있도록 했다. 해당 예시에서는 tuple이 20개를 넘지 않아 1개만 출력되었다.

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

=============================================================
 product_id          amount       location_id          address
=============================================================
      pr001               0          lo001          California-123
      pr002               0          lo001          California-123
      pr003               0          lo001          California-123
      pr004               0          lo001          California-123
      pr005               0          lo001          California-123
      pr009               0          lo001          California-123
      pr010               0          lo001          California-123
      pr001               0          lo015          California-181
      pr002               0          lo015          California-181
      pr012               0          lo015          California-181
=============================================================
```

location_id	product_id	amount
lo001	pr001	0
lo001	pr002	0
lo001	pr003	0
lo001	pr004	20
lo001	pr005	20
lo001	pr006	10
lo001	pr007	30
lo001	pr008	20
lo001	pr009	20
lo001	pr010	0
lo001	pr011	30
lo001	pr012	0

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

Input type : 없음

기능 설명:
캘리포니아에 위치한 매장에서 재고가 없는 상품을 출력한다. Amount는 상품 재고량이며 location id는 매장번호이다.

상세 설명:
location 테이블에 각 매장 위치와 매장 종류(store, warehouse)가 나와있다. 따라서 이를 이용해 캘리포니아에 위치한 매장정보를 먼저 추출했다. 그리고 해당 테이블과 stored in을 join한 뒤 매장별 amount가 0인 product를 나열했다. 또한 매장 별 재고 부족 상품이 다르기 때문에 location id와 product id 모두 출력했다.

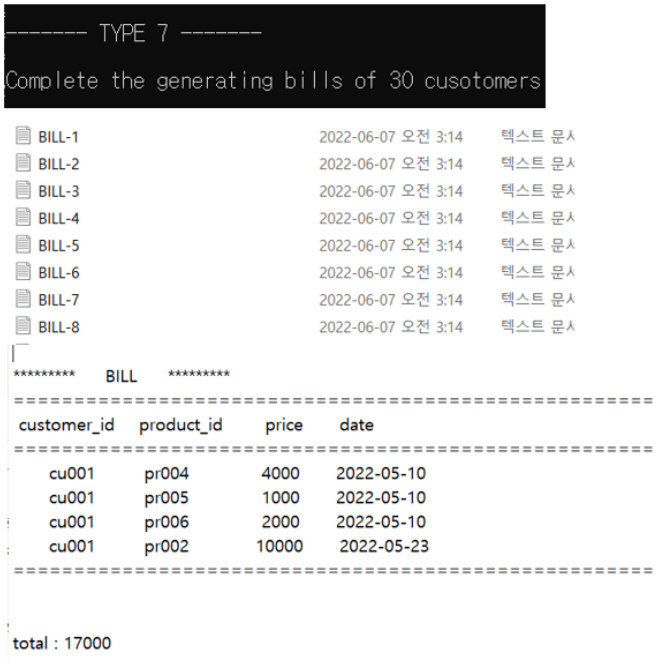
```
------ TYPE 6 -------

=================================================================
product_id     company_id      req_date       exp_date       arr_date
=================================================================
     tr012          co007    2022-05-10     2022-05-13     2022-05-15
     tr013          co001    2022-05-10     2022-05-13     2022-05-15
     tr014          co005    2022-05-11     2022-05-14     2022-05-20
=================================================================
```

TYPE 6
요구사항 :
Find those packages that were not delivered within the promised time

Input type : 없음

기능 설명: 지연된 배송의 product, 택배회사, 요청일, 예정일, 도착일을 보여준다. 도착일이 없더라도 예정일이 현재보다 앞서는 경우도 출력한다.

상세 설명:
Shipping table의 arr_date와 exp_date를 이용했다 만약 arr_date의 값이 exp_date보다 크다면 예상 도착일에 도착하지 못한 것으로 판단하여 추출했다. 또한 해당 예시에서는 없지만 arr_date가 null, 즉 아직 도착하지 않았지만 exp_date가 현재보다 작을 경우 또한 표시하도록 쿼리를 구성했다.



TYPE 7
요구사항 : Generate the bill for each customer for the past month

Input type : 없음
Output : txt 파일

기능 설명:
각 사용자별로 저번 달 기준 Bill을 txt 파일로 출력한다. 사용자 Id, 구매 내역, 가격 날짜, 총 구매 금액을 추출 했다

상세 설명:
저번 달 기준, 각 customer 별로 영수증을 생성했다. 각각의 사용자마다 bill을 출력하라고 하였으므로 전체 customer를 count한 뒤, 모든 고객의 Bill을 txt파일로 작성해 총 30개의 bill이 만들어졌다.
추출 과정은 Ordered item에 ordered 테이블을 join해 날짜를 추출했고 이를 그룹화하여 product_id 별로 count했다.

4. Function, Query

Int main(void)	DB INIT 및 초기화 수행 입력 값에 따라 TYPE 1~TYPE 7 수행 0 입력 시 종료.
Int printMenu()	MAIN TYPE 메뉴를 출력.
Int type1()	Type 1, Type 1-1 수행. 0 입력 시 종료
Int type2()	Type 2, Type 2-1 수행 0 입력 시 종료
Int type3()	Type 3, Type 3-1, Type 3-2 수행 0 입력 시 종료
Int type4()	Type 4, Type 4-1, Type 4-2 수행 0 입력 시 종료
Int type5()	Type 5 수행
Int type6()	Type 6 수행
Int type7()	Type 7 수행

Type1 `select C.customer_id, C.phone_number from customer C, (select distinct B.customer_id from (select * from ordereditem where tracking_num='tr%03d') A, ordered B where A.order_code = B.order_code) D where C.customer_id = D.customer_id;`	1. 먼저 order 를 customer 와 join 하여 order 를 기준으로 phone number 를 도출했다. 2. 그리고 이를 ordered item 테이블과 join 하여 파손된 tracking number 에 해당하는 customer 정보를 나열했고 tracking number X 에 대한 조건을 걸어 결과물을 도출했다.

Type 1-1
`insert into shipping values ('tr%03d', 'co001',Date_Format(now(), '%%Y-%%m-%%d'),null,null)`
`update ordereditem set tracking_num = 'tr%03d' where order_code = '%s'`
`select order_code, tracking_num, product_id from ordereditem where tracking_num = 'tr%03d'`
	1 번 쿼리의 경우, 오늘 날짜를 기준으로 shipping table 에 새로운 tuple 을 추가했다. 이 때, id 는 기존에 없는 것을 사용하기 위해 count(*) +1 을 이용했다.
	3 번 쿼리의 경우, 배송정보가 바뀌었으므로 기존 ordered item 의 tracking_num 을 업데이트했다.
	4 번 쿼리의 경우, 변경사항을 알려주기 위한 쿼리로, ordered item 에 tracking_number 를 조건으로 두었다.

Type 2 `select customer_id, sum(price) amount from (select order_code, B.price from ordereditem A, product B where A.product_id = B.product_id) C, ordered D where C.order_code=D.order_code and YEAR(date) = YEAR(CURRENT_DATE - INTERVAL 1 YEAR) group by customer_id order by sum(price) desc limit 1`	1. 먼저, 고객별 주문상품의 가격을 가져오기 위해 product table 과 ordered item talbe 을 join 했다. 2. 그리고 order table 과 join 하여 주문날짜를 가져왔고 주문날짜에 현재를 기준으로 2021 년 조건을 걸어 작년 데이터만 남도록 하였다. 3. 마지막으로 이를 가격 기준으로 sum 을 한 뒤 내림차순 정렬하고 가장 상단 tuple 을 가져오도록 했다.
Type 2-1 `with procnt as(select product_id, count(*) cnt from(select order_code from ordered where customer_id = '%s' and YEAR(date) = YEAR(CURRENT_DATE - INTERVAL 1 YEAR)) A, ordereditem B where A.order_code = B.order_code group by product_id) select * from procnt where cnt = (select max(cnt) from procnt)`	1. 해당 쿼리에서는 가상 테이블을 사용했다. 가상 테이블은 type 2 에서 구한 customer 를 기준으로 count 를 나열했다 2. 그리고 가상 테이블을 이용해 max(count)를 도출했고 이 값이 count 와 동일한 경우 표시되도록 구성했다.
Type 3 `select distinct product_id from ordereditem A , (select * from ordered where YEAR(date) = YEAR(CURRENT_DATE - INTERVAL 1 YEAR)) B where A.order_code = B.order_code;");`	1. 가장 먼저, order 테이블에서 2021 년 데이터를 추출했다. 2. 그 뒤 ordered item 테이블을 추출한 order code 를 기준으로 join 하고 distinct 를 사용해 중복을 제거했다.

Type 3-1 `select C.product_id, sum(D.price) amount from (select product_id from ordereditem A , (select * from ordered where YEAR(date) = YEAR(CURRENT_DATE - INTERVAL 1 YEAR)) B where A.order_code = B.order_code) C, product D where C.product_id =D.product_id group by C.product_id order by sum(D.price) desc limit %d;`	1. 앞서 type3 과 product_id 를 구한 것까지는 동일하나 여기에 product_id 를 기준으로 product table 을 join 하여 price 를 구했다. 2. 그 뒤 product_id 를 기준으로 sum(price) 그룹화를 실시했고 이를 다시 내림차순으로 정렬한 뒤 상위 k 개만큼 뽑을 수 있도록 했다.
Type 3-2 `select * from (select E.product_id, PERCENT_RANK() over (order by E.amount desc) as percent from (select C.product_id, sum(D.price) amount from (select product_id from ordereditem A , (select * from ordered where YEAR(date) = YEAR(CURRENT_DATE - INTERVAL 1 YEAR)) B where A.order_code = B.order_code) C, product D where C.product_id =D.product_id group by C.product_id) E) G where G.percent <=0.1;`	1. 해당 예에서도 ordered, ordered item, product 를 join 하여 작년 기준 판매된 product_id 와 price 를 구하는 것은 동일했다. 2. 그리고 product_id 를 기준으로 sum(price) 그룹화를 진행했고 여기에 PECENT_RANK()를 사용해 sum(price)를 기준으로 상위 퍼센트 비율을 도출했다 3. 그리고 이 퍼센트 값에 0.1 이라는 조건을 걸어 상위 10%를 표시했다.
Type 4 `select product_id, count(*) unit from ordereditem A , (select * from ordered where YEAR(date) = YEAR(CURRENT_DATE - INTERVAL 1 YEAR)) B where A.order_code = B.order_code group by product_id;`	1. 가장 먼저, ordered table 에서 2021 년의 조건을 걸어 2021 년 생성된 order 를 추출했다. 2. 그리고 이 추출된 테이블과 ordered item table 을 join 해 2021 년 판매된 상품들을 나열했고 3. 이를 다시 produc_id 를 기준으로 그룹화하여 count 를 표시했다.
Type 4-1 `select product_id, count(*) unit from ordereditem A , (select * from ordered where YEAR(date) = YEAR(CURRENT_DATE - INTERVAL 1 YEAR)) B where A.order_code = B.order_code group by product_id order by unit desc limit %d;`	1. 먼저 product_id 를 기준으로 count 를 구한 것은 type 4 와 동일하다. 2. 그리고 그 뒤 추출한 테이블에서 desc 정렬을 한 뒤 limit 을 걸어 총 k 개의 상품이 표시되도록 했다.
Type 4-2 `select * from (select E.product_id, E.unit ,PERCENT_RANK() over (order by E.unit desc) as percent from (select product_id, count(*) unit from ordereditem A , (select * from ordered where YEAR(date) = YEAR(CURRENT_DATE - INTERVAL 1 YEAR)) B where A.order_code = B.order_code group by product_id) E) G where G.percent <=0.1`	1. 마찬가지로 product_id 별 count 를 구하는 것까지는 동일하다 2. 그러나 그 뒤, Percent_rank() 함수를 내림차순으로 정렬하여 unit 기준 상위 10 percent 를 추출했다.
Type 5 `select distinct A.product_id, A.amount, A.location_id, address from storedin A,(select * from location where address like 'California%%' and type = 'store') B where A.location_id = B.location_id and amount = 0;`	1. store 테이블에 주소 및 매장 구분 정보가 있어 먼저 해당 테이블에서 california 이면서 store 인 data 들을 추출했다. 여기서 like 를 이용해 california 가 주소에 포함된다면 추출될 수 있게 구성했다.

	2. 그리고 이를 stored in 과 join 하여 california 내 매장들의 amount 를 추출했고 이 값이 0 인 경우 표시되도록 했다.
Type 6 `select * from shipping where exp_date < arr_date or (arr_date is null and exp_date < now());`	1. shipping table 에 도착 예정일과 도착일 정보가 모두 나와있어 조건문만 사용했다. 2. 첫 번째 조건은 exp_date< arr_date, 다시 말해 예정일보다 도착일이 늦을 경우 표시하도록 했고 3. 두 번째 조건은 Null 이지만 예정일이 현재보다 빠른 경우 표시하도록 했다.
Type 7 `select customer_id, product_id, price, date from (select order_code, A.product_id, B.price from ordereditem A, product B where A.product_id = B.product_id) C, ordered D where C.order_code=D.order_code and date between '2022-05-01' and '2022-05-31' and customer_id = 'cu%03d'`	1. 앞서 본 쿼리문과 유사하게 product table 과 ordered item table 을 join 하여 product_id 별 금액을 표시했고 ordered item 과 orderd 테이블을 join 하여 customer 별 order_code 를 얻을 수 있도록 했다. 2. 그리고 앞서 추출한 두 테이블을 order_code 를 기준으로 join 하였고 customer_id 를 조건으로 지정하여 고객별 BILL 을 표시하도록 했다.