

# Database System

## Project2

서강대학교 컴퓨터공학과

20191637

임수민

# 1. Decompose relation schema into BCNF

relation schema 에서 중복되는 정보를 줄이기 위해 relation schema 을 정규화하여 BCNF 로 decompose 하는 과정을 수행한다.

## 1.1 customer

customer(customer\_ID, name, online\_id, online\_password, online\_sign\_up\_date, email, phone\_number, address, birth\_date)

Functional dependencies:

- customer\_id -> name, online\_id, phone\_number
- online\_id -> online\_password, online\_sign\_up\_date, email, address, birth\_date

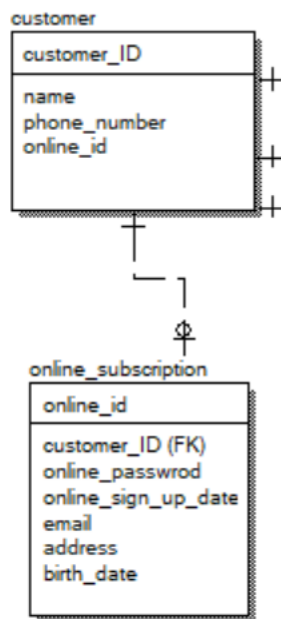
online\_id -> online\_password, online\_sign\_up\_date, email, address, birth\_date 는 trivial 한 functional dependency 가 아니고, online\_id 는 superkey 가 아니므로 customer 릴레이션은 BCNF 가 아니다. 따라서 BCNF decomposition 을 수행해야 한다.

BCNF decomposition 을 수행하여 customer 릴레이션을 customer 릴레이션과 online\_subscription 릴레이션으로 분해한다.

① customer(customer\_ID, name, phone\_number, online\_id)

② online\_subscription(online\_id, online\_password, online\_sign\_up\_date, email, address, birth\_date)

이 때 online\_subscription 릴레이션이 가지는 nontrivial functional dependency 는 online\_id -> online\_password, online\_sign\_up\_date, email, address, birth\_date 이고, {online\_id}는 superkey 이므로 online\_subscription 릴레이션은 BCNF 이다. 마찬가지로 customer 릴레이션이 가지는 nontrivial functional dependency 는 customer\_ID -> name, online\_id, phone\_number 이고, customer\_ID 는 superkey 이므로 customer 릴레이션도 BCNF 이다.



customer 릴레이션은 온라인, 오프라인 고객의 공통 정보를 저장하는 릴레이션이고, online\_subscription 는 온라인 사이트에 가입한 고객의 정보를 저장하는 릴레이션이다. 고객 중 온라인 사이트에서 가입하지 않은 고객이 있을 수 있고, 한 고객마다 한 개의 온라인 가입 정보만을 저장하므로, One to Zero or One Cardinality 으로 설정하였다. 또한 online\_id 만으로 Entity 를 구분할 수 있기 때문에 Non-Identifying 관계로 설정하였으며, 온라인 가입 정보는 항상 고객 정보를 가져야 하기 때문에 No Nulls 로 설정하였다.

## 1.2 card

card(card\_number, card\_company, customer\_ID, expiry\_date)

Functional dependencies:

- card\_number, card\_company -> customer\_ID, expiry\_date

{card\_number, card\_company}가 superkey 이므로 card 릴레이션은 BCNF 이다.

## 1.3 bank\_account

bank\_account(account\_number, bank\_name, customer\_ID, account\_name, balance)

Functional dependencies:

- account\_number, bank\_name -> customer\_ID, account\_name, balance

{account\_number, bank\_name}가 superkey 이므로 bank\_account 릴레이션은 BCNF 이다.

## 1.4 purchase

purchase(order\_date, order\_ID, customer\_ID, total\_price)

Functional dependencies:

- order\_date, order\_ID, customer\_ID -> total\_price

{order\_date, order\_ID, customer\_ID}가 superkey 이므로 purchase 릴레이션은 BCNF 이다.

## 1.5 contract\_of\_payment

contract\_of\_payment(customer\_ID, order\_date, order\_ID, account\_number, bank\_name)

trivial 한 functional dependency 밖에 없으므로 contract\_of\_payment 릴레이션은 BCNF 이다.

## 1.6 bill

bill(bill\_number, account\_number, bank\_name, order\_date, order\_ID, customer\_ID, bill\_date, due\_date, monthly\_charge, total\_charge)

Functional dependencies:

- bill\_number, account\_number, bank\_name, order\_date, order\_ID, customer\_ID -> bill\_date, due\_date, monthly\_charge, total\_charge

{bill\_number, account\_number, bank\_name, order\_date, order\_ID, customer\_ID}가 superkey 이므로 bill 릴레이션은 BCNF 이다.

## 1.7 delivery

delivery(tracking\_number, company\_name, order\_date, order\_ID, customer\_ID, sending\_date, promised\_delivery\_date, delivery\_date, delivery\_address, completed, issue, shipper\_name)

Functional dependencies:

- tracking\_number, company\_name -> order\_date, order\_ID, customer\_ID, sending\_date, promised\_delivery\_date, delivery\_date, delivery\_address, completed, issue, shipper\_name

{tracking\_number, company\_name}가 superkey 이므로 delivery 릴레이션은 BCNF 이다.

## 1.8 shipping\_company

shipping\_company(company\_name, CEO, email, phone\_number, address)

Functional dependencies:

- company\_name -> CEO, email, phone\_number, address

{company\_name}가 superkey 이므로 shipping\_company 릴레이션은 BCNF 이다.

## 1.9 sales\_data

sales\_data(sales\_ID, order\_date, order\_ID, customer\_ID, product\_ID, manufacturer\_name, total\_amount, total\_sales, warehouse\_ID, store\_ID, region)

Functional dependencies:

- sales\_ID -> order\_date, order\_ID, customer\_ID, product\_ID, manufacturer\_name, total\_amount, total\_sales, warehouse\_ID, store\_ID, region

{sales\_ID}가 superkey 이므로 sales\_data 릴레이션은 BCNF 이다.

## 1.10 product\_type

product\_type(type)

trivial 한 functional dependency 밖에 없으므로 product\_type 릴레이션은 BCNF 이다.

## 1.11 product

product(product\_ID, manufacturer\_name, type, product\_name, model, price)

Functional dependencies:

- product\_ID, manufacturer\_name -> type, product\_name, model, price

{product\_ID, manufacturer\_name}가 superkey 이므로 product 릴레이션은 BCNF 이다.

## 1.12 store

store(store\_ID, region, owner, phone\_number, address, opening\_date)

Functional dependencies:

- store\_ID, region -> owner, phone\_number, address, opening\_date

{store\_ID, region}가 superkey 이므로 store 릴레이션은 BCNF 이다.

## 1.13 warehouse

warehouse(warehouse\_ID, manager, phone\_number, address)

Functional dependencies:

- warehouse\_ID -> manager, phone\_number, address

{warehouse\_ID}가 superkey 이므로 warehouse 릴레이션은 BCNF 이다.

### 1.14 offline\_inventory

offline\_inventory(store\_ID, region, manufacturer\_name, product\_ID, stock)

Functional dependencies:

- store\_ID, region, manufacturer\_name, product\_ID -> stock

{store\_ID, region, manufacturer\_name, product\_ID}가 superkey 이므로 offline\_inventory 릴레이션은 BCNF 이다.

### 1.15 online\_inventory

online\_inventory(warehouse\_ID, manufacturer\_name, product\_ID, stock)

Functional dependencies:

- warehouse\_ID, manufacturer\_name, product\_ID -> stock

{warehouse\_ID, manufacturer\_name, product\_ID}가 superkey 이므로 online\_inventory 릴레이션은 BCNF 이다.

### 1.16 offline\_reorder

offline\_reorder(reorder\_ID, store\_ID, region, product\_ID, manufacturer\_name, quantity, reorder\_date, promised\_arrival\_date, arrival\_date, completed, issue)

Functional dependencies:

- reorder\_ID, store\_ID, region, manufacturer\_name, product\_ID -> quantity, reorder\_date, promised\_arrival\_date, arrival\_date, completed, issue

{reorder\_ID, store\_ID, region, manufacturer\_name, product\_ID}가 superkey 이므로 offline\_reorder 릴레이션은 BCNF 이다.

### 1.17 online\_reorder

online\_reorder(reorder\_ID, warehouse\_ID, product\_ID, manufacturer\_name, quantity, reorder\_date, promised\_arrival\_date, arrival\_date, completed, issue)

Functional dependencies:

- reorder\_ID, warehouse\_ID, manufacturer\_name, product\_ID -> quantity, reorder\_date, promised\_arrival\_date, arrival\_date, completed, issue

{reorder\_ID, warehouse\_ID, manufacturer\_name, product\_ID}가 superkey 이므로 online\_reorder 릴레이션은 BCNF 이다.

### 1.18 manufacturer

manufacturer(manufacturer\_name, CEO, email, phone\_number, address)

Functional dependencies:

- manufacturer\_name -> CEO, email, phone\_number, address

{manufacturer\_name}가 superkey 이므로 manufacturer 릴레이션은 BCNF 이다.

## 2. Physical Schema Diagram

relation schema 를 정규화하여 BCNF 의 형태로 logical schema diagram 을 만든 후 Physical Schema diagram 을 생성한다. 이 때 Physical Schema Diagram 은 data types, domain, constraints, relationship type, allowing nulls 를 보여주어야 한다.

### 2.1 domain 설정

#### - ID

Number / Integer / Not Null / Validation Constraint X

전자 제품 판매 회사에서 관리하는 Entity 를 구분할 때 쓰이는 ID 의 Domain 이다. 1 이상의 양의 정수의 값을 가지는 고유한 ID 를 나타내야 하므로 Validation Constraint 로 NaturalNumber 를 추가하였으며 다음과 같다.

#### - Name

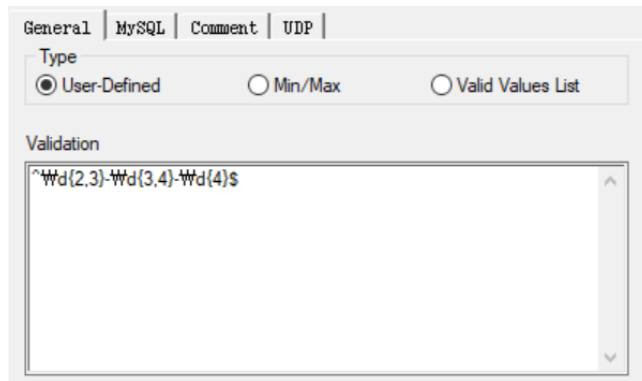
String / VARCHAR(20) / Not Null / Validation Constraint X

전자 제품 판매 회사에서 관리하는 모든 이름 정보에 관한 Domain 이다. 이름이 없는 경우 데이터 관리를 할 수 없으므로 Not Null 이다.

## - PhoneNumber

String / VARCHAR(20) / Allowing Nulls / Validation Constraint O

고객, 제조사, 운송 회사, 지점, 창고의 연락처정보를 저장하기 위한 Domain 이다. 연락처정보가 없을 수 있으므로 Null 을 허용하며, 전화번호는 일정 형식이 있으므로 Validation Constraint 으로 PhoneNumber 을 추가했으며 아래와 같다.



General | MySQL | Comment | UDP |

Type

☒ User-Defined    ☐ Min/Max    ☐ Valid Values List

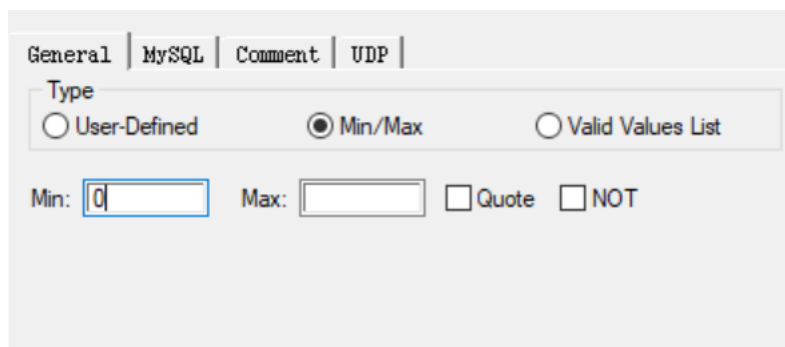
Validation

`^Wd(2,3)-Wd(3,4)-Wd(4)$`

## - Price

Number / Integer / Not Null / Validation Constraint O

전자 제품 판매 회사에서 관리하는 모든 가격 정보에 관한 Domain 이다. 가격이 없는 물건은 없으니 Not Null 이고, 가격은 음수가 될 수 없으므로 Validation Constraint 으로 UnsignedInt 를 추가했으며 다음과 같다.



General | MySQL | Comment | UDP |

Type

☐ User-Defined    ☒ Min/Max    ☐ Valid Values List

Min:     Max:     ☐ Quote    ☐ NOT

## - Charge

Number / Integer / Not Null / Validation Constraint O

전자 제품 판매 회사에서 관리하는 모든 청구 금액 정보에 관한 Domain 이다. 청구 금액이 없는 청구서는 없으니 Not Null 이고, 가격은 음수가 될 수 없으므로 Validation Constraint 으로 UnsignedInt 를 추가했으며 다음과 같다.



General | MySQL | Comment | UDP |

Type

☐ User-Defined ☒ Min/Max ☐ Valid Values List

Min:  Max:  ☐ Quote ☐ NOT

## - Quantity

Number / Integer / Not Null / Validation Constraint O

전자 제품 판매 회사에서 관리하는 모든 수량 정보에 관한 Domain 이다. 수량이 없는 물건은 없으니 Not Null 이고, 수량은 음수가 될 수 없으므로 Validation Constraint 으로 UnsignedInt 를 추가하였으며 다음과 같다.

General | MySQL | Comment | UDP |

Type

☐ User-Defined ☒ Min/Max ☐ Valid Values List

Min:  Max:  ☐ Quote ☐ NOT

## - DeliveryCompletion

Number / Integer / Not Null / Validation Constraint O

배송 완료 여부 정보를 위한 Domain 이다. 배송 완료 여부 정보는 배송이 완료되지 않은 경우, 배송이 예정 시간에 맞게 완료된 경우, 예정 배송 시간보다 늦게 완료된 경우, 배송이 실패한 경우 중 하나에 무조건 해당되어야 하므로 Not Null 이며, 0, 1, 2, 3으로 간단히 구분할 수 있도록 한다. Validation Constraint 으로 DeliveryCompletion 을 추가했으며 다음과 같다.

General | MySQL | Comment | UDP |

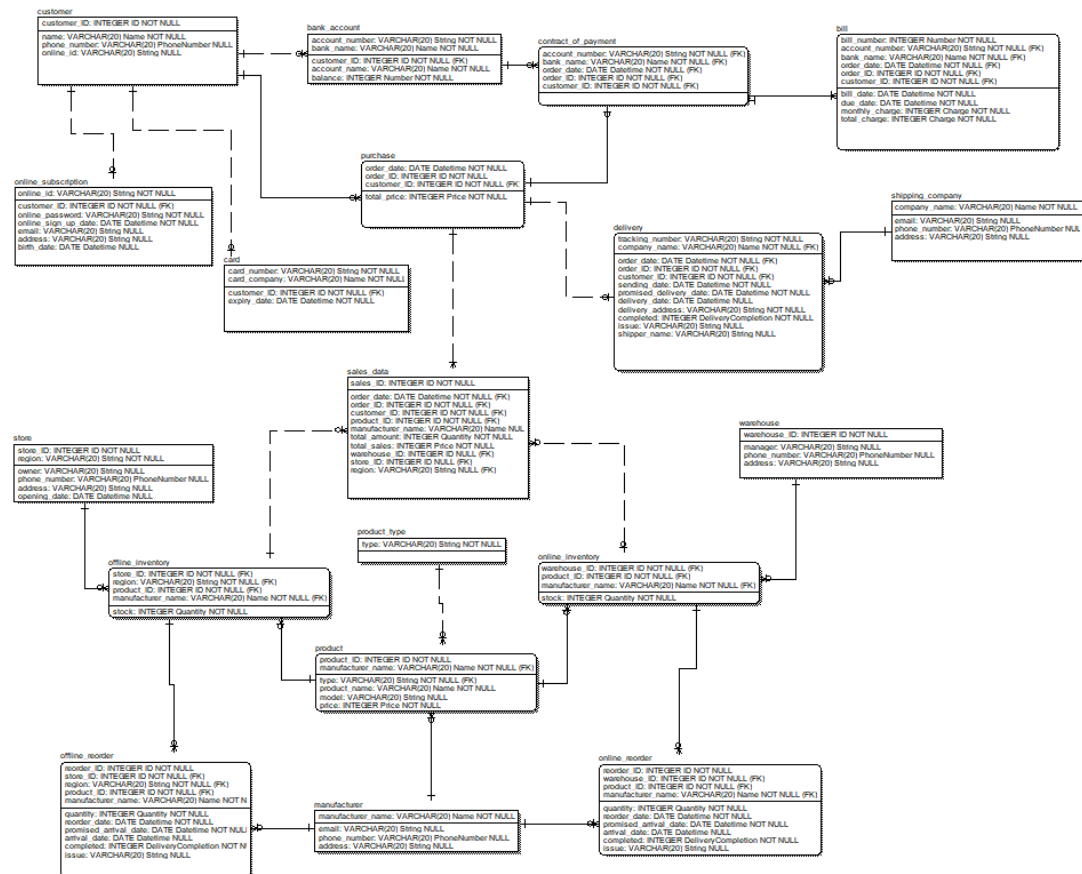
Type

☐ User-Defined ☐ Min/Max ☒ Valid Values List

Valid Value ☐ Quote ☐ NOT

	Valid Value	Display Value	Definition
	0	not completed	not completed
	1	delivery complet	delivery complete
	2	delivery delayed	delivery complete after its promise
	3	failed	failed

## 2.2 전체 구현 모습



## 3. Queries & Code implementation

MYSQL DBMS 와 Visual Studio 2019를 사용하여 위의 데이터베이스 모델을 구현하고 쿼리를 수행하는 애플리케이션 프로그램을 작성한다. MYSQL DBMS 와 애플리케이션 프로그램을 연결하기 위해 MYSQL Connector(ODBC)을 사용한다. 애플리케이션 프로그램은 여러 타입의 쿼리를 수행하기 위한 인터페이스를 제공한다.

프로그램 사용법과 프로그램을 사용하기 위해 미리 설치해야 하는 프로그램은 README 파일을 참고하면 된다. README 파일의 위치는 다음과 같다.

```
20191637.zip ----- Project2 ----- 20191637.cpp  
  
    |_ Project2.sln        |_ 20191637.txt  
  
    |_ README.txt         |_ libmysql.dll  
  
                            |_ Project2.vcxproj  
  
                            |_ Project2.vcxproj.filters  
  
                            |_ Project2.vcxproj.user
```

### 3.1 connection 생성

먼저 visual studio 에서 만든 C++ 프로그램과 MYSQL DBMS 사이의 connection 을 생성해야 한다. 코드는 다음과 같다.

```
#define _CRT_SECURE_NO_WARNINGS
...
#include "mysql.h"
...

#pragma comment(lib, "libmysql.lib")

const char* host = "localhost";
const char* user = "root";
const char* pw = "mysql";
const char* db = "project";

int main(void) {

    MYSQL* connection = NULL;
    MYSQL conn;
    MYSQL_RES* sql_result;
    MYSQL_ROW sql_row;

    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 1;
    }

    else
    {
        printf("Connection Succeed\n\n");

        if (mysql_select_db(&conn, db))
        {
            printf("%d ERROR : %s\n", mysql_errno(&conn), mysql_error(&conn));
            return 1;
        }

        ...
    }
}
```

위의 애플리케이션 프로그램을 실행하기 전에 MYSQL Workbench 를 통해 Hostname 이 "localhost", Username 은 "root", Password 는 "mysql"인 MYSQL

connection 을 생성해야 하고, 해당 커넥션과 연결된 MYSQL 서버에 "project" 데이터베이스가 생성되어 있어야 한다. 생성 과정은 README 파일에 설명하였다.

### 3.2 CRUD 파일

MYSQL 서버의 project 데이터베이스와 connection 이 성공적으로 생성되었으면, 프로그램은 CRUD 파일(20191637.txt)에 있는 create, insert 문을 데이터베이스에 보낸다. 이 때 table(relation)당 tuple 은 최소 15개 이상씩 있어야 하므로 한 테이블 당 삽입해야하는 데이터가 15개 이상이어야 한다. 그 후 아래의 쿼리를 수행하는 애플리케이션 프로그램이 작동되고, 애플리케이션 프로그램이 종료될 때 CRUD 파일에 있는 delete 문과 drop 문을 수행하여 데이터와 테이블을 모두 지운 다음 프로그램을 종료한다. 즉, 테이블의 생성과 삭제, 데이터의 삽입과 삭제는 CRUD 파일(20191637.txt)을 통해 각각 create, drop, insert, delete 문을 데이터베이스에 보내는 형식으로 수행한다.

#### ① Create

총 19개의 table(relation)을 생성하는 create 문을 MYSQL Server 에 보낸다.

#### ② Insert

위의 create 문으로 생성된 table 에 데이터를 삽입한다. 한 테이블 당 최소 15개의 tuple(데이터)를 삽입한다.

##### - customer

웹 사이트에 가입한 온라인 고객과 가입하지 않은 매장 고객 두 종류의 고객 데이터가 있어야 하므로, 15개의 온라인 고객 데이터와 온라인 사이트에 가입하지 않은 4개의 매장 고객 데이터를 삽입한다.

##### - online\_subscription

온라인 사이트에 가입한 15명의 고객에 해당하는 온라인 가입 정보 데이터 15개를 삽입한다.

##### - card

온라인 고객의 카드 정보만 저장할 수 있고 매장 고객의 카드 정보는 저장하지 않으므로, 각각의 온라인 고객에 해당하는 카드 데이터 15개를 삽입한다.

##### - bank\_account

온라인 사이트에 가입한 15명의 고객과 온라인 사이트에 가입하지 않은 4명의 매장 고객 총 19명의 고객에 대한 계좌 데이터를 삽입한다.

- **product\_type**

총 15개의 제품 종류 데이터를 삽입한다.

- **manufacturer**

총 16개의 제조사 데이터를 삽입한다.

- **product**

15개의 제품 종류 데이터와 16개의 제조사 데이터를 조합하여 총 24개의 제품 데이터를 삽입한다.

- **store**

15개의 오프라인 매장 데이터를 삽입한다.

- **warehouse**

15개의 창고 데이터를 삽입한다.

- **offline\_inventory**

15개의 오프라인 매장에 각각 3개의 제품 재고 정보가 있도록 총 45개의 데이터를 삽입한다.

- **online\_inventory**

15개의 창고에 각각 3개의 제품 재고 정보가 있도록 총 45개의 데이터를 삽입한다.

- **offline\_reorder**

오프라인 매장이 생성하는 총 15개의 reorder 데이터를 삽입한다.

- **online\_reorder**

창고가 생성하는 총 15개의 reorder 데이터를 삽입한다.

- **purchase**

19명의 고객에 대해 각각 1개씩 구매(주문) 데이터가 있도록 하여 총 19개의 구매 데이터를 삽입한다.

- **sales\_data**

고객이 한 번 주문할 때마다 제품을 3개씩 구매한다고 가정하여 19개의 구매(주문) 데이터와 연관된 각 제품의 판매 정보 데이터 총 57개를 삽입한다.

- **shipping\_company**

15개의 운송회사 데이터를 삽입한다.

- **delivery**

온라인으로 구매한 15개의 구매 데이터와 연관된 15개의 배송 데이터를 삽입한다.

- **contract\_of\_payment**

웹 사이트로 주문한 모든 주문(구매)에 대해 계좌로 구매 금액을 청구하여 샀다고 가정하여, 15개의 구매 데이터에 대한 15개의 지불 계약서 데이터를 삽입한다.

- **bill**

15개의 지불 계약서 데이터 모두 2달에 나누어서 청구한다고 가정하고, 그 중 1개는 아직 1번 더 청구할 청구 금액이 남은 것으로 가정하여(이후 TYPE 7 쿼리에서 자세히 설명) 총 29개의 청구서 데이터를 삽입한다.

### ③ Delete

19개의 테이블에 담긴 모든 데이터를 삭제한다.

### ④ Drop

19개의 모든 테이블을 삭제한다.

## 3.3 쿼리 수행

### (1) 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.

USPS 회사에서 배송 추적 번호가 X 인 패키지가 사고가 났다고 가정했을 때, 이 패키지를 주문한 고객의 연락처 정보를 찾는 쿼리는 다음과 같다.

```
select name, phone_number from delivery natural join customer where tracking_number='X' and company_name='USPS';
```

delivery 테이블에서 tracking\_number 은 'X'이고 company\_name 은 'USPS'인 튜플을 찾고, customer 테이블과 natural join 연산을 하여 고객의 이름과 전화번호를 구할 수 있다. 출력 결과는 다음과 같다.

```
---- 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.**

customer      phone_number
+-----+-----+
Gold          010-1515-1616
```

### ① TYPE 1-1

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

TYPE 1의 패키지의 정보를 찾는 쿼리는 다음과 같다.

```
select * from delivery natural join customer where tracking_number='X' and company_name='USPS';
```

delivery 테이블에서 tracking\_number 은 'X'이고 company\_name 은 'USPS'인 튜플을 찾아 패키지 정보를 찾고, customer 테이블과 natural join 연산을 하여 고객의 정보도 같이 찾을 수 있다. 출력 결과는 다음과 같다.

```
---- TYPE 1-1 ----
** Then find the contents of that shipment and create a new shipment of replacement items.**
tracking_number  shipping_company  order_date  customer  sending_date  promised_delivery_date  delivery_address  issue
+-----+-----+-----+-----+-----+-----+-----+-----+
X                USPS          2022-05-01  Gold      2022-05-01   2022-05-03             B101 street      accident
Success to create a new shipment!
```

그 다음 새로운 패키지를 생성하여 보내야 한다. 위의 쿼리를 보내 얻은 결과를 이용하여 데이터를 삽입하는 insert 문을 생성하여 쿼리를 보낸다. 생성된 쿼리는 다음과 같다.

```
insert into delivery (tracking_number, company_name, order_date, order_ID, customer_ID, sending_date, promised_delivery_date, delivery_address, completed)
values ('sql_row[1]', 'sql_row[2]', 'sql_row[3]', 'sql_row[4]', 'sql_row[0]', 'sql_row[5]', 'sql_row[6]', 'sql_row[8]', 0);
```

sql\_row[1], sql\_row[2], sql\_row[3], sql\_row[4], sql\_row[0], sql\_row[5], sql\_row[6], sql\_row[8]은 각각 TYPE 1의 쿼리로부터 얻은 tracking\_number, company\_name, order\_date, order\_ID, customer\_ID, sending\_date, promised\_delivery\_date,



delivery\_address, completed 값이다. 이 때 새로 생성된 패키지의 tracking\_number 는 원래 패키지와 다르게 해야 하므로 원래 패키지의 tracking\_number 문자열에 '1'을 붙여서 새로운 tracking\_number 로 설정하여 보낸다.

## (2) TYPE 2

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

지난 1년동안 가장 많이 (가격 기준) 구매한 고객을 찾는 쿼리는 다음과 같다.

```
with past_year_purchase as (select * from purchase where order_date >= '2021-01-01'),
customer_sum_price(customer_ID, sum_price) as (select customer_ID, sum(total_price) as
sum_price from past_year_purchase group by customer_ID)
```

```
select customer_ID, name, phone_number, online_id, sum_price from
customer_sum_price natural join customer where sum_price = (select max(sum_price)
from customer_sum_price);
```

현재로 2022년으로 가정한다면 2021년부터 지금까지의 데이터를 추출해야 하므로 purchase 테이블에서 order\_date 가 '2021-01-01'보다 큰 데이터를 추출해서 임시의 테이블 past\_year\_purchase 에 저장한다. 그 다음 past\_year\_purchase 테이블에서 customer\_ID 로 그룹을 지어 각 고객별로 구매한 금액의 총액을 저장하는 임시의 customer\_sum\_price 테이블을 생성한다. 그리고 customer\_sum\_price 임시 테이블과 customer 테이블을 natural join 연산과 where 절을 사용해 가장 많이 (가격 기준) 구매한 고객의 데이터를 구한다.

출력 결과는 다음과 같다.

---- TYPE 2 ----

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

customer	phone_number	online_id	total amount
Bourikas	010-1111-1212	kkkkk	15000

### ① TYPE 2-1

Then find the product that the customer bought the most.

지난 1년동안 가장 많이 (가격 기준) 구매한 고객이 가장 많이 산 제품을 찾는 쿼리는 다음과 같다.

```
with max_price_customer as (select * from sales_data where customer_ID=sql_row[0]),
```

```
product_amount as(select product_ID, manufacturer_name, sum(total_amount) as
product_total_amount from max_price_customer group by product_ID,
manufacturer_name)
```

```
select product_name, manufacturer_name, type, price, product_total_amount from
product natural join product_amount where product_total_amount = (select
max(product_total_amount) from product_amount);
```

위의 sql\_row[0]은 이전 TYPE 2의 쿼리에서 얻은 고객 ID 를 뜻한다. sales\_data 테이블에서 이 고객 ID 가 구매한 모든 데이터를 max\_price\_customer 임시 테이블에 저장하고, max\_price\_customer 을 product\_ID, manufacturer\_name 으로 그룹화하여 각 제품마다 팔린 총 수량을 저장하는 product\_amount 임시 테이블에 저장한다. 그 다음 product 와 product\_amount 테이블의 natural join 연산과 where 절을 사용해 해당 제품의 정보를 구한다.

출력 결과는 다음과 같다.

```
---- TYPE 2-1 ----
```

```
** Then find the product that the customer bought the most.**
```

product	manufacturer	type	price	number of buying products
NanoCell	LG	TV	1000	6

### (3) TYPE 3

Find all products sold in the past year.

지난 1년 동안 팔린 모든 제품들을 찾는 쿼리는 다음과 같다.

```
with product_sold as (select product_ID, manufacturer_name from sales_data where
order_date >= '2021-01-01'),
```

```
product_sold_group as(select * from product_sold group by product_ID,
manufacturer_name)
```

```
select product_name, manufacturer_name, type, price from product_sold_group natural
join product;
```

현재를 2022년으로 가정한다면 2021년부터 지금까지의 데이터를 추출해야 하므로 sales\_data 테이블에서 order\_date 가 '2021-01-01'보다 큰 데이터를 추출해서 임시의 테이블 product\_sold 에 저장한다. 그 다음 product\_sold 테이블에서 product\_ID 와 manufacturer 로 그룹을 지어 저장하는 임시의 product\_sold\_group 테이블을 생성한다. 그리고 product\_sold\_group 임시 테이블과 product 테이블을 natural join 연산을 통해 제품의 데이터를 구한다.

출력 결과는 다음과 같다.

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

product	manufacturer	type	price
iPhone 13	Apple	Cell Phones	100
Powerbeats Pro	Beats by Dr. Dre	Headphones	200
Egg Cooker	Bella	Kitchen Appliances	50
SelectTech 552	Bowflex	Fitness	500
EOS R6	Canon	Cameras	2000
Qualcomm Snapdragon	HP	Laptops	300
Classic 10692	Lego	Toys	50
Celeron N4500	Lenovo	Laptops	300
NanoCell	LG	TV	1000
Leg Strap	Nintendo	Video Games	50
Digital Scale	Omron	Health	50
Video Baby Monitor	Owlet	Baby Care	200
Galaxy S20	SamSung	Cell Phones	100
Outdoor Rock	Sonance	Speakers	300
Roam Smart Portable	Sonos	Speakers	200
Google TV	Sony	TV	1000
iPhone 12	Apple	Cell Phones	100
UltraWide Monitor	LG	Computers	200
Galaxy S21	SamSung	Cell Phones	100
iPhone 11	Apple	Cell Phones	100
Galaxy S22	SamSung	Cell Phones	100
iPad Pro	Apple	Tablets	300
Neo QLED	SamSung	TV	1000

### ① TYPE 3-1

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

지난 1년 동안 팔린 모든 제품들 중 가장 많이 팔린 (가격 기준) 제품 순대로 구하는 쿼리는 다음과 같다.

```
with product_sold as (select product_ID, manufacturer_name, total_sales from sales_data  
where order_date >= '2021-01-01'),
```

```
product_sold_group as(select product_ID, manufacturer_name, sum(total_sales) as  
sum_total_sales from product_sold group by product_ID, manufacturer_name order by  
sum_total_sales desc)
```

```
select product_name, manufacturer_name, type, price, sum_total_sales from  
product_sold_group natural join product;
```

과정은 위의 TYPE 3과 같지만 다른 점은 product\_sold\_group 임시 테이블을 생성할 때, 각 제품마다 팔린 총 금액을 저장하는 sum\_total\_sales 속성을 추가하고, sum\_total\_sales 속성의 내림차순으로 테이블을 생성하여 가장 많이

팔린 (가격 기준) 제품 순대로 정렬되도록 한다는 점이다. 그리고 product\_sold\_group 임시 테이블과 product 테이블을 natural join 연산을 통해 제품의 데이터를 구한다. 제품의 데이터를 출력할 때는 사용자가 입력한 k 개만큼 데이터를 출력하거나, 지난 1년동안 팔린 제품들의 종류 수가 k 보다 작으면 지난 1년동안 팔린 제품들 모두를 출력한다.

K 를 5로 입력했을 때 출력 결과는 다음과 같다.

```

---- TYPE 3-1 ----
** Then find the top k products by dollar-amount sold.**
Which K? : 5

```

rank	product	manufacturer	type	price	dollar-amount sold
1	EOS R6	Canon	Cameras	2000	9200
2	Neo QLED	SamSung	TV	1000	7600
3	NanoCell	LG	TV	1000	7000
4	Google TV	Sony	TV	1000	7000
5	SelectTech 552	Bowflex	Fitness	500	4500

## ② TYPE 3-2

And then find the top 10% products by dollar-amount sold.

입력하는 쿼리는 위의 TYPE 3-1과 같다.

```

with product_sold as (select product_ID, manufacturer_name, total_sales from sales_data
where order_date >= '2021-01-01'),

```

```

product_sold_group as(select product_ID, manufacturer_name, sum(total_sales) as
sum_total_sales from product_sold group by product_ID, manufacturer_name order by
sum_total_sales desc)

```

```

select product_name, manufacturer_name, type, price, sum_total_sales from
product_sold_group natural join product;

```

제품의 데이터를 출력할 때는 위의 TYPE 3에서 구한 튜플의 개수를 구하여 지난 1년 동안 팔린 제품의 종류 가짓수를 구하고, 이 가짓수의 10%만큼만 데이터를 출력한다.

출력 결과는 다음과 같다.

```

---- TYPE 3-2 ----
** And then find the top 10% products by dollar-amount sold.**

```

rank	product	manufacturer	type	price	dollar-amount sold
1	EOS R6	Canon	Cameras	2000	9200
2	Neo QLED	SamSung	TV	1000	7600

## (4) TYPE 4

Find all products by unit sales in the past year.

지난 1년 동안 unit sales 단위로 팔린 모든 제품들을 구하는 쿼리는 다음과 같다.

```
with product_sold as (select product_ID, manufacturer_name from sales_data where
order_date >= '2021-01-01'),
```

```
product_sold_group as(select * from product_sold group by product_ID,
manufacturer_name)
```

```
select product_name, manufacturer_name, type, price from product_sold_group natural
join product;
```

현재를 2022년으로 가정한다면 2021년부터 지금까지의 데이터를 추출해야 하므로 sales\_data 테이블에서 order\_date 가 '2021-01-01'보다 큰 데이터를 추출해서 임시의 테이블 product\_sold 에 저장한다. 그 다음 product\_sold 테이블에서 product\_ID 와 manufacturer 로 그룹을 지어 저장하는 임시의 product\_sold\_group 테이블을 생성한다. 그리고 product\_sold\_group 임시 테이블과 product 테이블을 natural join 연산을 통해 제품의 데이터를 구한다.

출력 결과는 다음과 같다.

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

product	manufacturer	type	price
iPhone 13	Apple	Cell Phones	100
Powerbeats Pro	Beats by Dr. Dre	Headphones	200
Egg Cooker	Bella	Kitchen Appliances	50
SelectTech 552	Bowflex	Fitness	500
EOS R6	Canon	Cameras	2000
Qualcomm Snapdragon	HP	Laptops	300
Classic 10692	Lego	Toys	50
Celeron N4500	Lenovo	Laptops	300
NanoCell	LG	TV	1000
Leg Strap	Nintendo	Video Games	50
Digital Scale	Omron	Health	50
Video Baby Monitor	Owlet	Baby Care	200
Galaxy S20	SamSung	Cell Phones	100
Outdoor Rock	Sonance	Speakers	300
Roam Smart Portable	Sonos	Speakers	200
Google TV	Sony	TV	1000
iPhone 12	Apple	Cell Phones	100
UltraWide Monitor	LG	Computers	200
Galaxy S21	SamSung	Cell Phones	100
iPhone 11	Apple	Cell Phones	100
Galaxy S22	SamSung	Cell Phones	100
iPad Pro	Apple	Tablets	300
Neo QLED	SamSung	TV	1000

① TYPE 4-1

Then find the top k products by unit sales.

지난 1년 동안 팔린 모든 제품들 중 가장 많이 팔린 (unit sales 기준) 제품 순대로 구하는 쿼리는 다음과 같다.

```
with product_sold as (select product_ID, manufacturer_name, total_amount from sales_data where order_date >= '2021-01-01'),
```

```
product_sold_group as(select product_ID, manufacturer_name, sum(total_amount) as sum_total_number from product_sold group by product_ID, manufacturer_name order by sum_total_number desc)
```

```
select product_name, manufacturer_name, type, price, sum_total_number from product_sold_group natural join product;
```

과정은 위의 TYPE 4과 같지만 다른 점은 product\_sold\_group 임시 테이블을 생성할 때 각 제품마다 팔린 총 수량을 저장하는 sum\_total\_number 속성을 추가하고, sum\_total\_number 속성의 내림차순으로 테이블을 생성하여 가장 많이 팔린 (unit sales 기준) 제품 순대로 정렬되도록 한다는 점이다. 그리고 product\_sold\_group 임시 테이블과 product 테이블을 natural join 연산을 통해 제품의 데이터를 구한다. 제품의 데이터를 출력할 때는 사용자가 입력한 k 개만큼 데이터를 출력하거나 지난 1년동안 팔린 제품들의 종류 수가 k 보다 작으면 지난 1년동안 팔린 제품들 모두를 출력한다.

K 를 8로 입력했을 때 출력 결과는 다음과 같다.

```
---- TYPE 4-1 ----
** Then find the top k products by unit sales.**
Which K? : 8
```

rank	product	manufacturer	type	price	number of unit sales
1	EOS R6	Canon	Cameras	2000	10
2	Celeron N4500	Lenovo	Laptops	300	10
3	Video Baby Monitor	Owlet	Baby Care	200	10
4	Roam Smart Portable	Sonos	Speakers	200	10
5	Neo QLED	SamSung	TV	1000	10
6	Egg Cooker	Bella	Kitchen Appliances	50	9
7	SelectTech 552	Bowflex	Fitness	500	9
8	Qualcomm Snapdragon	HP	Laptops	300	9

## ② TYPE 4-2

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

입력하는 쿼리는 위의 TYPE 4-1과 같다.

```
with product_sold as(select product_ID, manufacturer_name, total_amount from sales_data where order_date >= '2021-01-01'), product_sold_group as(select product_ID, manufacturer_name, sum(total_amount) as sum_total_number from product_sold group by product_ID, manufacturer_name order by sum_total_number desc) select
```

```
product_name, manufacturer_name, type, price, sum_total_number from
product_sold_group natural join product;
```

제품의 데이터를 출력할 때는 위의 TYPE 4에서 구한 튜플의 개수를 구하여 지난 1년 동안 팔린 제품의 종류 가짓수를 구하고, 이 가짓수의 10%만큼만 데이터를 출력한다.

출력 결과는 다음과 같다.

```

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

```

rank	product	manufacturer	type	price	number of unit sales
1	EOS R6	Canon	Cameras	2000	10
2	Celeron N4500	Lenovo	Laptops	300	10

## (5) TYPE 5

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

California 에 있는 모든 상점에서 재고가 없는 모든 제품들을 찾는 쿼리는 다음과 같다.

```
select product_name, manufacturer_name, type, price from offline_inventory natural join
product where region = 'California' and stock = 0;
```

offline\_inventory 테이블에서 지역이 'California'이고 stock(재고 수)는 0인 제품들을 찾는 다음, product 테이블과 natural join 연산을 통해 제품의 정보를 구한다.

출력 결과는 다음과 같다.

```

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

```

product	manufacturer	type	price
iPhone 13	Apple	Cell Phones	100
Galaxy S20	Samsung	Cell Phones	100
Neo QLED	Samsung	TV	1000

## (6) TYPE 6

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

제시간에 도착하지 못한 패키지를 찾는 쿼리는 다음과 같다.

```
select company_name, tracking_number, order_date, name, delivery_date,
delivery_address, shipper_name from delivery natural join customer where completed=2;
```

delivery 테이블에서 제시시간에 도착하지 못한 패키지의 completed 값은 2로 설정해 놓기 때문에 delivery 테이블에서 completed 가 2인 패키지의 정보를 찾는다.

출력 결과는 다음과 같다.

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

shipping_company	tracking_number	order_date	buyer	delivery_date	delivery_address	postman
LLLL	01	2021-05-01	Srinivasan	2021-05-05	B101 street	Kim
PPPP	01	2021-05-01	Tanaka	2021-05-05	B101 street	Kim
SSSS	01	2021-05-01	Bourikas	2021-05-05	B101 street	Kim
UUUU	01	2021-05-01	Aoi	2021-05-05	B101 street	Kim
YYYY	01	2021-05-01	Wu	2021-05-05	B101 street	Kim

## (7) TYPE 7

Generate the bill for each customer for the past month.

지난 달(past month)을 '2022-06'으로, 현재 달은 '2022-07'로 가정한다. 먼저 지난 달에 대한 청구서를 생성해야 하는 청구서 정보를 얻는 쿼리는 다음과 같다.

```
select * from bill where due_date >= '2022-06-30';
```

due\_date 가 '2022-06-30'보다 크면 지난 달에 대한 청구서를 생성해야 하므로, 해당되는 청구서에 대한 정보를 얻어 데이터를 삽입해야 한다. 새로운 청구서 데이터를 삽입하는 쿼리는 다음과 같다.

```
insert into bill values (새로운 bill_number, 'sql_row[1]', 'sql_row[2]', 'sql_row[3]',
sql_row[4], sql_row[5], '2022-06-30', 'sql_row[7]', sql_row[8], sql_row[9]);
```

sql\_row[1], sql\_row[2], sql\_row[3], sql\_row[4], sql\_row[5], sql\_row[7], sql\_row[8], sql\_row[9]은 각각 이전 쿼리로부터 얻은 account\_number, bank\_name, order\_date, order\_ID, customer\_ID, bill\_date, monthly\_charge, total\_charge 값이다. 이 때 새로 생성된 청구서의 bill\_number 는 이전 청구서의 bill\_number 에서 1을 더한 값으로 설정해야 한다. 또한 bill\_date 는 '2022-06-30'로 설정하여 데이터를 삽입하는 쿼리를 보낸다.

출력 결과는 다음과 같다.

```
----- TYPE 7 -----
** Generate the bill for each customer for the past month.**
Success to generate the bill for each customer for the past month!
```