

# **Database System Project 2**

담당 교수 : 정성원

이름 : 박준혁

학번 : 20171643

## 1. 프로젝트 개요

가상의 전자제품 거래 회사의 'DBA(Database Administrator)'가 되어, Manager의 요구사항을 만족시키는 데이터베이스를 설계하는 것이 본 프로젝트의 핵심이다. 이번 2차 프로젝트에서는, 지난 1차 프로젝트에서 제작한 데이터베이스 초기 디자인을 기반으로, 강의 시간에 배운 BCNF Decomposition을 적용해 최종적인 'Decomposed Logical Schema Diagram'을 만들고, 해당 구조를 Erwin을 이용해 'Physical Schema Diagram'으로 변환한다.

이어서 설계한 'Physical Model'에 조응하고, MySQL 내에서 활성화될 수 있는 일련의 가상 데이터 베이스를 직접 작성한 후, 이 데이터들과 MySQL, ODBC를 이용해 'Client 요구를 시시각각 처리하는 Electronics Vendor Company Client Program'을 만드는 것이 2차 프로젝트의 목표이다.

이를 위해 사이버 캠퍼스 공지사항에 명시된 각 종 프로그래밍 환경을 구축해야 한다. 본인은 Client Program의 구현에 ODBC C가 적용된 Visual Studio 2019를 이용하였으며, 다 차례의 디버깅 및 검증을 통해 프로그램의 완성도를 높였다.

또한, 상기한 '가상 데이터베이스 제작'도 본 프로젝트 수행 간에 상당히 중요한 부분이다. 본인은 Microsoft Excel 프로그램을 이용해 1차적으로 데이터를 작성했고, 그 과정에서 Project2에서 상정한 '문제 상황'이 최대한 반영되도록 노력했다. Excel로 제작한 데이터들을 '[TableCovert](#)'라는 온라인 서비스를 이용해 실제 MySQL CREATE & INSERT 문으로 변환해 데이터 베이스를 구축하였다.

자세한 프로젝트 수행 과정은 다음 항목에서부터 소개한다.

## 2. Logical Schema Diagram

### A. BCNF Decomposition

#### 1. 서론

BCNF Decomposition을 진행하기에 앞서 본인은 지난 1차 프로젝트 수행 간에 'Logical Schema Diagram' 구축에 상당히 공들였기 때문에 수정 사항이 그다지 많지 않을 것이라 예측하였고, 실제로도 그러하였다.

BCNF 분해에는 Chapter7 강의자료 61장에서 소개하는 'Simplified Test' 방식을 적극 도입하였다. 아래의 그림은 해당 Decomposition 방법을 Pseudo-Code 형태로 보여주고 있다. F Closure 대신 F로만 판단함에 주목하자.

```

result := {R};
done := false;
compute F+;
while (not done) do
    if (there is a schema  $R_i$  in result that is not in BCNF)
        then begin
            let  $\alpha \rightarrow \beta$  be a nontrivial functional dependency that
            holds on  $R_i$  such that  $\alpha^+$  does not contain  $R_i$ 
            and  $\alpha \cap \beta = \emptyset$ ;
            result := (result -  $R_i$ )  $\cup$  ( $R_i - \beta$ )  $\cup$  ( $\alpha, \beta$ );
        end
    else done := true;

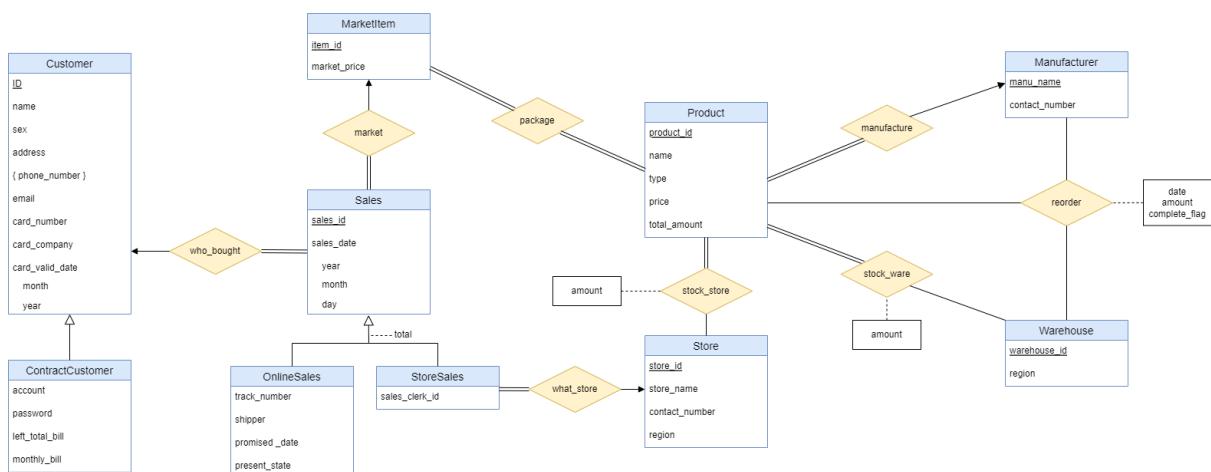
```

Note: each  $R_i$  is in BCNF, and decomposition is lossless-join.

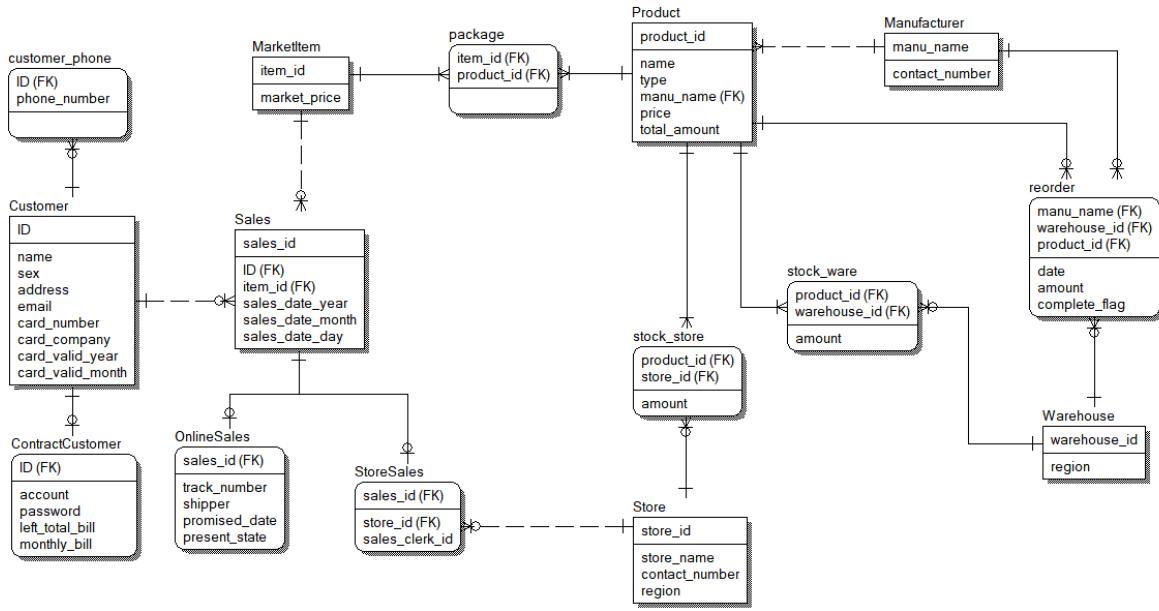
이 방법을 토대로 BCNF Decomposition을 진행했다.

## 2. Former E-R Diagram & Logical Schema Diagram

1차 프로젝트에서 설계한 E-R Diagram은 아래와 같다. 자세한 설계 내용은 지난 1차 프로젝트에서 제출한 Description File에 담겨있다.



이 Diagram을 기초로 ERwin 프로그램에서 제작한 'Logical Schema Diagram'은 아래 그림과 같다.



이제 이 Logical Model을 기초로 해 각 Relation(Table)들에 대해 BCNF Decomposition을 진행해보자.

### 3. Decomposition

기존 Diagram에는 위 그림에서 보이는 것처럼 총 15개의 Table이 존재한다. 각 Table에 대해 'Simplified Test'를 일일이 진행하였다. 이제부터 이 과정에 대해 소개한다. 이 중 일부 Table에 대해서만 실제 Decomposition 적용 과정도 상세히 다룬다.

#### ✓ Customer

<b>Before</b>	Customer(ID, name, sex, address, email, card_number, card_company, card_valid_year, card_valid_month)
<b>After</b>	1) Customer(ID, name, sex, address, email, card_number, card_company) 2) card_info(card_number, card_company, card_valid_year, card_valid_month)

Customer Table에 존재하는 Functional Dependency는 다음과 같이 표시 할 수 있다. (F Set)

- ID  $\rightarrow$  name, sex, address, email, card\_number, card\_company, card\_valid\_year, card\_valid\_month
  - ID는 나머지 모든 Attribute에 대해 Functional Dependency를 보이므로, ID의 Closure는 곧 전체 Attribute 집합과 같다. 따라서 이 Functional Dependency는 BCNF 조건을 충족한다.

- card\_number, card\_company -> card\_valid\_year, card\_valid\_month
  - Left Side가 Primary Key가 아니므로 이 Dependency가 문제가 될 것임을 어렵지 않게 예측할 수 있다. 당연히, (card\_number, card\_company)의 Closure를 구해보면, (card\_number, card\_company, card\_valid\_year, card\_valid\_month)로, BCNF Violation이 된다. 따라서 이를 기준으로 Customer Table을 쪼갤 수 있다.
  - 참고로, 이 Functional Dependency는, 현실 세계에서 카드사와 카드 번호 조합에 따라 카드 유효 기간 정보가 Unique하게 결정된다는 점을 반영해서 상정한 Dependency이다.

따라서, 이를 토대로 Customer Table을 분해하면 아래와 같다.

- 1) **Customer(ID, name, sex, address, email, card\_number, card\_company)**
- 2) **card\_info(card\_number, card\_company, card\_valid\_year, card\_valid\_month)**

두 New Table이 모두 Trivial하게 BCNF임이 자명하다.

하지만 이 Decomposition에는 한 가지 '**DB Client Program 운영 상의 Crucial한 문제**'를 내포하고 있다. 그것은 바로, Customer 정보 기입 이전에 카드 정보를 입력해야 하는, 상당히 불필요한 Routine의 추가를 강요하게 된다는 것이다. 왜냐, Customer Relation에서 card\_number, card\_company Attributes가 Foreign Key로서, Constraint의 적용 대상이기 때문이다.

이를 해결하기 위해선, Client Program을 구현할 때 Program 자체가 사용자의 카드 정보를 먼저 DB에 기록하고, 그 다음에 고객 정보를 기록하는 식으로, 프로그램이 순차적인 흐름을 따르도록 설계하는 방안이 가능하다.

또 다른 방안으로는, 차라리 이 Functional Dependency를 고려하지 않는 것이다. 어차피 카드 정보는 사실상 한 고객에 대해 하나의 정보만 기록될 것이기 때문에 Dependency 자체를 삭제하는 것이다. 대신 이를 위해 고객 정보 기입 프로그램에 허점이 발생하지 않도록 설계하는 것이 중요할 것이다.

이러한 상황에서 본인이 택한 방법은 후자와 유사하다. 카드 유효 기간과 같은 Card Validation 정보는 본 프로젝트2 수행 간에 전혀 중요하지 않은,

부가적이고, 본인이 자의적으로 추가한 정보에 불과하다. 따라서, 본인의 데이터베이스에서 해당 정보(card\_valid\_year/month) 자체를 그냥 삭제하는 것이다. 이는 곧, 위에서 설명했던 Decomposition을 폐기하고, 그냥 기존의 Customer를 그대로 두되, 상기한 두 속성만 제거함을 의미한다. 이는 프로젝트 수행의 편의를 위한 조치이며, 또한, 엄밀히 생각해보면 카드 유효 기간 정보가 Electronics Vendor Company DB에 기록되어야 하는지 명확하지 않다는 점을 고려한 조치이다. 따라서, 아래와 같이 최종 변경한다.

<b>Before</b>	Customer( <u>ID</u> , name, sex, address, email, card_number, card_company, card_valid_year, card_valid_month)
<b>After</b>	Customer( <u>ID</u> , name, sex, address, email, card_number, card_company)

#### ✓ **ContractCustomer**

<b>Before</b>	ContractCustomer( <u>ID</u> , account, password, left_total_bill, monthly_bill)
<b>After</b>	ContractCustomer( <u>ID</u> , account, password, left_total_bill, monthly_bill)

ContractCustomer Table에 존재하는 Functional Dependency는 다음과 같이 표시할 수 있다. (F Set)

- ID -> account, password, left\_total\_bill, monthly\_bill
  - ID의 Closure는 전체 Attribute 집합과 같으므로 BCNF 조건 충족
- account -> ID, password, left\_total\_bill, monthly\_bill
  - account의 Closure 역시 마찬가지로 전체 Attribute 집합과 같으므로 BCNF 조건을 충족한다.

즉, 이 Table은 자명하게 BCNF Form에 해당한다. 따라서, 변화 없이 그대로 유지한다.

#### ✓ **customer\_phone**

<b>Before</b>	customer_phone( <u>ID</u> , phone_number)
<b>After</b>	customer_phone( <u>ID</u> , phone_number)

이 테이블은 소속 Attribute가 모두 Primary Key로 정의되어 있다. 따라서, 자명하게 BCNF이다.

✓ **MarketItem**

<b>Before</b>	MarketItem(item_id, market_price)
<b>After</b>	MarketItem(item_id, market_price)

MarketItem Table에 존재하는 Functional Dependency는 다음과 같이 표시할 수 있다. (F Set)

- item\_id  $\rightarrow$  market\_price

역시나, 매우 자명하게 BCNF임을 알 수 있다. item\_id의 Closure를 F Set 하에서 구하면 당연히 전체 Attribute 집합과 동일하기 때문이다.

✓ **Product**

<b>Before</b>	Product(product_id, name, type, manu_name, price, total_amount)
<b>After</b>	Product(product_id, name, type, manu_name, price, total_amount)

Product Table에 존재하는 Functional Dependency는 다음과 같이 표시할 수 있다. (F Set)

- product\_id  $\rightarrow$  name, type, manu\_name, price, total\_amount
- name, type, manu\_name  $\rightarrow$  product\_id, price, total\_amount

역시나 자명하게 BCNF이다. 두 Functional Dependency의 Left Side의 Closure under F는 모두 명확하게 기존 Product Table과 동일하기 때문이다.

✓ **package**

<b>Before</b>	package(item_id, product_id)
<b>After</b>	package(item_id, product_id)

앞선 customer\_phone Table과 같은 이유로 자명하게 BCNF이다.

✓ **Manufacturer**

<b>Before</b>	Manufacturer(manu_name, contact_number)
<b>After</b>	Manufacturer(manu_name, contact_number)

앞선 MarketItem Table과 같은 이유로 자명하게 BCNF이다.

✓ **Sales**

<b>Before</b>	Sales(sales_id, ID, item_id, sales_date_year, sales_date_month, sales_date_day)
<b>After</b>	Sales(sales_id, ID, item_id, sales_date_year, sales_date_month, sales_date_day)

Sales Table에 존재하는 Functional Dependency는, Sales의 의미와 본 프로젝트 문제 상황을 고려할 시, 아래와 같이 단 하나의 Trivial Functional Dependency밖에 존재하지 않는다.

- $\text{sales\_id} \rightarrow \text{ID, item\_id, sales\_date\_year, sales\_date\_month, sales\_date\_day}$

따라서, 이 테이블 역시 자명하게 BCNF이다.

✓ **OnlineSales**

<b>Before</b>	OnlineSales(sales_id, track_number, shipper, promised_date, present_state)
<b>After</b>	OnlineSales(sales_id, track_number, shipper, promised_date, present_state)

OnlineSales Table에 존재하는 Functional Dependency는 다음과 같이 표시할 수 있다. (F Set)

- $\text{sales\_id} \rightarrow \text{track\_number, shipper, promised\_date, present\_state}$
- $\text{track\_number, shipper} \rightarrow \text{sales\_id, promised\_date, present\_state}$

역시나 자명하게 BCNF이다. 두 Functional Dependency의 Closure under F는 모두 명확하게 기존 OnlineSales Table과 동일하다.

✓ **Store**

<b>Before</b>	Store(store_id, store_name, contact_number, region)
<b>After</b>	Store(store_id, store_name, contact_number, region)

Store Table에 존재하는 Functional Dependency는 다음과 같이 표시할 수

있다. (F Set)

- $\text{store\_id} \rightarrow \text{store\_name}, \text{contact\_number}, \text{region}$
- $\text{store\_name}, \text{region} \rightarrow \text{store\_id}$
- $\text{contact\_number}, \text{region} \rightarrow \text{store\_id}$ ,

역시나 자명하게 BCNF이다. 세 Functional Dependency의 Closure under F는 모두 명확하게 기존 Store Table과 동일하기 때문이다. 참고로, 여기에 명시된 세 Functional Dependency는 관점과 현실 상황에 따라 충분히 다르게 표기될 수 있다. 하지만, 결과적으로 BCNF임에는 변함이 없을 것이다. 기본적으로 Unique 식별자인 ID가 존재하기 때문에 이러하다.

✓ **StoreSales**

Before	StoreSales(sales_id, store_id, sales_clerk_id)
After	StoreSales(sales_id, store_id, sales_clerk_id)

StoreSales Table에 존재하는 Functional Dependency는, StoreSales의 의미와 본 프로젝트 문제 상황을 고려할 시 단 하나의 Trivial Functional Dependency밖에 존재하지 않는다. 따라서, 이 테이블 역시 자명하게 BCNF이다.

✓ **Warehouse**

Before	Warehouse(warehouse_id, region)
After	Warehouse(warehouse_id, region)

앞선 MarketItem Table과 같은 이유로 자명하게 BCNF이다

✓ **stock\_store & stock\_ware**

Before	stock_store(product_id, store_id, amount) stock_ware(product_id, warehouse_id, amount)
After	stock_store(product_id, store_id, amount) stock_ware(product_id, warehouse_id, amount)

(product\_id, store\_id), (product\_id, warehouse\_id)가 각 테이블에서

Primary Key이다. amount속성의 의미를 고려할 시, 두 테이블 모두 오로지 Primary Key로부터 비롯되는 Dependency만 가지게 되고, 따라서 둘 다 자명하게 BCNF Form이라 판단할 수 있다.

✓ **reorder**

<b>Before</b>	reorder(manu_name, warehouse_id, product_id, date, amount, complete_flag)
<b>After</b>	Reorder(warehouse_id, product_id, date, amount, complete_flag)

reorder Table에 존재하는 Functional Dependency는 다음과 같이 표시할 수 있다. (F Set)

- manu\_name, warehouse\_id, product\_id  $\rightarrow$  date, amount, complete\_flag
- product\_id  $\rightarrow$  manu\_name

두 번째 Functional Dependency에서 문제가 발생함이 명확하다. product\_id가 단독으로는 Super Key가 될 수 없기 때문이다(Closure로 판단 가능). 따라서 이 테이블은 아래와 같이 분해할 수 있다.

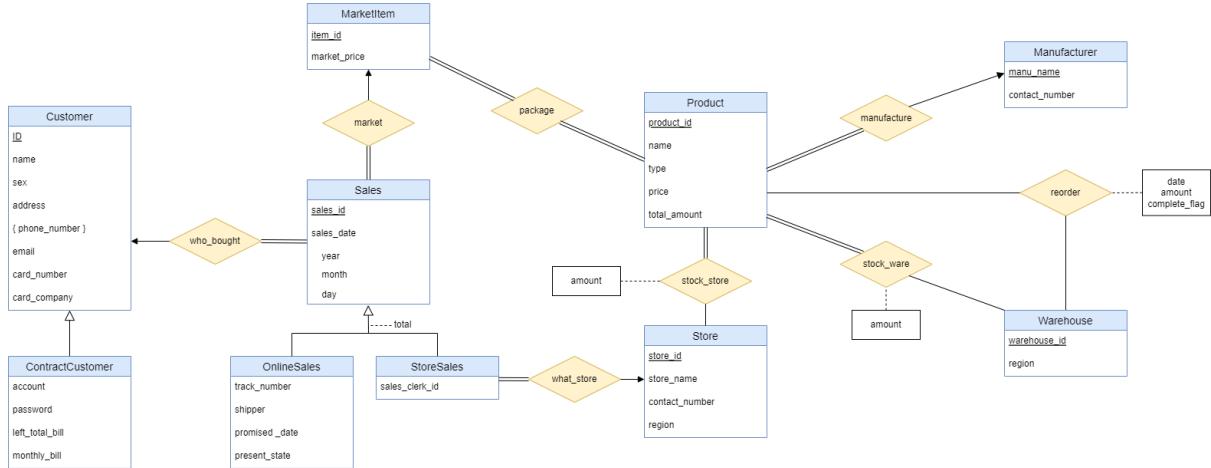
1) **reorder(warehouse\_id, product\_id, date, amount, complete\_flag)**

2) **product\_manu(product\_id, manu\_name)**

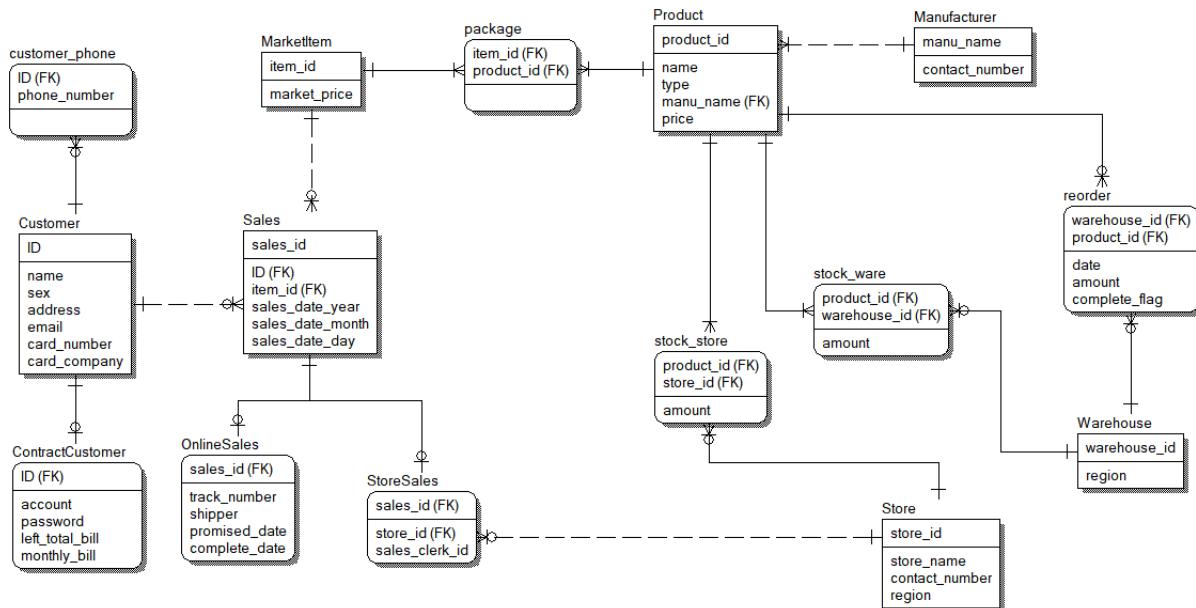
둘 다 자명하게 BCNF이다. 하지만, 두 번째 테이블은 이미 Product라는 테이블을 통해 구현된 부분이다. 따라서, 여기서 필요한 것은 reorder Table에서 단순히 manu\_name(기업명)속성을 지우는 일이다.

이렇게 해서 BCNF Decomposition을 마무리한다. 결과적으로 기존 Model에서 Customer Table과 reorder Table이 Not BCNF Form이었으며, 두 테이블 모두 하위 테이블로의 분해 대신 소속 Attribute를 삭제(이유: 불필요하기 때문)하는 방식으로 BCNF를 갖추게 수정하였다. 최종적으로 수정된 결과를 반영한 E-R Diagram과 Logical Schema Diagram은 아래 그림과 같다. Customer 와 reorder 부분을 중점적으로 확인해보자.

## ◆ Decomposed E-R Diagram



## ◆ Decomposed Logical Schema Diagram



(Hardcopy 제출물에 포함되어 있다)

BCNF Decomposition을 진행하면서 느낀 점은, BCNF에 대한 이론적 이해가 없더라도 “**Data Redundancy**를 방지하기 위해 어떻게 테이블을 구성해야 할까?”라는 고민을 끊임없이 한다면 자연스럽게 BCNF 형식의 Table을 축할 수 있다는 점이었다. Data Redundancy의 방지 노력은 곧 ‘유일 식별자’에 대한 고민이고, 이것이 나아가 BCNF로 이어진다는 것을 알 수 있었다. 실제로 본인의 1차 프로젝트 결과물이 전반적으로 BCNF 형식을 갖추고 있음에서 이를 알 수 있었다.

한편, 위의 'Simplified Test' 방식을 제외하고서도 직관적으로 BCNF 판단을 할 수 있겠다는 생각도 들었다. 아래의 서술이 바로 본인이 발견한 직관적 BCNF 판단 방법이다.

*"Table 내의 Attribute 중에 Non-Candidate Key인 Attribute 간에, 하나의 Attribute 집합이 다른 Attribute 집합을 Unique하게 결정하는 관계가 있을 경우, 이는 BCNF Violation일 확률이 매우 높다."*

사실, 이는 특별한 발견은 아니다. BCNF의 의미와 Simplified Test의 Flow를 천천히 들여다보면, 위의 서술을 정형화된 표현 방법으로 표현한 것임을 알 수 있다. 각설하고, BCNF Decomposition을 이렇게 마무리한다. Hardcopy 제출에 위의 Logical Schema Diagram이 포함된다.

### 3. Physical Schema Diagram

#### A. ERwin Implementation

##### 1. Attribute Revision

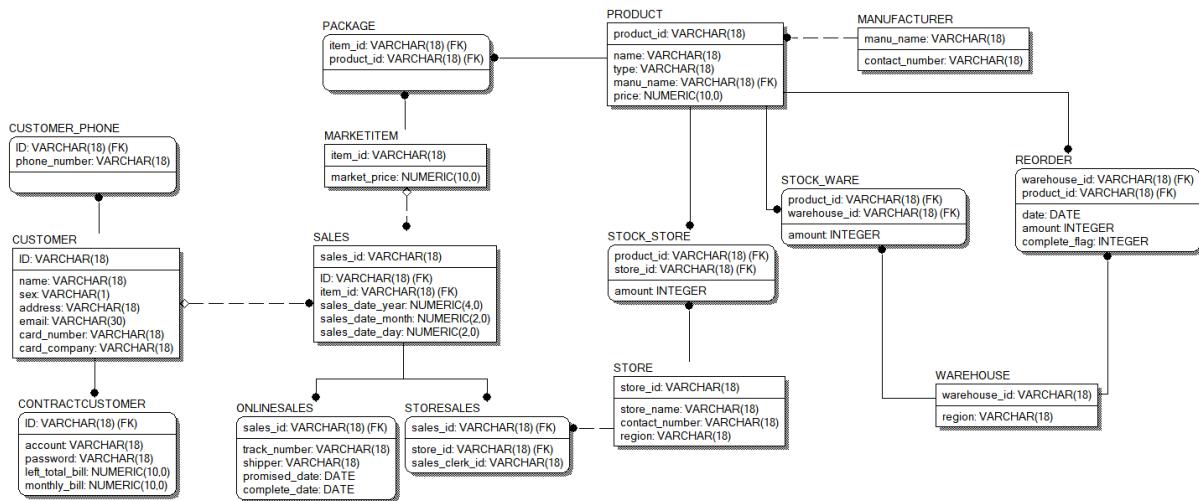
위에 첨부한 Logical Schema Diagram을 보면, E-R Diagram과 두 가지 부분에서 차이가 있다는 것을 알 수 있다.

- OnlineSales Table에서 present\_state라는 Flag 변수를 complete\_date라는 날짜 변수로 변경하였다. 실제 데이터베이스 구현에 있어서 promised\_date와 complete\_date의 비교를 통해 '도달 여부'를 판단하는 것이 좀 더 바람직하다고 판단했기 때문이다.
- Product Table에서 total\_amount라는 변수를 삭제했다. 이 부분은 Data Redundancy가 발생할 수 있는 부분으로, Physical Model 구현 과정에서 발견하였다. ODBC Client Program에서 Query를 통해 Store와 Warehouse에서의 Product Stock Amount를 직접 구하고 다 더하면 충분하다고 판단했다. 따로 total\_amount 속성을 추가로 두면 데이터 중복의 위험이 높아진다.

이 점을 반영한다. 이어서 다음으로 넘어가자.

## 2. Physical Schema Diagram Design

여태까지 설계한 내용을 바탕으로 ERwin 상에서 Physical Schema Diagram 을 구축한다. 각 Table 별로 소속 속성들의 Data Type, Domain, Constraint, NULL 허용 여부 등을 소개하겠다. Relationship Type의 경우, 1차 프로젝트에 서의 설명과 변함이 없기 때문에 자세한 설명은 생략한다.



1) **CUSTOMER** : 모든 소속 속성은 VARCHAR 형이다. ID나 sex 속성의 경우 길이를 제한할 수 있지만, 데이터 삽입 과정에서 불필요한 제한을 거는 행위라고 판단해, 굳이 설정하지 않았다.

이때, Primary Key 역할을 수행하는 ID 속성을 제외하고는 모든 속성이 NULL이 가능하다. CUSTOMER 테이블은 Online 고객과 In-Store 고객 정보를 모두 담기 때문에 이러한 조치를 취했다. In-Store 고객의 경우, 사실상 'Sales' 정보를 위한 고객 정보 생성일 뿐 그 외에는 의미가 없다. 따라서, 실제 이 DB가 사용될 경우, 특정 기간(더 이상 과거의 Sales 정보가 필요 없는 시기에 도달할 정도)을 주기로 name, sex, address, card info 등의 정보가 NULL인 Tuple들(In-Store Sales의 Customer Info)을 모두 제거하는 루틴이 필요할 것이다. 이를 위해, Online 고객의 경우 반드시 Frequent 회원 여부와 상관없이 name, card info 등의 정보를 기입하도록 유도해야 할 것이다.

(with CUSTOMER\_PHONE : One-to-Zero/One/More)

(with CONTRACTCUSTOMER : One-to-Zero/One)

- 2) **CUSTOMER\_PHONE** : 두 속성이 모두 VARCHAR이며, NOT NULL이다. 연락처 정보를 기입하는 고객의 경우, 연락처 개수와 상관없이 이 테이블에 정보가 기록될 것이며, 따라서 연락처 정보 누락을 방지하기 위해 NOT NULL 조건을 걸었다.
- 3) **CONTRACTCUSTOMER** : ID, account, password는 VARCHAR 형이고, bill 관련 속성 두 개는 NUMERIC Type이다. 모든 속성이 NOT NULL이다. 모든 속성 정보가 Frequent 회원 Handling을 위해 반드시 필요한 정보들이기 때문이다.
- 4) **MANUFACTURER** : 모든 속성이 VARCHAR이고, NOT NULL이다. 참고로, 본 Physical Schema Diagram 설계 간에 VARCHAR 형은 모두 길이가 최소 18자 이상인데, 이는 실제 프로그램 구현 및 데이터 생성/입력 간에 불필요한 제한 사항을 두지 않기 위함이다.
- 5) **PRODUCT** : price를 제외한 모든 속성이 VARCHAR 형이고, 모든 속성이 NOT NULL이다. 3)과 마찬가지 이유로, 모든 속성 정보가 Crucial하기 때문에 NULL값이 입력될 수 없도록 설정했다. 3)과 5)를 다루는 Client Program들은 이 점을 명확히 지키도록 설계되어야 할 것이다.
- 6) **MARKETITEM** : item\_id는 VARCHAR형, market\_price는 NUMERIC 형이다. 둘 다 꼭 필요한 정보이므로 NOT NULL이다. 참고로, 본 Physical Schema Diagram 설계 간에 NUMERIC 형은 모두 (10, 0)의 형식을 갖추는데(SALES 테이블은 예외), 이 역시 4)에서 언급한 이유와 마찬가지로, DB 이용 프로그램 구현 간에 자유도를 높이기 위함이다. 현실에서 제품 가격이 10자리(10억)까지 올라갈 일은 없을 것이다.
- 7) **PACKAGE** : 두 소속 속성이 모두 Foreign Key이므로 5)와 6)의 설정을 따라간다.

(with MARKETITEM & PRODUCT : One-to-One/More)

- 8) **SALES** : sales\_id는 VARCHAR 형에 NOT NULL이며, 날짜와 관련된 세 속성은 NUMERIC Type에, 마찬가지로 NOT NULL이다. 나머지 Foreign Key들은 자신들의 원본 테이블 설정을 따라간다. 날짜 정보들이 NOT NULL인 이유는 명확하다. 실제 판매 날짜를 알아야 마케팅 전략 설정을 용이하게 할 수 있기 때문이다(정보 검색에도 용이하다).

이때, DATE Type 대신 NUMERIC(4, 0)을 사용한 이유는 '이전 달', '작년' 등의 정보가 직접적으로 필요한 Query들을 효과적으로 다루기 위함이다. 8)을 다루는 Client Program(이를 테면 각 매장의 프로그램이나 통합 웹사이트)에서는 이 점을 고려해 현재 날짜에 특정 함수를 입혀 년, 월, 일로 분해하는 루틴이 필요할 것이다.

(with CUSTOMER : Zero/One/More-to-Zero/One)

(with ONLINESALES & STORESALES : One-to-Zero/One/More)

(with MARKETITEM : Zero/One-to-Zero/One/More)

- 9) **ONLINESALES** : complete\_date를 제외하곤 모두 NOT NULL이다. complete\_date의 경우, 아직 도착하지 않은 Shipment를 처리하기 위해 NULL이 허용되게 하였다. date 속성들은 모두 DATE Type이며, 나머지는 VARCHAR 형이다.

이때, 실제 프로젝트 Program 구현 상에서는 '아직 도달하지 않은 Shipment'에 대해선 NULL값을 입히지 않고, '2100-12-31'이란 특수 날짜를 기입하는 식으로 처리하였다. NULL이 있으면 아무래도 DB 운영 간에 굳이 좋을 점이 없을 것 같아서 그렇게 처리하였다(즉, NOT NULL이어도 무방하다는 말이다).

(with STORE : One-to-Zero/One/More)

- 10) **STORE** : 모든 속성이 NOT NULL이다. 모두 실제 DB 운영 간에 주요한 정보들이기 때문이다. 또한, 모든 속성이 문자열 데이터를 취급하면 되므로 VARCHAR 형이다.

- 11) **STORESALES** : 점원 정보에 대한 속성만 NULL이 허용된다. 점원 정보는 해당 테이블에서 Crucial한 정보는 아니기 때문이다. 물론, 만약 판매 실적에 따라 특정 점원을 포상하는 식의 운영이 필요할 경우, 이를 NOT NULL로 처리할 수도 있을 것이다. 한편, 나머지 속성은 Foreign Key이므로 원본 테이블의 설정을 따라간다.

- 12) **WAREHOUSE** : warehouse\_id와 region 속성 모두 NOT NULL이며, VARCHAR 타입이다.

**13) STOCK\_STORE** : amount 속성을 포함해 모든 속성이 NOT NULL이며, amount 속성의 경우 데이터 타입은 INTEGER이다. 이때, amount 정보는 무조건 0 이상이어야 하므로 실제 DDL 작성 시, check Directive를 통해 조건을 마련하도록 한다. 이는 하단의 STOCK\_WARE와 REORDER에서도 동일하게 적용된다.

(with PRODUCT & STORE : One-to-One/More)

**14) STORE\_WARE** : store\_id가 warehouse\_id인 걸 제외하면 STOCK\_STORE 와 동일하다.

(with PRODUCT & WAREHOUSE : One-to-One/More)

**15) REORDER** : 모든 속성이 NOT NULL이며, date는 DATE Type, amount, complete\_flag는 INTEGER Type이다. complete\_flag의 경우, 1이면 TRUE (물건 도달 완료), 0이면 FALSE(아직 미-도달)를 의미한다.

(with WAREHOUSE & PRODUCT : Zero-One-to-Zero/One/More)

## B. ERwin to MySQL DDL

### 1. Export as MySQL DDL Form

위에서 설명한대로 ERwin 상에서 Physical Schema Diagram을 구축한다. Tools 탭의 Derive New Model Option을 통해 Physical Schema Model 도출이 가능하다. 도출이 완료되면, 똑같이 Tools의 Forward Engineer Option에서 Schema Generation Option을 택해 MySQL 문법의 DDL을 도출할 수 있다.

DDL이 도출되면, 몇 가지 추가적인 옵션을 넣어준다.

- PRODUCT와 MARKETITEM에 있는 price 관련 Attribute들은 반드시 0보다 큰 값을 가져야 한다. 따라서, check (attribute\_name > 0) 설정을 추가해준다.
- STOCK\_STORE, STOCK\_WARE, REORDER의 amount Attribute들은 반드시 0 이상의 값을 가져야 한다. 따라서, check (amount > -1) 이란 설정을 추가해준다.
- SALES 테이블의 날짜 관련 속성들은 '가능한 값(예를 들어, year

정보의 경우, 1702년 이상, 2100년 미만)의 범위'를 지정해주어야 한다. 이 역시 check Directive를 이용해 설정한다.

이를 통해 최종적인 DDL 파일을 만들 수 있다. 이는 첨부 파일에 있는 20171643.txt 파일을 통해 확인할 수 있다.

## 2. Generate Virtual Data

2차 프로젝트 > DB\_Contents

이름
contractcustomer
customer
customer_phone
manufacturer
marketitem
onlinesales
package
product
reorder
sales
stock_store
stock_temp
stock_ware
store
store_sales
warehouse

1번 항목에서 작성한 DDL을 기반으로 가상 Electronics Vendor Company Database Data를 만들었다. 앞서 언급했던 것처럼, Microsoft Excel을 통해 1차적으로 데이터를 직접 만들었다. 수량 데이터는 Excel의 RAND()함수를 적절히 사용해 생성했으며, 나머지 문자열 및 날짜, 가격 등의 데이터는 모두 일일이 본인이 직접 만들었다.

이를 토대로, 역시나 앞서 언급한 것처럼, 'TableConvert'라는 웹사이트를 통해 각 데이터에 대한 INSERT 문을 자동으로 생성했다. 앞서 만든 DDL과 함께 이 Query 문들을 하나의 텍스트 파일(20171643.txt)에 통합했다(DROP을 위한 DROP 문들과 함께). 자세한 데이터 값 및 CRUD문 구성은 20171643.txt 파일을 통해 확인할 수 있다.

한편, 이러한 데이터 생성 과정에서, 최대한 본 프로젝트 문제 상황과 '현실적 관점'을 반영하고자 노력하였지만, 일부 데이터는 현실과 조응하지 않는 측면이 있을 수 있다(이를 테면, 판매 내역이 실제 상업 회사 판매 내역보다 훨씬 적거나, 특정 기업의 제품 가격이 실제 보다 너무 낮거나, 실제로는 존재하지 않는 상품이거나, 또는 각 지역에 매장과 물류 창고가 너무 많거나, 적거나 하는 등의 상황들). 그러나, 본 프로젝트 Query 처리에는 전혀 문제가 없도록 엄밀하게 데이터를 작성했다는 점을 강조한다. 즉, 프로젝트 수행 간에는 문제가 없는 수준으로 구축했다.

여담으로, 본 데이터베이스에는, Samsung, Apple, AMD와 같은 실제 전자제품 기업명이 등장하며, Airpod과 같이 실제 상품명도 등장한다. 또한, 각 매장이 있는 지역으로는 California를 비롯해, Seoul, New York, Paris, London 등 다양하다. FEDEX와 같은 물류 회사도 등장한다.

한편, 실제 API 구현에 앞서 몇 가지 가정을 설정한다. 이 가정은 프로젝트 수행자에 따라서 달라질 수 있는 요소이기 때문에 명시적으로 설명이 필요하다고 판단했다.

- 1) Broken Shipment에 대한 Re-Packaging 시, 만약 해당 Package에 복수의 Product가 들어있다면, 각 Product가 전체 Warehouse들에서 재고가 남아있는지를 판단한다. 지역과 상관없이 말이다. 각 Product가 모두 최소 한 곳 이상의 Warehouse에서 Stock이 남아있다면, 해당 Package의 재-배송이 가능하다고 가정한다. 즉, 여러 곳의 Warehouse에서 필요 제품을 불출해 한 곳의 (가상) 통합 센터로 모아놓고, 거기서 물품을 포장해 보낸다고 가정하는 것이다.
- 2) 모든 '구매'는 하나의 'Package(단일 Product일 수도, 복수 Product일 수도 있음)' 단위로 이루어진다. 즉, 만약 특정 고객이 여러 Package를 동시에 구매한다고 하면, 이 정보가 순차적으로 Database에 각 Package 별로 기록되는 것이다. 이는 실제 전자제품 판매 회사에서도 채택하고 있는 방법일 것이라 생각하는데, 이렇게 해야 판매 내역을 적절히 고려할 수 있다.
- 3) Off-line(In-Store) 판매 내역의 경우, 상술한 것처럼 ID를 제외하고는 NULL값들로 이루어진 특정 Customer Table Tuple과 연결된다. 따라서, 만약 본인이 설계한 DB로 실제 DB 운영이 이루어진다면, 주기적으로 이러한 Tuple들을 제거하는 작업이 필요할 것이다. 참고로, 본인이 이러한 방식을 고수한 이유는 분명하다. "만약 환불이 이루어지면 어떻게 처리할 것인가?"에 대한 고민에서 비롯된 것이다. 물론, 본 프로젝트 수행 간엔 이 문제를 고려할 필요는 없다. 따라서, In-Store 판매의 경우, 각 Store에 대한 Customer ID를 하나씩만 두어 관리할 수도 있다. 하지만 이렇게 하면 해당 '실제 고객'이 환불이나 교환을 요청할 경우, 또는 제품 Recall 사태가 벌어질 경우, 해당 고객을 추적하는 것이 불가능하다. 따라서, 이렇게 처리하였음을 다시 한 번 강조한다.

## 4. ODBC Implementation

### A. Implementation Detail

## 1. MySQL, ODBC Setting

사이버 캠퍼스 공지사항 내용을 따라 MySQL과 ODBC를 설치하고, Visual Studio 2019와 연동시킨다. 둘 모두 특별한 문제없이 수월하게 진행했다. Visual Studio에서 제공된 test 소스 코드를 컴파일한 후, CMD 창에서 MySQL을 따로 틀어놓고 여러 차례 교차 점검 및 디버깅을 함으로써 C 프로그램과 MySQL이 어떻게 소통하는지를 확인했다. (Workbench도 활용)

## 2. Implementation Detail

이제부터 본인이 개발한 Electronics Vendor Company MySQL C API Client Program의 구체적인 구현 내용을 소개한다. 기본적으로 첨부 코드에 주석을 달아놓았기 때문에, 코드 주석을 통해서도 구현 내용을 확인할 수 있다.

### (1) Declaration

```
#define MAX_LEN      13000
#define QRY_LEN       300
#define clear_buf()    while (getchar() != '\n') { input = 0; } // for brevity
#define clear_buf2()   while (getchar() != '\n');           // for brevity

/********************* Declaration ********************/
// Global variables for MySQL connection
const char* host = "localhost";
const char* user = "root";
const char* pw = "qkrwnsgur1";

// Subroutines for GUI support
void print_prompt(void);
void press_any_key(void);
void try_again_routine(void);

// Subroutines for MySQL setting or an error
void set_safety_mode(MYSQL* );
void unset_safety_mode(MYSQL* );
void error_routine(MYSQL* );

// Subroutines for each query processing
void type1_routine(MYSQL* );
void type1_sub1_routine(MYSQL*, char*, char* );
void type1_sub1_update(MYSQL*, MYSQL_RES*, char* );
bool type1_sub1_error_check(MYSQL*, MYSQL_RES* );
void type2_routine(MYSQL* );
void type2_sub1_routine(MYSQL*, char*, char* );
void type3_routine(MYSQL* );
void type3_sub1_routine(MYSQL* );
void type3_sub2_routine(MYSQL*, int );
void type4_routine(MYSQL* );
void type4_sub1_routine(MYSQL* );
void type4_sub2_routine(MYSQL*, int );
void type5_routine(MYSQL* );
void type6_routine(MYSQL* );
void type7_routine(MYSQL* );
```

- QRY\_LEN은 프로그램에서 MySQL에 전송하는 Query 문자열들의 길이를 나타낸다. 300자면 충분히 모든 Query를 Cover하기 때문에 300자로 설정하였다.
- MAX\_LEN은 CRUD File인 20171643.txt 파일을 프로그램에서 File Open 후 읽을 때 사용하는 'Read Line Maximum Length'이다. STOCK\_STORE INSERT의 경우 약 12,000자 정도가 한 라인으로 입력되기 때문에 넉넉하게 13000으로 설정하였다.
- host, user, pw는 MySQL Connection 시 사용하는 Parameter이며, Test 코드에 있던 db는 없앴다. CRUD File에서 CREATE 문과 USE 문으로 Schema를 생성하는데, 이때, Client Program에서 Default Schema Setting을 따로 하는 것은 굳이 필요 없다 판단하여 이렇게 진행하였다.
- 각 Function Declaration은 함수 이름을 통해 그 기능을 유추할 수 있다. 아래에서 더 자세히 소개한다.

## (2) Main Function

```

FILE* fp = fopen("20171643.txt", "rt");           // open CRUD file.
if (mysql_init(&conn) == NULL)                  // initialize the connection
    printf("mysql_init() error!");
connection = mysql_real_connect(&conn, host, user, pw, NULL, 3306, (const char*)"");
if (connection == NULL)
    error_routine(&conn);
else {
    while (fgets(line, sizeof(line), fp) != NULL) {
        if (!strcmp(line, "$$$\n"))           // read lines from CRUD file
            break;                          // '$$$' separates CREATE / INSERT
        mysql_query(connection, line);      // these are CREATEs & INSERTs
    }

    while (1) {
        print_prompt();                   // print prompt
        scanf("%c", &input);             // read the type
        clear_puf();

        switch (input) {                // handling each input
            case '1': type1_routine(connection); break;
            case '2': type2_routine(connection); break;
            case '3': type3_routine(connection); break;
            case '4': type4_routine(connection); break;
            case '5': type5_routine(connection); break;
            case '6': type6_routine(connection); break;
            case '7': type7_routine(connection); break;
            case '0': exit_flag = true; break;
            default: try_again_routine(); break;
        }
        printf("-----\n");

        if (exit_flag) break;           // if input is '0'
    }

    while (fgets(line, sizeof(line), fp) != NULL)
        mysql_query(connection, line);   // these are DELETEs & DROPs
}
mysql_close(connection);
}

```

File Open 후 fgets 함수를 통해 CRUD File을 읽어 들이고 있음에 주목하자. '\$\$\$'라는 라인을 읽을 때까지 Loop가 돈다. '\$\$\$'는 CRUD 파일에서 CREATE & INSERT 부분과 DELETE & DROP 부분을 구분하기 위해서 존재하는 Special Line이다.

CREATE & INSERT가 마무리 되면 while문을 통해 사용자의 입력을 처리한다. '0'이 입력될 때까지

끊임없이 반복된다. '0'이 입력되면 exit\_flag를 SET해 루프를 탈출한다. 나머지 Option에 대한 입력은 대응하는 프로시저를 통해 처리한다. 예외 처리 및 Prompt 출력 함수도 마련하였음도 확인할 수 있다.

### (3) 프로그램 GUI 구현을 위한 함수들

```
***** Subroutines for GUI support *****
void print_prompt(void) {
    printf("\n\n\n\n----- SELECT QUERY TYPES -----\\n\\n");
    printf("\t1. TYPE 1\\n");
    printf("\t2. TYPE 2\\n");
    printf("\t3. TYPE 3\\n");
    printf("\t4. TYPE 4\\n");
    printf("\t5. TYPE 5\\n");
    printf("\t6. TYPE 6\\n");
    printf("\t7. TYPE 7\\n");
    printf("\t0. QUIT\\n");
    printf("\n      > ");
}

void press_any_key(void) {
    printf("Press <enter> to continue...\\n");
    while (getchar() != '\\n');
    printf("\\n\\n");
}

void try_again_routine(void) {
    printf("\\n\\nYou pressed an incorrect input! Please try again...\\n\\n");
}
```

- 위의 세 함수는 본 프로그램의 GUI 구축을 위해 별도로 마련한 함수들이다. 각 Type의 Query 처리가 종료될 때마다 'Press-Any-Key' 상황이 발생한다. Newline Character '\n'로 끝나는 임의의 키보드 입력을 인식한다.

### (4) MySQL Setting 및 에러 처리에 관한 함수들

```
/* Subroutines for MySQL setting or an error */
// This procedure set the SAFE_MODE of MySQL (for blocking updates)
void set_safety_mode(MYSQL* conn) {
    char set_safe_query[QRY_LEN] = "set SQL_SAFE_UPDATES = 1;";
    mysql_query(conn, set_safe_query);
}

// This procedure unset the SAFE_MODE of MySQL (for allowing updates)
void unset_safety_mode(MYSQL* conn) {
    char unset_safe_query[QRY_LEN] = "set SQL_SAFE_UPDATES = 0;";
    mysql_query(conn, unset_safe_query);
}

// Error handling function
void error_routine(MYSQL* conn) {
    printf("%d ERROR : %s\\n", mysql_errno(conn), mysql_error(conn));
    exit(1);
}
```

- Safety Mode 설정과 관련된 함수들을 확인해보자. MySQL에서는 Client Program에서 DB Schema Value 변경을 시도하면, 기본적으로는 Safe Mode를 통해 변경을 거부한다. 본 프로젝트에서 Type1 과 Type7 Query 처리 간에 UPDATE 과정이 필요한데, 이때 거부되지 않도록 Safety Mode 설정 함수들을 도입한 것이다. UPDATE 가 이루어질 때, Set 함수와 Unset 함수로 둘러싸면 된다.

## (5) Type1 처리 함수

```

void type1_routine(MYSQL* conn) {
    char query1[QRY_LEN] = "select * from (select sales_id, ID, name from (select S.ID,
    MYSQL_RES* res_q1;
    MYSQL_ROW row_q1;
    char input, cus_id[10] = "", sale_id[10] = "";
    bool first_row_flag = true;           // print 'customer ~' only for the first row

    printf("\n\n\n\n----- TYPE 1 ----- \n\n");
    printf("** Assume the package shipped by USPS with tracking\n  number X is reported\n");
    printf("\t|-----\n");

    if (mysql_query(conn, query1) == 0) {
        res_q1 = mysql_store_result(conn);

        while ((row_q1 = mysql_fetch_row(res_q1)) != NULL) {
            if (first_row_flag) {
                printf("\t| Customer '%s(id: %s)'s contact info:\n", row_q1[2], row_q1[0]);
                strcpy(cus_id, row_q1[0]); strcpy(sale_id, row_q1[1]);
                first_row_flag = false;
            }
            printf("\t|\t - %s\n", row_q1[3]);           // print phone_number value
        }
        if (first_row_flag)           // if the result of query is empty,
            printf("\t| *ALERT: There's no such shipment! Check carefully again!\n");

        mysql_free_result(res_q1);
    }
    printf("\t|-----\n\n");
    press_any_key();
    if (first_row_flag) return;           // if the result of query is empty, exit
    while (1) {                         // subtype selection
        printf("\n----- Subtypes in TYPE 1 ----- \n\n");
        printf("\t1. TYPE 1-1\n\t0. QUIT\n\n      > ");
        scanf("%c", &input);
        clear_buf();

        if (input == '1') { type1_sub1_routine(conn, cus_id, sale_id); break; }
        else if (input == '0') { printf("\n\n"); break; }
        else printf("\n\nYou pressed an incorrect input! Please try again...\n\n");
    }
}

```

- 모든 Type 처리 Function은 다음과 같은 구성을 가진다. 먼저, Query 문자열을 마련한다. 이 문자열은 미리 MySQL Workbench 에서 실험 및 검증을 마친 문자열들이다. 그 다음, Query 처리 결

과를 토대로 필요에 따라 중첩 Query를 전송하거나 하는 등의 처리를 추가한다. 이어서, 처리가 끝나면, 바로 종료하거나, Subtype이 있는 경우, Subtype에 대한 입력을 받은 후, Subtype 처리 함수를 호출한다(본 Type1 설명 이후로는 각 함수의 전체 코드를 첨부하지는 않을 것이다. 어차피 코드가 첨부되기 때문에 굳이 전체 코드를 보고서에 담을 필요는 없다고 판단했다).

- Type1에서 사용한 Query는 다음과 같다.

```
select *
from (select sales_id, ID, name
      from (select S.ID, S.sales_id
            from sales S
           where S.sales_id in (select OS.sales_id
                                 from onlinesales OS
                                where OS.shipper = 'USPS'
                                  and OS.track_number = 'X'
                                  and complete_date = '2100-12-31')
          ) P natural join customer
      )
T natural join customer_phone
```

이 쿼리는 USPS 물류 회사의 Track Number 'X'인 배송건에 대해, 해당하는 고객을 찾아, 고객의 연락처(전화번호)를 출력하는 쿼리이다. 결과는 아래와 같다.

	ID	sales_id	name	phone_number
▶	11	1000014	Ronald	010-1234-1244
	11	1000014	Ronald	010-3333-0101

본 Type1 함수에선 이 정보를 출력한다. 첫 번째 출력 열에서만 고객 명과 ID를 명시하도록 처리한다(위의 first\_row\_flag).

만약, 이 쿼리의 결과가 Empty이면, 해당 배송건이 없다는 의미이므로 이에 대한 처리도 추가했음에 주목하자. 이어서 Subtype에 대한 입력을 받고, Subtype1 처리로 넘어간다(위 코드 확인).

## (6) Type1-1 처리 함수

- Subtype 처리에 앞서 아래의 두 함수가 필요하다.

- i. **type1\_sub1\_error\_check** : 앞선 항목에서 서술했던, Re-Package 가능 여부를 확인한다. Broken Shipment의 Package에 소속된 Product들이 모든 Warehouse에 대해 최소 한 곳 이상에서 재고가 있는지 확인하고, 모든 Product가 그러할 경우, Re-Package를 허용한다. 그렇지 않다면, Error를 알린다. 자세한 구현 내용은 첨부 코드에서 확인할 수 있다.
  - ii. **type1\_sub1\_update** : i 함수에서 'Re-Package 허용'을 알리면, MySQL의 실제 Schema에 대해 UPDATE를 수행하는 함수이다. STOCK\_WARE에서 현재 조회 중인 Product에 대한 각 Warehouse의 재고를 확인하고, 가장 처음으로 0이 아닌 재고를 가진 Warehouse에서 Replacement Item을 불출한다. 즉, amount 정보를 Decrement(UPDATE)한다.

이어서, ONLINESALES Table에 대해서도 UPDATE를 수행한다. 기존의 Track Number 'X'를 'newX'로 변경하고, promised\_date도 현재 날짜로 변경한다. complete\_date 및 다른 속성은 그대로 유지한다.

```
if (mysql_query(conn, query) == 0) {
    res = mysql_store_result(conn);

    if (type1_sub1_error_check(conn, res)) {      // i will explain here (next func)
        printf("\t|\n\t| *ALERT: Some products are currently out of stock in\n");
        printf("\t|           all warehouses regardless of regions, and must be\n");
        printf("\t|           reordered first. Thus, replacement is rejected!\n");
    }
} else {
    unset_safety_mode(conn);                      // ready for updating DB

    mysql_query(conn, query);
    res = mysql_store_result(conn);                // update
    type1_sub1_update(conn, res, sale_id);         // - revise the amount of stock
    |                                         // - adjust the shipment info
    set_safety_mode(conn);
}

mysql_free_result(res);
}
```

- Subtype1 처리 함수 내에선 앞서 설명한 두 함수를 통해 위와 같이 Type1-1을 처리한다. Error Check 함수에서 Error가 감지되면 Update Routine을 진행하지 않음을 주목하자. 자세한 Code-Level Detail은 첨부 소스 코드에서 확인할 수 있다.

## (7) Type2, Type2-1 처리 함수

- Type2 처리를 위해 다음과 같은 쿼리를 MySQL에 전송한다.

```
select C.ID, C.name
from (select S.ID, sum(market_price) sum_pay
      from (select *
            from sales
           where sales_date_year = year(current_date - interval 1 year)) S
      natural join MarketItem group by S.ID order by sum_pay desc) K
natural join customer C limit 1
```

작년 한 해 금액을 기준으로 가장 많이 구매한 고객을 찾는다. limit Directive(위에서부터 넘겨받은 숫자만큼 열을 자름)와 order by를 이용해 자동적으로 최다 구매(가격 기준) 고객을 찾고 있다. sum과 group by를 함께 이용해 ID별 구매 가격 총 합을 구해놓고 있음에 주목하자. 현재 DB에서는 Seoul 출신의 Son이란 고객이 최다 구매(가격 기준)자이다.

- Type2에서 Subtype이 있기 때문에, Subtype 입력을 받고 Type2-1 처리 함수를 호출한다. Type1에서는 따로 설명하지 않았는데, 이 과정에서도 오류 입력 처리 및, '0'으로 인한 탈출 처리가 구현되어 있다. 이는 모든 Subtype 입력 과정에서 동일하다.
- Type2의 Subtype 처리에는 두 가지 해석이 존재한다. 한 가지는, Type2에서 구한 최다 구매자가 작년 한 해 가장 많이 구매한 상품이 무엇인지를 찾는 것이고, 또 다른 해석으로는, 고객과 상관 없이 작년 한 해 가장 많이 구매가 이루어진 상품을 찾는 것이다. 본인은 맥락상 첫 번째 상황이 적합하다고 판단했고, 따라서 아래와 같이 쿼리를 이용해 Subtype 처리를 진행했다.

```
select product_id, P.name, P.manu_name
from (select product_id, count(*) count
      from (select *
            from sales
           where sales_date_year = year(current_date - interval 1 year) and ID = (?)
      natural join package
      group by product_id
      order by count desc) K
natural join product P limit 1
```

Type2에서 구한 최다 구매자에 대해, 작년 한 해에 구매한 각 Package의 Product들 중 가장 많이 등장하는 Product가 무엇인지 를 count를 통해 확인한다. 그리고 count에 대해 order by를 수행

해 내림차순 정렬하고, 거기서 limit Directive를 이용해 가장 많이 구매한 상품을 찾아낸다. 이때, 위 코드에서 (?) 부분엔 Type2에서 구한 ‘최다 구매자의 ID’가 들어간다. 이는 아래와 같이 C 소스 코드에서 Concatenation을 진행해 수행했다.

```
char query[QRY_LEN] = "select product_id, P.name, P.manu_name from (select product_id, count(*) as count from package group by product_id order by count desc) K natural join package group by product_id order by count desc) K";  
MYSQL_RES* res;  
MYSQL_ROW row;  
strcat(query, cus_id);  
strcat(query, "") T natural join package group by product_id order by count desc) K
```

#### (8) Type3, Type3-1, Type3-2 처리 함수

- Type3는 아래와 같은 쿼리를 이용한다. 작년 한 해에 판매된 모든 상품을 출력한다(기업명을 기준으로 정렬해 출력한다).

```
select product_id, P.name, P.manu_name
from (select *
      from sales
      where sales_date_year = year(current_date - interval 1 year)) S
natural join package natural join product P
group by product_id
order by manu_name
```

- 이어서 Subtype에 대한 입력을 받는다. Subtype1에 대한 처리는 다음과 같은 쿼리를 이용한다. 사용자로부터 입력 받은 k값이 아래 쿼리문의 limit '3'의 '3' 위치에 들어온다. '3'은 예시로 넣어놓은 값이다. 이때, price에 대한 합을 통해 dollar-amount를 구한다.

```
select product_id, P.name, sum(P.price) dollar_amount
from (select *
       from sales
      where sales_date_year = year(current_date - interval 1 year)) S
natural join package natural join product P
group by product_id
order by dollar_amount desc
limit 3
```

이를 위해, C 소스 코드 내에선, limit까지의 문자열을 마련해둔 후, 사용자로부터 입력 받은 k 값을 문자열로 취급해서 concat 함수로 문자열에 덧붙인다. 이 과정은 아래의 코드를 통해 확인할 수 있다

```

printf("\n\n\n----- TYPE 3-1 ----- \n\n");
printf("** Then find the top k products by dollar-amount sold. **\nWhich K? : ");
scanf("%s", input); // get k input (as string)
clear_buf2();
strcat(query, input); // and concatenate k to query
printf("\t| [ Product, ProductID, Dollar-Amount ]\n");
printf("\t|-----\n");

```

- Subtype2는 Subtype1과 동일한 쿼리문을 이용한다. 대신, 10%에 해당하는 정수 값을, Type3에서 미리 계산해놓은 '전체 상품 개수'를 이용해 함수 내에서 계산한다. 그 계산 결과 값을 limit의 Parameter로 넘겨 결과를 출력하는 것이다. 아래처럼 말이다.

```

int index = 1, k = (int)((float)total / 10.0); // calculating 10% index
char input[5];
// with flooring by coercion

```

C Language에서 제공하는 Casting을 적절히 이용한 것이다.

## (9) Type4, Type4-1, Type4-2 처리 함수

- Type4 처리는 Type3 처리와 매우 유사하다. 단지, Subtype1, 2 처리 시 price에 대한 sum인 Dollar-Amount가 아니라, Unit-Sale을 이용한다는 점만 다를 뿐이다. 아래의 Query를 보자.

```

select product_id, P.name, count(*) sales_count
from (select *
       from sales
      where sales_date_year = year(current_date - interval 1 year)) S
natural join package natural join product P
group by product_id
order by sales_count desc limit 4

```

앞서 설명한 것처럼, limit 다음에 k 입력으로 Concatenation을 수행해 Query 문을 C 소스 코드 내에서 완성한다. Type3와 다르게 product\_id별 Sales 횟수를 count로 구하고, 이를 기준으로 order by, limit를 수행해 Ranking을 매기고 있음에 주목하자.

- Type4-1, Subtype2(4-2)의 처리는 Type3에서 설명한 것과, 바로 위에서 설명한 내용으로 간단히 분석이 가능하다. 따라서, 중복되는 설명이므로 생략한다.

## (10) Type5 처리 함수

- Type5 처리에는 다음과 같은 쿼리를 이용한다.

```
# QUERY1
select store_id, store_name
from store
where region = 'California'

# QUERY2
select product_id, P.name, P.manu_name
from (select *
      from stock_store
      where amount = '0') K
natural join product P
where store_id = (?)
```

QUERY1을 먼저 수행한다. 이는 캘리포니아 지역의 각 Store 정보를 출력하는 간단한 쿼리이다.

이어서, 이 쿼리의 출력 열들에 대해 각각 QUERY2를 중첩적으로 수행한다. '(?)' 부분에 현재 조회 중인 열에 해당하는 store\_id를 덧붙이면 된다. 이 두 번째 쿼리는 해당하는 Store를 기준으로 STOCK\_STORE를 순회하며 재고가 0인 상품들을 모두 찾아주는 쿼리이다.

이렇게 구현한 Type Processing을 실제로 Execution하면, 아래 그림과 같이 결과가 출력된다.

```
----- TYPE 5 -----
** Find those products that are out-of-stock at every store in California. **
Store1
- Apple Iphone (id: 10002)
- Apple Ipad (id: 10005)
- Sony QLEDPVM (id: 10006)
- Apple MacBook (id: 10015)
- Cisco CIS-Router (id: 10020)
- SEIKO SEI-Watch (id: 10027)

Store6
- Apple Ipad (id: 10005)
- Sony SN-Game (id: 10016)
- Cisco CIS-Router (id: 10020)

Store11
- Sony QLEDPVM (id: 10006)
- HP HP-300 (id: 10009)
- Samsung SS-123 (id: 10010)
- Sony SN-Game (id: 10016)

Store14
- Samsung Galaxy20 (id: 10001)
- Apple Ipad (id: 10005)
- HP HP-300 (id: 10009)
- Samsung SS-123 (id: 10010)
- Sony SN-Game (id: 10016)
- SEIKO SEI-Watch (id: 10027)
- NOKIA NOKI-Phone (id: 10080)
```

그렇다. 위와 같은 중첩 처리는 '실제 DB Client Program스러운' 출력을 위해 도입한 것이다.

## (11) Type6 처리 함수

- Type6 처리는 Type5 처리와 그 Flow가 유사하다. 아래와 같이 두 쿼리를 작성하고, 중첩시킨다.

```
# QUERY1
select sales_id, item_id, promised_date, complete_date
from sales natural join (select sales_id, promised_date, complete_date
                           from onlinesales
                           where (promised_date - complete_date) < 0
                           and complete_date <> '2100-12-31') S
order by complete_date desc

# QUERY2
select P.manu_name, product_id, P.name
from package natural join product P
where item_id =
```

QUERY1은 ONLINESALES에서 '아직 도착하지 않은(complete\_date가 2100년 12월 31일인)' 배송 내역들은 제외하고, 그 중 '약속 날짜(promised\_date)'보다 '실제 도착 날짜(complete\_date)'가 더 뒤인 배송건들만 뽑아낸 다음, 이 결과 테이블과 sales를 Natural Join시켜 item\_id 정보와 함께 출력한다.

그 다음, QUERY1의 각 출력 열에서 item\_id를 뽑아낸 다음, QUERY2의 where 조건 끝에 덧붙인 후, QUERY2를 MySQL에 전송해 해당 Package에 포함된 Product들을 출력한다.

## (12) Type7 처리 함수

- Type7은 고객들에게 지난 달에 대한 'Monthly Bill'을 생성하는 쿼리이다. 이를 위해선 '지난 달에 구매를 진행한 고객'이 누군지부터 파악해야 한다. '지난 달'에 구매를 하지 않은 고객은, 기존의 Monthly Bill이 그대로 유지되기 때문이다.

```
select * from (select ID, item_id, left_total_bill, monthly_bill
               from (select ID, item_id
                     from sales
                     where sales_date_year = year(current_date)
                           and sales_date_month = month(current_date - interval 1 month)) P
               natural join contractcustomer) as K
natural join marketitem
```

따라서, Type7에는 좌측과 같은 Query 가 필요하다. 이 쿼리는 지난 달 구매를 진행한 고객 정

보를 출력하는 쿼리이다. 현재 DB 데이터 상에선 다음과 같은 결과가 출력된다.

	item_id	ID	left_total_bill	monthly_bill	market_price
▶	20039	3	0	0	850
	20026	18	0	0	100
	20015	2	0	0	2200
	20011	18	0	0	100
	20013	10	0	0	700

구매한 제품과 해당 제품의 판매 가격, 그리고 구매자의 ID, left\_total\_bill, monthly\_bill이 출력된다. 그림과 같이, 구현의 편의를 위해 Default Value를 Left Bill과 Monthly Bill을 모두 0으로 설정하였다. 변화를 명시적으로 확인하기 위해서이다.

여기서, 각 출력 열에 대해 아래와 같은 처리를 하는 C 코드를 작성한다.

```

while ((row = mysql_fetch_row(res)) != NULL) {
    sscanf(row[2], "%d", &left_bill);           // fetch left_total_bill
    sscanf(row[4], "%d", &charge);             // fetch 'this month's payment'

    left_bill += charge;                      // add charge to the left bill
    month_bill = (int)((float)left_bill / 12.0); // calculate monthly bill

    char make_bill[QRY_LEN] = "";
    sprintf(make_bill, "update contractcustomer set left_total_bill = %d, monthbill = %d where id = %s",
            left_bill, month_bill, row[1]);        // updates bill info
    mysql_query(conn, make_bill);

    printf("\t| - Charge %d$ to customer(id: %s, purchased item '%s')\n",
           charge, row[1], row[0]);
}

```

Left Bill과 '구매 상품 가격(Charge)'을 뽑아낸다. 이어서, 기존의 Left Bill에 Charge를 더한다. 그 다음, 새로운 Left Bill 값을 12(1년 단위 할부 제도라고 가정한다)로 실수 나누기를 진행한다. 계산 결과에서 소수점을 버린다. 버림으로 인해 12달 내에 Left Bill을 모두 소진하지 못하면(추가 구매 없이), 추가적으로 한 달 더 Monthly Bill 결제가 이루어진다고 가정하자.

새롭게 변화된 Left Bill과 Monthly Bill을 위 코드와 같이 UPDATE한다. 참고로, Type7 처리 함수의 시작과 끝에 Safety Mode Unset / Set 처리가 이루어진다.

- 이어서, 아래와 같이 새롭게 갱신된 Monthly Bill을 출력하는 코드를 작성한다. 새롭게 갱신된 고객에 대해서만 출력한다.

```
strcat(query, " group by ID");
mysql_query(conn, query);
res = mysql_store_result(conn);
while ((row = mysql_fetch_row(res)) != NULL)    // prints the new generated bill
    printf("\t| ~> New 'monthly bill' for customer(id: %s) is %s$\n",
           row[1], row[3]);

mysql_free_result(res);
```

- Type7 Query 수행은 한 달에 한 차례만 이루어져야 한다. 두 번 이상 해당 쿼리를 수행할 경우, 고객에게 과금을 하게 된다. 한편, 본 프로젝트에는 포함되지 않았지만, 실제 DB 운영에서는 매 달 새롭게 Monthly Bill을 갱신한 다음, 모든 Contract Customer에 대해 Monthly Bill Value만큼 실제로 Card 결제를 진행하는 프로그램이 필요할 것이다.

~> 본 프로그램 구현 간에 사용한 모든 쿼리는, 효율성보다도 가독성에 치중해 작성했다. 현대 CPU, MySQL 등은 모두 빠른 성능을 보이기 때문에, 성능 향상을 위한 Query보다는, ‘직관적으로 빠르게 이해하기 쉬운’ Query를 작성하는 데에 집중했다. 그것이 나에게도, Query를 읽는 이에게도 모두 좋을 것이라 판단했다.

### 3. Execution & Some Precautions

2번 항목에서 소개한대로 소스 코드를 작성해 최종적인 프로그램을 완성했다. 자세한 Code-Level Detail은 첨부된 소스 코드에서 확인할 수 있다.

한편, 본 프로그램을 정상적으로 실행하는 방법, 그리고 실행 간의 주의사항에 대해 간략하게 소개하겠다. 참고로 이 내용은 첨부된 README.txt 파일에도 담겨있다.

#### (1) 프로그램 실행법

- 사이버 캠퍼스에 공지된 MySQL, ODBC 설정을 정상적으로 수행한다. Visual Studio 2019와 MySQL의 연결, 그리고 Visual Studio 내에 MySQL 사용을 위한 라이브러리 포함 설정이 가장 중요하다.
- 이어서, 20171643.zip을 압축 해제 후, 생성된 20171643 폴더를

그대로 지정된 Visual Studio 2019 Repository 폴더에 옮겨놓는다  
(굳이 옮기지 않아도 상관은 없다).

- 그 다음, 20171643 폴더 내의 DatabaseSystemProject.sln 파일을 Visual Studio 2019를 통해 연다. 이때, libmysql.dll 파일이 'DatabaseSystemProject' 폴더 내에 포함되어 있어야 한다.
  - 솔루션이 정상적으로 열리면, 바로 컴파일하여 프로그램을 실행시키면 된다. 전체적인 파일 구조는 다음과 같다.
- [DBproject2]20171643 ---- 20171643.png
- 20171643.erwin
  - [project2]20171643.pdf
  - 20171643.zip -----
    - DatabaseSystemProject ----- 20171643.c
    - DatabaseSystemProject.sln - 20171643.txt
    - DatabaseSystem...

## (2) 주의사항

- 본 프로젝트에서는 채점 환경이 Visual Studio 2019이기 때문에 System 및 Signal 함수를 사용할 수 없어 'CTRL+C' 입력으로 인한 프로그램 강제 종료 시의 DB 삭제 기능을 포함시키지 못했다.
- 따라서, **프로그램을 실행시키면, 반드시 메인 Prompt 화면에서 0 번 옵션을 입력해 정상적으로 프로그램 자체가 스스로 종료하도록 해야 한다.**
- 그렇게 하지 않고 'CTRL+C'를 눌러 강제 종료를 시키면 프로그램이 '\$\$\$' 문자열 이후의 CRUD 파일 내 DELETE & DROP Routine을 수행하지 못해 MySQL 내에 본 프로그램으로 인해 새로 생성된 Schema가 그대로 남아있게 된다.

이렇게 해서, 데이터베이스 시스템 2차 프로젝트를 마무리한다. 본 프로젝트를 통해 직접 데이터베이스와 DB 응용 프로그램을 만들어봄으로써 데이터베이스에 대해 막연하게만 알고 있던 개념들을 자세히, 구체적으로 이해할 수 있었다. 본 프로젝트를 수행하면서 체득한 경험과 기술을, 학기가 종료되고 나서도, 나아가 추후 대학원 혹은 업계에 나가서도 언제든지 자유롭게 사용할 수 있도록 추후 프로젝트 전체 수행 과정을 다시 한 번 천천히 복기해볼 것이다.