

CSE4110 – Database System

Project2. Normalization and Query Processing



Spring 2022

20161230 박재형

[목차]

0. Introduction

1. Physical Schema

- 1.1. Attribute & Domain
- 1.2. Relationship
- 1.3. Physical Schema Diagram

2. BCNF Dependency

3. CRUD Implementation

- 3.1. Create
- 3.2. Insert
- 3.3. Delete & Drop

4. ODBC C language

- 4.1. TYPE 1
 - 4.1.1. TYPE 1-1
- 4.2. TYPE 2
 - 4.2.1. TYPE 2-1
- 4.3 TYPE 3
 - 4.3.1. TYPE 3-1
 - 4.3.2. TYPE 3-2
- 4.4. TYPE 4
 - 4.4.1. TYPE 4-1
 - 4.4.2. TYPE 4-2
- 4.5. TYPE 5
- 4.6. TYPE 6
- 4.7 TYPE 7

0. Introduction

프로젝트 1에서는 전자제품 판매 회사에 대한 논리적 데이터베이스를 구성하였다면, 프로젝트 2에서는 물리적 데이터베이스를 구현하고, 테이블들이 모두 BCNF 정규형을 만족하도록 분해한다. 그리고 ODBC 를 이용하여 MySQL DBMS 에서 직접 테이블을 만들고, 튜플을 삽입한 다음, 소비자가 원하는 쿼리에 대한 올바른 결과를 낼 수 있도록 C++ 코드를 작성한다.

1. Physical Schema

1.1 Attribute & Domain

1) customer

고객에 대한 정보를 제공한다. 기본적으로 고객을 식별할 수 있는 ID가 필요하며, 고객의 이름 또한 저장한다. 그리고, 상품 배송을 위해 고객의 연락처와 주소가 필요하다.

| Attribute name | Domain | Constraints |
|----------------|-------------|----------------|
| customer_ID | CHAR(12) | (PK), NOT NULL |
| customer_name | VARCHAR(20) | NOT NULL |
| phone_number | VARCHAR(15) | NOT NULL |
| address | VARCHAR(30) | NOT NULL |

customer_ID : 고객의 고유 ID. 가입일자 6자리와 해당 날짜에 가입한 순서 6자리로 ID를 구성한다. 하루에 100만명 미만의 신규 고객이 존재한다고 가정한다. (ex. 220410 + 000001)

customer_name : 고객의 이름을 의미한다.

phone_number : 고객의 전화번호를 의미한다.

address : 고객이 사는 지역을 의미한다.

2) payment

지불할 수단과 계좌/카드번호를 제공한다. 계좌인 경우 계좌 번호, 카드인 경우 카드 번호를 저장하며, 어떤 고객에 대한 정보인지 알기 위해 고객의 ID또한 저장한다.

| Attribute name | Domain | Constraints |
|----------------|-------------|----------------------------|
| payment_number | VARCHAR(20) | (PK), NOT NULL |
| payment_type | VARCHAR(10) | NOT NULL |
| customer_ID | CHAR(12) | (FK -> customer), NOT NULL |

payment_number : 지불할 수단의 개인 번호를 의미한다. (계좌번호 or 카드번호)

payment_type : 지불할 수단을 의미한다. account 또는 credit 또는 debit을 속성값으로 가진다.

customer_ID : 고객의 고유 ID. 'customer' entity를 식별할 수 있는 외래키이다.

3) package

여러 상품으로 구성되는 패키지에 대한 정보를 제공한다. 패키지 또한 하나의 상품으로 취급되어야 하므로 패키지마다 고유의 ID가 필요하며, 패키지의 이름과 패키지의 가격도 저장해야 한다.

| Attribute name | Domain | Constraints |
|----------------|-------------|----------------|
| package_ID | VARCHAR(20) | (PK), NOT NULL |
| package_name | VARCHAR(20) | NOT NULL |
| package_price | INTEGER | NOT NULL |

package_ID : 패키지의 고유 ID를 의미한다.

package_name : 패키지의 이름을 의미한다. (ex. Gateway PC)

package_price : 패키지의 가격을 의미한다.

4) inventory

현재 각 상점과 창고에 있는 상품들의 재고에 대한 정보를 제공한다. 특정 지역에 어떤 상품이 얼마나 잘 팔리는지, 얼마나 남았는지를 알기 위해 상점과 창고가 위치한 지역도 저장한다.

| Attribute name | Domain | Constraints |
|----------------|-------------|----------------|
| inventory_ID | VARCHAR(6) | (PK), NOT NULL |
| inventory_type | VARCHAR(10) | NOT NULL |
| region | VARCHAR(30) | NOT NULL |

inventory_ID : 상점 또는 창고의 고유 번호를 의미한다. inventory_type : 상점인지 창고인지를 의미한다.

region : 상점 또는 창고가 위치한 지역을 의미한다.

5) Product

판매하는 모든 상품에 대한 정보를 제공한다. 상품의 ID와 이름이 필요하며, 제품의 생산 일자에 따라 가치가 다를 수 있기 때문에 생산 일자 또한 저장한다. 그리고 제품의 가격과 제품의 제조사도 저장한다.

각 상품이 어느 창고 또는 어느 상점에서 판매 중인지 알 필요가 있기 때문에 인벤토리의 ID도 저장하며, 하나의 단일 상품이 패키지에 포함되어 있을 수 있고, 그 경우 해당 패키지에 어떤 상품들이 포함되어 있는지 알아야 하므로 패키지 ID 또한 저장한다.

| Attribute name | Domain | Constraints |
|-----------------|-------------|-------------------|
| product_ID | VARCHAR(20) | (PK), NOT NULL |
| product_name | VARCHAR(20) | NOT NULL |
| production_date | DATE | NOT NULL |
| product_price | INTEGER | NOT NULL |
| manufacturer | VARCHAR(30) | NOT NULL |
| inventory_ID | VARCHAR(6) | (FK -> inventory) |
| package_ID | VARCHAR(20) | (FK -> package) |

product_ID : 제품의 고유 모델명을 의미한다. production_date : 제품을 생산한 날짜를 의미한다. (ex. 20220410)

product_name : 제품의 이름을 의미한다. (ex. 갤럭시S20, 갤럭시워치4클래식)

product_price : 제품의 가격을 의미한다. manufacturer : 제품을 생산한 제조사를 의미한다. (ex. 삼성, LG)

inventory_ID : 제품이 있는 상점 또는 창고를 의미한다. 상점 또는 창고에 없는 경우 NULL값을 가질 수 있다.

package_ID : 제품이 속해 있는 패키지를 의미한다. 패키지에 속하지 않을 수도 있으므로 NULL값을 가질 수 있다.

6) bill

고객이 상품을 결제한 기록이 담긴 영수증에 대한 정보를 제공한다. 기본적으로 영수증을 식별할 수 있는 ID, 물품을 구매한 개수, 구매한 날짜가 필요하며, 구매한 고객이 누구인지도 알아야 한다.

그리고, 어떤 수단으로 지불했는지를 알 수 있는 지불 번호와, 단일 상품을 구매했는지 패키지를 구매했는지를 알아야 하고, 배송으로 전달하는 경우 추적 번호 또한 필요하다.

| Attribute name | Domain | Constraints |
|----------------|-------------|----------------------------|
| bill_ID | CHAR(14) | (PK), NOT NULL |
| product_count | INT | NOT NULL |
| purchase_date | TIMESTAMP | NOT NULL |
| customer_ID | CHAR(12) | (FK -> customer), NOT NULL |
| payment_number | VARCHAR(20) | (FK -> payment) |

| | | |
|-----------------|-------------|------------------|
| product_ID | VARCHAR(20) | (FK -> product) |
| package_ID | VARCHAR(20) | (FK -> package) |
| tracking_number | CHAR(12) | (FK -> shipment) |

bill_ID : 영수증의 고유 번호를 의미한다. 한 번에 여러 상품을 구매한 경우에, 구매한 각 모델에 대해서 별도로 저장한다. 구매일자 6자리 + count 8자리로 구성된다.

product_count : 상품의 구매 개수를 의미한다. purchase_date : 상품을 구매한 날을 의미한다.

customer_ID : 고객의 고유 번호를 의미한다.

payment_number : 고객이 지불하는데 이용한 수단의 번호. 만약 현금으로 지불한 경우 NULL값을 갖는다.

product_ID : 구매한 상품의 ID. 만약 패키지를 구매한 경우 NULL값을 갖는다.

package_ID : 구매한 패키지의 ID. 만약 단일 상품을 구매한 경우 NULL값을 갖는다.

tracking_number : 만약 매장에서 직접 구매한 경우 배송할 필요가 없으므로 NULL값을 갖는다.

7) reorder_list

상점 또는 창고에서 요청한 재주문에 대한 정보를 제공한다. 각 상품 모델에 대해서 각각 저장한다. 어떤 상품을 재주문했는지, 어느 상점 또는 창고에서 요청했는지에 대한 정보가 필요하다.

| Attribute name | Domain | Constraints |
|----------------|-------------|---------------------------|
| reorder_ID | CHAR(12) | (PK), NOT NULL |
| is_filled | VARCHAR(3) | NOT NULL |
| fill_count | INT | NOT NULL |
| product_ID | VARCHAR(20) | (FK -> product), NOT NULL |
| inventory_ID | VARCHAR(6) | (FK -> product), NOT NULL |

reorder_ID : 재주문한 요청의 고유 번호를 의미한다.

is_filled : 재주문 처리가 완료되었는지를 의미한다. 'YES' 또는 'NO'의 값을 갖는다.

fill_count : 채워야 하는 상품의 개수를 의미한다.

product_ID : 상품의 고유 번호를 의미한다.

inventory_ID : 상점 또는 창고의 고유 번호를 의미한다.

8) shipment

온라인으로 구매한 제품의 배송 관련 정보를 제공한다. 상품이 올바르게 배송되지 않은 경우 배송 정보를 확인하여 재배송을 해주어야 하기 때문에 추적 번호가 필요하다. 그리고, 예정된 시간 내에 도착했는지를 알기 위해 도착 예정 시간을 저장해야 하고, 배송이 완료되었는지 또한 저장해야 한다.

| Attribute name | Domain | Constraints |
|-----------------------|-------------|----------------|
| tracking_number | CHAR(12) | (PK), NOT NULL |
| shipping_company | VARCHAR(20) | NOT NULL |
| send_time | TIMESTAMP | NOT NULL |
| promised_deliver_time | TIMESTAMP | |
| is_delivered | VARCHAR(3) | NOT NULL |

tracking_number : 운송 추적 번호를 의미한다. 이 번호를 통해 운송중인 상품의 정보를 유일하게 식별할 수 있다. 운송 시작일자 6자리와 해당 일에 배송한 순서 6자리로 구성된다. (220401 + 000001)

shipping_company : 제품의 운송을 맡은 운송 회사명을 의미한다.

send_time : 제품의 운송을 시작한 시간을 의미한다.

promised_deliver_time : 도착하기로 약속한 시간을 의미한다. 별도의 기한이 없다면 NULL값을 갖는다.

is_delivered : 배송이 완료되었는지를 나타낸다. 'YES' 또는 'NO'의 값을 갖는다.

9) product_inventory

각 상품이 상점 또는 창고에 얼마나 구비되어 있는지를 알아야 한다. 이를 위해 각 상점/창고 별로 보유한 상품의 ID와 현재 재고를 저장한다.

| Attribute name | Domain | Constraints |
|----------------|-------------|-----------------------------------|
| inventory_ID | VARCHAR(6) | (PK), (FK -> inventory), NOT NULL |
| product_ID | VARCHAR(20) | (PK), (FK -> product), NOT NULL |
| stock | INT | NOT NULL |

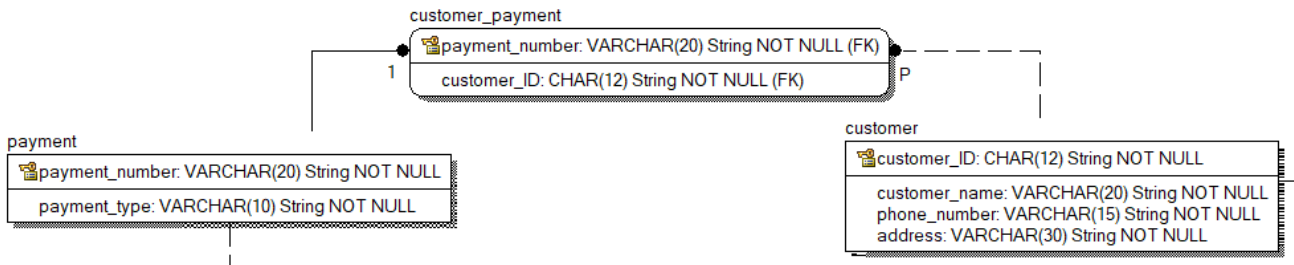
inventory_ID : 상점 또는 창고의 고유 번호이다.

product_ID : 상품의 고유 번호이다.

stock : 해당 상점/창고에 있는 상품의 현재 재고이다.

1.2. Relationship

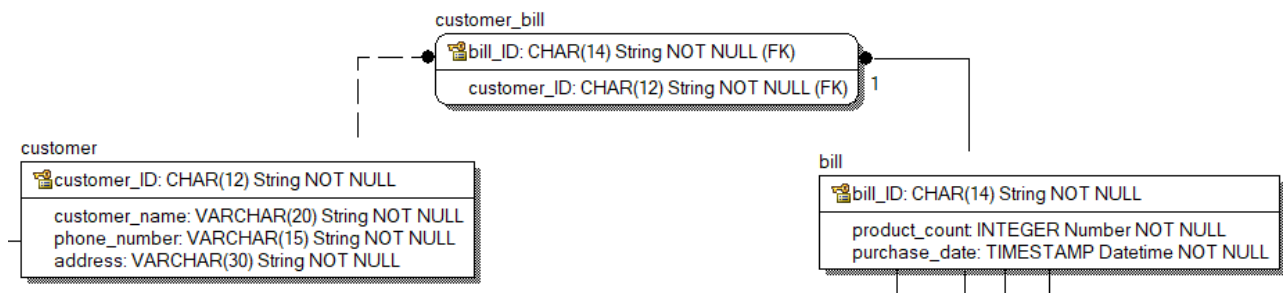
1) customer_payment



고객별로 지불하는 수단(계좌, 신용카드, 직불카드)과 그에 맞는 계좌/카드번호를 연결한다. 모든 지불 수단은 반드시 1명의 고객과 연결되어야 하고, 한 고객은 여러 지불 수단을 가질 수 있다. 따라서 many-to-one 관계로 이어준다.

하나의 계좌/카드번호로 고객의 ID를 식별할 수 있으므로 payment_number가 기본키가 된다.

2) customer_bill

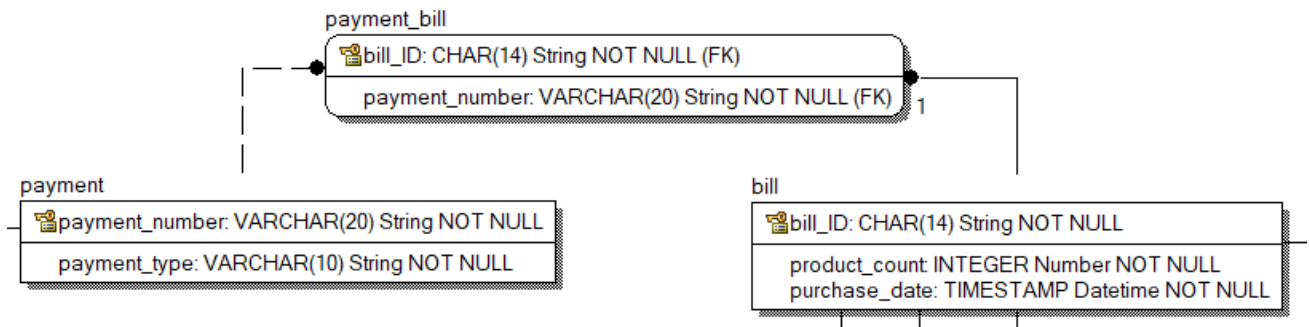


결제 기록이 담긴 영수증과 고객을 연결한다. 이를 통해 어떤 고객이 무슨 상품을 구매했는지를 파악할 수 있다.

한 고객이 여러 결제를 진행할 수 있으므로 many-to-one 관계를 이루며, 각 영수증에는 반드시 고객이 존재해야 하므로 total이다.

영수증의 번호로 고객의 ID를 식별할 수 있으므로 bill_ID가 기본키가 된다.

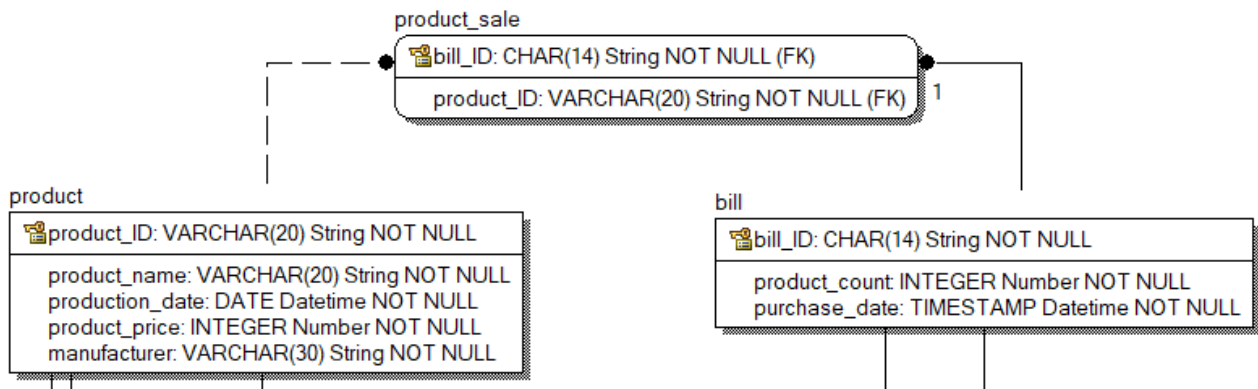
3) payment_bill



각 결제별로 이용한 결제 수단을 연결해준다. 각 결제 수단이 여러 상품 결제에서 사용될 수 있고, 하나의 상품 결제에서는 하나의 결제수단만 사용할 수 있으므로 one-to-many 관계이다.

영수증의 번호로 결제 수단의 번호를 식별할 수 있으므로 bill_ID가 기본키가 된다.

4) product_sale

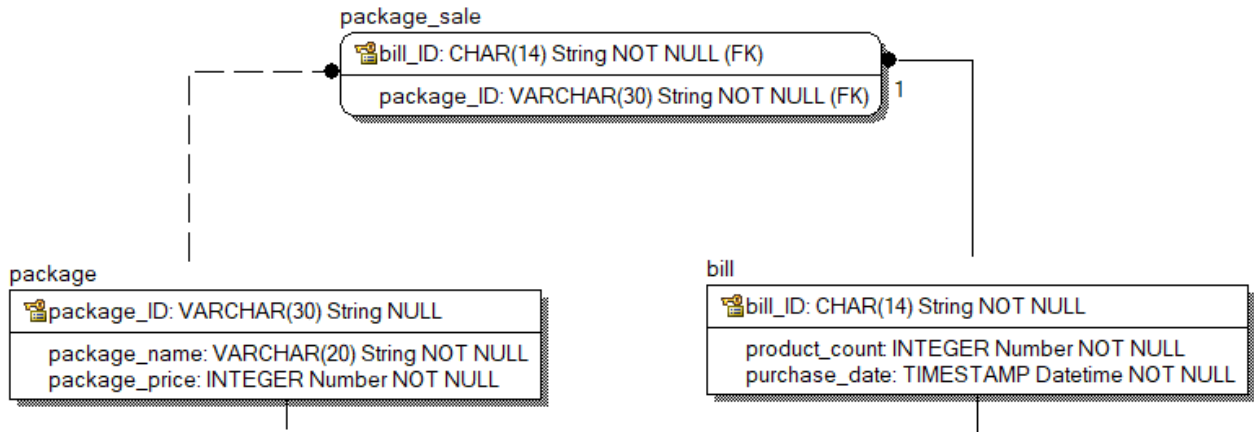


결제 기록이 담긴 영수증과 상품을 연결한다. 이를 통해 판매된 상품에 대한 정보를 파악할 수 있다.

결제 기록은 동일한 모델에 대해서만 한 영수증으로 저장하기 때문에, 한 영수증에 한 상품만 포함될 수 있고, 여러 고객이 동일한 상품을 구매한 경우 여러 영수증에 동일한 상품이 존재할 수 있으므로 many-to-one 관계를 이룬다.

영수증의 번호로 상품의 ID를 식별할 수 있으므로 bill_ID가 기본키가 된다.

5) package_sale

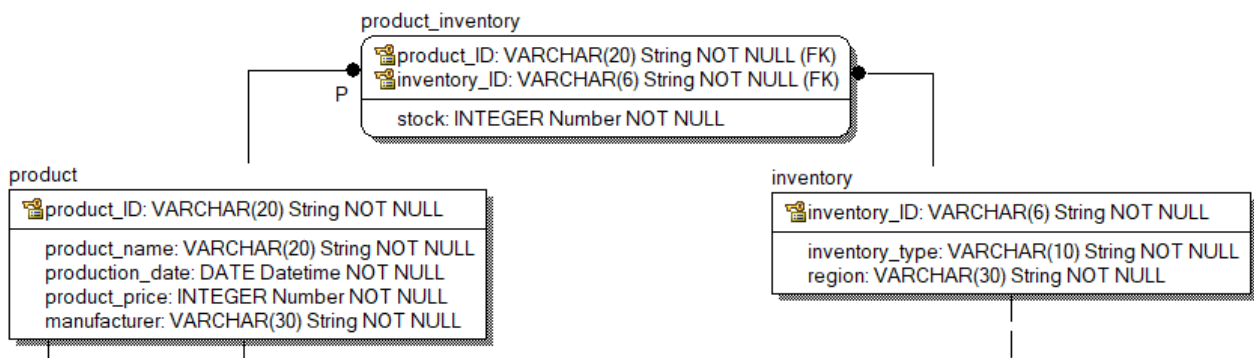


결제 기록이 담긴 영수증과 패키지를 연결한다. 이를 통해 판매된 패키지에 대한 정보를 파악할 수 있다.

결제 기록은 동일한 모델에 대해서만 한 영수증으로 저장하기 때문에, 한 영수증에 한 패키지만 포함될 수 있고, 여러 고객이 동일한 패키지를 구매한 경우 여러 영수증에 동일한 패키지가 존재할 수 있으므로 many-to-one 관계를 이룬다.

영수증의 번호로 패키지의 ID를 식별할 수 있으므로 bill_ID가 기본키가 된다.

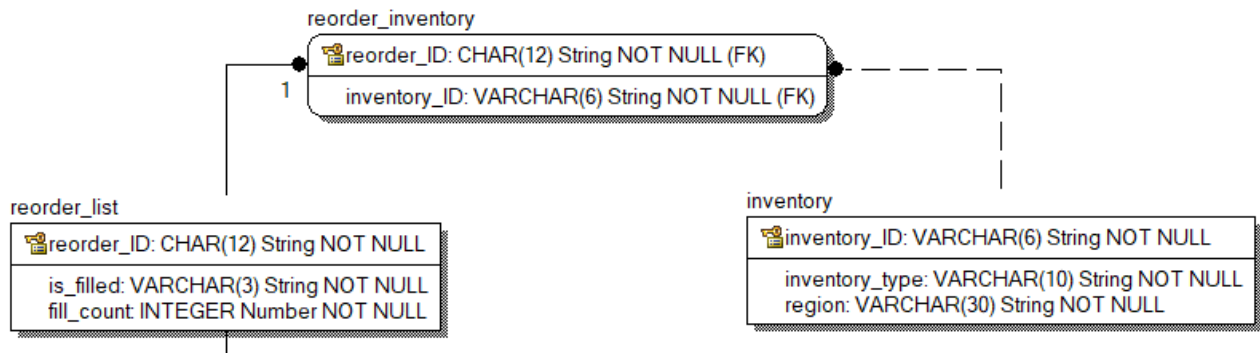
6) product_inventory



각 상점 또는 창고와 상품들을 연결해준다. 이를 통해 각 상점과 창고에 존재하는 상품의 정보를 알 수 있으며, stock이라는 relationship 속성을 통해 각 상품의 재고 현황을 파악할 수 있다. 각 상품은 여러 상점과 창고에 포함될 수 있고, 상점과 창고 또한 여러 상점에 해당될 수 있으므로 many-to-many 관계를 이룬다.

튜플을 식별하기 위해선 product_ID와 inventory_ID가 모두 필요하므로 기본키는 (product_ID, inventory_ID)가 된다.

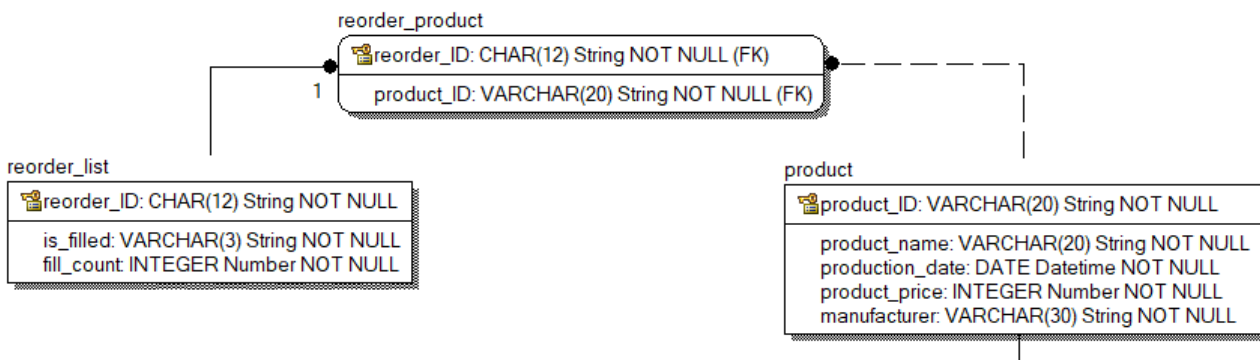
7) reorder_inventory



재주문 리스트와 상점/창고를 연결해준다. 재주문을 요청한 상점/창고에 대한 정보를 파악할 수 있다. 각 상점은 여러 번 재주문할 수 있으므로 many-to-one 관계를 이룬다.

재주문 리스트의 ID로 상점/창고를 식별할 수 있으므로 reorder_ID가 기본키가 된다.

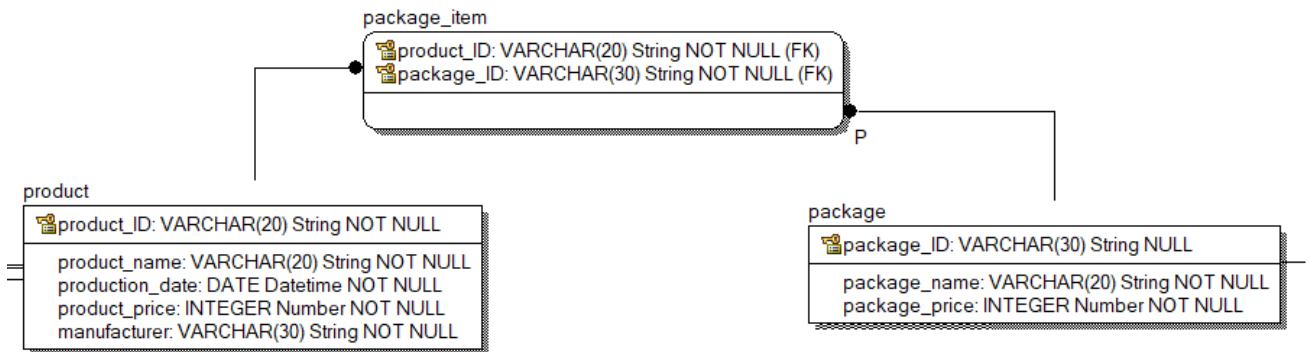
8) reorder_product



재주문 리스트와 상품을 연결해준다. 재주문한 상품에 대한 정보를 파악할 수 있다. 각 상품은 여러 번 재주문 될 수 있으므로 many-to-one 관계를 이룬다.

재주문 리스트의 ID로 상품을 식별할 수 있으므로 reorder_ID가 기본키가 된다.

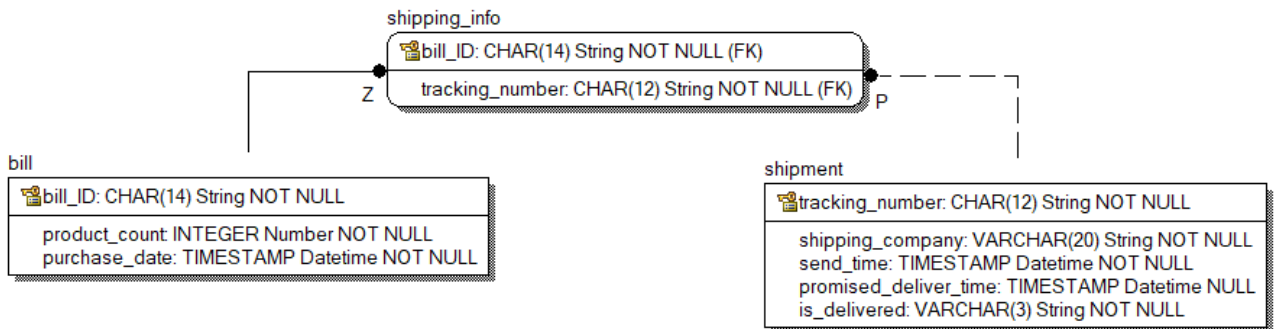
9) package_item



여러 상품을 모아서 하나의 패키지를 구성할 수 있으므로 상품과 패키지를 연결해준다. 각 상품은 여러 패키지에 들어있을 수 있고, 각 패키지에도 여러 상품이 포함되어 있으므로 many-to-many 관계로 이어준다.

many-to-many의 관계이므로 속성값 전체가 기본키가 된다.

10) shipping_info



결제 중 온라인 결제에 대해서 배송과 연결해준다. 여러 상품을 동시에 구매한 고객에게 묶음으로 배송하기 때문에 여러 영수증이 하나의 배송에 연결될 수 있으므로 many-to-one 관계를 갖는다. 또 오프라인 매장에서 구매하여 배송이 필요 없는 경우가 존재할 수도 있다.

영수증의 번호로 추적 번호를 식별할 수 있으므로 bill_ID가 기본키가 된다.

The ER diagram illustrates the following tables and their attributes:

- customer_payment**:
 - payment_number: VARCHAR(20) String NOT NULL (FK)
 - customer_ID: CHAR(12) String NOT NULL (FK)
- customer**:
 - customer_ID: CHAR(12) String NOT NULL (PK)
 - customer_name: VARCHAR(20) String NOT NULL
 - phone_number: VARCHAR(15) String NOT NULL
 - address: VARCHAR(30) String NOT NULL
- payment_bill**:
 - bill_ID: CHAR(14) String NOT NULL (FK)
 - payment_number: VARCHAR(20) String NOT NULL (FK)
- bill**:
 - bill_ID: CHAR(14) String NOT NULL (PK)
 - product_code: INTEGER Number NOT NULL
 - purchase_date: TIMESTAMP Datetime NOT NULL
- product**:
 - product_ID: VARCHAR(20) String NOT NULL (PK)
 - product_name: VARCHAR(20) String NOT NULL
 - production_date: DATE Datetime NOT NULL
 - product_price: INTEGER Number NOT NULL
 - manufacturer: VARCHAR(30) String NOT NULL
- product_sale**:
 - bill_ID: CHAR(14) String NOT NULL (FK)
 - product_ID: VARCHAR(20) String NOT NULL (FK)
- package_sale**:
 - bill_ID: CHAR(14) String NOT NULL (FK)
 - package_ID: VARCHAR(30) String NOT NULL (FK)
- shipping_info**:
 - bill_ID: CHAR(14) String NOT NULL (FK)
 - tracking_number: CHAR(12) String NOT NULL (FK)
- package**:
 - package_ID: VARCHAR(30) String NOT NULL (PK)
 - package_name: VARCHAR(20) String NOT NULL
 - package_price: INTEGER Number NOT NULL
- inventory**:
 - inventory_ID: VARCHAR(8) String NOT NULL (PK)
 - inventory_type: VARCHAR(10) String NOT NULL
 - region: VARCHAR(30) String NOT NULL
- new_order_inventory**:
 - new_order_ID: CHAR(12) String NOT NULL (FK)
 - inventory_ID: VARCHAR(8) String NOT NULL (FK)

Relationships are indicated by lines with crow's foot notation:

- customer_payment** to **customer**: 1 to P (FK)
- payment_bill** to **customer_payment**: 1 to P (FK)
- payment_bill** to **bill**: 1 to 1 (FK)
- product** to **product_sale**: 1 to P (FK)
- product** to **package_sale**: 1 to P (FK)
- product_sale** to **bill**: 1 to P (FK)
- package_sale** to **bill**: 1 to P (FK)
- shipping_info** to **bill**: 1 to P (FK)
- package** to **package_sale**: 1 to P (FK)
- inventory** to **new_order_inventory**: 1 to P (FK)

2. BCNF Dependency

기존의 Project 1에서 구현한 schema에서 BCNF를 만족하지 않는 테이블이 존재하지 않아 Decomposition은 수행하지 않았다. 대부분 Primary key만 결정자로 존재하였다.

대신 쿼리 수행을 위해 BCNF 조건을 충족하면서 일부 속성만 추가하였다. 각 테이블별로 BCNF를 만족하는지 살펴보자.

1) customer

| Attribute name | Domain | Constraints |
|----------------|-------------|----------------|
| customer_ID | CHAR(12) | (PK), NOT NULL |
| customer_name | VARCHAR(20) | NOT NULL |
| phone_number | VARCHAR(15) | NOT NULL |
| address | VARCHAR(30) | NOT NULL |

위 테이블에서 살펴볼 함수 종속성 ($\alpha \rightarrow \beta$)은 다음과 같다.

1) customer_ID \rightarrow (customer_name, phone_number, address)

2) phone_number \rightarrow (customer_ID, customer_name, address)

1)과 2) 모두 α 가 속성 전체를 식별할 수 있는 super key이므로 위 테이블은 BCNF를 만족한다. 이외의 모든 함수 종속성은 α 가 1)또는 2)의 결정자를 포함하므로 항상 super key가 된다.

2) payment

| Attribute name | Domain | Constraints |
|----------------|-------------|---------------------------------------|
| payment_number | VARCHAR(20) | (PK), NOT NULL |
| payment_type | VARCHAR(10) | NOT NULL |
| customer_ID | CHAR(12) | (FK \rightarrow customer), NOT NULL |

위 테이블에서 살펴볼 함수 종속성 ($\alpha \rightarrow \beta$)은 다음과 같다.

1) payment_number \rightarrow (payment_type, customer_ID)

2) (customer_ID, payment_type) \rightarrow (payment_number)

α 가 속성 전체를 식별할 수 있는 super key이므로 위 테이블은 BCNF를 만족한다. 이외의 모든 함수 종속성은 α 가 1) 또는 2)의 결정자를 포함하므로 항상 super key가 된다.

3) package

| Attribute name | Domain | Constraints |
|----------------|-------------|----------------|
| package_ID | VARCHAR(20) | (PK), NOT NULL |
| package_name | VARCHAR(20) | NOT NULL |
| package_price | INTEGER | NOT NULL |

위 테이블에서 살펴볼 함수 종속성 ($\alpha \rightarrow \beta$)은 다음과 같다.

1) package_ID \rightarrow (package_name, package_price)

α 가 속성 전체를 식별할 수 있는 super key이므로 위 테이블은 BCNF를 만족한다. 이외의 모든 함수 종속성은 α 가 1)의 결정자를 포함하므로 항상 super key가 된다.

4) inventory

| Attribute name | Domain | Constraints |
|----------------|-------------|----------------|
| inventory_ID | VARCHAR(6) | (PK), NOT NULL |
| inventory_type | VARCHAR(10) | NOT NULL |
| region | VARCHAR(30) | NOT NULL |

위 테이블에서 살펴볼 함수 종속성 ($\alpha \rightarrow \beta$)은 다음과 같다.

1) inventory_ID \rightarrow (inventory_type, region)

α 가 속성 전체를 식별할 수 있는 super key이므로 위 테이블은 BCNF를 만족한다. 이외의 모든 함수 종속성은 α 가 1)의 결정자를 포함하므로 항상 super key가 된다.

5) Product

| Attribute name | Domain | Constraints |
|-----------------|-------------|-------------------|
| product_ID | VARCHAR(20) | (PK), NOT NULL |
| product_name | VARCHAR(20) | NOT NULL |
| production_date | DATE | NOT NULL |
| product_price | INTEGER | NOT NULL |
| manufacturer | VARCHAR(30) | NOT NULL |
| inventory_ID | VARCHAR(6) | (FK -> inventory) |
| package_ID | VARCHAR(20) | (FK -> package) |

위 테이블에서 살퍼볼 함수 종속성 ($\alpha \rightarrow \beta$)은 다음과 같다.

- 1) product_ID \rightarrow (product_name, production_date, product_price, manufacturer, inventory_ID, package_ID)
- α 가 속성 전체를 식별할 수 있는 super key이므로 위 테이블은 BCNF를 만족한다. 이외의 모든 함수 종속성은 α 가 1)의 결정자를 포함하므로 항상 super key가 된다.

6) bill

| Attribute name | Domain | Constraints |
|-----------------|-------------|----------------------------|
| bill_ID | CHAR(14) | (PK), NOT NULL |
| product_count | INT | NOT NULL |
| purchase_date | TIMESTAMP | NOT NULL |
| customer_ID | CHAR(12) | (FK -> customer), NOT NULL |
| payment_number | VARCHAR(20) | (FK -> payment) |
| product_ID | VARCHAR(20) | (FK -> product) |
| package_ID | VARCHAR(20) | (FK -> package) |
| tracking_number | CHAR(12) | (FK -> shipment) |

위 테이블에서 살퍼볼 함수 종속성 ($\alpha \rightarrow \beta$)은 다음과 같다.

- 1) bill_ID \rightarrow (product_count, purchase_date, customer_ID, payment_number, product_ID, package_ID, tracking_number)
- α 가 속성 전체를 식별할 수 있는 super key이므로 위 테이블은 BCNF를 만족한다. 이외의 모든 함수 종속성은 α 가 1)의 결정자를 포함하므로 항상 super key가 된다.

7) reorder_list

| Attribute name | Domain | Constraints |
|----------------|-------------|---------------------------|
| reorder_ID | CHAR(12) | (PK), NOT NULL |
| is_filled | VARCHAR(3) | NOT NULL |
| fill_count | INT | NOT NULL |
| product_ID | VARCHAR(20) | (FK -> product), NOT NULL |
| inventory_ID | VARCHAR(6) | (FK -> product), NOT NULL |

위 테이블에서 살퍼볼 함수 종속성 ($\alpha \rightarrow \beta$)은 다음과 같다.

- 1) reorder_ID \rightarrow (is_filled, fill_count, product_ID, inventory_ID)

α 가 속성 전체를 식별할 수 있는 super key이므로 위 테이블은 BCNF를 만족한다. 이외의 모든 함수 종속성은 α 가 1)의 결정자를 포함하므로 항상 super key가 된다.

8) shipment

| Attribute name | Domain | Constraints |
|-----------------------|-------------|----------------|
| tracking_number | CHAR(12) | (PK), NOT NULL |
| shipping_company | VARCHAR(20) | NOT NULL |
| send_time | TIMESTAMP | NOT NULL |
| promised_deliver_time | TIMESTAMP | |
| is_delivered | VARCHAR(3) | NOT NULL |

위 테이블에서 살펴볼 함수 종속성 ($\alpha \rightarrow \beta$)은 다음과 같다.

1) tracking_number \rightarrow (shipping_company, send_time, promised_deliver_time, is_delivered)

α 가 속성 전체를 식별할 수 있는 super key이므로 위 테이블은 BCNF를 만족한다. 이외의 모든 함수 종속성은 α 가 1)의 결정자를 포함하므로 항상 super key가 된다.

9) product_inventory

| Attribute name | Domain | Constraints |
|----------------|-------------|--|
| inventory_ID | VARCHAR(6) | (PK), (FK \rightarrow inventory), NOT NULL |
| product_ID | VARCHAR(20) | (PK), (FK \rightarrow product), NOT NULL |
| stock | INT | NOT NULL |

위 테이블에서 살펴볼 함수 종속성 ($\alpha \rightarrow \beta$)은 다음과 같다.

1) (inventory_ID, product_ID) \rightarrow stock

α 가 속성 전체를 식별할 수 있는 super key이므로 위 테이블은 BCNF를 만족한다. 이외의 모든 함수 종속성은 α 가 1)의 결정자를 포함하므로 항상 super key가 된다.

3. CRUD Implementation

테이블을 생성하는 Create, 튜플을 추가하는 Insert, 튜플을 삭제하는 Delete, 테이블을 제거하는 Drop 쿼리들은 모두 txt파일로 저장하여, visual studio 2019에서 수행되는 C++ 파일에서 파일 입출력으로 불러오는 방식이다.

3.1. Create

```
create - Windows 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
create table customer (customer_ID char(12) not null, customer_name varchar(20) not null, phone_number varchar(15) not null, address varchar(30) not null, primary key(customer_ID));
create table payment (payment_number varchar(20) not null, payment_type varchar(10) not null, customer_ID char(12) not null, primary key(payment_number), foreign key(customer_ID) references customer(customer_ID) on delete cascade);
create table package (package_ID varchar(20) not null, package_name varchar(20) not null, package_price int not null, primary key(package_ID));
create table inventory (inventory_ID varchar(6) not null, inventory_type varchar(10) not null, region varchar(30) not null, primary key(inventory_ID));
create table product (product_ID varchar(20) not null, product_name varchar(20) not null, production_date date not null, product_price int not null, manufacturer varchar(30) not null, inventory_ID varchar(6), package_ID varchar(20), primary key(product_ID), foreign key(inventory_ID) references inventory(inventory_ID) on delete set null, foreign key(package_ID) references package(package_ID) on delete set null);
create table bill (bill_ID char(14) not null, product_count int not null, purchase_date timestamp not null, customer_ID char(12) not null, payment_number varchar(20), product_ID varchar(20), package_ID varchar(20), tracking_number char(12), primary key(bill_ID), foreign key(customer_ID) references customer(customer_ID) on delete cascade, foreign key(payment_number) references payment(payment_number) on delete cascade, foreign key(product_ID) references product(product_ID) on delete set null, foreign key(package_ID) references package(package_ID) on delete set null, foreign key(tracking_number) references shipment(tracking_number) on delete set null);
create table reorder_list (reorder_ID char(12) not null, is_filled varchar(3) not null, fill_count int not null, product_ID varchar(20) not null, inventory_ID varchar(6) not null, primary key(reorder_ID), foreign key(product_ID) references product(product_ID) on delete cascade, foreign key(inventory_ID) references inventory(inventory_ID) on delete cascade);
create table shipment (tracking_number char(12) not null, shipping_company varchar(20) not null, send_time timestamp not null, promised_deliver_time timestamp, is_delivered varchar(3) not null, primary key(tracking_number));
create table product_inventory (inventory_ID varchar(6) not null, product_ID varchar(20) not null, stock int not null, primary key(inventory_ID, product_ID), foreign key(inventory_ID) references inventory(inventory_ID) on delete cascade, foreign key(product_ID) references product(product_ID) on delete cascade);
```

보기 좋게 정리된 쿼리문은 다음과 같다. (실제로 20161230_1.txt로 저장하였다)

null값을 가질 수 있는 외래키의 경우, 참조하는 속성값이 사라지면 null값이 되도록(set null) 설정하였고, not null인 특성을 갖는 외래키의 경우, 참조하는 속성값이 사라지면 해당 튜플을 삭제하도록(cascade) 하였다.

```
CREATE TABLE customer (
    customer_ID CHAR(12) NOT NULL,
    customer_name VARCHAR(20) NOT NULL,
    phone_number VARCHAR(15) NOT NULL,
    address VARCHAR(30) NOT NULL,
    PRIMARY KEY (customer_ID)
);

CREATE TABLE payment (
    payment_number VARCHAR(20) NOT NULL,
    payment_type VARCHAR(10) NOT NULL,
    customer_ID CHAR(12) NOT NULL,
    PRIMARY KEY (payment_number),
    FOREIGN KEY (customer_ID)
        REFERENCES customer (customer_ID)
        ON DELETE CASCADE
);

CREATE TABLE package (
    package_ID VARCHAR(20) NOT NULL,
    package_name VARCHAR(20) NOT NULL,
    package_price INT NOT NULL,
    PRIMARY KEY (package_ID)
);

CREATE TABLE inventory (
    inventory_ID VARCHAR(6) NOT NULL,
    inventory_type VARCHAR(10) NOT NULL,
    region VARCHAR(30) NOT NULL,
    PRIMARY KEY (inventory_ID)
);

CREATE TABLE product (
    product_ID VARCHAR(20) NOT NULL,
    product_name VARCHAR(20) NOT NULL,
    production_date DATE NOT NULL,
    product_price INT NOT NULL,
    manufacturer VARCHAR(30) NOT NULL,
    inventory_ID VARCHAR(6),
    package_ID VARCHAR(20),
    PRIMARY KEY (product_ID),
    FOREIGN KEY (inventory_ID)
        REFERENCES inventory (inventory_ID)
        ON DELETE SET NULL,
    FOREIGN KEY (package_ID)
        REFERENCES package (package_ID)
        ON DELETE SET NULL
);

CREATE TABLE bill (
    bill_ID CHAR(14) NOT NULL,
    product_count INT NOT NULL,
    purchase_date TIMESTAMP NOT NULL,
    customer_ID CHAR(12) NOT NULL,
    payment_number VARCHAR(20),
    product_ID VARCHAR(20),
    package_ID VARCHAR(20),
    tracking_number CHAR(12),
    PRIMARY KEY (bill_ID),
    FOREIGN KEY (customer_ID)
        REFERENCES customer (customer_ID)
        ON DELETE CASCADE,
    FOREIGN KEY (payment_number)
        REFERENCES payment (payment_number)
        ON DELETE CASCADE,
    FOREIGN KEY (product_ID)
        REFERENCES product (product_ID)
        ON DELETE SET NULL,
    FOREIGN KEY (package_ID)
        REFERENCES package (package_ID)
        ON DELETE SET NULL,
    foreign key (tracking_number)
        references shipment (tracking_number)
        on delete set null
);
```

```

CREATE TABLE reorder_list (
    reorder_ID CHAR(12) NOT NULL,
    is_filled VARCHAR(3) NOT NULL,
    fill_count INT NOT NULL,
    product_ID VARCHAR(20) NOT NULL,
    inventory_ID VARCHAR(6) NOT NULL,
    PRIMARY KEY (reorder_ID),
    FOREIGN KEY (product_ID)
        REFERENCES product (product_ID)
        ON DELETE CASCADE,
    FOREIGN KEY (inventory_ID)
        REFERENCES inventory (inventory_ID)
        ON DELETE CASCADE
);


CREATE TABLE shipment (
    tracking_number CHAR(12) NOT NULL,
    shipping_company VARCHAR(20) NOT NULL,
    send_time TIMESTAMP NOT NULL,
    promised_deliver_time TIMESTAMP,
    is_delivered VARCHAR(3) NOT NULL,
    PRIMARY KEY (tracking_number)
);

CREATE TABLE product_inventory (
    inventory_ID VARCHAR(6) NOT NULL,
    product_ID VARCHAR(20) NOT NULL,
    stock INT NOT NULL,
    PRIMARY KEY (inventory_ID, product_ID),
    FOREIGN KEY (inventory_ID)
        REFERENCES inventory (inventory_ID)
        ON DELETE CASCADE,
    FOREIGN KEY (product_ID)
        REFERENCES product (product_ID)
        ON DELETE CASCADE
);

```

3.2. Insert

각 테이블 당 튜플이 최소 15개 이상 존재하도록 데이터를 추가하였다. (실제로 20161230_2.txt로 저장하였다)

 insert - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)


```

insert into customer values ('220401000001', 'Park', '01012345678', 'Mapo-gu');
insert into customer values ('210512000134', 'Kim', '01031486472', 'Gangnam-gu');
insert into customer values ('201213000047', 'Lee', '01034817946', 'Seocho-gu');
insert into customer values ('200602000007', 'Cho', '01097315468', 'Seodaemun-gu');
insert into customer values ('211130000072', 'Choi', '01067231975', 'Gimhae-si');
insert into customer values ('220102000002', 'Kang', '01084631482', 'Jongno-gu');
insert into customer values ('220325000121', 'Jeong', '01064024874', 'Gyeongju-si');
insert into customer values ('191026000056', 'Ko', '01076054813', 'Jeju-si');
insert into customer values ('200220000723', 'Son', '01082108031', 'Incheon-si');
insert into customer values ('210730000021', 'Yoo', '01097080234', 'Nowon-gu');
insert into customer values ('220207000030', 'Ahn', '01030879604', 'Daejeon-si');
insert into customer values ('200704000067', 'Shin', '01047024972', 'Suwon-si');
insert into customer values ('220520000003', 'Hong', '01071349412', 'Yeongdeungpo-gu');
insert into customer values ('200926000034', 'Yang', '01064805232', 'Songpa-gu');
insert into customer values ('220418000007', 'Lim', '01024801311', 'Goyang-si');
insert into payment values ('1002-254-884509', 'account', '220401000001');
insert into payment values ('012513-42-601237', 'account', '210512000134');
insert into payment values ('4330-1247-6312-4801', 'credit', '210512000134');
insert into payment values ('10-310802-10687', 'account', '201213000047');
insert into payment values ('7621-3148-6123-4872', 'credit', '200602000007');
insert into payment values ('460123-75-1309701', 'account', '211130000072');
insert into payment values ('943154-03-157130', 'account', '220102000002');
insert into payment values ('5317-6134-9645-3149', 'debit', '220102000002');
insert into payment values ('4021643-28-097457', 'account', '220325000121');
insert into payment values ('3204-6524-7861-4344', 'credit', '191026000056');
insert into payment values ('3310-4846-2422-7219', 'debit', '191026000056');
insert into payment values ('621503-84-960211', 'account', '200220000723');
insert into payment values ('8843-1211-7613-4653', 'credit', '210730000021');
insert into payment values ('33-451397-642221', 'account', '220207000030');

```

3.3. Delete & Drop

DBMS가 문제없이 수행되도록 프로그램이 종료될 때 Delete와 Drop을 해준다. (실제로 20161230_3.txt로 저장하였다)

 *delete_drop - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

```
delete from product_inventory
delete from shipment
delete from reorder_list
delete from bill
delete from product
delete from inventory
delete from package
delete from payment
delete from customer
drop table product_inventory
drop table shipment
drop table reorder_list
drop table bill
drop table product
drop table inventory
drop table package
drop table payment
drop table customer
```

4. ODBC C Language

테이블과 튜플이 모두 생성되고 나면, 주어진 쿼리를 수행할 수 있는 C code를 작성한다. 기본적으로 쿼리를 수행하는 과정은 다음과 같다.

1. mysql에 host 정보와 user ID, password, DB명을 이용하여 연결을 수행한다.

```
connection = mysql_real_connect(&conn, host, user, pw, db, 3306, (const char*)NULL, 0);
```

2. 연결에 성공하면 테이블을 생성하고 튜플을 삽입한다. 미리 작성해둔 txt 파일을 파일 입출력으로 불러와 한 줄씩 읽는다. (실제로 create.txt가 아니라 20161230_1.txt, insert.txt가 아니라 20161230_2.txt이다)

기본적으로 mysql_query 함수를 통해 모든 쿼리를 수행하며, 반환값으로 0을 가지는 경우 쿼리가 올바르게 수행되었다는 의미이고, 1을 가지는 경우 오류로 인해 쿼리가 올바르게 수행되지 않은 경우이다.

```
/* Create Table */
ifstream c_in("create.txt");
if (!c_in) return cout << "create file not exist", 0;
while (getline(c_in, s)) {
    state = mysql_query(connection, s.c_str());
}
c_in.close();

/* Insert Tuples */
ifstream i_in("insert.txt");
if (!i_in) return cout << "insert file not exist", 0;
while (getline(i_in, s)) {
    state = mysql_query(connection, s.c_str());
}
i_in.close();
```

3. 사용자는 원하는 TYPE의 쿼리를 선택한다. 서브쿼리가 존재하는 경우도 있으며, 서브쿼리 또한 사용자의 입력으로 수행된다. 존재하지 않는 번호를 입력한 경우 에러메시지를 출력하고 메인 메뉴로 돌아간다.

```
----- SELECT QUERY TYPES -----
1. TYPE 1
2. TYPE 2
3. TYPE 3
4. TYPE 4
5. TYPE 5
6. TYPE 6
7. TYPE 7
0. QUIT
```

4. 0을 입력하는 경우 튜플과 테이블이 모두 삭제된 후 프로그램이 종료된다. (실제로 20161230_3.txt이다)

```
/* Delete and Drop table */
ifstream d_in("delete_drop.txt");
if (!d_in) return cout << "delete_drop file not exist", 0;
while (getline(d_in, s)) {
    state = mysql_query(connection, s.c_str());
}
d_in.close();
mysql_close(connection);
```

4.1. TYPE 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. (1)

USPS 회사에 의해 운송되는, 추적 번호가 X 인 상품이 사고로 파괴되었다고 할 때, 고객에게 이 사실을 알려야 하므로 고객의 연락처를 알아내야 한다.

추적 번호는 회사와 상관없이 항상 유일하므로 X를 이용해서 고객의 연락처를 구한다.

만약 X 가 존재하지 않는 추적 번호인 경우, "Tracking number X Doesn't exist!" 메시지를 출력하고 메인 메뉴로 돌아간다. (77 ~ 98 줄)

```
cout << "Which X? : ";
string x; cin >> x;
query = "select customer.phone_number from customer natural join (bill natural join shipment) where tracking_number = '" + x + "'";
state = mysql_query(connection, query.c_str());
cout << "[phone_number]\n\n";
int cnt = 0;
if (state == 0)
{
    sql_result = mysql_store_result(connection);
    while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
    {
        printf("%s\n", sql_row[0]);
        cnt++;
    }
    mysql_free_result(sql_result);
}
if (cnt == 0) {
    cout << "Tracking number X Doesn't exist! \n";
    continue;
}
```

쿼리문은 다음과 같다.

```
SELECT customer.phone_number
FROM customer NATURAL JOIN (bill NATURAL JOIN shipment)
WHERE tracking_number = x
```

bill 과 shipment 를 natural join 하면 tracking_number 가 같은 튜플끼리 join 된다. 그리고 customer 와 natural join 하면 customer_ID 가 같은 튜플끼리 join 된다. 따라서 결국 tracking_number 를 통해 고객의 전화번호를 식별할 수 있다.

4.1.1. TYPE 1-1

- (TYPE 1-1) Then find the contents of that shipment and create a new shipment of replacement items. (2)

고객에게 전달한 뒤, 동일한 상품을 다시 배송해주어야 한다. 이를 위해 파괴된 배송에 대한 정보를 찾고, 새로운 배송을 생성해주어야 한다. 먼저, 파괴된 배송에 대한 정보를 찾는 쿼리를 수행한다. (103 ~ 119 줄)

```
cout << "\n----- TYPE 1-1 ----- \n\n";
cout << "*** Then find the contents of that shipment and create a new shipment of replacement items. **\n\n";
query = "select tracking_number, product_count, customer_Id, payment_number, product_ID,
package_ID, shipping_company from bill natural join shipment where tracking_number = '" + x + "'";
state = mysql_query(connection, query.c_str());
// shipment의 정보 출력
cout << "[tracking_number / product_count / customer_id / payment_number / product_ID / package_ID / shipping_company]\n\n";
string comp; // 배송 회사
if (state == 0)
{
    sql_result = mysql_store_result(connection);
    while ((sql_row = mysql_fetch_row(sql_result)) != NULL){
        printf("%s / %s / %s / %s / %s / %s\n", sql_row[0], sql_row[1], sql_row[2], sql_row[3], sql_row[4], sql_row[5], sql_row[6]);
        comp = sql_row[6];
    }
    mysql_free_result(sql_result);
}
```

쿼리문은 다음과 같다.

```
select tracking_number, product_count, customer_Id, payment_number, product_ID, package_ID, shipping_company
from bill natural join shipment
where tracking_number = x
```

bill 과 shipment 를 natural join 하게 되면 tracking_number 가 같은 튜플끼리 join 된다. 그리고 추적 번호는 shipment 테이블에서 유일하므로 결국 tracking_number 를 통해 배송에 대한 정보를 얻을 수 있다.

그 다음, 새로운 배송을 생성해준다. 먼저, "select now()" 쿼리를 통해서 현재 시간을 구한다. 그리고 현재 시간을 배송 시작 시간으로 설정하고, 도착 예정 일자를 7 월 5 일로 설정한다. (120 ~ 158 줄)

```
query = "select now()"; // 현재 시간
state = mysql_query(connection, query.c_str());
string date;
if (state == 0)
{
    sql_result = mysql_store_result(connection);
    while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
    {
        date = sql_row[0];
    }
    mysql_free_result(sql_result);
}
string arr_date = date; // 도착 예정 일자 7/5
arr_date[6] = '7';
arr_date[8] = '0';
arr_date[9] = '5';
string track_num = ""; // 새 추적 번호
track_num.push_back(date[2]);
track_num.push_back(date[3]);
track_num.push_back(date[5]);
track_num.push_back(date[6]);
track_num.push_back(date[8]);
track_num.push_back(date[9]);
string cnt = "";
int tmp = ship_count;
for (int i = 1; i <= 6; i++) {
    cnt = (char)((tmp % 10) + '0') + cnt;
    tmp /= 10;
}
track_num += cnt;
//새 배송 생성
query = "insert into shipment values ('" + track_num + "', '" + comp + "', '" + date + "', '" + arr_date + "', 'NO')";
state = mysql_query(connection, query.c_str());
if (state == 0)
{
    sql_result = mysql_store_result(connection);
    mysql_free_result(sql_result);
}
cout << "\nNew shipment of replacement items added!\n\n";
```

쿼리문은 다음과 같다.

```
insert into shipment values (track_num, comp, date, arr_date, 'NO');
```

마지막으로, 결제 정보에 있는 추적 번호를 새로운 추적 번호로 수정해준다. (159 ~ 166 줄)

```
query = "update bill set tracking_number = '" + track_num + "' where tracking_number = '" + x + "'";
state = mysql_query(connection, query.c_str());
if (state == 0)
{
    sql_result = mysql_store_result(connection);
    mysql_free_result(sql_result);
}
```

쿼리문은 다음과 같다.

```
update bill set tracking_number = track_num where tracking_number = x
```

4.2. TYPE 2

(TYPE 2) Find the customer who has bought the most (by price) in the past year. (3)

작년에 가장 많은 지출을 한 고객을 찾아야 한다. 상품과 패키지를 모두 고려해야 하기 때문에 상품과 패키지 각각에서 고객별로 지출의 총액을 미리 구해 두었다. 고객의 ID와 이름, 작년에 지출한 총 금액을 출력한다. (173 ~ 205 줄)

쿼리가 길어 코드 상에서 쿼리문은 생략하겠다.

```
state = mysql_query(connection, query.c_str());
cout << "[customer ID / total payment]\n\n";
string cid = "";
if (state == 0)
{
    sql_result = mysql_store_result(connection);
    while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
    {
        printf("%s / %s\n", sql_row[0], sql_row[1]);
        cid = sql_row[0];
    }
    mysql_free_result(sql_result);
}
query = "select customer_name from customer where customer_ID = '" + cid + "'";
state = mysql_query(connection, query.c_str());
cout << "[customer name]\n\n";
if (state == 0)
{
    sql_result = mysql_store_result(connection);
    while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
    {
        printf("%s\n", sql_row[0]);
    }
    mysql_free_result(sql_result);
}
```


쿼리문은 다음과 같다.

```
with product_sum(c_id, sum) as
  (select customer_ID, sum(product_price*product_count)
   from bill, product
   where bill.product_ID = product.product_ID and
   bill.purchase_date between '2021-01-01 00:00:00' and '2021-12-31 23:59:59'
   group by customer_ID),
package_sum(c_id, sum) as
  (select customer_ID, sum(package_price*product_count)
   from bill, package
   where bill.package_ID = package.package_ID and
   bill.purchase_date between '2021-01-01 00:00:00' and '2021-12-31 23:59:59'
   group by customer_ID)
select product_sum.c_id, (product_sum.sum + if(product_sum.c_id = package_sum.c_id, package_sum.sum, 0)) as total
from product_sum, package_sum
order by total desc
limit 1
```

상품과 패키지는 별도의 테이블로 관리하고, 상품과 패키지 지출 모두 고려해야 하기 때문에 with 문을 이용하여 각각의 테이블에서 고객들의 지출의 합을 계산해둔다. 그 다음, product_sum 과 package_sum 테이블을 cartesian product 을 하면, product_sum 의 고객 ID 와 package_sum 의 고객 ID 가 같은 경우만 package_sum 의 sum 을 해당 고객에게 더해주면 되므로 if 문을 걸어 이외의 경우는 모두 0 으로 계산하도록 한다.

4.2.1. TYPE 2-1

(TYPE 2-1) Then find the product that the customer bought the most. (4)

그 다음, 해당 고객이 가장 많이 구매한 상품을 찾아야 한다. 이전의 쿼리를 통해 고객의 ID 를 구했으므로 ID 를 이용해서 영수증 테이블에서 해당 고객이 구매한 모든 정보를 가져온 다음, 상품 ID 별로 합쳐서 개수 기준으로 내림차순 정렬한다. 그리고 제일 상단에 위치한 상품의 이름과 구매한 개수를 출력한다. (211 ~ 226 줄)

```
cout << "*** Then find the product that the customer bought the most **\n\n";
query = "select product_name, sum(product_count) as cnt ";
query += "from bill, product where bill.product_ID = product.product_ID and customer_ID = '" + cid + "' ";
query += "group by bill.product_ID order by cnt desc limit 1";
state = mysql_query(connection, query.c_str());
cout << "[product name / count]\n\n";
if (state == 0)
{
  sql_result = mysql_store_result(connection);
  while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
  {
    printf("%s / %s\n", sql_row[0], sql_row[1]);
  }
  mysql_free_result(sql_result);
}
```

쿼리문은 다음과 같다.

```
select product_name, sum(product_count) as cnt
from bill, product
where bill.product_ID = product.product_ID and customer_ID = cid
group by bill.product_ID
order by cnt desc
limit 1
```

bill 테이블과 product 테이블을 cartesian product 하고 product_ID 가 같은 튜플만 뽑아내면, product_ID 가 같은 튜플에 대해서만 join 연산을 한 효과를 얻는다. 그리고 이전에 구한 고객의 ID 를 통해서 해당 고객이 구매한 상품들을 구매 개수 기준으로 내림차순 정렬한 뒤 최상단의 상품을 구하면 가장 많이 구매한 상품을 얻을 수 있다.

4.3 TYPE 3

(TYPE 3) Find all products sold in the past year. (5)

작년에 팔린 모든 상품을 구해야 한다. 영수증의 판매 일자를 기준으로 2021 년에 판매된 모든 상품의 이름을 출력한다. (234 ~ 247 줄)

```
cout << "*** Find all products sold in the past year. **\n";
query = "select P.product_name from product as P, bill as B ";
query += "where P.product_ID = B.product_ID and purchase_date
        between '2021-01-01 00:00:00' and '2021-12-31 23:59:59' group by P.product_name ";
state = mysql_query(connection, query.c_str());
cout << "[product name] \n\n";
if (state == 0)
{
    sql_result = mysql_store_result(connection);
    while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
    {
        printf("%s\n", sql_row[0]);
    }
    mysql_free_result(sql_result);
}
```

쿼리문은 다음과 같다.

```
select P.product_name
from product as P, bill as B
where P.product_ID = B.product_ID and purchase_date between '2021-01-01 00:00:00' and '2021-12-31 23:59:59'
group by P.product_name
```

product 와 bill 을 cartesian product 하고, product_ID 가 같은 튜플만 뽑아내면 실제로 판매된 상품 목록을 얻을 수 있다. 그 중 2021 년에 판매된 튜플만 뽑아내면 2021 년에 판매된 모든 상품의 이름을 구할 수 있다.

4.3.1. TYPE 3-1

(TYPE 3-1) Then find the top k products by dollar-amount sold. (6)

그 중, 판매 금액이 가장 많은 상위 k 개의 상품을 구해야 한다. k 는 사용자입력으로 직접 입력하는 방식이고, 상품의 이름과 판매된 총액을 내림차순으로 출력한다. (256 ~ 272 줄)

```
cout << "Which k? : ";
string k; cin >> k;
query = "select P.product_name, sum(product_count*product_price) as sold from product as P, bill as B ";
query += "where P.product_ID = B.product_ID and purchase_date between '2021-01-01 00:00:00' and '2021-12-31 23:59:59'";
query += "group by P.product_name order by sold desc limit ";
query += k;

state = mysql_query(connection, query.c_str());
cout << "[product name / dollar-amount sold]\n\n";
if (state == 0)
{
    sql_result = mysql_store_result(connection);
    while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
    {
        printf("%s / %s \n", sql_row[0], sql_row[1]);
    }
    mysql_free_result(sql_result);
}
```

쿼리문은 다음과 같다.

```
select P.product_name, sum(product_count*product_price) as sold
from product as P, bill as B
where P.product_ID = B.product_ID and
      purchase_date between '2021-01-01 00:00:00' and '2021-12-31 23:59:59'
group by P.product_name
order by sold desc
limit k
```

상품별로 그룹을 지은 다음, 판매된 개수 * 상품의 가격으로 각 상품별 판매된 총액을 계산한다. 그리고 내림차순으로 정렬한 뒤 상위 k 개의 상품을 출력한다.

4.3.2. TYPE 3-2

(TYPE 3-2) And then find the top 10% products by dollar-amount sold. (7)

이번에는 판매 금액이 가장 많은 상위 k 개가 아닌, 상위 10%의 상품을 구해야 한다. 3-1 과 쿼리만 다르기 때문에 코드는 생략하겠다.

쿼리문은 다음과 같다.

```

select p_name, sold
from
    (select p_name, sold, percent_rank() over (order by sold desc) as per
      from (select P.product_name as p_name, sum(product_count*product_price) as sold
            from product as P, bill as B
            where P.product_ID = B.product_ID and
                  purchase_date between '2021-01-01 00:00:00' and '2021-12-31 23:59:59'
            group by P.product_name
            order by sold desc) T1
    ) T2
where T2.per <= 0.1

```

상위 k%의 데이터를 뽑아 내기 위해서는 percent_rank() 함수를 사용한다.

먼저, 상품별로 그룹을 지은 다음 판매된 개수 * 상품의 가격으로 각 상품별 판매된 총액을 계산하는 부분은 3-1)과 동일하다. 3-2)에서는 이 결과 테이블을 가지고 각 튜플 별로 총액을 기준으로 위에서 몇 번째에 위치하는지 백분율(per)을 계산한다. 백분율은 0 ~ 1 사이의 소숫값을 가지며, 이 결과로 나온 테이블이 T2이다.

마지막으로 T2 테이블에서 per값이 0.1 이하인 튜플을 선택하면 결과적으로 상위 10% 내의 상품을 구할 수 있게 된다.

4.4. TYPE 4

(TYPE 4) Find all products by unit sales in the past year. (8)

작년에 단위 기준으로 판매된 모든 상품을 구해야 한다. 결과적으로 3 번 쿼리와 동일하기 때문에 코드와 쿼리는 생략하겠다. (301 ~ 315 줄)

4.4.1. TYPE 4-1

(TYPE 4-1) Then find the top k products by unit sales. (9)

그 중, 개수 기준으로 가장 많이 팔린 상위 k 개의 상품을 구해야 한다. k 는 사용자가 직접 입력하는 방식이고, 3-1 쿼리에서 총액을 계산하는 부분을 개수로 바꿔주기만 하면 된다.

```

cout << "Which k? : ";
string k; cin >> k;
query = "select P.product_name, sum(B.product_count) as unit from product as P, bill as B ";
query += "where P.product_ID = B.product_ID and purchase_date between '2021-01-01 00:00:00' and '2021-12-31 23:59:59' ";
query += "group by P.product_name order by unit desc limit ";
query += k;

state = mysql_query(connection, query.c_str());
cout << "[product name / unit-sales]\n\n";
if (state == 0)
{
    sql_result = mysql_store_result(connection);
    while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
    {
        printf("%s / %s \n", sql_row[0], sql_row[1]);
    }
    mysql_free_result(sql_result);
}

```

쿼리문은 다음과 같다.

```
select P.product_name, sum(B.product_count) as unit
from product as P, bill as B
where P.product_ID = B.product_ID and purchase_date between '2021-01-01 00:00:00' and '2021-12-31 23:59:59'
group by P.product_name
order by unit desc
limit k
```

3-1 과 거의 동일하다.

4.4.2. TYPE 4-2

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

이번에는 개수 기준으로 가장 많이 팔린 상위 10%의 상품을 구해야 한다. 마찬가지로 3-2 쿼리에서 총액을 계산하는 부분을 개수로 바꿔주기만 하면 된다. 쿼리부분이 길어 코드상에선 생략하겠다. (344 ~ 361 줄)

```
cout << "*** And then find the top 10% products by unit sales. **\n";
//query = ""
state = mysql_query(connection, query.c_str());
cout << "[product name / unit-sales]\n\n";
if (state == 0)
{
    sql_result = mysql_store_result(connection);
    while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
    {
        printf("%s / %s \n", sql_row[0], sql_row[1]);
    }
    mysql_free_result(sql_result);
}
```

쿼리문은 다음과 같다.

```
select p_name, sold
from (select p_name, sold, percent_rank() over (order by sold desc) as per
      from (select P.product_name as p_name, sum(product_count) as sold
            from product as P, bill as B
            where P.product_ID = B.product_ID and
                  purchase_date between '2021-01-01 00:00:00' and '2021-12-31 23:59:59'
            group by P.product_name
            order by sold desc) T1
      ) T2
where T2.per <= 0.1
```

마찬가지로 3-2 와 거의 동일하다.

4.5. TYPE 5

(TYPE 5) Find those products that are out-of-stock at every store in California. (11)

California 의 모든 매장에서 품절된 상품을 구해야 한다. 재고를 저장하고 있는 product_inventory 테이블을 이용하여 지역이 California 인 매장에 있는 상품들 각각의 재고를 모두 더한 값이 0 인 상품을 구하면 된다. (368 ~ 382 줄)

```
query = "select product_name from product natural join(product_inventory natural join inventory)";
query += "where region = 'California' group by product_ID having sum(stock) = 0";
state = mysql_query(connection, query.c_str());
cout << "[product name] \n\n";
if (state == 0)
{
    sql_result = mysql_store_result(connection);
    while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
    {
        printf("%s \n", sql_row[0]);
    }
    mysql_free_result(sql_result);
}
```

쿼리문은 다음과 같다.

```
select product_name
from product natural join(product_inventory natural join inventory)
where region = 'California'
group by product_ID
having sum(stock) = 0
```

product_inventory와 inventory를 natural join하면 inventory_ID가 같은 튜플끼리 join되어, 각 inventory에 어떤 상품들이 얼마나 존재하는지 알 수 있다. 그 다음, 상품의 이름을 알기 위해 product 와 natural join 을 하고, 각 상품별로 group 을 지어 재고의 합을 계산했을 때 0 인 경우가 캘리포니아의 모든 상점에 재고가 없다는 의미가 된다.

4.6. TYPE 6

(TYPE 6) Find those packages that were not delivered within the promised time. (12)

도착 예정 시간 내에 도착하지 않은 패키지들을 구해야 한다. 현재 시간보다 도착 예정시간이 앞서지만, 아직 도착하지 않은 패키지들을 구하면 된다. (388 ~ 401 줄)

```
query = "select package_name, customer_ID, promised_deliver_time, shipping_company from package natural join(bill natural join shipment) ";
query += "where is_delivered = 'NO' and promised_deliver_time < now()";
state = mysql_query(connection, query.c_str());
cout << "[package_name / customer_ID / promised_deliver_time / shipping_company] \n\n";
if (state == 0)
{
    sql_result = mysql_store_result(connection);
    while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
    {
        printf("%s / %s / %s / %s\n", sql_row[0], sql_row[1], sql_row[2], sql_row[3]);
    }
    mysql_free_result(sql_result);
}
```

쿼리문은 다음과 같다.

```
select package_name, customer_ID, promised_deliver_time, shipping_company
from package natural join (bill natural join shipment)
where is_delivered = 'NO' and promised_deliver_time < now()
```

현재 시간은 now() 함수로 구할 수 있다. bill과 shipment를 natural join하면 배송되지 않은 패키지의 ID, 고객의 ID를 얻을 수 있다. 그리고 package와 natural join을 하면 패키지의 ID를 이용해서 패키지의 이름을 얻을 수 있다.

4.7 TYPE 7

(TYPE 7) Generate the bill for each customer for the past month. (13)

지난 달에 구매한 기록이 있는 고객들을 위해서 영수증을 생성해야 한다. 따라서 구매 기록 중 구매 일자가 지난 달인 모든 튜플을 출력하면 된다. (405 ~ 421줄)

```
query = "select * from bill where year(purchase_date) = year(current_date - interval 1 month) ";
query += "and month(purchase_date) = month(current_date - interval 1 month)";

state = mysql_query(connection, query.c_str());
cout << "[bill_ID / product_count / purchase_date / customer_ID / payment_number / product_ID / package_ID / tracking_number] \n\n";
if (state == 0)
{
    sql_result = mysql_store_result(connection);
    while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
    {
        printf("%s / %s / %s / %s / %s / %s / %s / %s\n", sql_row[0], sql_row[1], sql_row[2], sql_row[3], sql_row[4], sql_row[5], sql_row[6], sql_row[7]);
    }
    mysql_free_result(sql_result);
}
```

쿼리문은 다음과 같다.

```
select *
from bill
where year(purchase_date) = year(current_date - interval 1 month) and
      month(purchase_date) = month(current_date - interval 1 month)
```

현재 날짜는 current_date로 얻을 수 있다. 그리고 지난 달을 구하기 위해선 현재 날짜의 '월'을 1만큼 빼주면 된다. 이 때, 1월인 경우는 연도 또한 고려를 해주어야 하기 때문에 year도 마찬가지로 한 달을 뺀 연도로 계산을 해준다.

