Project 2

데이터베이스시스템

20200110 정태현

1. BCNF Decomposition

먼저 모든 릴레이션이 BCNF인지 아닌지 여부를 알기 위해서 수업시간에 배운 두가지 고려 사항이 있다. 첫 째, 주어진 릴레이션의 F 내에 존재하는 FD(a->B)가 trivial한가(B $\subseteq$ a) 이고, 둘 째는 첫 번째 조건에서의 a가 슈퍼키인지 확인하는 것이다. 이렇게 BCNF를 만족하는 릴레이션은 결국 1NF(제 1정규형), 2NF(제 2정규형), 3NF(제 3정규형)을 당연하게 만족한다. 이번 과제에서 모든 릴레이션이 BCNF를 만족하는지 여부를 확인하기 위한 과정을 좀 더 세분화하여 단계별로 확인할 예정이다. 그 과정은 다음과 같다.

logical 스키마의 릴레이션이라면 릴레이션 내에서 중복 값 속성을 인정하지 않기 때문에 모든 속성의 값들이 단일 값임은 자명하기에 1NF은 무조건 만족한다. 그리고 난 뒤 2NF를 만족하는지 확인해야 하는데, 이는 각 릴레이션이 모두 완전함수 종속(Full Functional Dependency)을 갖는지 확인해야 한다. 따라서 부분함수 종속(Partial Functional Dependency)이 있다면 이를 해소해야 2NF를 만족할 수 있다. 그리고 2NF를 만족한 릴레이션에서 이행적 함수종속(Transitive Functional Dependency)이 있다면 이를 해소해야 3NF(제 3정규형)를 만족할 수 있다. 마지막으로 이 3NF를 만족하고 있는 릴레이션 내에서 함수적 종속관계에서 결정자로서 역할을 하는 모든 결정자들이 반드시 슈퍼키라는 것을 확인하면 이 릴레이션은 BCNF(보이스-코드 정규형)를 만족하는 것이다. 따라서 1차 과제물로 제출했던 릴레이션 스키마를 전부 확인해 BCNF를 만족하는지 체크해야 한다.

이러한 일련의 과정들은 학교 수업에서 배운 함수 종속성(FD)을 이용하여 BCNF를 만족하는지 확인하는 방법을 단계별로 세분화 한 것이다. 즉 앞으로 하는 BCNF Decomposition에서 진행하는 세부 단계들은 주어진 릴레이션의 집합 F 내에 존재하는 모든 FD(a->B)가 trivial 한지(B $\subseteq$ a)여부를 확인하고, 두 번째로 해당 F안에 있는 FD들에서의 결정자인 a가 슈퍼키인지 여부를 확인하는 과정을 세분화 한 것이다. 따라서 이 과정에서 다음 단계로 넘어갈 수 없는 릴레이션들이 등장한다면 그 부분을 해소해야 BCNF를 만족 하기 때문에 릴레이션을 쪼개야 한다.

우선 logical 스키마에 있는 릴레이션이 BCNF를 모두 만족하는지 따져보기 이전에 먼저 1차 과제로 제출한 스키마들을 살펴본 결과 속성들 중 불필요한 속성들을 수정했어야 했다. 또한 2차

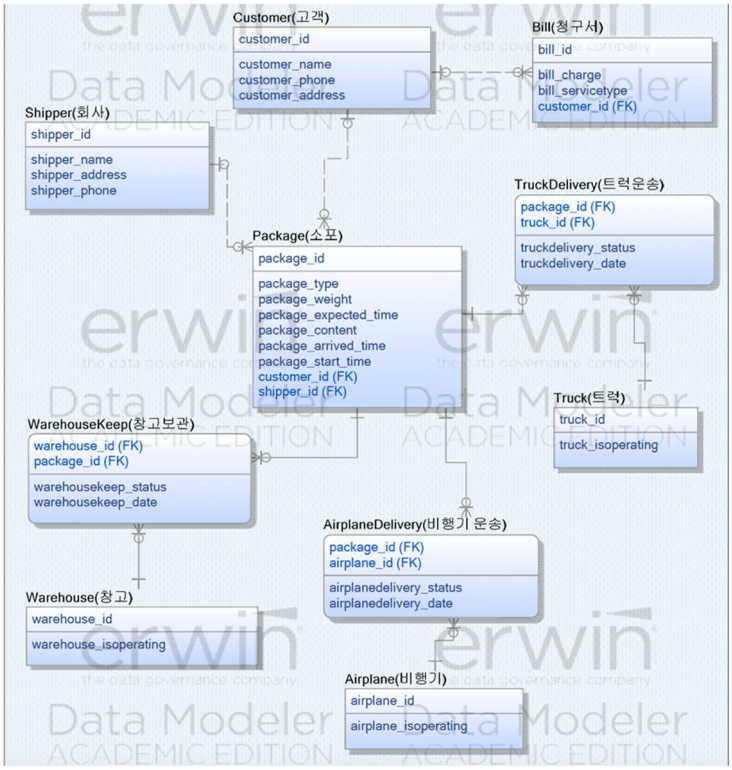
과제를 진행하면서 불필요한 릴레이션을 발견하여 이를 삭제하는 등의 수정 사항을 먼저 적용하였다. 이러한 변경, 수정 사항 등을 모두 적용한 뒤에 BCNF를 만족하는지 따져본 결과 모두 BCNF를 충족하고 있음을 확인하였다. 하여 우선 1차 과제에서 2차 과제로 이행함에 따라 어떤 부분이 수정, 삭제 되었는지 먼저 밝히고, 각 릴레이션이 BCNF를 만족하는지 확인해보겠다.

1차 과제와 달리 고객 릴레이션에 존재하였던 '1년간 납입금액' 속성과 '1년간 주문횟수' 속성은 삭제하였다. 왜냐하면 고객 릴레이션이 청구서 릴레이션과 연결되어 있고, 청구서의 청구 번호에 배송날짜와 고객id가 명시되어 있도록 속성을 만들었기 때문에 join연산을 통해 쉽게 해당 정보들을 뽑아낼 수 있다고 판단해서 릴레이션에서 앞서 말한 두 속성을 삭제하였다.

청구서 릴레이션의 경우 약간의 변화가 있었다. 위에서 말한 바와 같이 '청구번호' 속성을 int형 타입보다 저장 범위가 큰 long형 변수로 설정한 뒤, 이 값에 배송날짜+고객id+주문순서를 나열한 정수 값을 저장하도록 했다. 예를 들어 1번 고객 아이디를 갖고 있는 사람이 2023년 6월 10일에 물건 한 개를 주문했다면 202306100101이 청구번호가 되도록 설정했다. 그리고 청구서에서 구매 이력과 구매 방식을 삭제하였고 청구서 유형을 서비스 유형이라는 속성 이름으로 변화시켰다. '

트럭, 비행기, 창고 릴레이션의 경우 각각 기존에 트럭 보관 창고, 격납고 위치, 창고 위치를 두어서 각각의 주소를 표현하였는데, 생각해보니 이 DB에서 굳이 이 비행기는 원래 1번 격납고 소속이고, 이 트럭은 3번 트럭 창고 소속임을 표현할 필요는 없다고 판단하여 이 속성들을 삭제하였다. 그리고 트럭 운송, 비행기 운송 이 두 릴레이션에서는 현재 위치 속성을 삭제하였고, 운송 여부와 운송 날짜 속성을 추가하였다. 창고 보관 릴레이션에서는 보관 날짜만을 추가하였다.

마지막으로 1차 과제에 존재하였던 소포-배송기록 릴리이션의 경우 ER 다이어그램 당시 소포의 배송 기록을 표현하기 위한 중복 속성을 릴레이션 스키마로 바꾸는 과정에서 새롭게 만든 릴레이션이다. 왜냐하면 릴레이션 스키마에서의 릴레이션 내에서 중복 값 속성을 넣을 수 없기 때문에 새로운 릴레이션을 만들고 FK(외래키) 연결을 해서 데이터를 저장해야 한다. 그런데 2차 과제를 진행하면서 소포가 창고->비행기->트럭->목적지 이런 순서로 배송된다고 가정했는데, 이 정보들은 이미 창고 보관, 비행기 배송, 트럭 배송 릴레이션에 존재하기 때문에 불필요한 데이터의 중복이 있을 것 같아 소포-배송기록 릴레이션은 삭제하였다.



위 사진이 새롭게 정리한 릴레이션 스키마를 바탕으로 만든 logical 스키마이다.

이제부터 각 릴레이션에 대한 정보를 아래에 서술하고, 각 릴레이션이 BCNF를 만족하는지 검토하는 과정을 서술하겠다. 참고로 1차 과제에서는 릴레이션과 속성들을 모두 한글로 작성하였으나, 2차 과제 관련 수업 때 조교님께서 영어로 작성하는 것을 권하시는 듯 하여 2차 과제에서는 릴레이션과 속성을 모두 영어로 표현하였다.

Customer(customer_id, customer_name, customer_phone, customer_address)

Shipper(shipper_id, shipper_name, shipper_address, shipper_phone)

Package(package_id, package_type, package_weight, package_expected_time, pacakage_content, package_arrived_time, package_start_time, package_start_time, customer_id(FK), shipper_id(FK))

Bill(bill_id, bill_charge, bill_servicetype, customer_id(FK))

Warehouse(warehouse_id, warehouse_isoperating)

Airplane(airplane_id, airplane_isoperating)

Truck(truck_id, truck_isoperating)

WarehouseKeep(warehouse_id(FK), package_id(FK), warehousekeep_status, warehousekeep_date)

AirplaneDelivery(package_id(FK), airplane_id(FK), airplanedelivery_status, airplanedelivery_date)

TruckDelivery(package_id(FK), truck_id(FK), truckdelivery_status, truckdelivery_date)

먼저 앞서 말한 것처럼 각 릴레이션들은 모든 릴레이션 스키마의 기본 원칙인 '중복 값 속성을 허용하지 않는' 원칙을 지키고 있기에 모든 속성 값들은 단일 값을 만족한다. 따라서 여기 위에 있는 모든 릴레이션들은 1NF는 기본적으로 만족한다.

Customer 릴레이션은 customer_id가 단일 속성으로 기본키를 구성하고 있기에 부분함수 종속이 발생할 수 없고, 자연스럽게 2NF를 만족한다. 왜냐하면 부분함수종속(Partial Functional Dependency)이란 기본키가 복합 속성으로 구성 되어있는 복합키일 경우에 복합속성 중 일부 속성만으로도 어떤 속성을 결정지을 수 있는 함수 종속관계를 의미하기 때문에 기본키가 단일 속성으로 이루어져 있다면 해당 릴레이션에서 부분함수 종속은 절대 등장할 수 없다. 또한 기본키를 제외한 나머지 속성들 사이에서 어떠한 함수종속 관계를 발견할 수 없기 때문에 이행적 함수종속(Transitive Functional Dependency) 관계의 존재 가능성이 없다. 이행적 함수종속 관계가 나타나려면 기본키를 a라고 가정했을 때 "a->b, b->c 따라서 a->c"처럼 기존의 함수종속의 결과인 b이외의 또다른 c가 도출되어야 하기 때문에 기본키 이외의 또 다른 결정자(여기서 b의 역할을 담당할)가 등장해야 하는데, 기본키를 제외한 상황에서 어떠한 함수종속을 도출해 낼 수 없다면 애초에 이행적 함수종속 역시 존재할 가능성이 아예 없기 때문이다. 따라서 3NF도 자연스럽게 만족함을 알 수 있다. 마지막으로 이 릴레이션에 존재하는 결정자는 customer_id뿐이고 이 속성이 이 릴레이션의 유일한 후보키이기 때문에 당연히 슈퍼임은 전제되어 있는 것이고 따라서 Customer 릴레이션은 BCNF를 만족하고 있음을 알 수 있다. 이는 동시에 Customer 릴레이션의 경우 모든 FD(a->b)의 존재하는 모든 a가 기본키인 customer_id 외에 다른 속성으로 존재할 수 없음을 의미하기 때문에 BCNF를 만족하는 두 가지 조건을 모두 만족하는 것을 의미한다. 따라서 부분함수 종속이 없는지, 이행적 함수종속이 없는지, 모든 결정자가 슈퍼키인지 확인하는 세부 과정을 차례대로 거친다면 우리가 수업에서 배운 두 가지 충족 조건을 모두 만족할 수 밖에 없다는 것을 알 수 있다.

Shipper 릴레이션도 shipper_id가 단일 속성으로 기본키를 구성하고 있기 때문에 부분함수 종속이 발생할 가능성이 없으므로 자연스럽게 2NF를 만족한다. 또한 기본키가 아닌 나머지 속성들인 이름, 주소, 전화번호 사이에서 어떠한 함수종속 관계를 발견할 수 없기 때문에 이행적 함수종속의 존재 가능성 역시 없다. 이는 이 릴레이션의 모든 FD(a->B)에서의 결정자인 a가 shipper_id가 유일하다는 것을 의미한다. 이에 따라 3NF를 만족하고, 이 릴레이션 내에 존재하는 결정자는 shipper_id뿐이고 이 속성이 유일한 슈퍼키이면서 후보키이기 때문에 Shipper릴레이션 역시 BCNF

를 만족하고 있음을 알 수 있다.

Package 릴레이션도 package_id가 단일 속성으로 기본키를 구성하고 있기 때문에 부분함수 종속이 발생할 가능성이 없기에 2NF를 만족하고, 나머지 속성들 사이에서도 어떠한 함수종속 관계(FD)를 발견할 수 없기 때문에 이행적 함수종속의 가능성이 없다. 따라서 3NF를 만족하고, 이 릴레이션에 존재하는 결정자는 package_id 뿐이기 때문에 이 릴레이션 역시 BCNF를 만족하고 있다.

Bill 릴레이션도 bill_id가 단일 속성으로 기본키를 구성하고 있기 때문에 부분함수 종속이 발생할 가능성이 없어 2NF를 만족한다. 청구서(bill)안에 존재하는 청구번호(bill_id)안에 해당 청구에 대한 주문일자와 고객_id가 들어가 있기 때문에 bill_id만을 유일한 기본키로 설정하는 것이 가능하다. 그리고 나머지 속성들로 함수적 종속 관계를 도출해낼 수 없기 때문에 이 릴레이션에서 이행적 함수종속의 가능성이 없어 3NF를 만족하고, 이 릴레이션에서도 유일한 결정자가 후보키이자 기본키인 bill_id이기 때문에 BCNF를 만족한다.

Truck, Airplane, Warehouse 이 세 릴레이션 모두 기본키인 id 속성과 현재 동작 중인지 아니면 가동(운영)이 중단되었는지를 표현하는 Boolean 타입의 isoperating 속성만을 지니고 있다. Id값이 단일 속성으로 기본키를 구성하고 있기 때문에 부분함수 종속이 발생할 가능성이 없어 2NF를 만족한다. 그리고 id값 이외의 다른 속성이 결정자로서 존재할 수 없고, 유일한 결정자인 id가 유일한 후보키이자 기본키이기 때문에 3NF와 동시에 BCNF를 만족한다.

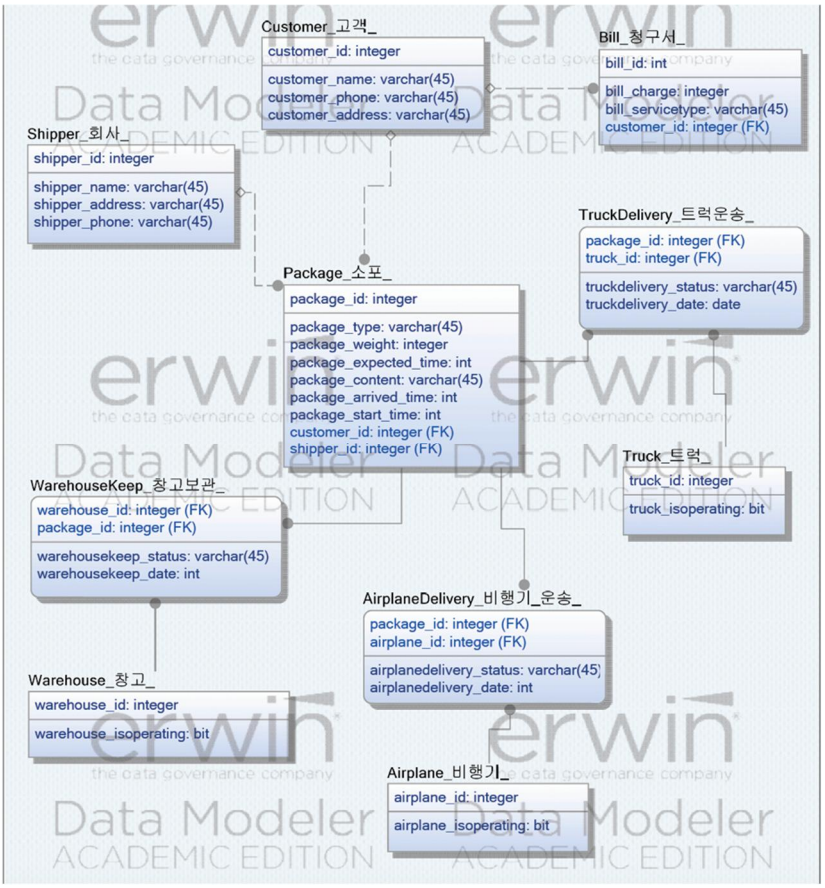
그리고 위의 세 릴레이션과 package 릴레이션을 연결하는 WarehouseKeep, AirplaneDelivery, TruckDelivery 릴레이션 역시 각각 구성이 동일한데, 모두 package_id와 Truck, Airplane, Warehouse중 하나의 PK값을 FK(외래키)로 받은 이 두 속성을 기본키로 구성하였다. 세 릴레이션 모두 기본키로 구성된 속성 중 어느 하나만으로 다른 속성을 결정할 수 없기 때문에 부분함수 종속이 발생하지 않아 2NF를 만족하고, 기본키를 제외한 속성들 사이에서 함수종속이 발생하지 않아 이행적 함수종속 역시 발생하지 않아 3NF를 만족하고, 기본키로 구성된 두 속성의 집합이 릴레이션 내에서 유일한 결정자이기 때문에 BCNF 역시 만족한다!

이로서 위의 제시된 logical 스키마 내에 있는 모든 릴레이션이 BCNF를 만족하고 있음을 알 수 있다.

2. Physical Schema Diagram

다음은 위에서 만든 logical Schema Diagram을 기준으로 하여 만든 Physical Schema Diagram에 대해서 설명하겠다. 이 부분에서는 각 릴레이션들의 속성 값들에 대해 자세히 설명하고 왜 그렇게 결정했는지에 대한 간략한 이유들을 서술하고자 한다.

아래에 있는 사진이 이번에 작성한 Physical Schema Diagram 이다.



먼저 Customer 릴레이션의 id는 정수형인 integer형으로 설정했다. 또한 이름 전화번호 주소 모두 varchar형으로 지정했으며 모두 여유롭게 공간을 충분히 할당했다. 그리고 Shipper 릴레이션의 경우 Customer 릴레이션과 동일하게 id는 interger형으로, 회사이름, 회사 주소, 회사 전화번호 속성은 모두 varchar형으로 지정하였다.

Bill 릴레이션의 경우 청구번호를 의미하는 속성인 bill_id를 long형 변수로 지정하였다. 앞서 짧

게 언급한 것처럼 청구 번호에 많은 정보를 담아 청구번호를 구성하면 더 편리할 것 같아 청구번호를 '해당 상품 주문날짜'+'주문고객의 id'+'동일한 날에 주문한 횟수정보'를 합쳐 12자리로 구성하였다. 따라서 다른 integer와 달리 범위가 더 큰 long형으로 설정하였다. 예를 들어 id가 2번인 고객이 2023년 6월 1일에 어떤 상품을 구매하면 이 상품에 대한 청구서의 청구번호는 202306010201이 되는 것이다. 그리고 해당 청구에 대한 청구 금액을 의미하는 bill_charge는 금액이기 때문에 integer형으로, bill_servicetype은 구매할 때 빠른 옵션이었는지 보통 옵션이었는지를 명시하기 위한 속성으로서 faster이나 normal이라는 값이 담길 것이기에 varchar로 설정하였다.

Package 릴레이션에서 id값은 integer로, 소포 유형은 크게 3가지로 나뉘는데 사이즈 별로 big middle small로 담길 것이기 때문에 varchar로 하였고, 소포 무게의 경우 정수인 integer로 선언하였다. 그리고 배송약속시간(expected_time)과 배송완료시간(arrived_time), 배송시작시간(start_time)은 모두 배송이 시작된 날짜를 기입하는 것인데, 20230601 이런 식으로 표현하기 위해 숫자형을 사용하였고, 그 중에서 long형을 사용하였다. Bill 릴레이션에서도 long형을 사용하였는데, erwin상에서 long형을 지정했음에도 Physical 스키마 다이어그램을 출력했을 때 int로 되어 나오는 모습이다. 그러나 실제 지정한 것은 long형이다. 그리고 내용물(content)속성은 글자를 넣어야 하기 때문에 varchar형으로 지정하였다.

Truck, Airplane, Warehouse 릴레이션 id는 integer형으로 설정하였고, isoperating 속성은 Boolean 타입으로 설정하였다. 예를 들어 현재 어떤 트럭이 잘 운행 중인지 또는 그렇지 않는지를 표현하기 위해 잘 운영 중이면 True(1), 그렇지 않으면 False(0) 값을 저장하기 위해 boolean타입으로 설정하였다. 사진에 나와있는 Erwin 출력에서 boolean으로 지정한 속성 값을 bit로 표현하고 있음을 알 수 있다. 그러나 실제로 erwin상에서 지정한 타입은 boolean이다.

TruckDelivery, AirplaneDelivery, WarehouseDelivery 릴레이션 모두 status 속성은 varchar로 설정하였다. 예를 들어 해당 소포가 해당 창고에 보관되었고 지금은 목적지로 배송이 완료되었다면 배송 후라는 의미의 'later'가 저장된다. 역시 마찬가지로 만약 어떤 소포가 해당 트럭이나 비행기에 저장되었고 지금은 목적지로 배송이 완료되었다면 동일하게 'later'라는 값이 저장된다. 그리고 지금 현재 트럭에 실려있거나 비행기에 실려 배송 중이라면 해당 값으로 'now'가 저장되도록 하였다. 따라서 varchar형으로 설정하였다. 그리고 세 릴레이션에는 모두 date 속성을 지정하였는데 이는 트럭과 비행기 창고에서 해당 상품을 품고 있었던 날짜를 저장하는 속성이다. 만약 어떤 소포가 2번 창고를 지나 3번 비행기를 실린 뒤 1번 트럭에 실릴 예정이라면 앞서 언급한 status 속성은 prior가 될 것이고, date값은 아직 실리지 않았기 때문에 NULL값이 되어야 할 것이다. 따라서 이 속성은 NULL을 허용하도록 하였다.

3. Queries

먼저 쿼리에 대한 설명에 앞서 과제 제출 파일인 c++파일과 mysql을 연결하는 방법을 조교님께서 저녁 수업을 통해 아주 자세히 알려주셔서 과제를 쉽게 끝낼 수 있었다. 그리고 지금부터 아래 c++예시 코드 각 쿼리 요청을 처리하는 함수들 중 1-1 커리를 처리한 예시를 바탕으로 어떻게 이 프로그램이 쿼리를 처리하고 정보를 정리해 출력해주는지 살펴보도록 하겠다.

먼저 아래 사진은 각 쿼리를 처리하는 함수들 중 1-1의 코드 사진이다.

```
void query1_1(MYSQL* connection, int truck) {

    char update[10000];
    printf("------- TYPE 1-1 -------₩n");
    printf("₩n");
    printf("Find all customers who had a package on the truck at the time of the crash.₩n");

    sprintf(update, "SELECT customer.customer_name FROM truck_delivery INNER JOIN package ON truck_delivery.package_id = pa
    mysql_query(connection, update); // 쿼리 보내는 함수

    sql_result = mysql_store_result(connection); // 쿼리 결과를 여기에 저장
    printf("<customer_name>₩n₩n");
    while ((sql_row = mysql_fetch_row(sql_result)) != NULL) {
        printf("%s ₩n", sql_row[0]);
    }
    printf("₩n₩n₩n");

}
```

함수 query1_1은 매개변수로 MYSQL connection 포인터 변수와 truck에 대한 정보를 받는다. 뒤에 1번 쿼리에 대한 처리 예시가 나오겠지만, Type1번에서는 '어떤 트럭이 사고가 났을 때'를 가정하고 있기 때문에 사용자로부터 x 값을 scanf 함수로 입력 받는다. 이 입력 받은 값은 트럭의 id 값이기에 이 함수는 사고 난 트럭의 id 값을 매개변수 truck을 통해 전달받는다. 그리고 sprint() 함수를 통해 내가 SQL에 보낼 쿼리를 정리하고 그 쿼리 문장은 문자열 배열 update에 담긴다. 그리고 mysql_query()함수를 통해 해당 쿼리를 보내고, 그 결과를 mysql_store_result()함수로 받아와 sql_result로 받아 온 뒤 이 결과(테이블)의 튜플들을 한 줄 씩 while문으로 처리하는 것이다. Query1_1말고 다른 함수들의 실행 메커니즘은 동일하다. 미리 정해진 select문을 해당 함수가 실행될 때마다 요청하여 결과를 받아와 튜플 하나씩 출력해주는 것이다. 그럼 이제 각 7개의 쿼리들이 잘 동작하는지 확인해보도록 하겠다.

< Type 1-1>

```
**************************************************
*********** JTH Package Delivery DataBase ***********
------- SELECT QUERY TYPES -------

        1. TYPE 1
        2. TYPE 2
        3. TYPE 3
        4. TYPE 4
        5. TYPE 5
        0. QUIT
1
**  Assume truck X is destroyed in a crash. **
Which truck X? : 3

----- Subtypes in TYPE 1 -----
    1. TYPE 1-1.
    2. TYPE 1-2.
    3. TYPE 1-3.
1
------- TYPE 1-1 -------

Find all customers who had a package on the truck at the time of the crash.
<customer_name>

Ronald Reagan
Theodore Roosevelt
```

Type 1-1은 사고가 났다고 가정하는 트럭의 인덱스를 입력하면 해당 트럭이 사고가 났을 때 배송 중이던 상품의 고객 이름을 출력해주는 쿼리이다. 출력 예시로 미루어 볼 때 3번 트럭이 사고가 났을 때 해당 트럭에 있는 소포의 customer의 이름은 로널드 레이건과 시어도어 루즈벨트라는 것을 알 수 있다.

< Type 1-2>

```
------- SELECT QUERY TYPES -------

        1. TYPE 1
        2. TYPE 2
        3. TYPE 3
        4. TYPE 4
        5. TYPE 5
        0. QUIT
1
**  Assume truck X is destroyed in a crash. **
Which truck X? : 2

----- Subtypes in TYPE 1 -----
    1. TYPE 1-1.
    2. TYPE 1-2.
    3. TYPE 1-3.
2
------- TYPE 1-2 -------

Find all recipients who had a package on that truck at the time of the crash.
<customer_name>

Franklin D. Roosevelt
John F. Kennedy
```

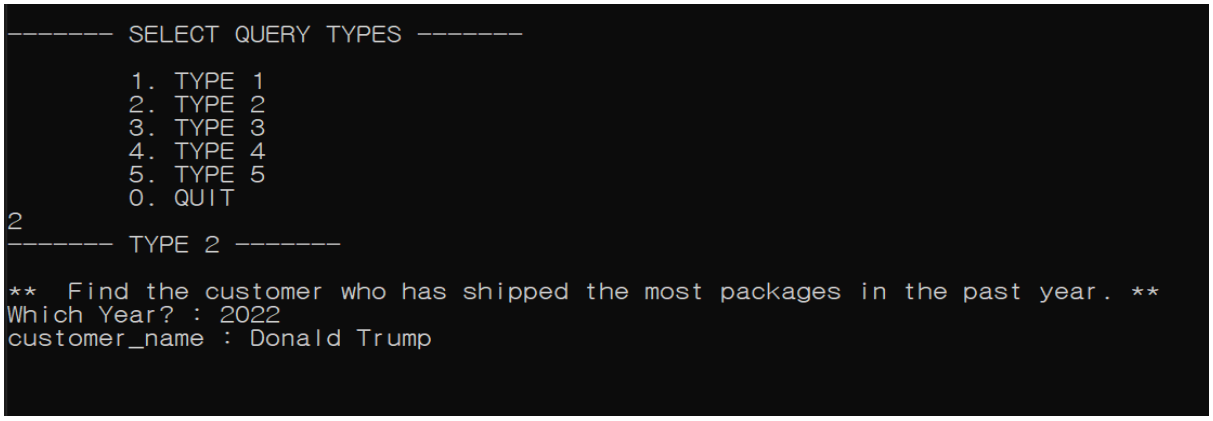
Type 1-2은 바로 이전 쿼리인 **Type 1-1**와 유사한데, 이것 역시도 사고가 났다고 가정하는 트럭의 인덱스를 입력해야 하고, 이번에는 이전과 달리 해당 트럭이 배송 중이던 소포의 고객이 아닌 실제로 이 소포를 수령하는 수령자를 찾아주는 쿼리이다. 따라서 결과를 보면 알 수 있듯이 2번 트럭이 사고가 났을 때 해당 트럭에 있는 소포의 recipient의 이름은 프랭클린 루즈벨트와 존 F 케네디라는 것을 알 수 있다.

< Type 1-3>

```
------- SELECT QUERY TYPES -------

        1. TYPE 1
        2. TYPE 2
        3. TYPE 3
        4. TYPE 4
        5. TYPE 5
        0. QUIT
1
**  Assume truck X is destroyed in a crash. **
Which truck X? : 1

----- Subtypes in TYPE 1 -----
    1. TYPE 1-1.
    2. TYPE 1-2.
    3. TYPE 1-3.
3
------- TYPE 1-3 -------

Find the last successful delivery by that truck prior to the crash.
<truck_delivery_date>      <package_content>       <customer_name>

     20230507                smartphone              George Washington
```

Type 1-3도 사고가 났다고 가정하는 트럭의 id를 입력해야 한다. 이 id를 입력하고 나면 해당 트럭이 사고가 나기 이전 가장 최근에 성공적으로 배송을 마친 소포의 관한 정보를 알 수 있다. 예시에서는 1번 트럭을 입력했다. 출력되는 결과를 볼 때, 1번 트럭이 사고가 나기 이전 가장 최근에 배송에 성공한 배송은 2023년 05월 07일에 스마트폰을 조지 워싱턴에게 배송한 것이었다는 것을 알 수 있다.

< Type 2>



Type 2는 연도를 입력하면 해당 연도에서 가장 많은 소포를 배송 받은 customer를 출력해주는 쿼리이다. 결과 예시를 볼 때 2022년에 가장 많은 소포를 배송 받은 사람은 도널드 트럼프임을 알 수 있다.

< Type 3>

```
------- SELECT QUERY TYPES -------

        1. TYPE 1
        2. TYPE 2
        3. TYPE 3
        4. TYPE 4
        5. TYPE 5
        0. QUIT
3
------- TYPE 3 -------

**  Find the customer who has spent the most money on shipping in the past year. **
Which Year? : 2021
customer_name : John Adams
```

Type 3는 연도를 입력하면 해당 연도에서 소포를 주문하는데 가장 큰 금액을 쓴 customer를 찾아주는 쿼리이다. 예시를 보면 알 수 있듯이, 2021년에 가장 많은 물건을 주문한 사람은 존 아담스임을 알 수 있다.

< Type 4>

```
------- SELECT QUERY TYPES -------

        1. TYPE 1
        2. TYPE 2
        3. TYPE 3
        4. TYPE 4
        5. TYPE 5
        0. QUIT
4
------- TYPE 4 -------

**  Find the packages that were not delivered within the promised time. **
<customer_name>         <package_content>       <package_expected_time>         <package_arrived_time>

Abraham Lincoln                 TV                      20230412                        20230413
Donald Trump                    car                     20220509                        20220510
Donald Trump                    smartphone              20220520                        20220521
Franklin D. Roosevelt           car                     20230303                        20230305
Ronald Reagan                   TV                      20230507                        20230508
Dwight D. Eisenhower            TV                      20230520                        20230521
Harry S. Truman                 car                     20220508                        20220509
Andrew Jackson                  smartphone              20220207                        20220210
Woodrow Wilson                  TV                      20230405                        20230408
Lyndon B. Johnson               smartphone              20220518                        20220519
Thomas Jefferson                smartphone              20210509                        20210510
```

Type 4는 이 물류 회사 데이터베이스에서 배송을 약속한 날짜(package_expected_time)보다 실제 배송을 받은 시간(package_arrived_time)이 늦은 배송 사례들을 출력해주는 쿼리이다. 이 사례를 통해 총 11번의 배송에서 배송 약속 시간보다 늦게 배송이 이루어진 것을 알 수 있다. 화면에 출

력되는 정보는 왼쪽부터 해당 배송의 customer정보와 그 배송이 이루어질 때 배송되는 상품의 종류가 나오고, 배송 시 약속한 시간과 실제 배송이 이루어지는 시간이 8자를 통해 나타나고 있다. 한 예시를 보면 tv를 주문한 에이브리험 링컨의 경우 2023년 4월 12일에 배송을 받기로 약속을 받았었는데, 실제 배송이 이루어진 것은 하루 늦은 13일이었다.

< Type 5>

```
------- SELECT QUERY TYPES -------

        1. TYPE 1
        2. TYPE 2
        3. TYPE 3
        4. TYPE 4
        5. TYPE 5
        0. QUIT
5
------- TYPE 5 -------

**  Generate the bill for each customer for the past month. Consider creating several types of bills. **
Which Year and Month? : 2021 5
************** BILL ***************

Customer Name: John Adams
Customer Address: Massachusetts
Charge: $30000
Bill_service_type: fast

          Thank you
**********************************
```

```
************** BILL ***************

Customer Name: George H. W. Bush
Customer Address: Texas
Charge: $1100
Bill_service_type: fast

          Thank you
**********************************

************** BILL ***************

Customer Name: Gerald Ford
Customer Address: Michigan
Charge: $30000
Bill_service_type: nomal

          Thank you
**********************************
```

```
************** BILL ***************

Customer Name: Gerald Ford
Customer Address: Michigan
Charge: $30000
Bill_service_type: nomal

           Thank you
**********************************

************** BILL ***************

Customer Name: Thomas Jefferson
Customer Address: Virginia
Charge: $1100
Bill_service_type: fast

           Thank you
**********************************

************** BILL ***************

Customer Name: William J. Clinton
Customer Address: Arkansas
Charge: $30000
Bill_service_type: nomal

           Thank you
**********************************
```

Type 5는 사용자로부터 검색하고자 하는 해당 연도와 월을 입력 받으면, 해당 연도와 그 달(Month)에 상품을 주문하여 이에 대한 청구서를 받은 사람들의 청구서를 출력해주는 쿼리이다. 예시를 통해 알 수 있듯이 2021년 5월에는 존 아담스, 조지 h.w 부시, 제럴드 포드, 토마스 제퍼슨 등이 상품 주문에 대한 비용 청구서를 받았음을 알 수 있다. 청구서에는 해당 상품 내역과 상품 가격 그리고 청구서를 받는 customer의 이름과 주소 등이 간략하게 적혀 있다.

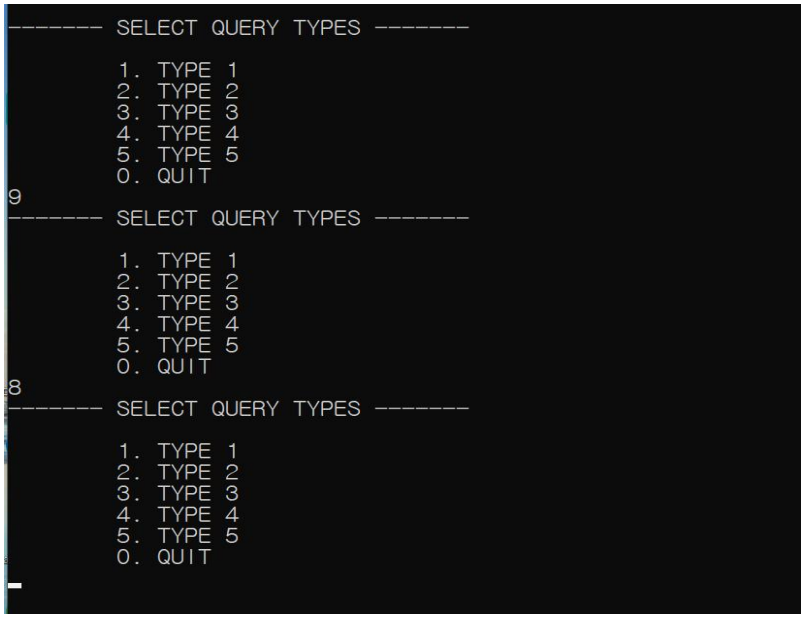
<0 입력 시 처리>

이 프로그램에서 메인 화면에서 0을 입력하면 프로그램이 종료된다. 이 부분도 기능에 포함시켰기 때문에 잘 작동하는지 확인해보겠다.

```
------- SELECT QUERY TYPES -------

        1. TYPE 1
        2. TYPE 2
        3. TYPE 3
        4. TYPE 4
        5. TYPE 5
        0. QUIT
0

C:₩Users₩nex_d₩Desktop₩temp₩x64₩Debug₩temp.exe(프로세스 3400개)이(가) 종료되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...
```

예시에서 볼 수 있는 것처럼 처음 타입을 선택하는 메인 화면에서 0을 입력하면 그대로 프로그램이 종료됨을 알 수 있다.

<예외 처리>



메인 메뉴에서 0~5를 벗어나는 숫자를 입력했을 때 자동으로 다시 메인 메뉴로 돌아올 수 있도록 예외 처리한 부분을 확인하였다.

4. 소감

이번 과제는 개인적으로 실력 성장에 있어 매우 큰 도움이 되었다. 작은 규모의 가상 세계를 구축하고 그 안에서 쿼리를 통해 정보를 가져오고 하는 등의 재미있는 게임 같았지만, 이 것을 만들어보면서 실제 수업시간에 배운 것들을 연습하는 방법과, C언어와 같은 프로그래밍 언어랑 SQL이랑 연결하는 방법 등을 직접 적용할 수 있어 많은 도움이 되었다. 그리고 중간고사 이전에 배웠던 SQL질의도 이번 기회를 통해 더 많이 늘 수 있었다. 사실 수업 때나 시험을 위해서 공부를

할 때는 그냥 질의하는 방법도 그냥 외우고 문제 풀 수 있을 정도로만 하고 만족 했었는데, 실제로 원하는 정보를 뽑아 내기 위해 머리를 싸매고 고민하면서 SQL 질의가 정말 재미있다는 것을 깨닫는 계기가 되었다. 그리고 올해부터 웹개발 학회에서 백엔드를 담당하기 위해 웹개발 프레임워크를 한창 배우고 있는데, 이번 학기 동안 배운 DB 내용이 앞으로 백엔드 개발자가 되는 과정에 큰 도움이 될 것 같고, 데이터베이스라는 수업을 정말 어렵게 공부하고 있고, 이 과제 역시 쉽지 않았지만 많은 성장을 느낄 수 있는 좋은 계기가 되었던 것 같아 기쁘다. 확실히 과제를 하니 교실에서 배웠던 내용이 더욱 친근하게 느껴지는 것 같다. 단순히 지식을 쌓는게 아니라 그 지식을 바탕으로 무엇인가 하나라도 만들어 낼 수 있다는 것에 뿌듯했고, 진짜 살아있는 지식을 배우는 것 같았다.