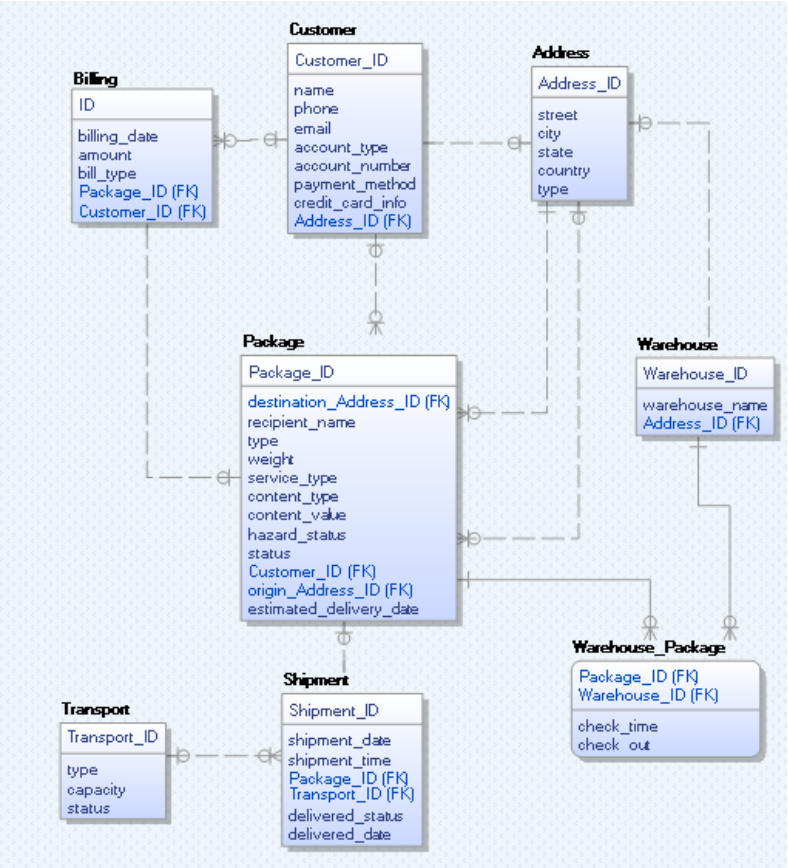
Package Delivery System

컴퓨터공학과 20191619 이동석

1. 개요

본 레포트에서는 project1에서 설계한 관계 스키마를 바탕으로 BCNF를 만족하는 논리 스키마를 작성한다. 이후, Erwin을 사용해 물리 스키마를 작성하고, 스키마의 data types, domain, constraints, relationship type, nullable 등을 확정한다. 다음으로, MySQL과 ODBC를 활용해 Visual Studio 2022에서 최종 데이터 베이스 모델을 구현한다.

2. BCNF 체크



위 그림은 project1에서 사용한 관계 스키마이다. Billing에서 실수로 중복입력된 Package_ID, Customer_ID를 삭제하였다. 각 엔티티 별로 FD(Functional Dependency)를 확인하고, BCNF를 만족하는지 확인한다.

2-1. Customer

우선 Customer의 각 속성에 대해 간략하게 알아보면, 각 사용자의 고유 ID인 Customer_ID, 그리고 사용자의 name(중복가능), phone, email(고유), account_type(정기/비정기), account_number(계좌번호), payment_method(결제 방법: 현금/신용카드/체크카드/카카오페이 등), credit_card_info(신용카드 정보), 그리고 Address_ID(FK, 주소 ID) 로 이루어진다. 모든 속성은 반드시 null이 될 수 없고, 정보가 존재해야한다. 이를 고려해서 Customer의 FD를 구해보면 다음과 같다.

1. Customer_ID -> name, phone, email, account_type, account_number, payment_method, credit_card_info, Address_ID
2. credit_card_info -> account_type, payment_method
3. account_number -> account_type, payment_method
4. phone -> email, name (eamil -> phone, name)

우선, 1번 Customer_ID는 기본키의 역할을 하므로 슈퍼키이다. 따라서 BCNF의 조건을 만족한다. 다음으로, 2번 credit_card_info는 사람마다 유니크한 카드정보를 가지고 있으므로 슈퍼키라고 생각할 수 있으며 따라서 BCNF 조건을 만족한다. 3번의 경우 사람마다 계좌번호가 다르므로 역시 슈퍼키이며, 조건을 만족한다. 4번의 경우 phone은 한 사람이 여러 개의 핸드폰 번호를 가질 수는 있지만, 조건적으로 반드시 하나의 본인명의의 핸드폰을 등록시키도록 만들면 슈퍼키로 볼 수 있다. 따라서, BCNF를 만족하고 Customer의 경우 BCNF를 만족한다고 볼 수 있다.

2-2. Package

Package의 각 속성은 각 화물의 고유 ID인 Package_ID, 수령자의 이름인 recipient_name, type(화물의 유형: 일반, 고가, 액체 등), weight(무게), service_type(국

제, 국내, 당일 등), content_type(가전, 서류 등), content_value(상품 가치), hazard_status(위험 상태), estimated_delivery_date(예상 배송도착 시간), status(배송 상태: 준비, 이동, 사고 등), destination/origin_Address_ID(출발/도착 주소 ID), Customer_ID(고객 ID) 로 이루어진다. 마찬가지로 FD를 찾아보면 다음과 같다.

1. Package_ID 는 기본키로서 나머지 속성에 대해 FD를 가진다.
2. content_type -> content_value, hazard_status, type, weight
3. origin/destination_Address_ID -> service_type, estimate_delivery_date

우선 1번은 앞서 설명한 것과 동일한 이유이다. 다음으로 2번은 택배의 내용물이 무엇인지에 따라 가치가 정해진다. 또한, 내용의 위험성에 따라 hazard_status가 결정되고 택배회사는 위험물품에 대해 배송을 거절할 수 있다. 또한, 물건에 따라 일반/액체/고가 등 유형을 화물 유형을 지정할 수 있으며 무게가 정해진다. 3번의 경우에는 출발-도착 지역에 따라 국제와 국내로 구분이 가능하며 당일배송의 가능유무도 알 수 있을 것이다. 또한, 해당 위치를 기반으로 예상 도착 시간도 알 수 있다. 결론적으로 2번과 3번의 FD가 생기게 된다.

그러나, BCNF를 만족하기 위해서는 trivial 하거나, content_type과 origin/destination _Address_ID 가 슈퍼키여야 한다. 그러나, 슈퍼키가 아니므로 BCNF를 만족하지 않는다. 따라서, 2번 FD에 대해 분해를 먼저 진행하면 다음과 같다.
a U b = { **content_type**, content_value, hazard_status, type, weight }

R – (b – a) = { **Package_ID**, recipient_name, status, Customer_ID, origin/destination_Address_ID, service_type, estimated_delivery_date }

다음으로 3번 FD에 대해 분해를 진행하면 다음과 같다.

a U b = { **origin/destination_Address_ID**, service_type, estimated_delivery_date }

R – (b – a) = { **Package_ID**, recipient_name, status, Customer_ID, content_type, origin/destination_Address_ID }

2-3 Billing

Billing의 속성은 마찬가지로 고유 ID로 PK를 설정했지만, 생각해보면 Customer_ID와 Pakcage_ID의 조합으로 고유하게 Bill을 식별할 수 있다. 따라서 PK로 Customer_ID와 Package_ID를 사용하도록 바꿨다. 다음으로, billing_date는 결제날짜를 말하며, amount는 결제 금액, 그리고 bill_type은 고객으로부터 입력받아야 한다고 판단하여 삭제하였다. 이 경우 FD를 살펴보면 기본키에 의한 FD만을 가지는 것을 확인할 수 있고 따라서 BCNF의 조건을 만족한다고 볼 수 있다.

2-4. Address

Address는 고유 ID와 street, city, state, country 그리고 type(창고, 거주지, 회사 등)의 속성으로 이루어져있다. Bill과 마찬가지로 FD는 기본키에 의한 FD만을 가지는 것을 알 수 있다. 따라서 BCNF를 만족한다.

2-5. Warehouse

Warehouse도 창고의 고유 ID를 나타내기 위한 ID와 이름, 그리고 주소 ID로 표현된다. 마찬가지로 FD는 기본키에 의한 FD만을 가지는 것을 알 수 있다. 따라서 BCNF를 만족한다.

2-6. Warehouse_Package

Package_ID와 Warehouse_ID 로 복합 기본키를 가진다. 속성으로는 창고의 입/출 시간을 알 수 있다. 마찬가지로 FD는 기본키에 의한 FD만을 가지는 것을 알 수 있다. 따라서 BCNF를 만족한다.

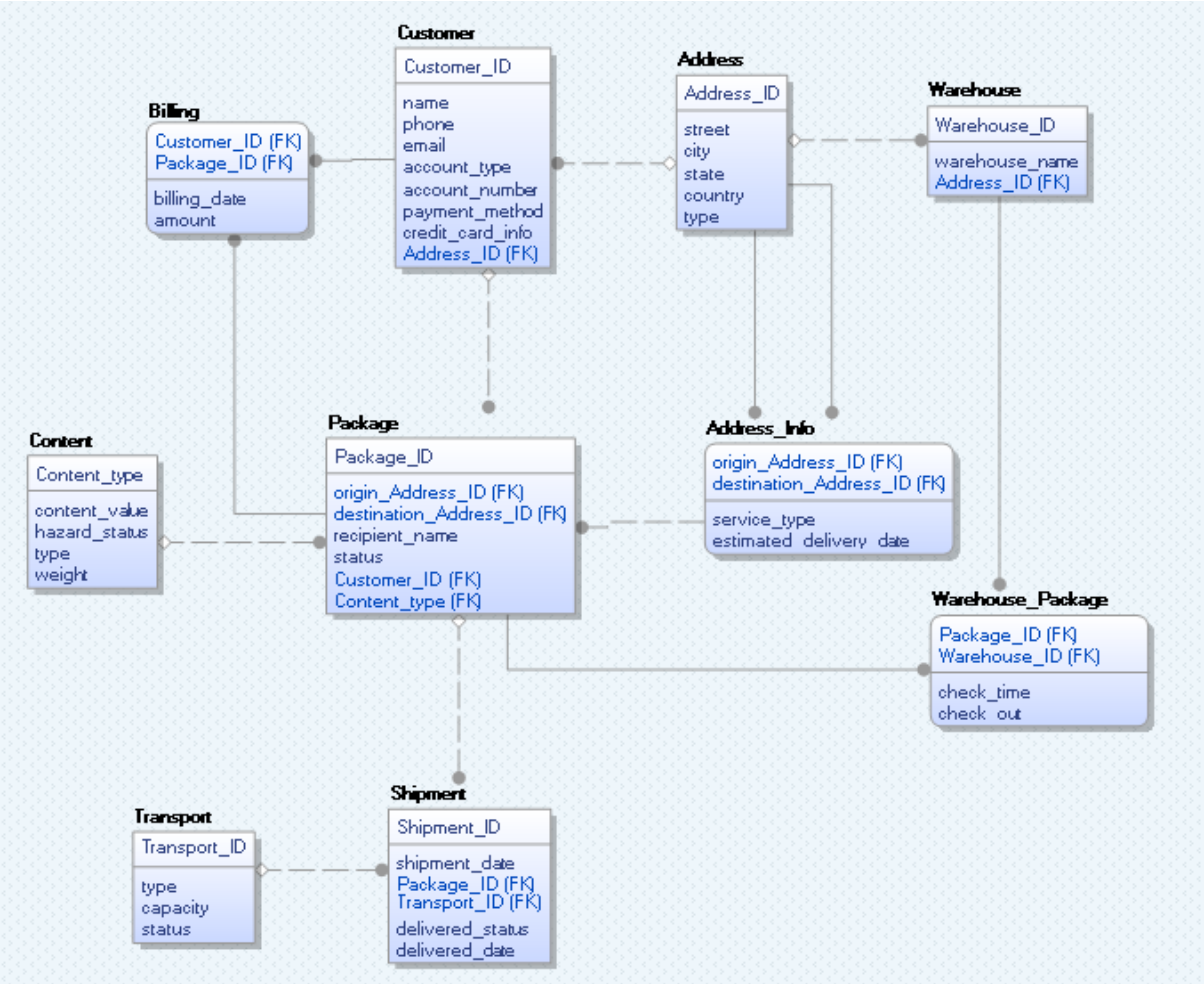
2-7 Shipment

소포의 정보를 나타내기 위해 Shipment_ID로 고유 식별자를 가진다. 또한, shipment_date(발송 날짜, 기존에 있던 shipment_time 삭제), delivered_status(배송 상태), delivered_date(배송 날짜) 를 가지며, 각 Shipment에 포함된 Pacakge_ID를 외래키로 가진다. 또한, 운송수단의 정보를 알기 위해 Transport_ID를 외래키로 가진다. 기본키외에 다른 FD를 찾아볼 수 없으므로 따라서 BCNF를 만족한다.

2-8. Transport

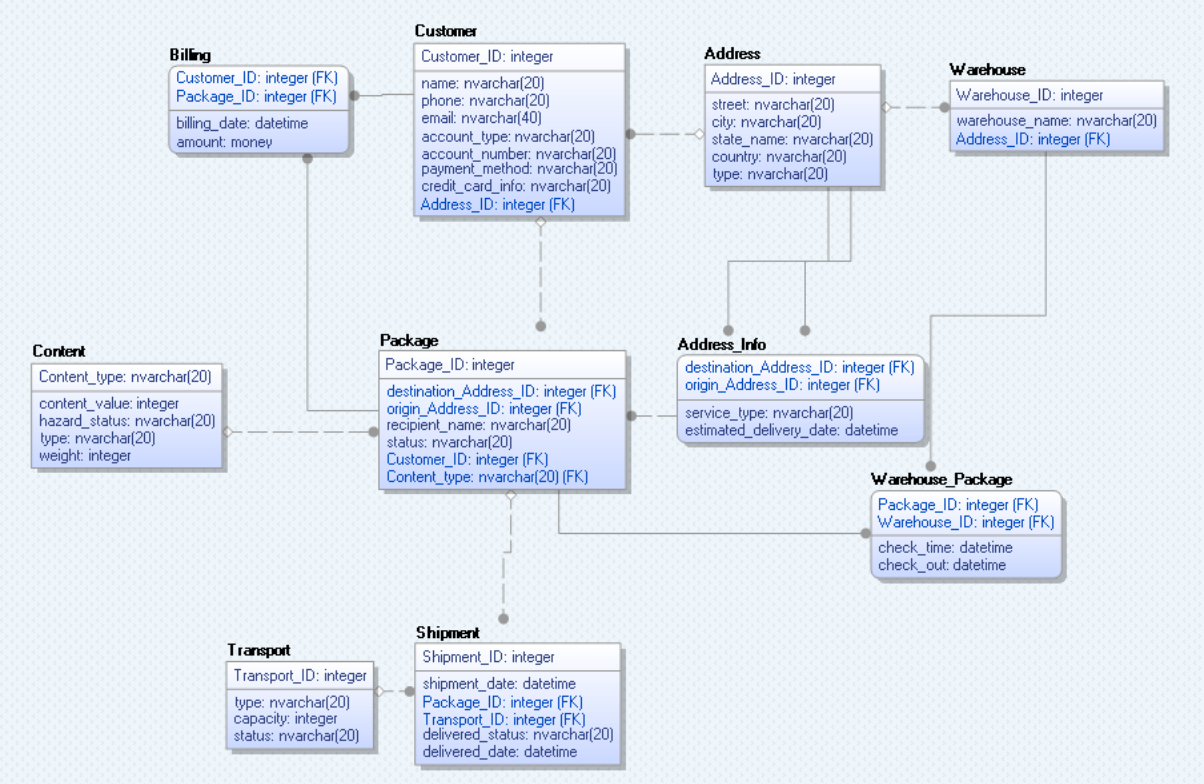
운송 수단의 ID(트럭번호와 같은 고유 ID)를 가지며, type(비행기, 배, 차 등), capcity(용량), status(사고, 운행 중, 정비 등)의 정보를 가지며 마찬가지로 FD는 기본키에 의한 FD만을 가지는 것을 알 수 있다. 따라서 BCNF를 만족한다.

3. Logical Schema



앞선 분해를 반영한, 모든 엔티티가 BCNF를 만족하는 Logical Schema는 위 그림과 같다.

4. Physical Schema



우선, 각 테이블의 고유 ID는 모두 Integer로 지정했으며 대부분의 속성은 필수라고 생각하여 not null이다. null이 가능한 경우만 언급하도록 하겠다. 또한, deafult 데이터 타입을 nvarchar(20)으로 지정했다. 대부분의 경우 해당 데이터타입을 가진다.

4-1 Customer

고객은 기본적으로 주소정보를 가지며, Address 테이블을 참조하고 Address_ID를 통해 관리된다. 또한 가족구성원인 경우에는 같은 주소를 가지는 서로 다른 고객이 존재할 수 있으므로, many-to-one으로 관계를 맺도록 하였다.

- Customer_ID : ID는 해당 테이블의 기본키 역할로, 고유번호를 나타낸다. 따라서, Integer로 지정했다. 1~N 의 번호가 할당될수도 있고, 학번과 유사하게 20230000~ 형식으로 들어오는 방법 등이 있을 수 있다.
- name : 이름은 문자열 형식으로 들어가기 때문에 nvarchar(20)으로 지정했다.
- phone : 010-xxxx-xxxx 형식으로 입력받으며 nvarchar(20)으로 지정했다. 고객은 반드시 하나의 핸드폰을 인증해야한다고 가정했다.

- email : example@example.com 형식으로 받으며 nvarchar(40)으로 지정했다. 고객은 정보수신 등을 위해 반드시 하나의 이메일을 등록하고 인증해야하며, 2개 이상을 등록하지 못한다고 가정했다.
- account_type : 정기(계약) / 비정기 고객으로 나뉘며 nvarchar(20)으로 지정했다.
- account_number : 고객의 계좌번호(환불 등)는 (기업)97403216801019 와 같은 형식으로 들어간다. 본인명의의 계좌번호로 가정했으며 nvarchar(20)으로 지정했다.
- payment_method : 결제 방법은 현금/카드/카카오페이 등의 방법이 있으며, 마찬가지로 nvarchar(20)으로 지정했다. default는 카드로 가정한다.
- credit_card_info : 사용자는 반드시 하나의 본인명의 카드 정보를 등록해야한다. 가족구성원의 카드를 사용하거나, 다른 사람의 카드정보를 등록하지 못하도록 가정했다. 카드 정보역시 xxxx-xxxx-xxxx-xxxx 형식이므로 nvarchar(20)으로 지정했다.

4-2 Package

Package는 어떤 고객의 물품인지 알기 위해 Customer_ID를 외래키로 참조한다. 또한, 하나의 고객은 여러 개의 Package를 주문할 수 있으므로 one-to-many의 관계를 갖는다. 또한, 출발/도착 장소를 저장하기 위해서 Address_Info의 ID를 외래키로 참조한다. 마찬가지로, 출발/도착 장소가 동일할 수 있으므로 many-to-one의 관계를 가진다. 또한, Package의 내용에 대한 정보를 Content_type을 참조하여 알 수 있다. 여러 패키지의 Content_type 이 동일할 수 있으므로 many-to-one의 관계를 갖는다.

- recipinet_name: 이름은 문자열 형식으로 들어가기 때문에 nvarchar(20)으로 지정했다.
- status : Package 단위의 배송상태 정보를 저장하기 위한 속성으로 준비,이동,사고 등의 정보를 저장하기 위해 nvarchar(20)으로 지정했다.

4-3 Billing

Billing은 Package와 Customer를 참조하여 복합 기본키를 가진다. 따라서, Package와 Customer에 대해서 Identifying 하며, 각 테이블에 many-to-one 관계를 가진다.

- billing_date : 날짜 및 시간을 나타내기 위해서 datetime으로 타입을 지정했다.

- amount : 결제금액을 나타내기 위해서 데이터 유형으로 money를 지정했지만, 지원하지 않는 유형일 수 있으므로 double로 지정하는 것도 좋을 것 같다.
- bill_type : 고객이 요청하는 것으로 생각하여 해당 속성을 삭제하였다.

4-4 Content

기존 Package를 BCNF를 만족시키도록 정규화 시켜 분해해서 나온 테이블이다. Package와 one-to-many 관계를 가진다.

- Cotnet_type : 가전, 서류, 음료 등과 같이 내용물을 나타내며 기본키로서 사용된다. nvarchar(20)으로 지정했다. 조금 더 세밀하게 LG그램, LG티비 등으로 나눌 수 있지만 이번 프로젝트에서는 간단한 타입만을 지정한다.
- content_value : 앞서 말했듯, 내용물이 무엇인지에 따라 가치가 정해진다. 가치는 integer로 정수형태로 정보를 저장한다.
- hazard_status : 내용의 위험성에 따라 결정되고 택배회사는 위험물품에 대해 배송을 거절할 수 있다. nvarchar(20)으로 지정했다.
- type : 물건에 따라 일반/액체/고가 등 유형을 화물 유형을 지정할 수 있다. nvarchar(20)으로 지정했다.
- weight : 물건에 따라 무게가 정해진다. 따라서 Integer로 지정했다.

4-5 Address

각 주소정보에 따라 고유 Address_ID를 가진다. 각 속성은 속성의 이름에서 알 수 있듯이 street, city, state_name, country, 그리고 창고, 거주지, 회사 등의 정보를 저장하는 type 속성이 있다. nvarchar(20)으로 지정했다.

4-6 Address_Info

해당 테이블 역시 Content테이블과 마찬가지로 기존 Package를 BCNF를 만족시키도록 정규화 시켜 분해해서 나온 테이블이다. Address 테이블을 참조하여 출발/도착지를 복합 기본키로 가진다. 따라서, Address와 identifying 관계를 가지며 many-to-one이다.

- service_type : 국제, 국내(당일옵션) 등의 서비스를 출발/도착 주소로 제공 가능한 서비스 유형을 사용자에게 제공하고 저장한다. 마찬가지로 nvarchar(20)으로 지정했다.
- estimated_delivery_date : 위치를 기반으로 대략적인(혹은 저장된 데이터를 베이스로) 예상 도차시간을 제공할 수 있다.

4-7 Warehouse

창고의 위치를 알기 위해서 Address 테이블을 참조한다. 또한 각 창고는 고유 ID를 가진다. 창고의 이름은 앞선 이름들과 마찬가지로 nvarchar(20)으로 지정했다.

4-8 Warehouse_Package

Pakcage와 Warehosue의 ID를 복합 기본키로 가진다. 따라서, Package와 Warehouse 사이에 identifying 하며, many-to-one의 관계를 가진다.

- check_time : 창고에 Package가 들어온적이 있다면 반드시 null이 될 수 없으며 날짜 및 시간을 나타내기 위해 datetime으로 지정했다.
- check_out : null이 될 수 있는 속성이다. Package가 창고에 들어오고 다시 나가기 전까지는 null 값을 가진다. 마찬가지로 datetime으로 지정했다.

4-9 Shipment

고유 Shipment_ID를 가진다. Shipment는 여러 Package_ID를 속성으로 가지며 때문에 Package와 many-to-one의 관계를 가진다.

- shipmnet_date : 마찬가지로 선적 날짜 및 시간 정보저장을 위해 datetime으로 지정했다.
- delivered_status : Shipment의 배송 상태를 저장하기 위함이며 nvarchar(20)으로 설정했다.
- delivered_date : 도착 날짜를 저장하기 위해 datetime으로 지정했다. 도착 전까지는 null이 가능하다.

4-10 Transport

운송수단의 고유 ID (예로 트럭번호 1721 등)을 나타낸다.

- type : 배, 트럭, 비행기 등의 타입을 나타내는 속성이며 nvarchar(20)으로 지정했다.
- capacity : 운송수단의 적재용량을 의미하며 integer로 나타냈다.
- status : 현재 운송수단의 상태를 저장하기 위한 속성이다. 사고, 정비, 이동, 사용 등을 의미하며, nvarchar(20)으로 지정했다.

5. 데이터 생성

우선 physical schema를 기준으로 사용할 데이터를 직접 생성해보면 다음과 같다. 데이터를 생성할 때 최대한, 논리적으로 무결성을 지키면서 생성하려 노력했다. 주요 데이터 Customer, Address, Package만 소개한다.

5-1 Customer

우선 11명의 고객 데이터를 insert into Customer values 구문으로 데이터를 넣는다. ID는 1~11로 고유한 값을 주고 나머지 값은 지인의 이름이나 번호로 임의로 할당했다. 또한, Address_ID를 참조하는데, 이때 Address 테이블에서 type이 warehouse를 제외하고 임의의 주소 ID를 지정했다.

ex) insert into Customer values (1, "Lee DongSeok", "010-6257-9694", "ryan1766@naver.com", "regular", "974-032168-01-019", "account", "9440-1234-4444-5555", 1);

5-2 Address

다음으로 주소이다. 주소의 경우에도 역시 임의로 설정했으며 1~11의 고유값을 가진다.

ex) insert into Address values (11, "55th street", "Newyork", "New Jersey","america",

"company");

5-3 Package

이제, 각 고객들이 어떤 Package를 주문했다고 가정하고 Package 데이터를 생성해야 한다. 각 Pakcage_ID는 하나의 택배 상자라고 생각하면 된다. 또한, 도착 장소가 동일하다면 같은 운송수단으로 배송하는 것이 더 좋은 데이터로 판단했다. 또한, 일부 Package는 창고에 있을 수 있다. 이 데이터에서는 Package_ID 8,9,10을 창고에 놓고 상태를 wait로 하였다. 또한, 1,2,3은 사고가 났다고 가정 후 상태를 wait로 하였다. 그 외 나머지 Package는 모두 배송을 완료하였다고 가정하고 done으로 하였다. 또한, 배송된 물품의 시간은 모두 2022로 하여, 나머지 쿼리에 대해 수행될 수 있도록 하였다.

ex) insert into Package values (4, 3, 4, "name2", "done", 2, "electronics");

6. C코드 및 쿼리

우선 기본 메인은 while(1)을 통해 사용자로부터 0을 입력받기 전까지 계속 명령을 수행한다. 또한 주어진 뼈대 코드를 이용해, DB와 연결 후 해당 쿼리문을 수행하도록 하였다. 20191619.txt에 저장된 쿼리문을 수행하면, physical schema와 동일한 테이블이 생성되고 데이터가 생성된다. 이제, 파일을 한줄씩 읽고, mysql_query를 통해 수행하게 된다. 기본적인 동작은 과제에서 요구하는 것과 동일하다.

6-1 Type 1

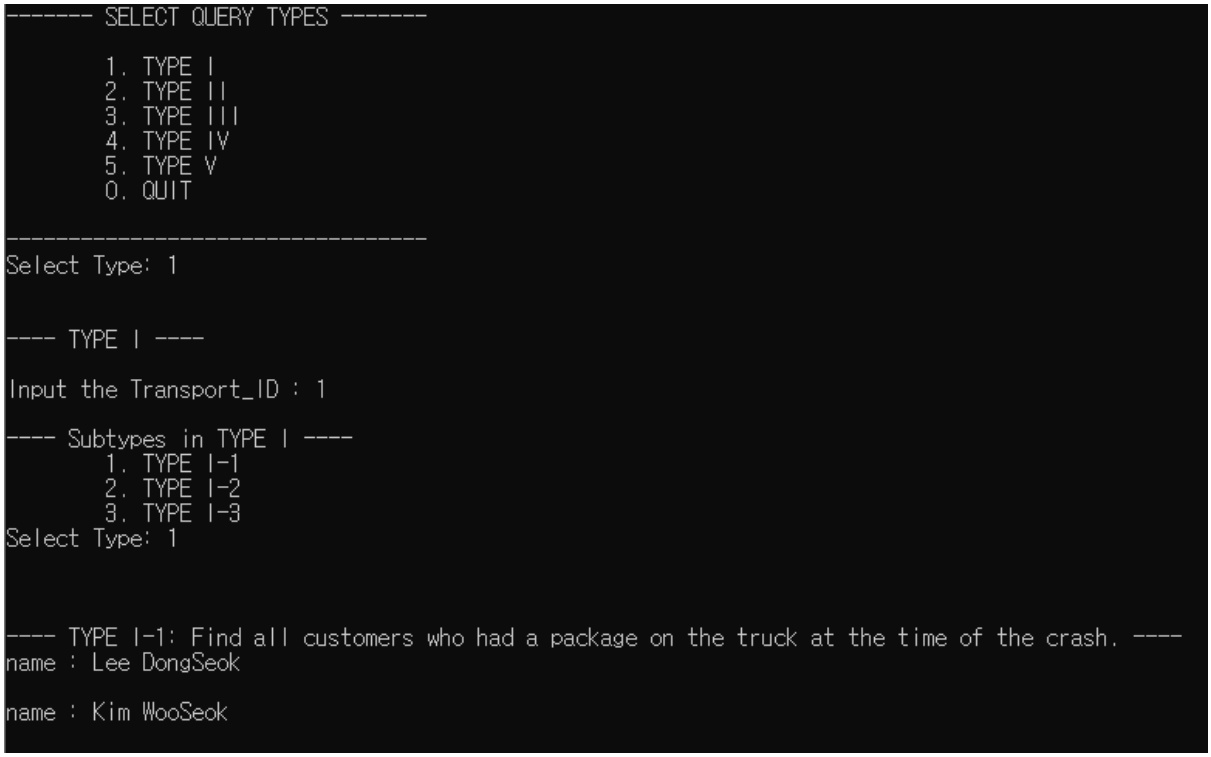
Type1은 `execute_query_type_1(connection);`함수에 작성되었다. 해당 함수를 수행하면, Transport_ID를 입력받는다. (임의로 생성한 데이터에서는 ID=1인 경우에 사고로 가정했다) 이후 Submenu를 보여주고 다시 사용자로부터 입력받는다. 이때, 사용자가 0을 입력하면 처음 메인메뉴로 돌아가도록 작성했다. 다음은 각 SubeType 별 쿼리문이다.

Type 1-1

"SELECT DISTINCT c.name FROM Customer AS c INNER JOIN Package p ON c.Customer_ID = p.Customer_ID INNER JOIN Shipment s ON p.Package_ID =

s.Package_ID INNER JOIN Transport t ON s.Transport_ID = t.Transport_ID WHERE t.Transport_ID = %d AND t.status = 'accident' AND s.delivered_status = 'no'"

사용자로부터 입력받은 Transport_ID로 부터 Transport 테이블의 status가 accident 그리고 Shipment 테이블에서 delivered_status가 no인 Pakcage를 주문한 Customer를 찾는다. JOIN을 사용하여 Customer, Package, Shipment 및 Transport 테이블을 연결했다.



Type 1-2

SELECT DISTINCT p.recipient_name FROM Package AS p INNER JOIN Shipment s ON p.Package_ID = s.Package_ID WHERE s.Transport_ID = %d AND s.delivered_status = 'no'

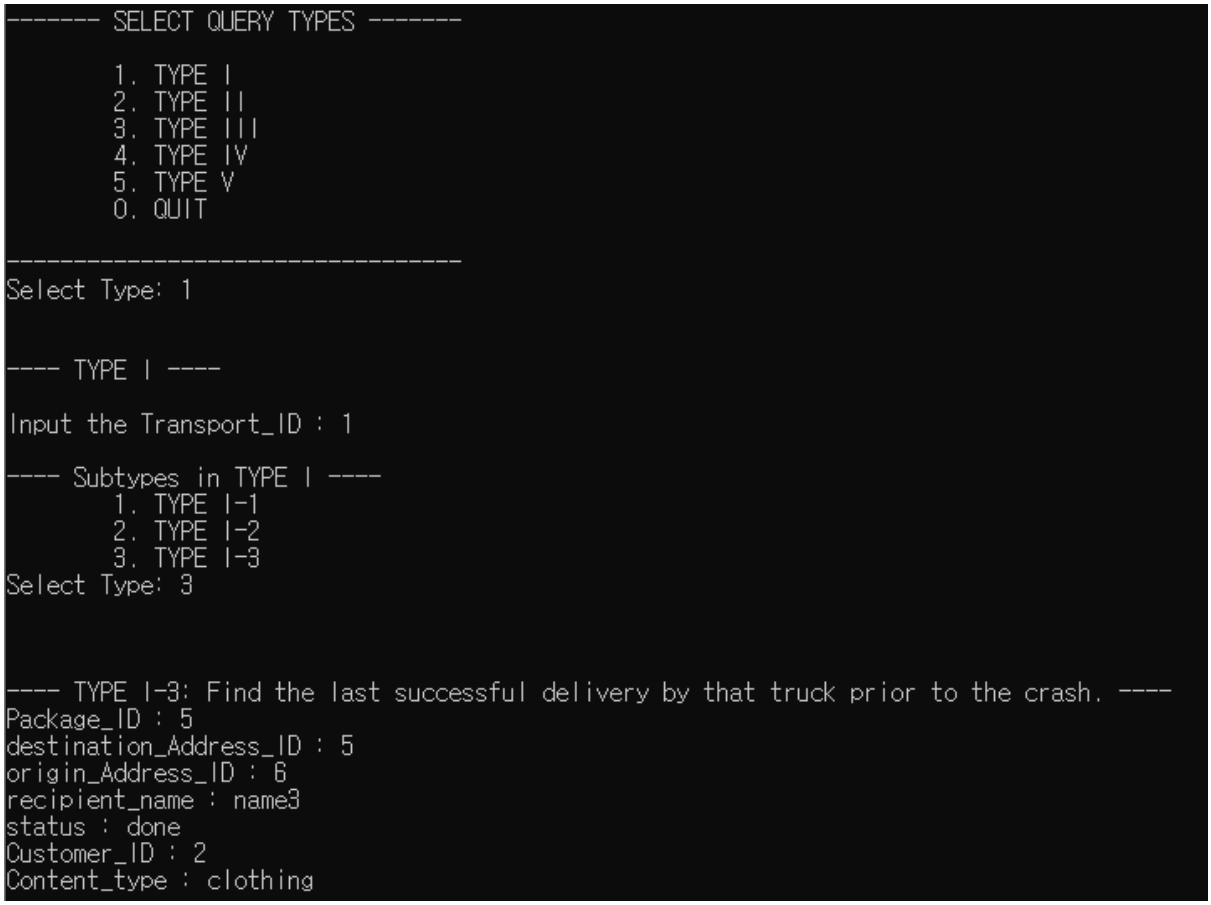
마찬가지로 사용자로부터 입력받은 Transport_ID로 부터 Transport 테이블의 status가 accident 그리고 Shipment 테이블에서 delivered_status가 no인 Pakcage를 찾고, recipient name을 찾는다. JOIN을 사용하여 Customer, Package, Shipment 및 Transport 테이블을 연결했다.

Type 1-3

SELECT s.Package_ID INTO @last_delivered_package_id FROM Shipment AS s WHERE s.Transport_ID = %d AND s.delivered_status = 'yes' ORDER BY s.delivered_date DESC LIMIT 1

SELECT p.* FROM Package AS p WHERE p.Package_ID = @last_delivered_package_id

사용자로부터 입력받은 Transport_ID로 부터 shipment의 delivered_status가 yes인 것을 찾고 ORDER BY와 LIMIT를 사용하여 가장 최근에 배송된 Package를 찾는다. 다음으로, @last_delivered_package_id에 저장한다. 이제, 해당 변수에 저장된 정보를 통해, 해당 Package의 모든 정보를 출력한다.

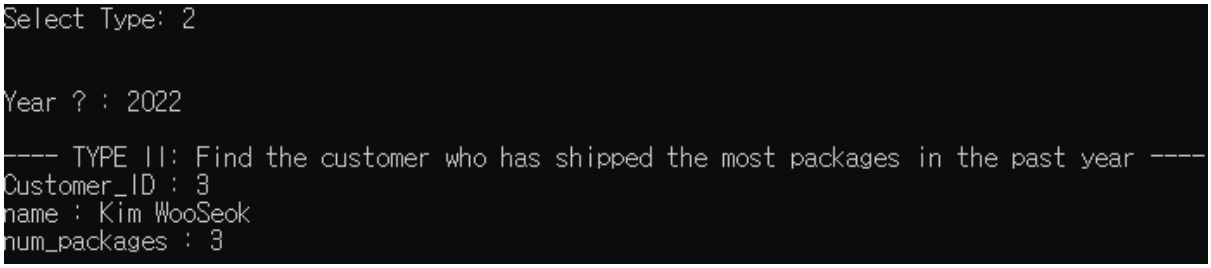


6-2 Type 2

SELECT c.Customer_ID, c.name, COUNT(p.Package_ID) AS num_packages FROM Package p JOIN Shipment s ON p.Package_ID = s.Package_ID JOIN Customer c ON c.Customer_ID = p.Customer_ID WHERE YEAR(s.shipment_date) = %d GROUP BY c.Customer_ID ORDER BY num_packages DESC LIMIT 1", year

마찬가지로 사용자로부터 year를 입력받는다. 예로 2022를 입력받으면, shipment_date에서 해당 연도에 가장 많은 패키지를 발송한 고객을 찾는다. 이때, Shipment에서 Package_ID를 찾고 Package_ID를 통해 Customer_ID를 찾는다.

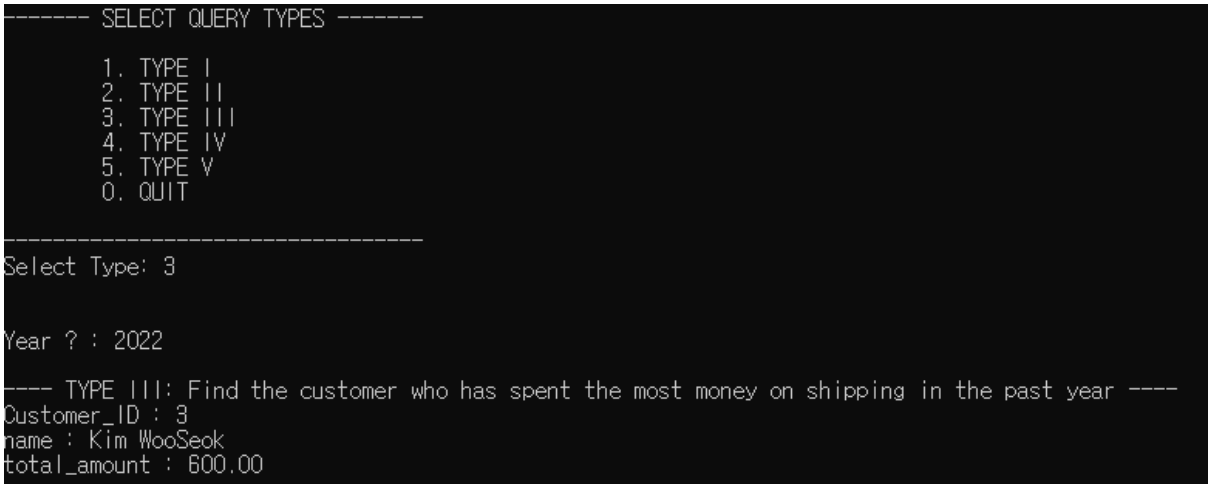
GROUP BY c.Customer_ID 구문을 통해 각 고객별로 그룹화하고, 각 그룹 내에서 패키지의 개수를 세어 'num_packages'에 저장한다. 이후, 정렬 한 후 가장 많은 순서를 반환한다.



6-3 Type 3

SELECT b.Customer_ID, c.name, SUM(b.amount) AS total_amount FROM Billing b JOIN Customer c ON b.Customer_ID = c.Customer_ID WHERE YEAR(b.billing_date) = %d GROUP BY b.Customer_ID ORDER BY total_amount DESC LIMIT 1", year

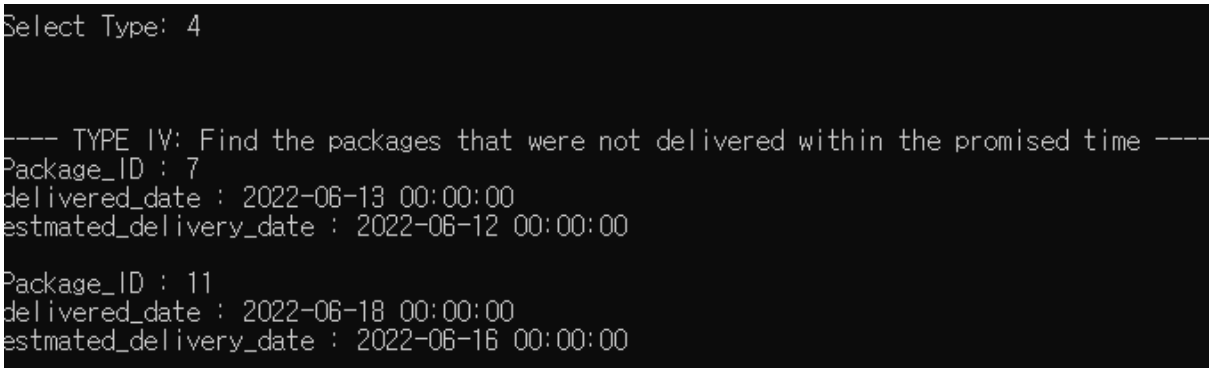
앞선 쿼리와 매우 비슷하다. 우선 사용자로부터 year를 입력받는다. 이후 가장많은 금액을 지불한 고객을 Billing 테이블에서 찾는다. 이때, GROUP BY함수를 사용하여 customer_id로 그룹화 하고 SUM함수를 통해 총 bill을 계산하며 ORDER BY를 통해 정렬한다.



6-4 Type 4

SELECT p.Package_ID, s.delivered_date, a.estmated_delivery_date FROM Package p JOIN Shipment s ON p.Package_ID = s.Package_ID JOIN Address_Info a ON p.destination_Address_ID = a.destination_Address_ID AND p.origin_Address_ID = a.origin_Address_ID WHERE s.delivered_date > a.estmated_delivery_date

이 쿼리는 예상 배송 날짜(약속 날짜) 보다 늦게 배송된 패키지를 모두 찾는다. 이 때, Package로부터 Shipment에서 배송 날짜를 확인하고 Address_Info로부터 예상 배송날짜를 확인한다. 이후, 날짜를 비교하여 원하는 반환값을 출력한다.



6-5 Type 5

Type1과 비슷하게 구현했다. 우선, 사용자가 5번 메뉴를 누르면 Submenu가 나오며, 사용자는 생성하고 싶은 타입의 bill을 선택할 수 있다. 각 타입에 따른 공통 쿼리는 다음과 같다.

SELECT * FROM Customer WHERE Customer_ID = %d", customer_id

SELECT a.* FROM Address a JOIN Customer c ON a.Address_ID = c.Address_ID
WHERE c.Customer_ID = %d", customer_id

해당 쿼리는 사용자로부터 Customer_ID를 입력받고 고객 정보와 주소정보를 출력해 준다. 또한, 아래 변수를 활용해 각 가격을 책정했다. 따로, 데이터 테이블을 만들어 관리 하는 것 보다 자주 변경될 사항 같다고 생각하여 변수로 선정하였다.

```
int express = 200;
int international = 250;
int domestic = 100;
int truck = 50;
int ship = 100;
int airplane = 300;
```

Simple Bill

```
sprintf(query,
    "SELECT SUM(b.amount "
    "+ CASE a.service_type WHEN 'express' THEN %d WHEN 'international' THEN %d WHEN 'domestic' THEN %d ELSE 0 END "
    "+ CASE t.type WHEN 'truck' THEN %d WHEN 'ship' THEN %d WHEN 'airplane' THEN %d ELSE 0 END) as total "
    "FROM Billing b "
    "JOIN Package p ON b.Package_ID = p.Package_ID "
    "JOIN Address_Info a ON p.destination_Address_ID = a.destination_Address_ID and p.origin_Address_ID = a.origin_Address_ID "
    "JOIN Shipment s ON b.Package_ID = s.Package_ID "
    "JOIN Transport t ON s.Transport_ID = t.Transport_ID "
    "WHERE b.Customer_ID = %d AND YEAR(b.billing_date) = %d AND MONTH(b.billing_date) = %d", express, international, domestic, t
```

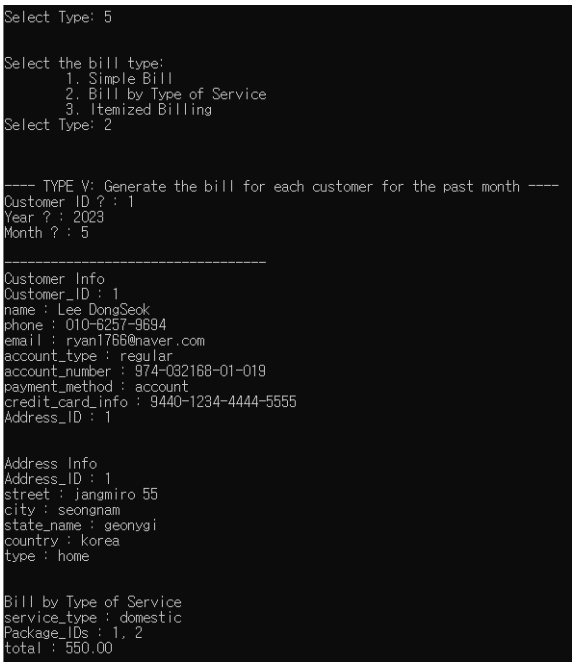
위 코드는 사용자가 지불해야할 총 비용을 출력한다. 이때, Billing테이블로부터 사용자가 배송요청한 Package_ID를 찾고, 해당 Package_ID로부터 Address_Info에 저장되어있는 service_type을 찾는다. 다음으로, Shipment와 Transport로부터 사용자의 배송에 사용된 운송수단을 찾는다. 이후, SUM으로 총 비용을 출력한다.

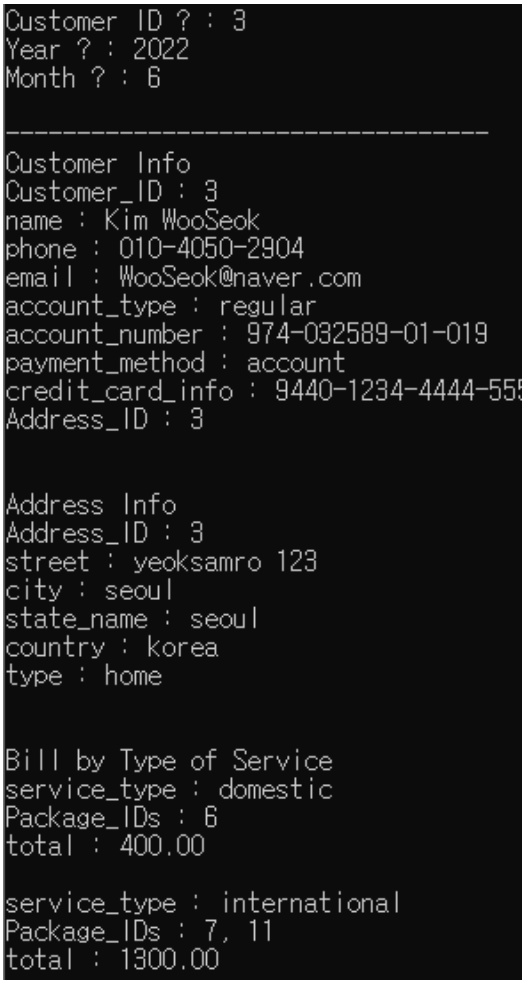
```
------- SELECT QUERY TYPES -------

        1. TYPE I
        2. TYPE II
        3. TYPE III
        4. TYPE IV
        5. TYPE V
        0. QUIT

---------------------------------
Select Type: 5


Select the bill type:
        1. Simple Bill
        2. Bill by Type of Service
        3. Itemized Billing
Select Type: 1



---- TYPE V: Generate the bill for each customer for the past month ----
Customer ID ? : 1
Year ? : 2023
Month ? : 5

---------------------------------
Customer Info
Customer_ID : 1
name : Lee DongSeok
phone : 010-6257-9694
email : ryan1766@naver.com
account_type : regular
account_number : 974-032168-01-019
payment_method : account
credit_card_info : 9440-1234-4444-5555
Address_ID : 1


Address Info
Address_ID : 1
street : jangmiro 55
city : seongnam
state_name : geonygi
country : korea
type : home


Simple Bill
total : 550.00
```

Service Type

```
"SELECT a.service_type, GROUP_CONCAT(b.Package_ID SEPARATOR ', ') as Package_IDs, SUM(b.amount "
"+ CASE a.service_type WHEN 'express' THEN %d WHEN 'international' THEN %d WHEN 'domestic' THEN %d ELSE 0 END "
"+ CASE t.type WHEN 'truck' THEN %d WHEN 'ship' THEN %d WHEN 'airplane' THEN %d ELSE 0 END) as total "
"FROM Billing b "
"JOIN Package p ON b.Package_ID = p.Package_ID "
"JOIN Address_Info a ON p.destination_Address_ID = a.destination_Address_ID and p.origin_Address_ID = a.origin_Address_ID "
"JOIN Shipment s ON b.Package_ID = s.Package_ID "
"JOIN Transport t ON s.Transport_ID = t.Transport_ID "
"WHERE b.Customer_ID = %d AND YEAR(b.billing_date) = %d AND MONTH(b.billing_date) = %d "
"GROUP BY a.service_type", express, international, domestic, truck, ship, airplane, customer_id, year, month);
```

앞선 Simple Bill을 살짝 변형하여 만들었다. 비슷하게 우선 Billing으로부터 사용자가 배송에 요청한 Package를 찾는다. 해당 Package_ID로부터 Address_Info에 저장되어 있는 service_type을 찾는다. 다음으로, Shipment와 Transport로부터 사용자의 배송에 사용된 운송수단을 찾는다. 이때 총 비용을 계산하는 것은 비슷하지만, 각 service_type으로 그룹화 하고 해당 서비스에 따른 총 가격을 계산한다.





Itemize

마지막으로 각 Package에 따른 bill이다. 마찬가지로 앞서 설명한 쿼리들과 매우 유사하다

```
"SELECT b.Package_ID, a.service_type, t.type, SUM(b.amount "
"+ CASE a.service_type WHEN 'express' THEN %d WHEN 'international' THEN %d WHEN 'domestic' THEN %d ELSE 0 END "
"+ CASE t.type WHEN 'truck' THEN %d WHEN 'ship' THEN %d WHEN 'airplane' THEN %d ELSE 0 END) as total "
"FROM Billing b "
"JOIN Package p ON b.Package_ID = p.Package_ID "
"JOIN Address_Info a ON p.destination_Address_ID = a.destination_Address_ID and p.origin_Address_ID = a.origin_Address_ID "
"JOIN Shipment s ON b.Package_ID = s.Package_ID "
"JOIN Transport t ON s.Transport_ID = t.Transport_ID "
"WHERE b.Customer_ID = %d AND YEAR(b.billing_date) = %d AND MONTH(b.billing_date) = %d "
"GROUP BY b.Package_ID, a.service_type, t.type", express, international, domestic, truck, ship, airplane, customer_id, year,
```

다만 이번에는 Package_ID, service_type, Transport_type으로 그룹화하여 각 Package_ID에 따른 총 비용을 계산하여 보여준다.

```
---------------------------------
Select Type: 5


Select the bill type:
        1. Simple Bill
        2. Bill by Type of Service
        3. Itemized Billing
Select Type: 3



---- TYPE V: Generate the bill for each customer for the past month ----
Customer ID ? : 1
Year ? : 2023
Month ? : 5

---------------------------------
Customer Info
Customer_ID : 1
name : Lee DongSeok
phone : 010-6257-9694
email : ryan1766@naver.com
account_type : regular
account_number : 974-032168-01-019
payment_method : account
credit_card_info : 9440-1234-4444-5555
Address_ID : 1


Address Info
Address_ID : 1
street : jangmiro 55
city : seongnam
state_name : geonygi
country : korea
type : home


Itemized Billing
Package_ID : 1
service_type : domestic
type : truck
total : 250.00

Package_ID : 2
service_type : domestic
type : truck
total : 300.00
```