데이터베이스시스템 Project 2

20170364 김수빈

Index

1. Physical Schema

① BCNF Form

Project 1에서 설계한 데이터베이스 모델은 이미 BCNF 에 해당한다. 따라서 추가적인 작업을 수행하지 않았다. 각 relation 에서 성립하는 nontrivial functional dependency $\alpha \rightarrow \beta$ 에서 α 는 모두 superkey 이다. 따라서 Project 1 에서 사용한 schema에서 더 이상의 decomposition 을 진행하지 않는다. 다만, 주어진 Type들에 대한 결과를 도출하기 위하여 몇몇 relation의 attribute를 수정하였는데, 그 내용에 대해서는 다음 항목에서 구체적으로 설명한다.

BCNF 의 simplified test를 이용해 대표적으로 customer relation에 대하여 테스트해보면,

customer (customer_id, name, address, phone_num, email)
FD = {customer_id -> name, customer_id -> address, customer_id - > phone_num, customer_id -> email}
customer_id+ = (customer_id, name, address, phone_num, email)

따라서 FD 의 $\alpha -\rangle \beta$ 에서 α 에 해당하는 customer_id 의 closure 가 *customer* 의 attribute 를 모두 포함하므로, 이는 BCNF를 만족한다.

② Modification

Type IV 에 사용하기 위해, year, month, date, time 으로 각각 구분되어 있던 속성들을 하나의 정수형으로 표현한 datetime NUMERIC(12.0) 속성을 timetable relation에 추가하였다(ex. year = 2020, month = 01, date = 31, time = 15.30 datetime = 202001311530). 비교연산자를 이용하여 datetime 값의 대소 비교를 통해 어떤 timetable tuple이 선행하는지를 쉽게 파악할 수 있다.

Type IV 에 사용하기 위해, package relation에 promised 속성을 추가하였다. 해당 package 의 출발 시간으로부터 일 수 기준으로 더한 값을 저장하고 있는 속성이다(ex. 발송 시간이 202001311530인 package의 timeliness 가 overnight 인 경우, promised에 202002012359 를 저장한다). 해당 속성과 실제 택배가 도착한 시간을 나타내는 timetable 의 datetime 값을 비교하여 택배가 도착해야 하는 시간 내에 잘 도착했는지를 파악할 수 있다.

③ Physical Schema

* 가독성을 위하여 Erwin 대신 workbench 에서 작업한 Table create 쿼리를 통해 설명한다.
* foreign key의 경우, referenced table 에서의 속성과 동일하므로 설명을 생략한다.

▷ customer

```
create table customer
(customer_id    varchar(8),
 name           varchar(20) not null,
 address        varchar(50),
 phone_num      varchar(11) not null,
 email          varchar(30),
 primary key (customer_id));
```

customer_id	(PK) 회원가입시 등록하게 되는 아이디로, unique한 값을 가진다. ID가 너무 길면 관리가 불편하므로 적당히 8자 정도로 설정하였다. PK이므로 null 값이 자동으로 허용되지 않는다.
name	회원의 이름이다. 초기에는 한국어 이름을 생각하여 최대 5자까지만 허용하였으나, 인코딩 방식을 변경하지 않기 위해 영문 이름으로의 등록 방식을 채택하여 적당히 20자 정도로 설정하였다. 이름은 존재하지 않는 경우가 있을 수 없으므로, null값을 허용하지 않는다.
address	해당 고객의 기본 주소지 정보이다. 적당히 50자 정도의 문자열로 설정하였다. 회원가입시에 기본 주소 정보를 등록하지 않는 경우도 많고, package를 보낼 때는 *destination*에서 주소를 가져오게 되므로 *customer*에서의 속성 address는 null값을 허용한다.
phone_num	해당 고객의 기본 휴대폰 번호 정보이다. 대한민국의 표준 휴대전화 번호를 기준으로 하여 총 11자 (010xxxxxxxx)의 문자열로 설정하였다. 초기에는 휴대폰 '번호' 이므로 NUMERIC(11.0) 으로 설정하였으나, 0으로 시작하게 되는 휴대폰 번호의 특성상 (010, 011 등) 번호로 저장할 경우 이를 숫자로 인식하여 맨 앞의 0이 사라지게 되어 문자 형태로 저장하는 방식을 택하였다. *customer* 에 존재하는 정보는 택배를 '발송' 하는 회원의 정보이므로 휴대폰 번호는 필수 정보가 아니라고 판단하여 null을 허용하였다.
email	해당 고객의 이메일 주소 정보이다. 적당히 30자 정도의 문자열로 설정하였다. 택배를 '발송'하는 고객의 정보이지만 적어도 하나 이상의 연락망은 필요하다고 판단하여 phone_num 이 null을 허용한 만큼 email은 null을 허용하지 않았다. (※ phone_num 과 email 중 어떤 것에 null을 허용할 것인가에 대한 판단은 실존하는 쇼핑몰 등의 사이트의 회원가입 양식을 참고하였다.)

▷ destination

```
create table destination
(address    varchar(50),
 name       varchar(20),
 phone_num  varchar(11),
 primary key (address, name, phone_num));
```

address	(PK) 택배가 도착해야 하는 주소, 즉 목적지를 의미한다. *customer*에서 회원의 주소를 설정했던 것과 동일하게 길이 50의 문자열로 설정하였다.
name	(PK) 택배를 받는 사람의 이름, 즉 수취인을 의미한다. *customer*에서 회원의 이름을 설정했던 것과 동일하게 길이 20으로 설정하였다.
phone_num	(PK) 택배를 받는 사람의 전화번호를 의미한다. *customer*에서 회원의 휴대폰 번호를 설정했던 것과 동일하게 길이 11의 문자열로 설정하였다.

▷ package

```
create table package
(package_id     varchar(10),
 customer_id    varchar(8) not null,
 package_type   varchar(6) check (package_type in ('flat', 'small', 'larger')) not null,
 weight         numeric(7,2) check (weight > 0) not null,
 timeliness     varchar(10) check (timeliness in ('overnight', 'second day', 'longer')) not null,
 promissed      numeric(12,0),
 caution_code   varchar(6) check (caution_code in ('hazard', 'intl')),
 address        varchar(50) not null,
 name           varchar(20) not null,
 phone_num      varchar(11) not null,
 primary key (package_id),
 foreign key (customer_id) references customer (customer_id)on delete cascade,
 foreign key (address, name, phone_num) references destination (address, name, phone_num)on delete cascade);
```

package_id	(PK) 택배의 운송장 번호이다. 알파벳 코드와 숫자로 이루어진 id로 설정할 것이므로 길이 10의 문자열로 결정하였다.
package_type	택배의 유형을 의미한다. flat, small, larger 의 세 가지 값만을 가질 수 있으므로 check in 을 통해 constraint를 추가하고, 모든 택배는 이 세 가지 유형 이외의 경우가 존재하지 않으므로 null을 허용하지 않는다.
weight	택배의 무게를 의미한다. 소수점 아래 두 자리까지 표시하기 위해 NUMERIC(7.2)로 설정하였다. 무게는 0 이하일 수 없으므로 항상 양수인지를 확인하고, 무게는 가격 측정 등의 중요 지표가 되므로 null을 허용하지 않는다.
timeliness	택배의 운송 신속도를 의미한다. overnight, second day, longer 의 세 가지 값만을 가질 수 있으므로 check in을 통해 constraint를 추가하고, 모든 택배는 이 세 가지 유형 이외의 경우가 존재하지 않으므로 null을 허용하지 않는다.
promised	택배가 도착해야 하는 시간을 의미한다. *package의* datetime과 동일하게 길이 12로 설정하였다. timeliness를 통해서 도출할 수 있는 값으로, timeliness가 overnight인 경우, 택배의 발송일 다음날 23시 59분까지, timeliness가 second day인 경우, 택배의 발송일 이틀 후 23시 59분까지, timeliness가 longer인 경우 무한대로 설정된다. timeliness가 longer인 경우, 허용하는 year보다 큰 값, 즉 무한대로 값을 채워 넣을 수도 있지만 그냥 null로 남겨두어도 되므로 null을 제한하지 않는다.
caution_code	특별한 관리가 필요한 택배에 부여되는 코드이다. hazard 와 intl (international)의 두 가지 값만을 가질 수 있으므로 check in을 통해 constraint를 추가하고, 특별한 관리가 필요하지 않은 택배의 경우 아무런 코드도 부여되지 않고 null로 남아있으므로 null을 제한하지 않는다.

▷ contract

```
create table contract
(customer_id    varchar(8),
 duration       numeric(2,0) not null,
 pay_date       numeric(2,0) not null,
 pay_amount     numeric(7,0) check (pay_amount >= 0) not null,
 primary key (customer_id),
 foreign key (customer_id) references customer (customer_id)on delete cascade);
```

duration	계약의 지속 기간을 의미한다. 단위는 개월로 저장되므로 길이 2의 NUMERIC으로 설정하였다. 계약 기간은 누락되어서는 안 되는 정보이므로 null을 허용하지 않는다.
pay_date	계약금 납부일을 의미한다. 매 달 해당 날짜에 고객은 계약된 금액을

	납부하게 된다. 날짜 정보만을 담고 있으면 되므로 길이 2의 NUMERIC으로 설정하였고, duration과 마찬가지로 누락되어서는 안 되는 정보이므로 null을 허용하지 않는다.
pay_amount	납부해야 하는 금액을 의미한다. 납부액은 음수일 수 없으므로 0 이상인지를 체크하고, 마찬가지로 누락되어서는 안 되는 정보이므로 null을 허용하지 않는다.

▷ bill

```
create table bill
(customer_id    varchar(8),
 year           numeric(4,0) check (year > 1701 and year < 2100),
 month          numeric(2,0) check (month <= 12 and month >= 1),
 owed           numeric(7,0) check (owed >= 0) not null,
 primary key (customer_id, year, month),
 foreign key (customer_id) references customer (customer_id) on delete cascade);
```

year	택배 영수증이 생성된 연도를 의미한다. 연도이므로 길이 4의 NUMERIC으로 설정하였고, 터무니없는 값의 잘못된 삽입을 방지하기 위하여 1701년과 2100년 사이임을 체크하는 constraint를 추가하였다.
month	택배 영수증이 생성된 월을 의미한다. 월이므로 길이 2의 NUMERIC으로 설정하였고, 1부터 12 사이의 수인지를 체크하는 constraint를 추가하였다.
owed	해당 택배 예약 건에 대하여 지불해야 하는 금액을 의미한다. 지불해야 하는 금액이 음수일 수는 없으므로 0 이상인지를 확인하는 constraint를 추가하였다. 누락되어서는 안 되는 정보이므로 null을 허용하지 않는다.

▷ history

```
create table history
(package_id     varchar(8),
customer_id varchar(8) not null,
year            numeric(4,0) check (year > 1701 and year < 2100) not null,
month           numeric(2,0) check (month <= 12 and month >= 1) not null,
date            numeric(2,0) check (date >= 1 and date <= 31) not null,
time            numeric(4,2) not null,
price           numeric(7,0) check (price >= 0),
primary key (package_id),    foreign key (package_id) references package (package_id)
on delete cascade,
foreign key (customer_id) references customer (customer_id)     on delete cascade   );
```

year	고객의 택배 발송 연도 정보를 의미한다. 편의상 해당 택배 bill의 생성 연도와 동일하게 데이터를 생성하였으나, 실제로는 bill은 결제 시점을 기준으로 생성되므로 약간의 차이가 존재할 수 있다. 연도이므로 길이 4의 NUMERIC으로 설정하였고, 터무니없는 값의 잘못된 삽입을 방지하기 위하여 1701년과 2100년 사이임을 체크하는 constraint를 추가하였다.
month	고객의 택배 발송 월 정보를 의미한다. 편의상 해당 택배 bill의 생성 월과 동일하게 데이터를 생성하였으나, 실제로는 bill은 결제 시점을 기준으로 생성되므로 약간의 차이가 존재할 수 있다. 월이므로 길이 2의

	NUMERIC으로 설정하였고, 1부터 12 사이의 수인지를 체크하는 constraint를 추가하였다.
date	고객의 택배 발송 일 정보를 의미한다. 날짜이므로 길이 2의 NUMERIC으로 설정하였고, 1부터 31 사이의 수인지를 체크하는 constraint를 추가하였다.
time	고객의 택배 발송 시간 정보를 의미한다. 시간이므로 총 길이 4의 NUMERIC으로 설정하였고, 시간과 분을 나눠서 표시하기 위해 소수점 아래 두 자리는 분으로 구분하여 NUMERIC(4.2)의 형태로 설정하였다.
price	발송한 택배의 가격 정보를 의미한다. 원(₩)을 기준으로 하므로 소수점 아래의 수는 고려하지 않고 NUMERIC(7.0)으로 설정하였다. 또한 가격은 음수일 수 없으므로 0 이상인지를 확인하는 constraint를 추가하였다. 이 때, *contract*의 tuple로 존재하는 고객의 경우, 해당 택배 건의 가격과 관계 없이 정액제로 지불하게 되므로 price에 null을 삽입한다.

▷ timetable

```
create table timetable
(package_id     varchar(8),
year            numeric(4,0) check (year > 1701 and year < 2100),
month           numeric(2,0) check (month <= 12 and month >= 1),
date            numeric(2,0) check (date >= 1 and date <= 31),
time            numeric(4,2),
datetime        numeric(12,0),
location        varchar(50) not null,
transportation  varchar(9)      check (transportation in ('truck', 'plane', 'warehouse')) not null,
transp_id       numeric(4,0),
arrive          numeric(1,0) check (arrive in (0,1)) not null,
primary key (package_id, year, month, date, time),
foreign key (package_id) references package (package_id) on delete cascade);
```

year	배송되고 있는 택배의 날짜 정보 중 연도를 나타낸다. 연도이므로 길이 4의 NUMERIC으로 설정하였고, 터무니없는 값의 잘못된 삽입을 방지하기 위하여 1701년과 2100년 사이임을 체크하는 constraint를 추가하였다.
month	배송되고 있는 택배의 날짜 정보 중 월 정보를 나타낸다. 월이므로 길이 2의 NUMERIC으로 설정하였고, 1부터 12 사이의 수인지를 체크하는 constraint를 추가하였다.
date	배송되고 있는 택배의 날짜 정보 중 일 정보를 나타낸다. 일이므로 길이 2의 NUMERIC으로 설정하였고, 1부터 31 사이의 수인지를 체크하는 constraint를 추가하였다.
time	배송되고 있는 택배의 시간 정보를 나타낸다. 하나의 택배가 한 번의 이동으로 배송 완료되는 것이 아니기 때문에, 하나의 택배가(동일한 package_id를 가진 tuple이) 시간차를 두고 여러 번 *timetable* 내에 존재한다. 다른 relation 내의 시간 정보를 나타내는 속성들과 마찬가지로 NUMERIC(4.2) 로 설정하여, 소수점 아래 두 자리 수는 분을 나타내는데 사용한다.
datetime	tuple들간의 시간 선후관계를 보다 더 간편하게 비교하기 위한 속성으로, 네 개로 나눠져 있던 year, month, date, time 속성의 값들을 하나로 합쳐 길이 12(4+2+2+4)의 NUMERIC으로 표현한다.
location	해당 시간대에 택배가 위치해있는 장소를 나타낸다.
transportation	해당 시간대에 택배 이송에 이용된 수단을 나타낸다. 현재로서는 plane, truck, warehouse 의 세 가지 값만을 가질 수 있으므로 check in

	을 통해 constraint를 추가했고, null 값을 허용하지 않는다.
transp_id	transportation 에 해당하는 id 번호, 일종의 차량번호 같은 것으로, id가 존재하지 않는 transportation이 있을 수 있으므로 null을 허용한다.
arrive	하나의 package_id 에 대하여 여러 개의 tuple이 존재하기 때문에, 가장 마지막 이동에 해당하는 tuple, 즉 최종 목적지로의 이동을 나타내는 tuple을 보다 더 빠르게 파악하기 위한 일종의 flag로 사용한다. 0이면 중간 이동 과정, 1이면 최종 목적지 도착을 의미한다. 따라서 check in으로 0, 1의 두 가지 값만을 가지도록 constraint를 추가한다.

2. Database Explanation.

* 쿼리 테스트를 위하여 임의로 삽입한 약 15개의 데이터에 대하여 간략하게 설명하고 실제 query 실행 결과와 비교하여 검토한다.

① customer

customer_id	name	address	phone_num	email

```
insert into customer values ('bin430', 'Soobin Kim', 'Seoul, South Korea', '01012345678', 'bin430@naver.com');
insert into customer values ('abc123', 'abc Kim', 'Busan, South Korea', '01012341234', 'abc@naver.com');
insert into customer values ('def456', 'def Kim', 'NY, South Korea', '01088888888', 'def@naver.com');
insert into customer values ('hahahaha', 'haha Lee', 'Jeju island', '01034566543', 'haha@google.com');
insert into customer values ('me', 'me park', 'Hello world', '01099999999', 'me@google.com');
insert into customer values ('sogang', 'sogang Kim', 'Mapo gu, Seoul, South Korea', '01055555555', 'sogang@ac.kr');
insert into customer values ('univ', 'univ Kim', 'Baekbum ro, Seoul, South Korea', '01022223333', 'univ@sogang.ac.kr');
insert into customer values ('zzzz', 'zzz Lee', 'dont know, South Korea', '01092839292', 'zzz@naver.com');
insert into customer values ('hihi', 'hihi park', 'smileland', '01098765432', 'hihi@google.com');
insert into customer values ('so many', 'so min', 'Daegu, South Korea', '01077777777', 'somany@daum.net');
insert into customer values ('database', 'db kim', 'database lab, South Korea', '01056565656', 'db@naver.com');
insert into customer values ('toinsert', 'insert Jung', 'mysql, South Korea', '01076543210', 'insert@google.com');
insert into customer values ('hardhard', 'hard Seo', 'hard gu, South Korea', '01098765432', 'hard@daum.net');
insert into customer values ('noidea', 'idea Lim', 'idea gu, South Korea', '01076547654', 'idea@naver.com');
insert into customer values ('onemore', 'one Kim', 'one one, South Korea', '01057575757', 'one@google.com');
```

삽입한 *customer* 의 tuple들이다. 자명하므로 추가적인 설명은 생략한다.

② package

package_id	customer_id	package_type	weight	timeliness	promised	caution_code	address	name	phone_num

```
insert into package values ('ABC10000', 'bin430', 'flat', '00500.50', 'overnight', '202004212359', null, 'Seoul, South Korea', 'Soobin Kim', '01012345678');
insert into package values ('ABC11000', 'abc123', 'larger', '00230.50', 'overnight', '201902192359', null, 'Busan, South Korea', 'abc Kim', '01012341234');
insert into package values ('ABC12000', 'onemore', 'flat', '34561.00', 'overnight', '202008132359', 'hazard', 'one one, South Korea', 'one Kim', '01057575757');
insert into package values ('ABC13000', 'bin430', 'flat', '03400.00', 'overnight', '201807272359', null, 'Seoul, South Korea', 'Soobin Kim', '01012345678');
insert into package values ('ABC14000', 'onemore', 'larger', '00400.30', 'overnight', '202008132359', null, 'Seoul, South Korea', 'Soobin Kim', '01012345678');
insert into package values ('ABC15000', 'sogang', 'small', '00500.00', 'second day', '202001292359', null, 'Mapo gu, South Korea', 'sogang Kim', '01055555555');
insert into package values ('DEF10000', 'sogang', 'flat', '00800.50', 'second day', '201812192359', 'hazard', 'Mapo gu, South Korea', 'sogang Kim', '01055555555');
insert into package values ('DEF11000', 'univ', 'larger', '05600.00', 'second day', '201912022359', null, 'Seoul, South Korea', 'Soobin Kim', '01012345678');
insert into package values ('DEF12000', 'univ', 'flat', '08700.50', 'second day', '202004222359', null, 'Baekbum ro, South Korea', 'univ Kim', '01022223333');
insert into package values ('DEF13000', 'bin430', 'flat', '00100.00', 'second day', '201804232359', 'hazard', 'Seoul, South Korea', 'Soobin Kim', '01012345678');
insert into package values ('DEF14000', 'bin430', 'larger', '00200.50', 'second day', '202004142359', null, 'Seoul, South Korea', 'Soobin Kim', '01012345678');
insert into package values ('GHI10000', 'database', 'small', '00100.50', 'second day', '202006122359', null, 'database lab, South Korea', 'db Kim', '01056565656');
insert into package values ('GHI11000', 'database', 'flat', '00100.12', 'longer', '300000000000', 'hazard', 'Seoul, South Korea', 'Soobin Kim', '01012345678');
insert into package values ('GHI12000', 'def456', 'larger', '00530.50', 'longer', '300000000000', 'intl', 'NY, South Korea', 'def Kim', '01088888888');
insert into package values ('GHI13000', 'def456', 'flat', '00530.00', 'longer', '300000000000', null, 'NY, South Korea', 'def Kim', '01088888888');
```

(Type I-2)

```
** Find all recipients who had a package on that truck at the time of the crash **
Customer Name : sogang Kim
Customer Name : def Kim
```

*timetable*을 보면, package_id DEF10000, GHI13000 이 Truck 1721 사고의 영향을 받았음을 할 수 있다. *package* 에서 해당 package_id 를 갖는 tuple을 찾아보면, 해당 택배의 수취인이 “def Kim” 과 “sogang Kim” 임을 알 수 있다.

(Type I-1)

```
** Find all customers who had a package on the truck at the time of the crash **
Customer ID : sogang
Customer ID : def456
```

*timetable*을 보면, package_id DEF10000, GHI13000 이 Truck 1721 사고의 영향을 받았음을 할 수 있다. *package* 에서 해당 package_id 를 갖는 tuple을 찾아보면, 해당 택배를 발송한 고객이 sogang, def456 임을 알 수 있다.

③ destination

address	name	phone_num

```
insert into destination values ('Seoul, South Korea', 'Soobin Kim', '01012345678');
insert into destination values ('dont know, South Korea', 'zzz Lee', '01092839292');
insert into destination values ('new address to deliver', 'new one', '01011111111');
insert into destination values ('another new address', 'new two', '01022222222');
insert into destination values ('NY, South Korea', 'def Kim', '01088888888');
insert into destination values ('Busan, South Korea', 'abc Kim', '01012341234');
insert into destination values ('one one, South Korea', 'one Kim', '01057575757');
insert into destination values ('Mapo gu, South Korea', 'sogang Kim', '01055555555');
insert into destination values ('Baekbum ro, South Korea', 'univ Kim', '01022223333');
insert into destination values ('database lab, South Korea', 'db Kim', '01056565656');
insert into destination values ('smileland, South Korea', 'hihi park', '01098765432');
insert into destination values ('mysql, South Korea', 'insert Jung', '01076543210');
insert into destination values ('new place1', 'new person1', '01099999999');
insert into destination values ('new place2', 'new person2', '01048489393');
insert into destination values ('new place3', 'new person3', '01098980707');
```

주소지로 저장되어 있는 데이터들을 생성하였다. 고객 정보에 저장되어있는 고객의 주소 이외에도, 고객이 추가적으로 배송지 목록을 저장했을 수 있으므로 새로운 value 를 갖는 tuple 들이 존재한다. (new place1, new place2, new place3)

④ timetable

package_id	year	month	date	time	datetime	location	transportation	transp_id	arrive

```
insert into timetable values ('ABC10000', '2020', '04', '20', '15.32', '202004201532', 'Jeju island', 'plane', '522', '0');
insert into timetable values ('ABC10000', '2020', '04', '21', '10.17', '202004211017', 'Busan', 'truck', '1000', '0');
insert into timetable values ('ABC10000', '2020', '04', '22', '06.02', '202004220602', 'Seoul', 'truck', '1721', '1');
insert into timetable values ('GHI11000', '2020', '06', '10', '09.10', '202006100910', 'Jeju island', 'plane', '552', '0');
insert into timetable values ('GHI11000', '2020', '06', '10', '17.20', '202006101720', 'Ulsan', 'warehouse', '100', '0');
insert into timetable values ('GHI11000', '2020', '06', '11', '14.15', '202006111415', 'Seoul', 'truck', '1002', '1');
insert into timetable values ('DEF12000', '2020', '04', '20', '10.15', '202004201015', 'Busan', 'truck', '1000', '0');
insert into timetable values ('DEF12000', '2020', '04', '20', '23.15', '202004202315', 'Seoul', 'truck', '1721', '1');
insert into timetable values ('ABC12000', '2020', '08', '12', '12.10', '202008121210', 'Paju', 'truck', '1723', '0');
insert into timetable values ('ABC12000', '2020', '08', '13', '19.30', '202008131930', 'Seoul', 'truck', '1823', '1');
insert into timetable values ('GHI12000', '2020', '04', '13', '23.40', '202004132340', 'Seoul', 'truck', '1000', '0');
insert into timetable values ('GHI12000', '2020', '04', '15', '06.15', '202004150615', 'NY', 'plane', '525', '1');
insert into timetable values ('DEF10000', '2018', '12', '17', '23.57', '202012172357', 'Gangwon', 'truck', '1721', '0');    #crash
insert into timetable values ('GHI13000', '2018', '12', '17', '12.45', '202012171245', 'Busan', 'truck', '1002', '0');
insert into timetable values ('GHI13000', '2018', '12', '17', '23.50', '202012172350', 'Gangwon', 'truck', '1721', '0');    #crash
```

(Type I-3)

```
** Find the last successful delivery by that truck prior to the crash **
Package ID : ABC10000
```

Truck 1721의 사고의 영향을 받지 않은 package의 id는, ABC10000, DEF12000이고, 둘 중에서 더 최근의 package는 ABC10000임을 확인할 수 있다.

(Type IV)

```
** Find those packages that were not delivered within the promised time **
Package ID : ABC10000
```

package 의 promised를 보면, ABC10000의 경우 202004212359인데, *timetable* 에서 실제로 ABC10000 가 배송 완료된 시간은 4월 22일 오전 6시 2분이다. 따라서 ABC10000 가 늦게 도착했음을 확인할 수 있다.

⑤ history

package_id	customer_id	year	month	date	time	price

```
insert into history values ('ABC10000', 'bin430', '2020', '04', '20', '15.00', NULL);
insert into history values ('ABC11000', 'abc123', '2019', '02', '18', '13.19', '4500');
insert into history values ('ABC12000', 'onemore', '2020', '08', '12', '08.39', '2500');
insert into history values ('ABC13000', 'bin430', '2018', '07', '26', '09.30', NULL);
insert into history values ('ABC14000', 'onemore', '2020', '08', '12', '10.32', '2500');
insert into history values ('ABC15000', 'sogang', '2020', '01', '27', '18.20', NULL);
insert into history values ('DEF10000', 'sogang', '2018', '12', '17', '17.32', NULL);
insert into history values ('DEF11000', 'univ', '2019', '11', '30', '09.37', '2500');
insert into history values ('DEF12000', 'univ', '2020', '04', '20', '06.32', '2500');
insert into history values ('DEF13000', 'bin430', '2018', '04', '21', '11.47', NULL);
insert into history values ('DEF14000', 'bin430', '2020', '04', '12', '10.09', NULL);
insert into history values ('GHI10000', 'database', '2020', '06', '10', '08.11', NULL);
insert into history values ('GHI11000', 'database', '2020', '06', '19', '15.12', NULL);
insert into history values ('GHI12000', 'def456', '2020', '04', '13', '16.39', '2500');
insert into history values ('GHI13000', 'def456', '2018', '12', '16', '17.15', '2500');
```

(Type II)

```
** Find the customer who has shipped the most packages in certain year **
Which Year? : 2018
Customer ID : bin430
```

실제로 *history* 에서, 2018년의 내역을 보면 'bin430' 이 2건, 'sogang'이 1건, 'def456'이 1건으로 'bin430'이 가장 많은 택배를 발송했음을 알 수 있다.

⑥ contract

customer_id	duration	date	pay_amount

```
insert into contract values ('bin430', '06', '30', '5000');
insert into contract values ('sogang', '03', '15', '5000');
insert into contract values ('database', '12', '01', '7000');
```

contract 의 경우, *bill* 에 데이터를 삽입할 때, owed의 value로 *history*의 price를 넣을지 아니면 정기 결제 금액을 넣을지를 결정하기 위하여 frequent customer 를 판단하기 위해 존재하는 table로, 실제로 query test 과정에서는 사용되지 않고, 15명의 고객에 대하여 모든 고객이 contract를 가지고 있는 것이 아니기 때문에 15개 미만의 데이터를 삽입하였다.

⑦ bill

customer_id	year	month	owed

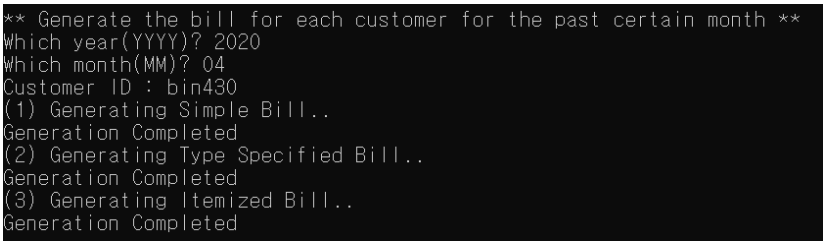
```
insert into bill values ('univ', '2019', '11', '7500');
insert into bill values ('univ', '2020', '02', '2500');
insert into bill values ('abc123', '2019', '02', '4500');
insert into bill values ('onemore', '2020', '08', '5000');
insert into bill values ('def456', '2020', '04', '2500');
insert into bill values ('def456', '2018', '12', '5500');
insert into bill values ('bin430', '2020', '04', '5000');
insert into bill values ('sogang', '2020', '01', '5000');
insert into bill values ('database', '2020', '06', '7000');
insert into bill values ('bin430', '2019', '06', '5000');
insert into bill values ('sogang', '2019', '06', '5000');
insert into bill values ('database', '2019', '06', '7000');
insert into bill values ('bin430', '2018', '04', '5000');
insert into bill values ('sogang', '2018', '12', '5000');
insert into bill values ('database', '2018', '06', '7000');
```

(Type III)

```
** Find the customer who has spent the most money on shipping in the past certain year **
Which Year? : 2019
Customer ID : univ
```

bill 에서, 2019년의 내역을 보면, 'univ' 가 7500원, 'abc123'이 4500원, 'bin430'이 5000원, 'sogang'이 5000원, 'database'가 7000원으로 'univ'가 가장 많은 돈을 지출했음을 알 수 있다.

(Type V)



bill_bin430_2020_4(1).txt - 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

<SIMIPLE BILL>

| ID | NAME | ADDRESS | OWED |

| bin430 | Soobin Kim | Seoul, South Korea | 5000 |

bill_bin430_2020_4(2).txt - 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

<TYPE SPECIFIED BILL>

| ID | TYPE | TOTAL PRICE | WEIGHT | TIMELINESS | ADDRESS |

| bin430 | flat | Frequent Client(Contract) | 500.50 | overnight | Seoul, South Korea |

| bin430 | larger | Frequent Client(Contract) | 200.50 | second day | Seoul, South Korea |

bill_bin430_2020_4(3).txt - 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

<ITEMIZED BILL>

| ID | TYPE | TOTAL PRICE | TIMELINESS | ADDRESS | NAME | PHONE NUMBER |

| ABC10000 | bin430 | 2020 | 4 | 20 | 15.00 | Frequent Client(Contract) | flat | 500.50 | overnight | 202004212359 | None | Seoul, South Korea | Soobin Kim | 01012345678 |

| DEF14000 | bin430 | 2020 | 4 | 12 | 10.09 | Frequent Client(Contract) | larger | 200.50 | second day | 202004142359 | None | Seoul, South Korea | Soobin Kim | 01012345678 |

텍스트 파일로 각각 :

bill_(*CUSTOMER_ID*)_(*YEAR*)_(*MONTH*)(1).txt,

bill_(*CUSTOMER_ID*)_(*YEAR*)_(*MONTH*)(2).txt,

bill_(*CUSTOMER_ID*)_(*YEAR*)_(*MONTH*)(3).txt

에 해당하는 고객의 bill이 생성되었음을 확인할 수 있다.

3. Query Explanation

* input으로부터 받아서 처리되는 값은 대문자 기울임으로 표현하였다.

① Type I

▷ I-1.
```
"select customer_id
      from timetable, package
      where transportation = 'truck'
            and transp_id = (TRUCK NUMBER)
            and arrive = 0
            and timetable.package_id = package.package_id"
```

사고가 난 Truck ID (명세서에 의해 1721로 고정한다) 가 들어올 때까지 입력을 받은 후, 해당 ID를 이용하여 query를 생성한다. *timetable*에 각 package가 어떤 운송 수단을 이용하였는지에 대한 정보가 저장되어 있고, *package*에 해당 택배를 누가 보냈는지에 대한 정보가 저장되어 있다. 이 때, customer_id 는 unique한 값인 반면에 고객의 이름인 name은 동명이인의 발생으로 인해 서로 다른 tuple에 중복되는 값이 존재할 수 있다. 그렇기 때문에 명확한 구분을 위하여 처음부터 *package*에 name 없이 customer_id 만을 저장하였다. 따라서 해당 query 에서도 사고가 난 교통편에 누가 발송한 택배가 들어있었는지에 대한 정보를 반환하는 과정에서, customer_id 를 보이도록 설계하였다. 또한 truck은 plane 과 warehouse 를 거쳐 마지막 운송 수단이므로, 최종 목적지로 택배를 이동하는 과정에 사용된다. 그렇기 때문에 해당 truck을 통해 이동했는데(transportation = 'truck' and transp_id = 1721) 도착 정보가 없다면 (arrive = 0) 해당 택배는 사고가 난 truck에 실려 있었다는 것으로 이해할 수 있다.

▷ I-2. "select name
from timetable, package
where transportation = 'truck'
and transp_id = *(TRUCK NUMBER)*
and arrive = 0
and timetable.package_id = package.package_id"

I-1과 동일한 방식으로 데이터를 가져올 테이블 (from ~) 을 생성하지만, 이번에는 누가 택배를 받지 못했는가에 대한 정보를 반환해야 하기 때문에 customer_id 가 아니라 name을 반환한다. *package* 에는 customer_id 와 name 이 모두 column으로 존재하는데, 이 때 주의할 점은 customer_id 는 택배를 보내는, 즉 택배 서비스를 신청한 고객의 회원 id이고, name 은 택배를 받는 사람, 즉 수취인의 이름이다. 따라서 customer_id 에 해당하는 고객의 이름이 꼭 name 의 value로 저장되어 있지 않을 수도 있다.

▷ I-3. "select package_id
from timetable
where transportation = 'truck'
and transp_id = *(TRUCK NUMBER)* and arrive = 1
and datetime = (select max(datetime)
from timetable
where transportation = 'truck'
and transp_id = *(TRUCK NUMBER)*
and arrive = 1)"

사고가 난 교통편이 가장 최근에 배송에 성공한 package를 찾기 위해서는, 먼저 해당 교통편(Truck 1721)이 배송에 성공한 모든 package의 정보가 담긴 table을 생성해야 한다. subquery의 from, where clause를 이용해서 이를 생성하고, 이렇게 만들어진 table에서 가장 최근의 배송건을 찾기 위해서 max(datetime)을 select clause를 통해 반환한다. 그리고 *timetable*에서 반환된 datetime과 일치하는 tuple을 찾아 해당 tuple의 package_id를 반환하면 해당 package_id 가 사고가 난 교통편이 가장 최근에 배송 완료한 package의 id가 된다.

② Type II

▷ "with sent as (select customer_id, count(customer_id) as each_sent
from history
where year = *(YEAR)*
group by customer_id)
select customer_id
from sent
where each_sent = (select max(each_sent) from sent)"

특정 연도에 가장 많이 택배를 보낸 고객을 찾기 위해서는, 먼저 해당 연도에 각 고객이 몇 개의 택배를 발송했는지에 대한 정보가 담긴 table을 생성해야 한다. 따라서 이를 만족하는 *sent* 라는 table을 생성하고, *sent* 내에 각 고객이 발송한 택배의 건수를 each_sent 라는 column으로 저장한다. *sent* 를 생성하고 나면, each_sent 값이 최대인 tuple의 customer_id 를 반환하면 된다.

③ Type III

▷
```
"select customer_id
 from bill
 where year = (YEAR) and owed = (select max(owed)
                                 from bill where year = (YEAR))"
```

특정 연도에 가장 돈을 많이 지출한 고객을 찾기 위해서는, *bill* 을 이용하면 된다. *bill* 에는 frequent customer (*contract* 에 정보가 존재하는 고객) 의 경우, owed 의 value 로 무조건 *contract* 의 pay_amount 값이 들어가게 된다. 따라서 *bill* table 하나만으로 해당 조건에 부합하는 데이터를 도출할 수 있다.

④ Type IV

▷
```
"with timetable_done as (select package_id, datetime
                         from timetable where arrive = 1)
select package_id
from package natural join timetable_done
where promissed < datetime"
```

먼저 *timetable_done* 이라는 table에 *timetable* 의 tuple들 중 최종 목적지로 향하는 마지막 이동에 해당하는 tuple, 즉 arrive 값이 1인 tuple만 골라 저장한다. 그리고 *timetable_done* 과 package 를 통해, 도착해야 하는 시간의 마지노선인 promised 와 실제로 도착한 시간인 datetime을 비교해, datetime이 promised보다 늦은 시간인 경우의 package_id 를 반환한다.

⑤ Type V

▷ Simple Bill :
```
"select customer_id, name, address, owed
 from bill natural join customer
 where customer_id = '(ID)' and year = (YEAR)
       and month = (MONTH)"
```

*bill*에는 customer_id, owed 만 저장되어 있고, name 과 address는 저장되어 있지 않으므로, *customer*와의 natural join을 통해 생성한 table로부터 input으로 받은 연도와 달, customer id에 해당하는 tuple들을 골라낸다. 여기에서도 Type I과 마찬가지로, 고객의 '이름' 대신 unique 한 고객의 'ID'를 이용하였다.

▷ Type specified Bill :
```
"select package.customer_id, package_type,
        sum(price) as total_price, weight,
        timeliness, address
 from package natural join history
 where year = (YEAR) and month = (MONTH)
       and customer_id = '(ID)'
 group by customer_id, package_type"
```

package 와 *history* 의 natural join 을 통해 생성한 table에는 각 package를 보낸 고객 정보와 해당 packet의 정보가 모두 나오게 된다. 여기에서 input으로 받은 연도와 달, customer id 에 해당하는 tuple들에 대하여 customer_id 와 package_type 으로 group by clause 를 적용하여 해당 고객의 package bill 정보를 type 별로 보여준다.

▷ Itemized Bill : "select *
from history natural join package
where customer_id = '*(ID)*' and year = *(YEAR)*
and month = *(MONTH)*"

*package*와 *history*를 natural join 하여 생성한 table에는 각 package에 대한 정보와 해당 package를 어떤 고객이, 언제 보냈는지에 대한 정보가 함께 들어있으므로 해당 table에서 input으로 들어온 값들에 부합하는 tuple들에 대하여 보든 column을 반환한다.

4. Code Explanation

* 소스코드 '20170364.c'의 주석으로 대체한다.