HW2 REPORT

HARD COPY

20211584 장준영

1. HW1의 관계형 모델과의 차이점

- 기본 구현

char(n) type으로 설정된 attribute들을 모두 varchar(n) type으로 바꾸었다. 모두 가변적인 길이의 문자열을 값으로 갖기 때문이다. 예를 들어, type의 box attribute는 'envelope', 'large box', 'small box' 등의 값을 갖는데, 전부 문자열의 길이가 다르고 고정되어있지도 않다.
transportation에서 출발 시각과 도착 시각을 관리하기 위해 date(년, 월 일)과 time(시, 분, 초) attribute를 따로 관리했었는데, 이를 date로 통합하고 type을 datetime으로 바꾸었다. datetime은 년, 월, 일, 시, 분, 초를 모두 저장하는 자료형으로, 좀 더 간결하게 데이터를 관리할 수 있다.
package의 received 속성이 불필요하다고 판단하였다. 따라서, 이 속성을 삭제하였다. 또, transportation의 속성 중 transportation으로 이름이 겹치는 것이 존재하는데, 헷갈림을 방지하기 위해 속성의 이름을 trans로 바꾸었다.

-BCNF Decomposition

customer -> customer / house
R={ID, name, zip_code, address} F={(1)ID->name, zip_code, address, (2)address->zip_code} (※zip_code는 지도를 조그만 구역으로 나누고, 그 구역별로 부여하는 번호이므로 address를 통해 zip_code를 구별할 수 있다.) (1) Trivial dependency (primary key) (2) address+={address, zip_code} (R을 포함하지 않음) {address, zip_code} AND {ID, name, zip_code}={zip_code} (공집합이 아님) 따라서 (2) dependency는 BCNF를 위배한다. →R1={address, zip_code}, R2={address, ID, name}
contract_info -> contract_info / account
R={ID, account_num, depositor_name} F={(1)ID->account_num, depositor_name, (2)account_num->depositor_name} (※하나의 account_num에 두 개 이상의 depositor_name이 생길 수 없다고 보았다. 하지만, 한 명의 depositor가 여러 개의 account_num을 가질 수는 있을 것이다.) (1) Trivial dependency (primary key) (2) account_num+={account_num, depositor_name} (R을 포함하지 않음) {account_num, depositor_name} AND {ID, depositor_name}={depositor_name} (공집

합이 아님) 따라서 (2) dependency는 BCNF를 위배한다. →R1={account_num, depositor_name}, R2={account_num, ID}
type, package 수정
R={box, weight, timeliness} F={(1)box, weight, timeliness->box, weight, timeliness, (2)weight->box} (※가정 하에선 weight에 따라 box를 정한다. 따라서 weight로 box를 전부 구별할 수 있다.) (1) Trivial dependency (primary key) (2) weight+={weight, box} (R을 포함하지 않음) {weight, box} AND {box, timeliness}={box} (공집합이 아님) 따라서 (2) dependency는 BCNF를 위배한다. →R1={weight, box}, R2={weight, timeliness} 하지만, timeliness를 따로 분리해내기 위해 R2를 만들면 불필요한 Entity가 많아질 것으로 판단하였다. 따라서, timeliness 속성은 package entity의 attribute로 합병하고, type entity에는 weight와 box만 넣도록 수정하였다. 동시에 package도 기존에 foreign key로 받아오던 box를 받아오지 않게 되었다.
transportation -> transportation, trans_type
R={trans_id, trans, start_date, end_date} F={(1)trans_id, trans, start_date->end_date, (2)trans_id->trans} (※trans_id를 통해 trans를 완전히 구별할 수 있다.) (1) Trivial dependency (primary key) (2) trans_id+={trans_id, trans} (R을 포함하지 않음) {trans_id, trans} AND {trans, start_date, end_date}={trans} (공집합이 아님) 따라서 (2) dependency는 BCNF를 위배한다. →R1={trans_id, trans}, R2={trans_id, start_date, end_date}
이외 나머지의 entity는 trivial dependency만 갖기 때문에 BCNF 상태이다.

- BCNF Decomposed Logical Schema Diagram

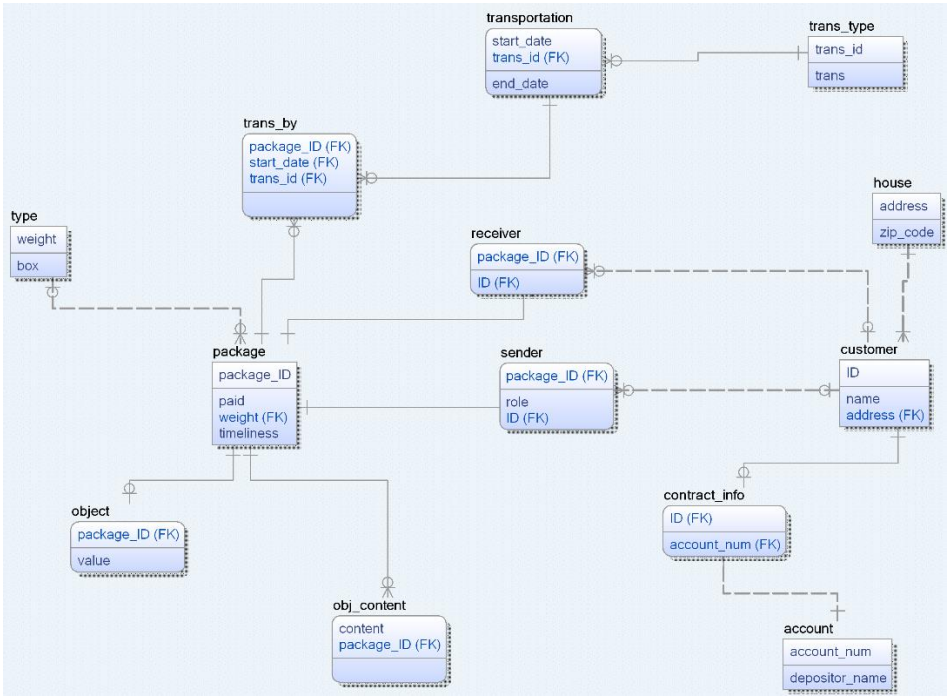
BCNF Decomposition을 통해 새로 생성된 entity는 trans_type, house, account이고, 수정된 entity는 type, package이다. 각각 entity의 attribute는 위의 decomposition 결과대로 설정되었다.

(1) house – customer : 하나의 주소에 여러 고객이 거주할 수 있다.

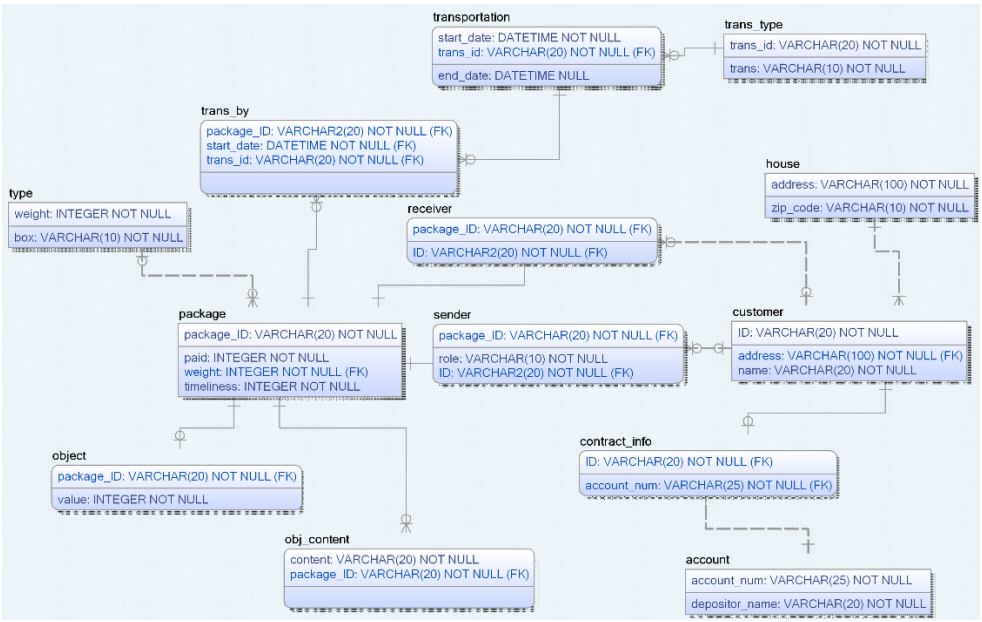
(2) trans_type – transportation : 운송 수단이 이용되지 않을 수도, 한 번 이용될 수도, 여러 번 이용될 수도 있다.

(3) account – contract_info : 하나의 계약 당 하나의 계좌번호만 사용하도록 설정하였다.

(4) type – package : type이 package에 할당 되지 않을 수도, 하나에 할당 될 수도, 여러 개에 할당 될 수도 있다.



2. Physical Schema Diagram



(1) customer

- ID : primary key이다. 최대 20자리 가변적 문자열이다. primary key이므로 NOT NULL이다.

- address : house를 참조한 foreign key이다. 최대 100자리 가변적 문자열이다. 보내는 사람이나 받는 사람 모두 주소가 꼭 필요하므로 NOT NULL로 설정하였다.

- name : 최대 20자리 가변적 문자열이다. 보내는 사람이나 받는 사람 모두 이름이 필요하므로 NOT NULL로 설정하였다.

(2) house

- zip_code : 최대 10자리 가변적 문자열이다. 택배를 보낼 때 우편번호가 필수이므로 NOT NULL로 설정하였다.

(3) account

- account_num : primary key이다(NOT NULL). 최대 25자리 가변적 문자열이다.

- depositor_name : 최대 20자리 가변적 문자열이다. 입금자명 없이 입금할 수 없으므로 NOT NULL로 설정하였다.

(4) trans_type

- trans_id : primary key이다(NOT NULL). 최대 20자리 가변적 문자열이다.

- trans : 최대 10자리 가변적 문자열이다. 운송 수단 정보를 저장하기 위한 entity인데 운송 수단이 없을 수 없으므로 NOT NULL로 설정하였다.

(5) transportation

- start_date : primary key이다(NOT NULL). 날짜와 시간 정보를 저장한다.

- end_date : 출발은 했지만 도착하지 않았을 수 있으므로, NULL을 허용한다. 날짜와 시간 정보를 저장한다.

(6) type

- weight : primary key이다(NOT NULL). 정수형 자료이다.

- box : 최대 10자리 가변적 문자열이다. primary key인 weight가 정해지는 순간 box의 종류도 정해질 수 있기 때문에 NOT NULL로 설정하였다.

(7) package

- package_ID : primary key이다(NOT NULL). 최대 20자리 가변적 문자열이다.

- paid : 정수형 자료이다. 선금을 받거나 받지 않는 경우 모두 정수로 값을 표현할 수 있기 때문에 NOT NULL로 설정하였다.

- timeliness : 정수형 자료이다. 택배 발송을 준비하면서 비용을 설정하는데 필수적이기 때문에 NOT NULL로 설정하였다.

(8) sender

- role : 최대 10자리 가변적 문자열이다. sender는 customer, company, prepaid 셋 중 무조건 하나의 역할을 맡고 있기 때문에 NOT NULL로 설정하였다.

(9) object

- value : 정수형 자료이다. 종속적인 entity의 속성이기도 하고, 통관이나 위험 물질 발송 등 value가 필수적인 경우에만 사용하기 때문에 NOT NULL로 설정하였다.

(10) obj_content

- content : primary key이다(NOT NULL). 최대 20자리 가변적 문자열이다.

3. Queries / C code implementation(CRUD, ODBC)

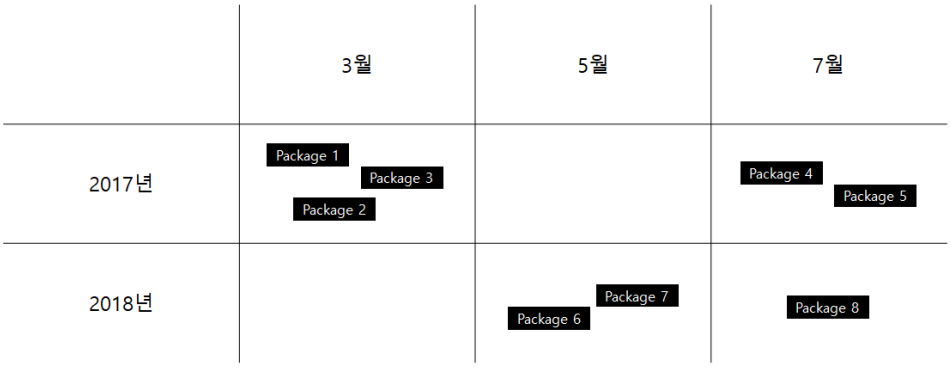
- ODBC를 이용한 schema building

DB를 세우기 위해 가장 먼저 table을 만들어야 한다. table 생성, 값 추가 등 CRUD query 단계는 C 프로세스에서 txt 파일을 읽어와 진행한다.

```
create table transportation (
	trans_ID	varchar(20),
	start_date	datetime not null,
	end_date	datetime,
	primary key (trans_ID, start_date),
	check (start_date<=end_date)
);
create table trans_type (
	trans_ID	varchar(20),
	trans	varchar(10) not null,
	primary key (trans_ID),
	check (trans in ('warehouse', 'airplane', 'truck'))
);
```

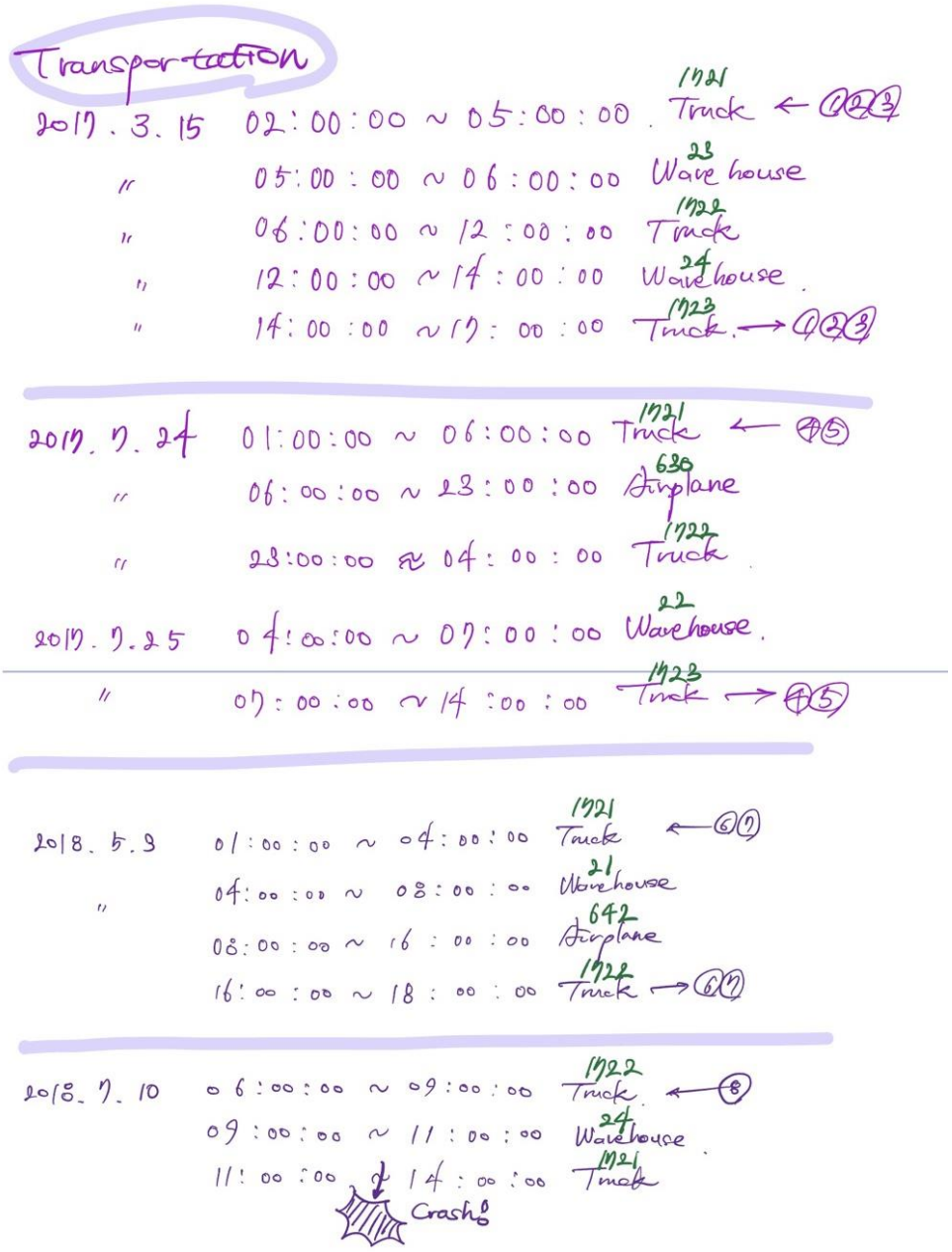
table 생성 query 중 두 가지를 예시로 가져왔다. 우선 table의 이름, attribute의 이름과 type을 선언한다. 이후 null/not null을 구분하여 type 뒤에 달아주고, primary key나 foreign key를 작성해준다. check는 constraints를 위한 것으로, transportation에서는 start_date가 end_date보다 작거나 같아야 하고, trans_type에서는 trans가 세 가지 중 하나여야 하는 등의 제약 조건이 필요하다. 앞서 설계한 모델의 모든 entity에 대해 위와 같은 과정을 수행하면, table 생성을 마치게 된다.

이후 table에 값을 추가해야 한다. 다음과 같은 계획으로 값을 추가하였다.



고객 A	1
고객 B	2, 6
고객 C	3, 4, 5, 7, 8

2017년 3월, 5월, 7월, 2018년 3월, 5월, 7월에 각각 사진과 같이 package가 배송된다. 고객 A는 package 1을, B는 2, 6을, C는 3, 4, 5, 7, 8을 보낸다. package의 특성과 받는 사람은 임의로 설정된다.



transportation 과정을 그린 사진이다. Type 1 query를 용이하게 진행하기 위해, 2018년 7월 10일 11시에 출발하는 1721번 truck에서 crash가 있다고 가정한다.

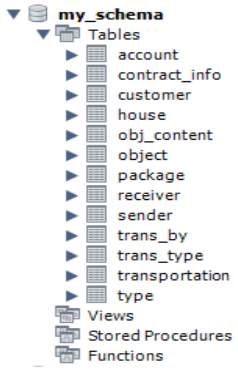
Examples of insert queries
delete from account; #deletion before insertion to prevent collision insert into house values ('35, Baekbeom-ro, Mapo-gu, Seoul, Republic of Korea','04107');

```
insert into customer values ('1023','35, Baekbeom-ro, Mapo-gu, Seoul, Republic of Korea','Junyeong');
insert into type values (1, 'envelope');
insert into package values ('12591', 2000, 1, 48);
insert into sender values ('12591', '1023', 'customer');
insert into receiver values ('12591', '0131');
insert into trans_type values ('1721', 'truck');
insert into transportation values ('1721', '2017-03-15 02:00:00', '2017-03-15 05:00:00');
insert into trans_by values ('12598', '1721', '2018-07-10 11:00:00');
...
```

위와 같이 create, delete, insert query로 이루어진 txt 파일을 작성하고 나면, C 코드에서 파일을 읽어들여 query를 실행해야 한다. 이를 위해 C/C++ ODBC API를 사용한다.

```
const char* db = "my_schema";

FILE* crud_txt;
char query[5000];
crud_txt = fopen("queries.txt", "r");
printf("Building schemas...₩n");
while (!feof(crud_txt)) {
        fgets(query, 4999, crud_txt);
        state = mysql_query(connection, query);
}
printf(">> Building schemas complete.₩n₩n₩n");
```



CRUD query가 작성된 queries.txt 파일을 불러온다. 이후 EOF를 만날 때까지 파일을 한 줄씩 읽어 들여, mysql_query를 통해 query를 진행한다. 실행이 완료되면, MySQL Workbench의 schema에 성공적으로 적용된다. 이후 원하는 값을 확인하기 위한 query를 실행할 때도 동일한 방식으로 진행한다. 아래는 Type 4의 query를 처리하기 위한 과정이다.

```
case 4:
        printf("₩n---- TYPE 4 ----₩n₩n**Find those packages that were not delivered whthin the promised time in certain year**₩n₩nWhich Year? : ");
        scanf("%d", &temp);
        sprintf(type_query, <query>, temp, temp + 1);
        strcpy(query, type_query);
        strcpy(type_query, "");
        printf("₩n");
        state = mysql_query(connection, query);
        if (state == 0) {
                sql_result = mysql_store_result(connection);
                while ((sql_row = mysql_fetch_row(sql_result)) != NULL) {
                        printf("package_ID : %s, latency : %s₩n", sql_row[0], sql_row[1]);
                }
        }
```

```
        break;
```

사용자가 입력하는 값을 <query>에 적용하여 온전한 query를 만들고, 이를 mysql_query로 실행한다. 실행이 성공적이면 0이 return된다. mysql_store_result를 통해 결과 집합을 sql_result에 저장하고, mysql_fetch_row를 통해 한 row씩 sql_row에 저장하고 포인터를 옮긴다. 이런 과정을 통해, C/C++에서 MySQL의 데이터 처리를 할 수 있다.

- Select Queries (Type 1~5)

(1) Type 1

2018년 7월 10일 11시에 출발하는 1721번 truck에서 crash가 있다고 가정하였다.

1-1

```
select distinct ID, name
from (
        select *
        from trans_by natural join transportation
        where trans_ID='1721' and start_date='2018-07-10 11:00:00'
        ) as trans_info join sender as S on S.package_id=trans_info.package_id join customer using(id)
```

	ID	name
▶	0602	Yurim

```
**Find all customers(senders) who had a package on the truck at the time of the crash**
Sender ID : 0602, Sender name : Yurim
```

1-2

```
select distinct ID, name
from (
        select *
        from trans_by natural join transportation
        where trans_ID='1721' and start_date='2018-07-10 11:00:00'
        ) as trans_info join receiver as R on R.package_id=trans_info.package_id join customer using(id)
```

	ID	name
▶	0131	Sujin

```
**Find all recipients(receivers) who had a package on the truck at the time of the crash**
Receiver ID : 0131, Receiver name : Sujin
```

1-3

```
select ID, name, package_ID
from trans_by join (
        select trans_ID, max(start_date) as max
        from trans_by natural join transportation
        where trans_ID='1721' and start_date<'2018-07-10 11:00:00'
        ) as max_info on max_info.max=trans_by.start_date join sender as s using (package_id) join customer as c using (id)
```

ID	name	package_ID
0131	Sujin	12596
0602	Yurim	12597

```
**Fine the last successful delivery by that truck prior to the crash**

Package ID : 0131, Sender ID : Sujin, Sender name : 12596
Package ID : 0602, Sender ID : Yurim, Sender name : 12597
```

(2) Type 2

Type 2부터는 년도나 달 등을 사용자에게 입력받는다. 여기선 Year 입력이 2018이라고 가정하였다.

```
select ID, name
from customer natural join (
        select ID, count(package_ID) as counting
        from sender natural join (
                select package_id, min(start_date) as package_start_date
                from trans_by
                group by package_id
                having ('2018-01-01 00:00:00'<=package_start_date and '2019-01-01 00:00:00'>package_start_date) #Year : 2018
                ) as package_info
        group by ID
    order by counting asc
    ) as year_info
```

결과는 다음과 같다. 입력받은 년도에 많은 package를 배송한 순서대로 고객의 ID, 이름

을 확인할 수 있다.

	ID	name
▶	0131	Sujin
	0602	Yurim

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

ID : 0131, name : Sujin
```

프로그램을 통해서는 가장 많이 배송한 한 명만 출력된다.

(3) Type 3

```
select ID, name, package_cost
from customer natural join (
        select ID, sum((3000+500*weight+500*48*(1/timeliness))) as package_cost
        from package natural join(
                select ID, package_ID
                from sender natural join (
                        select package_id, min(start_date) as package_start_date
                        from trans_by
                        group by package_id
                        having ('2018-01-01 00:00:00'<=package_start_date and '2019-
01-01 00:00:00'>package_start_date) #Year : 2018
                        ) as package_info
                ) as year_info
        group by ID
        ) as info
order by package_cost desc
```

결과는 다음과 같다. 입력받은 년도에 돈을 사용한 순서대로 고객의 ID, 이름, 사용한 돈을 확인할 수 있다. (※이 query에서는 float을 int로 casting하지 않고, C 코드 내에서 casting 했다. Type 5의 query에서는 cast()를 이용한 casting을 진행하였다.)

	ID	name	package_cost
▶	0602	Yurim	20500.0000
	0131	Sujin	3833.3333

```
**Find the customer who has spent the most money on shipping in certain year**
Which Year? : 2018

ID : 0602, name : Yurim, money spent : 20500
```

프로그램을 통해서는 가장 돈을 많이 사용한 한 명만 출력된다.

(4) Type 4

```
select package_ID, timediff(duration,timeliness) as latency
from(
        select package_ID, timediff(end,start) as duration, sec_to_time(3600*timeliness)
as timeliness
        from (select package_ID, min(start_date) as start, max(end_date) as end
                from trans_by natural join transportation
                group by package_ID
         having ('2017-01-01 00:00:00'<=start and '2018-01-01 00:00:00'>start)
                ) as package_info natural join package #Year : 2017
        ) as info
where duration>timeliness
```

결과는 다음과 같다. 입력받은 년도에 예정된 시간에 도착하지 못한 package의 ID와, 얼마나 늦었는지 latency를 확인할 수 있다.

	package_ID	latency
▶	12592	03:00:00

```
**Find those packages that were not delivered whthin the promised time in certain year**
Which Year? : 2017

package_ID : 12592, latency : 03:00:00
```

(5) Type 5

```
select S.id as sender_id, C1.name as sender_name, role, R.id as receiver_id, C2.name as
receiver_name, package_id, paid, weight, timeliness,
cast(3000+500*weight+500*48*(1/timeliness) as signed) as package_cost,
(cast(3000+500*weight+500*48*(1/timeliness) as signed)-paid) as owed, start
from package natural join (
        select package_ID, min(start_date) as start, max(end_date) as end
```

```
        from trans_by natural join transportation
        group by package_ID
        having ('2017-03-01 00:00:00'<=start and '2017-04-01 00:00:00'>start) #Year : 2017, Month : 3
        ) as package_info join sender as S using (package_ID) join receiver as R using (package_ID)  join customer as C1 on C1.ID=S.ID join customer as C2 on C2.ID=R.ID
```

결과는 다음과 같다. 입력받은 년도와 달에 배송된 package에 대한 모든 정보를 볼 수 있다.

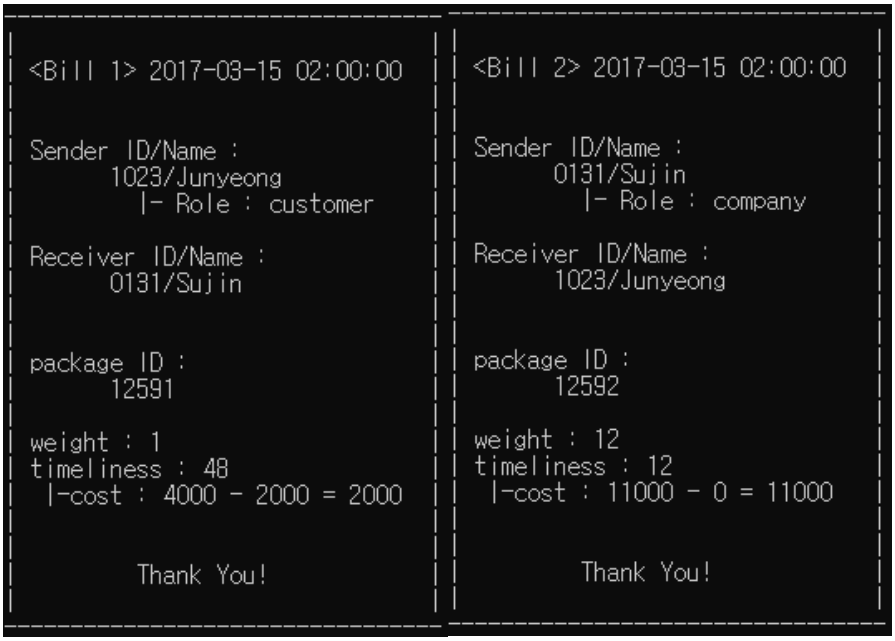
sender_id	sender_name	role	receiver_id	receiver_name	package_id	paid	weight	timeliness	package_cost	owed	start
1023	Junyeong	customer	0131	Sujin	12591	2000	1	48	4000	2000	2017-03-15 02:00:00
0131	Sujin	company	1023	Junyeong	12592	0	12	12	11000	11000	2017-03-15 02:00:00
0602	Yurim	customer	1023	Junyeong	12593	0	6	24	7000	7000	2017-03-15 02:00:00

이를 통해, 세부적인 정보를 담은 Bill을 출력한다.

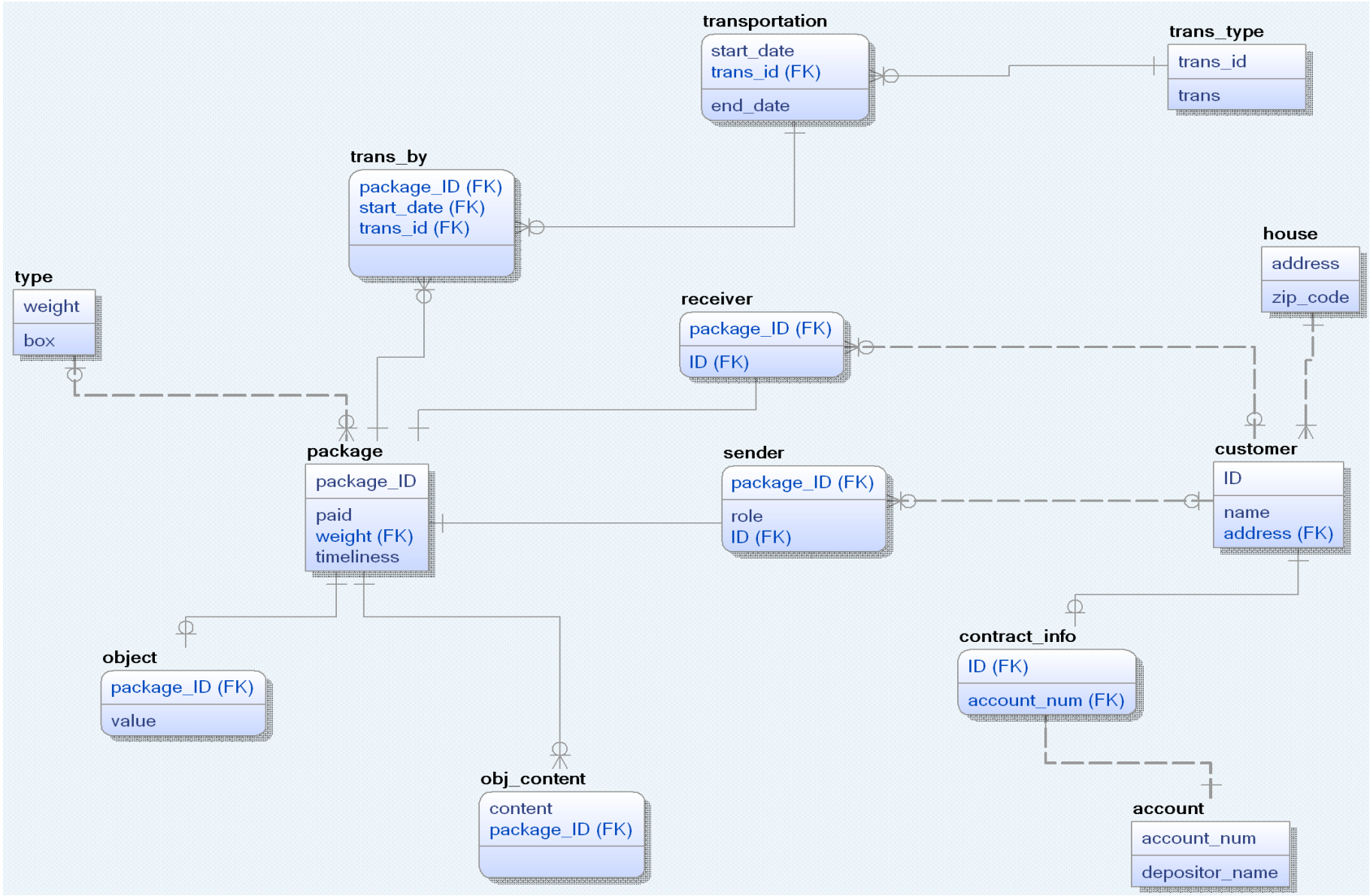
```
state = mysql_query(connection, query);
if (state == 0)
{
        sql_result = mysql_store_result(connection);
        int i = 1;
        while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
        {
                printf("₩n---------------------------------₩n");
                printf("|₩t₩t₩t₩t|₩n");
                printf("| <Bill %d> %s₩t|₩n", i++, sql_row[11]);
                printf("|₩t₩t₩t₩t|₩n");
                printf("|₩t₩t₩t₩t|₩n");
                printf("| Sender ID/Name : ₩t₩t|₩n");
                printf("|₩t%s/%s₩t₩t|₩n", sql_row[0], sql_row[1]);
                printf("|₩t  |- Role : %s₩t|₩n",sql_row[2]);
                printf("|₩t₩t₩t₩t|₩n");
                printf("| Receiver ID/Name : ₩t₩t|₩n");
                printf("|₩t%s/%s₩t₩t|₩n", sql_row[3], sql_row[4]);
                printf("|₩t₩t₩t₩t|₩n");
                printf("|₩t₩t₩t₩t|₩n");
                printf("| package ID : ₩t₩t₩t|₩n");
                printf("|₩t%s₩t₩t₩t|₩n", sql_row[5]);
                printf("|₩t₩t₩t₩t|₩n");
                printf("| weight : %s₩t₩t₩t|₩n",sql_row[7]);
                printf("| timeliness : %s₩t₩t|₩n", sql_row[8]);
                printf("|  |-cost : %s - %s = %s₩t|₩n", sql_row[9], sql_row[6],
sql_row[10]);
                printf("|₩t₩t₩t₩t|₩n");
                printf("|₩t₩t₩t₩t|₩n");
                printf("|₩t  Thank You!   ₩t₩t|₩n");
                printf("|₩t₩t₩t₩t|₩n");
                printf("---------------------------------₩n");
        }
        i = 1;
        mysql_free_result(sql_result);
```

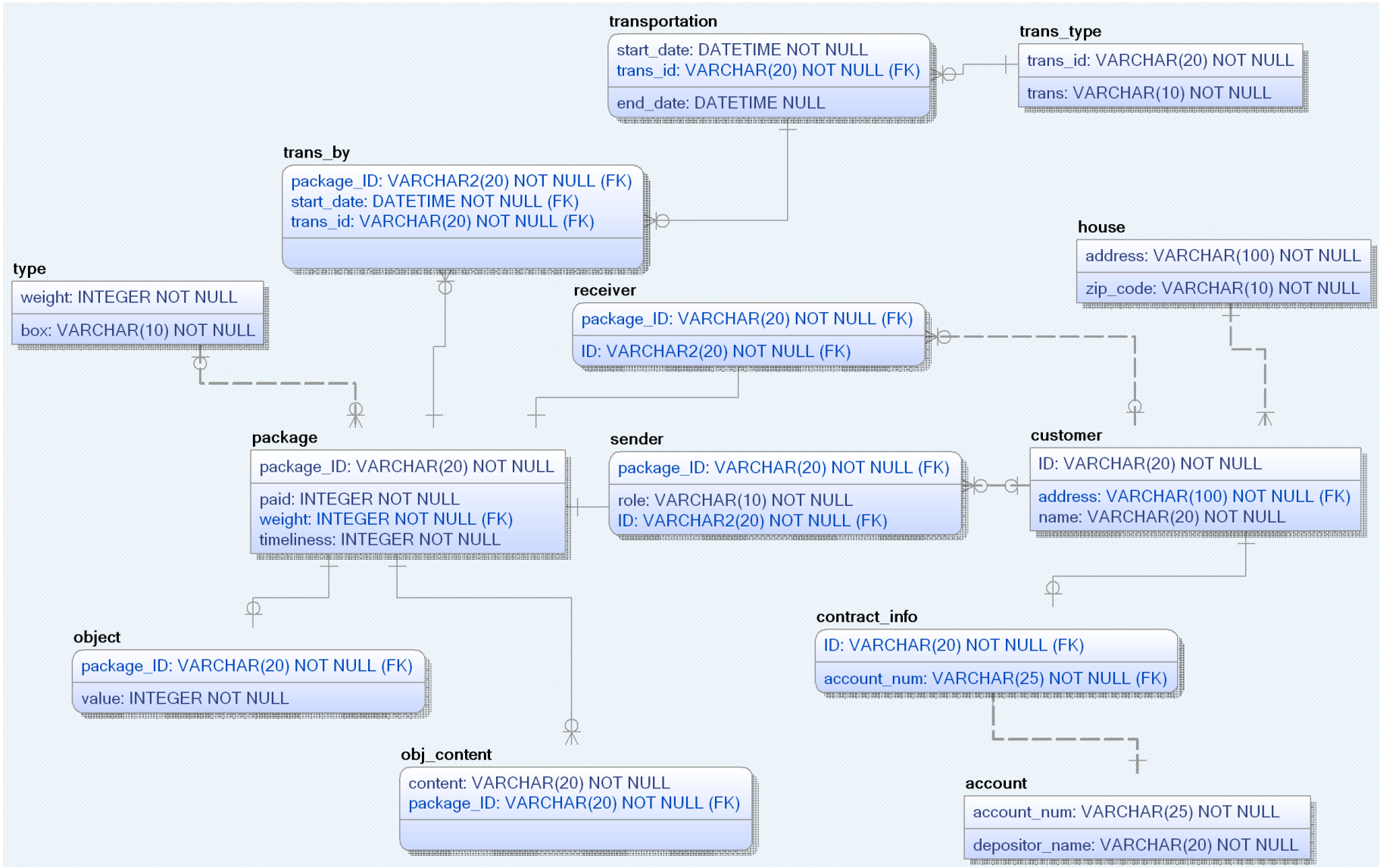
```
}
```

출력은 다음과 같다.



▶ **모든 query가 처음 설계한대로 올바른 값을 출력하고 있다.**


transportation
start_date
trans_id (FK)
end_date
trans_type
trans_id
trans
trans_by
package_ID (FK)
start_date (FK)
trans_id (FK)
house
address
zip_code
type
weight
box
receiver
package_ID (FK)
ID (FK)
package
package_ID
paid
weight (FK)
timeliness
sender
package_ID (FK)
role
ID (FK)
customer
ID
name
address (FK)
object
package_ID (FK)
value
contract_info
ID (FK)
account_num (FK)
obj_content
content
package_ID (FK)
account
account_num
depositor_name


transportation
start_date: DATETIME NOT NULL
trans_id: VARCHAR(20) NOT NULL (FK)
end_date: DATETIME NULL
trans_type
trans_id: VARCHAR(20) NOT NULL
trans: VARCHAR(10) NOT NULL
trans_by
package_ID: VARCHAR2(20) NOT NULL (FK)
start_date: DATETIME NOT NULL (FK)
trans_id: VARCHAR(20) NOT NULL (FK)
house
address: VARCHAR(100) NOT NULL
zip_code: VARCHAR(10) NOT NULL
type
weight: INTEGER NOT NULL
box: VARCHAR(10) NOT NULL
receiver
package_ID: VARCHAR(20) NOT NULL (FK)
ID: VARCHAR2(20) NOT NULL (FK)
package
package_ID: VARCHAR(20) NOT NULL
paid: INTEGER NOT NULL
weight: INTEGER NOT NULL (FK)
timeliness: INTEGER NOT NULL
sender
package_ID: VARCHAR(20) NOT NULL (FK)
role: VARCHAR(10) NOT NULL
ID: VARCHAR2(20) NOT NULL (FK)
customer
ID: VARCHAR(20) NOT NULL
address: VARCHAR(100) NOT NULL (FK)
name: VARCHAR(20) NOT NULL
object
package_ID: VARCHAR(20) NOT NULL (FK)
value: INTEGER NOT NULL
contract_info
ID: VARCHAR(20) NOT NULL (FK)
account_num: VARCHAR(25) NOT NULL (FK)
obj_content
content: VARCHAR(20) NOT NULL
package_ID: VARCHAR(20) NOT NULL (FK)
account
account_num: VARCHAR(25) NOT NULL
depositor_name: VARCHAR(20) NOT NULL