데이터베이스 미니프로젝트3 최종(2차) 제출 보고서

18012248 신민기

0. 구현 정도

기능	구현 단계
Schema Check	모두 구현
Select Query	모두 구현
Insert Query	모두 구현
Update Query	모두 구현
Delete Query	모두 구현
Custom Query	시나리오 1개: 사용자 임의 쿼리 시나리오 4개: 동적 입력값 쿼리 실행
Join	- 2개 table join Select: 3개 / Update: 2개 / Delete: 3개 Custom: 2개 - 3개 table join Select: 1개 / Update: 1개

1. 최종 요구사항 분석

customer

attribute: pk인 id와 이름(name), 주소(address), 전화번호(phone_no), 이메일(email)로 구성한다. Customer의 pk는 package에 의해 참조된다.

package와 1:N 관계를 갖는다. Customer 1명을 선택했을 때, 대응되는 package 수는 0~N 개의

범위를 갖는다.

한 customer의 package의 status가 1에 도달하면, package table의 튜플은 삭제 가능하다. Customer에 대응되는 package가 0일 경우, customer가 수령할 택배가 없음을 의미한다.

Package

- attribute: 고객 ID(customer_id), 택배 번호(pack_no), 무게(weight), 배송 상태(status)로 구성한다. Customer의 pk값을 참조하며 partial key인 pack_no로 식별자를 구성한다.

- customer와 N:1 관계를 갖는다. Package 1개를 선택했을 때, 대응되는 customer 수는 1~1 범위를 가지며 전체 참여다.

- driver와 N:1 관계를 갖는다. Package 1개를 선택했을 때, 대응되는 driver 수는 마찬가지로 1~1 범위를 가지며 전체 참여다.

- 배송 상태는 0과 1로 표현한다. Table 내에 튜플이 생성되면 status는 0으로 시작한다. 한 택배는 1개의 driver를 배정받는다. Status가 1에 도달하는 순간 customer에게 배송이 완료되었다 판단하고 튜플을 삭제한다.

Driver

- attribute: pk인 id와 이름(name), 전화번호(phone_no), 배송할 택배 개수(packages), 월 누적 택배 개수(total_packs), car table 참조키(cno)로 구성한다.

- package와 1:N 관계를 갖는다. Driver 1명을 선택했을 때, 대응되는 package 수는 0~N 범위를 갖는다.

- Car, Branch와 degree가 3인 relation을 갖는다. Driver 1명을 선택했을 때, 대응되는 Car의 수는 1~1 범위를 갖는다. 대응되는 Branch 수는 1~1 범위를 갖는다. Driver는 소속 지점과 배송 수단이 무조건 1개가 mapping 되어야 하는 전체 참여 관계다.

- Driver는 package와 1:N 관계를 맺는다. 다수의 package를 책임질 수 있다. Package와 매핑되면 packages라는 attribute를 1씩 올린다. Packages attribute는 Car 엔티티의 capacity의 영향을 받는다. Driver는 사전에 매핑된 Car 인스턴스가 있다. Car 인스턴스의 capacity가 20이라면, Driver는 20개 이상의 택배를 배정받을 수 없다.

- Driver는 Branch와 1:N 관계를 맺는다. Driver는 무조건 소속 branch가 있어야 한다. Driver는 Car와 1:1 관계를 맺는다. Driver는 무조건 매핑된 car 인스턴스가 있어야만 택배 운송을 할 수 있다. 따라서 해당 relation에서 Driver는 전체 참여다.

- Driver의 Salary는 total_packs의 영향을 받는다. Total_packs는 0과 양의 정수 범위를 가진다. 0일 경우, 월급은 없다. 0~4일 경우, 기본급을 수령할 수 있다. 5 이상일 경우, 기본급 + (누적 택배 개수 X 15000)원의 월급을 수령한다. Total_packs는 package와 매핑되는 시점에 1씩 증가하고 배송이 완료되는 경우에도 total_packs는 줄어들지 않는다. 다만, 매월 초에 0으로 초기화된

다.

Car

- attribute는 pk인 id와 차에 실을 수 있는 택배의 무게(capa_w), 택배의 개수(capa_n)으로 구성한다. 해당 엔티티의 인스턴스와 매핑된 drive는 배송할 택배 개수(packages)를 capability를 초과하지 못한다는 제약이 생긴다.

- Driver, Branch와 degree가 3인 relation을 갖는다. Car 1개를 선택했을 때, 대응되는 driver 수는 0~1 범위를 갖는다. Car의 입장에서 driver를 지정받지 못한 car가 존재할 수 있다는 것이다. 대응되는 branch 수는 1~1 범위를 갖는 전체 참여 관계다. 어느 한 car는 소속 branch가 항상 있어야 한다.

- Car는 Driver와 1:1 관계에서 부분 참여다. Driver는 무조건 Car가 필요하지만, Car는 배정받은 Driver가 없을 수 있다.

- Car는 Branch와 N:1 관계에서 전체 참여다. Car는 무조건 Branch에 소속되어야 한다.

Branch

- attrubute는 pk인 id와 지점 이름(name), 지점 번호(phone_no), 상위 지점(manage)으로 구성한다.

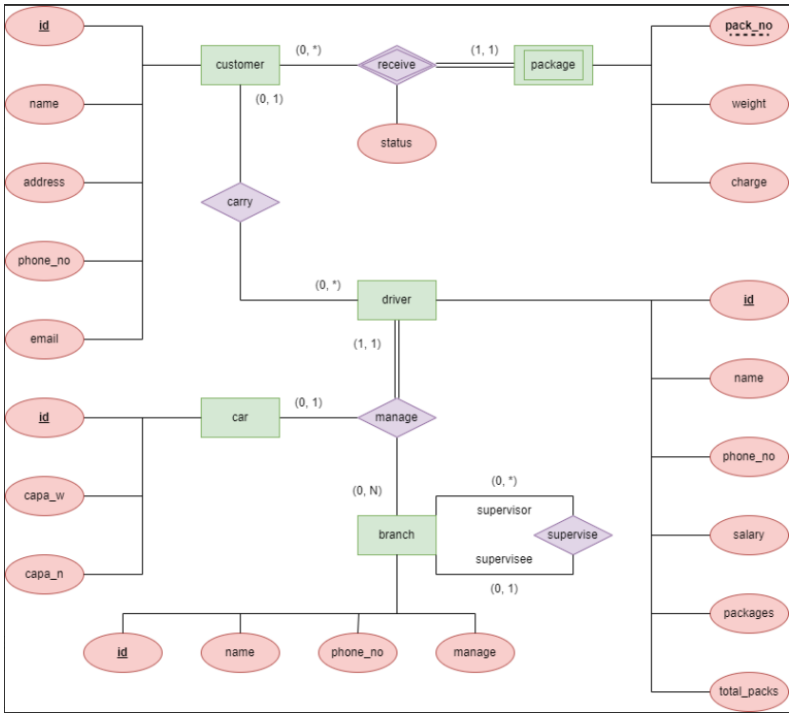
- Driver, Car와 degree가 3인 relation을 갖는다. Branch 1개를 선택했을 때, 대응되는 driver의 수는 0~N 범위를 갖는다. 대응되는 Car의 수는 0~N 범위를 갖는다.

- Branch는 자기 자신의 엔티티와 순환적 관계를 갖는다. 하위 지점을 Low branch, 하위 지점을 관리하는 상위 지점을 High branch라 가정하면, Low branch는 High branch와 N:1 관계를 가진다. Low branch 1개를 선택했을 때, 대응되는 High branch 수는 0~1의 범위를 갖는다. 대응되는 High branch가 0이라면 해당 지점은 최상위 지점 혹은 본사가 될 것이다. High branch는 Low branch와 1:N 관계를 가진다. High branch 1개를 선택했을 때, 대응되는 Low branch 수는 0~N 범위를 가진다. 대응되는 Low branch가 N이라면 해당 지점은 많은 하위 지점을 관리하는 것을 의미한다. 0의 경우, 해당 지점은 관리하는 지점이 없는 최하위 지점을 의미한다.

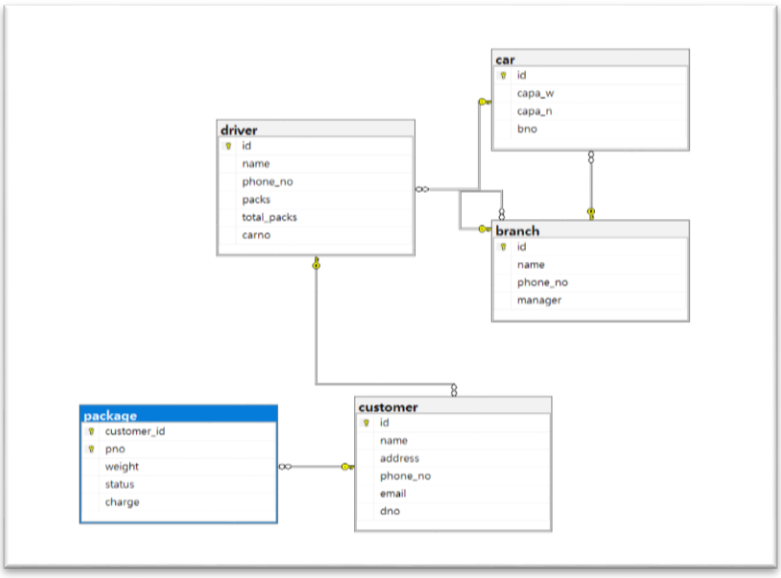
- Branch는 Driver와 1:N 관계에서 부분 참여다. 해당 branch에 소속된 driver는 0일 수도 있다.

- Branch는 Car와 1:N 관계에서 부분 참여다. 해당 branch에서 관리하는 배달 운송 수단은 0개일 수도 있다.

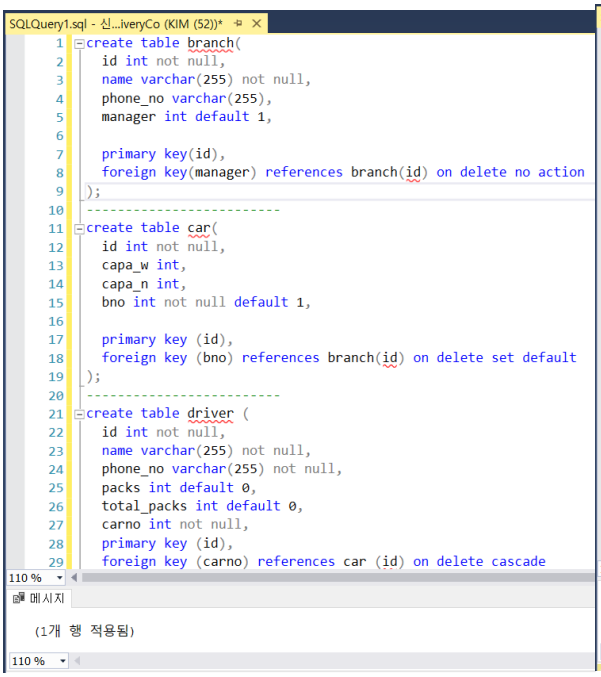
2. 최종 ER_Diagram



3. 최종 DB Schema



4. 최종 SQL 캡쳐 화면



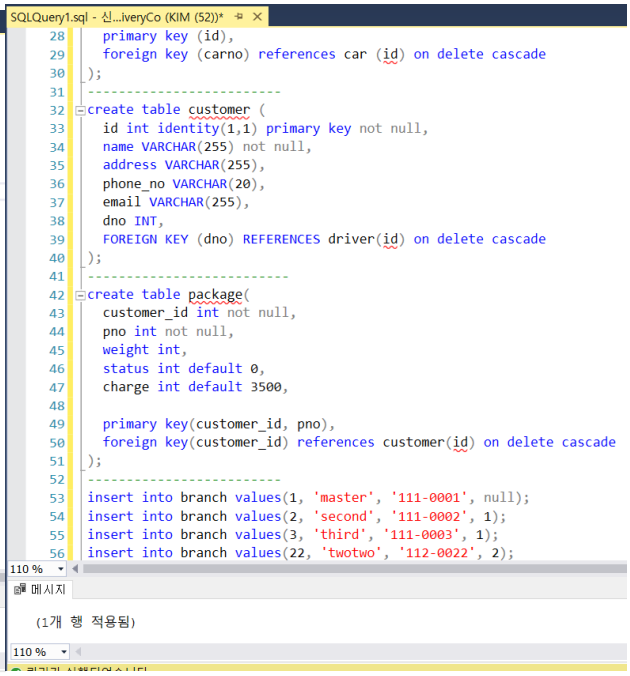
```sql
create table branch(
    id int not null,
    name varchar(255) not null,
    phone_no varchar(255),
    manager int default 1,

    primary key(id),
    foreign key(manager) references branch(id) on delete no action
);
-------------------------
create table car(
    id int not null,
    capa_w int,
    capa_n int,
    bno int not null default 1,

    primary key (id),
    foreign key (bno) references branch(id) on delete set default
);
-------------------------
create table driver (
    id int not null,
    name varchar(255) not null,
    phone_no varchar(255) not null,
    packs int default 0,
    total_packs int default 0,
    carno int not null,
    primary key (id),
    foreign key (carno) references car (id) on delete cascade
```

메시지

(1개 행 적용됨)

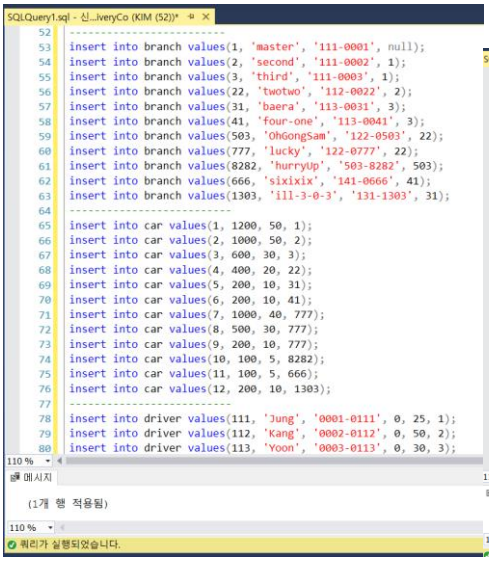


```sql
    primary key (id),
    foreign key (carno) references car (id) on delete cascade
);
-------------------------
create table customer (
    id int identity(1,1) primary key not null,
    name VARCHAR(255) not null,
    address VARCHAR(255),
    phone_no VARCHAR(20),
    email VARCHAR(255),
    dno INT,
    FOREIGN KEY (dno) REFERENCES driver(id) on delete cascade
);
-------------------------
create table package(
    customer_id int not null,
    pno int not null,
    weight int,
    status int default 0,
    charge int default 3500,

    primary key(customer_id, pno),
    foreign key(customer_id) references customer(id) on delete cascade
);
-------------------------
insert into branch values(1, 'master', '111-0001', null);
insert into branch values(2, 'second', '111-0002', 1);
insert into branch values(3, 'third', '111-0003', 1);
insert into branch values(22, 'twotwo', '112-0022', 2);
```

메시지

(1개 행 적용됨)

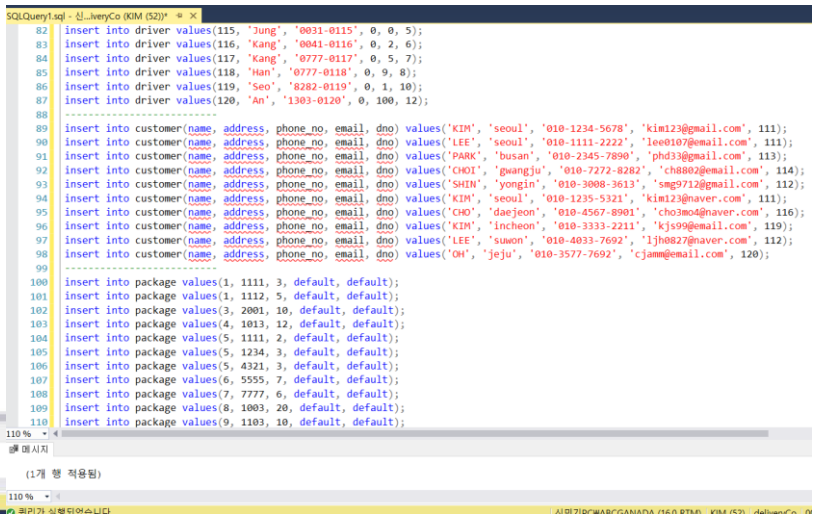


```sql
-------------------------
insert into branch values(1, 'master', '111-0001', null);
insert into branch values(2, 'second', '111-0002', 1);
insert into branch values(3, 'third', '111-0003', 1);
insert into branch values(22, 'twotwo', '112-0022', 2);
insert into branch values(31, 'baera', '113-0031', 3);
insert into branch values(41, 'four-one', '113-0041', 3);
insert into branch values(503, 'OhGongSam', '122-0503', 22);
insert into branch values(777, 'lucky', '122-0777', 22);
insert into branch values(8282, 'hurryUp', '503-8282', 503);
insert into branch values(666, 'sixisix', '141-0666', 41);
insert into branch values(1303, 'ill-3-0-3', '131-1303', 31);
-------------------------
insert into car values(1, 1200, 50, 1);
insert into car values(2, 1000, 50, 2);
insert into car values(3, 600, 30, 3);
insert into car values(4, 400, 20, 22);
insert into car values(5, 200, 10, 31);
insert into car values(6, 200, 10, 41);
insert into car values(7, 1000, 40, 777);
insert into car values(8, 500, 30, 777);
insert into car values(9, 200, 10, 777);
insert into car values(10, 100, 5, 8282);
insert into car values(11, 100, 5, 666);
insert into car values(12, 200, 10, 1303);
-------------------------
insert into driver values(111, 'Jung', '0001-0111', 0, 25, 1);
insert into driver values(112, 'Kang', '0002-0112', 0, 50, 2);
insert into driver values(113, 'Yoon', '0003-0113', 0, 30, 3);
```

메시지

(1개 행 적용됨)

쿼리가 실행되었습니다.



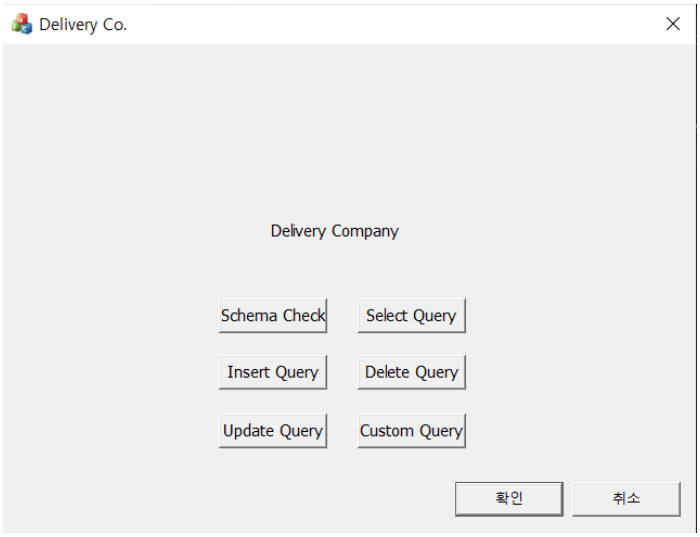
```sql
insert into driver values(115, 'Jung', '0031-0115', 0, 0, 5);
insert into driver values(116, 'Kang', '0041-0116', 0, 2, 6);
insert into driver values(117, 'Kang', '0777-0117', 0, 5, 7);
insert into driver values(118, 'Han', '0777-0118', 0, 9, 8);
insert into driver values(119, 'Seo', '8282-0119', 0, 1, 10);
insert into driver values(120, 'An', '1303-0120', 0, 100, 12);
-------------------------
insert into customer(name, address, phone_no, email, dno) values('KIM', 'seoul', '010-1234-5678', 'kim123@gmail.com', 111);
insert into customer(name, address, phone_no, email, dno) values('LEE', 'seoul', '010-1111-2222', 'lee0107@email.com', 111);
insert into customer(name, address, phone_no, email, dno) values('PARK', 'busan', '010-2345-7890', 'phd33@gmail.com', 113);
insert into customer(name, address, phone_no, email, dno) values('CHOI', 'gwangju', '010-7272-8282', 'ch8802@email.com', 114);
insert into customer(name, address, phone_no, email, dno) values('SHIN', 'yongin', '010-3008-3613', 'smg9712@gmail.com', 112);
insert into customer(name, address, phone_no, email, dno) values('KIM', 'seoul', '010-1235-5321', 'kim123@naver.com', 111);
insert into customer(name, address, phone_no, email, dno) values('CHO', 'daejeon', '010-4567-8901', 'cho3mo4@naver.com', 116);
insert into customer(name, address, phone_no, email, dno) values('KIM', 'incheon', '010-3333-2211', 'kjs99@email.com', 119);
insert into customer(name, address, phone_no, email, dno) values('LEE', 'suwon', '010-4033-7692', 'ljh0827@naver.com', 112);
insert into customer(name, address, phone_no, email, dno) values('OH', 'jeju', '010-3577-7692', 'cjamm@email.com', 120);
-------------------------
insert into package values(1, 1111, 3, default, default);
insert into package values(1, 1112, 5, default, default);
insert into package values(3, 2001, 10, default, default);
insert into package values(4, 1013, 12, default, default);
insert into package values(5, 1111, 2, default, default);
insert into package values(5, 1234, 3, default, default);
insert into package values(5, 4321, 3, default, default);
insert into package values(6, 5555, 7, default, default);
insert into package values(7, 7777, 6, default, default);
insert into package values(8, 1003, 20, default, default);
insert into package values(9, 1103, 10, default, default);
```

메시지

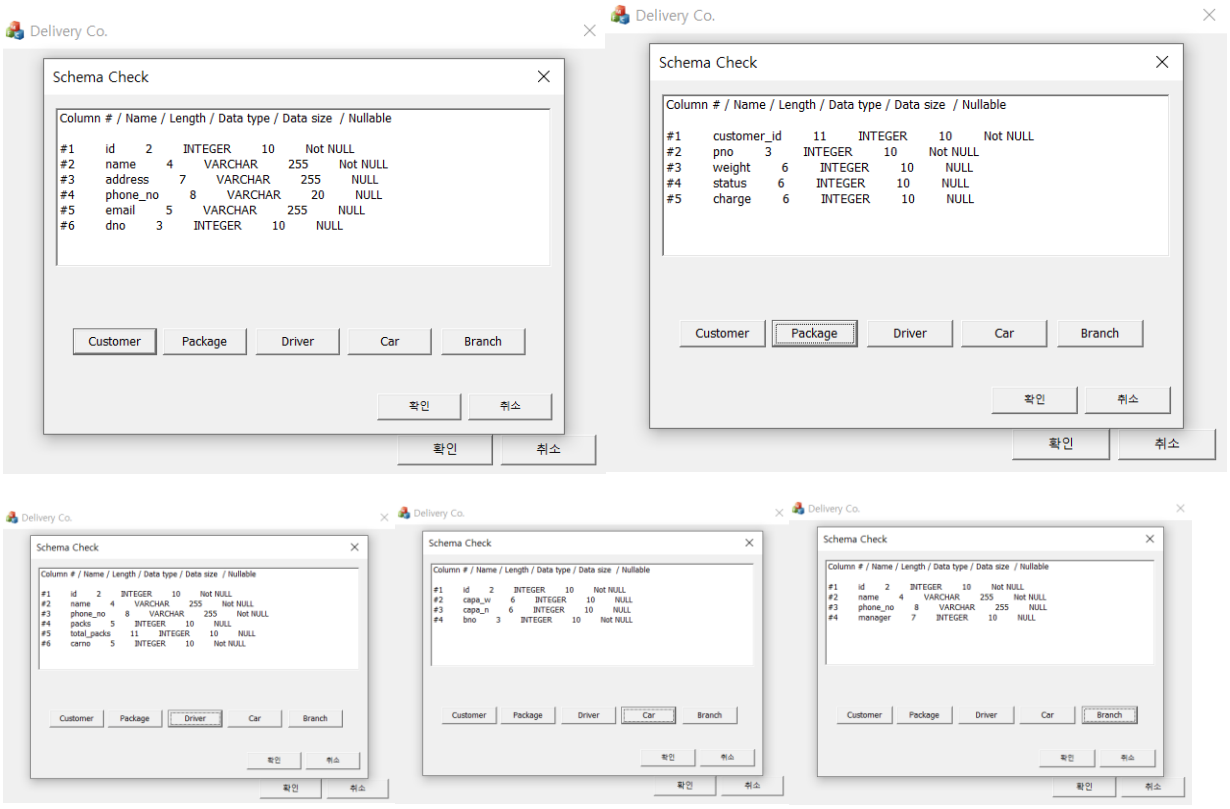
(1개 행 적용됨)

5. 프로그램 실행 캡쳐 화면

0) 메인페이지



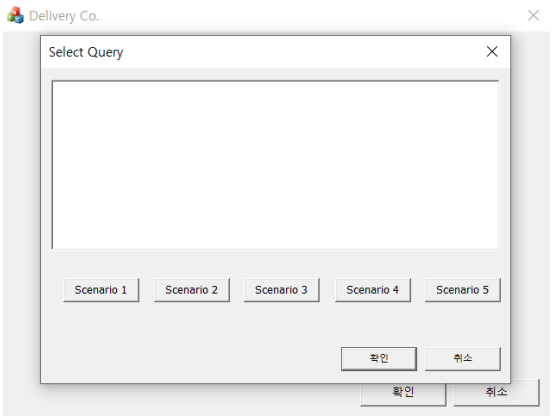
1) Schema Check



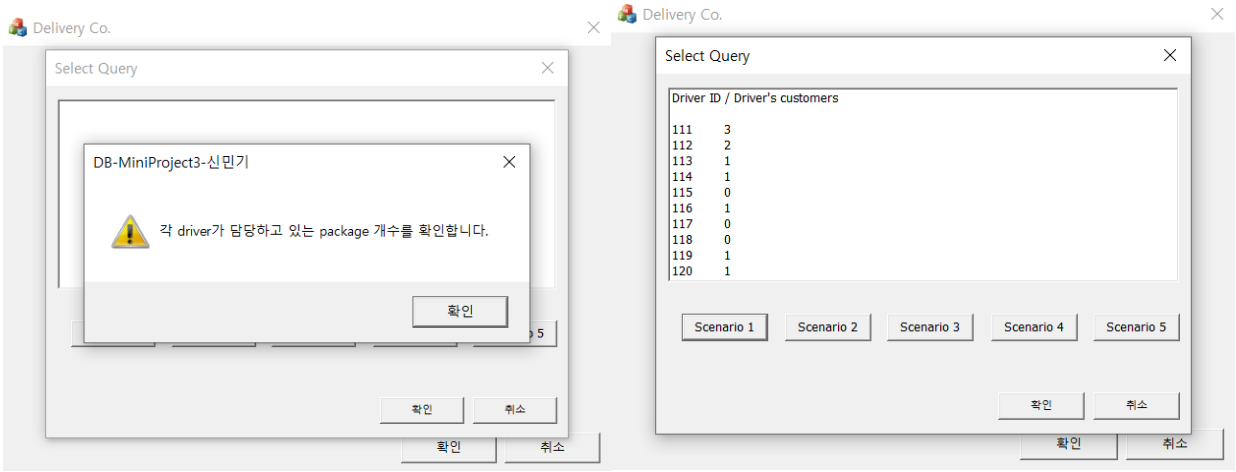
- 쿼리: `Select * from <Table>`

- 테이블 버튼 클릭시, 첫 줄에 애트리뷰트 설명 후 실제 데이터 값들 출력.

2) Select Query

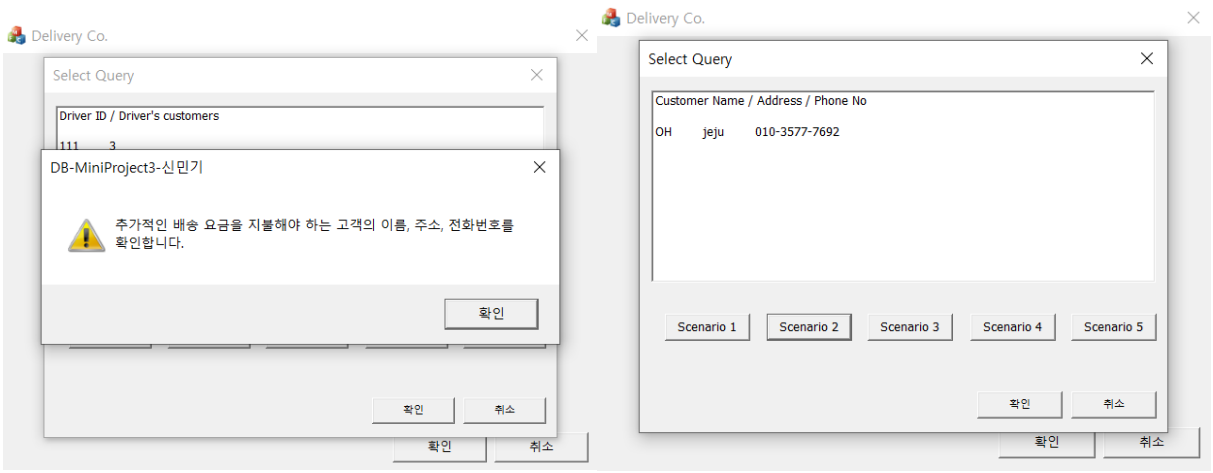


- Scenario 1: 각 driver가 담당하고 있는 Package 개수를 확인합니다.



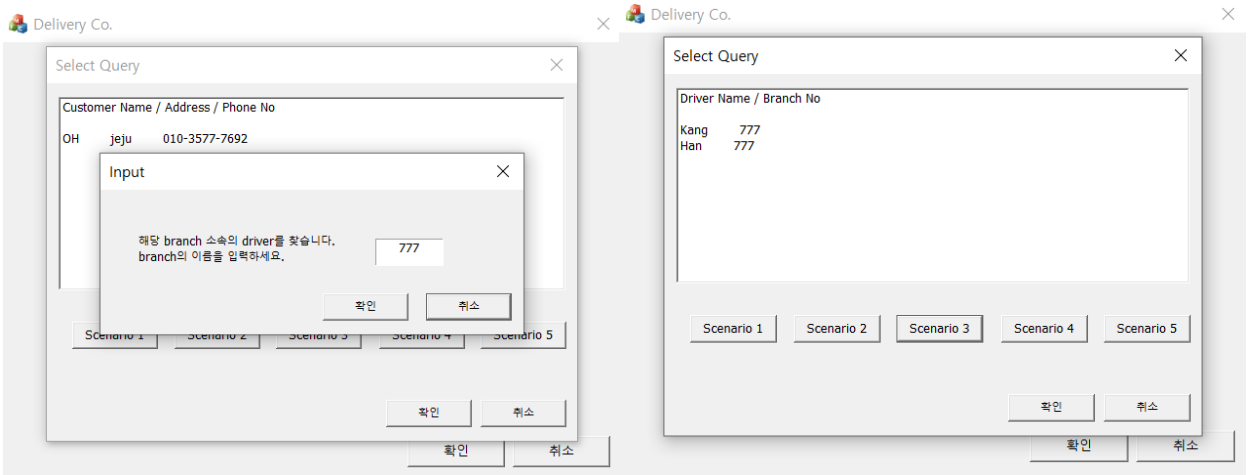

```
- select id, (select count(*) from customer where customer.dno = driver.id) as total_customer from driver;
```

- Scenario 2: 추가 배송 요금을 지불해야 하는 고객의 이름, 주소, 전화번호 select.




```
- select name, address, phone_no from customer where address='jeju';
```

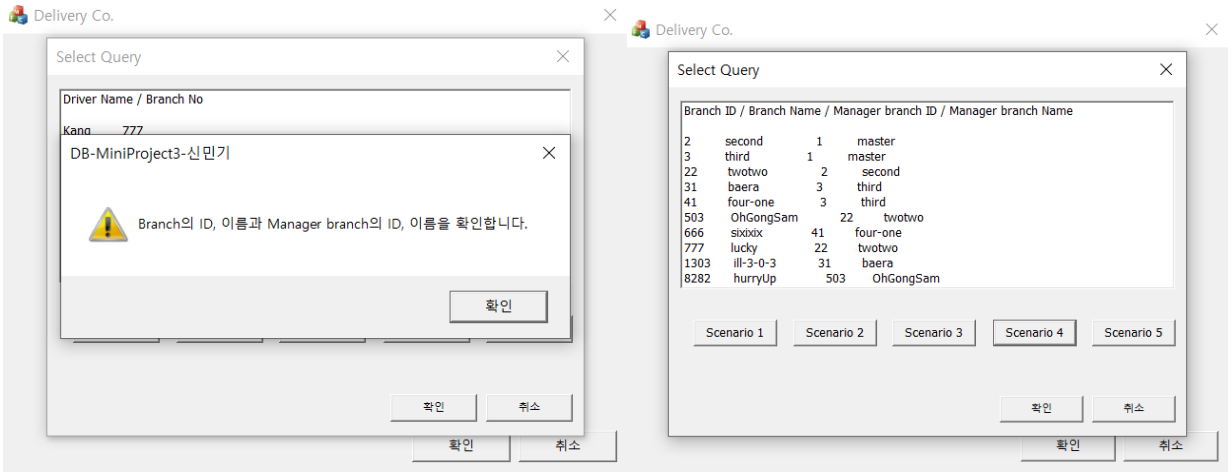
- Scenario 3: 해당 branch 소속의 driver를 select.




```
- select driver.name, car.bno from driver, car where driver.carno = car.id and car.bno = '%s';
```

- 쿼리의 '%s'는 프로그램 상에서 입력값에 해당한다.

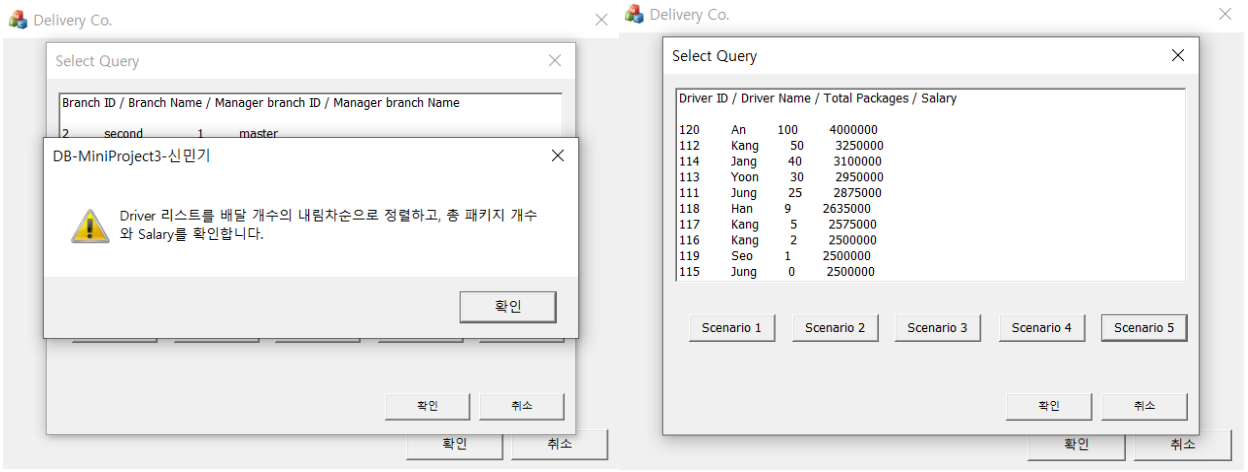
- Scenario 4: Branch의 ID, 이름과 Manager branch의 ID, 이름을 확인합니다.



- `select lb. id, lb.name, hb.id, hb.name from branch as lb, branch as hb where lb.manager = hb.id;`

- branch 테이블의 순환적 관계를 이용한 select 시나리오다.

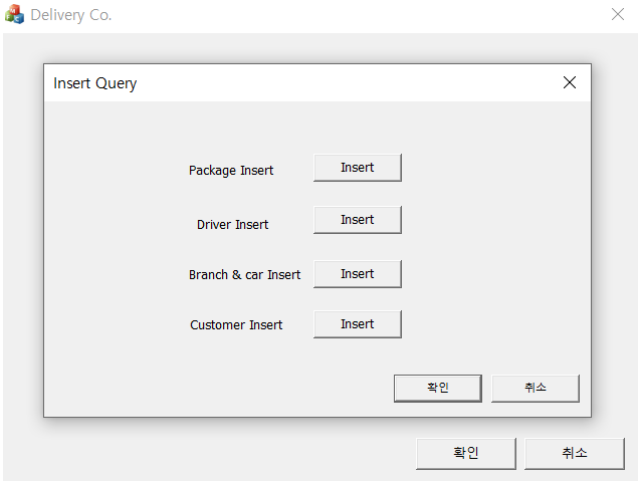
- Scenario 5: Driver 리스트를 배달 개수의 내림차순으로 정렬하고, 총 패키지 개수와 Salary를 확인합니다.



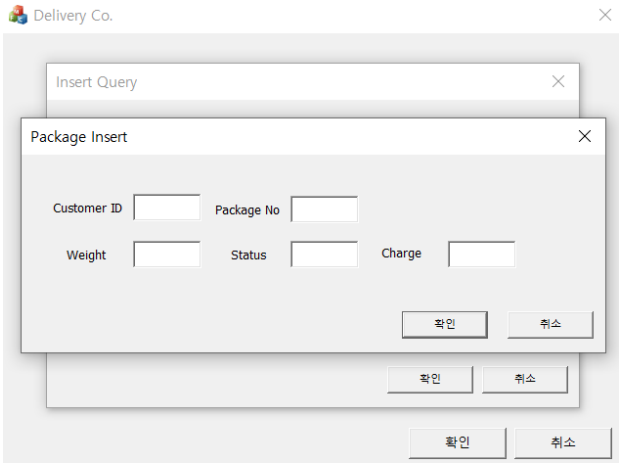
- `SELECT id, name, total_packs, CASE WHEN total_packs >= 3 THEN 2500000 + (15000 * total_packs) ELSE 2500000 END AS Salary FROM driver order by total_packs desc;`

- driver가 배달한 총 택배 개수가 3개 미만이라면 salary는 기본급(2,500,000)이다. 3개 이상일 경우 기본급에 배달한 택배 개수 * 15,000의 액수를 수령한다.

3) Insert Query



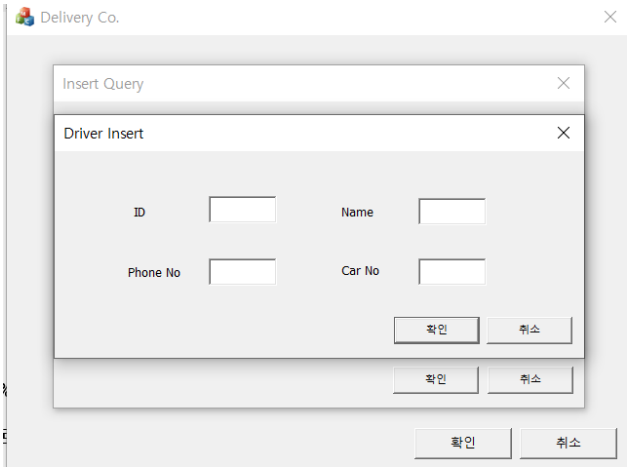
- Scenario 1: 택배 삽입



- `insert into package values('%s', '%s', '%s', '%s', '%s');`

- package 테이블의 모든 애트리뷰트를 프로그램상에서 입력값으로 받아 확인버튼을 누르면 택배 튜플이 삽입된다.

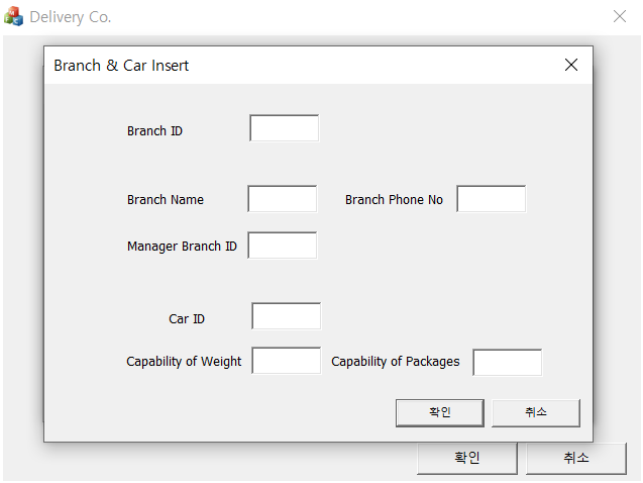
- Scenario 2: driver 삽입



- `insert into driver values('%s', '%s', '%s', 0, 0, '%s');"), str1, str2, str3, str4;`

- driver 삽입시 packs와 total_packs는 0으로 삽입된다. ID, Name, Phone number, 외래키 Car number를 프로그램에서 입력받는다.

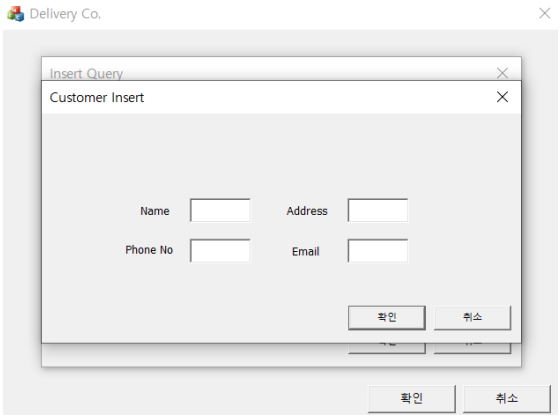
- Scenario 3: branch, Car 삽입



- `insert into branch values('%s', '%s', '%s', '%s');`
- `insert into car values('%s', '%s', '%s', '%s');`

- Branch ID는 필수적으로 입력해야 한다. 모든 칸을 입력해도 되지만, Car 튜플 하나만 삽입할 수 있다. Branch ID를 입력하고, Car ID, Capability of Weight, Capability of Packages만 입력해도 삽입이 가능하다.

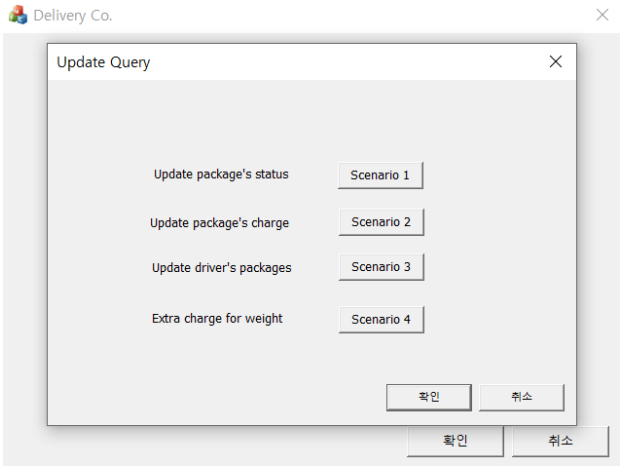
- Scenario 4: Customer 삽입



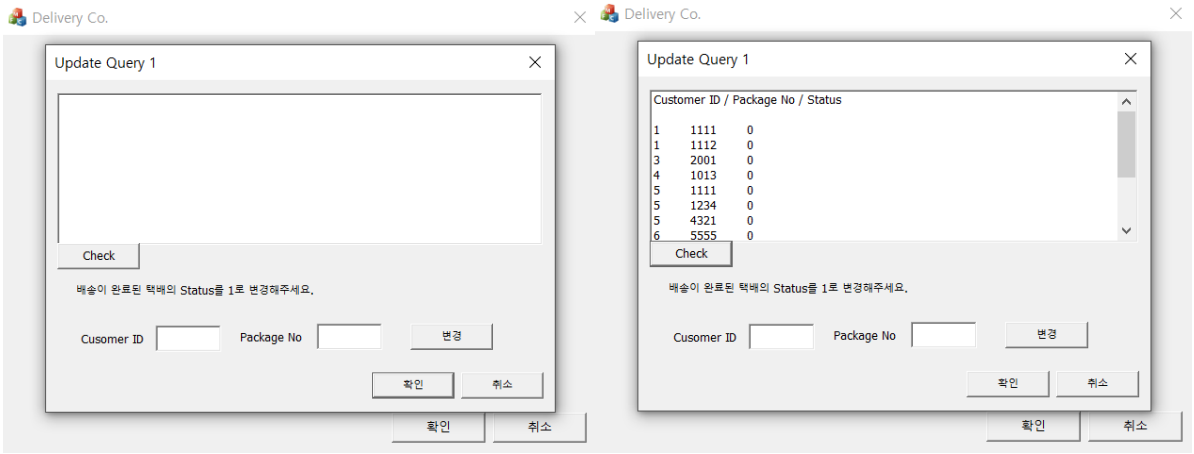
- `insert into customer(name, address, phone_no, email, dno) values('%s', '%s', '%s', '%s', NULL);`

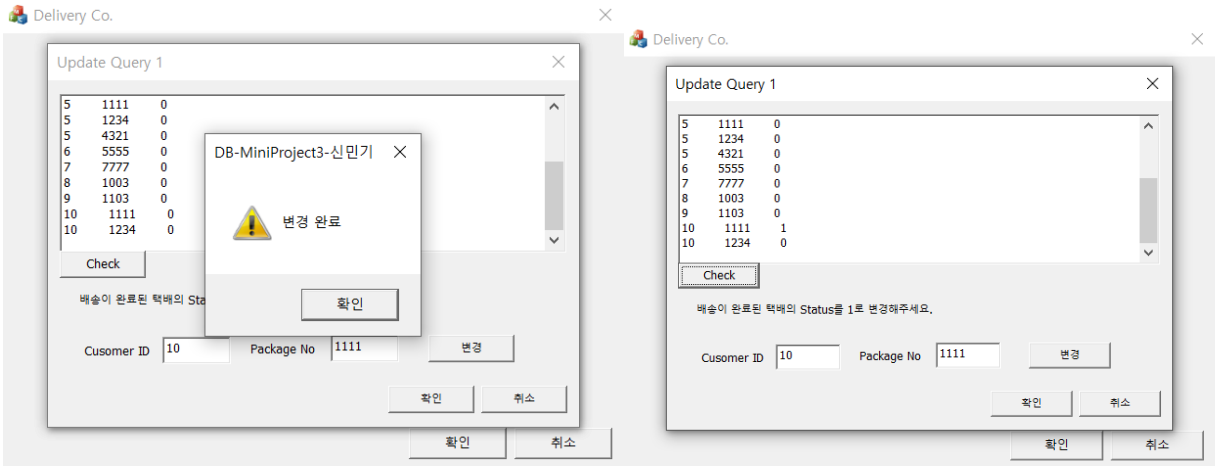
- 고객 튜플 삽입시, 외래키 dno는 최초 NULL로 삽입된다. Primary key 'Id'는 MsSQL의 자동 증가 쿼리로 create했기 때문에 이름, 주소, 전화번호, 이메일을 프로그램에 입력받아 튜플을 삽입한다.

4) Update Query



- Scenario 1: 배송이 완료된 택배의 Status Update.





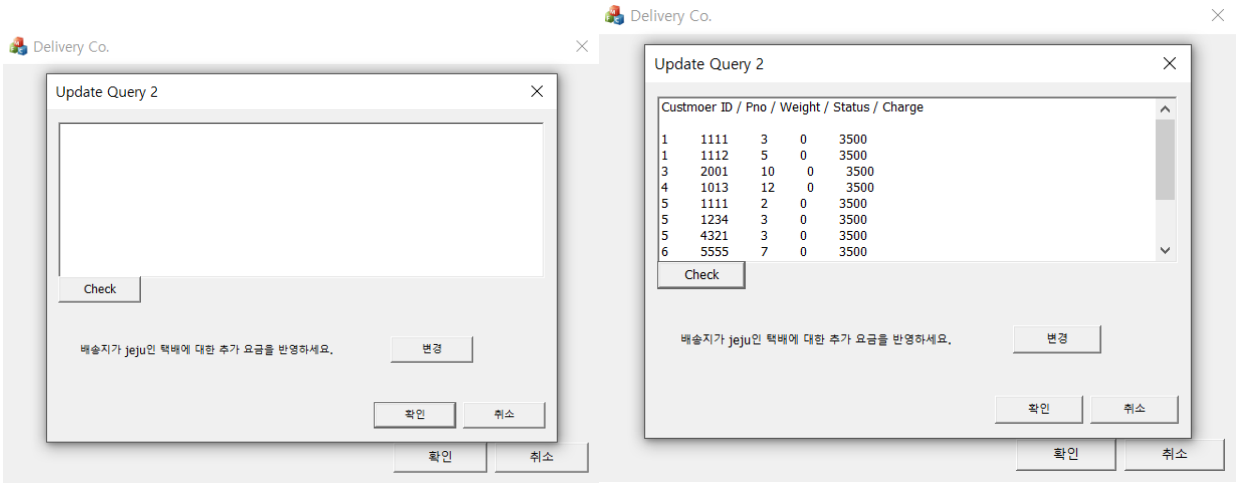
- Check: `select customer_id, pno, status from package;`

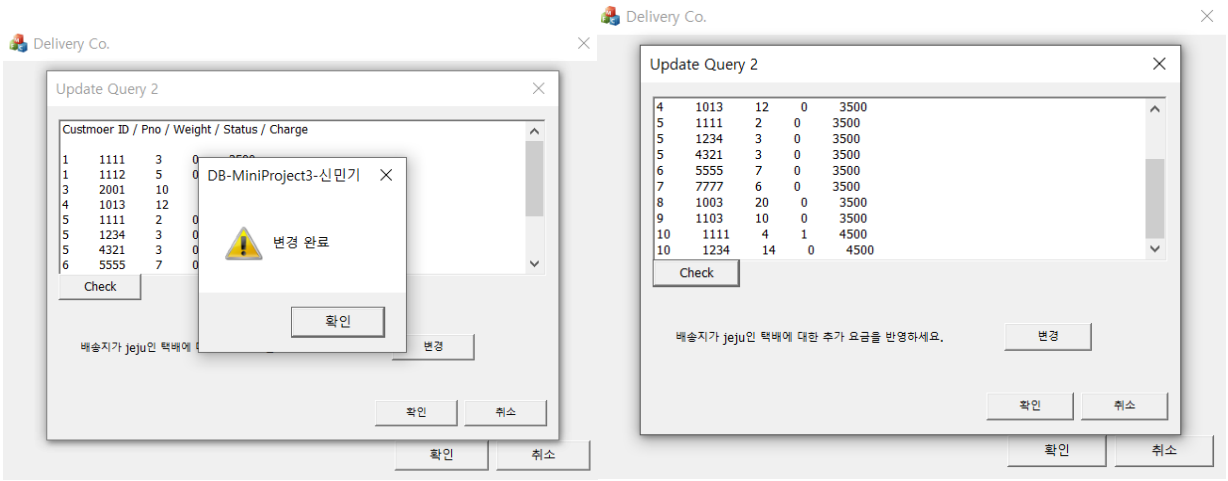
- Update: `update package set status = 1 where customer_id = '%s' and pno = '%s';`

- Check 버튼 입력시, 택배들의 status를 select한다. Package 테이블은 customer의 id와 package number를 복합키로 사용하기 때문에, 이 두 가지를 입력하고 변경버튼을 누르면 status가 1로 변경된다. Status가 1이라하면 배송이 완료되었다는 뜻이

다.

- Scenario 2: 배송지가 제주도인 택배에 대한 charge Update.



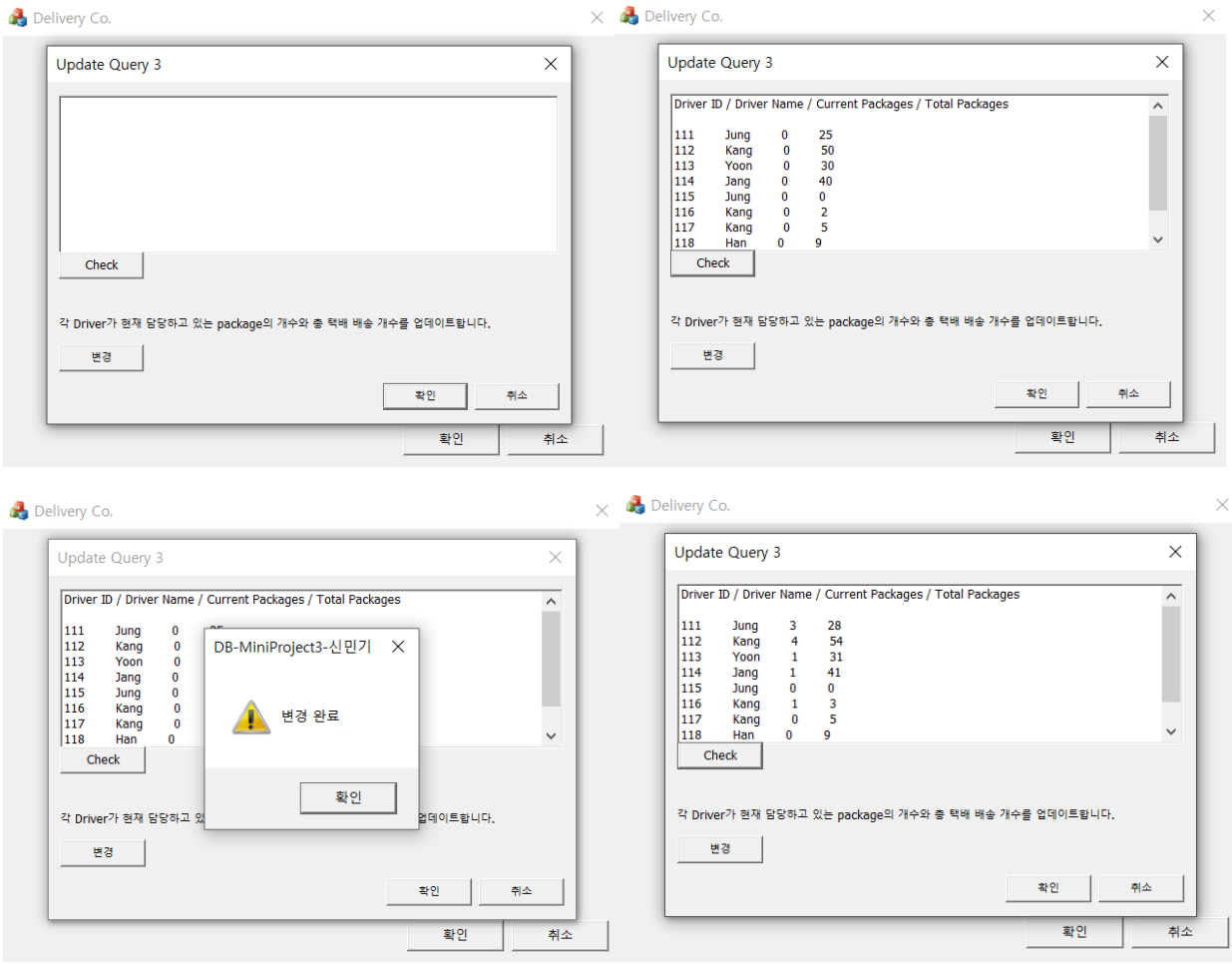


- Check: `SELECT * FROM package;`

- Update: `update package set charge = charge + 1000 where customer_id in (select id from customer where address = 'jeju');`

- 배송지가 제주인 경우 추가 배송 요금이라는 요구 사항이 있다. 기본 배송요금 3500에서 제주도로 갈 택배에 1000을 추가하는 시나리오다.

- Scenario 3: Driver의 현재 택배 개수와 총 택배 개수 Update.



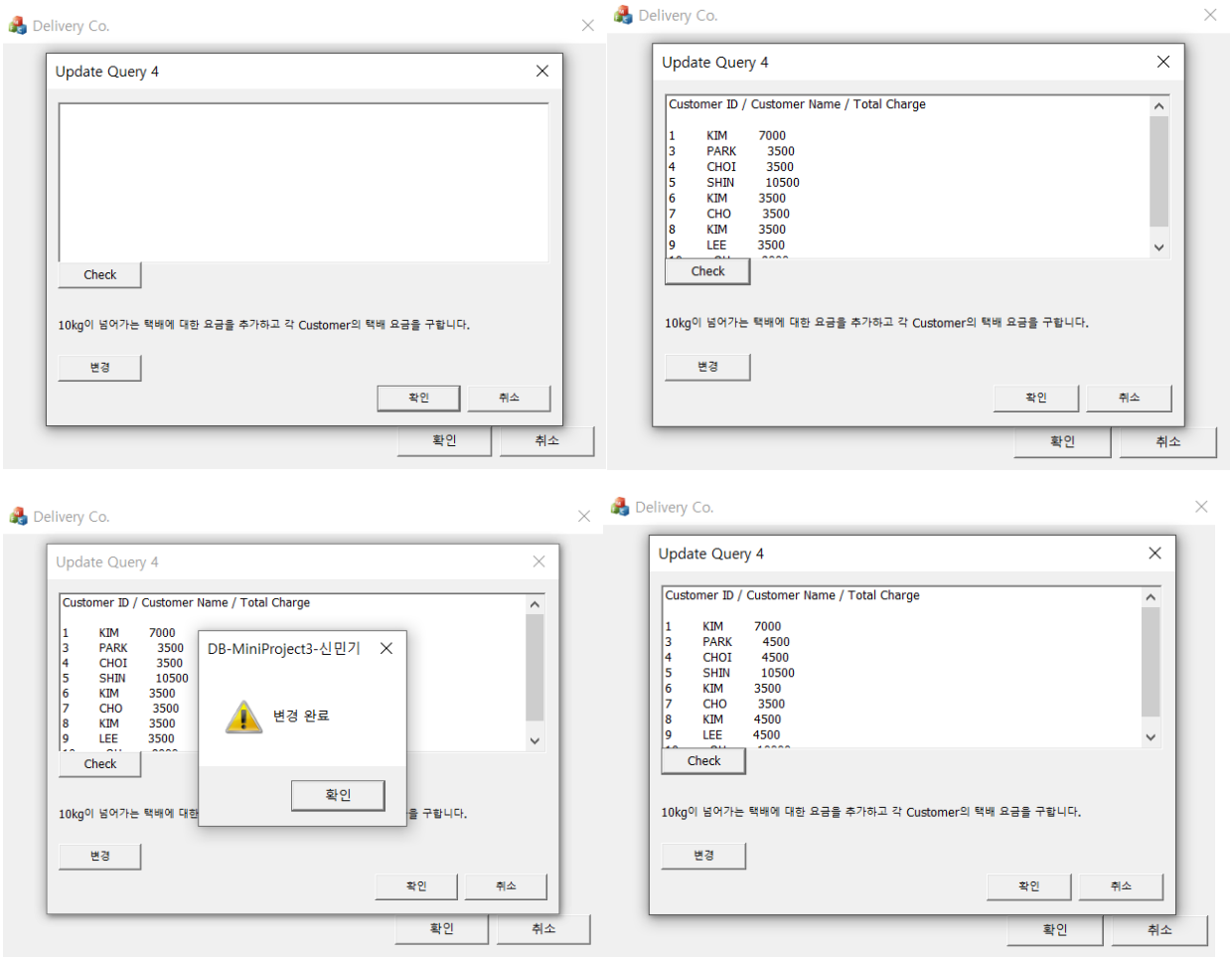
- Check: `select id, name, packs, total_packs from driver;`

- Update1: `update driver set packs = (select count(*) from customer, package where driver.id = customer.dno and customer.id = package.customer_id and package.status < > 1);`

- Update2: `update driver set total_packs = total_packs + packs;`

- check 버튼 클릭시, driver의 현재 택배 개수, 총 택배 개수를 보여준다. Update1은 driver의 현재 택배 개수를 업데이트 한다. Update 쿼리에 중첩 질의를 사용하며, driver, customer, package 3개의 테이블을 join한다. 여기서 Status가 1인(배송이 완료된) 택배 튜플에 대해서는 count하지 않는다. Update2는 업데이트된 현재 택배 개수를 총 택배 개수에 더하는 방식으로 총 택배 개수에 반영한다.

- Scenario 4: 무게가 10kg이 넘어가는 택배에 추가 요금 Update.

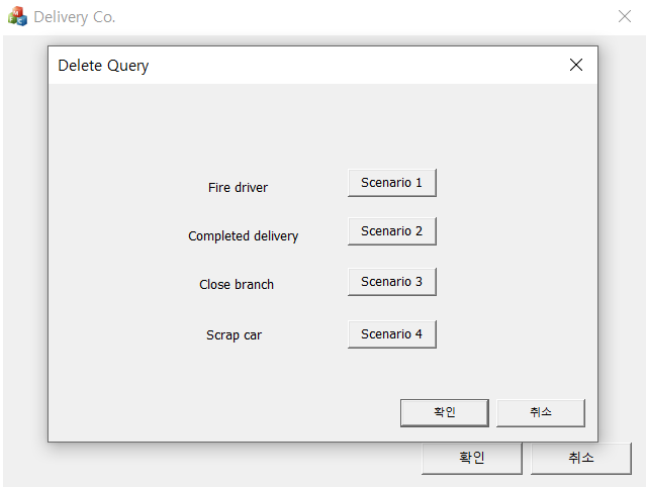


- Check: `select customer.id, customer.name, SUM(package.charge) as total_charge from customer, package where customer.id = package.customer_id group by customer.id, customer.name;`

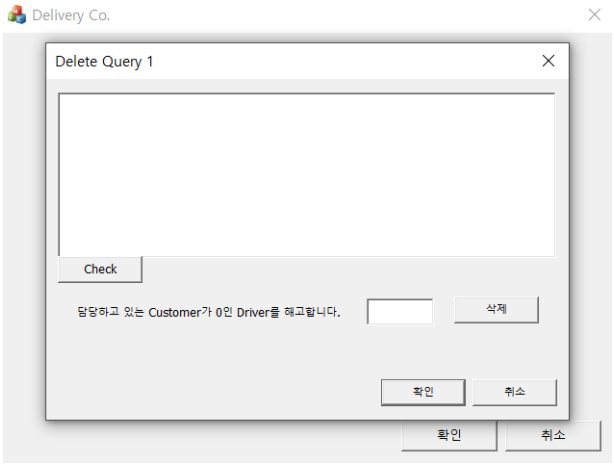
- Update: `update package set charge = charge + 1000 where weight >= 10;`

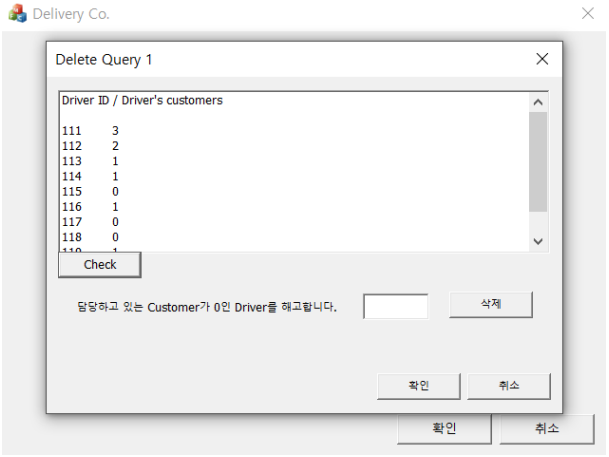
Check 버튼 클릭시, customer의 총 택배 요금을 group by로 묶어 보여준다. Update 버튼 클릭시, 택배 무게가 10kg 이상이면 요금을 1000 더한다.

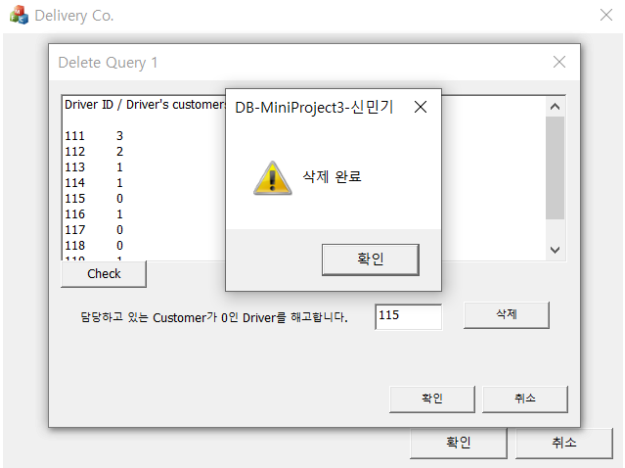
5) Delete Query

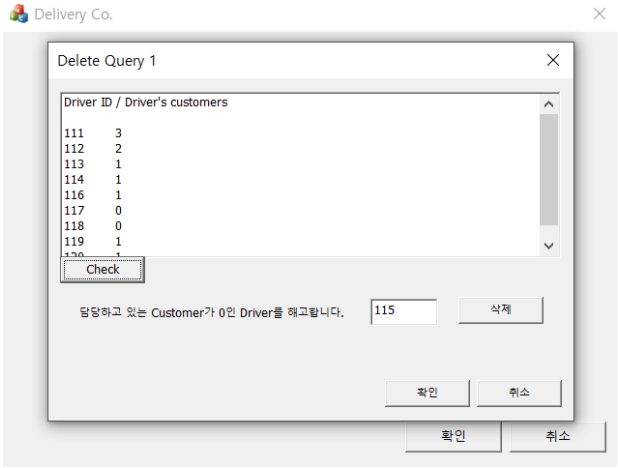


- Scneario 1: 담당 customer가 없는 driver를 해고한다.







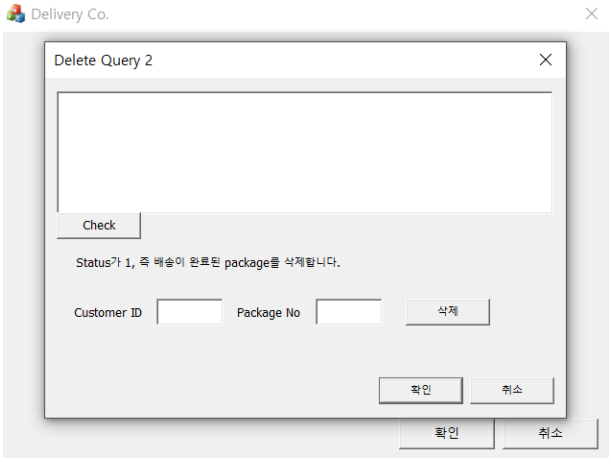


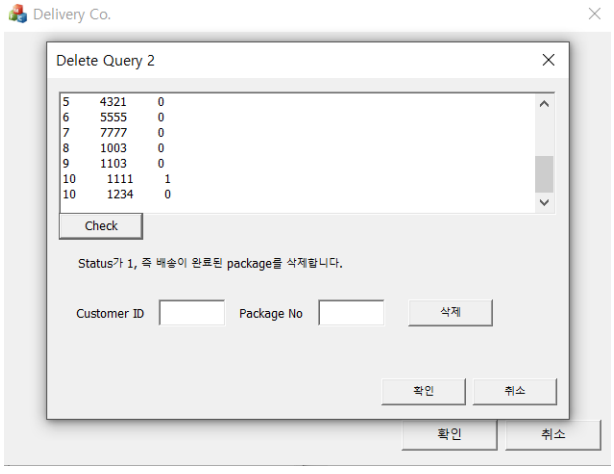
- Check: `select id, (select count(*) from customer where customer.dno = driver.id) as total_customer from driver;`

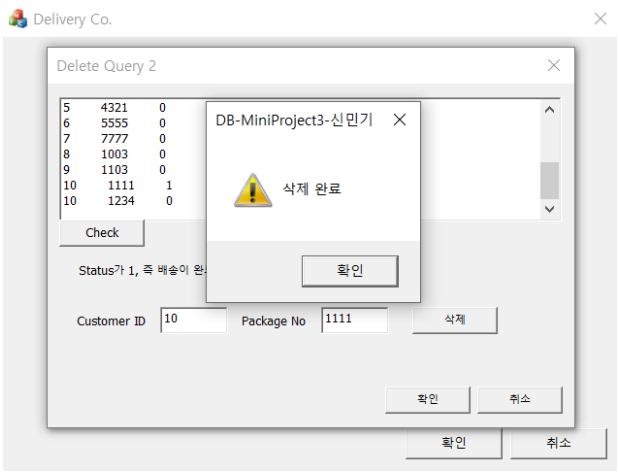
- Delete: `delete from driver where id = '%s';`

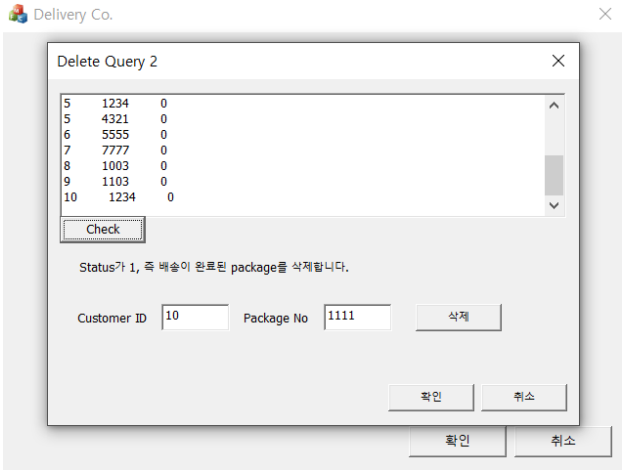
- driver와 customer는 1:N 관계이다. Check 버튼 클릭시, driver와 매핑된 customer 튜플의 개수를 보여준다. 이 중 0인 driver의 id를 입력값으로 받아 삭제한 후 다시 check 버튼을 클릭하면 해당 driver 튜플이 삭제된 것을 볼 수 있다.

- Scenario 2: 배송 완료 택배 삭제





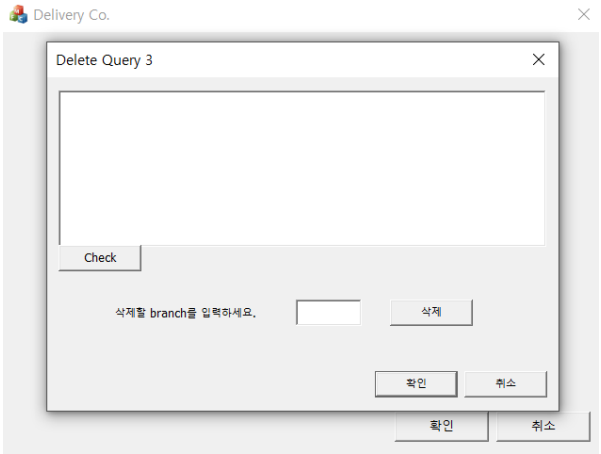


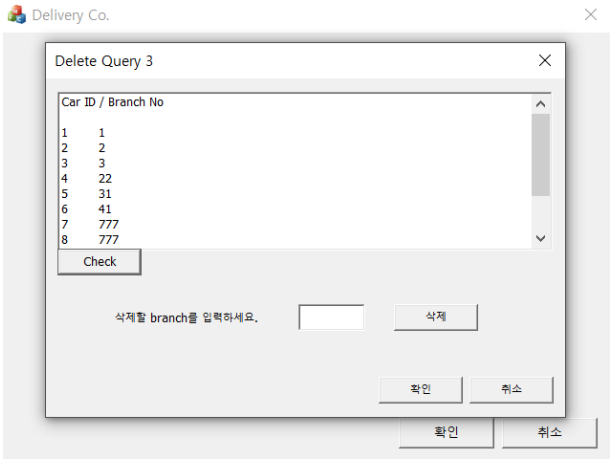


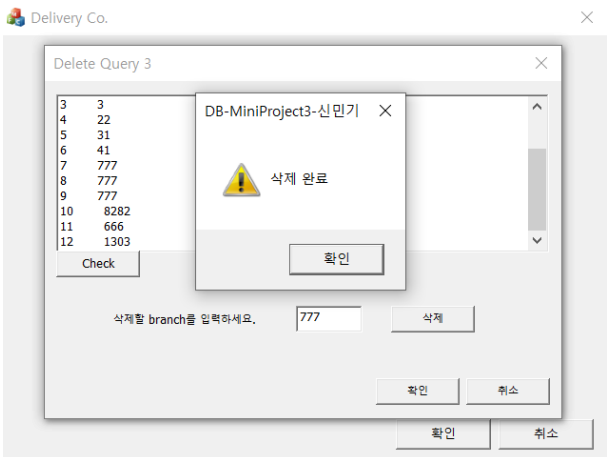
- Check: `select customer_id, pno, status from package;`
- Delete: `delete from package where customer_id = '%s' and pno = '%s';`

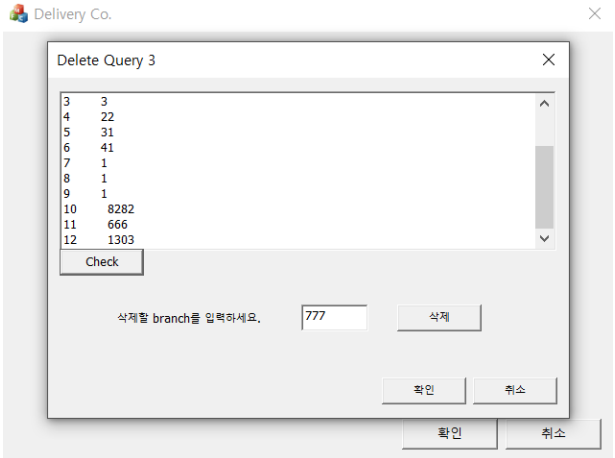
- Check 버튼 클릭시, customer ID와 택배 번호, 배송 상태를 보여준다. 택배 엔티티는 customer ID와 택배 번호를 복합키로 사용하고 있기 때문에, 이 둘을 입력값으로 받아 삭제 버튼을 클릭한 후 check 버튼을 다시 누르면 해당 튜플이 삭제된 것을 확인할 수 있다.

- Scenario 3: branch 삭제







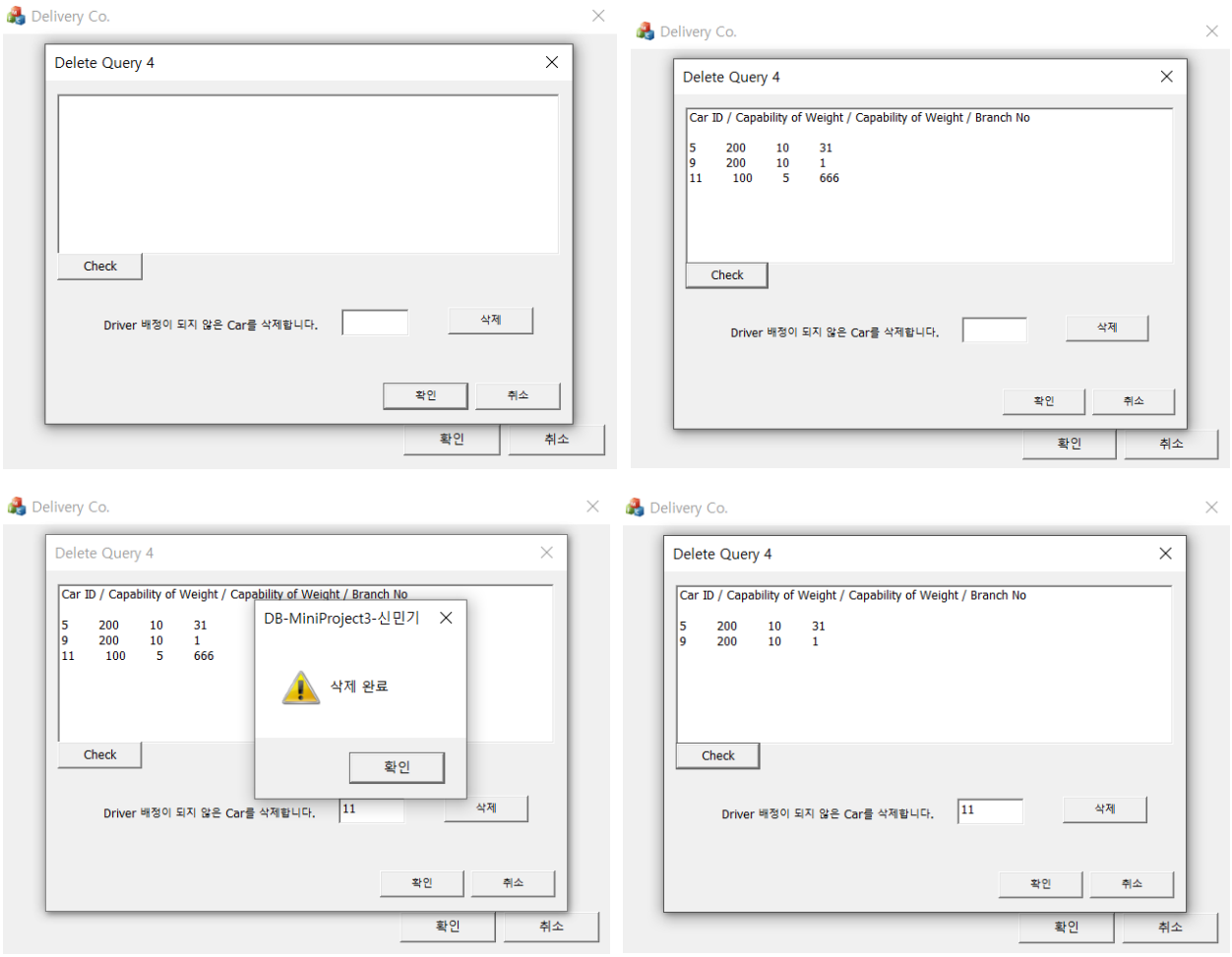


- Check: `select id, bno from car;`

- Delete: `delete from branch where id = '%s';`

- Check 버튼 클릭시, 차량의 번호와 branch의 번호를 출력한다. 위 그림처럼 777 branch에는 3대의 차량이 소속되어 있다. 이 상황에서 해당 branch를 삭제할 경우, on delete set default로 인해 default branch로 소속이 이전된다.

- Scenario 4: Driver 매핑이 없는 Car 튜플 삭제

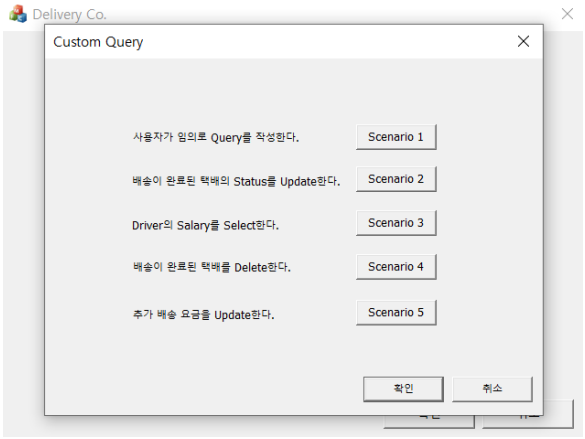


- Check: `select * from car where id not in (select carno from driver);`

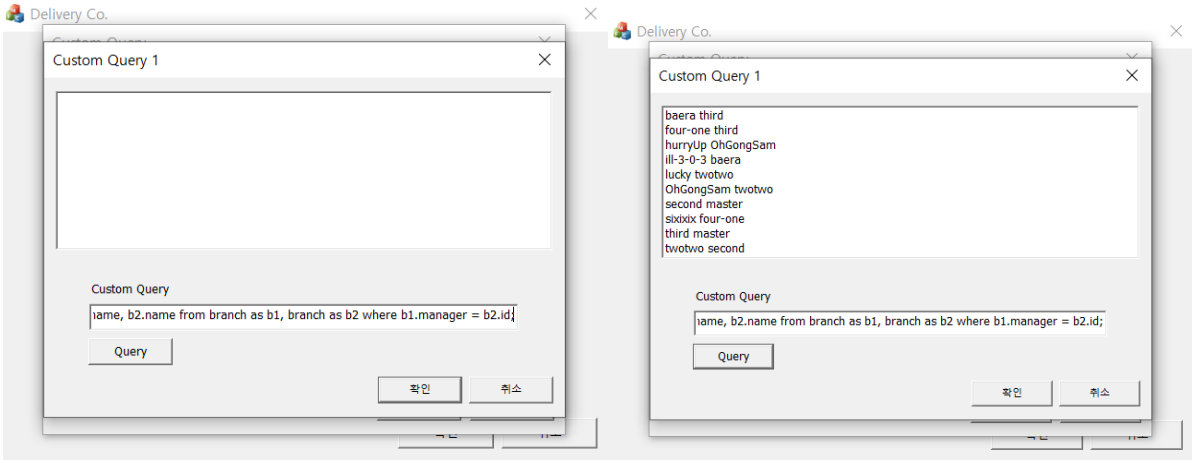
- Delete: `delete from car where car.id = '%s';`

- Check 버튼 클릭시, driver와 매핑이 없는 car의 정보를 출력한다. Car ID를 입력값으로 하여 삭제 버튼을 클릭한 후 Check 버튼을 누르면 해당 car 튜플이 삭제된다.

6) Custom Query



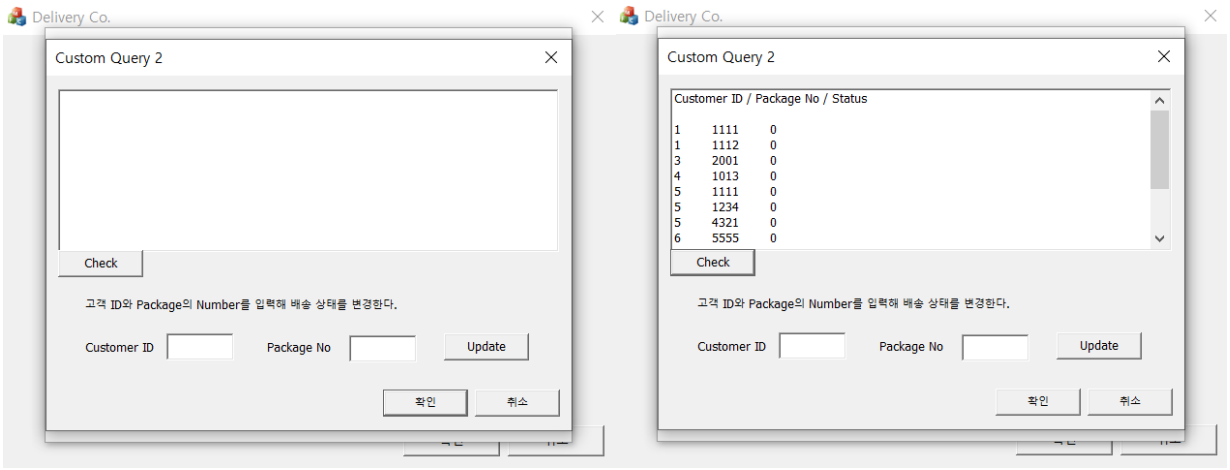
- Scenario 1: 사용자가 임의로 Query를 작성한다.

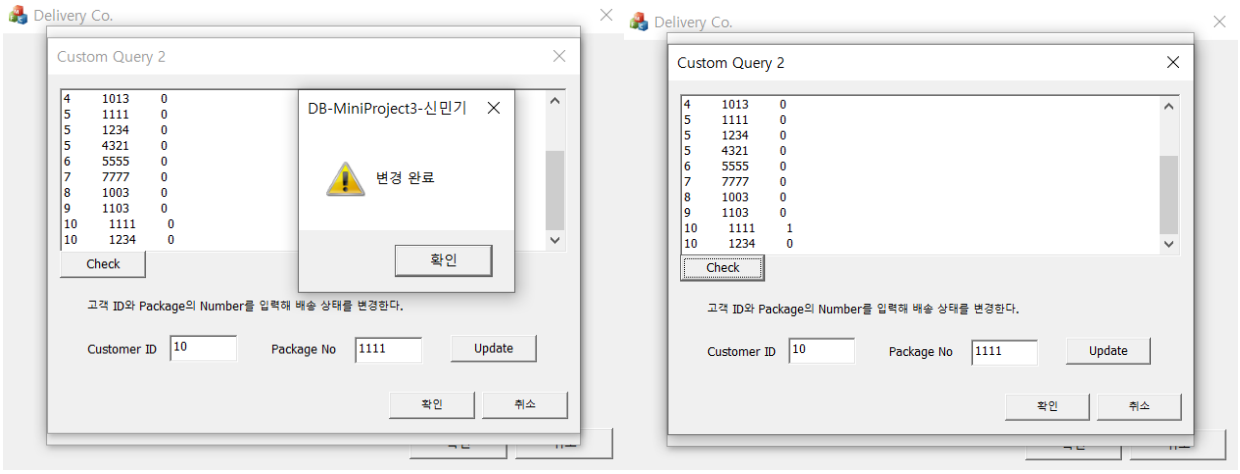


- `select b1.name, b2.name from branch as b1, branch as b2 where b1.manager = b2.id;`

- 하위 브랜치의 이름과 매니저 브랜치의 이름을 select하는 쿼리를 예시로 입력했다.

- Scenario 2: 고객 ID와 Package No를 입력받아 배송 상태를 변경한다.



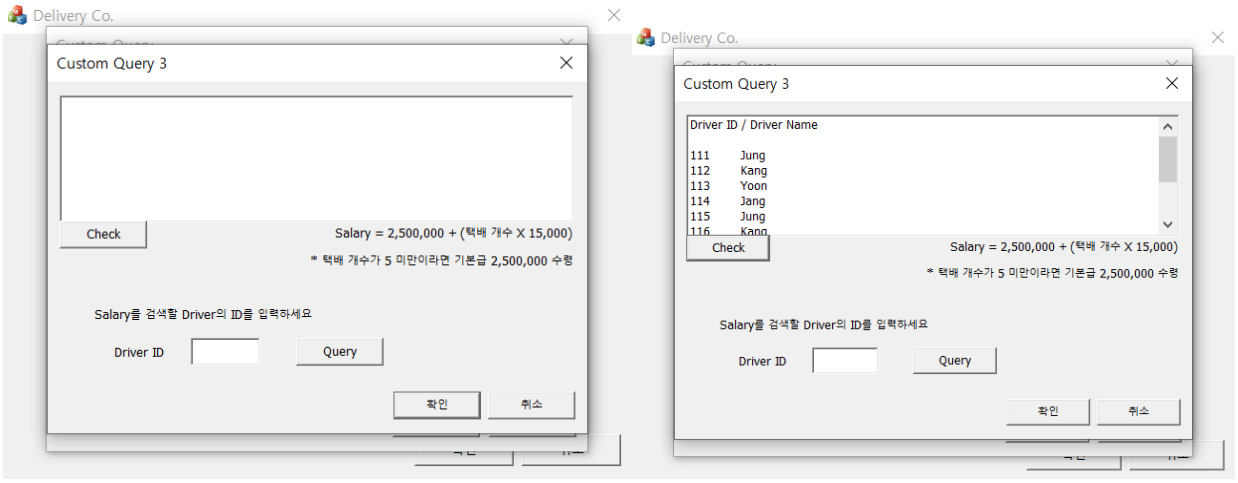


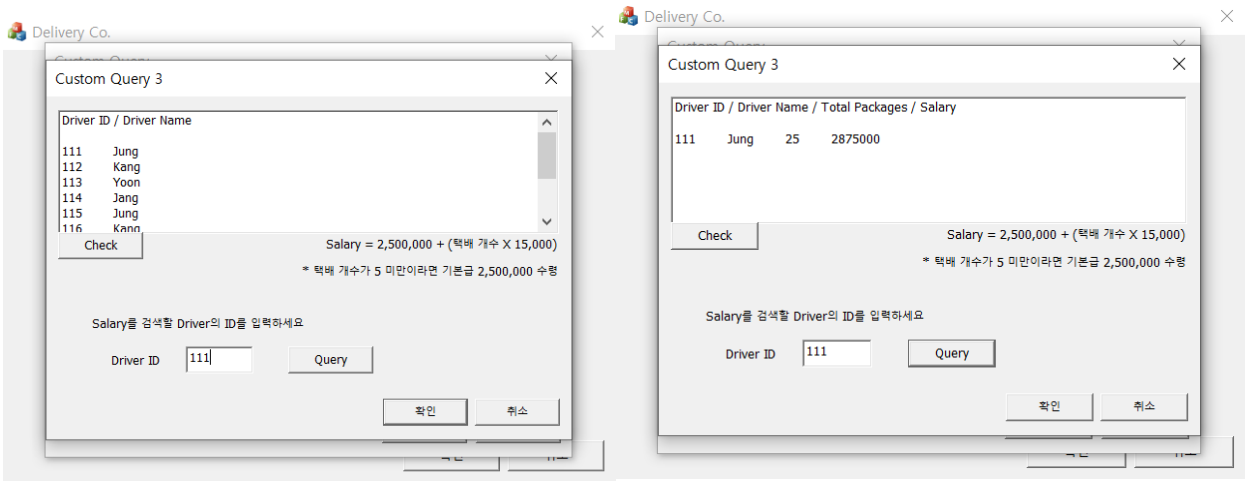
- Check: `select customer_id, pno, status from package;`

- Update: `update package set status = 1 where customer_id = '%s' and pno = '%s';`

- Check 버튼 클릭시, 택배 튜플들의 배송 상태를 select한다. 택배 튜플은 복합키이므로 pk값인 Customer ID와 Pakcage 번호를 입력값으로 받고 Update 버튼을 클릭하면 status가 0에서 1로 변경된다.

- Scenario 3: Driver의 Salary를 select한다.



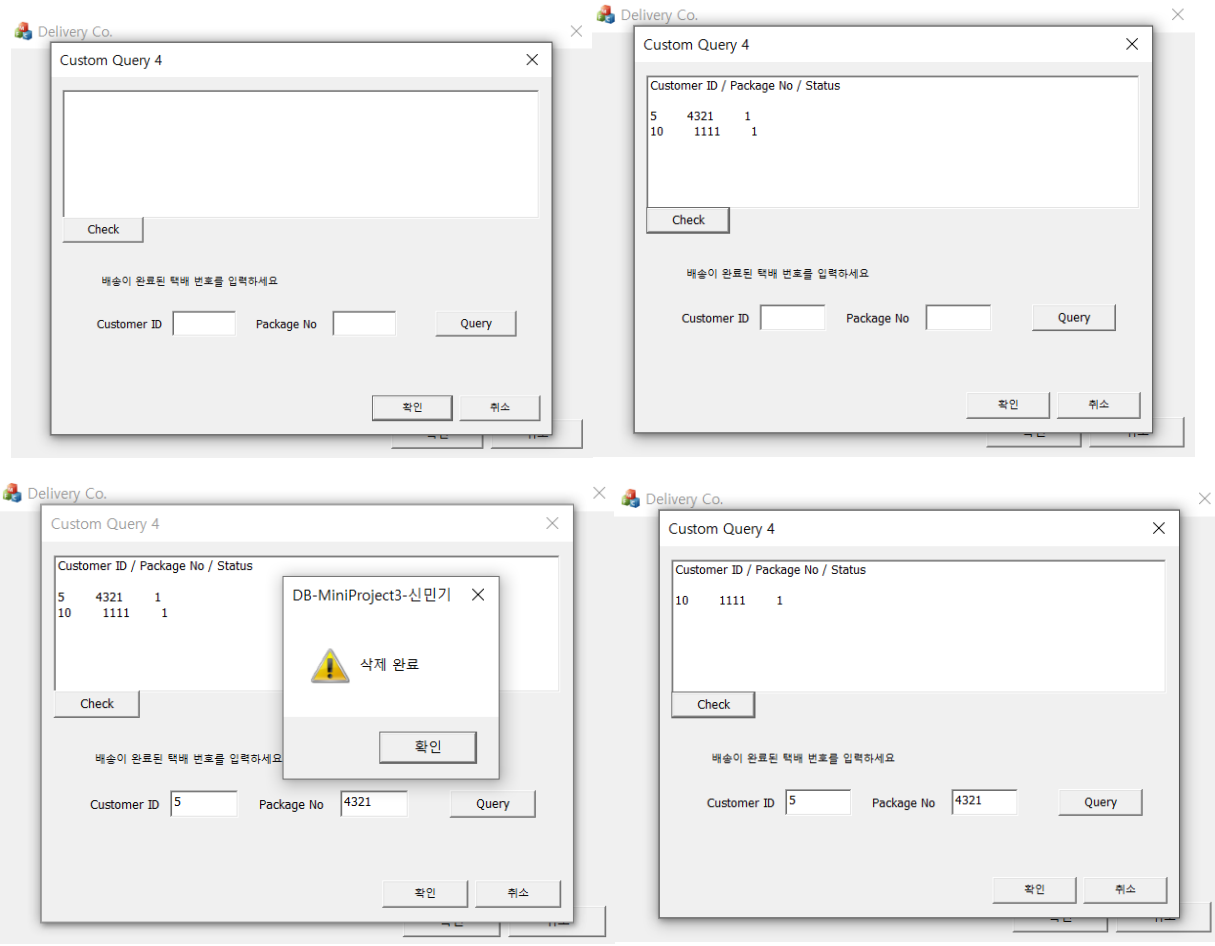


- Check: `SELECT id, name from driver;`

- Query: `SELECT id, name, total_packs, CASE WHEN total_packs >= 5 THEN 2500000 + (15000 * total_packs) ELSE 2500000 END AS Salary FROM driver where id = '%s';`

- Check 버튼 클릭시 현재 Driver의 id와 이름을 출력한다. 입력값에 driver ID를 넣고 Query 버튼을 클릭하면, Total Packages를 토대로 Salary를 계산하여 List Box에 출력한다.

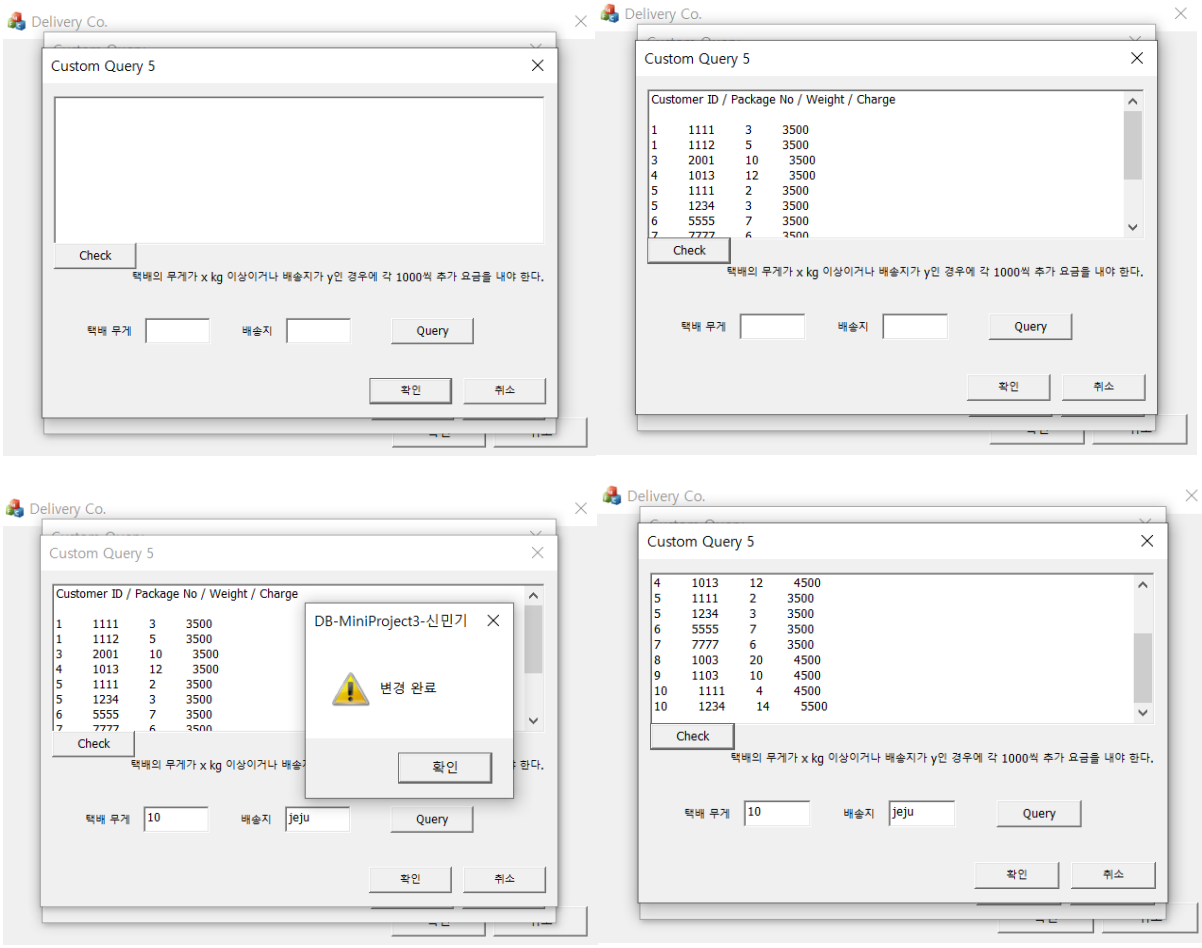
- Scenario 4: 배송이 완료된 택배를 삭제한다.



- Check: `select customer_id, pno, status from package where status = 1;`
- Query: `delete from package where customer_id = '%s' and pno = '%s';`

- Check 버튼 클릭시, 배송이 완료된 택배 튜플들을 select한다. 고객 ID와 택배 번호를 입력값으로 받고 Query 버튼을 누르면 입력값에 해당하는 택배 튜플이 삭제된다.

- Scenario 5: 추가 배송 요금을 Update한다.



- Check: `select customer_id, pno, weight, charge from package;`

- Query1: `update package set charge = charge + 1000 where weight >= '%s';`

- Query2: `update package set charge = charge + 1000 where customer_id in (select id from customer where address = '%s');`

- Check 버튼 클릭시 package 튜플의 고객 번호, 택배 번호, 무게, 요금 애트리뷰트를 select한다. 택배 무게, 배송지에 대한 입력값을 받는다. 쿼리 버튼을 클릭하면 2개의 쿼리를 순차적으로 진행한다. 1번째로 무게에 대한 요금을 Update한다. 2번째로 배송지에 대한 요금을 Update한다.