과목명: 데이터베이스 시스템

2분반

《Project #2》

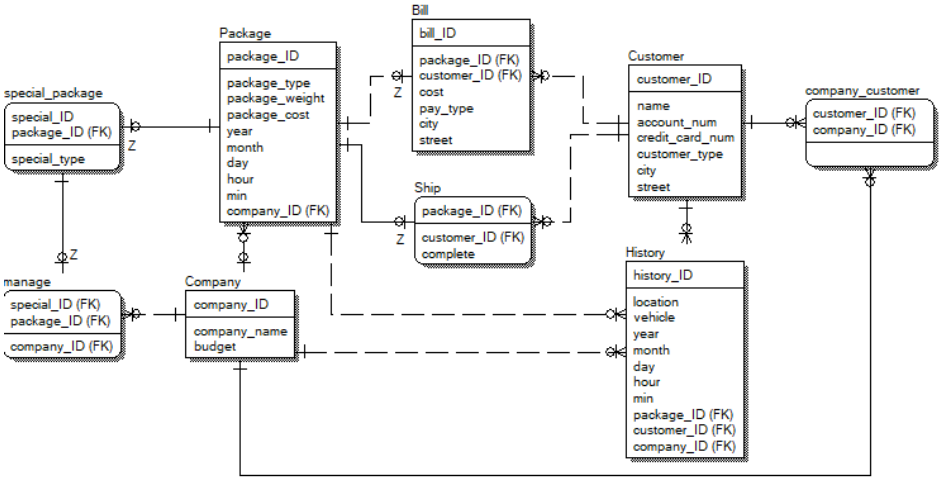
서강대학교 [컴퓨터공학과]

[20171666]

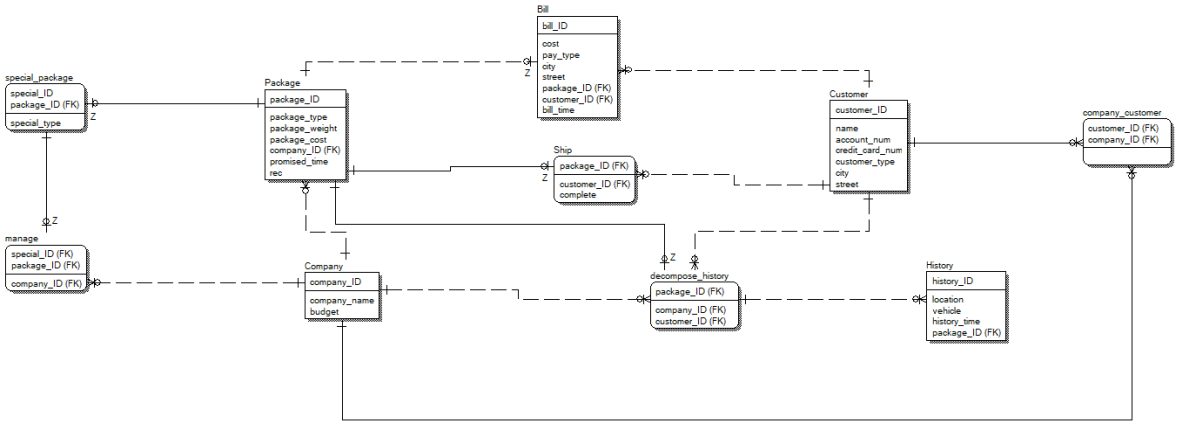
[이예은]

1. BCNF decomposition

1번 프로젝트의 relational schema는 다음과 같고,



BCNF로 decomposition을 했을 때의 schema는 다음과 같다.



전반적으로 프로젝트 1과 별다른 차이점이 없지만, history table이 decomposed_history로 분리되었다. 프로젝트 1의 각 table 별로 BCNF 과정을 살펴보자.

Bill	우선 bill은 프로젝트 1과 비교해 결제 시간을 나타내는 datetime 타입의 attribute bill_time을 추가했다. Bill의 functional dependency를 보면, Bill_ID-> customer_ID, cost, pay_type, city, street, package_ID package_ID->customer_ID, cost, pay_type, city, street, bill_ID 만 존재한다. 이 때, bill_ID는 primary key, (package_ID)+가 bill과 같으므로 package_ID는 super key가 되어 BCNF form을 만족한다. Customer_ID는 같은 customer가 여러 개의 package를 시킬 수 있고 다양한 pay_type으로 계산할 수 있으며 customer의 주소 이외의 장소에서 결제가 가능하다고 프로젝트 1에서 가정했으므로 functional dependency가 아무것도 성립하지 않는다. Cost와 pay_type, city, street, time 모두 비슷한 이유로 같은 cost, pay_type 등등일 때 같은 attribute를 가진

	다고 결정해 주지 않는다. 그러므로 bill은 이미 BCNF form을 만족해 decompose 할 필요가 없다.
Package	우선 package는 프로젝트1과 비교해서 decompose말고도 차이점이 있다. 이전 year, month등으로 composite attribute를 구분했지만, datetime 타입 변수 promised_time으로 이를 통합했다. 또한 뒤의 query를 위해 수령자 recipient_name의 attribute를 추가했다. 이제 이 때 functional dependency를 보면, Package_ID->package (package_ID는 primary key) 밖에 존재하지 않는다. 다른 attribute들은 모두 본인 이외의 attribute들을 정해주지 않는다. 예를 들어, package_type을 보면, 같은 envelope 타입에 다른 weight, cost, company_ID등을 가질 수 있다. 이는 다른 attribute들에도 동일하게 적용된다. 그러므로 functional dependency가 하나 밖에 존재하지 않고, 이 때 package_ID가 primary key이므로 BCNF를 만족한다. 그러므로 decompose 하지 않는다.
Customer	Customer의 functional dependency를 보면, Customer_ID->customer (customer_ID는 primary key) Account_num, credit_card_num->name, customer_type, city, street, customer_ID 이다. Account_num이나 credit_card_num은 null이 가능한 attribute이기 때문에 단독으로는 functional dependency를 가지지 않는다. 그러므로 둘이 합쳐져 해당 명의의 customer를 알아낼 수 있다. Name은 동명이인이 가능하다 가정했으므로 같은 이름으로 다른 customer 정보를 가질 수 있어 FD를 가지지 않는다. 다른 attribute 또한 중복이 가능하므로 다른 attribute를 정해주지 않는다. (account_num, credit_card_num)+가 customer 이므로 (account_num, credit_card_num)은 super key이다. 그러므로 customer 또한 BCNF를 만족해 decompose 하지 않는다.
Ship	ship에서의 functional dependency는 package_ID->ship (package_ID가 primary key) 밖에 없다. Customer_ID는 한 명의 customer가 여러 개의 package를 가질 수 있으므로 FD를 가지지 않고, complete 또한 배송 완료된 다양한 package를 가질 수 있어 FD가 없다. Package_ID가 primary key이므로 ship은 BCNF를 만족하고 decompose 하지 않는다.
Company	Company의 functional dependency는 Company_ID->company(company_ID가 primary key) 밖에 없다. Company_name은 중복될 수 있고 budget 또한 같은 budget에 다양한 company가 존재할 수 있다. Company_ID가 primary key이므로 company는 BCNF를 만족하고 decompose 하지 않는다.

History	History 또한 year, month 등으로 나뉜 시간정보를 datetime 타입의 history_time attribute로 통합했다. History의 functional dependency를 살펴보면, History_ID->history (history_ID가 primary key), Package_ID->customer_ID, company_ID 가 된다. 다른 attribute, location, vehicle, time등은 같은 장소, 운송수단, 시간에 다른 package 등을 가질 수 있으므로 FD가 없다. 저 functional dependency에서 (package_ID)+는 customer_ID, company_ID이고 이는 history와 같지 않으므로 package_ID는 super key가 아니다. 그러므로 (package_ID, customer_ID, company_ID)와 right side를 제거하고 left side를 추가한 (history_ID, location, vehicle, history_time, package_ID)로 decompose 된다. 여기서 (package_ID, customer_ID, company_ID) 이 부분을 decomposed_history로 이름붙여 새로운 table을 생성한다. History는 (history_ID, location, vehicle, history_time, package_ID)로 교체한다. 바꾼 이후 history를 보면 FD가 History_ID->history (history_ID가 primary key) 밖에 없으므로 BCNF form을 만족한다. 그러므로 더 decompose하지 않는다.
Decomposed_history	Decomposed_history의 FD를 보면 Package_ID->decomposed_history (package_ID가 primary key) 이므로 더 decomposed 되지 않는다.
Special_package	Special_package의 FD를 보면, Special_ID, package_ID->special_type (Special_ID, package_ID가 primary key) 이고, special_type은 같은 type이라도 다른 package를 가질 수 있으므로 FD가 없다. 그러므로 Special_ID, package_ID가 primary key 로 BCNF form을 만족하고 decompose 하지 않는다.
Manage	Manage의 FD를 보면, Special_ID, package_ID->company_ID (Special_ID, package_ID가 primary key)이고 같은 company가 다양한 special package를 가질 수 있으므로 FD가 없다. 그러므로 Special_ID, package_ID가 primary key 로 BCNF form을 만족하고 decompose 하지 않는다.
Company_customer	Company_customer는 두 attribute, company_ID, customer_ID가 모두 primary key이므로 이미 BCNF form을 만족하고 더 decompose 되지 않는다.

2. Physical schema 적용

Bill	Bill_ID, package_ID, customer_ID는 varchar(6) 타입으로 만들고 cost는 최대 8자리의 dec, pay_type, city, street는 varchar(20) 타입으로 만든다. Bill_time 은 datetime 타입으로 만든다. 전부 NULL을 허용하지 않는다.
Package	ID는 전부 varchar(6) 타입으로, package_type, recipient_name은 varchar(20), weight는 최대 4자리의 dec, cost는 최대 8자리의 dec으로 만든다. 또한 promised_time은 datetime 타입으로 만들고 전부 NULL을 허용하지 않는다.
Customer	ID는 varchar(6) 타입으로, name, account_num, credit_card_num, customer_type, city, street 모두 varchar(20) 타입으로 만든다. 이때 account_num과 credit_card_num만 NULL을 허용하고 나머지는 허용하지 않게 한다.
Ship	ID는 varchar(6)으로, complete는 varchar(10) 타입으로 만든다. 전부 NULL을 허용하지 않는다.
Company	ID는 varchar(6)으로, company_name은 varchar(20), budget은 최대 10자리의 decimal로 만든다. 전부 NULL을 허용하지 않는다.
History	ID는 varchar(6)으로, location과 vehicle은 varchar(20)으로 만든다. History_time은 datetime 타입으로 만들고 전부 NULL을 허용하지 않는다.
Decomposed_history	전부 varchar(6) 타입으로 하고 NULL을 허용하지 않는다.
Special_package	ID는 varchar(6)으로, special_type은 varchar(20)으로 만든다. NULL을 허용하지 않는다.
Manage	전부 varchar(6) 타입으로 하고 NULL을 허용하지 않는다.
Company_customer	전부 varchar(6) 타입으로 하고 NULL을 허용하지 않는다.

3. Query code

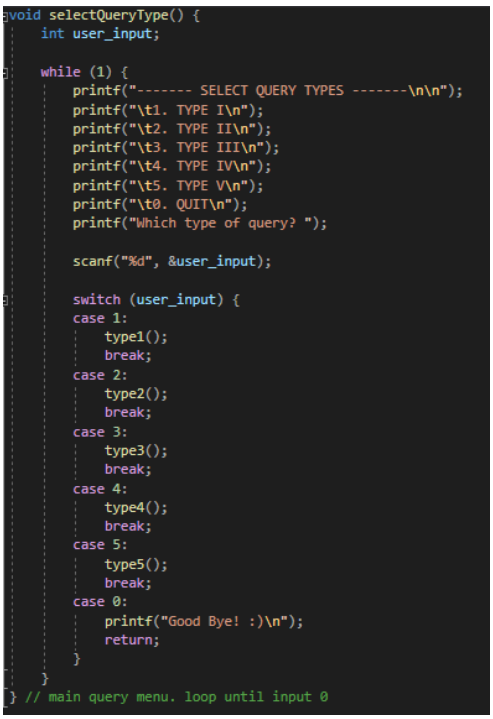
1) Create

```
void create_DB() {
    FILE* fp = fopen("20171666.txt", "r");
    char query[1000];
    int state = 0;
    int i = 0;

    while (fgets(query, sizeof(query), fp) != NULL) {
        //printf("%s", query);
        if (query[0] == 'd') {
            strcpy(drop_sql[i], query);
            i++;
        }
        else if (query[0] == 'c' || query[0] == 'i') {
            state = mysql_query(connection, query);
        }
    }

    fclose(fp);
} // create table, insert data and save drop sql queries.
```

Txt 파일을 열어 파일이 끝날 때까지 한 줄씩 읽어온다. 한 줄을 읽어서 d로 시작한다면 'drop' query이므로 drop query문을 저장하는 drop_sql 배열에 저장한다.

C나 i로 시작한다면 'creat', 'insert' query이므로 mysql_query(connection, query)를 통해 데이터베이스에 해당 query문을 보내준다. 이를 통해 데이터베이스에 table과 그에 맞는 데이터들이 생성된다.

2) Select query type



```
void selectQueryType() {
    int user_input;

    while (1) {
        printf("------- SELECT QUERY TYPES -------\n\n");
        printf("\t1. TYPE I\n");
        printf("\t2. TYPE II\n");
        printf("\t3. TYPE III\n");
        printf("\t4. TYPE IV\n");
        printf("\t5. TYPE V\n");
        printf("\t0. QUIT\n");
        printf("Which type of query? ");

        scanf("%d", &user_input);

        switch (user_input) {
        case 1:
            type1();
            break;
        case 2:
            type2();
            break;
        case 3:
            type3();
            break;
        case 4:
            type4();
            break;
        case 5:
            type5();
            break;
        case 0:
            printf("Good Bye! :)\n");
            return;
        }
    }
} // main query menu. loop until input 0
```

메인 메뉴이다. 유저가 0을 입력하기 전까지 계속 유저의 입력을 받는다. 받은 입력에 따라 알맞은 type 함수를 불러준다. 0을 입력했다면 함수를 종료한다.

3) Type 1

```c
void type1() {
    int user_input;
    int truck_num;

    printf("---- TYPE I ----\n");
    printf("** Truck 1721 is destroyed in a crash at 2020-04-13\n\n");
    // crushed at my certain time == 2020-04-13

    while (1) {
        printf("---- Subtypes in TYPE I ----\n");
        printf("\t1. TYPE I-1\n");
        printf("\t2. TYPE I-2\n");
        printf("\t3. TYPE I-3\n");
        printf("Which type of query? ");

        scanf("%d", &user_input);

        switch (user_input) {
        case 1:
            subType1();
            break;
        case 2:
            subType2();
            break;
        case 3:
            subType3();
            break;
        case 0:
            printf("\n\n");
            return;
        }
    }
} // menu type 1.
```

Type 1 query 처리를 위한 함수이다. 트럭 1721이 2020년 04월 13일에 사고가 났다고 가정한다. 이후 유저 입력으로 0이 들어오기 전까지 유저의 입력을 받는다. 유저 입력에 따라 알맞은 type 1의 sub type 함수를 불러주고 0이 들어왔다면 함수를 종료한다.

4) Type 1-1

```c
void subType1() {
    char query[1000];
    int state;

    printf("---- TYPE I-1 ----\n");
    printf("** Find all Find all customers who had a package on the truck at the time of the crash. **\n");
    //sql and print result
    strcpy(query,"select distinct customer.name from customer, decomposed_history, history \
    where customer.customer_ID = decomposed_history.customer_ID and decomposed_history.package_ID = history.package_ID \
        and history.vehicle = \'truck1721\' and date(history.history_time) = \'2020-04-13\'");

    state = mysql_query(connection, query);
    if (state == 0)
    {
        printf("Customer name: ");
        sql_result = mysql_store_result(connection);
        while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
        {
            printf("%s ", sql_row[0]);
        }
        mysql_free_result(sql_result);
    }
    printf("\n\n");
    return;
} // type 1-1.
```

Type 1-1의 query를 처리하기 위한 함수이다. 트럭 1721이 사고 났을 때 그 트럭 안에 있는 package의 customer를 얻기 위해 다음과 같은 query를 입력한다.

```sql
select distinct customer.name from customer, decomposed_history, history
where customer.customer_ID = decomposed_history.customer_ID
and decomposed_history.package_ID = history.package_ID
and history.vehicle = 'truck1721'
and date(history.history_time) = '2020-04-13'
```

이를 입력해 데이터베

이스로 보낸다. 이 쿼리의 의미는, history에서 2020-04-13에 truck1721을 타고 있는 history 정보를 뽑아 그 history의 package_ID 정보를 이용해 decomposed_history에서 해당 package의 customer_ID를 찾고 다시 이 ID를 이용해 customer에서 해당 ID의 customer name을 찾아준다. 출력은 다음과 같다.

```
---- TYPE I ----
** Truck 1721 is destroyed in a crash at 2020-04-13

---- Subtypes in TYPE I ----
        1. TYPE I-1
        2. TYPE I-2
        3. TYPE I-3
Which type of query? 1
---- TYPE I-1 ----
** Find all Find all customers who had a package on the truck at the time of the crash. **
Customer name: james jack
```

History date set, decomposed와 customer를 보면 2020-04-13인 history에서 package ID를 찾고 성공적으로 얻었다는 것을 알 수 있다.

```
insert into history values ('000300','okjunHub','truck1721','000400','2020-01-20 00:00:01');
insert into history values ('000301','mapoHub','truck3369','000401','2020-02-19 07:10:01');
insert into history values ('000302','wondagnHub','truck2245','000400','2020-01-21 00:00:01');
insert into history values ('000303','okjunHub','truck1721','000402','2020-04-13 00:00:01');
insert into history values ('000304','okjunHub','truck4567','000403','2020-01-17 00:00:01');
insert into history values ('000305','hongHub','truck1002','000403','2020-01-19 00:00:01');
insert into history values ('000306','sogangHub','truck1721','000404','2020-02-01 00:00:01');
insert into history values ('000307','sogangHub','truck8947','000405','2019-03-13 00:00:01');
insert into history values ('000308','wondangHub','truck4532','000405','2019-03-08 00:00:01');
insert into history values ('000309','seoulHub','truck8947','000406','2020-04-11 00:00:01');
insert into history values ('000310','okjunHub','truck1721','000406','2020-04-13 00:00:01');
insert into history values ('000311','daeguHub','truck1247','000407','2020-04-10 00:00:01');
insert into history values ('000312','mapoHub','truck1721','000407','2020-04-12 00:00:01');
insert into history values ('000313','busanHub','truck0041','000409','2019-02-25 00:00:01');
insert into history values ('000314','sogangHub','truck0041','000410','2020-04-03 00:00:01');

insert into decomposed_history values ('000400','000100','000200');
insert into decomposed_history values ('000401','000101','000201');
insert into decomposed_history values ('000402','000101','000200');
insert into decomposed_history values ('000403','000102','000200');
insert into decomposed_history values ('000404','000102','000200');
insert into decomposed_history values ('000405','000102','000202');
insert into decomposed_history values ('000406','000102','000203');
insert into decomposed_history values ('000407','000100','000204');
insert into decomposed_history values ('000408','000103','000204');
insert into decomposed_history values ('000409','000103','000205');
insert into decomposed_history values ('000410','000104','000206');
insert into decomposed_history values ('000411','000101','000205');
insert into decomposed_history values ('000412','000102','000207');
insert into decomposed_history values ('000413','000100','000208');
insert into decomposed_history values ('000414','000101','000209');

insert into customer values ('000200','james','123212-154521','1235-45874-4532','monthly','goyang','sesol');
insert into customer values ('000201','james','','2335-34532-1213','infrequent','mapo','sogang');
insert into customer values ('000202','mary','112212-697521','4635-12332-9832','infrequent','goyang','bakseok');
insert into customer values ('000203','jack','','1235-45874-4532','infrequent','goyang','sesol');
insert into customer values ('000204','lucy','179212-165421','7835-43574-1232','infrequent','mapo','shinchon');
insert into customer values ('000205','shen','198212-151771','','monthly','mapo','hongdae');
insert into customer values ('000206','stella','','1235-53234-4122','infrequent','goyang','zichuk');
insert into customer values ('000207','william','123212-154521','','monthly','seoul','shinchon');
insert into customer values ('000208','scar','489521-789853','4585-41874-3032','infrequent','busan','haeundae');
insert into customer values ('000209','michael','','1235-45874-4532','infrequent','goyang','madu');
insert into customer values ('000210','jane','834512-097421','4335-07784-9452','monthly','seoul','gongduck');
insert into customer values ('000211','lucy','128451-745521','','monthly','goyang','madu');
insert into customer values ('000212','ben','123845-924567','','monthly','goyang','wondang');
insert into customer values ('000213','lizz','','5635-86644-4322','infrequent','seoul','hongdae');
insert into customer values ('000214','asha','127897-045621','2735-97354-0732','monthly','seoul','itaewon');
```

5) Type 1-2

```c
void subType2() {
    int state;
    char query[1000];

    printf("---- TYPE I-2 ----\n");
    printf("** Find all recipients who had a package on that truck at the time of the crash. **\n");

    strcpy(query, "select distinct package.recipient_name from package, decomposed_history, history \
    where package.package_ID = decomposed_history.package_ID and decomposed_history.package_ID = history.package_ID\
        and history.vehicle = \'truck1721\' and date(history.history_time) = \'2020-04-13\'");

    state = mysql_query(connection, query);
    if (state == 0)
    {
        printf("Recipient name: ");
        sql_result = mysql_store_result(connection);
        while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
        {
            printf("%s ", sql_row[0]);
        }
        mysql_free_result(sql_result);
    }
    printf("\n\n");
    return;
} // type 1-2
```

Type 1-2의 query를 처리하기 위한 함수이다. 트럭 1721이 사고 났을 때 그 트럭 안에 있는 package의 recipient_name를 얻기 위해 다음과 같은 query를 입력한다.

```
select distinct package.recipient_name from package, decomposed_history, history
where package.package_ID = decomposed_history.package_ID
and decomposed_history.package_ID = history.package_ID
and history.vehicle = 'truck1721'
and date(history.history_time) ='2020-04-13'
```

큰 틀은 1-1과 같으며, customer 테이블로 가서 customer의 이름을 알아내야 했던 것과 달리 package_ID를 이용해 package에 접근해 해당 ID에 맞는 recipient_name을 알아낸다. 출력은 다음과 같다.

```
---- Subtypes in TYPE I ----
        1. TYPE I-1
        2. TYPE I-2
        3. TYPE I-3
Which type of query? 2
---- TYPE I-2 ----
** Find all recipients who had a package on that truck at the time of the crash. **
Recipient name: jerk peter
```

Package 데이터를 보면 해당 package id의 recipient를 성공적으로 얻은 걸 확인할 수 있다.

```
insert into package values ('000400','envelope','1','12000','000100','2020-01-22 00:00:01','james');
insert into package values ('000401','largebox','3','12000','000101','2020-02-21 00:10:01','ben');
insert into package values ('000402','smallbox','1','8000','000101','2020-04-15 02:00:05','jerk');
insert into package values ('000403','largebox','2','7000','000102','2020-01-18 04:00:01','elizabeth');
insert into package values ('000404','smallbox','1','11000','000102','2020-02-02 02:40:01','stella');
insert into package values ('000405','smallbox','2','10000','000102','2019-03-11 10:00:01','bob');
insert into package values ('000406','smallbox','2','200000','000102','2020-04-18 20:04:14','peter');
insert into package values ('000407','envelope','2','19000','000100','2020-04-15 00:00:01','lily');
insert into package values ('000408','envelope','1','36000','000103','2020-04-03 01:04:01','maggy');
insert into package values ('000409','largebox','4','7000','000103','2019-02-24 07:14:01','cloudy');
insert into package values ('000410','smallbox','1','28000','000104','2020-04-06 07:10:01','april');
insert into package values ('000411','largebox','5','1000','000101','2019-02-10 08:14:01','brown');
insert into package values ('000412','largebox','4','45000','000102','2020-05-12 08:03:02','lucy');
insert into package values ('000413','envelope','1','14000','000100','2020-05-20 04:10:01','anne');
insert into package values ('000414','envelope','1','12000','000101','2020-05-14 15:04:01','elsa');
```

6) Type 1-3

```
void subType3() {
    char query[1000];
    int state;

    printf("---- TYPE I-3 ----\n");
    printf("** Find the last successful delivery by that truck prior to the crash. **\n");

    strcpy(query, "select decomposed_history.package_ID, decomposed_history.customer_ID from decomposed_history, history\
    where  decomposed_history.package_ID = history.package_ID\
        and\
        history.history_time = (select max(history.history_time) from history, ship\
    where history.package_ID = ship.package_ID and\
        ship.complete = \'complete\' and\
        history.vehicle = \'truck1721\' and\
        date(history.history_time) < '2020-04-13'); ");

    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 ", sql_row[0]);
            printf("Customer ID: %s ", sql_row[1]);
        }
        mysql_free_result(sql_result);
    }
    printf("\n\n");
} // type 1-3
```

Type 1-3의 query를 처리하기 위한 함수이다. 가장 마지막으로 트럭 1721이 배달한 package의 정보를 얻기 위해 다음과 같은 query를 입력한다.

```
select decomposed_history.package_ID, decomposed_history.customer_ID from decomposed_history, history
where decomposed_history.package_ID = history.package_ID
and history.history_time = (select max(history.history_time) from history, ship
where history.package_ID = ship.package_ID
and ship.complete = 'complete'
and history.vehicle = 'truck1721'
and date(history.history_time) < '2020-04-13');
```

이 쿼리의 의미는 ship의 complete 정보가 complete로 배송이 완료된 상태이고, history의 vehicle이 truck1721, history_time이 2020-04-13보다 작은 튜플을 먼저 뽑아 이 중 가장 큰 history_time을 뽑는다. 이 얻은 정보를 원래 history_time과 비교해 해당 history_time을 가진 튜플을 뽑고, 이 튜플의 package_ID를 이용해 decomposed_history에서 package_ID, customer_ID를 얻는다. 출력은 다음과 같다.

```
---- Subtypes in TYPE I ----
        1. TYPE I-1
        2. TYPE I-2
        3. TYPE I-3
Which type of query? 3
---- TYPE I-3 ----
** Find the last successful delivery by that truck prior to the crash. **
Package ID: 000407 Customer ID: 000204
```

History 데이터에서 보면 truck1721이 2020-04-13 이전 가장 마지막으로 배송한 시각이 2020-04-12인 것을 볼 수 있고, 이 때 package id가 000407인 걸 확인할 수 있다. 또한 ship을 보면 000407이 complete 된 것을 확인할 수 있다.

```
insert into ship values ('000400','000200','complete');
insert into ship values ('000401','000201','complete');
insert into ship values ('000402','000200','uncomplete');
insert into ship values ('000403','000200','complete');
insert into ship values ('000404','000200','complete');
insert into ship values ('000405','000202','complete');
insert into ship values ('000406','000203','uncomplete');
insert into ship values ('000407','000204','complete');
insert into ship values ('000408','000204','complete');
insert into ship values ('000409','000205','complete');
insert into ship values ('000410','000206','uncomplete');
insert into ship values ('000411','000205','complete');
insert into ship values ('000412','000207','uncomplete');
insert into ship values ('000413','000208','uncomplete');
insert into ship values ('000414','000209','uncomplete');
```

7) Type 2

```
void type2() {
    char user_input[5];
    int state;
    char query[1000];

    printf("---- TYPE II ----\n");

    while (1) {
        printf("** Find the customer who has shipped the most packages in the past certain year. **\n");
        printf("Which year? ");

        scanf("%s", &user_input);
        if (strcmp(user_input,"0") == 0) {
            printf("\n");
            return;
        } // get user input
        //sql
        strcpy(query, "with countship(customer_ID, ship_count) AS \
        (select ship.customer_ID, count(*) as ship_count\
            from ship, package\
        where ship.package_ID = package.package_ID\
        and year(package.promised_time) = \'");
        strcat(query, user_input);
        strcat(query, "\'\
            group by ship.customer_ID)\
            select DISTINCT ship.customer_ID from ship, countship\
        where countship.ship_count = (select max(countship.ship_count) from countship)\
            and countship.customer_ID = ship.customer_ID");

        state = mysql_query(connection, query);
        if (state == 0)
        {
            sql_result = mysql_store_result(connection);
            while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
            {
                printf("Customer ID: %s ", sql_row[0]);
            }
            mysql_free_result(sql_result);
        }
        printf("\n\n");
    }

} // type 2.
```

Type 2를 처리하기 위한 함수이다. 특정 년도에 대한 정보를 유저의 입력으로 받고 해당 년도에 가장 많은 package를 배달시킨 customer를 찾기 위해 다음과 같은 쿼리를 보낸다.

```
with countship(customer_ID, ship_count) AS
(select ship.customer_ID, count(*) as ship_count
from ship, package
where ship.package_ID = package.package_ID
and year(package.promised_time) = 'userInput'
group by ship.customer_ID)
select DISTINCT ship.customer_ID from ship, countship
where countship.ship_count = (select max(countship.ship_count) from countship)
and countship.customer_ID = ship.customer_ID
```

이 쿼리의 의미는 우선 countship이라는 새로운 임시 테이블을 생성한다. 이 countship은 먼저 package의 promisied time을 보고 해당 년도에 배송 예정인 package들을 뽑는다. 이후 ship에서 해당 package id에 맞는 customer id가 몇 번 나왔는지 새로운 attribute ship_count에 저장한다. 이러면 countship에는 해당 년도에 package를 배송한 customer들의 id와 해당 customer가 몇 번 시켰는지의 정보가 들어있게 된다. 이제 이 countship에서 max(ship_count)를 갖고 있는 튜플을 찾는다. 이후 이 튜플의 customer id와 ship의 customer id가 같은 것을 찾아 이를 얻는다.

각각 2020년과 2019년을 입력했을 때, 000200 customer가 4번으로 가장 많고, 000205가 2번으로 가장 많으므로 다음과 같이 출력된다.

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

        1. TYPE I
        2. TYPE II
        3. TYPE III
        4. TYPE IV
        5. TYPE V
        0. QUIT
Which type of query? 2
---- TYPE II ----
** Find the customer who has shipped the most packages in the past certain year. **
Which year? 2020
Customer ID: 000200

** Find the customer who has shipped the most packages in the past certain year. **
Which year? 2019
Customer ID: 000205
```

이를 유저 입력으로 0을 받기 전까지 반복한다.

8) Type 3

```c
void type3() {
    char user_input[5];
    int state;
    char query[1000];

    printf("---- TYPE III ----\n");

    while (1) {
        printf("** Find the customer who has spent the most money on shipping in the past certain year.**\n");
        printf("Which year? ");

        scanf("%s", &user_input);
        if (strcmp(user_input, "0") == 0) {
            printf("\n");
            return;
        } // get user input
        //sql
        strcpy(query, "with payment(customer_ID, total_pay) AS \
        (select bill.customer_ID, sum(bill.cost) as total_pay\
            from bill\
        where year(bill.bill_time) = \'");
        strcat(query, user_input);
        strcat(query, "\'\
            group by bill.customer_ID)\
            select DISTINCT bill.customer_ID from bill, payment\
        where payment.total_pay = (select max(payment.total_pay) from payment)\
            and bill.customer_ID = payment.customer_ID");

        state = mysql_query(connection, query);
        if (state == 0)
        {
            sql_result = mysql_store_result(connection);
            while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
            {
                printf("Customer ID: %s ", sql_row[0]);
            }
            mysql_free_result(sql_result);
        }
        printf("\n\n");
    }
} // type 3.
```

Type 3를 처리하기 위한 함수이다. 해당 년도에 가장 많은 돈을 지불한 customer를 찾기 위해 다음과 같은 쿼리를 보낸다.

```sql
with payment(customer_ID, total_pay) AS
(select bill.customer_ID, sum(bill.cost) as total_pay
from bill
where year(bill.bill_time) = 'userInput'
group by bill.customer_ID)
select DISTINCT bill.customer_ID from bill, payment
where payment.total_pay = (select max(payment.total_pay) from payment)
and bill.customer_ID = payment.customer_ID
```

이 쿼리는 위의 type 2와 거의 비슷하다. payment라는 임시 테이블을 만들어, bill의 지불 년도가 user의 입력과 같은 bill에서 customer id에 따라 group하고 해당 customer가 사용한 cost를 더한 total_pay attribute를 만든다. 이제 이 payment 테이블에서 가장 많이 지불한 금액을 뽑고, 그 금액에 맞는 customer id를 얻는다. 각각 2020과 2019를 입력했을 때, 000203 customer가 200000원으로 가장 많이 지불했고, 000205가 2019에서 가장 많이 지불했으므로 다음과 같은 출력이 나온다.

```
---- TYPE III ----
** Find the customer who has spent the most money on shipping in the past certain year.**
Which year? 2020
Customer ID: 000203

** Find the customer who has spent the most money on shipping in the past certain year.**
Which year? 2019
Customer ID: 000205
```

이를 유저 입력으로 0을 받기 전까지 반복한다.

9) Type 4

```
void type4() {
    char user_input[5];
    int state;
    char query[1000];

    printf("---- TYPE IV ----\n");
    printf("** Find those packages that were not delivered within the promised time **\n");

    //sql
    strcpy(query, "select package.package_ID from package, history\
    where package.package_ID = history.package_ID\
    and date(package.promised_time) < date(history.history_time)\
        group by package.package_ID");

    state = mysql_query(connection, query);
    if (state == 0)
    {
        printf("Package ID: ");
        sql_result = mysql_store_result(connection);
        while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
        {
            printf("%s ", sql_row[0]);
        }
        mysql_free_result(sql_result);
    }
    printf("\n\n");

    return;
} // type 4
```

Type 4를 처리하기 위한 함수이다. Package의 promised time보다 더 늦게 도착한 package의 정보를 얻기 위해 해당 쿼리를 보낸다.

```
select package.package_ID from package, history
where package.package_ID = history.package_ID
and date(package.promised_time) < date(history.history_time)
group by package.package_ID
```

이 쿼리는 package와 history에서 정보를 얻는다. 먼저 package의 promised time보다 큰 history time이 있을 경우 제 시간에 도착한 package가 아니므로 이를 얻으면 된다. 그러므로 package의 promised time과 history time을 비교해 history time이 더 큰 튜플을 얻고, 이 package id를 얻는다. 출력 결과는 다음과 같다.

```
---- TYPE IV ----
** Find those packages that were not delivered within the promised time **
Package ID: 000403 000405 000409
```

10) Type 5

```c
void type5() {
    char id[20];
    char year[10];
    char month[10];
    int state;
    char query[1000];
    FILE* outf = fopen("bill.txt", "w");

    getchar();
    printf("---- TYPE V ----\n");
    printf("** Generate the bill for each customer for the past certain month. **\n");
    printf("Customer ID : ");
    scanf("%[^\n]s", id);
    getchar();
    printf("Which month(YYYY-MM)? ");
    scanf("%[^\n]s", year);
    strcpy(month, year+5); // get user input and divided into year and month
    year[4] = '\0';
    //sql

    strcpy(query,"select customer.name, customer.city, customer.street, sum(bill.cost)\
        from bill, customer\
    where customer.customer_ID = \'");
    strcat(query, id);
    strcat(query, "\'\
        and customer.customer_ID = bill.customer_ID\
        and year(bill.bill_time) = \'");
    strcat(query, year);
    strcat(query, "\'\
        and month(bill.bill_time) = \'");
    strcat(query, month);
    strcat(query, "\'\
        group by customer.customer_ID");
```

```c
    state = mysql_query(connection, query);
    if (state == 0)
    {
        fprintf(outf, "Customer name\tAddress-city\tstreet\tAmount\n");
        sql_result = mysql_store_result(connection);
        while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
        {
            fprintf(outf,"%s\t\t%s\t\t%s\t\t%s\n", sql_row[0], sql_row[1], sql_row[2], sql_row[3]);
        }
        mysql_free_result(sql_result);
    }
    fprintf(outf,"\n"); // print simple bill

    strcpy(query, "select customer.name, bill.pay_type, customer.city, customer.street, sum(bill.cost)\
        from bill, customer\
    where customer.customer_ID = \'");
    strcat(query, id);
    strcat(query, "\'\
        and customer.customer_ID = bill.customer_ID\
        and year(bill.bill_time) = \'");
    strcat(query, year);
    strcat(query, "\'\
        and month(bill.bill_time) = \'");
    strcat(query, month);
    strcat(query, "\'\
        group by bill.pay_type");
    state = mysql_query(connection, query);
    if (state == 0)
    {
        fprintf(outf, "=========================================================================\n");
        fprintf(outf, "Customer name\tPay type\tAddress-city\tstreet\tAmount\n");
        sql_result = mysql_store_result(connection);
        while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
        {
            fprintf(outf, "%s\t\t%s\t\t%s\t\t%s\t\t%s\n", sql_row[0], sql_row[1], sql_row[2], sql_row[3], sql_row[4]);
        }
        mysql_free_result(sql_result);
    }
    fprintf(outf, "\n"); // print bill with pay type
```

```
    strcpy(query, "select package.package_ID, bill.cost, package.package_type, bill.pay_type, ship.complete\
        from bill, package, ship\
    where ship.customer_ID= \'");
    strcat(query, id);
    strcat(query, "\'\
        and ship.customer_ID=bill.customer_ID\
        and year(bill.bill_time) = \'");
    strcat(query, year);
    strcat(query, "\'\
        and month(bill.bill_time) = \'");
    strcat(query, month);
    strcat(query, "\'\
        and package.package_ID=bill.package_ID\
        and package.package_ID = ship.package_ID");

    state = mysql_query(connection, query);
    if (state == 0)
    {
        fprintf(outf, "=================================================================\n");
        fprintf(outf, "Package ID\tCost\tPackage type\tPay type\tDelivery complete\n");
        sql_result = mysql_store_result(connection);
        while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
        {
            fprintf(outf, "%s\t\t%s\t\t%s\t\t%s\t\t\t%s\n", sql_row[0], sql_row[1], sql_row[2], sql_row[3], sql_row[4]);
        }
        mysql_free_result(sql_result);
    }
    fprintf(outf, "\n"); // print itemize billing

    fclose(outf);
    printf("Generation Completed\n\n");

    return;
} // type 5
```

Type 5를 처리하기 위한 함수이다. 가장 먼저 유저 입력으로 bill을 얻으려는 customer의 id를 입력받는다. 그리고 특정 년도와 달에 대한 정보를 입력으로 받는다. 이를 다음 쿼리에서 사용하기 편하게 하기 위해 년도를 분리해 year 변수에 넣고, 달을 분리해 month 변수에 넣는다. 이후 첫 번째, simple bill 출력을 위해 다음과 같은 쿼리를 보낸다.

```
select customer.name, customer.city, customer.street, sum(bill.cost)
from bill, customer
where customer.customer_ID = 'userInput'
and customer.customer_ID = bill.customer_ID
and year(bill.bill_time) = 'userInput'
and month(bill.bill_time) = 'userInput'
group by customer.customer_ID
```

bill time의 년도와 달이 맞는 튜플을 뽑고, 그 중에서 받은 customer id와 같은 id를 가지는 튜플을 뽑는다. 이 튜플에서 customer의 이름과 주소, customer id로 group하고 sum을 해서 얻은 해당 달에 사용한 총 지불 금액을 얻는다.

Bill with pay type을 얻기 위해 다음과 같은 쿼리를 보낸다.

```
select customer.name, bill.pay_type, customer.city, customer.street, sum(bill.cost)
from bill, customer
where customer.customer_ID = 'userInput'
and customer.customer_ID = bill.customer_ID
and year(bill.bill_time) = 'userInput'
and month(bill.bill_time) = 'userInput'
group by bill.pay_type
```

위와 큰 틀은 동일하다. 다만 group을 pay type으로 하여 해당 customer의 pay type 별 총 금액

과 pay type을 추가로 얻는다.

마지막 itemize billing을 얻기 위해 다음과 같은 쿼리를 보낸다.

```
select package.package_ID, bill.cost, package.package_type, bill.pay_type, ship.complete
from bill, package, ship
where ship.customer_ID= 'userInput'
and ship.customer_ID=bill.customer_ID
and year(bill.bill_time) = 'userInput'
and month(bill.bill_time) = 'userInput'
and package.package_ID=bill.package_ID
and package.package_ID = ship.package_ID
```

특정 년도와 달에 맞는 bill 튜플을 얻고, package id에 맞는 package 튜플과 ship 튜플을 찾아 package id와 bill cost, package type, pay type, ship 완료 여부를 얻는다. 이 3가지 쿼리를 돌려 얻은 정보를 'bill.txt' 파일에 입력한다. 이후 메인 메뉴로 돌아간다. Bill.txt의 출력은 다음과 같다.

```
---- TYPE V ----
** Generate the bill for each customer for the past certain month. **
Customer ID : 000204
Which month(YYYY-MM)? 2020-04
Generation Completed
```

```
Customer name   Address-city    street  Amount
lucy        mapo        shinchon        57000

=========================================================================
Customer name   Pay type    Address-city    street  Amount
lucy        prepaid     mapo        shinchon        21000
lucy        infrequent      mapo        shinchon        36000

=========================================================================
Package ID  Cost    Package type    Pay type    Delivery complete
000407      21000       envelope        prepaid     complete
000408      36000       envelope        infrequent      complete
```

11) Drop

```
void drop_DB() {
    int i = 0;
    int state = 0;

    while (1) {
        if (drop_sql[i][0] == '0')
            break;
        //printf("%s", drop_sql[i]);
        state = mysql_query(connection, drop_sql[i]);
        i++;
    }
} // drop the tables I make
```

테이블을 drop 하는 함수이다. 다른 테이블과 중복되어 문제가 생기는 것을 방지하기 위해 drop_sql 배열을 하나씩 순회하며 이전 create_DB 함수에서 얻은 drop query를 수행한다. Drop_sql의 내용이 0으로 초기화 된 상태를 만나면 drop query의 끝에 도달한 것이므로 함수를 종료한다.

12) Main

```
int main(void) {
    if (mysql_init(&conn) == NULL)
        printf("mysql_init() error!");

    connection = mysql_real_connect(&conn, host, user, pw, db, 3306, (const char*)NULL, 0);
    if (connection == NULL)
    {
        printf("%d ERROR : %s\n", mysql_errno(&conn), mysql_error(&conn));
        return 1;
    } // connect to database

    create_DB();

    printf("***********************************************\n");
    printf("** Sogang Package Delivery Management System **\n");
    printf("***********************************************\n\n\n");

    selectQueryType();
    drop_DB();

    mysql_close(connection);
    return 0;
}
```

메인 함수이다. 가장 먼저 해당 데이터베이스 이름과 패스워드, host등을 이용해 데이터베이스와 연결한다. 이후 create_DB 함수를 불러 테이블을 생성하고 selectQueryType 함수를 불러 각 타입별 쿼리를 수행할 수 있도록 한다. selectQueryType 함수가 끝났다면 drop_DB 함수를 불러 create 해줬던 테이블들을 삭제해준다. 이후 데이터베이스와 연결을 끊고 프로그램을 종료한다.

4. Simplified test

이전 history를 decomposed_history와 history를 생각해보자.

History_ID->history (history_ID가 primary key)

Package_ID->customer_ID, company_ID

가 있는데, BCNF 여부를 판단하기 위해 F+를 전부 확인하지 않고, simplified test로 현재 있는 F만 확인한다. F에서 BCNF를 violate하는 것이 하나도 없다면 F+에도 없는 것이므로 F만 확인하는 것이다. 그러므로 해당 history만 확인하면 (package_ID)+는 (package_ID, customer_ID, company_ID)로 history와 같지 않아 BCNF가 아니다. 따라서 BCNF 알고리즘에 의해 나누면 R1(package_ID, customer_ID, company_ID), R2(history_ID, location, vehicle, history_time, package_ID) 로 나뉘게 된다. 이제 나눠진 것을 살펴보면, R1의 FD는 Package_ID->customer_ID, company_ID로 (package_ID)+ = R1이 되어 super key가 된다. 그러므로 BCNF를 만족한다. R2의 FD는

history_ID ->location, vehicle, history_time, package_ID 로 (history_ID)+ = R2가 되어 super key가 된다. 그러므로 R2도 BCNF를 만족한다. 각 R1, R2 모두 F+를 전부 보지 않고 F만으로 확인했을 때, BCNF를 violate 하지 않으므로 둘 다 BCNF form을 만족한다.