

데이터베이스시스템

Project 2 report

A Real-Estate Office

Spring 2024

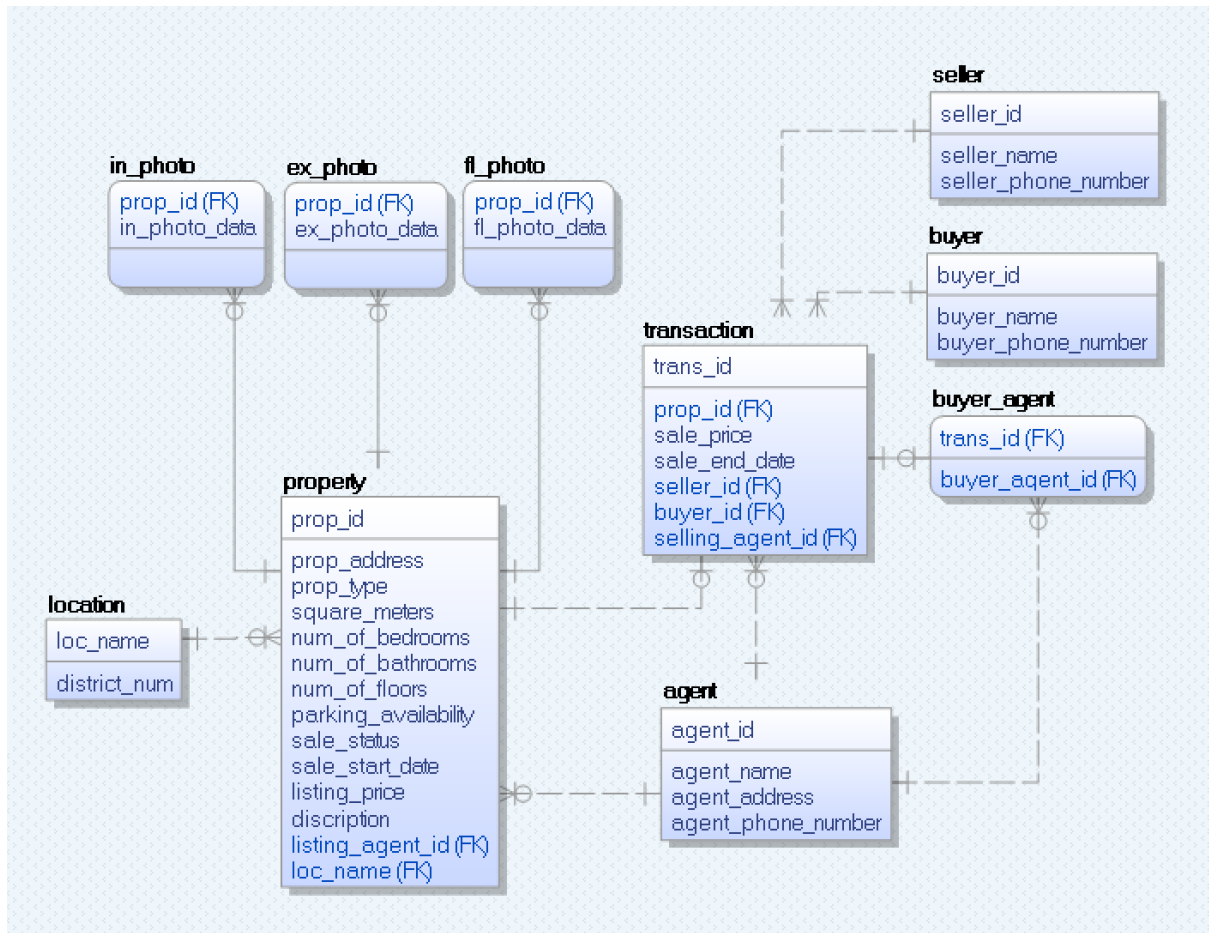
경제학과 4학년

20200562 신서영

목차

1. Decomposed Logical Schema Diagram.....	3
2. Physical Schema diagram.....	5
3. ODBC implementation within MYSQL.....	10

1. Decomposed Logical Schema Diagram



후술할 한 가지 부분을 제외하고는 프로젝트1의 Logical Schema Diagram과 동일하다.

1) property

FD : {prop_id -> 전부}

prop_id가 슈퍼키이기 때문에 BCNF를 만족한다.

2) location

FD : {loc_name -> district_num}

loc_name이 슈퍼키이기 때문에 BCNF를 만족한다.

프로젝트 1과 비교하여 수정된 entity이다. 원래는 office_of_edu entity를 추가적으로 가지고 있었으나 FD인 {district_num -> office_of_edu, office_of_edu -> district_num}은 BCNF를 만족하지 않을 뿐더러 서로 1대1 대응되는 중복이 심한 데이터이기 때문에 삭제하였다.

3) transaction

FD : {tran_id -> 전부, prop_id -> 전부}

trans_id와 prop_id는 슈퍼키이기 때문에 BCNF를 만족한다.

4) seller

FD : {seller_id -> 전부, seller_phone_number -> 전부}

seller_id와 seller_phone_number는 슈퍼키이기 때문에 BCNF를 만족한다.

5) buyer

FD : {buyer_id -> 전부, buyer_phone_number -> 전부}

buyer_id와 buyer_phone_number는 슈퍼키이기 때문에 BCNF를 만족한다.

6) agent

FD : {agent_id -> 전부, agent_phone_number -> 전부}

agent_id와 agent_phone_number는 슈퍼키이기 때문에 BCNF를 만족한다.

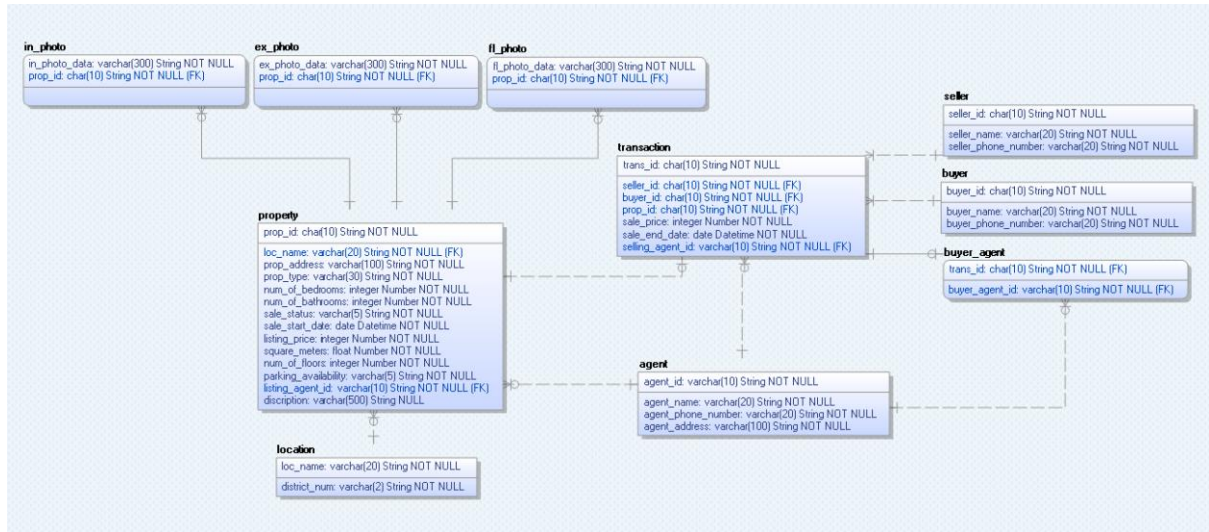
한 장소에서 두 명의 agent가 활동할 수 있기 때문에 agent_address는 결정자로 취급하지 않았다.

7) in_photo, ex_photo, fl_photo

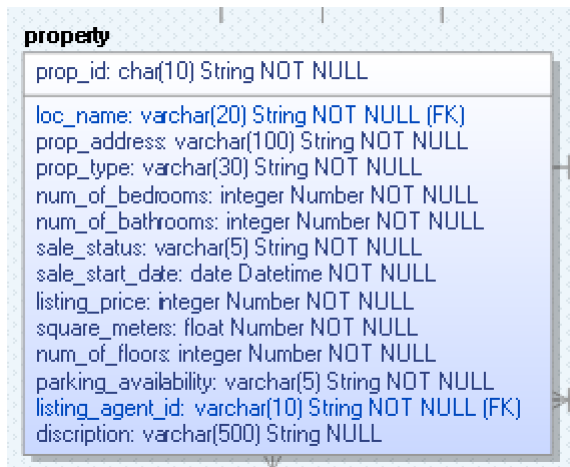
FD : X (non-trivial한 FD가 존재하지 않는다.)

따라서 BCNF이다.

2. Physical Schema diagram



1) property



부동산 schema이다. description을 제외한 모든 항목이 NULL값을 허용하지 않는다.

prop_id : 부동산을 고유하게 식별하기 위한 PK이다. domain은 char[10]이며, 10자리 숫자를 문자열로 저장한다.

loc_name : parent의 속성을 상속한다.

prop_address : 부동산의 주소이다. domain은 varchar[100]이며 가변 길이의 문자열로 저장된다.

prop_type : 부동산의 타입으로, studio, one-bedroom, multi-bedroom apartments or detached houses가 들어올 수 있다. domain은 varchar[30]이며 가변 길이의 문자열로 저장된다.

num_of_bedrooms : 침실 개수 정보를 저장한다. domain은 integer이다.

num_of_bathrooms : 욕실 개수 정보를 저장한다. domain은 integer이다.

sale_status : 판매 여부 정보를 저장한다. domain은 varchar[5]이다. true는 판매중, false는 판매 완료를 의미한다.

sale_start_date : 판매가 시작된 날짜 정보를 저장한다. domain은 DATE이다.

listing_price : 판매중인 가격을 저장한다. domain은 integer이다.

square_meters : 부동산 면적 정보를 저장한다. domain은 float으로 지정해주었다.

num_of_floors : 층수 정보를 저장한다. domain은 integer이다.

parking_availability : 주차 가능 여부를 저장한다. domain은 varchar[5]이다. true이면 주차 가능, false이면 주차 불가하다.

listing_agent_id : 부동산을 올린 agent의 id이다. parent의 속성을 상속한다.

description : 추가 설명이 있는 경우 저장한다. domain은 넉넉하게 varchar[500]으로 지정해주었다. 추가 설명이 없을 수 있으므로 NULL을 허용한다.

2) location



location	
loc_name:	varchar(20) String NOT NULL
district_num:	varchar(2) String NOT NULL

지역 정보를 담는 schema이다.

loc_name : 지역구 정보를 저장한다. PK이므로 null을 허용하지 않는다. domain은 varchar[20]이다. 영문명으로 저장하기 때문에 문자열 길이를 넉넉하게 잡았다.

district_num : 구역 정보를 저장한다. 지역구에 대응하는 구역 번호가 반드시 필요하기 때문에 null을 허용하지 않는다. 1구역부터 11구역까지 있으므로 domain은 varchar[2]로 설정해주었다.

3) transaction

transaction	
trans_id	char(10) String NOT NULL
seller_id	char(10) String NOT NULL (FK)
buyer_id	char(10) String NOT NULL (FK)
prop_id	char(10) String NOT NULL (FK)
sale_price	integer Number NOT NULL
sale_end_date	date Datetime NOT NULL
selling_agent_id	varchar(10) String NOT NULL (FK)

거래 정보를 담는 schema이다. 모든 속성이 null을 허용하지 않는다.

trans_id : 거래 id이다. PK이므로 null을 허용하지 않는다. domain은 char[10]이며, 10자리 숫자를 문자열로 저장한다.

seller_id, buyer_id, prop_id : 구매자, 판매자, 부동산 id이다. parent의 속성을 상속한다.

sale_price : 실제로 판매된 가격 정보를 저장한다. 반드시 필요한 정보이기 때문에 null을 허용하지 않는다. 비교 연산이 가능해야 하므로 domain을 integer로 저장해주었다.

sale_end_date : 판매 완료 날짜 정보를 저장한다. 반드시 필요한 정보이기 때문에 null을 허용하지 않는다. domain은 date이다.

selling_agent_id : 판매자측 agent의 id이다. parent의 속성을 상속한다.

4) seller

seller	
seller_id	char(10) String NOT NULL
seller_name	varchar(20) String NOT NULL
seller_phone_number	varchar(20) String NOT NULL

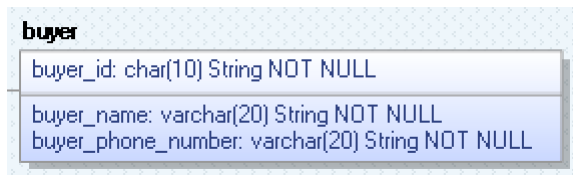
판매자 정보를 담는 schema이다.

seller_id : 판매자 id이다. PK이기 때문에 null을 허용하지 않는다. domain은 char[10]이며, 10자리 숫자를 문자열로 저장한다.

seller_name : 판매자 이름을 저장한다. 반드시 필요한 정보이기 때문에 null을 허용하지 않는다. domain은 varchar[20]이다.

seller_phone_number : 판매자의 휴대폰 번호를 저장한다. 본 db는 한 명의 판매자가 하나의 휴대폰 번호만 등록 가능하도록 디자인되었다. 판매자가 가입할 때 중복 가입을 방지하기 위한 수단으로 사용되므로 null을 허용하지 않는다. domain은 varchar[20]이다.

5) buyer



buyer	
buyer_id:	char(10) String NOT NULL
buyer_name:	varchar(20) String NOT NULL
buyer_phone_number:	varchar(20) String NOT NULL

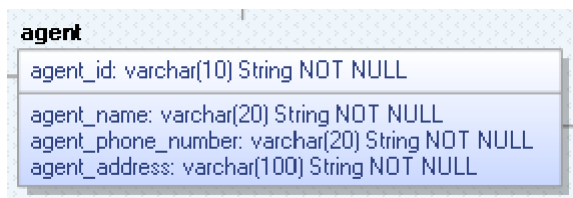
구매자 정보를 담는 schema이다.

buyer_id : 구매자 id이다. PK이기 때문에 null을 허용하지 않는다. domain은 char[10]이며, 10자리 숫자를 문자열로 저장한다.

buyer_name : 구매자 이름을 저장한다. 반드시 필요한 정보이기 때문에 null을 허용하지 않는다. domain은 varchar[20]이다.

buyer_phone_number : 구매자의 휴대폰 번호를 저장한다. 본 db는 한 명의 구매자가 하나의 휴대폰 번호만 등록 가능하도록 디자인되었다. 구매자가 가입할 때 중복 가입을 방지하기 위한 수단으로 사용되므로 null을 허용하지 않는다. domain은 varchar[20]이다.

6) agent



agent	
agent_id:	varchar(10) String NOT NULL
agent_name:	varchar(20) String NOT NULL
agent_phone_number:	varchar(20) String NOT NULL
agent_address:	varchar(100) String NOT NULL

공인중개사 정보를 담은 schema이다. 공인중개사 또는 공인중개사무소 정보를 저장한다.

agent_id : agent id이다. PK이기 때문에 null을 허용하지 않는다. domain은 char[10]이며, 10자리 숫자를 문자열로 저장한다.

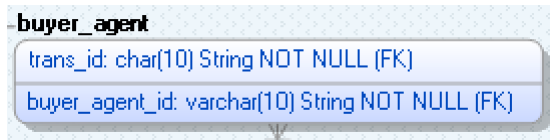
agent_name : agent 이름을 저장한다. 반드시 필요한 정보이기 때문에 null을 허용하지 않는다. domain은 varchar[20]이다.

agent_phone_number : 구매자의 휴대폰 번호를 저장한다. 본 db는 한 명의 agent가 하나의 휴대폰 번호만 등록 가능하도록 디자인되었다. agent가 가입할 때 중복 가입을 방지하기 위한 수단으로 사용되므로 null을 허용하지 않는다. domain은 varchar[20]이다.

agent_address : 공인중개사 주소를 저장한다. 중개사무소 주소는 반드시 필요하므로 null

을 허용하지 않는다. domain은 varchar[100]이다.

7) buyer_agent



buyer_agent
trans_id: char(10) String NOT NULL (FK)
buyer_agent_id: varchar(10) String NOT NULL (FK)

trans_id : 거래 번호이다. 다른 schema의 PK를 참조하는 FK이므로 null을 허용하지 않는다. parent의 domain을 상속한다.

buyer_agent_id : 구매자측 agent의 id이다. 다른 schema의 PK를 참조하는 FK이므로 null을 허용하지 않는다. parent의 domain을 상속한다.

8) in_photo, ex_photo, fl_photo



in_photo
in_photo_data: varchar(300) String NOT NULL
prop_id: char(10) String NOT NULL (FK)

in_photo_data : 사진 url을 저장하는 속성이다. null을 허용해야 한다고 생각할 수 있지만 property에 꼭 대응되는 schema가 아니며 photo가 존재하지 않는다면 테이블 자체가 존재하지 않기에 null을 허용하지 않는다. domain은 varchar[300]으로 넉넉하게 잡아주었다.

prop_id : property id이다. 다른 schema의 PK를 참조하는 FK이므로 null을 허용하지 않는다. parent의 domain을 상속한다.

ex_photo와 fl_photo에 대한 설명은 생략한다.

3. ODBC implementation within MYSQL

Visual Studio 2022 환경에서 C에 SQL을 임베딩하였다. mysql 라이브러리에 내장된 함수를 이용하여 db에서 데이터를 불러와서 수정/추가/삭제를 진행하였다.

코드는 while문 안에서 scanf 함수를 통해 실행할 쿼리 번호를 입력받고 서브쿼리의 경우 중첩된 while문을 통하여 쿼리 번호를 입력받아 실행하는 패턴으로 구성하였다.

```
while (true) {
    printf("-----SELECT QUERY TYPES-----\n");
    printf("\n");
    printf("    1. TYPE 1\n");
    printf("    2. TYPE 2\n");
    printf("    3. TYPE 3\n");
    printf("    4. TYPE 4\n");
    printf("    5. TYPE 5\n");
    printf("    6. TYPE 6\n");
    printf("    7. TYPE 7\n");
    printf("    0. QUIT\n");
    scanf("%d", &choice);
    if (!choice) break;
    else if (choice == 1) {
        query = "select * from property where sale_status='TRUE' and loc_name='Mapo'";
        state = mysql_query(connection, query);
        if (state == 0) {
            sql_result = mysql_store_result(connection);
            printf("** Fine the address of homes for sale in the district Mapo **\n");
            while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
                printf("%s %s %s %s\n", sql_row[0], sql_row[1], sql_row[2], sql_row[3], sql_row[8]);
            mysql_free_result(sql_result);
        }
        printf("\n");
    }
}
```

이해를 돕기 위한 예시는 위와 같다.

while문 안에서 메인메뉴를 출력한 뒤 0이면 break를 걸어 while문 밖으로 나가고, 1~7번이면 쿼리를 실행한다.

1) TYPE 1 : Find address of homes for sale in the district “Mapo”.

```

else if (choice == 1) {
    query = "select * from property where sale_status='TRUE' and loc_name='Mapo';";
    state = mysql_query(connection, query);
    if (state == 0) {
        sql_result = mysql_store_result(connection);
        printf("** Fine the address of homes for sale in the district Mapo **\n");
        while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
            printf("%s %s %s %s\n", sql_row[0], sql_row[1], sql_row[2], sql_row[3], sql_row[8]);
        mysql_free_result(sql_result);
    }
    printf("\n");
}

```

select * from property where sale_status='TRUE' and loc_name='Mapo';

마포구에서 판매중인 집의 주소를 찾는 기본적인 쿼리이다. property 테이블 안에 판매상태와 지역구 데이터가 모두 저장되어 있기 때문에 별도의 join 필요없이 조건 두 개를 걸어서 간단히 구할 수 있다.

코드의 흐름은 다음과 같다. mysql_query함수를 통해 state에 mysql_query의 반환값을 받고 성공적으로 쿼리가 실행되었다면(반환값이 0이라면) mysql_store_result함수를 통해 결과값을 받아온다. 받아온 결과값에서 mysql_fetch_row함수를 통해 한 행씩 fetch해온 후 한 열씩 차례로 출력한다.

1.1) TYPE 1-1 : Then find the costing between ₩1,000,000,000 and ₩1,500,000,000.

```

while (true) {
    printf("----- Subtypes in TYPE 1 ----- \n");
    printf("1. TYPE 1-1\n");
    scanf("%d", &choice);
    if (choice == 0)
        break;
    else if (choice == 1) {
        query = "select * from property where sale_status='TRUE' and listing_price >= 1000000000 and listing_price <= 1500000000;";
        state = mysql_query(connection, query);
        if (state == 0) {
            sql_result = mysql_store_result(connection);
            printf("** Find the costing between ₩1,000,000 and ₩1,500,000,000 **\n");
            while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
                printf("%s %s %s %s\n", sql_row[0], sql_row[1], sql_row[2], sql_row[3], sql_row[8]);
            mysql_free_result(sql_result);
        }
    }
    printf("\n");
}
}

```

select * from property where sale_status='TRUE' and listing_price >= 1000000000 and listing_price <= 1500000000;

쿼리 1-1은 쿼리 1에서 이어지는 서브쿼리로 해석하고 1억에서 1억 5천 사이의 가격대에 판매중인 마포구 부동산의 주소를 찾는 쿼리를 작성하였다. property 테이블 안에 판매상태와 가격 데이터가 모두 저장되어 있기 때문에 별도의 join 필요없이 조건 세 개를 걸어 간단히 구할 수 있다.

코드의 흐름은 TYPE1과 유사하므로 설명을 생략하겠다.

2) TYPE 2 : Find the address of homes for sale in the 8th school district.

```
else if (choice == 2) {
    query = "select * from property natural join location where sale_status='TRUE' and district_num = '8'";
    state = mysql_query(connection, query);
    if (state == 0) {
        sql_result = mysql_store_result(connection);
        printf("** Find the address of homes for sale in the 8th school district. **\n");
        while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
            printf("%s %s %s %s\n", sql_row[0], sql_row[1], sql_row[2], sql_row[3]);
        mysql_free_result(sql_result);
    }
    printf("\n");
}
```

select * from property natural join location where sale_status='TRUE' and district_num = '8';

처음으로 join을 필요로 하는 쿼리가 등장하였다. 학군 정보는 location table에 저장되어 있으므로 property와 join을 하고 조건을 두 개 걸어서 8학군에 위치한 판매중인 부동산의 주소를 구했다.

마찬가지로 코드의 흐름은 TYPE1과 유사하므로 설명을 생략하겠다.

2.1) TYPE 2-1 : Then find properties with 4 or more bedrooms and 2 bathrooms.

```
while (true) {
    printf("----- Subtypes in TYPE 2 ----- \n");
    printf(" 1. TYPE 2-1\n");
    scanf("%d", &choice);
    if (choice == 0)
        break;
    else if (choice == 1) {
        query = "select * from property natural join location where sale_status='TRUE' and district_num = '8' and num_of_bedrooms >= 4 and num_of_bathrooms = 2";
        state = mysql_query(connection, query);
        if (state == 0) {
            sql_result = mysql_store_result(connection);
            printf("** Then find properties with 4 or more bedrooms and 2 bathrooms. **\n");
            while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
                printf("%s %s %s %s\n", sql_row[0], sql_row[1], sql_row[2], sql_row[3]);
            mysql_free_result(sql_result);
        }
    }
    printf("\n");
}
```

select *

from property natural join location

where sale_status='TRUE' and district_num = '8'

and num_of_bedrooms >= 4 and num_of_bathrooms = 2;

쿼리 2-1 또한 쿼리 2에서 이어지는 서브쿼리로 해석하고 쿼리 2의 두 조건을 적용하고 새롭게 추가된 두 조건을 합쳐서 쿼리를 작성하였다.

마찬가지로 코드의 흐름은 TYPE1과 유사하므로 설명을 생략하겠다.

3) TYPE 3 : Find the name of the agent who has sold the most properties in the year 2022 by total won value.

```
else if (choice == 3) {
    query = "SELECT a.agent_id, a.agent_name, SUM(t.sale_price) AS total_sales_price W
            FROM agent a JOIN transaction t ON a.agent_id = t.selling_agent_id W
            WHERE YEAR(t.sale_end_date) = 2022 W
            GROUP BY a.agent_id, a.agent_name W
            ORDER BY total_sales_price DESC LIMIT 1;";
    state = mysql_query(connection, query);
    if (state == 0) {
        sql_result = mysql_store_result(connection);
        printf("** Find the name of the agent who has sold the most properties in the year 2022 by total won value. **\n");
        while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
            printf("%s %s %s\n", sql_row[0], sql_row[1], sql_row[2]);
        mysql_free_result(sql_result);
    }
    printf("\n");
}
```

SELECT a.agent_id, a.agent_name, SUM(t.sale_price) AS total_sales_price

FROM agent a JOIN transaction t ON a.agent_id = t.selling_agent_id

WHERE YEAR(t.sale_end_date) = 2022

GROUP BY a.agent_id, a.agent_name

ORDER BY total_sales_price DESC LIMIT 1;

본격적으로 쿼리가 복잡해졌다. 사실 더 간단한 쿼리를 작성할 수도 있었지만, 3-1, 3-2 쿼리를 작성하는 과정에서 3번 쿼리를 자연스럽게 구할 수 있었으므로 위와 같은 쿼리를 작성하였다. 먼저 agent의 이름을 구해야 하므로 agent와 transaction 테이블을 JOIN하였다. 또한 판매한 부동산의 원화가치가 가장 높은 agent를 구해야 하므로 GROUP BY를 이용하여 agent별로 그룹화한 후 SUM을 이용해 total_sales_price를 구하였다. 그리고 이 값을 기준으로 내림차순으로 정렬한 후 가장 위에 있는 데이터를 뽑아내는 쿼리를 작성하였다.

마찬가지로 코드의 흐름은 TYPE1과 유사하므로 설명을 생략하겠다.

3.1) TYPE 3-1 : Then find the top k agents in the year 2023 by total won value.

```
while (true) {
    printf("----- Subtypes in TYPE 3 -----Wn");
    printf("    1. TYPE 3-1Wn");
    printf("    2. TYPE 3-2Wn");
    scanf("%d", &choice);
    if (choice == 0)
        break;
    else if (choice == 1) {
        printf("** Then find the top k agents in the year 2023 by total won value. **Wn");
        printf("Which K ? : ");
        int k; scanf("%d", &k); char k_str[10];
        sprintf(k_str, "%d", k);

        query = strjoin("SELECT a.agent_id, a.agent_name, SUM(t.sale_price) AS total_sales_price W
                        FROM agent a JOIN transaction t ON a.agent_id = t.selling_agent_id W
                        WHERE YEAR(t.sale_end_date) = 2023 W
                        GROUP BY a.agent_id, a.agent_name W
                        ORDER BY total_sales_price DESC LIMIT ", k_str);
        state = mysql_query(connection, query);
        free(query);
        if (state == 0) {
            sql_result = mysql_store_result(connection);
            printf("** Then find the top k agents in the year 2023 by total won value. **Wn");
            while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
                printf("%s %s %sWn", sql_row[0], sql_row[1], sql_row[2]);
            mysql_free_result(sql_result);
        }
        printf("Wn");
    }
}
```

SELECT a.agent_id, a.agent_name, SUM(t.sale_price) AS total_sales_price

FROM agent a JOIN transaction t ON a.agent_id = t.selling_agent_id

WHERE YEAR(t.sale_end_date) = 2023

GROUP BY a.agent_id, a.agent_name

ORDER BY total_sales_price DESC LIMIT k;

3번 쿼리부터는 메인쿼리와 서브쿼리의 질문이 달라지므로 각각의 쿼리를 분리하여 해석하였다. 3-1번 쿼리는 3번 쿼리와 유사하므로 설명을 생략하겠다.

사용자로부터 입력받은 k와 직접 작성한 쿼리문을 합쳐야 최종적인 쿼리를 넘겨줄 수 있었기 때문에 strjoin함수를 이용하여 두 문자열을 합쳤다. 그리고 이 값을 mysql_query함수에 인자로 넘겨서 쿼리를 실행하고 데이터를 가져올 수 있었다. 마지막엔 strjoin함수를 통해 동적 할당된 문자열을 가리키고 있는 query를 free해주었다.

3.2) TYPE 3-2 : And then find the bottom 10% agents in the year 2021 by total won value.

```

else if (choice == 2) {
    state = mysql_query(connection, "SELECT COUNT(DISTINCT selling_agent_id) FROM transaction WHERE YEAR(sale_end_date) = 2021; ");
    if (state == 0) {
        sql_result = mysql_store_result(connection);
        sql_row = mysql_fetch_row(sql_result);
        int k = (int)(atoi(sql_row[0]) * 0.1);
        mysql_free_result(sql_result);
        char k_str[10];
        sprintf(k_str, "%d", k);

        query = strjoin("SELECT a.agent_id, a.agent_name, SUM(t.sale_price) AS total_sales_price ₩
                        FROM agent a JOIN transaction t ON a.agent_id = t.selling_agent_id ₩
                        WHERE YEAR(t.sale_end_date) = 2021 ₩
                        GROUP BY a.agent_id, a.agent_name ₩
                        ORDER BY total_sales_price ASC LIMIT ", k_str);
        state = mysql_query(connection, query);
        if (state == 0) {
            sql_result = mysql_store_result(connection);
            printf("** And then find the bottom 10% agents in the year 2021 by total won value. **\n");
            while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
                printf("%s %s %s\n", sql_row[0], sql_row[1], sql_row[2]);
            mysql_free_result(sql_result);
        }
    }
    printf("\n");
}

```

SELECT COUNT(DISTINCT selling_agent_id) FROM transaction WHERE YEAR(sale_end_date) = 2021;

이번에는 쿼리 한 개가 더 필요했다. bottom 10% agent가 몇 명인지를 알기 위해 가장 먼저 transaction 테이블에 존재하는 agent의 명수를 찾는 쿼리를 작성하였다. 중복을 제거하기 위해 DISTINCT를 사용했고 agent를 COUNT한 결과를 sql_result에 받았다.

sql_result는 1행 1열로 구성되어 있기 때문에 받아온 결과를 간단히 저장하고 int로 변환하여 계산한 다음 sprintf를 이용하여 k_str에 결과값을 저장하였다. 그리고 두 문자열을 join하여 최종적인 쿼리를 완성하였다.

4) TYPE 4 : For each agent, compute the average selling price of properties sold in 2022, and the average time the property was on the market.

```

else if (choice == 4) {
    query = "SELECT a.agent_id, a.agent_name, AVG(t.sale_price) AS avg_sale_price, AVG(DATEDIFF(t.sale_end_date, p.sale_start_date)) AS avg_sale_duration ₩
            FROM agent a JOIN transaction t ON a.agent_id = t.selling_agent_id JOIN property p ON t.prop_id = p.prop_id ₩
            WHERE YEAR(t.sale_end_date) = 2022 ₩
            GROUP BY a.agent_id, a.agent_name ₩
            ORDER BY avg_sale_price DESC;";
    state = mysql_query(connection, query);
    if (state == 0) {
        sql_result = mysql_store_result(connection);
        printf("** For each agent, compute the average selling price of properties sold in 2022, and the average time the property was on the market. **\n");
        while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
            printf("%s %s %s %s\n", sql_row[0], sql_row[1], sql_row[2], sql_row[3]);
        mysql_free_result(sql_result);
    }
    printf("\n");
}

```

```

SELECT  a.agent_id,  a.agent_name,  AVG(t.sale_price)  AS  avg_sale_price,
        AVG(DATEDIFF(t.sale_end_date, p.sale_start_date)) AS avg_sale_duration

FROM agent a JOIN transaction t ON a.agent_id = t.selling_agent_id JOIN property p
ON t.prop_id = p.prop_id

WHERE YEAR(t.sale_end_date) = 2022

GROUP BY a.agent_id, a.agent_name

ORDER BY avg_sale_price DESC;

```

3번 쿼리에서 조금 변형되었다. 이번에는 average selling price와 부동산이 시장에 머물렀던 평균기간을 계산해야 한다. 앞선 쿼리들처럼 GROUP BY를 통해 데이터를 agent 별로 묶어주었고 SUM 대신 AVG 함수를 이용해 쉽게 평균을 구할 수 있었다. 부동산이 시장에 머물렀던 평균기간 또한 DATEDIFF와 AVG를 이용하여 간편하게 구할 수 있었다. 정렬 기준은 average sale price로 설정해주었다.

코드의 흐름은 TYPE1과 유사하므로 설명을 생략하겠다.

4.1) TYPE 4-1 : Then compute the maximum selling price of properties sold in 2023 for each agent.

```

while (true) {
    printf("----- Subtypes in TYPE 4 -----Wn");
    printf("    1. TYPE 4-1Wn");
    printf("    2. TYPE 4-2Wn");
    scanf("%d", &choice);
    if (choice == 0)
        break;
    else if (choice == 1) {
        query = "SELECT a.agent_id, a.agent_name, MAX(t.sale_price) AS max_sale_price W
                FROM agent a JOIN transaction t ON a.agent_id = t.selling_agent_id W
                WHERE YEAR(t.sale_end_date) = 2023 W
                GROUP BY a.agent_id, a.agent_name W
                ORDER BY max_sale_price DESC;";
        state = mysql_query(connection, query);
        if (state == 0) {
            sql_result = mysql_store_result(connection);
            printf("** Then compute the maximum selling price of properties sold in 2023 for each agent. **Wn");
            while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
                printf("%s %s %sWn", sql_row[0], sql_row[1], sql_row[2]);
            mysql_free_result(sql_result);
        }
        printf("Wn");
    }
}

```

```

SELECT a.agent_id, a.agent_name, MAX(t.sale_price) AS max_sale_price

```

```

FROM agent a JOIN transaction t ON a.agent_id = t.selling_agent_id

```


WHERE YEAR(t.sale_end_date) = 2023

GROUP BY a.agent_id, a.agent_name

ORDER BY max_sale_price DESC;

앞선 쿼리들과 요구하는 바가 크게 다르지 않아서 어렵지 않게 구할 수 있었다. 이번에는 agent별로 판매한 가장 고가의 부동산을 구해야 하므로 MAX를 이용하였다.

코드의 흐름은 TYPE1과 유사하므로 설명을 생략하겠다.

4.2) TYPE 4-2 : And then compute the longest time the property was on the market for each agent.

```
else if (choice == 2) {
    query = "SELECT a.agent_id, a.agent_name, MAX(DATEDIFF(t.sale_end_date, p.sale_start_date)) AS max_days_on_market W
            FROM agent a JOIN transaction t ON a.agent_id = t.selling_agent_id JOIN property p ON t.prop_id = p.prop_id W
            GROUP BY a.agent_id, a.agent_name W
            ORDER BY max_days_on_market ASC;";
    state = mysql_query(connection, query);
    if (state == 0) {
        sql_result = mysql_store_result(connection);
        printf("** And then compute the longest time the property was on the market for each agent. **\n");
        while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
            printf("%s %s %s\n", sql_row[0], sql_row[1], sql_row[2]);
        mysql_free_result(sql_result);
    }
    printf("\n");
}
```

SELECT a.agent_id, a.agent_name, MAX(DATEDIFF(t.sale_end_date, p.sale_start_date)) AS max_days_on_market

FROM agent a JOIN transaction t ON a.agent_id = t.selling_agent_id JOIN property p ON t.prop_id = p.prop_id

GROUP BY a.agent_id, a.agent_name

ORDER BY max_days_on_market ASC;

이번 쿼리 또한 메인쿼리와 상관없는 서브쿼리로 해석하였다. WHERE 조건을 삭제하고 MAX 안에 DATEDIFF를 사용하여 부동산이 시장에 머물렀던 가장 긴 기간을 구했다.

코드의 흐름은 TYPE1과 유사하므로 설명을 생략하겠다.

5) TYPE 5 : Show photos of the most expensive studio, one-bedroom, multi-bedroom apartment(s), and detached house(s), respectively, from the database.

```

else if (choice == 5) {
    printf("** Show photos of the most expensive studio, one - bedroom, W
multi - bedroom apartment(s), and detached house(s), respectively, from the database. **\n");
    query = "SELECT p.prop_id, p.prop_type, p.listing_price, W
            ip.in_photo_data, ep.ex_photo_data, fp.fl_photo_data W
            FROM property p W
            LEFT OUTER JOIN in_photo ip ON p.prop_id = ip.prop_id W
            LEFT OUTER JOIN ex_photo ep ON p.prop_id = ep.prop_id W
            LEFT OUTER JOIN fl_photo fp ON p.prop_id = fp.prop_id W
            WHERE p.prop_type = 'studio' W
            ORDER BY p.listing_price DESC W
            LIMIT 1;";
    state = mysql_query(connection, query);
    if (state == 0) {
        sql_result = mysql_store_result(connection);
        while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
            printf("%s %s %s %s %s %s\n", sql_row[0], sql_row[1], sql_row[2], sql_row[3], sql_row[4], sql_row[5]);
        mysql_free_result(sql_result);
    }
}

```

SELECT p.prop_id, p.prop_type, p.listing_price,

ip.in_photo_data, ep.ex_photo_data, fp.fl_photo_data

FROM property p

LEFT OUTER JOIN in_photo ip ON p.prop_id = ip.prop_id

LEFT OUTER JOIN ex_photo ep ON p.prop_id = ep.prop_id

LEFT OUTER JOIN fl_photo fp ON p.prop_id = fp.prop_id

WHERE p.prop_type = 'studio'

ORDER BY p.listing_price DESC

LIMIT 1;

종류별로 가장 고가인 property의 사진을 가져오기 위해 property와 in_photo, ex_photo, fl_photo를 JOIN하였다. 이때 photo는 property의 종류에 따라 있을 수도 있고 없을 수도 있으므로 property에 대해 LEFT OUTER JOIN하였다. 위 쿼리는 studio에 대한 쿼리이며, one-bedroom apartments, multi-bedroom apartments, detached house 또한 동일한 방식으로 쿼리를 작성하였다.

코드의 흐름은 TYPE1과 유사하므로 설명을 생략하겠다.

6) TYPE 6 : Record the sale of a property that had been listed as being available. This entails storing the sales price, the buyer, the selling agent, the buyer's agent(if any), and the date.

```

else if (choice == 6) {
    char* trans_id, prop_id[20], seller_id[20], buyer_id[20], selling_agent_id[20], buyer_agent_id[20], sale_end_date;
    trans_id = random_id_generator(connection);
    printf("** Record the sale of a property that had been listed as being available. **\n");
    printf("ex. id = 1234567890, date = yyyy-mm-dd.\n");
    query = (char*)malloc(sizeof(char) * 500);
    printf("1. property id : ");
    scanf("%s", prop_id);
    snprintf(query, 500, "SELECT COUNT(*) FROM property WHERE prop_id='%s' AND sale_status = 'TRUE';", prop_id);
    state = mysql_query(connection, query);
    sql_result = mysql_store_result(connection);
    sql_row = mysql_fetch_row(sql_result);
    // listing된 부동산이 아닐 경우 다시 입력받기
    while (sql_row && atoi(sql_row[0]) == 0) {
        mysql_free_result(sql_result);
        printf("Invalid property ID. Try again. : ");
        scanf("%s", prop_id);
        snprintf(query, 500, "SELECT COUNT(*) FROM property WHERE prop_id='%s' AND sale_status = 'TRUE';", prop_id);
        state = mysql_query(connection, query);
        sql_result = mysql_store_result(connection);
        sql_row = mysql_fetch_row(sql_result);
    }
}

```

SELECT COUNT(*) FROM property WHERE prop_id=prop_id AND sale_status = 'TRUE';

6번 쿼리는 사용자(아마도 agent)에게 prop_id, seller_id, buyer_id, selling agent id, sale_end_date를 입력받아 부동산에 대한 판매를 발생시키는 쿼리이다. 가장 먼저 해야 할 것은 입력받은 prop_id가 유효한 ID인지 확인하는 것이다. 위 쿼리문이 바로 그 역할을 수행한다. prop_id가 테이블에 존재하고 sale_status가 판매중이라면 쿼리의 결과값으로 1이 나올 것이고, 아니라면 0이 나올 것이다. 확인 결과 유효한 prop_id가 아니라면 유효한 prop_id가 나올 때까지 계속해서 입력받는다.

추가적으로 위 코드의 random_number_generator 함수는 random한 10자리 ID를 생성하고 기존의 ID와 중복되는지 확인한 후 unique한 ID라면 return하는 함수이다.

```

printf("2. seller id : ");
scanf("%s", seller_id);
printf("3. buyer id : ");
scanf("%s", buyer_id);
printf("4. selling agent id : ");
scanf("%s", selling_agent_id);
printf("5. buyer's agent id (if not, please enter 'n') : ");
scanf("%s", buyer_agent_id);
printf("6. date : ");
scanf("%s", sale_end_date);
mysql_query(connection, "START TRANSACTION;");
// transaction 테이블에 추가
snprintf(query, 500, "INSERT INTO transaction VALUES ('%s', '%s', '%s', '%s', (SELECT listing_price FROM property WHERE prop_id='%s'), '%s', '%s');", W, trans_id, seller_id, buyer_id, prop_id, prop_id, sale_end_date, selling_agent_id);
free(trans_id);
state = mysql_query(connection, query);
if (state != 0) {
    mysql_query(connection, "ROLLBACK;");
    free(query);
    printf("seller, buyer, selling_agent, buyer's agent 중 등록되어 있지 않은 고객이 있어 메인으로 돌아갑니다. 등록 후 다시 시도해주세요.\n");
    continue;
}

```

START TRANSACTION;

이번 프로젝트를 수행하며 가장 흥미로웠던 부분이다. 은행 거래 시 발생하는 데이터의 변경이 atomic해야 하는 것처럼, property의 판매를 발생시킬 때에도 몇 개의 쿼리문이 atomic하게 수행되어야 한다. 이를 위해 START TRANSACTION, ROLLBACK, COMMIT

쿼리문을 군데군데 적절하게 넣어주었다.

```
INSERT INTO transaction VALUES (trans_id, seller_id, buyer_id, prop_id, (SELECT
listing_price FROM property WHERE prop_id=prop_id), sale_end_date,
selling_agent_id);
```

위 사진의 코드는 사용자에게 유효한 prop_id를 입력받은 후 여타 정보들도 한꺼번에 입력받은 다음 쿼리를 실행시킨다. 만약 쿼리가 실패한다면 FK 제약조건이 걸렸을 확률이 높다. 따라서 사용자에게 seller, buyer, selling_agent, buyer_agent 중 db에 등록되지 않은 데이터가 있음을 알려준다. 사용자가 입력한 데이터가 각각의 테이블에 적절히 존재하는 데이터라면 쿼리가 성공적으로 실행될 것이다. 예시는 다음과 같다.

```
Connection Succeed

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

1. TYPE 1
2. TYPE 2
3. TYPE 3
4. TYPE 4
5. TYPE 5
6. TYPE 6
7. TYPE 7
0. QUIT
6
** Record the sale of a property that had been listed as being available. **
ex. id = 1234567890, date = yyyy-mm-dd.
1. property id : 2633503989
Invalid property ID. Try again. :
```

▲ Invalid property ID

```
** Record the sale of a property that had been listed as being available. **
ex. id = 1234567890, date = yyyy-mm-dd.
1. property id : 1250197763
2. seller id : 7852241020
3. buyer id : 3023008429
4. selling agent id : 4456220837
5. buyer's agent id (If not, please enter 'n') : 8635862359
6. date : 2024-06-08
Successfully updated.
```

▲ valid data set

```
** Record the sale of a property that had been listed as being available. **
ex. id = 1234567890, date = yyyy-mm-dd.
1. property id : 1250197763
Invalid property ID. Try again. :
```

▲ 판매완료 상태로 변경된 property를 또다시 판매완료 상태로 바꾸려는 경우

```
// 구매자측 중개사가 존재한다면 buyer_agent 테이블에 추가
if (strcmp(buyer_agent_id, "n") && strcmp(buyer_agent_id, "N")) {
    sprintf(query, 500, "INSERT INTO buyer_agent VALUES ('%s', '%s');", trans_id, buyer_agent_id);
    state = mysql_query(connection, query);
    if (state != 0) {
        mysql_query(connection, "ROLLBACK;");
        free(query);
        printf("agent 테이블에 해당 agent가 존재하지 않아 메인으로 돌아갑니다. agent를 먼저 추가해주세요.\n");
        continue;
    }
}
```

INSERT INTO buyer_agent VALUES (trans_id, buyer_agent_id);

구매자측 agent는 존재할 수도 있고 그렇지 않을 수도 있으므로 if문을 작성하여 추가하는 경우를 구분해주었다. agent 추가에 실패하면 ROLLBACK; 쿼리문을 실행시켜 초기 상태로 되돌린다.

```
// property 테이블의 sale_status를 false(판매완료)로 업데이트
sprintf(query, 500, "UPDATE property SET sale_status = 'FALSE' WHERE prop_id = '%s';", prop_id);
state = mysql_query(connection, query);
if (state != 0) {
    mysql_query(connection, "ROLLBACK;");
    free(query);
    printf("property 테이블의 sale_status를 변경하는 도중 오류가 발생해 메인으로 돌아갑니다.\n");
    continue;
}
state = mysql_query(connection, "COMMIT;");
printf("Successfully updated.\n\n");
free(query);
```

UPDATE property SET sale_status = 'FALSE' WHERE prop_id = prop_id;

property 테이블의 sale_status를 변경하는 쿼리이다. 마찬가지로 실패하면 ROLLBACK이 일어난다. 성공했다면 COMMIT; 쿼리문으로 업데이트를 확정짓는다.

7) TYPE : Add a new agent to the database.

```
else if (choice == 7) {
    char agent_name[21], agent_phone_number[21], agent_address[101];
    query = (char*)malloc(sizeof(char) * 500);

    printf("Agent name : ");
    scanf("%s", agent_name);
    printf("Agent phone number : ");
    scanf("%s", agent_phone_number);
    printf("Agent address : ");
    scanf("%s", agent_address);
}
```

new agent 추가를 위해 사용자에게 agent 정보를 입력받는 코드이다.

```
snprintf(query, 500, "SELECT COUNT(*) FROM agent WHERE agent_phone_number='%s'", agent_phone_number);
state = mysql_query(connection, query);

if (state == 0) {
    sql_result = mysql_store_result(connection);
    sql_row = mysql_fetch_row(sql_result);
    if (sql_row && atoi(sql_row[0]) == 0) {
        snprintf(query, 500, "INSERT INTO agent VALUES ('%s', '%s', '%s', '%s');", W
            random_id_generator(connection), agent_name, agent_phone_number, agent_address);
        state = mysql_query(connection, query);
        if (state == 0)
            printf("Agent가 성공적으로 추가되었습니다.\n");
        else
            printf("Agent 추가를 실패했습니다.\n");
    }
    else
        printf("이미 가입된 agent입니다.\n");
    mysql_free_result(sql_result);
}
free(query);
printf("\n");
```

SELECT COUNT(*) FROM agent WHERE agent_phone_number={agent_phone_number}
agent의 phone number가 기존 db에 존재한다면 쿼리문은 1을 반환한다. 이 경우 이미 가입된 agent임을 알려서 중복 가입을 방지한다.

INSERT INTO agent VALUES (random_id_generator(connection), agent_name, agent_phone_number, agent_address);

새로운 agent라면 입력받은 정보를 바탕으로 db에 추가하고 성공 메시지를 출력한다.