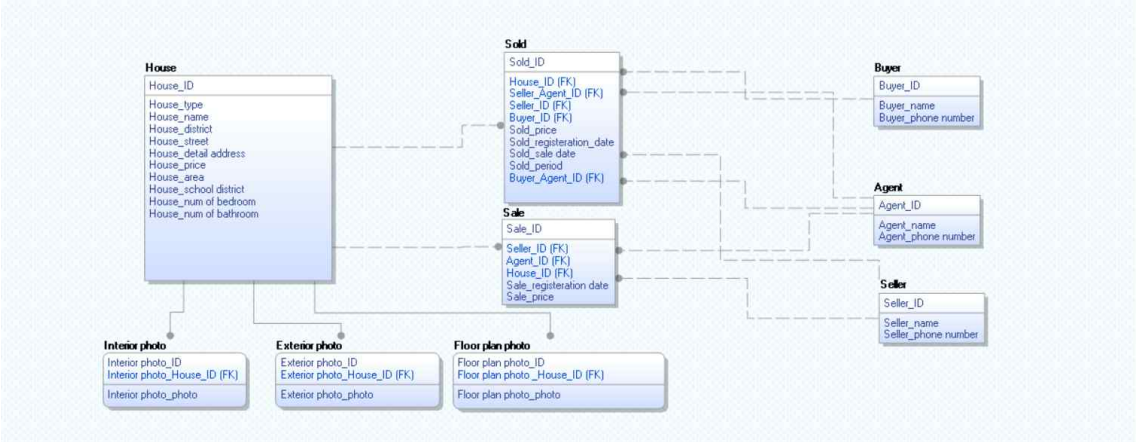
데이터베이스 시스템 과제 2

20201586 백기원

1. BCNF Decomposition



기존의 구성하였던 table은 다음과 같다. 그럼 table 하나씩 BCNF를 진행해 보겠다.

1-1. House 테이블

- 속성: House_ID (PK), House_type, House_name, House_district, House_street, House_detail_address, House_price, House_school_district, House_area, House_num_of_bedroom, House_num_of_bathroom
- 후보 키: House_ID
- 함수적 종속성: House_ID -> 모든 속성
- BCNF 상태: 만족 (House_ID가 후보 키이므로 모든 함수적 종속성의 결정자가 후보 키이다.)

1-2. Interior_photo 테이블

- 속성: Interior_photo_ID (PK), Interior_photo_House_ID (FK), Interior_photo_photo
- 후보 키: Interior_photo_ID
- 함수적 종속성: Interior_photo_ID -> 모든 속성
- BCNF 상태: 만족 (Interior_photo_ID가 후보 키이므로 모든 함수적 종속성의 결정자가 후보 키이다.)

1-3. Exterior_photo 테이블

- 속성: Exterior_photo_ID (PK), Exterior_photo_House_ID (FK), Exterior_photo_photo
- 후보 키: Exterior_photo_ID
- 함수적 종속성: Exterior_photo_ID -> 모든 속성
- BCNF 상태: 만족 (Exterior_photo_ID가 후보 키이므로 모든 함수적 종속성의 결정자가 후보 키이다.)

1-4. Floor_plan_photo 테이블

- 속성: Floor_plan_photo_ID (PK), Floor_plan_photo_House_ID (FK), Floor_plan_photo_photo
- 후보 키: Floor_plan_photo_ID
- 함수적 종속성: Floor_plan_photo_ID -> 모든 속성
- BCNF 상태: 만족 (Floor_plan_photo_ID가 후보 키이므로 모든 함수적 종속성의 결정자가 후보 키이다.)

1-5. Sold 테이블

- 속성: Sold_ID (PK), Sold_House_ID (FK), Sold_Seller_Agent_ID (FK), Sold_Seller_ID (FK), Sold_Buyer_ID (FK), Sold_registration_date, Sold_sale_date, Sold_price, Sold_Buyer_Agent_ID (FK)
- 후보 키: Sold_ID
- 함수적 종속성: Sold_ID -> 모든 속성
- BCNF 상태: 만족 (Sold_ID가 후보 키이므로 모든 함수적 종속성의 결정자가 후보 키이다.)

1-6. Sale 테이블

- 속성: Sale_ID (PK), Sale_Seller_ID (FK), Sale_Seller_Agent_ID (FK), Sale_House_ID (FK), Sale_registration_date, Sale_price
- 후보 키: Sale_ID
- 함수적 종속성: Sale_ID -> 모든 속성
- BCNF 상태: 만족 (Sale_ID가 후보 키이므로 모든 함수적 종속성의 결정자가 후보 키이다.)

1-7. Buyer 테이블

- 속성: Buyer_ID (PK), Buyer_name, Buyer_phone_number
- 후보 키: Buyer_ID

- 함수적 종속성: Buyer_ID -> 모든 속성
- BCNF 상태: 만족 (Buyer_ID가 후보 키이므로 모든 함수적 종속성의 결정자가 후보 키이다.)

1-8. Agent 테이블

- 속성: Agent_ID (PK), Agent_name, Agent_phone_number
- 후보 키: Agent_ID
- 함수적 종속성: Agent_ID -> 모든 속성
- BCNF 상태: 만족 (Agent_ID가 후보 키이므로 모든 함수적 종속성의 결정자가 후보 키이다.)

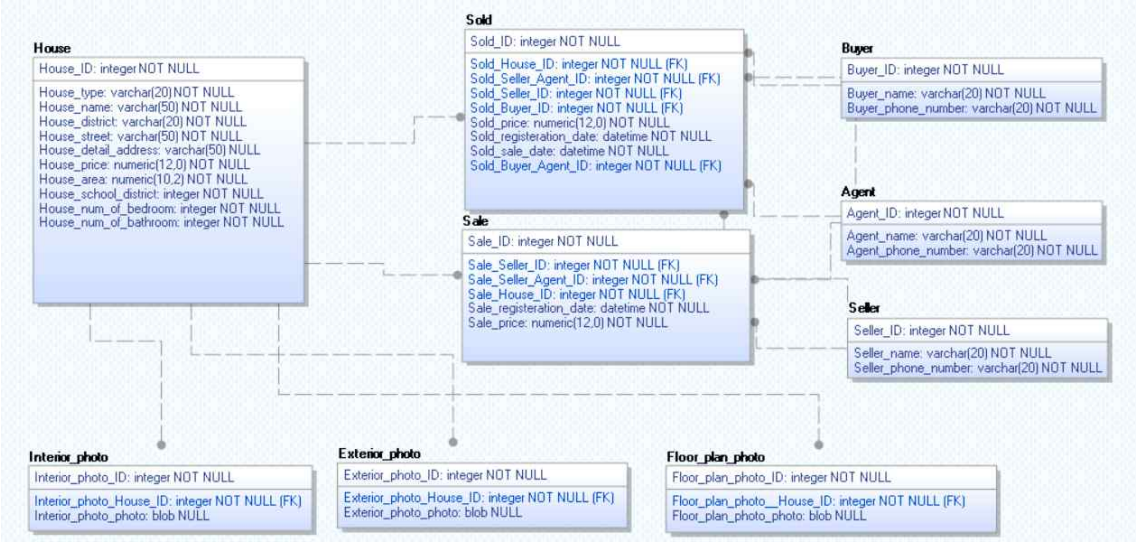
1-9. Seller 테이블

- 속성: Seller_ID (PK), Seller_name, Seller_phone_number
- 후보 키: Seller_ID
- 함수적 종속성: Seller_ID -> 모든 속성
- BCNF 상태: 만족 (Seller_ID가 후보 키이므로 모든 함수적 종속성의 결정자가 후보 키이다.)

1-10. 변경된 테이블

모든 테이블이 BCNF를 만족하므로 변경된 테이블은 없다.

2. Physical Schema Diagram



Physical Schema Diagram은 다음과 같다. 테이블 속성의 data type, domain, constraints, allowing nulls와 relationship type에 대해서 살펴보겠다.

2-1. House 테이블

속성 이름	data type	allowing nulls
House_ID (PK)	integer	NOT NULL
House_type	varchar(20)	NOT NULL
House_name	varchar(50)	NOT NULL
House_district	varchar(20)	NOT NULL
House_street	varchar(50)	NOT NULL
House_detail_address	varchar(50)	NULL
House_price	numeric(12,0)	NOT NULL
House_school_district	integer	NOT NULL
House_area	numeric(10,2)	NOT NULL
House_num_of_bedroom	integer	NOT NULL
House_num_of_bathroom	integer	NOT NULL

- House_ID (PK): ID이므로 정수로 설정하였다. 또한 primary key이기에 NOT NULL이다.
- House_type: 길이가 가변적이므로 varchar(20)을 사용하였다. NULL은 허용하지 않는다.
- House_name: 아파트의 이름은 짧지 않은 경우가 많으므로 varchar(50)으로 설정하였다. NULL은 허용하지 않는다.
- House_district: 길이가 가변적이므로 varchar(20)을 사용하였다. NULL은 허용하지 않는다.
- House_street: 길이가 가변적이므로 varchar(50)을 사용하였다. NULL은 허용하지 않는다.
- House_detail_address: 길이가 가변적이므로 varchar(50)을 사용하였다. 주택 같은 경우에는 몇 동 몇 호처럼 세부 주소가 필요하지 않은 경우가 있으므로 NULL을 허용한다.
- House_price: 원화이기에 정수를 사용하였다. 하지만 값은 integer의 범위를 넘어설 수 있으므로 numeric(12,0)을 사용하였다. NULL은 허용하지 않는다.

- House_school_district: 학군의 번호가 정수이므로 integer를 사용하였다. NULL은 허용하지 않는다.
- House_area: 면적은 보통 소수 둘째 자리까지 표현하기에 numeric(10,2)을 사용하였다.
- House_num_of_bedroom: 개수가 정수이므로 integer를 사용하였다. NULL은 허용하지 않는다.
- House_num_of_bathroom: 개수가 정수이므로 integer를 사용하였다. NULL은 허용하지 않는다.

2-2. Interior_photo 테이블

속성 이름	data type	allowing nulls
Interior_photo_ID (PK)	integer	NOT NULL
Interior_photo_House_ID (FK)	integer	NOT NULL
Interior_photo_photo	blob	NULL

- Interior_photo_ID (PK): ID이므로 정수로 설정하였다. 또한 primary key이기에 NOT NULL이다.
- Interior_photo_House_ID (FK): ID이므로 정수로 설정하였다. 또한 primary key이기에 NOT NULL이다.
- Interior_photo_photo: 사진이므로 blob으로 설정하였다. 집의 종류에 따라 필수이기도 하고 아니기도 하므로 NULL이 가능하다.

2-3. Exterior_photo 테이블

속성 이름	data type	allowing nulls
Exterior_photo_ID (PK)	integer	NOT NULL
Exterior_photo_House_ID (FK)	integer	NOT NULL
Exterior_photo_photo	blob	NULL

- Exterior_photo_ID (PK): ID이므로 정수로 설정하였다. 또한 primary key이기에 NOT NULL이다.
- Exterior_photo_House_ID (FK): ID이므로 정수로 설정하였다. 또한 primary key이기에 NOT NULL이다.
- Exterior_photo_photo: 사진이므로 blob으로 설정하였다. 집의 종류에 따라 필수이기도 하고 아니기도 하므로 NULL이 가능하다.

2-4. Floor_plan_photo 테이블

속성 이름	data type	allowing nulls
Floor_plan_photo_ID (PK)	integer	NOT NULL
Floor_plan_photo_House_ID (FK)	integer	NOT NULL
Floor_plan_photo_photo	blob	NULL

- Floor_plan_photo_ID (PK): ID이므로 정수로 설정하였다. 또한 primary key이기에 NOT NULL이다.
- Floor_plan_photo_House_ID (FK): ID이므로 정수로 설정하였다. 또한 primary key이기에 NOT NULL이다.

- Floor_plan_photo_photo: 사진이므로 blob으로 설정하였다. 집의 종류에 따라 필수이기도 하고 아니기도 하므로 NULL이 가능하다.

2-5. Sold 테이블

속성 이름	data type	allowing nulls
Sold_ID (PK)	integer	NOT NULL
Sold_House_ID (FK)	integer	NOT NULL
Sold_Seller_Agent_ID (FK)	integer	NOT NULL
Sold_Seller_ID (FK)	integer	NOT NULL
Sold_Buyer_ID (FK)	integer	NOT NULL
Sold_registration_date	datetime	NOT NULL
Sold_sale_date	datetime	NOT NULL
Sold_price	numeric(12,0)	NOT NULL
Sold_Buyer_Agent_ID (FK)	integer	NOT NULL

- Sold_ID (PK): ID이므로 정수로 설정하였다. 또한 primary key이기에 NOT NULL이다.
- Sold_House_ID (FK): ID이므로 정수로 설정하였다. NULL을 허용하지 않는다.
- Sold_Seller_Agent_ID (FK): ID이므로 정수로 설정하였다. NULL을 허용하지 않는다.
- Sold_Seller_ID (FK): ID이므로 정수로 설정하였다. NULL을 허용하지 않는다.
- Sold_Buyer_ID (FK): ID이므로 정수로 설정하였다. NULL을 허용하지 않는다.
- Sold_registration_date: 날짜를 표현해야 하므로 datetime으로 설정하였다. NULL을 허용하지 않는다.
- Sold_sale_date: 날짜를 표현해야 하므로 datetime으로 설정하였다. NULL을 허용하지 않는다.
- Sold_price: 원화이기에 정수를 사용하였다. 하지만 값은 integer의 범위를 넘어설 수 있으므로 numeric(12,0)을 사용하였다. NULL은 허용하지 않는다.
- Sold_Buyer_Agent_ID (FK): ID이므로 정수로 설정하였다. NULL을 허용하지 않는다.

2-6. Sale 테이블

속성 이름	data type	allowing nulls
Sale_ID (PK)	integer	NOT NULL
Sale_Seller_ID	integer	NOT NULL
Sale_Seller_Agent_ID (FK)	integer	NOT NULL
Sale_House_ID (FK)	integer	NOT NULL
Sale_registration_date	datetime	NOT NULL
Sale_price	numeric(12,0)	NOT NULL

- Sale_ID (PK): ID이므로 정수로 설정하였다. 또한 primary key이기에 NOT NULL이다.
- Sale_Seller_ID : ID이므로 정수로 설정하였다. NULL을 허용하지 않는다.
- Sale_Seller_Agent_ID (FK): ID이므로 정수로 설정하였다. NULL을 허용하지 않는다.
- Sale_House_ID (FK): ID이므로 정수로 설정하였다. NULL을 허용하지 않는다.
- Sale_registration_date: 날짜를 표현해야 하므로 datetime으로 설정하였다. NULL을 허용하지 않는다.

- Sale_price: 원화이기에 정수를 사용하였다. 하지만 값은 integer의 범위를 넘어설 수 있으므로 numeric(12,0)을 사용하였다. NULL은 허용하지 않는다.

2-7. Buyer 테이블

속성 이름	data type	allowing nulls
Buyer_ID (PK)	integer	NOT NULL
Buyer_name	varchar(20)	NOT NULL
Buyer_phone_number	varchar(20)	NOT NULL

- Buyer_ID (PK): ID이므로 정수로 설정하였다. 또한 primary key이기에 NOT NULL이다.
- Buyer_name: 길이가 가변적이므로 varchar(20)을 사용하였다. NULL은 허용하지 않는다.
- Buyer_phone_number: 길이가 가변적이므로 varchar(20)을 사용하였다. NULL은 허용하지 않는다.

2-8. Agent 테이블

속성 이름	data type	allowing nulls
Agent_ID (PK)	integer	NOT NULL
Agent_name	varchar(20)	NOT NULL
Agent_phone_number	varchar(20)	NOT NULL

- Agent_ID (PK): ID이므로 정수로 설정하였다. 또한 primary key이기에 NOT NULL이다.
- Agent_name: 길이가 가변적이므로 varchar(20)을 사용하였다. NULL은 허용하지 않는다.
- Agent_phone_number: 길이가 가변적이므로 varchar(20)을 사용하였다. NULL은 허용하지 않는다.

2-9. Seller 테이블

속성 이름	data type	allowing nulls
Seller_ID (PK)	integer	NOT NULL
Seller_type	varchar(20)	NOT NULL
Seller_name	varchar(20)	NOT NULL

- Seller_ID (PK): ID이므로 정수로 설정하였다. 또한 primary key이기에 NOT NULL이다.
- Seller_name: 길이가 가변적이므로 varchar(20)을 사용하였다. NULL은 허용하지 않는다.
- Seller_phone_number: 길이가 가변적이므로 varchar(20)을 사용하였다. NULL은 허용하지 않는다.

2-10. relationship type

relation은 지난 과제 때와 변하지 않았다.

이름	entity1	entity2	type	설명

sale complete	House	Sold	one to many (total)	매물 중 판매가 완료된 것들이다. 판매된 것은 모두 등록이 되어있어야 하므로 total이고, 한 매물이 여러번 판매 되기도 한다. 이 데이터 베이스에서는 한번 구매할 때 집은 하나씩만 할 수 있다. 따라서 one to many이다.
sale in progress	House	Sell	one to one (total)	매물 중 판매 중이라는 뜻이다. 판매 중인 것은 모두 등록이 되어있어야 하므로 total이다. 한번에 한 매물당 하나만 등록 가능하므로 one to one이다.
buying	Buyer	Sold	one to many (total)	누가 무엇을 샀는지에 관련된 realtion이다. 판매된 것은 무조건 구매자가 있기때문에 total이다. 또한 한 구매자가 여러 개의 집 계약을 할 수 있다. 그리고 한 집이 여러 명에게 팔릴 수 없으므로 one to many이다.
selling	Seller	Sell	one to many (total)	판매자가 현재 판매 중이라는 뜻이다. 판매 중인 매물은 무조건 판매자가 있어야 하므로 total이다. 또한 한 판매자가 한 번에 여러 개를 파는 중일 수 있다. 그리고 한 집을 여러 명이 판매할 수 없으므로 one to many이다.
sold before	Seller	Sold	one to many (total)	판매자가 이미 판매했다는 뜻이다. 전에 판매된 매물은 무조건 판매자가 필요하므로 total이다. 또한 한 판매자가 여러 개를 판매했을 수 있다. 그리고 한 집을 판매할 때 한 번에 여러 명이 판매할 수 없으므로 one to many이다.
Agent sold	Agent	Sold	one to many (total)	판매된 매물의 판매를 맡았던 중개사라는 뜻이다. 판매된 매물은 무조건 중개사를 통해서 거래되므로 total이다. 또한 한 Agent가 여러 개의 판매를 맡았을 수 있다. 그러나 계약할 때 한집을 여러 중개사가 중개하지 않으므로 one to many이다.
Agent bought	Agent	Sold	one to many	판매된 매물의 구매 중계를 맡았던 중개사라는 뜻이다. 구매할 때 중개사가 필수는 아니므로 total이 아니다. 또한 한 중개사가 여러 개의 구매를 맡았을 수 있다. 그러나 한 번 구매할 때 여러 명이 중개하지 않으므로 one to many이다.
Agent sell	Agent	Sell	one to many (total)	판매 중인 매물의 판매 중계를 맡고 있다는 의미이다. 판매 시 중개사가 무조건 필요하므로 total이다. 또한 한 중개사가 여러 개를 중개 중일 수 있다. 하지만 판매할 때 여러 명이 중개하지 않으므로 one to many이다.
Interior	House	Interior photo	one to many	매물의 내부 사진 정보이다. 모든 내부 사진은 해당하는 매물이 있으므로 total이다. 한 매물당 여러 사진을 가질 수 있다. 그리고 한 사진이 여러

			(total)	집의 내부를 보여주지는 못하므로 one to many이다.
Exterior	House	Exterior photo	one to many (total)	매물의 외부 사진 정보이다. 모든 외부 사진은 해당하는 매물이 있으므로 total이다. 한 매물당 여러 사진을 가질 수 있다. 그리고 한 사진이 여러 집의 외부를 보여주지는 못하므로 one to many이다.
Floor plan	House	Floor plan photo	one to one (total)	매물의 평면도 사진 정보이다. 모든 평면도 사진은 해당하는 매물이 있으므로 total이다. 한 매물당 평면도는 하나이므로 one to one이다.

3. CRUD

DB를 구성하기 위해서 CRUD.txt를 작성하였다. 아래는 CRUD.txt의 일부분과 이에 대한 설명이다.

3-1. DB 생성

```
CREATE DATABASE project;
USE project;
```

- DB를 생성하고 선택하는 부분이다.

3-2. House 생성

```
CREATE TABLE House (House_ID INTEGER PRIMARY KEY, House_type VARCHAR(20) NOT NULL, House_name VARCHAR(50) NOT NULL, House_district VARCHAR(20) NOT NULL, House_street VARCHAR(50) NOT NULL, House_detail_address VARCHAR(50) NULL, House_price NUMERIC(12,0) NOT NULL, House_school_district INTEGER NOT NULL, House_area NUMERIC(10,2) NOT NULL, House_num_of_bedroom INTEGER NOT NULL, House_num_of_bathroom INTEGER NOT NULL);
INSERT INTO House VALUES (1, 'one-bedroom', 'one', 'Mapo', '1ro', '101 unit', 100000000, 2, 11.11, 1, 1);
```

- House table을 생성하고 값을 넣는 부분이다. 위에서 선언한 datatype, domain을 그대로 적용하였다.

3-3. Interior_photo 생성

```
CREATE TABLE Interior_photo (Interior_photo_ID INTEGER PRIMARY KEY, Interior_photo_House_ID INTEGER NOT NULL, Interior_photo_photo BLOB NULL, FOREIGN KEY (Interior_photo_House_ID) REFERENCES House(House_ID));
INSERT INTO Interior_photo VALUES (1, 1, 'interior1.jpg');
```

- Interior_photo table을 생성하고 값을 넣는 부분이다. 위에서 선언한 datatype, domain

을 그대로 적용하였다.

3-4. Exterior_photo 생성

CREATE TABLE Exterior_photo (Exterior_photo_ID INTEGER PRIMARY KEY, Exterior_photo_House_ID INTEGER NOT NULL, Exterior_photo_photo BLOB NULL, FOREIGN KEY (Exterior_photo_House_ID) REFERENCES House(House_ID));
INSERT INTO Exterior_photo VALUES (1, 1, 'exterior1.jpg');
- Exterior_photo table을 생성하고 값을 넣는 부분이다. 위에서 선언한 datatype, domain을 그대로 적용하였다.

3-5. Floor_plan_photo 생성
CREATE TABLE Floor_plan_photo (Floor_plan_photo_ID INTEGER PRIMARY KEY, Floor_plan_photo_House_ID INTEGER NOT NULL, Floor_plan_photo_photo BLOB NULL, FOREIGN KEY (Floor_plan_photo_House_ID) REFERENCES House(House_ID));
INSERT INTO Floor_plan_photo VALUES (1, 1, 'floorplan1.jpg');
- Floor_plan_photo table을 생성하고 값을 넣는 부분이다. 위에서 선언한 datatype, domain을 그대로 적용하였다.

3-6. Buyer 생성
CREATE TABLE Buyer (Buyer_ID INTEGER PRIMARY KEY, Buyer_name VARCHAR(20) NOT NULL, Buyer_phone_number VARCHAR(20) NOT NULL);
INSERT INTO Buyer VALUES (1, 'buy1', '010-0001-0001');
- Buyer table을 생성하고 값을 넣는 부분이다. 위에서 선언한 datatype, domain을 그대로 적용하였다.

3-7. Agent 생성
CREATE TABLE Agent (Agent_ID INTEGER PRIMARY KEY, Agent_name VARCHAR(20) NOT NULL, Agent_phone_number VARCHAR(20) NOT NULL);
INSERT INTO Agent VALUES (1, 'agent1', '010-0002-0001');
- Agent table을 생성하고 값을 넣는 부분이다. 위에서 선언한 datatype, domain을 그대로 적용하였다.

3-8. Seller 생성
CREATE TABLE Seller (Seller_ID INT PRIMARY KEY, Seller_name VARCHAR(20), Seller_phone_number VARCHAR(20));
INSERT INTO Seller VALUES (1, 'seller1', '010-0003-0001');
- Seller table을 생성하고 값을 넣는 부분이다. 위에서 선언한 datatype, domain을 그대로 적용하였다.

3-9. Sale 생성

```
CREATE TABLE Sale (Sale_ID INTEGER PRIMARY KEY, Sale_Seller_ID INTEGER NOT NULL, Sale_Seller_Agent_ID INTEGER NOT NULL, Sale_House_ID INTEGER NOT NULL, Sale_registration_date DATETIME NOT NULL, Sale_price NUMERIC(12,0) NOT NULL, FOREIGN KEY (Sale_Seller_ID) REFERENCES Seller(Seller_ID), FOREIGN KEY (Sale_Seller_Agent_ID) REFERENCES Agent(Agent_ID), FOREIGN KEY (Sale_House_ID) REFERENCES house(House_ID));
INSERT INTO Sale VALUES (1, 1, 1, 1, '2020-01-01', 100000000);
```

- Sale table을 생성하고 값을 넣는 부분이다. 위에서 선언한 datatype, domain을 그대로 적용하였다.

3-10. Sold 생성

```
CREATE TABLE Sold (Sold_ID INTEGER PRIMARY KEY, Sold_House_ID INTEGER NOT NULL, Sold_Seller_Agent_ID INTEGER NOT NULL, Sold_Seller_ID INTEGER NOT NULL, Sold_Buyer_ID INTEGER NOT NULL, Sold_registration_date DATETIME NOT NULL, Sold_sale_date DATETIME NOT NULL, Sold_price NUMERIC(12,0) NOT NULL, Sold_Buyer_Agent_ID INTEGER NOT NULL, FOREIGN KEY (Sold_House_ID) REFERENCES house(House_ID), FOREIGN KEY (Sold_Seller_Agent_ID) REFERENCES Agent(Agent_ID), FOREIGN KEY (Sold_Seller_ID) REFERENCES Seller(Seller_ID), FOREIGN KEY (Sold_Buyer_ID) REFERENCES Buyer(Buyer_ID), FOREIGN KEY (Sold_Buyer_Agent_ID) REFERENCES Agent(Agent_ID));
INSERT INTO Sold VALUES (1, 11, 6, 1, 1, '2021-01-10', '2021-01-27', 1100000000, 1);
```

- Sold table을 생성하고 값을 넣는 부분이다. 위에서 선언한 datatype, domain을 그대로 적용하였다.

3-11. table 삭제

```
$$$
DROP TABLE Sold;
DROP TABLE Sale;
DROP TABLE Buyer;
DROP TABLE Agent;
DROP TABLE Seller;
DROP TABLE Floor_plan_photo;
DROP TABLE Exterior_photo;
DROP TABLE Interior_photo;
DROP TABLE  House;
```

- 각 테이블과 DB를 삭제하는 부분이다.

4. Query

query를 처리하기 위해서 위에서 작성한 CRUD 파일을 통해 DB에 입력한 후, 숫자를 입력받아 원하는 query를 입력받는다. 모든 query와 subquery는 0을 입력받기 전까지 루프를 돈다. 처음에 연결이 성공하면 다음과 같이 출력된다.

```
------- 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

----------------------------------
Enter type :
```

이제 원하는 TYPE의 숫자를 입력하면 해당 query를 실행해준다. query를 실행하는 방법은 문자열로 query를 생성한 후, 예제파일에 있는 mysql_query 함수를 통해 query를 전송, mysql_store_result, mysql_fetch_row 함수를 통해 결과값을 읽어온다. query 송수신에 대해 간략하게 설명하였으므로, 아래에는 query에 대해 집중적으로 설명하겠다.

```
else if (cmd == 5) {
    printf("  **studio**₩n");
    const char* query_1 =
        "SELECT I.Interior_photo_photo, E.Exterior_photo_photo, F.Floor_plan_photo_photo₩n"
        "FROM House H₩n"
        "LEFT JOIN Interior_photo I ON H.House_ID = I.Interior_photo_House_ID₩n"
        "LEFT JOIN Exterior_photo E ON H.House_ID = E.Exterior_photo_House_ID₩n"
        "LEFT JOIN Floor_plan_photo F ON H.House_ID = F.Floor_plan_photo_House_ID₩n"
        "WHERE H.House_type = 'studio'₩n"
        "ORDER BY H.House_price DESC₩n"
        "LIMIT 1;";
    int state = 0;
    state = mysql_query(connection, query_1);
    if (state == 0) {
        sql_result = mysql_store_result(connection);
        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);
    }
    else {
        printf("Query failed₩n");
    }
```

4-1. TYPE 1

```
const char* query = "SELECT House_street, House_detail_address\nFROM House\nWHERE House_district = 'Mapo';";
```

집에 대한 정보는 house table에 저장되어 있으므로 house에서 찾는다. district이 'mapo'인 곳을 찾아야 하므로 이부분은 where로 구현하였고, 해당 집의 주소를 가져온다.

4-1-1. TYPE 1-1

```
const char* sub_query = "SELECT House_street, House_detail_address\nFROM House\nWHERE House_district = 'Mapo'\nAND House_price BETWEEN 1000000000 AND 1500000000;";
```

TYPE 1의 집 중에서 가격이 100000000 ~ 150000000인 집을 골라야하므로 where에 해당 조건을 추가해준 query이다.

TYPE 1, TYPE 1-1의 결과는 다음과 같다.

```
------- 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

---------------------------------
Enter type : 1
 1ro  101 unit
 6ro  606 unit
 10ro  1010 unit
 11ro  1101 unit
 16ro  101 unit
 21ro  606 unit
 26ro  1101 unit
 31ro  101 unit
 36ro  606 unit
 40ro  1010 unit
 41ro  1101 unit
 46ro  101 unit


----- Subtypes in TYPE 1 -----
        1. TYPE 1-1.
Enter subtype : 1
 10ro  1010 unit
 11ro  1101 unit
```

4-2. TYPE 2

```
const char* query = "SELECT House_district, House_street, House_detail_address\nFROM House\nWHERE House_school_district = 8;";
```

TYPE 2 역시 집의 주소를 가져오는 것이므로 from house 해주었다. 또한 학군의 정보가 8인 집들이므로 where로 학군 정보를 저장하는 House_school_district를 활용해 해당 집들을 찾았다.

4-2-1. TYPE 2-1

```
const char* sub_query = "SELECT House_district, House_street, House_detail_address\nFROM House\nWHERE House_school_district = 8\nAND House_num_of_bedroom >= 4\nAND House_num_of_bathroom = 2;";
```

TYPE 2에서 침실 수와 화장실 수를 추가적으로 검색하는 query를 where과 and로 연결시켜서 구현해주었다.

TYPE 2, TYPE 2-1의 결과는 다음과 같다.

```
------- 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

---------------------------------
Enter type : 2
 Gangnam  2ro  202 unit
 Seocho  9ro  909 unit
 Gangnam  32ro  202 unit
 Seocho  39ro  909 unit


----- Subtypes in TYPE 2 -----
        1. TYPE 2-1.
Enter subtype : 1
 Gangnam  32ro  202 unit
 Seocho  39ro  909 unit
```

4-3. TYPE 3

```
const char* query = "SELECT Agent_name \nFROM Agent\nJOIN Sold ON Agent.Agent_ID = Sold.Sold_Seller_Agent_ID\nWHERE YEAR(Sold_sale_date) = 2022\nGROUP BY Agent.Agent_ID\nORDER BY SUM(Sold_price) DESC\nLIMIT 1;";
```

Agent 정보와 판매된 정보가 모두 필요하므로 Agent와 Sold를 Agent의 ID를 기준으로 join 해주었다. 또한 판매 된 날짜의 년도를 Year를 통해 추출해서 2022년인 것만 추출한다. 그 후 판매 금액의 합을 기준으로 내림차순으로 정렬 후 하나만 출력하면 최고로 많이 판 에이전트가 출력된다.

4-3-1. TYPE 3-1

```
char query_template[] = "SELECT Agent_name \nFROM Agent\nJOIN Sold ON Agent.Agent_ID = Sold.Sold_Seller_Agent_ID\nWHERE YEAR(Sold_sale_date) = 2023\nGROUP BY Agent.Agent_ID\nORDER BY SUM(Sold_price) DESC\nLIMIT %d;";
sprintf(sub_query, query_template, k);
```

숫자를 입력받아 변수 k에 저장한 뒤, LIMIT의 숫자로 활용하면 n명을 출력할 수 있다. 문자열을 선언할 때 이부분은 %d로 비워둔 후, sprintf를 이용해서 삽입해 주었다.

4-3-2. TYPE 3-2

```
const char* sub_query = "SELECT Agent_name" "\nFROM(" "\nSELECT Agent.Agent_name, SUM(Sold.Sold_price) AS TotalSales," "\nROW_NUMBER() OVER(ORDER BY SUM(Sold.Sold_price) ASC) AS RowNum" "\nFROM Agent" "\nJOIN Sold ON Agent.Agent_ID = Sold.Sold_Seller_Agent_ID" "\nWHERE YEAR(Sold.Sold_sale_date) = 2021" "\nGROUP BY Agent.Agent_ID" "\n) AS RankedAgents" "\nWHERE RowNum <= (SELECT ROUND(COUNT(*) * 0.1) FROM(" "\nSELECT Agent.Agent_ID" "\nFROM Agent" "\nJOIN Sold ON Agent.Agent_ID = Sold.Sold_Seller_Agent_ID" "\nWHERE YEAR(Sold.Sold_sale_date) = 2021" "\nGROUP BY Agent.Agent_ID" "\n) AS TotalAgents);";
```

Rownumber를 활용해서 10%를 가려낸다. 아래의 where문에서는 2021년에 판매한 것이 있는 에이전트의 숫자를 agent, sold table을 join한 뒤, count해서 센다. 후에 0.1을 곱하여서서 10%의 숫자를 골라낸다. 위에서는 마찬가지로 두 table을 join한 뒤에 판매금액을 기준으로 오름차순 정렬한다. 후에 출력하는 개수는 앞에서 계산한 10%에 해당하는 숫자만 출력하도록 한다.

TYPE 3, TYPE 3-1, TYPE 3-2의 결과는 다음과 같다.

```
------- 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

---------------------------------
Enter type : 3
 agent5


----- Subtypes in TYPE 3 -----
        1. TYPE 3-1.
        2. TYPE 3-2.
Enter subtype : 1

** Then find the top k agents in the year 2023 by total won value. **
 Which K? : 5
 agent18
 agent11
 agent8
 agent5
 agent4


----- Subtypes in TYPE 3 -----
        1. TYPE 3-1.
        2. TYPE 3-2.
Enter subtype : 2
 agent6
 agent8
```

4-4. TYPE 4

```
const char* query = "SELECT Agent_name, AVG(Sold_price) AS avg_price, AVG(DATEDIFF(Sold_sale_date, Sold_registration_date)) AS avg_market_time\nFROM Agent\nJOIN Sold ON Agent.Agent_ID = Sold.Sold_Seller_Agent_ID\nWHERE YEAR(Sold_sale_date) = 2022\nGROUP BY Agent.Agent_ID;";
```

우선 Agent와 Sold를 join해서 에이전트 정보와 판매된 매물의 정보를 가져온다. 평균 가

격은 각 agent 들의 2022년 기록을 AVG함수를 통해 평균을 낼 수 있다. 또한 날짜의 평균은 datediff 함수를 통해 등록일과 판매일의 차이를 구한 뒤, AVG를 통해 평균을 내면 된다.

4-4-1. TYPE 4-1

```
const char* sub_query = "SELECT Agent_name, \nMAX(Sold_price) AS max_price\nFROM Agent\nJOIN Sold ON Agent.Agent_ID = Sold.Sold_Seller_Agent_ID\nWHERE YEAR(Sold_sale_date) = 2023\nGROUP BY Agent.Agent_ID;";
```

우선 Agent와 Sold를 join해서 에이전트 정보와 판매된 매물의 정보를 가져온다. 이후 agent id를 기준으로 group으로 묶어 준다. 후에 MAX(Sold_price)를 통해 최고 가격을 출력해주면 된다.

4-4-2. TYPE 4-2

```
const char* sub_query = "SELECT Agent_name, MAX(DATEDIFF(Sold_sale_date, Sold_registration_date)) AS max_market_time\nFROM Agent\nJOIN Sold ON Agent.Agent_ID = Sold.Sold_Seller_Agent_ID\nGROUP BY Agent.Agent_ID;";
```

우선 Agent와 Sold를 join해서 에이전트 정보와 판매된 매물의 정보를 가져온다. 이후 agent id를 기준으로 group으로 묶어 준다. 후에 datediff로 등록일과 판매일의 차이를 구한 뒤, MAX()를 통해 최고 기간을 출력해주면 된다.

TYPE 4, TYPE 4-1의 결과는 다음과 같다.

```
------- 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

---------------------------------
Enter type : 4
 agent10  1500000000.0000  175.0000
 agent11  1600000000.0000  71.0000
 agent19  2400000000.0000  48.0000
 agent3  2700000000.0000  107.0000
 agent5  3050000000.0000  78.0000
 agent6  3600000000.0000  45.0000
 agent9  3900000000.0000  51.0000
 agent13  4300000000.0000  59.0000
 agent15  4500000000.0000  91.0000
 agent17  4700000000.0000  44.0000
 agent20  5000000000.0000  77.0000


----- Subtypes in TYPE 4 -----
        1. TYPE 4-1.
        2. TYPE 4-2.
Enter subtype : 1
 agent7
 agent12
 agent14
 agent1
 agent2
 agent4
 agent5
 agent8
 agent11
 agent18
```

```
----- Subtypes in TYPE 4 -----
        1. TYPE 4-1.
        2. TYPE 4-2.
Enter subtype : 2
 agent6
 agent7
 agent8
 agent9
 agent10
 agent11
 agent12
 agent13
 agent14
 agent15
 agent16
 agent17
 agent18
 agent19
 agent20
 agent1
 agent3
 agent4
 agent5
 agent2
```

4-5. TYPE 5

```
const char* query_1 ="SELECT I.Interior_photo_photo, E.Exterior_photo_photo, F.Floor_plan_photo_photo\n""FROM House H\n""LEFT JOIN Interior_photo I ON H.House_ID = I.Interior_photo_House_ID\n""LEFT JOIN Exterior_photo E ON H.House_ID =
```

```
E.Exterior_photo_House_ID\n""LEFT JOIN Floor_plan_photo F ON H.House_ID =
F.Floor_plan_photo_House_ID\n""WHERE H.House_type = 'studio'\n""ORDER BY
H.House_price DESC\n" "LIMIT 1;";
const char* query_1 ="SELECT I.Interior_photo_photo, E.Exterior_photo_photo,
F.Floor_plan_photo_photo\n""FROM House H\n""LEFT JOIN Interior_photo I ON
H.House_ID = I.Interior_photo_House_ID\n"“LEFT JOIN Exterior_photo E ON H.House_ID =
E.Exterior_photo_House_ID\n""LEFT JOIN Floor_plan_photo F ON H.House_ID =
F.Floor_plan_photo_House_ID\n""WHERE H.House_type = 'one-bedroom'\n" "ORDER BY
H.House_price DESC\n" "LIMIT 1;";
const char* query_1 ="SELECT I.Interior_photo_photo, E.Exterior_photo_photo,
F.Floor_plan_photo_photo\n""FROM House H\n""LEFT JOIN Interior_photo I ON
H.House_ID = I.Interior_photo_House_ID\n"“LEFT JOIN Exterior_photo E ON H.House_ID =
E.Exterior_photo_House_ID\n""LEFT JOIN Floor_plan_photo F ON H.House_ID =
F.Floor_plan_photo_House_ID\n""WHERE H.House_type = 'multi-bedroom'\n"
"ORDER BY H.House_price DESC\n" "LIMIT 1;";
const char* query_1 ="SELECT I.Interior_photo_photo, E.Exterior_photo_photo,
F.Floor_plan_photo_photo\n""FROM House H\n""LEFT JOIN Interior_photo I ON
H.House_ID = I.Interior_photo_House_ID\n"“LEFT JOIN Exterior_photo E ON H.House_ID =
E.Exterior_photo_House_ID\n""LEFT JOIN Floor_plan_photo F ON H.House_ID =
F.Floor_plan_photo_House_ID\n""WHERE H.House_type = 'detach'\n" "ORDER BY
H.House_price DESC\n" "LIMIT 1;";
```

총 4개의 house type에 대해서 내부 사진, 외부, 평면도를 불러오는 query이다. 따라서 house와 Interior_photo, Exterior_photo, Floor_plan과 모두 join한다. 이때 사진을 담은 table은 nulll을 허용하므로 left join을 해야 한다. 그 후 4개의 house type을 각각 따로 where로 탐색한 후 가격기준 내림차순으로 정렬한 뒤 limit으로 하나 만 출력하면 가장 비싼 집의 사진을 출력할 수 있다.

TYPE 5의 결과는 다음과 같다.

```
------- 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

---------------------------------
Enter type : 5
  **studio**
 interior40.jpg  (null)  (null)
  **one-bedroom**
 interior35.jpg  (null)  (null)
  **multi-bedroom**
 (null)  exterior25.jpg  floorplan25.jpg
  **apartment**
 (null)  exterior30.jpg  floorplan30.jpg
```

4-6. TYPE 6

```
int Sold_ID, B_ID = 0, H_ID = 0, A_ID = 0;
```

```
char query_template[] = "SELECT Sale_House_ID, Sale_Seller_Agent_ID, Sale_Seller_ID, Sale_registration_date, Sale_price\nFROM Sale\nWHERE Sale_House_ID = %d;";
sprintf(query, query_template, H_ID);
```

판매한 내용을 기록하기 위해서는 판매 중인 table Sale을 불러와야 한다. H_ID에 판매된 집의 ID를 입력받은 후, 이를 sprinf로 query에 삽입하여 where로 찾고 집의 정보를 가져온다.

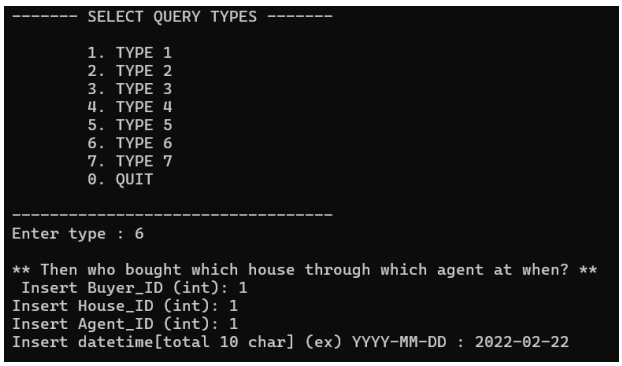
```
state = mysql_query(connection, query);
if (state == 0) {
        sql_result = mysql_store_result(connection);
        while ((sql_row = mysql_fetch_row(sql_result)) != NULL) {
                sprintf(query_1, "INSERT INTO Sold VALUES (%d, %s, %s, %s, %d, '%s', '%s', %s, %d);", Sold_ID++, sql_row[0], sql_row[1], sql_row[2], B_ID, sql_row[3], datetime, sql_row[4], A_ID);
        }

        mysql_free_result(sql_result);
}
```

다음과 같이 sale table의 정보를 가져와서 새로운 query에 삽입한다. 해당 queyr에는 사용자로부터 입력받은 구매자의 ID, 구매일자, 구매자의 에이전트 ID 등도 삽입한다. 이 query는 INSERT INTO Sold VALUES를 통해서 판매된 매물의 정보를 Sold에 삽입한다.

```
char query_2[64];
sprintf(query_2, "DELETE FROM Sale\nWHERE Sale_House_ID = %d;", H_ID);
```

매물이 판매되었기에 판매 중인 매물을 기록하는 Sale table에서는 삭제되어야 한다. 따라서 해당하는 query는 다음과 같이 작성하였다.

TYPE 6의 결과는 다음과 같다.



4-7. TYPE 7

```
int A_ID = 0;
```

```
char A_name[20], A_phone[20];
```

위의 변수들에 Agent의 ID, 이름, 전화번호를 입력받는다. 그 후

```
char query[100];
sprintf(query, "INSERT INTO Agent VALUES(%d, '%s', '%s');", A_ID, A_name, A_phone);
```

sprintf를 통해 query를 만들어서 새로운 Agent를 삽입한다.
TYPE 7의 결과는 다음과 같다.