

[CSE4110] DataBase System Project #2

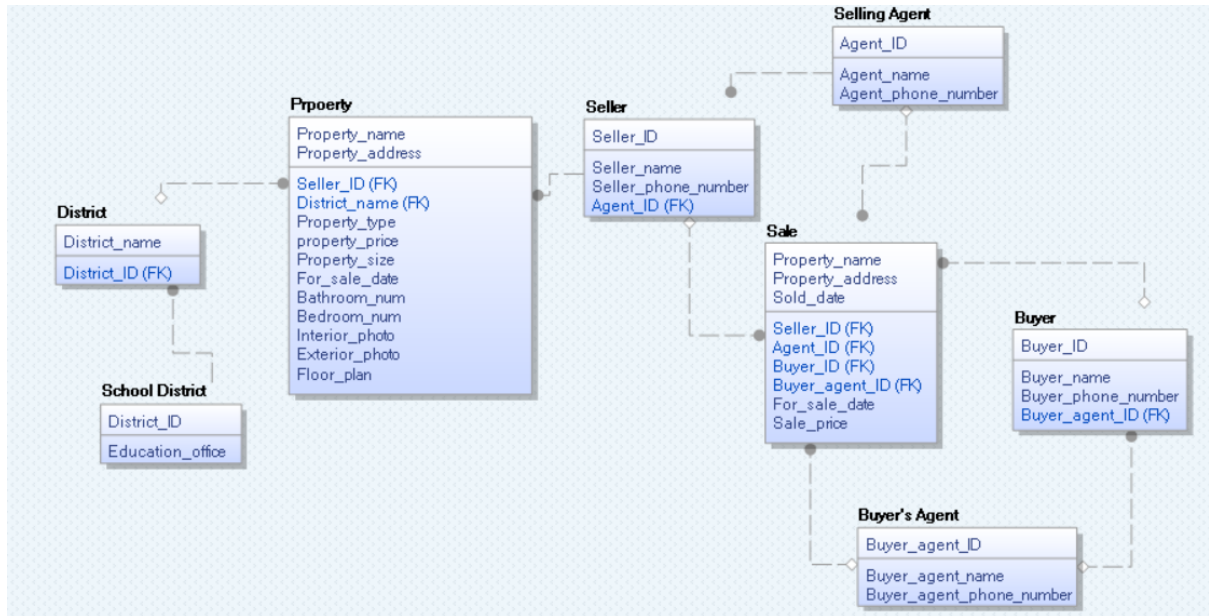
Normalization and Query Processing

"A Real-Estate Office"

20191264 윤성민

1. BCNF Decomposition

-아래의 Schema는 Project #1에서 설계한 Logical Schema에서 data set을 넣었을 때 발생할 수 있는 data redundancy, 혹은 foreign key constraints 위반을 방지하기 위해 조금 더 수정한 Logical Schema이다.



이렇게 설계했던 Logical Schema의 relation들에 대해서 functional dependency를 찾아보고 그에 따른 BCNF를 결정하겠다.

1-1. Seller



-Seller relation에서의 functional dependency는 다음과 같다.

- ① Seller_ID → (Seller_name, Seller_phone_number, Agent_ID)
- ② Seller_phone_number → (Seller_ID, Seller_name, Agent_ID)

먼저 Seller_ID는 해당 relation의 primary key이므로 superkey의 역할을 한다. 또한 Seller마다 모두 phone number가 다르므로 Seller_phone_number 또한 superkey의 역할을 한다. 따라서 Seller relation은 BCNF form이다.

1-2. Selling_Agent



-Selling_Agent relation에서의 functional dependency는 다음과 같다.

① Agent_ID \rightarrow (Agent_name, Agent_phone_number)

② Agent_phone_number \rightarrow (Agent_ID, Agent_name)

먼저 Agent_ID는 해당 relation의 primary key이므로 superkey의 역할을 한다. 또한 Selling_agent모두 phone number가 다르므로 Agent_phone_number 또한 superkey의 역할을 한다. 따라서 Selling_Agent relation은 BCNF form이다.

1-3. Buyer



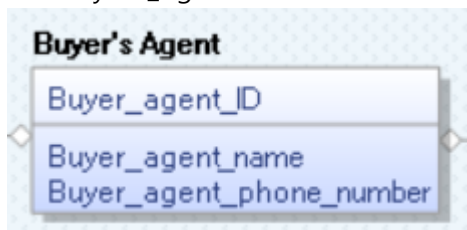
-Buyer relation에서의 functional dependency는 다음과 같다.

① Buyer_ID \rightarrow (Buyer_name, Buyer_phone_number, Buyer_agent_ID)

② Buyer_phone_number \rightarrow (Buyer_ID, Buyer_name, Buyer_agent_ID)

먼저 Buyer_ID는 해당 relation의 primary key이므로 superkey의 역할을 한다. 또한 Buyer마다 모두 phone number가 다르므로 Buyer_phone_number 또한 superkey의 역할을 한다. 따라서 Buyer relation은 BCNF form이다.

1-4. Buyer's_Agent



-Buyer's_Agent relation에서의 functional dependency는 다음과 같다.

① Buyer_agent_ID \rightarrow (Buyer_agent_name, Buyer_agent_phone_number)

② Buyer_agent_phone_number \rightarrow (Buyer_agent_ID, Buyer_agent_name)

먼저 Buyer_agent_ID는 해당 relation의 primary key이므로 superkey의 역할을 한다. 또한 Buyer_agent 모두 phone number가 다르므로 Buyer_agent_phone_number 또한 superkey의 역할을 한다. 따라서 Buyer's_Agent relation은 BCNF form이다.

1-5. District



-District relation에서의 functional dependency는 다음과 같다.

① District_name → District_ID

서울시의 행정구역의 이름을 나타내는 District_name은 총 25개의 구로 구성되어 있고 각 구역마다 1~11학군이 정해져 있다. 따라서 District_name은 해당 relation에서 primary key의 역할을 하므로 superkey의 역할 또한 한다. 따라서 District relation은 BCNF form이다.

1-6. School_District



-School_District relation에서의 functional dependency는 다음과 같다.

① District_ID → Education_office

② Education_office → District_ID

서울시의 1~11학군에는 각 학군을 담당하는 교육청이 1개씩 존재한다. 이에 따라 해당 relation에서 primary key인 District_ID뿐만 아니라 Education_office 또한 superkey의 역할을 한다. 따라서 School_District relation은 BCNF form이다

1-7. Sale



-Sale relation에서의 functional dependency는 다음과 같다.

① (Property_name, Property_address, Sold_date)

→ (Seller_ID, Agent_ID, Buyer_ID, Buyer_agent_ID, For_sale_date, Sale_price)

Sale relation은 부동산의 판매기록을 담고 있는 relation이다. 따라서 과거에 팔렸었던 매물이 시간이 지나 또 한 번 팔리는 경우가 생길 수도 있고 같은 판매자가 여러 개의 매물을 팔 수도 있다. 따라서 (Property_name, Property_address, Sold_date)의 3개의 attribute가 primary key를 구성하여 매물의 이름, 도로명 주소, 그리고 판매된 날짜로 모든 기록을 구분할 수 있게 된다. 따라서 (Property_name, Property_address, Sold_date)는 primary key로 superkey의 역할을 한다. 그러므로 Sale relation은 BCNF form이다.

1-8. Property



-Property relation에서의 functional dependency는 다음과 같다.

① $(Property_name, Property_address) \rightarrow (Seller_ID, District_name, Property_type, Property_price, Property_size, For_sale_date, Bathroom_num, Bedroom_num, Interior_photo, Exterior_photo, Floor_plan)$

② $Property_address \rightarrow District_name$

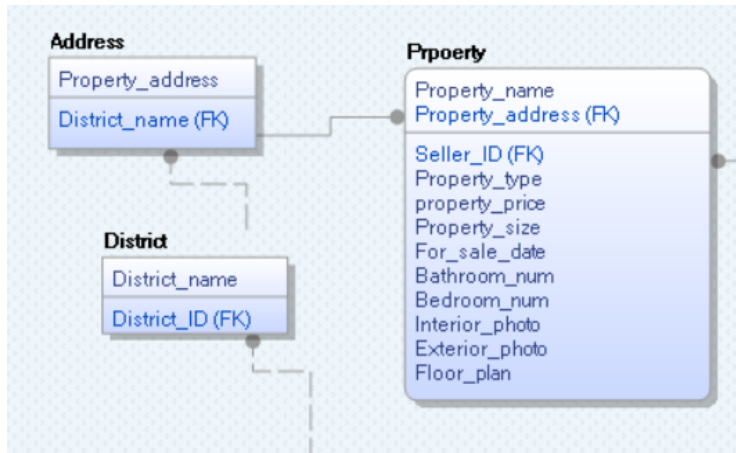
먼저 ①의 functional dependency에서는 $(Property_name, Property_address)$ 가 primary key로 superkey의 역할을 하므로 BCNF의 조건을 만족한다. 하지만 ②의 functional dependency의 경우 trivial하지 않으며 $property_address$ 는 primary key가 아니므로 superkey의 역할을 하지 못한다. 따라서 Property는 ②에 대해서 BCNF Decomposition이 필요하다. BCNF Decomposition은 아래와 같은 Algorithm을 이용해 수행하였다.

```

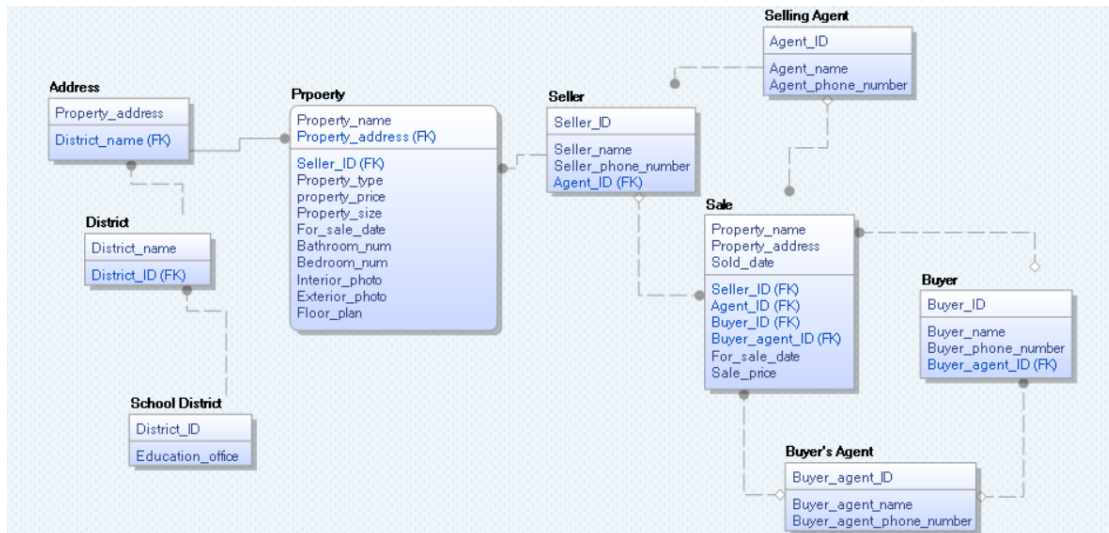
result := {R};
done := false;
compute F+;
while (not done) do
    if (there is a schema  $R_i$  in result that is not in BCNF)
        then begin
            let  $\alpha \rightarrow \beta$  be a nontrivial functional dependency that
            holds on  $R_i$  such that  $\alpha^+$  does not contain  $R_i$ 
            and  $\alpha \cap \beta = \emptyset$ ;
            result := (result -  $R_i$ )  $\cup$  ( $R_i - \beta$ )  $\cup$  ( $\alpha, \beta$ );
        end
    else done := true;
end
    
```

Note: each R_i is in BCNF, and decomposition is lossless-join.

BCNF Decomposition을 통해 $Property_address$ 를 primary key로 하고 $District_name$ 을 attribute으로 갖는 Address라는 새로운 relation을 만든다. Address relation은 서울시의 주소를 저장하고 있는 relation이게 되고 각 $property_address$ 가 어느 $District_name$, 즉 어느 행정구역에 속하는 주소인지를 나타내 주는 relation이 된다. 이 후 Property relation이 Address의 Property의 $Property_address$ 를 reference하도록 하고 Address relation이 District relation의 $District_name$ 을 reference하도록 한다. 이러한 과정을 통해 decomposition을 완료하면 Property와 Address, District 이 3개의 relation은 아래와 같은 형태를 갖는다.

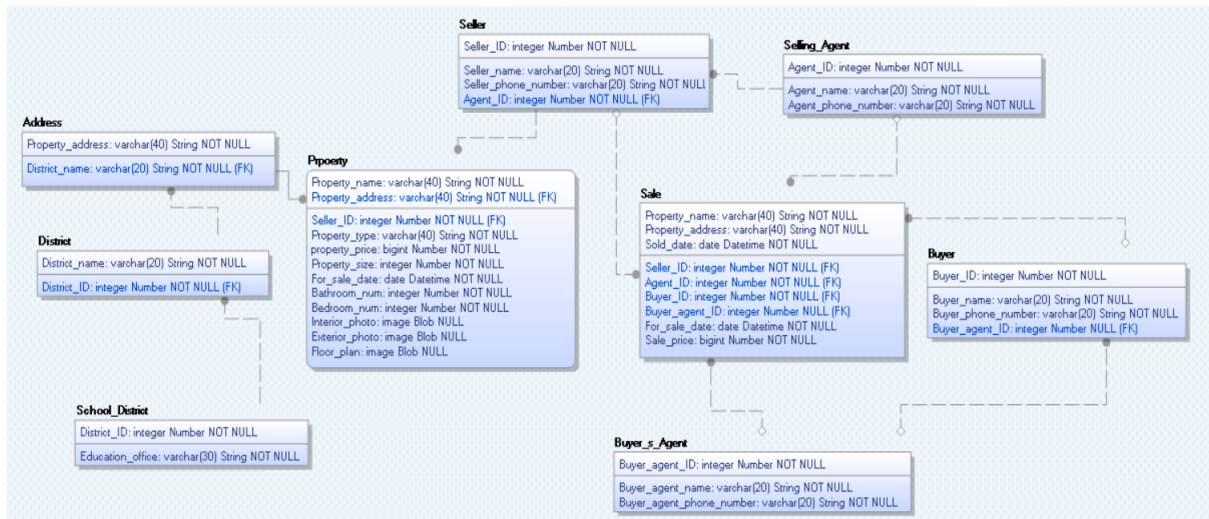


이렇게 이전에 설계한 Logical Schema에서 각 relation마다 functional dependency를 확인하고 BCNF의 만족 여부를 확인해 만족하지 않는 경우 BCNF Decomposition까지 완료한 전체 Logical Schema는 다음과 같다.

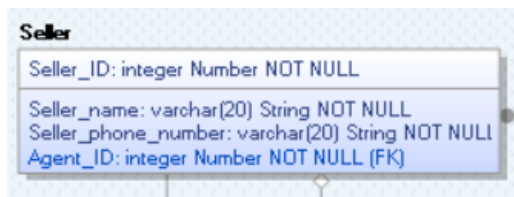


2. Physical Schema

-위에서 BCNF Decomposition을 통해 얻은 Logical Schema를 Physical Schema로 변환하면 다음과 같다.



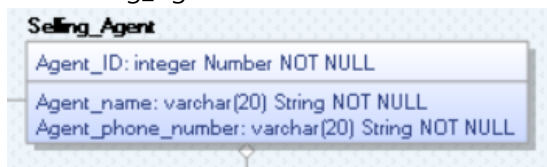
2-1. Seller



-부동산 판매자에 대한 정보를 저장하는 Seller relation이다.

- ① Seller_ID: 각 판매자에게 고유하게 부여되는 ID로 int형을 가지며 primary key이므로 NULL을 허용하지 않는다.
- ② Seller_name: 각 판매자의 이름을 나타낸다. varchar(20)의 문자열 자료형을 가지며 모든 판매자는 이름을 갖기 때문에 NULL을 허용하지 않는다.
- ③ Seller_phone_number: 각 판매자의 핸드폰 번호를 나타낸다. int형을 가지며 모든 판매자는 핸드폰 번호를 갖기 때문에 NULL을 허용하지 않는다.
- ④ Agent_ID: 각 판매자를 담당하는 판매 중개인의 고유 ID로 int형을 가지며 Selling_agent에서 reference하는 foreign key이다. 모든 판매자는 한 명의 판매 중개인을 가져야 하므로 NULL을 허용하지 않는다. 모든 판매자는 한 명의 판매 중개인을 갖지만 판매자를 담당하고 있지 않은 판매 중개인이 존재할 수 있으므로 Sale과 Selling_Agent는 Many-to-One의 relationship을 갖는다.

2-2. Selling_Agent

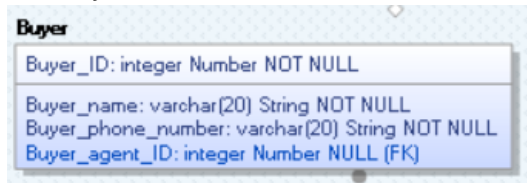


-판매 중개인에 대한 정보를 저장하는 Selling_Agent relation이다.

- ① Agent_ID: 각 판매 중개인에게 고유하게 부여되는 ID로 int형을 가지며 primary key이므로 NULL을 허용하지 않는다.

- ② Agent_name: 각 판매 중개인의 이름을 나타낸다. varchar(20)의 문자열 자료형을 가지며 모든 판매 중개인은 이름을 갖기 때문에 NULL을 허용하지 않는다.
- ③ Agent_phone_number: 각 판매 중개인의 핸드폰 번호를 나타낸다. int형을 가지며 모든 판매 중개인은 핸드폰 번호를 갖기 때문에 NULL을 허용하지 않는다.

2-3. Buyer

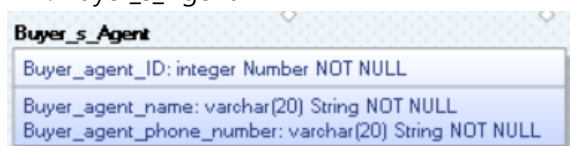


Buyer	
Buyer_ID	integer Number NOT NULL
Buyer_name	varchar(20) String NOT NULL
Buyer_phone_number	varchar(20) String NOT NULL
Buyer_agent_ID	integer Number NULL (FK)

-부동산 구매자에 대한 정보를 저장하는 Buyer relation이다.

- ① Buyer_ID: 각 구매자에게 고유하게 부여되는 ID로 int형을 가지며 primary key이므로 NULL을 허용하지 않는다.
- ② Buyer_name: 각 구매자의 이름을 나타낸다. varchar(20)의 문자열 자료형을 가지며 모든 구매자는 이름을 갖기 때문에 NULL을 허용하지 않는다.
- ③ Buyer_phone_number: 각 구매자의 핸드폰 번호를 나타낸다. int형을 가지며 모든 구매자는 핸드폰 번호를 갖기 때문에 NULL을 허용하지 않는다.
- ④ Buyer_Agent_ID: 각 구매자를 담당하는 구매 대리인의 고유 ID로 int형을 가지며 Buyer_s_agent에서 reference하는 foreign key이다. 구매 대리인을 고용하지 않고 구매를 진행하는 판매자가 존재할 수 있으므로 NULL을 허용한다. 구매 대리인을 고용한 구매자는 한 명의 구매 대리를 갖지만 구매자를 담당하고 있지 않은 구매 대리인이 존재할 수 있으므로 Buyer와 Buyer_s_Agent는 Many-to-One의 relationship을 갖는다.

2-4. Buyer_s_Agent

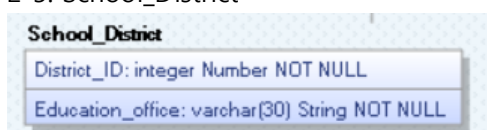


Buyer_s_Agent	
Buyer_agent_ID	integer Number NOT NULL
Buyer_agent_name	varchar(20) String NOT NULL
Buyer_agent_phone_number	varchar(20) String NOT NULL

-구매 대리인에 대한 정보를 저장하는 Buyer_s_Agent relation이다.

- ① Buyer_Agent_ID: 각 구매 대리인에게 고유하게 부여되는 ID로 int형을 가지며 primary key이므로 NULL을 허용하지 않는다.
- ② Buyer_Agent_name: 각 구매 대리인의 이름을 나타낸다. varchar(20)의 문자열 자료형을 가지며 모든 구매 대리인은 이름을 갖기 때문에 NULL을 허용하지 않는다.
- ③ Buyer_Agent_phone_number: 각 구매 대리인의 핸드폰 번호를 나타낸다. int형을 가지며 모든 구매 대리인은 핸드폰 번호를 갖기 때문에 NULL을 허용하지 않는다.

2-5. School_District

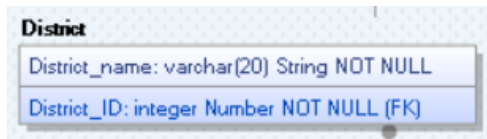


School_District	
District_ID	integer Number NOT NULL
Education_office	varchar(30) String NOT NULL

-각 학군을 담당하는 교육청에 대한 정보를 저장하는 School_District relation이다.

- ① District_ID: 1~11학군의 각 학군을 나타낸다. int형을 가지며 primary key이므로 NULL을 허용하지 않는다.
- ② Educaion_office: 각 학군을 담당하는 교육청의 이름을 나타낸다. varchar(30)의 문자열 자료형을 가지며 NULL을 허용하지 않는다.

2-6. District



-서울시의 25개의 행정구역에 해당하는 학군을 저장하는 District relation이다.

- ① District_name: 서울시의 25개의 행정구역의 이름을 나타낸다. varchar(20)의 문자열 자료형을 가지며 primary key이므로 NULL값을 허용하지 않는다.
- ② District_ID: 1~11학군의 각 학군을 나타낸다. int형을 가지며 School_District에서 reference하는 foreign key이다. 각 행정구역은 1개의 학군을 가지므로 NULL을 허용하지 않는다. 하나의 행정구역에는 하나의 학군이 지정되지만 하나의 학군에는 여러 행정구역이 포함된다. 따라서 District와 School_District는 Many-to-One의 relationship을 갖는다.

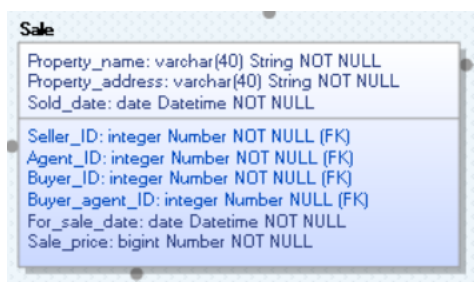
2-7. Address



-서울시의 각 부동산의 주소에서 행정 구역 하위의 주소와 행정구역을 저장하는 Address relation이다(이번 프로젝트에서는 모든 서울시의 행정구역에 포함된 주소를 담기 어렵기 때문에 일부만을 data set으로 설정했다).

- ① Property_address: 서울시의 각 부동산의 주소에서 행정구역 하위의 주소(예시: 대흥동 804)를 나타낸다. varchar(40)의 문자열 자료형을 가지며 primary key이므로 NULL을 허용하지 않는다.
- ② District_name: 서울시의 25개의 행정구역의 이름을 나타낸다. varchar(20)의 문자열 자료형을 가지며 District에서 reference하는 foreign key이다. 각 Property_address는 하나의 District_name을 가지므로 NULL값을 허용하지 않는다. 하나의 행정구역에는 여러 property_address가 존재하므로 Address와 District는 Many-to-one의 relationship을 갖는다.

2-8. Sale



-부동산 판매 기록을 저장하는 Sale relation이다.

- ① Property_name: 판매된 부동산의 이름을 나타낸다. varchar(40)의 문자열 자료형을 가지며 primary key에 속하므로 NULL을 허용하지 않는다.
- ② Property_address: 판매된 부동산의 행정구역을 제외한 주소를 나타낸다. varchar(40)의 문자열 자료형을 가지며 primary key에 속하므로 NULL을 허용하지 않는다.
- ③ Sold_date: 부동산이 판매된 날짜를 나타낸다. Datetime의 domain의 Date 자료형을 가지며 primary key에 속하므로 NULL을 허용하지 않는다.
- ④ Seller_ID: 부동산을 판매한 판매자의 고유 ID를 나타낸다. int형을 가지며 Seller에서 reference한 foreign key이다. 판매기록에는 판매자가 반드시 포함되어야 하므로 NULL을 허용하지 않는다.
- ⑤ Agent_ID: 부동산 거래에 참여한 판매 중개인의 고유 ID를 나타낸다. int형을 가지며 Selling_Agent에서 reference한 foreign key이다. 판매 기록에는 판매 중개인이 반드시 포함되어야 하므로 NULL을 허용하지 않는다.
- ⑥ Buyer_ID: 부동산을 구매한 구매자의 고유 ID를 나타낸다. int형을 가지며 Buyer에서 reference한 foreign key이다. 판매기록에는 구매자가 반드시 포함되어야 하므로 NULL을 허용하지 않는다.
- ⑦ Buyer_Agent_ID: 부동산 거래에 참여한 구매 대리인의 고유 ID를 나타낸다. int형을 가지며 Buyer_s_Agent에서 reference한 foreign key이다. 구매 대리인을 고용하지 않는 구매자가 존재할 수 있으므로 NULL을 허용한다
- ⑧ For_sale_date: 판매된 부동산이 판매 가능으로 등록되었던 날짜를 나타낸다. Date의 자료형을 가지며 판매 기록에는 반드시 포함되어야 하므로 NULL을 허용하지 않는다.
- ⑨ Sale_price: 판매된 부동산의 가격을 나타낸다. 억 단위의 데이터가 들어갈 수 있어 big int형을 가지며 판매 가격 또한 반드시 판매 기록에 포함되어야 하므로 NULL을 허용하지 않는다.

2-9. Property

Property	
Property_name	: varchar(40) String NOT NULL
Property_address	: varchar(40) String NOT NULL (FK)
Seller_ID	: integer Number NOT NULL (FK)
Property_type	: varchar(40) String NOT NULL
property_price	: bigint Number NOT NULL
Property_size	: integer Number NOT NULL
For_sale_date	: date Datetime NOT NULL
Bathroom_num	: integer Number NOT NULL
Bedroom_num	: integer Number NOT NULL
Interior_photo	: image Blob NULL
Exterior_photo	: image Blob NULL
Floor_plan	: image Blob NULL

-판매 가능한 부동산들의 정보를 저장하는 Property relation이다.

- ① Property_name: 판매 가능한 부동산의 이름을 나타낸다. varchar(40)의 문자열 자료형을 가지며 primary key에 속하므로 NULL을 허용하지 않는다.
- ② Property_address: 판매 가능한 부동산 주소를 나타낸다. varchar(40)의 문자열 자료형을 가지며 primary key에 속하므로 NULL을 허용하지 않는다.
- ③ Seller_ID: 부동산을 판매하는 판매자의 고유 ID를 나타낸다. int형을 가지며 Seller에서 reference한 foreign key이다. 판매 가능한 부동산에 대해서는 판매자가 반드시 포함되어야 하므로

NULL을 허용하지 않는다.

④ Property_type: 판매 가능한 부동산의 유형을 나타낸다. varchar(40)의 문자열 자료형을 가지며 판매 가능 부동산의 유형에 따라 다른 attribute의 값이 정해질 수 있으므로 NULL을 허용하지 않는다.

⑤ Property_price: 판매 가능한 부동산의 가격을 나타낸다. 억 단위의 데이터가 들어갈 수 있어 big int형을 가지며 부동산의 가격에 대한 정보가 있어야 판매가 가능하므로 NULL을 허용하지 않는다.

⑥ Property_size: 판매 가능한 부동산의 면적(평수)을 나타낸다. int형을 가지며 부동산의 정보에서 해당 부동산의 크기는 포함되어야 하므로 NULL을 허용하지 않는다.

⑦ For_sale_date: 부동산이 판매 가능으로 등록된 날짜를 나타낸다. Date의 자료형을 가지며 해당 부동산이 판매되었을 때 판매기록을 남기기 위해 반드시 포함되어야 하므로 NULL을 허용하지 않는다.

⑧ Bathroom_num & Bedroom_num: 판매 가능한 부동산에 포함된 화장실 수와 침실 수를 나타낸다. int형을 가지며 판매 정보에서 필요한 정보이므로 NULL을 허용하지 않는다.

⑨ Interior_photo & Exterior_photo & Floor_plan: 각각 부동산의 내부사진, 외부사진, 평면도의 사진을 나타낸다. blob domain의 image의 자료형을 가진다. property_type에 따라 studio, one-bedroom apartment는 한 장 이상의 interior_photo만을, multi-bedroom apartment, detached houses는 한 장이상의 exterior_photo와 floorplan만을 반드시 가져야 하므로 반드시 있어야 할 사진을 제외하고는 필수는 아니기 때문에 모두 NULL값을 허용한다.

3. CRUD Implementation

-Visual Studio를 통해 sql 쿼리를 수행하는 코드를 작성하기 전에 쿼리들을 수행하기 위해 필요한 database와 그에 포함되는 table들을 CREATE하고 data set들을 INSERT하는 과정이 필요하다. 또한 쿼리 수행을 다 마친 뒤 해당 table들을 모두 DROP하는 과정 또한 필요하다. 이를 텍스트 파일에 CRUD 쿼리로 작성한 뒤 코드로 해당 텍스트 파일을 읽어 수행하면 된다.

CREATE DATABASE project;

USE project;

먼저 위와 같이 project라는 이름을 가진 database를 생성하여 해당 database를 사용할 것임을 작성한다.

```
CREATE TABLE School_District ( District_ID INTEGER NOT NULL, Education_office VARCHAR(30) NOT NULL, PRIMARY KEY (District_ID) );
CREATE TABLE District ( District_name VARCHAR(20) NOT NULL, District_ID INTEGER NOT NULL, FOREIGN KEY (District_ID) REFERENCES School_District(District_ID), PRIMARY KEY (District_name) );
CREATE TABLE Address ( Property_address VARCHAR(40) NOT NULL, District_name VARCHAR(20) NOT NULL, PRIMARY KEY (Property_address), FOREIGN KEY (District_name) REFERENCES District(District_name) );
CREATE TABLE Selling_Agent ( Agent_ID INTEGER NOT NULL, Agent_name VARCHAR(20) NOT NULL, Agent_phone_number VARCHAR(20) NOT NULL, PRIMARY KEY (Agent_ID) );
CREATE TABLE Buyer_s_Agent ( Buyer_agent_ID INTEGER NOT NULL, Buyer_agent_name VARCHAR(20) NOT NULL, Buyer_agent_phone_number VARCHAR(20) NOT NULL, PRIMARY KEY (Buyer_agent_ID) );
CREATE TABLE Seller ( Seller_ID INTEGER NOT NULL, Seller_name VARCHAR(20) NOT NULL, Seller_phone_number VARCHAR(20) NOT NULL, Agent_ID INTEGER NOT NULL, PRIMARY KEY (Seller_ID), FOREIGN KEY (Agent_ID) REFERENCES Selling_Agent(Agent_ID) );
CREATE TABLE Buyer ( Buyer_ID INTEGER NOT NULL, Buyer_name VARCHAR(20) NOT NULL, Buyer_phone_number VARCHAR(20) NOT NULL, Buyer_agent_ID INTEGER, PRIMARY KEY (Buyer_ID), FOREIGN KEY (Buyer_agent_ID) REFERENCES Buyer_s_Agent(Buyer_agent_ID) );
CREATE TABLE Property ( Property_name VARCHAR(40) NOT NULL, Property_address VARCHAR(40) NOT NULL, Seller_ID INTEGER NOT NULL, Property_type VARCHAR(40) NOT NULL, Property_price BIGINT NOT NULL, Property_size INTEGER NOT NULL, For_sale_date DATE NOT NULL, Bathroom_num INTEGER NOT NULL, Bedroom_num INTEGER NOT NULL, Interior_photo BLOB NULL, Exterior_photo BLOB NULL, Floor_plan BLOB NULL, PRIMARY KEY (Property_name, Property_address), FOREIGN KEY (Property_address) REFERENCES Address(Property_address), FOREIGN KEY (Seller_ID) REFERENCES Seller(Seller_ID) );
CREATE TABLE Sale ( Property_name VARCHAR(40) NOT NULL, Property_address VARCHAR(40) NOT NULL, Sold_date DATE NOT NULL, Seller_ID INTEGER NOT NULL, Agent_ID INTEGER NOT NULL, Buyer_ID INTEGER NOT NULL, Buyer_agent_ID INTEGER NULL, For_sale_date DATE NOT NULL, Sale_price BIGINT NOT NULL, PRIMARY KEY (Property_name, Property_address, Sold_date), FOREIGN KEY (Seller_ID) REFERENCES Seller(Seller_ID), FOREIGN KEY (Agent_ID) REFERENCES Selling_Agent(Agent_ID), FOREIGN KEY (Buyer_ID) REFERENCES Buyer(Buyer_ID), FOREIGN KEY (Buyer_agent_ID) REFERENCES Buyer_s_Agent(Buyer_agent_ID) );
```

다음은 위에서 작성한 physical schema의 relation들을 토대로 필요한 table들을 CREATE한다.

형식은 "CREATE TABLE <relation이름> <필요한 attribute DATE_TYPE NULL여부> PRIMARY KEY <primary key>"와 같은 형식으로 적어준다. Foreign key가 존재한다면 어느 table에서 reference하는지 명시해주면 된다. CREATE 할 때 주의할 점은 table들의 CREATE 순서이다. CREATE를 할 때는 referenced된 table들을 먼저 만들어 주어야 reference하는 table들을 오류 없이 만들 수 있으며 이후 INSERT를 통해 data들을 넣을 때도 오류 없이 INSERT 할 수 있다. 따라서 CREATE의 순서는 위에서 나와 있듯이 School_District → District → Address → Selling_Agent → Buyer_s_Agent → Seller → Buyer → Property → Sale 순으로 CREATE하였다.

```
INSERT INTO School_District (District_ID, Education_office) VALUES (1, 'Dongbu-office'), (2, 'Seobu-office'), (3, 'Nambu-office'), (4, 'Bukbu-office'), (5, 'Jungbu-office'), (6, 'GangdongSongpa-office'), (7, 'Gangseo-office'), (8, 'Gangnam-office'), (9, 'DongjakGwanak-office'), (10, 'SeongdongGwangjang-office'), (11, 'Seongbuk-office');
```

다음은 각 table들에 필요한 data들을 INSERT 해야 한다. 위와 같이 INSERT를 하는 데 이때 "INSERT INTO <table 이름> (<attribues>) VALUES (data sets), (data sets)" 와 같은 형식으로 필요한 만큼 data를 넣어주면 된다. INSERT 또한 CREATE 생성 순서에 맞춰 data들을 INSERT 해준다. 위의 코드는 School_District table에 필요한 table들을 삽입하는 CRUD 쿼리문이다.

```
DROP TABLE IF EXISTS Sale;
DROP TABLE IF EXISTS Property;
DROP TABLE IF EXISTS Buyer;
DROP TABLE IF EXISTS Seller;
DROP TABLE IF EXISTS Buyer_s_Agent;
DROP TABLE IF EXISTS Selling_Agent;
DROP TABLE IF EXISTS Address;
DROP TABLE IF EXISTS District;
DROP TABLE IF EXISTS School_District;
DROP DATABASE project;
```

마지막으로 모든 sql 쿼리문을 마치고 database 사용을 종료하면 모든 table들을 삭제하기 위해 "DROP TABLE IF EXIST <table 이름>"의 형식을 DROP 쿼리를 사용한다. 또한 DROP DATABASE project를 통해 project이름의 database 또한 삭제한다. 이 때 table을 DROP 할 때는 CREATE에서의 순서와는 반대로 reference한 table 먼저 DROP 해준다. 따라서 Sale → Property → Buyer → Seller → Buyer_s_Agent → Selling_Agent → Address → District → School_District 순으로 DROP 하게 쿼리를 작성한다.

4. SQL Query (C++)

-이제 주어진 쿼리들을 수행할 수 있게끔 sql 쿼리를 C++ 코드에 작성해준다. 먼저 C++코드를 MySQL database를 연결해준 뒤 텍스트 파일을 열어 작성한 CRUD 텍스트 파일을 읽어와 table을 CREATE하고 dataset을 INSERT한다.

```
FILE* fp = fopen("crud_file.txt", "rt");    // open CRUD file.
char line[MAX_LEN];

if (mysql_init(&conn) == NULL)
    printf("mysql_init() error!");

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

    while (fgets(line, sizeof(line), fp) != NULL) {
        if (!strcmp(line, "$$$\n"))
            break;
        mysql_query(connection, line);
    }

    if (mysql_select_db(&conn, "project")) {
        printf("%d ERROR : %s\n", mysql_errno(&conn), mysql_error(&conn));
        return 1;
    }
}
```

이후에는 이번 프로젝트에서 요구하는 13개의 쿼리를 수행할 sql을 작성하여 문자열 변수에 저장한 뒤 mysql_store_result 함수와 mysql_fetch_row 함수를 통해 결과값을 내고 화면에 출력하도록 한다. 이제부터 각 쿼리 항목에 해당하는 sql에 대해 설명하겠다.

① TYPE 1

```
if (q_num == 1) {
    printf("---- TYPE 1 ----\n\n");
    printf("Find address of homes for sale in the district \"Mapo\".\n");
    query = "SELECT A.Property_address "
            "FROM Property P "
            "JOIN Address A ON P.Property_address = A.Property_address "
            "JOIN District D ON A.District_name = D.District_name "
            "WHERE D.District_name = 'Mapo';";
    state = mysql_query(connection, query.c_str());
    if (state != 0) {
        printf("Query Error: %s\n", mysql_error(connection));
        continue;
    }
}
```

먼저 TYPE 1은 행정구역이 "Mapo"인 판매 가능한 집들의 주소를 찾아야 하는 쿼리이다. 따라서 district_name을 attribute로 하는 District relation과 Address relation을 join하고 Property_address를 attribute로 하는 Property relation과 Address relation을 join하여 주소가 District_name이 "Mapo"인 집의 property_address를 판매 가능한 집들의 정보를 담은 Property relation에서 select하도록 sql을 작성하였다. TYPE 1에 대한 결과는 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

1

----- TYPE 1 -----

Find address of homes for sale in the district "Mapo".

125 Mapo Road

② TYPE 1-1

```
if (sub_qnum == 1) {
    printf("---- TYPE 1-1 ----\n\n");
    printf("Find homes costing between ₩1,000,000,000 and ₩1,500,000,000.\n");
    query = "SELECT A.Property_address "
            "FROM Property P "
            "JOIN Address A ON P.Property_address = A.Property_address "
            "JOIN District D ON A.District_name = D.District_name "
            "WHERE D.District_name = 'Mapo' "
            "AND P.property_price BETWEEN 1000000000 AND 1500000000;";
    state = mysql_query(connection, query.c_str());
    if (state != 0) {
        printf("Query Error: %s\n", mysql_error(connection));
        continue;
    }
}
```

다음으로 TYPE 1-1은 TYPE 1의 subquery로 10억에서 15억 사이의 매물의 가격을 가져야 하는 조건이 추가되었다. 따라서 앞에서 적은 쿼리문에서 where구문에 AND 작성 이후 property_price가 10억에서 15억 사이의 값을 갖는 property_address를 select하도록 작성하였다. TYPE 1-1에 대한 결과는 다음과 같다. (TYPE 1-1의 경우 TYPE 1의 subquery이므로 아래와 같은 형식으로 나오게끔 작성하였다)

----- SUBTYPES in TYPE1 -----

1. TYPE 1-1

1

----- TYPE 1-1 -----

Find homes costing between ?1,000,000,000 and ?1,500,000,000.

125 Mapo Road

③ TYPE 2

```
else if (q_num == 2) {
    printf("---- TYPE 2 ----\n\n");
    printf("Find the address of homes for sale in the 8th school district.\n");
    query = "SELECT Property_address FROM Property WHERE Property_address IN(SELECT Property_address FROM Address";
    query += "WHERE District_name IN(SELECT District_name FROM District WHERE District_ID = 8)); ";
    state = mysql_query(connection, query.c_str());
    if (state != 0) {
        printf("Query Error: %s\n", mysql_error(connection));
        continue;
    }
}
```

다음은 TYPE 2로 이번에는 학군이 8학군인 판매 가능한 매물의 주소를 찾아야 한다. Where과 IN을 사용하여 먼저 District_ID가 8인 District_name을 District relation에서 select하고 select한 District_name을 갖는 property_address를 Address에서 select해서 이러한 property_address를 Property relation에서 select하도록 작성하였다. 이에 대한 결과는 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

-----
2
---- TYPE 2 ----

Find the address of homes for sale in the 8th school district.
140 Gangnam Boulevard
```

④ TYPE 2-1

```
if (sub_qnum == 1) {
    printf("---- TYPE 2-1 ----\n\n");
    printf("Find properties with 4 or more bedrooms and 2 bathrooms.\n");
    query = "SELECT Property_address FROM Property WHERE Property_address IN (SELECT Property_address FROM Address";
    query += "WHERE District_name IN(SELECT District_name FROM District WHERE District_ID = 8)) AND Bedroom_num >= 4 AND Bathroom_num = 2; ";
    state = mysql_query(connection, query.c_str());
    if (state != 0) {
        printf("Query Error: %s\n", mysql_error(connection));
        continue;
    }
}
```

다음으로 TYPE 2-1은 TYPE 2의 subquery로 위의 TYPE 2의 쿼리에서 침실의 개수가 4개 이상, 그리고 화장실의 개수가 2개라는 조건이 추가되었다. 따라서 AND를 사용해 Property relation안의 attribute들인 Bedroom_num과 Bathroom_num이 각각 4 이상이고 2인 property_address를 select하도록 작성하였다.

```
----- SUBTYPES in TYPE2 -----

1. TYPE 2-1

1
---- TYPE 2-1 ----

Find properties with 4 or more bedrooms and 2 bathrooms.
140 Gangnam Boulevard
```


⑤ TYPE 3

```
if (q_num == 3) {
    printf("---- TYPE 3 ----\n\n");
    printf("Find the name of the agent who has sold the most properties in the year 2022 by total won value.\n");
    query = "SELECT Agent_name "
            "FROM Selling_Agent "
            "JOIN (SELECT Agent_ID "
            "FROM Sale "
            "WHERE YEAR(Sold_date) = 2022 "
            "GROUP BY Agent_ID "
            "ORDER BY SUM(Sale_price) DESC "
            "LIMIT 1) AS TopAgent ON Selling_Agent.Agent_ID = TopAgent.Agent_ID;";
    state = mysql_query(connection, query.c_str());
    if (state != 0) {
        printf("Query Error: %s\n", mysql_error(connection));
        continue;
    }
}
```

TYPE 3에서는 2022년에 판매한 매물의 가격 총합이 가장 높은 판매 중개인의 이름을 찾아야 한다. 판매된 날짜가 2022년이기 때문에 Sale relation의 attribute인 Sold_date가 2022인 Agent_ID들을 기준으로 그룹화하고 각 판매중개인의 sale_price의 합계를 SUM을 통해 계산하여 합계를 기준으로 ORDER BY를 통해 내림차순(DESC)으로 정렬하도록 한다. 가장 큰 금액을 가진 한 명의 agent만을 선택하기 위해 LIMIT 1을 사용한다. 이렇게 서버쿼리를 작성한 후 서버쿼리의 결과인 TopAgent와 Selling_Agent 테이블을 Agent_ID를 기준으로 JOIN하여 이에 해당하는 Agent_name을 Selling_Agent에서 select하도록 작성하였다. 결과는 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

-----
3
---- TYPE 3 ----

Find the name of the agent who has sold the most properties in the year 2022 by total won value.
Xavier
```

⑥ TYPE 3-1

```
if (sub_qnum == 1) {
    printf("---- TYPE 3-1 ----\n\n");
    printf("Find the top k agents in the year 2023 by total won value.\n");
    int k; cin >> k;
    query = "SELECT SA.Agent_name "
            "FROM Selling_Agent SA "
            "JOIN (SELECT Agent_ID "
            "FROM Sale "
            "WHERE YEAR(Sold_date) = 2023 "
            "GROUP BY Agent_ID "
            "ORDER BY SUM(Sale_price) DESC "
            "LIMIT " + to_string(k) + ") AS TopAgents "
            "ON SA.Agent_ID = TopAgents.Agent_ID;";
    state = mysql_query(connection, query.c_str());
    if (state != 0) {
        printf("Query Error: %s\n", mysql_error(connection));
        continue;
    }
}
```


다음은 TYPE 3-1로 2023년에 판매한 매물의 가격의 총합이 가장 높은 k명의 판매 중개인의 이름을 찾아야 하는 쿼리이다. 앞에서 TYPE 3와 연도와 찾아야 할 판매 중개인의 인원 수만 차이가 있으므로 동일하게 sql을 작성해주고 연도만 2023으로 바꾸고 찾아야 하는 인원이 k명이므로 LIMIT k를 사용하여 k명을 찾도록 작성하였다. k값의 경우 cin을 이용해 입력 받고 입력받은 k는 to_string함수를 통해 문자열로 바꿔 작성하였다. 결과는 다음과 같다.

```

----- SUBTYPES in TYPE3 -----

    1. TYPE 3-1
    2. TYPE 3-2
    0. TO MAIN MENU

1
---- TYPE 3-1 ----

Find the top k agents in the year 2023 by total won value.
3
Wendy
Steve
Mike

```

⑦ TYPE 3-2

```

else if (sub_gnum == 2) {
    printf("---- TYPE 3-2 ----\n\n");
    printf("Find the bottom 10%% agents in the year 2021 by total won value.\n");

    // Step 1: Calculate the total number of agents
    query = "SELECT CEIL(COUNT(DISTINCT Agent_ID) * 0.1) "
            "FROM Sale "
            "WHERE YEAR(Sold_date) = 2021;";
    state = mysql_query(connection, query.c_str());
    if (state != 0) {
        printf("Query Error: %s\n", mysql_error(connection));
        return EXIT_FAILURE;
    }

    // Step 2: Fetch bottom 10% agents by total won value in 2021
    query = "SELECT SA.Agent_name "
            "FROM Selling_Agent SA "
            "JOIN (SELECT Agent_ID "
            "        FROM Sale "
            "        WHERE YEAR(Sold_date) = 2021 "
            "        GROUP BY Agent_ID "
            "        ORDER BY SUM(Sale_price) ASC "
            "        LIMIT " + to_string(bottom_limit) + ") AS BottomAgents "
            "ON SA.Agent_ID = BottomAgents.Agent_ID;";
    state = mysql_query(connection, query.c_str());
    if (state != 0) {
        printf("Query Error: %s\n", mysql_error(connection));
        return EXIT_FAILURE;
    }
}

```

TYPE 3-2는 2021년에 판매한 매물의 가격 총합의 하위 10%에 해당하는 agent의 이름을 찾아야 한다. 먼저 setp1에서는 Sale relation에서 Sold_date의 연도가 2021인 agent를 찾고 이러한

agent의 수를 COUNT를 이용해 계산한 뒤 0.1을 곱한 값을 CEIL을 이용해 올림하여 최하위 10%의 판매 중개인의 수를 계산한다. 다음으로 step2에서는 2021년에 판매한 매물의 가격의 총합을 오름차순으로 정렬하여 LIMIT 값을 앞에서 CEIL로 구한 값으로 하여 하위 10%의 판매 중개인의 이름인 agent_name을 select하도록 작성하였다. 결과는 다음과 같다.

```

----- SUBTYPES in TYPE3 -----

      1. TYPE 3-1
      2. TYPE 3-2
      0. TO MAIN MENU
2
---- TYPE 3-2 ----

Find the bottom 10% agents in the year 2021 by total won value.
Uma

```

⑧ TYPE 4

```

else if (q_num == 4) {
    printf("---- TYPE 4 ----\n\n");
    printf("Compute the average selling price of properties sold in 2022, and the average time the property was on the market.\n");
    query = "SELECT AVG(Sale_price), AVG(DATEDIFF(Sold_date, For_sale_date)) FROM Sale WHERE YEAR(Sold_date) = 2022;";
    state = mysql_query(connection, query.c_str());
    if (state != 0) {
        printf("Query Error: %s\n", mysql_error(connection));
        continue;
    }
}

```

다음으로 TYPE 4는 2022에 팔린 부동산의 판매 가격의 평균과 판매 가능으로 등록되어 있던 기간의 평균을 구해야 한다. Sale relation에서 Sold_date의 연도가 2022인 데이터들의 sale_price의 평균을 AVG를 통해 구하고 DATEDIFF를 통해 Sold_date와 For_sale_date의 날짜 차이를 계산하여 그에 대한 평균을 AVG를 통해 구하도록 작성하였다. 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
-----
4
---- TYPE 4 ----

Compute the average selling price of properties sold in 2022, and the average time the property was on the market.
Average Price: 1303750000.0000, Average Time on Market: 25.0000 days

```

⑨ TYPE 4-1

```

if (sub_qnum == 1) {
    printf("---- TYPE 4-1 ----\n\n");
    printf("Compute the maximum selling price of properties sold in 2023 for each agent.\n");
    query = "SELECT Agent_name, MAX(Sale_price) FROM Selling_Agent ";
    query += "JOIN Sale ON Selling_Agent.Agent_ID = Sale.Agent_ID WHERE YEAR(Sold_date) = 2023 GROUP BY Agent_name; ";
    state = mysql_query(connection, query.c_str());
    if (state != 0) {
        printf("Query Error: %s\n", mysql_error(connection));
        continue;
    }
}

```

TYPE 4-1은 각 판매 중개인의 2023년의 가장 높은 부동산 판매가를 구해야 한다. 먼저 Sale relation에서 Sold_date가 2023년인 데이터와 Selling_agent relation를 ID를 기준으로 join하여 name으로 그룹화한 뒤 각 agent_name과 MAX를 이용해 구한 sale_price의 최대값을 select하도록 작성하였다. 이에 대한 결과는 다음과 같다.

```
----- SUBTYPES in TYPE4 -----
```

- 1. TYPE 4-1
- 2. TYPE 4-2
- 0. TO MAIN MENU

1

```
---- TYPE 4-1 ----
```

Compute the maximum selling price of properties sold in 2023 for each agent.

```
Agent: Wendy, Max Price: 1250000000
Agent: Quinn, Max Price: 275000000
Agent: Steve, Max Price: 1150000000
Agent: Mike, Max Price: 290000000
```

⑩ TYPE 4-2

```
else if (sub_qnum == 2) {
    printf("---- TYPE 4-2 ----\n\n");
    printf("Compute the longest time the property was on the market for each agent in 2022.\n");
    query = "SELECT Agent_name, MAX(DATEDIFF(Sold_date, For_sale_date)) FROM Selling_Agent ";
    query += "JOIN Sale ON Selling_Agent.Agent_ID = Sale.Agent_ID WHERE YEAR(Sold_date) = 2022 GROUP BY Agent_name ";
    state = mysql_query(connection, query.c_str());
    if (state != 0) {
        printf("Query Error: %s\n", mysql_error(connection));
        continue;
    }
}
```

TYPE 4-2는 2022년에 판매된 매물의 판매 등록 기간이 가장 긴 시간을 판매 중개인 별로 구해야 한다. 앞에서 TYPE 4-1에서 쓴 sql과 비슷하게 작성하고 연도를 2022로 바꾸고 MAX 값을 DATEDIFF를 이용해 구한 판매 등록 기간으로 설정하여 작성하였다. 결과는 다음과 같다.

```
----- SUBTYPES in TYPE4 -----
```

- 1. TYPE 4-1
- 2. TYPE 4-2
- 0. TO MAIN MENU

2

```
---- TYPE 4-2 ----
```

Compute the longest time the property was on the market for each agent in 2022.

```
Agent: Xavier, Longest Time: 21 days
Agent: Paul, Longest Time: 30 days
Agent: Rachel, Longest Time: 18 days
Agent: Uma, Longest Time: 31 days
```

⑪ TYPE 5

```
else if (q_num == 5) {
    printf("---- TYPE 5 ----\n\n");
    printf("Show photos of the most expensive studio, one-bedroom, multi-bedroom apartment(s), and detached house(s).\n");

    const char* query_studio =
        "SELECT 'Studio' AS Property_type, p.Interior_photo, p.Exterior_photo, p.Floor_plan "
        "FROM Property p "
        "WHERE p.Property_type = 'studio' "
        "ORDER BY p.Property_price DESC LIMIT 1;";

    const char* query_one_bedroom =
        "SELECT 'One-bedroom' AS Property_type, p.Interior_photo, p.Exterior_photo, p.Floor_plan "
        "FROM Property p "
        "WHERE p.Property_type = 'one-bedroom apartment' "
        "ORDER BY p.Property_price DESC LIMIT 1;";

    const char* query_multi_bedroom =
        "SELECT 'Multi-bedroom' AS Property_type, p.Interior_photo, p.Exterior_photo, p.Floor_plan "
        "FROM Property p "
        "WHERE p.Property_type = 'multi-bedroom apartment' "
        "ORDER BY p.Property_price DESC LIMIT 1;";

    const char* query_detached =
        "SELECT 'Detached House' AS Property_type, p.Interior_photo, p.Exterior_photo, p.Floor_plan "
        "FROM Property p "
        "WHERE p.Property_type = 'detached houses' "
        "ORDER BY p.Property_price DESC LIMIT 1;";
}
```

TYPE 5는 부동산의 유형인 studio, one-bedroom apartment, multi-bedroom apartment, 그리고

detached houses의 4가지 별로 가장 비싼 매물의 사진을 보여주는 것이다. 각 유형 별로 sql문을 따로 작성하였다. Property relation에서 각 유형별로 property_price를 기준으로 내림차순으로 정렬하여 LIMIT 1으로 가장 큰 값을 가진 property의 이름과 그에 해당하는 interior photo, exterior photo, floor plan을 select하도록 작성하였다. 이에 대한 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
-----
5
---- TYPE 5 ----

Show photos of the most expensive studio, one-bedroom, multi-bedroom apartment(s), and detached house(s).
Most expensive studio:
Type: studio, Interior Photo: interior photo 9, Exterior Photo: (null), Floor Plan: (null)

Most expensive one-bedroom apartment:
Type: one-bedroom apartment, Interior Photo: interior photo 10, Exterior Photo: (null), Floor Plan: (null)

Most expensive multi-bedroom apartment:
Type: multi-bedroom apartment, Interior Photo: (null), Exterior Photo: exterior photo 13, Floor Plan: floorplan 13

Most expensive detached houses:
Type: detached houses, Interior Photo: (null), Exterior Photo: exterior photo 12, Floor Plan: floorplan 12

```

⑫ TYPE 6

```

else if (q_num == 6) {
    printf("---- TYPE 6 ----\n\n");
    printf("Record the sale of a property that had been listed as available.\n");

    query = "SELECT Property_name, Property_address FROM Property;";

    query = "SELECT Buyer_ID, Buyer_name FROM Buyer;";

    query = "SELECT Property_price, Seller_ID, For_sale_date "
            "FROM Property WHERE Property_name = '" + property_name +
            "' AND Property address = '" + property_address + "';";

    query = "SELECT Agent ID FROM Seller WHERE Seller ID = " + to_string(seller_id) + ";";

    query = "SELECT Buyer_agent_ID FROM Buyer WHERE Buyer_ID = " + to_string(buyer_id) + ";";

    query = "SELECT DATE(NOW());";

    query = "INSERT INTO Sale (Property_name, Property_address, Sold_date, Seller_ID, Agent_ID, Buyer_ID, Buyer_agent_ID, For_sale_date, Sale_price) VALUES ('" +
            property_name + "', '" + property_address + "', '" + sold_date + "', " +
            to_string(seller_id) + ", " + to_string(agent_id) + ", " + to_string(buyer_id) + ", " +
            to_string(buyer_agent_id) + ", '" + for_sale_date + "', " + to_string(property_price) + ");";
    state = mysql_query(connection, query.c_str());

    query = "DELETE FROM Property WHERE Property_name = '" + property_name +
            "' AND Property address = '" + property_address + "';";
    state = mysql_query(connection, query.c_str());
}

```

TYPE 6는 판매 가능한 매물을 판매 기록에 기록하는 쿼리이다. 따라서 먼저 select를 통해 Property relation에서 모든 판매 가능한 부동산의 name과 address를 보여준 뒤 팔 부동산을 입력 받고 또한 Buyer relation의 명단에서 구매할 구매자의 ID를 입력 받아 해당 부동산의 정보와 부동산을 구매할 구매자의 정보를 가진다. 그 뒤 해당 부동산을 판매하는 판매자의 ID와 판매 중개인의 ID를 각각 Seller, Selling_Agent relation에서 얻어오고 거래에 참여하는 구매자의 구매 대리인의 ID 또한 Buyer's_Agent에서 얻어 온 뒤 DATE(NOW())라는 쿼리를 사용해 현재 날짜를 얻어 INSERT를 통해 Sale relation에 정보를 추가한다. 그 뒤 판매 기록에 추가한 해당 부동산에

대한 정보는 Property에서 DELETE를 통해 삭제해준다. 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

```
-----
```

6

```
---- TYPE 6 ----
```

Record the sale of a property that had been listed as available.

Available Properties:

Property Name: YeongdeungpoDetached, Address: 130 Yeongdeungpo Drive

Property Name: SeodaemunDetached, Address: 126 Seodaemun Lane

Property Name: DobongOneBed, Address: 132 Dobong Avenue

Property Name: DongdaemunStudio, Address: 123 Dongdaemun Street

Property Name: GangnamLuxury, Address: 140 Gangnam Boulevard

Property Name: MapoMultiBed, Address: 125 Mapo Road

Property Name: NowonStudio, Address: 131 Nowon Street

Property Name: KueumcheonMultiBed, Address: 129 Kueumcheon Road

Property Name: EunpyeongStudio, Address: 127 Eunpyeong Boulevard

Property Name: JongroDetached, Address: 134 Jongro Boulevard

Property Name: KuroOneBed, Address: 128 Kuro Street

Property Name: YongsanMultiBed, Address: 133 Yongsan Lane

Property Name: JoonglangOneBed, Address: 124 Joonglang Avenue

Enter the Property Name: YeongdeungpoDetached

Enter the Property Address: 130 Yeongdeungpo Drive

Buyers:

Buyer ID: 1023, Buyer Name: Leo

Buyer ID: 1029, Buyer Name: Penny

Buyer ID: 1723, Buyer Name: Adam

Buyer ID: 1947, Buyer Name: Mona

Buyer ID: 2839, Buyer Name: Quincy

Buyer ID: 2847, Buyer Name: Bella

Buyer ID: 2938, Buyer Name: Kara

Buyer ID: 3847, Buyer Name: Nina

Buyer ID: 3948, Buyer Name: Jack

Buyer ID: 3950, Buyer Name: Cathy

Buyer ID: 4839, Buyer Name: Dylan

Buyer ID: 5748, Buyer Name: Emma

Buyer ID: 5849, Buyer Name: Oscar

Buyer ID: 6382, Buyer Name: Frank

Buyer ID: 7485, Buyer Name: Gina

Buyer ID: 8293, Buyer Name: Henry

Buyer ID: 9284, Buyer Name: Ivy

Enter the Buyer ID: 1023

Property sale recorded successfully.

⑬ TYPE 7

```
else if (q_num == 7) {
    printf("----- TYPE 7 -----\\n\\n");
    printf("Add a new agent to the database.\\n");
    string agent_name, agent_phone;
    int ID;
    bool uniqueID = false;

    printf("Enter Agent Name: ");
    cin >> agent_name;
    printf("Enter Agent Phone: ");
    cin >> agent_phone;

    // Loop until a unique ID is found
    while (!uniqueID) {
        ID = rand() % 9999 + 1;
        query = "SELECT COUNT(*) FROM Selling_Agent WHERE Agent_ID = " + to_string(ID) + ";";
        state = mysql_query(connection, query.c_str());
        sql_result = mysql_store_result(connection);
        sql_row = mysql_fetch_row(sql_result);
        int count = atoi(sql_row[0]);
        mysql_free_result(sql_result);

        if (count == 0) {
            uniqueID = true;
        }
    }

    query = "INSERT INTO Selling_Agent (Agent_ID, Agent_name, Agent_phone_number) VALUES ("
        + to_string(ID) + ", '" + agent_name + "', '" + agent_phone + "')";
    state = mysql_query(connection, query.c_str());
}
```

TYPE 7은 새로운 판매 중개인은 추가하는 쿼리이다. 먼저 등록할 판매 중개인의 이름과 휴대폰 번호를 입력 받고 random()을 통해 1~9999까지의 값에서 무작위로 하나의 숫자를 ID로 설정한다. 이때 ID는 Selling_Agent의 primary key이므로 겹치는 값이 있으면 안 되기 때문에 해당 값이 이미 table에 존재하는지 확인하고 없으면 해당 값으로 결정하고 만약 있다면 다시 설정해준다. 마지막에는 INSERT를 통해 새로운 판매 중개인의 정보를 넣어준다. 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

-----
7
----- TYPE 7 -----

Add a new agent to the database.
Enter Agent Name: Alex
Enter Agent Phone: 010-2394-5860
Agent added successfully.
```