CSE4110–DatabaseSystem

Project 2

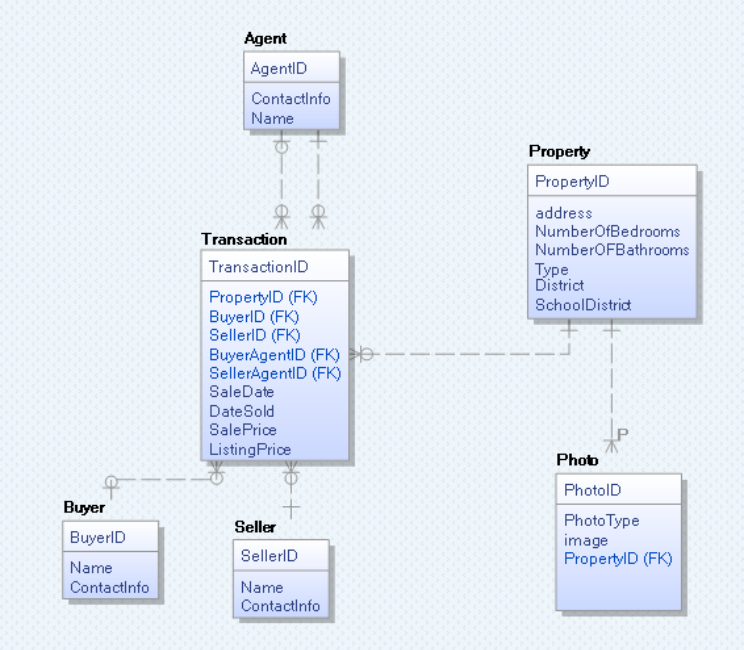
20191098 백승주

1. Project 개요

Project 1에서 부동산 관련 데이터를 이용해 logical schema를 설계했다. 이를 토대로 bcnf 분해, physical design, mysql과 visual studio를 연결해 crud.txt 파일을 읽고 쿼리를 실행하는 프로그램을 개발하자.

2. BCNF Decomposition

Bcnf 분해를 하기 전 먼저 logical schema에 대해 간단히 살펴보겠다. Project1에서 약간의 수정을 거친 logical schema다.



Attribute

- Agent (에이전트):
  - AgentID: 에이전트의 고유 식별자.
  - ContactInfo: 에이전트의 연락처 정보.
  - Name: 에이전트의 이름.

- Buyer (구매자):
    - BuyerID: 구매자의 고유 식별자.
    - Name: 구매자의 이름.
    - ContactInfo: 구매자의 연락처 정보.
- Seller (판매자):
    - SellerID: 판매자의 고유 식별자.
    - Name: 판매자의 이름.
    - ContactInfo: 판매자의 연락처 정보.
- Property (부동산):
    - PropertyID: 부동산의 고유 식별자.
    - address: 부동산의 주소.
    - NumberOfBedrooms: 침실 수.
    - NumberOfBathrooms: 욕실 수.
    - Type: 부동산 유형 (예: 아파트, 주택 등).
    - District: 지역.
    - SchoolDistrict: 학군.
- Transaction (거래):
    - TransactionID: 거래의 고유 식별자.
    - PropertyID (FK): 부동산의 외래 키, Property 테이블과 연결.
    - BuyerID (FK): 구매자의 외래 키, Buyer 테이블과 연결.
    - SellerID (FK): 판매자의 외래 키, Seller 테이블과 연결.
    - BuyerAgentID (FK): 구매자 에이전트의 외래 키, Agent 테이블과 연결.
    - SellerAgentID (FK): 판매자 에이전트의 외래 키, Agent 테이블과 연결.
    - SaleDate: 판매 날짜.
    - DateSold: 거래 완료 날짜. 이 속성이 null 이면 판매중이란 뜻.
    - SalePrice: 판매 가격.
    - ListingPrice: 초기 등록 가격.
- Photo (사진):
    - PhotoID: 사진의 고유 식별자.
    - PhotoType: 사진 유형 (예: 외관, 실내 등).
    - image: 사진 데이터.
    - PropertyID (FK): 부동산의 외래 키, Property 테이블과 연결.

Relationship

- Agent - Transaction:
    - 한 에이전트(Agent)는 여러 거래(Transaction)에서 구매자 에이전트(BuyerAgent) 또는 판매자 에이전트(SellerAgent)로서 참여할 수 있다. 한 명의 agent 는 0 개의 거래, 1 개의 거래, 혹은 여러 개의 거래가 가능(cardinality : Zero, one or more)
    - Transaction 테이블의 BuyerAgentID 는 nullable 이지만 SellerAgentID 는 not null 이다. 판매되지 않은 물품도 거래 항목에 있을 수 있기 때문
- Buyer - Transaction:
    - 한 구매자(Buyer)는 여러 거래(Transaction), 또는 아예 참여하지 않을 수 있다. cardinality : Zero, one or more
    - Transaction 테이블의 BuyerID 는 nullable 이다.
- Seller - Transaction:
    - 한 판매자(Seller)는 여러 거래(Transaction)에 참여할 수 있다. . cardinality : Zero, one or more
    - Transaction 테이블의 SellerID 는 not nullable 이다.
- Property - Transaction:
    - 한 부동산(Property)는 여러 거래(Transaction)에 포함될 수 있다. . cardinality : Zero, one or more
    - Transaction 테이블의 PropertyID 는 not nullable 이다.
- Property - Photo:
    - 한 부동산(Property)는 여러 사진(Photo)을 가질 수 있다. 또한 한 부동산은 아예 사진을 가지지 않을 수는 없다. cardinality : one or more
    - Photo 테이블의 PropertyID 는 not nullable 이다.

BCNF 여부 조사 및 분해

1) Agent

Agent entity에는 AgentID ->(ContactInfo,Name)을 제외하고는 다른 FD가 존재하지 않고 이때의 AgentID는 primary 키이기 때문에 BCNF를 만족한다. Name의 경우 동명이인이 있을 수 있어 name에 의해 contactinfo는 결정되지는 않는다.

2) Seller

SellerID-> name,contactinfo 이외의 FD는 존재하지 않고 sellerID 는 primarykey이기 때문에 bcnf를 만족한다. Name, contact info는 동일하지만 다른 이가 존재할 가능성이 있다.

3) Buyer – seller와 마찬가지

4) Photo

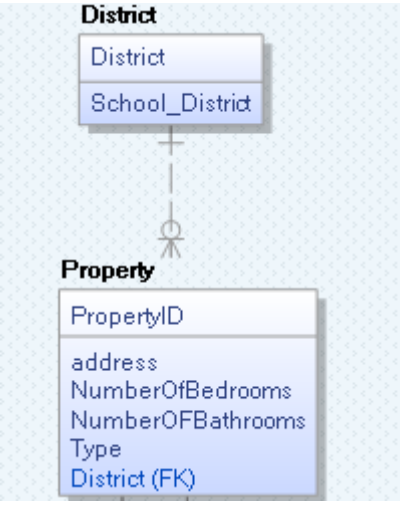
PhotoID->PhotoType,image, PropertyID이다. 그렇지만 하나의 propertyID에 대해 여러 종류의 사진이 존재할 수 있기 때문에 propertyID는 나머지 열을 결정하지 않으며 같은 Photoype에 대해서도 여러 이미지, 그리고 해당 포토 타입의 여러 부동산 사진이 존재할 수 있어 그 외의 fd는 존재하지 않는다. 단, 이 때 image 값은 해당 데이터를 만들어놓고 업자가 아직 사진을 안 찍었을수 있기 때문에 null 값을 허용하며 이에 따라 다른 속성들을 결정하지 못한다.

5) Transaction

같은propertyID, 즉 같은 부동산이라도 거래가 여러 번 일어났을 수도 있고 같은 구매자,판매자, agent들이 여러 번 거래를 했을 수도 있고 같은 날짜에 제각기 다른 거래가 일어났을 수 있다. 따라서 transactionID만이 나머지 속성들을 결정하며 그 외의 FD는 존재하지 않는다. 따라서 bcnf 만족

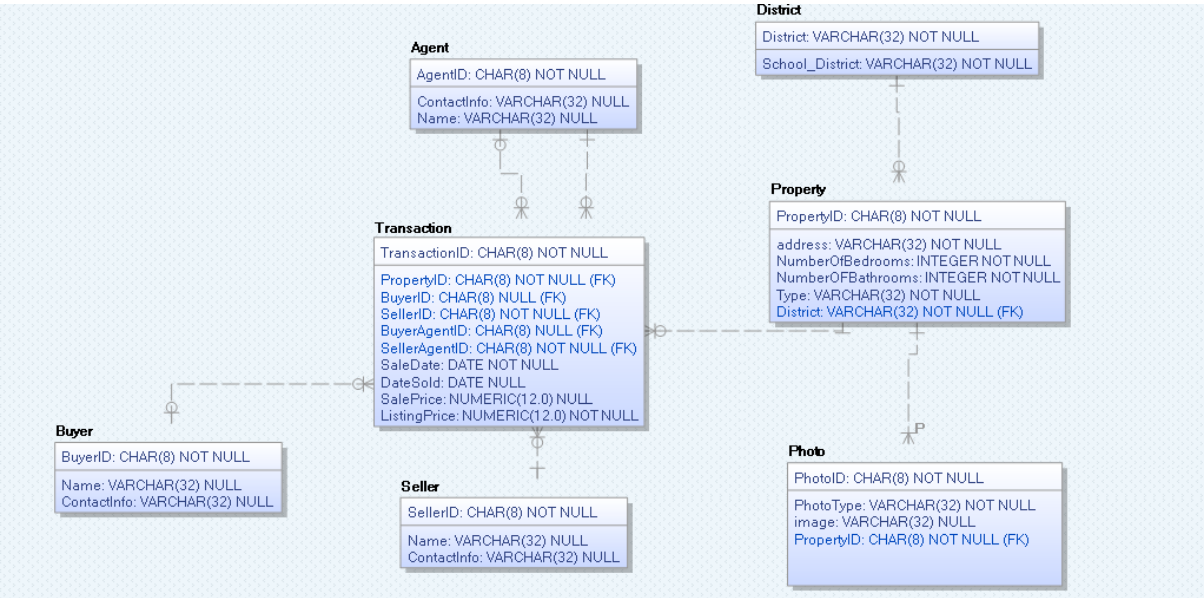
6) Property 테이블

PropertyID -> address, numberofbedrooms,numberofbathrooms,district,schooldistrict 이고 District - > SchoolDistrict 이다. 단, address -> NUmberofbedrooms나 address->District 는 성립하지 않다고 봤는데 그 이유는 같은 주소에 다른 건물이 세워져 방의 개수가 달라질 수도 있고 같은 도로명 주소를 가진 지역이 존재해 도로명 주소에 따른 지역 결정이 항상 옳은 것이 아닐수도 있기 때문이다.
찾아낸 FD 중에서 District는 propertyID와 다르게 primary 키가 아니기 때문에 bcnf를 만족하지 않기 때문에 분해를 진행한다.이에 따라 새롭게 만들어지는 entity는 District entity 이다. District entity는 district와 school district를 열로 가지며 district->schooldistrict가 성립하며 이 때 district entity에서 district가 primary key 이기 때문에 이렇게 분해될 경우 property와 새롭게 만들어진 district 모두 bcnf가 성립한다.



이 때 District는 not nullable이며 cardinality는 zero,one,or more이다. 하나의 Distrct 안에는 여러 부동산, 혹은 아예 부동산 없을 수 있다.

3. Physical schema



1) Agent

AgentID값은 ID를 입력하는 값이므로 자료형으로 고정된 형식인 char(8)을 사용한다. 앞으로의 모든 ID들은 char(8)을 사용하기로 한다. AgentID는 primary키이기에 당연히 not null이다. Contactinfo와 name의 경우 그 길이가 달라질 수 있으므로 VARCHAR(32)에 저장하며 이 둘은 꼭 필요한 정보들이 아니기에 NUll값이 될 수 있다.

2) District

Distrct를 입력받기 위한 VARCHAR(32)에 대해 이 자료형에는 구 이름을 영문으로 입력

하면 된다. Project의 조건에서 지역을 서울로 한정시키고 있기 때문에 지역 이름만으로 primary key가 되는 것이 가능하다. 학군의 경우 1~8 정수로 나타나는 자료형이기 때문에 Integer자료형을 사용한다. 서울로 한정되어 있기 때문에 null값이 될 수 없다.

3) Seller

ID는 char(8), 크기가 달라질 수 있는 name,contactinfo는 varchar(32)로 저장한다, agent entity와 마찬가지다. Null 값이 될수 있다.

4) Buyer

seller와 그 특성들 동일

5) Photo

Photo entity의 경우 문제에 제시된 조건 중 하나인
for studio or one-bedroom apartments, there must be at least one interior photo. For multi-bedroom apartments or detached houses, there should be at least one exterior photo and one floor plan each
에 의해서 모든 부동산들은 사진을 최소 하나씩은 가져야한다. 따라서 cardnality는 위 그림과 같다.

PhotoID: CHAR(8) NOT NULL - PhotoID는 고유 식별자이며, 고정된 형식의 문자형 자료이므로 CHAR(8)을 사용하고, primary key로 설정하여 NOT NULL입니다.

PhotoType: VARCHAR(32) NOT NULL - PhotoType은 사진의 유형을 나타내며, 다양한 길이의 문자열이 들어갈 수 있으므로 VARCHAR(32)를 사용한다. 모든 사진은 type이 분류돼야함

image: VARCHAR(32) NULL - image는 사진의 실제 데이터 경로를 저장하며, 길이가 가변적이므로 VARCHAR(32)를 사용하고, 아직 사진을 입력 안 찍었을 수 있으므로 null

PropertyID (FK): CHAR(8) NOT NULL - PropertyID는 부동산을 참조하는 외래 키로, CHAR(8)을 사용하고 not NULL이다.

6) Transaction

TransactionID: CHAR(8) NOT NULL - TransactionID는 고유 식별자이며, 고정된 형식의 문자형 자료이므로 CHAR(8)을 사용하고, primary key로 설정하여 NOT NULL입니다.

BuyerID (FK): CHAR(8) NOT NULL - BuyerID는 구매자를 참조하는 외래 키로, CHAR(8)을 사용하고 NULL이다. (판매가 안 됐을 수 있음)

PropertyID (FK): CHAR(8) NOT NULL - PropertyID는 부동산을 참조하는 외래 키로, CHAR(8)을 사용하고 NOT NULL.

SellerID (FK): CHAR(8) NOT NULL - SellerID는 판매자를 참조하는 외래 키로, CHAR(8)을 사용하고 NOT NULL.

BuyerAgentID (FK): CHAR(8) NULL - BuyerAgentID는 구매자 에이전트를 참조하는 외래 키로, CHAR(8)을 사용하고, 판매완료가 안 됐을 수 있으므로 NULL 값을 허용.

SellerAgentID (FK): CHAR(8) NULL- SellerAgentID는 판매자 에이전트를 참조하는 외래 키로, CHAR(8)을 사용하고, 꼭 필요한 정보이므로 NULL 값을 허용.

SaleDate: DATE NOT NULL - SaleDate는 판매 시작 날짜이므로 NOT NULL로 설정.

DateSold: DATE NULL - DateSold는 실제 거래 완료 날짜를 나타내며, 판매중일 수 있으므로 NULL 값을 허용합니다.

SalePrice: NUMBER(12, 0) NULL - SalePrice는 판매 가격을 나타내며,큰 수이기 때문에 Numberic 자료형을 사용.판매 중일 수 있어서 NULL 값을 허용합니다.

ListingPrice: NUMBER(12, 0) NOT NULL - ListingPrice는 초기 등록 가격을 나타내며, 반드시 필요한 정보이므로 NUMBER(12, 0) 자료형을 사용하고 NOT NULL로 설정.

7) Transaction

PropertyID: CHAR(8) NOT NULL - PropertyID는 고유 식별자이며, 고정된 형식의 문자형 자료이므로 CHAR(8)을 사용하고, primary key로 설정하여 NOT NULL.

address: VARCHAR(32) NULL - address는 부동산의 주소를 나타내며, 길이가 가변적이므로 VARCHAR(32)를 사용하고 NULL 값을 허용하지 않음.(주소는 부동산에서 필수)

NumberOfBedrooms: INTEGER NOT NULL NumberOfBedrooms는 침실 수를 나타내며, 후술할 쿼리들을 위해 반드시 필요한 정보이므로 INTEGER 자료형을 사용하고 NOT NULL로 설정.

NumberOfBathrooms: INTEGER NOT NULL - NumberOfBathrooms는 욕실 수를 나타내며, 반드시 필요한 정보이므로 INTEGER 자료형을 사용하고 NOT NULL로 설정합니다.

Type: VARCHAR(32) NOT NULL - Type은 부동산의 유형을 나타내며, 쿼리들을 위해 반드시 필요한 정보이므로 VARCHAR(32)를 사용하고 NOT NULL로 설정합니다.

District (FK): VARCHAR(32) NOT NULL District는 지역을 나타내며, 외래 키로 사용되고 VARCHAR(32)를 사용하며 NOT NULL로 설정다.

추가 제약조건

1) NumberOfBedrooms INTEGER NOT NULL CHECK (NumberOfBedrooms >= 0),

NumberOfBathrooms INTEGER NOT NULL CHECK (NumberOfBathrooms >= 0),

방들의 개수는 0보다 같거나 커야함

2) CHECK (DateSold >= SaleDate)

판매완료날짜가 더 나중이어야한다 판매등록날짜보다.

3) SalePrice NUMERIC(12, 0) NULL CHECK (SalePrice > 0 OR SalePrice IS NULL)
ListingPrice NUMERIC(12, 0) NOT NULL CHECK (ListingPrice > 0),

가격이므로 0보다는 커야한다.

결과적인 테이블들 생성 sql 문들은 다음과 같다.

CREATE TABLE District (District VARCHAR(32) NOT NULL, SchoolDistrict INTEGER NOT NULL, PRIMARY KEY (District));

CREATE TABLE Property (PropertyID CHAR(8) NOT NULL, address VARCHAR(32) NOT NULL, NumberOfBedrooms INTEGER NOT NULL CHECK (NumberOfBedrooms >= 0), NumberOfBathrooms INTEGER NOT NULL CHECK (NumberOfBathrooms >= 0), District VARCHAR(32) NOT NULL, Type VARCHAR(32) NOT NULL, PRIMARY KEY (PropertyID), FOREIGN KEY (District) REFERENCES District(District));

CREATE TABLE Photo (PhotoID CHAR(8) NOT NULL, image VARCHAR(32) NULL, PhotoType VARCHAR(32) NOT NULL, PropertyID CHAR(8) NOT NULL, PRIMARY KEY (PhotoID), FOREIGN KEY (PropertyID) REFERENCES Property(PropertyID));

CREATE TABLE Agent (AgentID CHAR(8) NOT NULL, Name VARCHAR(32) NULL, ContactInfo VARCHAR(32) NULL, PRIMARY KEY (AgentID));

CREATE TABLE Seller (SellerID CHAR(8) NOT NULL, Name VARCHAR(32) NOT NULL, ContactInfo VARCHAR(32) NOT NULL, PRIMARY KEY (SellerID));

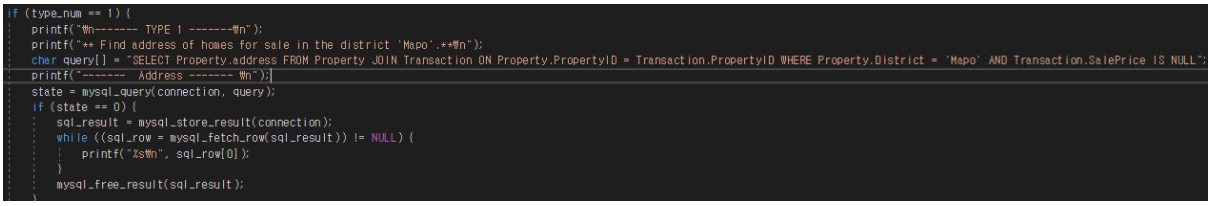
CREATE TABLE Buyer (BuyerID CHAR(8) NOT NULL, Name VARCHAR(32) NULL, ContactInfo VARCHAR(32) NULL, PRIMARY KEY (BuyerID));

CREATE TABLE Transaction (TransactionID CHAR(8) NOT NULL, PropertyID CHAR(8) NOT NULL, BuyerAgentID CHAR(8) NULL, SellerAgentID CHAR(8) NOT NULL, DateSold DATE NULL, SaleDate DATE NOT NULL, SalePrice NUMERIC(12, 0) NULL CHECK (SalePrice > 0 OR SalePrice IS NULL), ListingPrice NUMERIC(12, 0) NOT NULL CHECK (ListingPrice > 0), SellerID CHAR(8) NOT NULL, BuyerID CHAR(8) NULL, PRIMARY KEY (TransactionID), FOREIGN KEY (BuyerAgentID) REFERENCES Agent(AgentID), FOREIGN KEY (SellerAgentID) REFERENCES Agent(AgentID), FOREIGN KEY (PropertyID) REFERENCES Property(PropertyID), FOREIGN KEY (SellerID) REFERENCES Seller(SellerID), FOREIGN KEY (BuyerID) REFERENCES Buyer(BuyerID), CHECK (DateSold >= SaleDate));

4. Code implememtation

1) 쿼리 1-

(TYPE1) Find address of homes for sale in the district “Mapo”. (1)



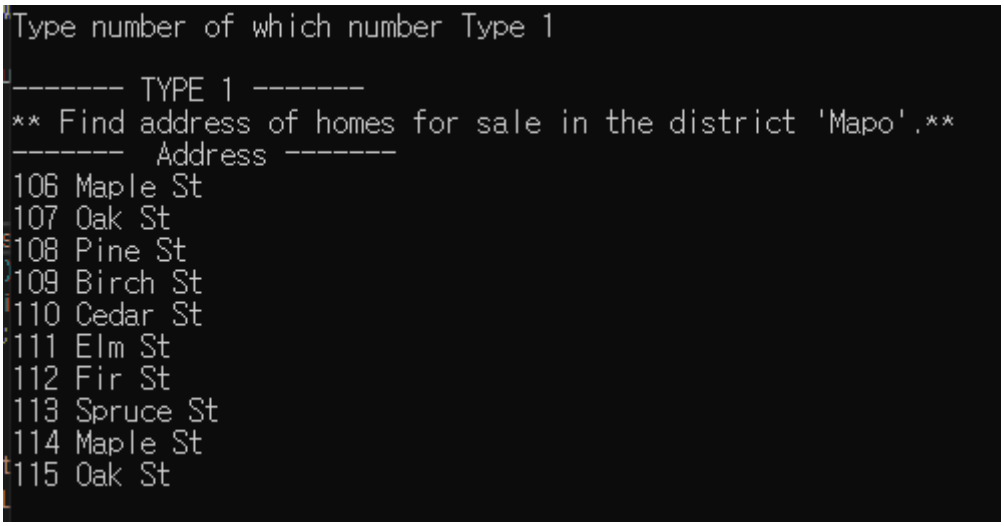
```
if (type_num == 1) {
    printf("\n------- TYPE 1 -------\n");
    printf("++ Find address of homes for sale in the district 'Mapo'.++\n");
    char query[] = "SELECT Property.address FROM Property JOIN Transaction ON Property.PropertyID = Transaction.PropertyID WHERE Property.District = 'Mapo' AND Transaction.SalePrice IS NULL";
    printf("-------  Address ------- \n");
    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\n", sql_row[0]);
        }
        mysql_free_result(sql_result);
    }
```

먼저 1번을 입력했을 때의 쿼리는 다음과 같다.

```
char query[] = "SELECT Property.address FROM Property JOIN Transaction ON
Property.PropertyID = Transaction.PropertyID WHERE Property.District = 'Mapo' AND
```

```
Transaction.SalePrice IS NULL";
```

제시문의 district 만족시키기 위해 district Mapo 조건을 만족하고 for sale을 만족시키기 위해 SalePrice 값이 null값인 값들을 검색했다. Transaction entity를 설계할 때 ListingPrice는 시장가, SalePRice는 최종판매가로 설정했기 때문에 SalePrice 값이 null이란 뜻은 아직 판매중이란 뜻이다. 이에 따른 결과는 다음과 같다.

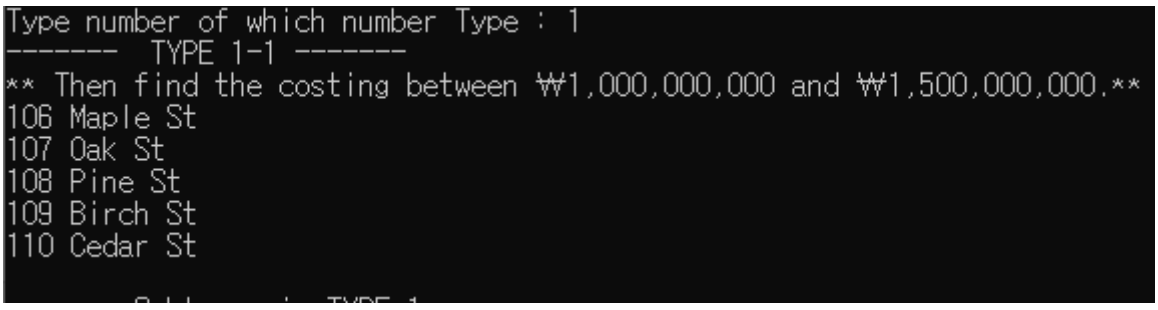


사용자가 1을 입력해 Type 1을 선택했고 그에 따라 Type1의 제시된 쿼리가 실행되어 판매중인 부동산 중 Mapo District에 속한 부동ㄷ산들의 Address 정보들을 찾았다.
위 과정들을 시행하고 난 후 다음의 제시된 쿼리 1-1 내용을 실행해야 한다.

- (TYPE 1-1) Then find the costing between ₩ 1,000,000,000 and ₩1,500,000,000. (2)
위 제시된 쿼리문에 따라 앞서 1에서 탐색한 마포구에 속한 판매 중인 부동산 중에서 해당 가격 범위 내의 부동산들을 필터링해야 한다. 이를 수행하기 위한 쿼리는 다음과 같다.

```
char subquery[] = "SELECT Property.address FROM Property JOIN Transaction ON
Property.PropertyID = Transaction.PropertyID WHERE Property.District = 'Mapo' AND
Transaction.SalePrice IS NULL AND Transaction.ListingPrice BETWEEN 1000000000 AND
1500000000";
```

위 쿼리를 통해 property의 District가 마포, 그리고 SalePrice가 NULL(판매중을 의미), 그리고 ListingPrice가 해당 범위 내의 있는 부동산들의 address 정보들을 찾는다. 그 실행 결과는 다음과 같다.



이전 1의 결과에서 가격 범위 내의 데이터만 필터링 된 것을 확인할 수 있다.

2) 쿼리 2

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

```
else if (type_num == 2) {
    printf("₩n------- TYPE 2 -------₩n");
    printf("** Find the address of homes for sale in the 8t
    char query[] = "SELECT Property.address FROM Property J
    printf("₩n-------  Address of 8th school ------- ₩n");
    state = mysql_query(connection, query);
    if (state == 0) {
        sql_result = mysql_store_result(connection);
        while ((sql_row = mysql_fetch_row(sql_result)) != N
            printf("%s₩n", sql_row[0]);
        }
        mysql_free_result(sql_result);
    }
    int subtype_num;
    bool subflag = false;
```

2번을 입력했을 때 쿼리는 다음과 같다.

```
char query[] = "SELECT Property.address FROM Property JOIN Transaction ON
Property.PropertyID = Transaction.PropertyID WHERE Property.District IN (SELECT
District FROM District WHERE SchoolDistrict = 8) AND Transaction.SalePrice IS NULL";
```

District entity의 School district 속성을 이용해 해당 속성이 8, 다시 말해 8학군인 부동산들을 property entity에서 찾고 그 중 for sale인 것을 찾기 위해 SalePrice = Null인 것들 것 찾는다. 그 실행 결과는 다음과 같다.

```
------- TYPE 2 -------
** Find the address of homes for sale in the 8th school district.**

-------  Address of 8th school -------
200 Pine St
201 Oak St
202 Maple St
203 Birch St
300 Pine St
301 Oak St
302 Maple St
303 Birch St

------- Subtypes in TYPE 2 -------
        1. TYPE 2-1.
Type number of which number Type :
```

위 결과에 따라 8학군인 부동산들의 상세주소(address)가 출력되었다. 이후 사용자는 1ㅇ을 입력함으로써 2-1 subtype을 실행할 수 있다.

(TYPE 2-1) Then find properties with 4 or more bedrooms and 2 bathrooms

위 제시문을 실행하기 위한 쿼리는 다음과 같다.

```
char subquery[] = "SELECT Property.address FROM Property JOIN Transaction ON
Property.PropertyID = Transaction.PropertyID WHERE Property.District IN (SELECT District
FROM District WHERE SchoolDistrict = 8) AND Transaction.SalePrice IS NULL AND
Property.NumberOfBedrooms >= 4 AND Property.NumberOfBathrooms = 2";
```

앞선 2번 쿼리의 결과로 8학군인 부동산들 중 NumberOFBEdrooms 속성이 4개 이상, 그리고 NUmberOFBathrooms 속성이 2개 이상인 부동산들을 찾는다. 그에 따른 결과는 다음과 같다.

```
-------  TYPE 2-1 -------
** Then find properties with 4 or more bedrooms and 2 bathrooms.**
200 Pine St
202 Maple St
301 Oak St
302 Maple St

-------- Subtypes in TYPE 2 --------
```

2의 결과에서 추가 조건으로 필터링 되었다.

3) 쿼리 3

3번 쿼리들을 시작하기 전에 모든 agent들이 각 년도별로 얼마나 판매했는지를 알려주는 쿼리 결과를 출력하는 것을 주석으로 달아두었다. 이를 통해 앞으로 있을 3번 쿼리들이 제대로 시행됐는지 확인할 수 있다.

```
else if (type_num == 3) {
    /* 3번 쿼리들이 맞는지 확인용
    // Display total sales per agent per year
    printf("₩n------- Total Sales per Agent per Year ------- ₩n");
    char total_sales_query[] = "SELECT YEAR(SaleDate) AS Year, Agent.Name, SUM
    state = mysql_query(connection, total_sales_query);
    if (state == 0) {
        sql_result = mysql_store_result(connection);
        printf("Year₩tAgent Name₩tTotal Sales₩n");
        while ((sql_row = mysql_fetch_row(sql_result)) != NULL) {
            printf("%s₩t%s₩t%s₩n", sql_row[0], sql_row[1], sql_row[2]);
        }
        mysql_free_result(sql_result);
    }
    */
```

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

```
char query[] = "SELECT Agent.Name FROM Agent JOIN (SELECT SellerAgentID,
SUM(SalePrice) AS TotalSale FROM Transaction WHERE YEAR(SaleDate) = 2022 GROUP BY
SellerAgentID ORDER BY TotalSale DESC LIMIT 1) AS TopAgent ON Agent.AgentID =
TopAgent.SellerAgentID";
```

이 SQL 쿼리는 2022년 동안 가장 많은 총 판매 금액을 기록한 에이전트의 이름을 찾는다. 먼저 Transaction 테이블에서 2022년 동안의 총 판매 금액을 계산하고, 이를 기준

으로 가장 높은 금액을 기록한 에이전트를 Limit 1을 통해 찾는다. 그런 다음, Agent 테이블과 조인하여 해당 에이전트의 이름을 반환합니다.

위 쿼리의 결과는 다음과 같다.

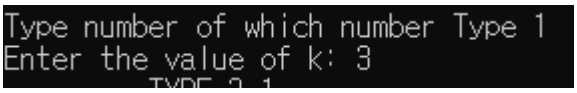
그 후 서브쿼리로 3-1, 3-2이 존재한다. 3번의 서브쿼리들의 경우 앞선 서브쿼리들과는 다르게 3번의 결과에 대해 중첩해서 조건을 적용시키는 것이 아닌 새로운 쿼리이다.

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

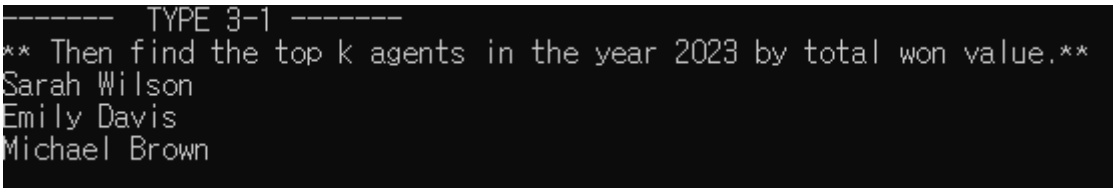
먼저 3-1에 대한 서브쿼리문은 다음과 같다.

```
sprintf(subquery, "SELECT Agent.Name FROM Agent JOIN (SELECT SellerAgentID,
SUM(SalePrice) AS TotalSale FROM Transaction WHERE YEAR(SaleDate) = 2023 GROUP BY
SellerAgentID ORDER BY TotalSale DESC LIMIT %d) AS TopAgents ON Agent.AgentID =
TopAgents.SellerAgentID", k);
```

이 SQL 쿼리는 2023년 동안 총 판매 금액이 가장 많은 상위 k명의 에이전트를 찾는다. Transaction 테이블에서 2023년 동안의 판매 금액을 계산하고 이를 기준으로 상위 k명의 에이전트를 추출한 후, Agent 테이블과 조인하여 해당 에이전트들의 이름을 반환합니다. 여기서 k는 사용자가 지정한 값이다.



위와 같이 k값을 입력받고 그 결과는 다음과 같다.



위와 같이 상위 3명 agent를 출력했다. 이 때 현재 agent의 수를 넘어서 k값을 입력한다면 현재 agent들만을 출력한다. 현재 crud 파일의 agent수는 10명이다.

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

3-2를 수행하기 위해서는 먼저 아래 명령문들을 통해 하위 10%가 몇 명인지를 구한다.

```
printf("** Find the bottom 10%% agents in the year 2021 by total won value.**\n");

// First, calculate the number of agents to include in the bottom 10%
char count_query[] = "SELECT COUNT(DISTINCT SellerAgentID) FROM Transaction WHERE YEAR(SaleDate) = 2021";
state = mysql_query(connection, count_query);
int agent_count = 0;
if (state == 0) {
    sql_result = mysql_store_result(connection);
    if ((sql_row = mysql_fetch_row(sql_result)) != NULL) {
        agent_count = atoi(sql_row[0]);
    }
    mysql_free_result(sql_result);
}

int bottom_10_percent_count = (int)(agent_count * 0.1);
```

이 때, 하위 10%의 count를 구할 때 int bottom_10_percent_count = (int)(agent_count * 0.1);, (int)를 통해 소수점 이하를 제거해 정수 부분만 남긴다. 따라서 12명이라면 10프로 값은 1.2이므로 소수 부분을 제거해 1명만이 출력될 것이다. 위에서 얻은 bottom_10_percent_count 값을 바탕으로 쿼리를 구성하면 다음과 같다.

```
sprintf(subquery, "SELECT Agent.Name FROM Agent JOIN (SELECT SellerAgentID,
SUM(SalePrice) AS TotalSale FROM Transaction WHERE YEAR(SaleDate) = 2021 GROUP BY
SellerAgentID ORDER BY TotalSale ASC LIMIT %d) AS BottomAgents ON Agent.AgentID =
BottomAgents.SellerAgentID ORDER BY TotalSale ASC", bottom_10_percent_count);
```

이 SQL 쿼리는 2021년 동안 총 판매 금액이 가장 적은 하위 bottom_10_percent_count 명의 에이전트를 찾는다. Transaction 테이블에서 2021년 동안의 판매 금액을 계산하고 이를 기준으로 하위 bottom_10_percent_count명의 에이전트를 추출한 후, Agent 테이블과 조인하여 해당 에이전트들의 이름을 반환한다. 그 결과는 다음과 같다.

```
--------  TYPE 3-2 --------
** Find the bottom 10% agents in the year 2021 by total won value.**
Emily Lee
```

현재 agent들이 10명이기 때문에 1명만이 출력되었다.

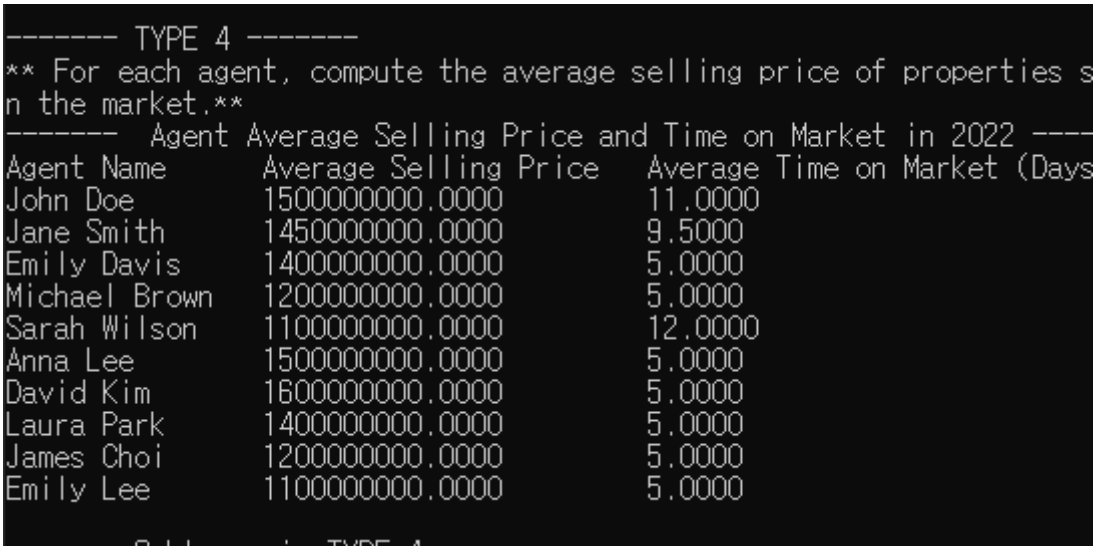
4) 쿼리 4

(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

```
char query[] = "SELECT Agent.Name, AVG(Transaction.SalePrice) AS AvgSellingPrice,
AVG(DATEDIFF(Transaction.DateSold, Transaction.SaleDate)) AS AvgTimeOnMarket FROM
Agent JOIN Transaction ON Agent.AgentID = Transaction.SellerAgentID WHERE
YEAR(Transaction.SaleDate) = 2022 GROUP BY Agent.Name";
```

이 SQL 쿼리는 2022년 동안 각 에이전트의 평균 판매 가격과 평균 시장 체류 시간을 계산한다. Transaction 테이블에서 2022년 동안의 거래 데이터를 기반으로, 에이전트별로 판매된 부동산의 평균 판매 가격(AvgSellingPrice)과 평균 시장 체류 시간

(AvgTimeOnMarket, 판매일(SaleDate) - 매물로 나온 날짜(SaleDate))을 계산한다. 그런 다음, Agent 테이블과 조인하여 각 에이전트의 이름과 함께 이 값을 반환한다.



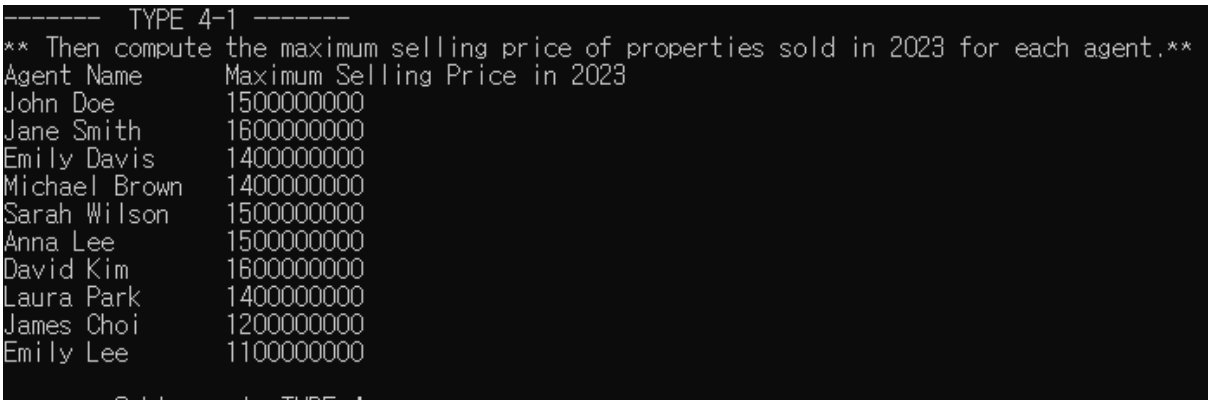
```
------- TYPE 4 -------
** For each agent, compute the average selling price of properties s
n the market.**
-------  Agent Average Selling Price and Time on Market in 2022 ----
Agent Name      Average Selling Price   Average Time on Market (Days
John Doe        1500000000.0000         11.0000
Jane Smith      1450000000.0000         9.5000
Emily Davis     1400000000.0000         5.0000
Michael Brown   1200000000.0000         5.0000
Sarah Wilson    1100000000.0000         12.0000
Anna Lee        1500000000.0000         5.0000
David Kim       1600000000.0000         5.0000
Laura Park      1400000000.0000         5.0000
James Choi      1200000000.0000         5.0000
Emily Lee       1100000000.0000         5.0000
```

그리고 type4에는 서브쿼리 4-1, 과 4-2가 존재한다 4번의 3번의 서브쿼리들처럼 4번의 결과에 대해 중첩해서 조건을 적용시키는 것이 아닌 새로운 쿼리이다.

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

```
char subquery[] = "SELECT Agent.Name, MAX(Transaction.SalePrice) AS MaxSellingPrice
FROM Agent JOIN Transaction ON Agent.AgentID = Transaction.SellerAgentID WHERE
YEAR(Transaction.SaleDate) = 2023 GROUP BY Agent.Name";
```

이 SQL 쿼리는 2023년 동안 각 에이전트가 판매한 부동산 중 가장 높은 판매 가격을 계산한다. Transaction 테이블의 SaleDate가 2023년인 거래를 대상으로, 에이전트별로 최대 판매 가격을 추출한 후, Agent 테이블과 조인하여 해당 에이전트의 이름과 함께 반환한다.



```
-------  TYPE 4-1 -------
** Then compute the maximum selling price of properties sold in 2023 for each agent.**
Agent Name      Maximum Selling Price in 2023
John Doe        1500000000
Jane Smith      1600000000
Emily Davis     1400000000
Michael Brown   1400000000
Sarah Wilson    1500000000
Anna Lee        1500000000
David Kim       1600000000
Laura Park      1400000000
James Choi      1200000000
Emily Lee       1100000000
```

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

```
char subquery[] = "SELECT Agent.Name, MAX(DATEDIFF(Transaction.DateSold,
Transaction.SaleDate)) AS LongestTimeOnMarket FROM Agent JOIN Transaction ON
Agent.AgentID = Transaction.SellerAgentID GROUP BY Agent.Name";
```

이 쿼리는 각 에이전트가 판매한 부동산 중 가장 오래 시장에 머문 기간을 계산한다. Transaction 테이블의 SaleDate와 DateSold를 기준으로 부동산이 시장에 머문 일수를 계산하고, 에이전트별로 가장 오래 걸린 시간을 추출한다. 그런 다음, Agent 테이블과 조인하여 해당 에이전트의 이름과 함께 반환한다.

5) 쿼리 5

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

5번 제시문을 수행하기 위해 각각 studio, one-bedroom, multi-bedroom apartment(s), and detached house 별로 동일한 작업을 시행한다. 먼저 studio에 대해 다음의 쿼리를 수행한다.

```
char query_studio[] = "SELECT Property.PropertyID FROM Property JOIN Transaction ON
Property.PropertyID = Transaction.PropertyID WHERE Property.Type = 'Studio' ORDER BY
Transaction.ListingPrice DESC LIMIT 1";
```

위 작업을 Property 테이블에서 가장 비싼 스튜디오의 PropertyID를 가져오고 PropertyID를 사용하여 Photo 테이블에서 해당 스튜디오의 사진을 가져오는 다음의 photo_query 쿼리를 실행한다

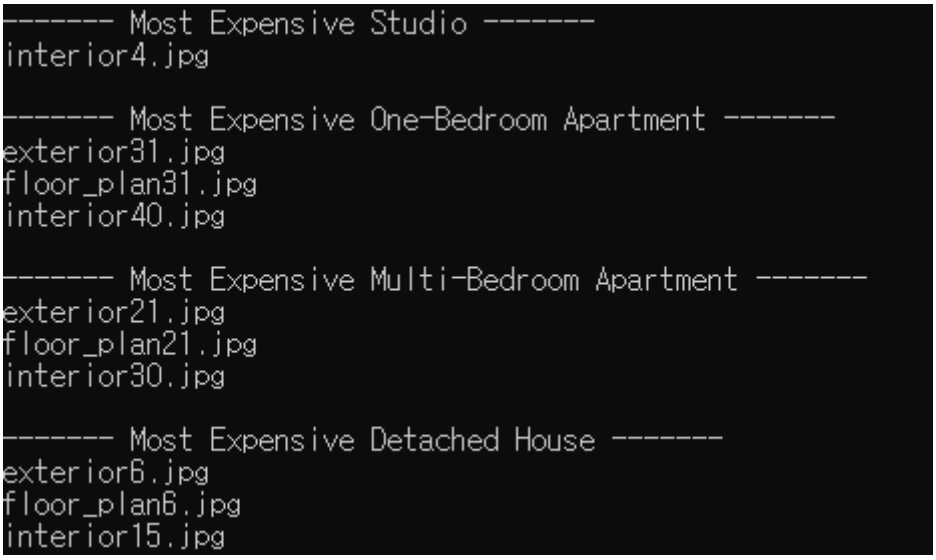
```
sprintf(photo_query, "SELECT Photo.image FROM Photo WHERE Photo.PropertyID = '%s'",
property_id);
```

단, 이 때 image 데이터 값은 원래 실제 데이터베이스 설계를 할 때는 이미지 파일이 들어가야하지만 여기서는 txt로 대체했다.

위 과정과 마찬가지로 one-bedroom, multi-bedroom apartment(s), and detached house에 대해서도 똑 같은 과정을 반복한다.단, 이때 apartment의 경우 부동산의 Type 뿐만 아니라 NUMberofBedroo,도 체크해 원룸이냐 여러 개의 방이냐를 구분한다.

```
char query_one_bed[] = "SELECT Property.PropertyID FROM Property JOIN Transaction ON
Property.PropertyID = Transaction.PropertyID WHERE Property.Type = 'Apartment' AND
Property.NumberOfBedrooms = 1 ORDER BY Transaction.ListingPrice DESC LIMIT 1";
```

모든 Property TYpe들에 대해 쿼리를 수행한 결과는 다음과 같다.



6) 쿼리 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

type_num이 6일 때는 단순 쿼리 결과를 출력하는 것이 아닌 기록을 해야 한다, 판매가 완료되지 않은 부동산을 판매 완료로 기록하는 기능을 수행합니다. 전체 코드는 다음과 같은 순서로 진행된다

1. 판매되지 않은 부동산 조회

```
printf("** For sale of a property that had been listed as being available.**₩n");

// Query to fetch all unsold properties
char fetch_unsold_query[] = "SELECT TransactionID FROM Transaction WHERE SalePrice IS NULL";
state = mysql_query(connection, fetch_unsold_query);
if (state == 0) {
    sql_result = mysql_store_result(connection);
    printf("------- Unsold Properties ------- ₩n");
    while ((sql_row = mysql_fetch_row(sql_result)) != NULL) {
        printf("Transaction ID: %s₩n", sql_row[0]);
    }
    mysql_free_result(sql_result);
}
else {
    printf("Failed to fetch unsold properties: %s₩n", mysql_error(connection));
    return 1; // Exit if fetching unsold properties fails
}

char transaction_id[9], buyer_agent_id[9], date_sold[20], sale_price[20], buyer_id[9];
bool valid_transaction_id = false, valid_buyer_agent_id = false, valid_buyer_id = false;
```

- Transaction 테이블에서 SalePrice가 NULL인 모든 거래를 조회한다.(for sale)

- 조회된 TransactionID를 출력한다.

- 조회 실패 시 오류 메시지를 출력하고 프로그램을 종료한다.

- 사용자는 새로 업데이트할 때에는 제시된 TransactionID중 하나만을 선택해야 한다.

```
** Record the sale of a property that had been listed as being available.**
------- Unsold Properties -------
Transaction ID: T0000010
Transaction ID: T0000011
Transaction ID: T0000012
Transaction ID: T0000013
Transaction ID: T0000014
Transaction ID: T0000015
Transaction ID: T0000016
Transaction ID: T0000017
Transaction ID: T0000018
Transaction ID: T0000019
Transaction ID: T0000020
Transaction ID: T0000021
Transaction ID: T0000022
Transaction ID: T0000023
Transaction ID: T0000024
Transaction ID: T0000025
Transaction ID: T0000026
Enter Transaction ID of the property to be marked as sold:
```

판매되지 않은 부동산들의 TransactionID 가 출력된 모습

2. 판매 정보 입력 및 유효한 입력값 검증

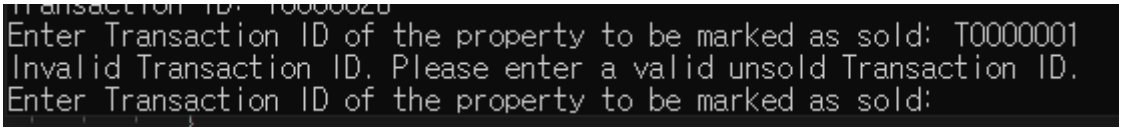
```
// Validate Transaction ID
while (!valid_transaction_id) {
    printf("Enter Transaction ID of the property to be marked as sold: ");
    scanf("%8s", transaction_id);

    // Check if entered Transaction ID is unsold
    sprintf(fetch_unsold_query, "SELECT COUNT(*) FROM Transaction WHERE TransactionID = '%s' AND SalePrice IS NULL", transaction_id);
    state = mysql_query(connection, fetch_unsold_query);
    if (state == 0) {
        sql_result = mysql_store_result(connection);
        if ((sql_row = mysql_fetch_row(sql_result)) != NULL && atoi(sql_row[0]) > 0) {
            valid_transaction_id = true;
        }
        else {
            printf("Invalid Transaction ID. Please enter a valid unsold Transaction ID.\n");
        }
        mysql_free_result(sql_result);
    }
    else {
        printf("Failed to verify Transaction ID: %s\n", mysql_error(connection));
        return 1; // Exit if verifying Transaction ID fails
    }
}
```

- 쿼리 생성: 입력된 Transaction ID가 판매되지 않은 상태인지 확인하는 쿼리를 생성한다.

SELECT COUNT(*) FROM Transaction WHERE TransactionID = '%s' AND SalePrice IS NULL 쿼리를 사용하여, 해당 Transaction ID의 판매 가격이 NULL인지 확인한다.

- 쿼리 실행 및 결과 확인:

쿼리를 실행하고 결과를 저장한다. 결과가 0보다 크면(atoi(sql_row[0]) > 0), 즉 유효한 Transaction ID이면 valid_transaction_id를 true로 설정합니다. 그렇지 않으면 사용자에게 잘못된 Transaction ID를 입력했음을 알리고 다시 입력을 요청한다.



<다시 입력 요청한 프로그램 화면>,

- 쿼리 실패 처리: 쿼리 실행이 실패하면 오류 메시지를 출력하고 프로그램을 종료합니다.

위와 같은 유효값 검증을 BuyerID, BuyerAgentID에 대해서도 반복한다, 만일 사용자가 데이터베이스에 없는 BuyerID나 BuyerAgentID를 입력했을 경우 다시 입력하게 한다.

이후 판매완료 날짜와 판매 가격을 입력받는다, 이때 판매 완료 날짜의 경우 판매 등록 날짜(SaleDate)보다 나중인지를 확인해야 한다.

```
// Validate Date Sold
while (!valid_date_sold) {
    printf("Enter Date Sold (YYYY-MM-DD): ");
    scanf("%19s", date_sold);

    // Fetch SaleDate for the given TransactionID
    char fetch_sale_date_query[256];
    sprintf(fetch_sale_date_query, "SELECT SaleDate FROM Transaction WHERE TransactionID = '%s'", transaction_id);
    state = mysql_query(connection, fetch_sale_date_query);
    if (state == 0) {
        sql_result = mysql_store_result(connection);
        if ((sql_row = mysql_fetch_row(sql_result)) != NULL) {
            // Compare SaleDate and DateSold
            if (strcmp(sql_row[0], date_sold) < 0) {
                valid_date_sold = true;
            }
            else {
                printf("Invalid Date Sold. It must be later than the SaleDate (%s).\n", sql_row[0]);
            }
        }
        mysql_free_result(sql_result);
    }
    else {
        printf("Failed to fetch SaleDate: %s\n", mysql_error(connection));
        return 1; // Exit if fetching SaleDate fails
    }
```

```
Enter Date Sold (YYYY-MM-DD): 1999-09-09
Invalid Date Sold. It must be later than the SaleDate (2023-04-01).
Enter Date Sold (YYYY-MM-DD):
```

<SaleDate보다 이전 값을 입력해 다시 입력해야 되는 상황>

3. 거래 업대이트 쿼리 생성 및 실행

sprintf(query, "UPDATE Transaction SET BuyerAgentID = '%s', BuyerID = '%s', DateSold = '%s', SalePrice = %s WHERE TransactionID = '%s' AND SalePrice IS NULL", buyer_agent_id, buyer_id, date_sold, sale_price, transaction_id);

업데이트 쿼리를 생성하여 실행한다. 성공적으로 업데이트되면 트랜잭션을 커밋하고 업데이트 또는 커밋이 실패하면 오류 메시지를 출력한다.

위 코드들을 통해 사용자로부터 유효한 Transaction ID, Buyer Agent ID, Buyer ID를 입력받고, 해당 정보를 사용하여 Transaction 테이블을 업데이트하고, 이를 데이터베이스에 반영힌다.

```
Enter Transaction ID of the property to be marked as sold: T0000001
Invalid Transaction ID. Please enter a valid unsold Transaction ID.
Enter Transaction ID of the property to be marked as sold: T0000010
Enter Buyer Agent ID: B0000001
Invalid Buyer Agent ID. Please enter a valid Buyer Agent ID.
Enter Buyer Agent ID: A0000001
Enter Buyer ID: B0000001
Enter Date Sold (YYYY-MM-DD): 1999-09-09
Invalid Date Sold. It must be later than the SaleDate (2023-04-01).
Enter Date Sold (YYYY-MM-DD): 2023-03-03
Invalid Date Sold. It must be later than the SaleDate (2023-04-01).
Enter Date Sold (YYYY-MM-DD): 2023-04-05
Enter Sale Price: 1000
Transaction updated successfully.
Transaction committed successfully.
```

<데이터를 업데이트한 프로그램 화면>

위 과정을 통해 아래 mysql workbench 화면에서 실제로 해당 데이터가 업데이트 된 것을 확인할 수 있다.

▶	T0000010	P0000010	A0000001	A0000003	2023-04-05	2023-04-01	1000	1200000000	S0000003	B0000001

7) (TYPE 7) Add a new agent to the database

위 쿼리를 싫행ㅎ기 위한 프로그램은 다음의 과정들을 거친다.

1. 사용자 입력 받기

```c
char agent_id[9], name[33], contact_info[33];
printf("Enter Agent ID: ");
scanf("%8s", agent_id);
printf("Enter Agent Name: ");
scanf("%32s", name);
printf("Enter Agent Contact Info: ");
scanf("%32s", contact_info);
```

2. 데이터베이스에 새로운 에이전트 추가

sprintf(query, "INSERT INTO Agent (AgentID, Name, ContactInfo) VALUES ('%s', '%s', '%s')", agent_id, name, contact_info);
위 쿼리를 통해 입력된 정보를 기반으로 Agent 테이블에 새로운 에이전트를 추가하는 SQL 쿼리를 생성한다.

이 때 이미 존재하는 agentID를 입력했을 경우 아래와 같은 오류가 뜬다.

```
** Add a new agent to the database.**
Enter Agent ID: A0000001
Enter Agent Name: f
Enter Agent Contact Info: f
Failed to add new agent: Duplicate entry 'A0000001' for key 'agent.PRIMARY'
-------- SELECT QUERY TYPES --------
```

<A00000001은 이미 존재하기 때문에 발생한 오류>

3. 성공하면 성공 메시지 출력 후 모든 agent 출력

정상적으로 agent 테이블에 새 agent가 입력됐을 경우 성공 메시지를 출력하고 그 결과를 확인시켜주기 위해 모든 agent들을 출력해서 보여준다.

```
if (state == 0) {
    printf("New agent added successfully.₩n");

    // Commit the transaction
    if (mysql_commit(connection) == 0) {
        printf("Transaction committed successfully.₩n");

        // Print all agents
        printf("₩n---- All Agents ----₩n");
        char all_agents_query[] = "SELECT * FROM Agent";
        state = mysql_query(connection, all_agents_query);
        if (state == 0) {
            sql_result = mysql_store_result(connection);
            printf("AgentID₩t₩tName₩t₩tContactInfo₩n");
            while ((sql_row = mysql_fetch_row(sql_result)) != NULL) {
                printf("%s₩t₩t%s₩t₩t%s₩n", sql_row[0], sql_row[1], sql_row[2]);
            }
            mysql_free_result(sql_result);
        }
        else {
            printf("Failed to retrieve agents: %s₩n", mysql_error(connection));
        }
    }
    else {
        printf("Failed to commit transaction: %s₩n", mysql_error(connection));
    }
}
else {
    printf("Failed to add new agent: %s₩n", mysql_error(connection));
}
```

```
---- All Agents ----
AgentID         Name            ContactInfo
A0000001    John Doe            john.doe@example.com
A0000002    Jane Smith          jane.smith@example.com
A0000003    Emily Davis         emily.davis@example.com
A0000004    Michael Brown       michael.brown@example.com
A0000005    Sarah Wilson        sarah.wilson@example.com
A0000006    Anna Lee            anna.lee@example.com
A0000007    David Kim           david.kim@example.com
A0000008    Laura Park          laura.park@example.com
A0000009    James Choi          james.choi@example.com
A0000010    Emily Lee           emily.lee@example.com
A0000011    abcd                dd
```

<새롭게 A0000011 Agent를 입력한 결과 모습>