DB Project2 Report

20191048 김도솔

1. BCNF Decomposition

Project 1에서 설계한 부동산의 관계형 스키마는 다음과 같다.

buyer(buyer_id, name, phone_number)
agent(agent_id, name, phone_number, hire_date)
sale(sale_id, sale_price, sale_date, property_id(FK), buyer_id (FK), selling_agent_id (FK), buyers_agent_id (FK))
property(property_id, address, price, type, number_of_bedrooms, number_of_bathrooms, status, listing_date, seller_id (FK), district_id(FK))
seller(seller_id, name, phone_number)
photo(photo_id, type, file_path, property_id(FK))
district(district_id, school_district)

이를 Project Requirements에 따라 각 relation에 관해 BCNF에 해당되는지 검사한 결과는 다음과 같다.

1) **buyer(buyer_id(PK), name, phone_number)**

   functional dependency: buyer_id → name, phone_number

   PK인 buyer_id에 의해 다른 모든 속성이 결정되기 때문에 BCNF를 만족시킨다. 이 외에 다른 functional dependency는 나타나지 않는다. name의 경우 동명이인이 존재할 수 있기 때문에 다른 속성을 결정시킬 수 없어 제외했다.

2) **agent(agent_id(PK), name, phone_number, hire_date)**

   functional dependency: agent_id → name, phone_number, hire_date

   PK인 agent_id에 의해 다른 모든 속성이 결정되기 때문에 BCNF를 만족시킨다. 이 외에 다른 functional dependency는 나타나지 않는다. 마찬가지로 name의 경우 동명이인이 존재할 수 있기 때문에 다른 속성을 결정시킬 수 없어 제외했다.

3) **sale(sale_id(PK), sale_price, sale_date, property_id(FK), buyer_id (FK), selling_agent_id (FK) , buyers_agent_id (FK))**

   functional dependency: sale_id → sale_price, sale_date, property_id, buyer_id, selling_agent_id, buyers_agent_id

   PK인 sale_id에 의해 다른 모든 속성이 결정되기 때문에 BCNF를 만족시킨다. 이 외에

다른 functional dependency는 나타나지 않는다.

4) property(property_id(PK), address, price, type, number_of_bedrooms, number_of_bathrooms, status, listing_date, seller_id (FK), district_id(FK))

functional dependency:

a. property_id → address, price, type, number_of_bedrooms, number_of_bathrooms, status, listing_date, seller_id, district_id

b. address → property_id, price, type, number_of_bedrooms, number_of_bathrooms, status, listing_date, seller_id, district_id

a의 경우 PK인 property_id에 의해 다른 모든 속성이 결정되고, b의 경우 super key인 address에 의해 다른 모든 속성이 결정된다. address는 property의 도로명주소 + 상세주소를 나타낸다. 도로명주소는 원칙적으로 도로명이 전국에서 중복되지 않도록 이름을 부여하기 때문에 모든 매물은 동일한 address 값을 가질 수 없다. 따라서 고유성이 보장되므로 이는 super key로 간주될 수 있다. a, b 이 외에 다른 functional dependency는 나타나지 않기에 이 릴레이션은 BCNF를 만족시킨다.

5) seller(seller_id(PK), name, phone_number)

functional dependency: (seller_id) → (name, phone_number)

PK인 seller_id에 의해 다른 모든 속성이 결정되기 때문에 BCNF를 만족시킨다. 이 외에 다른 functional dependency는 나타나지 않는다. 마찬가지로 name의 경우 동명이인이 존재할 수 있기 때문에 다른 속성을 결정시킬 수 없어 제외했다.

6) photo(photo_id(PK), type, file_path, property_id(FK))

functional dependency:

a. photo_id → type, file_path, property_id

b. file_path → photo_id, type, property_id

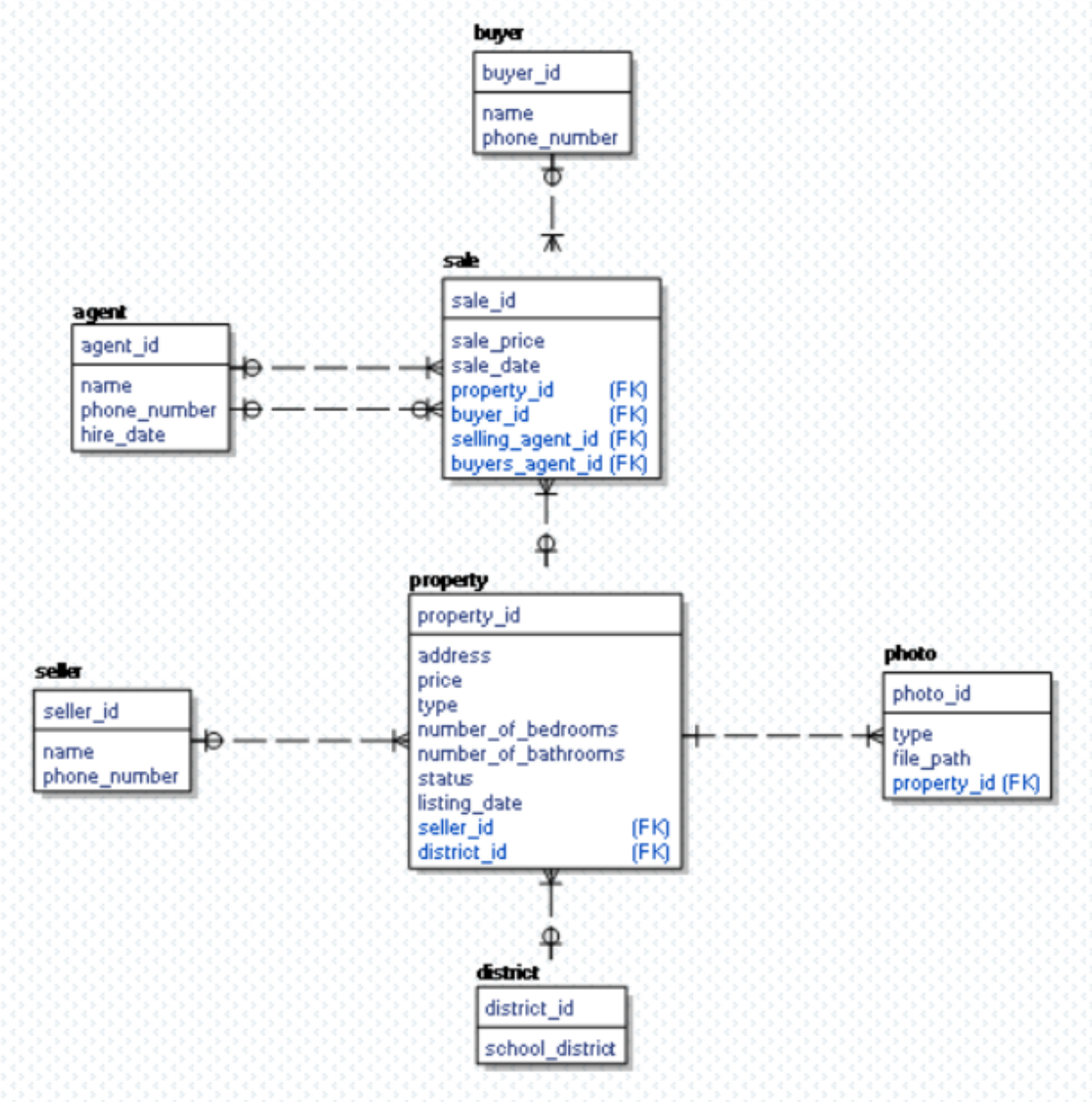
a의 경우 PK인 photo_id에 의해 다른 모든 속성이 결정된다. b의 경우 super key인 file_path에 의해 다른 모든 속성이 결정된다. file_path는 photo마다 고유한 파일 저장 경로를 나타내므로 모든 사진은 동일한 file_path 값을 가질 수 없다. 따라서 고유성이 보장되므로 이는 super key로 간주될 수 있다. a, b 이 외에 다른 functional dependency는 나타나지 않기에 이 릴레이션은 BCNF를 만족시킨다.

7) district(district_id(PK), school_district)

functional dependency: district_id → school_district

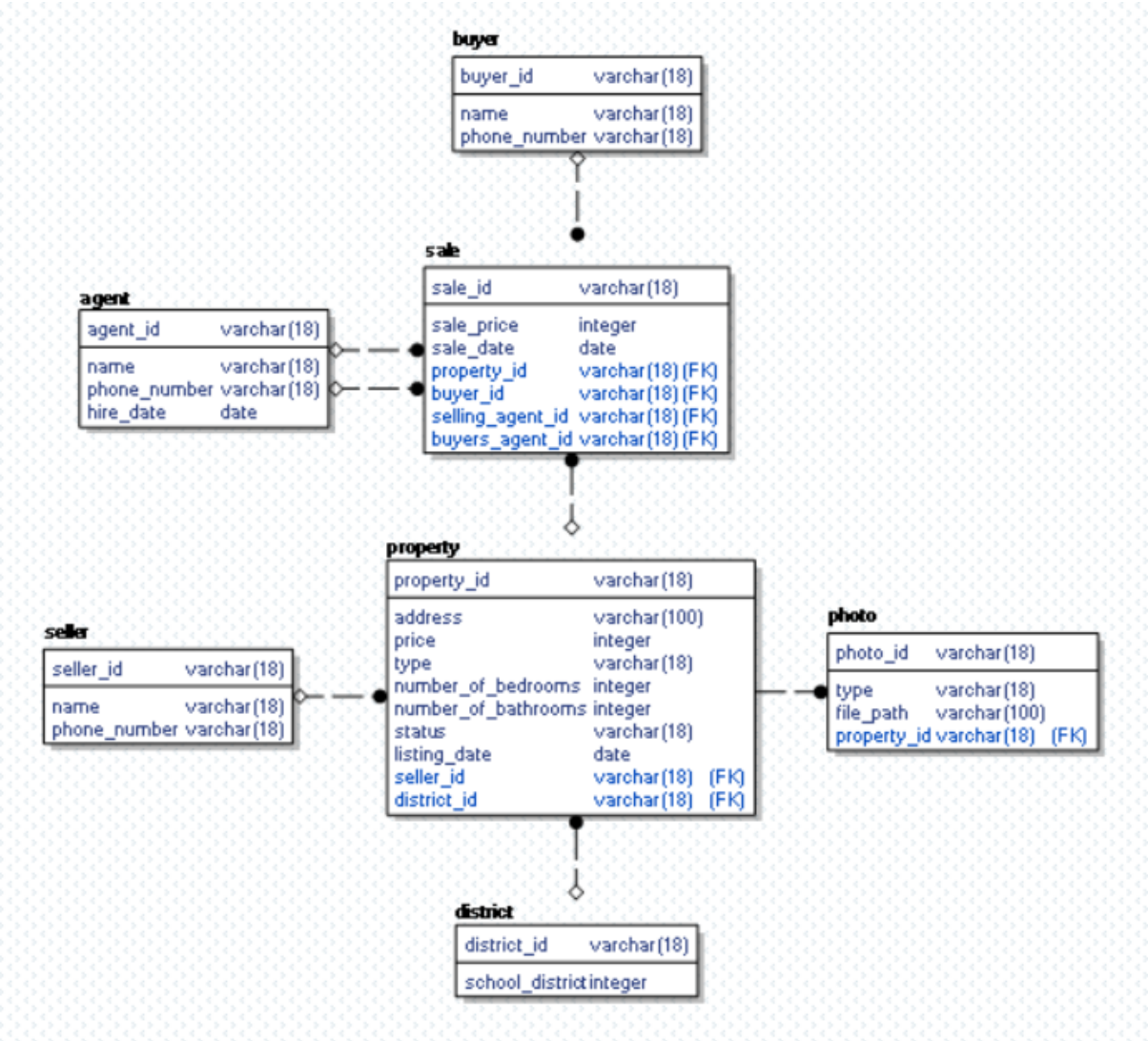
PK인 district_id에 의해 다른 모든 속성이 결정되기 때문에 BCNF를 만족시킨다. 이 외에 다른 functional dependency는 나타나지 않는다.

위의 분석에 따르면 Project 1에서 구성한 logical schema의 모든 relation은 BCNF를 만족시킨다. 따라서 Decomposed Logical Schema Diagram은 Project 1의 Logical Schema Diagram과 동일하다.



[Decomposed Logical Schema Diagram]

2. Physical Schema Diagram



위의 이미지는 Erwin을 이용해 만든 Physical Schema Diagram이다. Project Requirements에 따라 data types, domain, constraints, relationship type, allowing nulls 등을 설정했다. 각 릴레이션별 상세한 속성 설정은 다음과 같다.

2-1. Attribute detail

entity	attribute	data type, null option	상세 설명
agent	agent_id(PK)	varchar(18), not null	에이전트의 고유 식별자. 'AGT0001'과 같은 형식으로 저장되어 varchar(18)로 설정했다. 또한 PK로 모든 에이전트를 유일하게 식별하는 데 사용되기에 not null로 설정했다.
	name	varchar(18), not null	에이전트의 이름. 'Dosol Kim'과 같은 값이 들어가기에 varchar(18)로 설정했다. 또한 모든 에이전트의 이름을 알아야 하기 때문에 not null로 설정했다.
	phone_number	varchar(18), not null	에이전트의 업무 전화번호. '010-1234-5678'과 같은 값이 입력되기에 varchar(18)로 설정했다. 모든 에이전트는 업무 전화번호를 가지고 있어야 하기 때문에 not null로 설정했다.
	hire_date	date, not null	에이전트가 고용된 날짜. 에이전트의 경력을 추적하는 데 사용될 수 있다. '2023-03-05'와 같은 형식으로 입력되기 때문에 date type으로 설정했다. 모든 에이전트는 고용 날짜가 존재하기 때문에 not null로 설정했다.
buyer	buyer_id(PK)	varchar(18), not null	구매자의 고유 식별자. 'BUY0001'과 같은 형식으로 저장되어 varchar(18)로 설정했다. 또한 PK로 모든 구매자를 유일하게 식별하는 데 사용되기에 not null로 설정했다.
	name	varchar(18), not null	구매자의 이름. 'Dosol Kim'과 같은 값이 들어가기에 varchar(18)로 설정했다. 또한 모든 구매자의 이름을 알아야 하기 때문에 not null로 설정했다.
	phone_number	varchar(18), not null	구매자의 전화번호. '010-1234-5678'과 같은 값이 입력되기에 varchar(18)로 설정했다. 모든 구매자의 전화번호를 가지고 있어야 하기 때문에 not null로 설정했다.
seller	seller_id(PK)	varchar(18),	판매자의 고유 식별자. 'SEL0001'과 같은

		not null	형식으로 저장되어 varchar(18)로 설정했다. 또한 PK로 모든 판매자를 유일하게 식별하는 데 사용되기에 not null로 설정했다.
	name	varchar(18), not null	판매자의 이름. 'Dosol Kim'과 같은 값이 들어가기에 varchar(18)로 설정했다. 또한 모든 판매자의 이름을 알아야 하기 때문에 not null로 설정했다.
	phone_number	varchar(18), not null	판매자의 전화번호. '010-1234-5678'과 같은 값이 입력되기에 varchar(18)로 설정했다. 모든 판매자의 전화번호를 가지고 있어야 하기 때문에 not null로 설정했다.
property	property_id(PK)	varchar(18), not null	부동산의 고유 식별자. 'P0001'과 같은 형식으로 저장되어 varchar(18)로 설정했다. 또한 PK로 모든 부동산 유일하게 식별하는 데 사용되기에 not null로 설정했다.
	address	varchar(100), not null	정보의 중복을 방지하기 위해 시, 구를 제외한 부동산의 주소(도로명주소 + 상세주소)를 나타낸다. 예를 들어 '201, 35, Baekbeom-ro'와 같은 값이 들어간다. 주소의 길이는 넉넉하게 varchar(100)으로 설정했다. 모든 property는 상세주소를 가지고 있어야 하기 때문에 not null로 설정했다. query에서 주소 출력이 필요한 경우, 명세서에 이 office는 서울시의 매물만 다룬다고 나와있으므로 서울시는 고정으로 출력하고, 구 정보는 district_id(FK)를 통해 참조해 전체 주소를 출력할 수 있다.
	price	integer, not null	부동산이 시장에 올라온 가격. 예를 들어 '1,000,000,000'과 같은 값이 들어간다. 단위는 won(₩)이고 정수 단위로 측정되기에 integer로 설정했다. 모든 부동산은 시장 가격이 존재해야 하기 때문에 not null로 설정했다.
	type	varchar(18), not null	부동산의 유형. studio, one_bedroom, multi_bedroom, detached_house로 구성

			된다. 따라서 varchar(18)로 설정했고, 모든 부동산은 type이 정해져야 하기 때문에 not null로 설정했다.
	number_of_bedrooms	integer, not null	부동산에 포함된 침실의 수. 0, 1, 2 등의 값이 들어가기 때문에 integer로 설정했고, 모든 부동산은 침실 수가 입력되어야 하므로 not null로 설정했다.
	number_of_bathrooms	integer, not null	부동산에 포함된 욕실의 수. 0, 1, 2 등의 값이 들어가기 때문에 integer로 설정했고, 모든 부동산은 욕실 수가 입력되어야 하므로 not null로 설정했다
	status	varchar(18), not null	부동산의 판매 상태 on_the_market, sold_out으로 구성된다. 따라서 varchar(18)으로 설정했고, 모든 부동산은 판매 상태가 정해져야 하기 때문에 not null로 설정했다.
	listing_date	date, not null	부동산이 시장에 등록된 날짜. '2023-03-05'와 같은 형식으로 입력되기 때문에 date type으로 설정했다. 모든 부동산은 시장 등록 날짜가 존재해야 하기 때문에 not null로 설정했다.
	seller_id (FK)	varchar(18), not null	이 부동산을 판매하는 판매자의 id를 외래 키로 참조한다. 모든 부동산은 판매자 정보가 있어야 하기에 not null로 설정했다.
	district_id(FK)	varchar(18), not null	이 부동산이 속한 지구의 id를 외래키로 참조한다. 모든 부동산은 지구 정보가 있어야 하기에 not null로 설정했다.
sale	sale_id(PK)	varchar(18), not null	판매 기록의 고유 식별자. 'S0001'과 같은 형식으로 저장되어 varchar(18)로 설정했다. 또한 PK로 모든 판매 기록을 유일하게 식별하는 데 사용되기에 not null로 설정했다.
	sale_price	integer, not null	부동산이 실제 거래된 가격. 예를 들어 '1,000,000,000'과 같은 값이 들어간다. 단위는 won(₩)이고 정수 단위로 측정되기에 integer로 설정했다. 모든 판매 기록은 부동산 거래 가격이 존재해야 하기 때문에 not null로 설정했다.

	sale_date	date, not null	판매가 완료된 날짜. '2023-03-05'와 같은 형식으로 입력되기 때문에 date type으로 설정했다. 모든 판매 기록은 거래가 성사된 날짜가 존재해야 하기 때문에 not null로 설정했다.
	property_id(FK)	varchar(18), not null	판매된 부동산의 id를 외래 키로 참조한다. 모든 판매 기록에는 거래된 부동산의 정보가 있어야 하기에 not null로 설정했다.
	buyer_id (FK)	varchar(18), not null	부동산을 구매한 구매자의 id를 외래 키로 참조한다. 모든 판매 기록에는 거래에 참여한 구매자의 정보가 있어야 하기에 not null로 설정했다.
	selling_agent_id (FK)	varchar(18), not null	부동산의 판매를 담당한 에이전트의 id를 외래 키로 참조한다. 모든 판매 기록에는 거래에 참여한 agent의 정보가 있어야 하기에 not null로 설정했다.
	buyers_agent_id (FK)	varchar(18), null	구매자를 대리한 에이전트의 id를 외래 키로 참조한다. 이때 명세서에 작성된 대로 buyer의 agent는 존재하는 경우에만 기록할 수 있도록 null 값을 허용했다.
photo	photo_id(PK)	varchar(18), not null	사진의 고유 식별자. 'IMG0001'과 같은 형식으로 저장되어 varchar(18)로 설정했다. 또한 PK로 모든 사진을 유일하게 식별하는 데 사용되기에 not null로 설정했다.
	type	varchar(18), not null	사진의 유형. interior, exterior, floor_plan으로 구성된다. 따라서 varchar(18)으로 설정했고, 모든 사진은 유형이 정해져야 하기 때문에 not null로 설정했다.
	file_path	varchar(100), not null	사진 파일이 저장된 경로. 'C:₩Users₩user₩Pictures₩photo.jpg'와 같은 형식으로 저장되어. varchar(100)으로 설정했다. 모든 사진은 저장된 경로가 존재해야 하기 때문에 not null로 설정했다.
	property_id(FK)	varchar(18), not null	사진이 찍힌 부동산의 id를 외래 키로 참조한다. 모든 사진은 부동산 정보가

			있어야하기 때문에 not null로 설정했다.
district	district_id(PK)	varchar(18), not null	서울 지구의 고유 식별자. Mapo-gu, Seodaemun-gu, Nowon-gu 등으로 구성된다. 따라서 varchar(18)으로 설정했고, PK로 모든 지구를 유일하게 식별하는 데 사용되기에 not null로 설정했다.
	school_district	integer, not null	각 지구가 속한 학군. 1, 2, 3 등으로 구성된다. 서울의 모든 지구는 학군 정보가 존재해야 하기에 not null로 설정했다.

3. Queries

다음은 프로젝트에서 설계한 모델을 기반으로 명세서에서 요구되는 질의문을 작성한 결과이다.

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

```
SELECT property_id, address
FROM property
WHERE district_id = 'Mapo-gu'
  AND status = 'on_the_market'
```

이 쿼리는 부동산 데이터베이스에서 마포구(Mapo-gu)에 위치하고 현재 판매 중(on_the_market)인 집의 고유 식별자(property_id)와 주소(address)를 조회한다. property 테이블에서 district_id 가 Mapo-gu 이고 status 가 on_the_market 인 행을 선택하여 해당 집들의 정보를 반환하도록 구성했다.

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

```
SELECT property_id, address, price
FROM property
WHERE district_id = 'Mapo-gu'
  AND status = 'on_the_market'
  AND price BETWEEN 1000000000 AND 1500000000
```

이 쿼리는 마포구(Mapo-gu)에 위치하고 현재 판매 중(on_the_market)이며, 가격이 10 억 원에서 15 억 원 사이인 집의 고유 식별자(property_id), 주소(address), 및 가격(price)을 조회한다. property 테이블에서 district_id 가 Mapo-gu, status 가 on_the_market, 그리고 price 가 1,000,000,000 원에서 1,500,000,000 원 사이인 행을 선택하여 해당 집들의 정보를 반환하도록 구성했다.

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

```
SELECT property.property_id, property.address, property.district_id
FROM property
JOIN district ON property.district_id = district.district_id
WHERE district.school_district = 8
  AND property.status = 'on_the_market'
```

이 쿼리는 8 학군(school_district = 8)에 위치하고 현재 판매 중(on_the_market)인 집들의 고유 식별자(property_id), 주소(address), 그리고 지구 식별자(district_id)를 조회한다. property 테이블과 district 테이블을 district_id 를 기준으로 조인(JOIN)하여, district.school_district 가 8 인 행을 선택하고, 그 중에서 property.status 가 on_the_market 인 행만 반환하도록 구성했다.

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

```
SELECT property.property_id, property.address, property.district_id,
property.number_of_bedrooms, property.number_of_bathrooms
FROM property
JOIN district ON property.district_id = district.district_id
WHERE district.school_district = 8
  AND property.status = 'on_the_market'
  AND property.number_of_bedrooms >= 4
  AND property.number_of_bathrooms >= 2
```

이 쿼리는 8 학군(school_district = 8)에 위치하고 현재 판매 중(on_the_market)인 집들 중에서 침실이 4 개 이상(number_of_bedrooms >= 4)이고 욕실이 2 개 이상(number_of_bathrooms >= 2)인 집들의 고유 식별자(property_id), 주소(address), 지구 식별자(district_id), 침실 수(number_of_bedrooms), 및 욕실 수(number_of_bathrooms)를 조회한다. property 테이블과 district 테이블을 district_id 를 기준으로 조인(JOIN)하여, district.school_district 가 8 인 행을 선택하고, 그 중에서 property.status 가 on_the_market 이면서 침실과 욕실 조건을 만족하는 행만 반환하도록 구성했다.

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

```
SELECT agent.agent_id, agent.name, SUM(sale.sale_price)
FROM sale
JOIN agent ON sale.selling_agent_id = agent.agent_id
WHERE YEAR(sale_date) = 2022
GROUP BY agent.agent_id
ORDER BY SUM(sale_price) DESC
LIMIT 1
```

이 쿼리는 2022 년에 가장 많은 매출을 올린 에이전트의 ID(agent_id), 이름(name), 총 매출액(SUM(sale.sale_price))을 조회한다. sale 테이블과 agent 테이블을 selling_agent_id 로 조인(JOIN)하여 2022 년에 판매된 매물만 선택하고(YEAR(sale_date) = 2022), 에이전트별로 그룹화(GROUP BY agent.agent_id)한 뒤 총 매출액 기준으로 내림차순 정렬(ORDER BY SUM(sale_price) DESC)하여 가장 상위의 에이전트 하나를 반환(LIMIT 1)하도록 구성했다.

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

```
SELECT agent.agent_id, agent.name, SUM(sale.sale_price) AS total_sales
FROM sale
JOIN agent ON sale.selling_agent_id = agent.agent_id
WHERE YEAR(sale.sale_date) = 2023
GROUP BY agent.agent_id, agent.name
ORDER BY total_sales DESC
```

```
LIMIT k
```

이 쿼리는 2023 년에 가장 많은 매출을 올린 상위 k 명의 에이전트의 ID(agent_id), 이름(name), 총 매출액(total_sales)을 조회한다. sale 테이블과 agent 테이블을 selling_agent_id 로 조인(JOIN)하여 2023 년에 판매된 매물만 선택하고(YEAR(sale.sale_date) = 2023), 에이전트별로 그룹화(GROUP BY agent.agent_id, agent.name)한 뒤 총 매출액 기준으로 내림차순 정렬(ORDER BY total_sales DESC)하여 상위 k 명의 에이전트를 반환(LIMIT k)하도록 구성했다.

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

```
SELECT CEIL(COUNT(DISTINCT selling_agent_id) * 0.1)
FROM sale
WHERE YEAR(sale_date) = 2021
```

첫 번째 쿼리는 2021 년에 판매를 기록한 에이전트의 수를 기준으로 하여 하위 10%에 해당하는 에이전트의 수(count)를 계산한다. sale 테이블에서 2021 년에 판매된 매물의 selling_agent_id 를 기준으로 중복을 제거하여(DISTINCT), 에이전트 수의 10%를 계산한다(CEIL(COUNT(DISTINCT selling_agent_id) * 0.1)). cpp 코드 상에서는 이 값을 count 변수에 할당한 후 두 번째 쿼리를 실행시키도록 하였다.

```
SELECT agent.agent_id, agent.name, SUM(sale.sale_price) AS total_sales
FROM sale
JOIN agent ON sale.selling_agent_id = agent.agent_id
WHERE YEAR(sale.sale_date) = 2021
GROUP BY agent.agent_id, agent.name
ORDER BY total_sales ASC
LIMIT count
```

두 번째 쿼리는 첫 번째 쿼리에서 계산된 count 값을 사용하여, 2021 년에 가장 적은 매출을 올린 하위 10% 에이전트의 ID(agent_id), 이름(name), 총 매출액(total_sales)을 조회한다. sale 테이블과 agent 테이블을 selling_agent_id 로 조인(JOIN)하여 2021 년에 판매된 매물만 선택하고(YEAR(sale.sale_date) = 2021), 에이전트별로 그룹화(GROUP BY agent.agent_id, agent.name)한 뒤 총 매출액 기준으로 오름차순 정렬(ORDER BY total_sales ASC)하여 하위 count 명의 에이전트를 반환(LIMIT count)하도록 구성했다.

(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.

```
SELECT s.selling_agent_id, AVG(s.sale_price) AS avg_selling_price_2022,
AVG(TIMESTAMPDIFF(DAY, p.listing_date, s.sale_date)) AS avg_time_on_market_days_2022
FROM sale s
JOIN property p ON s.property_id = p.property_id
WHERE YEAR(s.sale_date) = 2022
```

```
GROUP BY s.selling_agent_id
```

이 쿼리는 2022 년에 판매된 매물에 대해 각 에이전트의 평균 판매 가격과 매물이 시장에 나와 있었던 평균 일수를 계산한다. sale 테이블과 property 테이블을 property_id 로 조인(JOIN)하여 2022 년에 판매된 매물만 선택하고(YEAR(s.sale_date) = 2022), 에이전트별로 그룹화(GROUP BY s.selling_agent_id)한 뒤 각 에이전트의 평균 판매 가격과 평균 시장 나와 있던 일수를 계산(AVG())한다.

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

```
SELECT s.selling_agent_id, MAX(s.sale_price) AS max_selling_price_2023
FROM sale s
WHERE YEAR(s.sale_date) = 2023
GROUP BY s.selling_agent_id
```

이 쿼리는 2023 년에 판매된 매물 중 각 에이전트가 판매한 매물의 최고 판매 가격을 계산한다. sale 테이블에서 2023 년에 판매된 매물만 선택하고(YEAR(s.sale_date) = 2023), 에이전트별로 그룹화(GROUP BY s.selling_agent_id)한 뒤 각 에이전트가 판매한 매물의 최고 판매 가격을 계산(MAX(s.sale_price))한다.

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

```
SELECT s.selling_agent_id, MAX(TIMESTAMPDIFF(DAY, p.listing_date, s.sale_date)) AS
longest_time_on_market_days
FROM sale s
JOIN property p ON s.property_id = p.property_id
GROUP BY s.selling_agent_id
```

이 쿼리는 각 에이전트가 판매한 매물 중 시장에 나와 있었던 최장 기간을 계산한다. sale 테이블과 property 테이블을 property_id 로 조인(JOIN)하여, 매물이 시장에 나와 있었던 기간을 일 단위로 계산(TIMESTAMPDIFF(DAY, p.listing_date, s.sale_date))하고, 에이전트별로 그룹화(GROUP BY s.selling_agent_id)한 뒤 각 에이전트가 판매한 매물 중 시장에 나와 있었던 최장 기간을 계산(MAX())한다.

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

```
SELECT property.type, photo.type, photo.file_path, photo.property_id, property.price
FROM photo
JOIN property ON photo.property_id = property.property_id
WHERE
  property.type IN ('studio', 'one_bedroom', 'multi_bedroom', 'detached_house')
  AND property.price =
    (SELECT MAX(p2.price)
```

```
    FROM property p2
    WHERE p2.type = property.type)
```

이 쿼리는 각각 가장 비싼 스튜디오, 원룸, 멀티룸 아파트, 단독 주택의 사진이 저장된 링크를 조회한다. photo 테이블과 property 테이블을 property_id 로 조인하여, 특정 매물 유형(스튜디오, 원룸, 멀티룸 아파트, 단독 주택)에 해당하는 매물 중 가장 비싼 매물의 사진을 선택한다. property 테이블에서 각 유형별로 가장 높은 가격(MAX(p2.price))을 가진 매물을 찾고, 해당 매물의 사진을 photo 테이블에서 조회하도록 구성했다.

(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.

```
INSERT INTO sale (sale_id, sale_price, sale_date, property_id, buyer_id, selling_agent_id,
buyers_agent_id)
VALUES ('S0001', 1000000000, 2024-06-10, 'P0001', 'BUY0001', 'AGT0003', 'AGT0005')
```

이 쿼리는 시장에 나와 있던 매물의 판매 기록을 추가한다. 판매 가격, 구매자, 판매 에이전트, 구매자의 에이전트(있는 경우), 판매 날짜 등의 정보를 sale 테이블에 삽입한다. 예를 들어, 'S0001'이라는 판매 ID 로 'P0001' 매물이 'AGT0003' 판매 에이전트와 'AGT0005' 구매자 에이전트를 통해 'BUY0001' 구매자에게 10 억 원에 판매되었음을 2024 년 6 월 10 일자로 기록할 수 있다. 실제 cpp 코드에서는 사용자로부터 값을 입력 받은 후 그 값을 쿼리에 추가할 수 있도록 하였다.

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

```
INSERT INTO agent (agent_id, name, phone_number, hire_date)
VALUES ('AGT0001', 'Dosol Kim', '010-1234-5678', 2000-03-05)
```

이 쿼리는 새로운 에이전트를 데이터베이스에 추가한다. 에이전트의 ID, 이름, 전화번호, 고용 날짜 등의 정보를 agent 테이블에 삽입한다. 예를 들어, 'AGT0001'이라는 에이전트 ID 로 'Dosol Kim'이라는 이름의 에이전트가 '010-1234-5678' 전화번호로 2000 년 3 월 5 일에 고용되었음을 기록할 수 있다. 마찬가지로 실제 cpp 코드에서는 사용자로부터 값을 입력 받은 후 그 값을 쿼리에 추가할 수 있도록 하였다.

4. ODBC implementation within MySQL

1) .cpp 코드 설명

작성한 코드는 MySQL 데이터베이스와 상호작용하는 C++ 프로그램으로, 사용자로부터 입력을 받아 다양한 데이터베이스 쿼리를 실행하고, 결과를 출력한다. 프로그램의 전체 구조를 간략하게 설명하면 다음과 같다

```
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>
#include <string.h>
#include "mysql.h"

#pragma comment(lib, "libmysql.lib")

const char* host = "localhost";
const char* user = "root";
const char* pw = "ehthfsql12#";

using namespace std;

#define MAX_LEN         13000
```

stdio.h, string.h 및 mysql.h 헤더 파일을 포함한다. 이때 mysql.h는 MySQL 데이터베이스와 상호작용하기 위해 필요한 함수들을 정의한다. _CRT_SECURE_NO_WARNINGS로 보안 경고를 비활성화하고 MAX_LEN을 통해 읽을 수 있는 최대 문자열 길이를 정의한다. 또한 host, user, pw 등의 MySQL 접속 정보를 정의한다.

```
int main(void) {
    MYSQL* connection = NULL;
    MYSQL conn;
    MYSQL_RES* sql_result;
    MYSQL_ROW sql_row;

    // open CRUD file.
    FILE* fp = fopen("CRUD.txt", "rt");
    if (fp == NULL) {
        printf("file does not exist!\n");
        return 0;
    }
    char line[MAX_LEN];

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

    // the first NULL can be replaced to an existing db instance.
    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");
```

다음으로 main 함수에서는 MYSQL 객체와 기타 필요한 변수들을 선언하고 초기화한다. CRUD.txt 파일을 열어 데이터베이스 초기화 명령어(create, insert)를 읽어들인다. 그후 mysql_real_connect 함수를 사용하여 MySQL 서버에 연결한다. 연결 성공 여부를 확인하고 오류가 발생하면 프로그램을 종료한다.

```
printf("Connection Succeed\n\n");

while (fgets(line, sizeof(line), fp) != NULL) {
    if (!strcmp(line, "$$$\n"))      // read lines from CRUD file, '$$$' separates CREATE / DELETE parts.
        break;
    mysql_query(connection, line);
}

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

int select;
// start
while (1) {
    printf("\n---------- SELECT QUERY TYPES ----------\n\n");
    printf("\t1. TYPE 1\n");
    printf("\t2. TYPE 2\n");
    printf("\t3. TYPE 3\n");
    printf("\t4. TYPE 4\n");
    printf("\t5. TYPE 5\n");
    printf("\t6. TYPE 6\n");
    printf("\t7. TYPE 7\n");
    printf("\t0. QUIT\n\n");
    printf("Enter the type: ");
    scanf("%d", &select);
    printf("\n");

    switch (select) {
```

정상적으로 연결되었으면 데이터베이스를 선택하고, 무한 루프를 통해 사용자 입력을 받아 다양한 쿼리를 실행한다. 각 메뉴는 switch-case 문을 통해 구현되어 있으며, 사용자가 입력한 번호에 따라 해당 기능을 실행한다.

```
case 0: {
    printf("\n---------- QUIT ----------\n");
    // comment out if you want to persist example db instance.
    while (fgets(line, sizeof(line), fp) != NULL)
        mysql_query(connection, line); // these are DELETEs & DROPs.
    mysql_close(connection);
    return 0;
    break;
}
```

TYPE 0: 사용자가 0을 입력하면 프로그램 종료한다. 이때 CRUD.txt에서 남은 SQL 명령어를 실행하여 데이터베이스를 초기 상태로 되돌리고, MySQL 연결을 닫는다.

```
case 1: {
    while (1) {
        printf("---------- TYPE 1 ----------\n");
        printf("** Find address of homes for sale in the district “Mapo”. **\n");

        char query[] = "SELECT property_id, address FROM property WHERE district_id = 'Mapo-gu' AND status = 'on_the_market'";

        int state = 0;
        state = mysql_query(connection, query);
        if (state == 0) {
            sql_result = mysql_store_result(connection);
            while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
            {
                printf("[%s] %s, Mapo-gu, Seoul\n", sql_row[0], sql_row[1]);
            }
            mysql_free_result(sql_result);
        }
```

TYPE 1: 마포구(Mapo-gu)에 위치하고 현재 판매 중(on_the_market)인 property를 찾는다. 그 결과

로 property_id와 full address를 출력할 수 있도록 하였다.

```
while (1) {
    int sub;
    printf("\n---------- Subtypes in TYPE 1 ----------\n");
    printf("\t1. TYPE 1-1\n");
    printf("\t0. QUIT\n\n");
    printf("Enter the subtype: ");
    scanf("%d", &sub);
    if (sub == 0) break;
    else if (sub == 1) {
        printf("\n---------- TYPE 1-1 ----------\n");
        printf("** Then find the costing between ₩1,000,000,000 and ₩1,500,000,000. **\n");

        char query[] = "SELECT property_id, address, price FROM property WHERE district_id = 'Mapo-gu' AND status = 'on_the_market' AND price BETWEEN 1000000000 AND 1500000000";

        int state = 0;
        state = mysql_query(connection, query);
        if (state == 0) {
            sql_result = mysql_store_result(connection);
            while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
            {
                printf("[%s] %s, Mapo-gu, Seoul | price: %s\n", sql_row[0], sql_row[1], sql_row[2]);
            }
            mysql_free_result(sql_result);
        }
        continue;
    }
    else {
        printf("Wrong choice.");
        continue;
    }
}
break;
```

TYPE 1-1: 마포구(Mapo-gu)에 위치하고 현재 판매 중이며 가격이 10억에서 15억 사이인 property를 찾는다. 그 결과로 property_id와 full address, price를 출력할 수 있도록 하였다.

```
case 2: {
    while (1) {
        printf("---------- TYPE 2 ----------\n");
        printf("** Find the address of homes for sale in the 8th school district. **\n");

        char query[] = "SELECT property.property_id, property.address, property.district_id FROM property JOIN district ON\
            property.district_id = district.district_id WHERE district.school_district = 8 AND property.status = 'on_the_market'";

        int state = 0;
        state = mysql_query(connection, query);
        if (state == 0) {
            sql_result = mysql_store_result(connection);
            while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
            {
                printf("[%s] %s, %s, Seoul\n", sql_row[0], sql_row[1], sql_row[2]);
            }
            mysql_free_result(sql_result);
        }
```

TYPE 2: 8학군 내 현재 판매 중인 property를 조회한다. 그 결과로 property_id와 full address를 출력할 수 있도록 하였다.

```
while (1) {
    int sub;
    printf("\n---------- Subtypes in TYPE 2 ----------\n");
    printf("\t1. TYPE 2-1\n");
    printf("\t0. QUIT\n\n");
    printf("Enter the subtype: ");
    scanf("%d", &sub);
    printf("\n");
    if (sub == 0) break;
    else if (sub == 1) {
        printf("\n---------- TYPE 2-1 ----------\n");
        printf("** Then find properties with 4 or more bedrooms and 2 bathrooms. **\n");

        char query[] = "SELECT property.property_id, property.address, property.district_id, property.number_of_bedrooms, property.number_of_bathrooms\
            FROM property JOIN district ON property.district_id = district.district_id WHERE district.school_district = 8 AND property.status = 'on_the_market'\
            AND property.number_of_bedrooms >= 4 AND property.number_of_bathrooms >= 2";

        int state = 0;
        state = mysql_query(connection, query);
        if (state == 0) {
            sql_result = mysql_store_result(connection);
            while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
            {
                printf("[%s] %s, %s, Seoul\n", sql_row[0], sql_row[1], sql_row[2]);
                printf("        number_of_bedrooms: %s, number_of_bathrooms: %s\n", sql_row[3], sql_row[4]);
            }
            mysql_free_result(sql_result);
        }
        continue;
    }
    else {
        printf("Wrong choice.");
        continue;
    }
}
break;
```

TYPE 2-1: 8학군 내 판매 중인 property 중에서 침실 4개 이상, 욕실 2개 이상인 것을 찾는다. 그 결과로 property_id와 full address, 침실 수, 욕실 수를 출력하도록 하였다.

```
case 3: {
    while (1) {
        printf("---------- TYPE 3 ----------\n");
        printf("** Find the name of the agent who has sold the most properties in the year 2022 by total won value. **\n");

        char query[] = "SELECT agent.agent_id, agent.name, SUM(sale.sale_price) FROM sale JOIN agent ON sale.selling_agent_id = agent.agent_id\
            WHERE YEAR(sale_date) = 2022 GROUP BY agent.agent_id ORDER BY SUM(sale_price) DESC LIMIT 1";

        int state = 0;
        state = mysql_query(connection, query);
        if (state == 0) {
            sql_result = mysql_store_result(connection);
            while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
            {
                printf("[%s] %s | total won value: \\%s\n", sql_row[0], sql_row[1], sql_row[2]);
            }
            mysql_free_result(sql_result);
        }
        printf("\n");
```

TYPE 3: 2022년에 가장 많은 총 금액의 부동산을 판매한 에이전트를 찾는다. 그 결과로 agent_id, name, total won value를 출력하도록 하였다.

```
while (1) {
    int sub;
    printf("\n---------- Subtypes in TYPE 3 ----------\n");
    printf("\t1. TYPE 3-1\n");
    printf("\t2. TYPE 3-2\n");
    printf("\t0. QUIT\n\n");
    printf("Enter the subtype: ");
    scanf("%d", &sub);
    printf("\n");
    if (sub == 0) break;
    else if (sub == 1) {
        printf("\n---------- TYPE 3-1 ----------\n");
        printf("** Then find the top k agents in the year 2023 by total won value. **\n");
        char k[5] = {};
        printf("Enter the k(<=10): ");
        scanf("%s", &k);
        if (k == 0) break;

        char query[300] = "SELECT agent.agent_id, agent.name, SUM(sale.sale_price) AS total_sales FROM sale JOIN agent ON sale.selling_agent_id = agent.agent_id\
            WHERE YEAR(sale.sale_date) = 2023 GROUP BY agent.agent_id, agent.name ORDER BY total_sales DESC LIMIT ";
        strcat(query, k);

        int state = 0;
        state = mysql_query(connection, query);
        if (state == 0) {
            sql_result = mysql_store_result(connection);
            while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
            {
                printf("[%s] %s | total won value: \\%s\n", sql_row[0], sql_row[1], sql_row[2]);
            }
            mysql_free_result(sql_result);
        }
        printf("\n");
        continue;
    }
```

TYPE 3-1: 2022년에 가장 많은 총 금액의 부동산을 판매한 상위 k명의 에이전트를 찾는다. 이때 사용자로부터 k 값을 입력받아 쿼리에 삽입할 수 있도록 했다. 결과로는 agent_id, name, total won value를 출력하도록 하였다.

```
else if (sub == 2) {
    printf("\n---------- TYPE 3-2 ----------\n");
    printf("** And then find the bottom 10%% agents in the year 2021 by total won value. **\n");

    // First query to count the number of agents in bottom 10%
    char count_query[] = "SELECT CEIL(COUNT(DISTINCT selling_agent_id) * 0.1) FROM sale WHERE YEAR(sale_date) = 2021";
    int count = 0;

    int state = mysql_query(connection, count_query);
    if (state == 0) {
        sql_result = mysql_store_result(connection);
        if ((sql_row = mysql_fetch_row(sql_result)) != NULL) {
            count = atoi(sql_row[0]);
        }
        mysql_free_result(sql_result);
    }

    if (count > 0) {
        // Second query to get the bottom 10% agents by total won value
        char query[512];
        snprintf(query, sizeof(query),
            "SELECT agent.agent_id, agent.name, SUM(sale.sale_price) AS total_sales FROM sale JOIN agent ON sale.selling_agent_id = agent.agent_id\
            WHERE YEAR(sale.sale_date) = 2021 GROUP BY agent.agent_id, agent.name ORDER BY total_sales ASC LIMIT %d", count);

        state = mysql_query(connection, query);
        if (state == 0) {
            sql_result = mysql_store_result(connection);
            while ((sql_row = mysql_fetch_row(sql_result)) != NULL) {
                printf("[%s] %s | total won value: \\%s\n", sql_row[0], sql_row[1], sql_row[2]);
            }
            mysql_free_result(sql_result);
        }
    }
    else {
        printf("No agents found for the bottom 10%% criteria.\n");
    }
    printf("\n");
    continue;
```

TYPE 3-2: 2021년에 총 금액 기준 하위 10%의 에이전트를 찾는다. 그 결과로 agent_id, name, total won value를 출력하도록 하였다.

```
case 4: {
    while (1) {
        printf("---------- TYPE 4 ----------\n");
        printf("** For each agent, compute the average selling price of properties sold in 2022, and the average time the property was on the market. **\n");

        char query[] = "SELECT s.selling_agent_id, AVG(s.sale_price) AS avg_selling_price_2022, AVG(TIMESTAMPDIFF(DAY, p.listing_date, s.sale_date))\
            AS avg_time_on_market_days_2022 FROM sale s JOIN property p ON s.property_id = p.property_id WHERE YEAR(s.sale_date) = 2022 GROUP BY s.selling_agent_id";

        int state = 0;
        state = mysql_query(connection, query);
        if (state == 0) {
            sql_result = mysql_store_result(connection);
            while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
            {
                double price = atof(sql_row[1]);
                double time = atof(sql_row[2]);
                printf("[%s] avg price: \\%10.0f | avg time: %4.1f days\n", sql_row[0], price, time);
            }
            mysql_free_result(sql_result);
        }
        printf("\n");
```

TYPE 4: 2022년에 에이전트별로 부동산 평균 판매 가격과 평균 판매 기간을 계산한다. 그 결과 agent_id와 avg_price, avg_time을 출력하도록 했다.

```
while (1) {
    int sub;
    printf("\n---------- Subtypes in TYPE 4 ----------\n");
    printf("\t1. TYPE 4-1\n");
    printf("\t2. TYPE 4-2\n");
    printf("\t0. QUIT\n\n");
    printf(": ");
    scanf("%d", &sub);
    if (sub == 0) break;
    else if (sub == 1) {
        printf("\n---------- TYPE 4-1 ----------\n");
        printf("** Then compute the maximum selling price of properties sold in 2023 for each agent. **\n");

        char query[] = "SELECT s.selling_agent_id, MAX(s.sale_price) AS max_selling_price_2023 FROM sale s WHERE YEAR(s.sale_date) = 2023 GROUP BY s.selling_agent_id";
        int state = 0;
        state = mysql_query(connection, query);
        if (state == 0) {
            sql_result = mysql_store_result(connection);
            while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
            {
                double price = atof(sql_row[1]);
                printf("[%s] max price: \\%10.0f\n", sql_row[0], price);
            }
            mysql_free_result(sql_result);
        }
        printf("\n");
        continue;
    }
```

TYPE 4-1: 2023년에 에이전트별로 가장 높은 판매 가격을 찾는다. 그 결과 agent_id, max price를 출력하도록 하였다.

```
else if (sub == 2) {
    printf("\n---------- TYPE 4-2 ----------\n");
    printf("** And then compute the longest time the property was on the market for each agent. **\n");

    char query[] = "SELECT s.selling_agent_id, MAX(TIMESTAMPDIFF(DAY, p.listing_date, s.sale_date)) AS longest_time_on_market_days\
        FROM sale s JOIN property p ON s.property_id = p.property_id GROUP BY s.selling_agent_id";

    int state = 0;
    state = mysql_query(connection, query);
    if (state == 0) {
        sql_result = mysql_store_result(connection);
        while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
        {
            double time = atof(sql_row[1]);
            printf("[%s] longest time: %4.1f days\n", sql_row[0], time);
        }
        mysql_free_result(sql_result);
    }
    printf("\n");
    continue;
}
else {
    printf("Wrong choice.");
    continue;
}
```

TYPE 4-2: 에이전트별로 부동산이 시장에 가장 오래 있었던 기간을 찾는다. 그 결과로 agent_id와 longest time을 출력하도록 하였다.

```c
case 5: {
    printf("---------- TYPE 5 ----------\n");
    printf("** Show photos of the most expensive studio, one-bedroom, multi-bedroom apartment(s), and detached house(s), respectively, from the database. **\n");

    char query[] = "SELECT property.type, photo.type, photo.file_path, photo.property_id, property.price FROM photo JOIN property ON photo.property_id = property.property_id\
        WHERE property.type IN ('studio', 'one_bedroom', 'multi_bedroom', 'detached_house') AND property.price = (SELECT MAX(p2.price) FROM property p2 WHERE p2.type = property.type)";
    int state = 0;
    state = mysql_query(connection, query);
    if (state == 0)
    {
        sql_result = mysql_store_result(connection);
        while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
        {
            printf("[%s] %s file path: %s (%s, price: %s)\n", sql_row[0], sql_row[1], sql_row[2], sql_row[3], sql_row[4]);
        }
        mysql_free_result(sql_result);
    }
    printf("\n");
    break;
}
```

TYPE 5: 가장 비싼 스튜디오, 원베드룸, 멀티베드룸 아파트 및 독립주택의 사진을 보여준다. 그 결과로 property의 type, photo의 type, 사진이 저장된 경로, property_id, price를 출력할 수 있도록 하였다.

```c
case 6: {
    printf("---------- TYPE 6 ----------\n");
    printf("** 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. **\n");

    char sale_id[20], property_id[20], buyer_id[20], selling_agent_id[20], buyers_agent_id[20];
    int sale_price;
    char sale_date[20];

    printf("\nEnter Sale ID(>=S0051): ");
    scanf("%s", sale_id);
    printf("Enter Property ID(P0001-P0060): ");
    scanf("%s", property_id);
    printf("Enter Buyer ID(BUY0001-BUY0010): ");
    scanf("%s", buyer_id);
    printf("Enter Selling Agent ID(AGT0001-AGT0010): ");
    scanf("%s", selling_agent_id);
    printf("Enter Buyer's Agent ID(AGT0001-AGT0010) or NULL if none: ");
    scanf("%s", buyers_agent_id);
    printf("Enter Sale Price: ");
    scanf("%d", &sale_price);
    printf("Enter Sale Date (YYYY-MM-DD): ");
    scanf("%s", sale_date);
    printf("\n");

    char query[1024];
    if (strcmp(buyers_agent_id, "NULL") == 0) {
        snprintf(query, sizeof(query), "INSERT INTO sale (sale_id, sale_price, sale_date, property_id, buyer_id, selling_agent_id, buyers_agent_id)\
            VALUES ('%s', %d, '%s', '%s', '%s', '%s', NULL)",
            sale_id, sale_price, sale_date, property_id, buyer_id, selling_agent_id);
    }
    else {
        snprintf(query, sizeof(query), "INSERT INTO sale (sale_id, sale_price, sale_date, property_id, buyer_id, selling_agent_id, buyers_agent_id)\
            VALUES ('%s', %d, '%s', '%s', '%s', '%s', '%s')",
            sale_id, sale_price, sale_date, property_id, buyer_id, selling_agent_id, buyers_agent_id);
    }

    int state = 0;
    state = mysql_query(connection, query);
    if (state == 0) {
        printf("Sale record added successfully.\n");
    }
    else {
        printf("Failed to add sale record: %s\n", mysql_error(connection));
    }
    printf("\n");

    char query2[100];
    snprintf(query2, sizeof(query), "SELECT * FROM sale WHERE sale_id = '%s'", sale_id);

    int state2 = 0;
    state = mysql_query(connection, query2);
    if (state2 == 0)
    {
        sql_result = mysql_store_result(connection);
        while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
        {
            printf("%s | %s | %s | %s | %s | %s\n", sql_row[0], sql_row[1], sql_row[2], sql_row[3], sql_row[4], sql_row[5], sql_row[6]);
        }
        mysql_free_result(sql_result);
    }
    printf("\n");
```

TYPE 6: 판매 완료된 부동산의 판매 정보를 기록한다. sale table에 들어갈 수 있도록 사용자로부터 판매 id, 판매 가격, 판매 날짜, property_id, buyer_id, 판매 에이전트 id, 구매자 에이전트 id를 각

각 입력 받아 쿼리에 포함될 수 있도록 하였다. 이때 데이터베이스 무결성을 유지할 수 있도록 각 value의 가능한 범위를 지정해주었다. buyer_agent_id의 경우 null 값 입력이 가능하기 때문에 경우의 수를 나눠서 쿼리를 처리했다. 다음으로 sale table에 사용자가 입력한 data를 insert하는 쿼리를 실행한 후 정상 동작하였는지 확인하기 위해 추가적으로 SELECT * FROM sale WHERE sale_id = '사용자가 입력한 sale_id' 쿼리를 실행한 결과를 출력한다.

```
case 7: {
    printf("---------- TYPE 7 ----------\n");
    printf("** Add a new agent to the database. **\n");

    char agent_id[20], name[20], phone_number[20], hire_date[20];

    printf("\nEnter Agent ID (>=AGT0011): ");
    scanf("%s", agent_id);
    getchar();

    printf("Enter Name: ");
    fgets(name, sizeof(name), stdin);
    name[strcspn(name, "\n")] = '\0';

    printf("Enter Phone Number: ");
    scanf("%s", phone_number);
    getchar();

    printf("Enter Hire Date (YYYY-MM-DD): ");
    scanf("%s", hire_date);
    getchar();

    printf("\n");

    char query[512];
    snprintf(query, sizeof(query), "INSERT INTO agent (agent_id, name, phone_number, hire_date) VALUES ('%s', '%s', '%s', '%s')",
        agent_id, name, phone_number, hire_date);

    int state = 0;
    state = mysql_query(connection, query);
    if (state == 0) {
        printf("Agent added successfully.\n");
    }
    else {
        printf("Failed to add agent: %s\n", mysql_error(connection));
    }
    printf("\n");


    char query2[100];
    snprintf(query2, sizeof(query), "SELECT * FROM agent WHERE agent_id = '%s'", agent_id);

    int state2 = 0;
    state = mysql_query(connection, query2);
    if (state2 == 0)
    {
        sql_result = mysql_store_result(connection);
        while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
        {
            printf("%s | %s | %s | %s\n", sql_row[0], sql_row[1], sql_row[2], sql_row[3]);
        }
        mysql_free_result(sql_result);
    }
    printf("\n");

    break;
}
default: {
    printf("Wrong choice.");
    continue;
}
```

TYPE 7: 새로운 에이전트를 데이터베이스에 추가한다. agent table에 들어갈 수 있도록 사용자로부터 에이전트 ID, 이름, 전화번호 및 고용 날짜를 각각 입력 받아 쿼리에 포함될 수 있도록 하였다. 이때 데이터베이스 무결성을 유지할 수 있도록 각 value의 가능한 범위를 지정해주었다. agent table에 사용자가 입력한 data를 insert하는 쿼리를 실행한 후 정상 동작하였는지 확인하기 위해

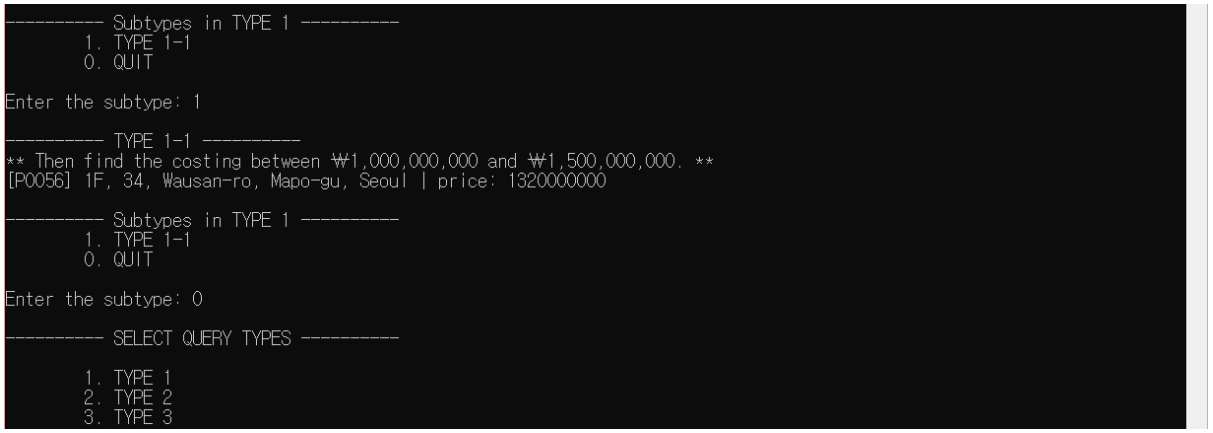
추가적으로 SELECT * FROM agent WHERE agent_id = '사용자가 입력한 agent_id' 쿼리를 실행한 결과를 출력한다.

2) 프로그램 실행 화면

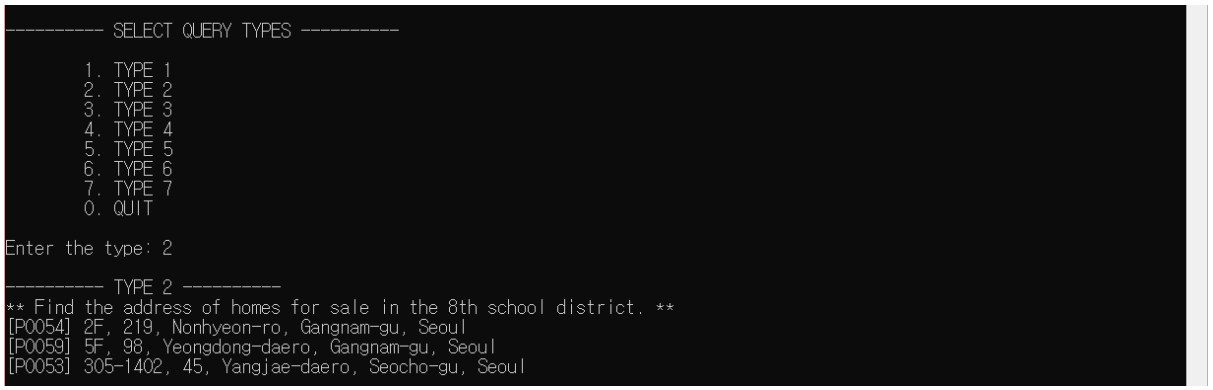
프로그램 실행 시 mysql server와 정상 연결되면 화면에 Connection Succeed를 출력한다. 그후 프로그램은 각 TYPE을 처리하기 위한 사용자 인터페이스를 출력한다. 메뉴에서 유형 중 하나를 선택한 후, 사용자는 쿼리의 값을 표준 입력(stdin)으로 입력하고, 프로그램은 쿼리 실행 결과를 표준 출력(stdout)으로 출력한다. 각 유형의 쿼리는 사용자가 0(quit)을 입력할 때까지 계속 실행되며, 쿼리 출력이 완료되거나 서브 쿼리 선택(TYPE 1, 2, 3, 4)을 종료한 후에는 메인 쿼리 선택 메뉴로 돌아간다. 만약 쿼리 메뉴에 없는 숫자를 입력하면 두 번째 사진과 같이 Wrong choice를 출력한 뒤 다시 쿼리 선택 메뉴를 출력한다.

위와 같이 사용자가 stdin에 1을 입력하면 type 1에 해당되는 쿼리와 결과 데이터를 출력한다.

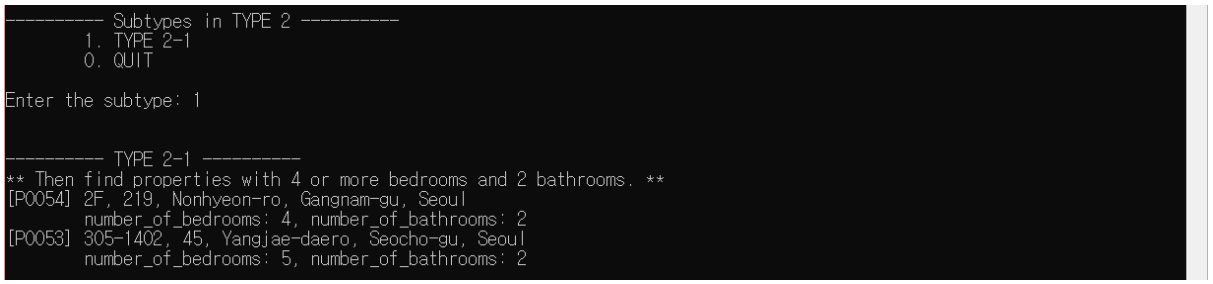
"Mapo" 지구에서 판매 중인 집의 주소를 출력하는 것을 확인할 수 있다. type 1의 경우 subquery가 존재하기에 쿼리 결과 출력 후 sub query type을 입력하라는 문구가 나온다.



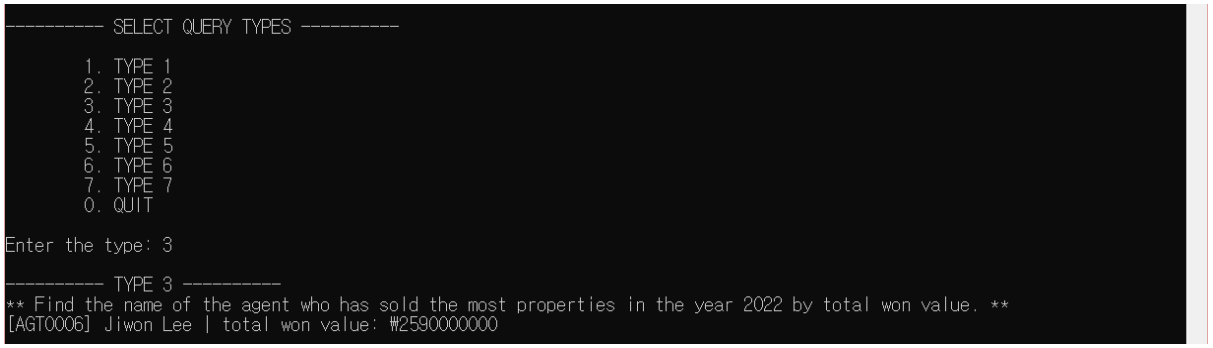
sub type으로 1을 입력하면 type 1-1에 해당되는 쿼리와 결과 데이터를 출력한다. "Mapo" 지구에서 판매 중인 집 중 가격이 1,000,000,000원에서 1,500,000,000원 사이인 집의 주소와 가격을 출력하는 것을 확인할 수 있다. 이어서 sub type으로 0을 입력하면 서브 쿼리 선택 메뉴 출력을 종료하고 메인 쿼리 선택 메뉴가 출력된다.



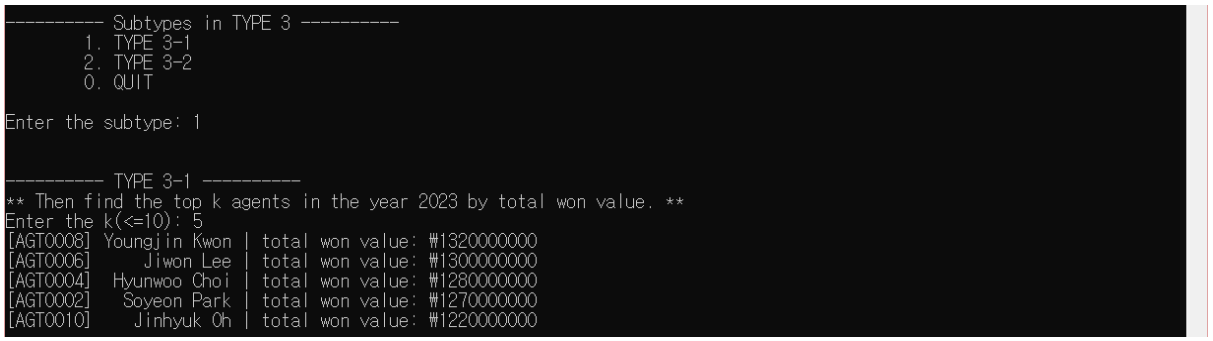
type으로 2를 입력하면 type 2에 해당되는 쿼리와 결과 데이터를 출력한다. 8학군에서 판매 중인 집의 주소를 출력하는 것을 확인할 수 있다.



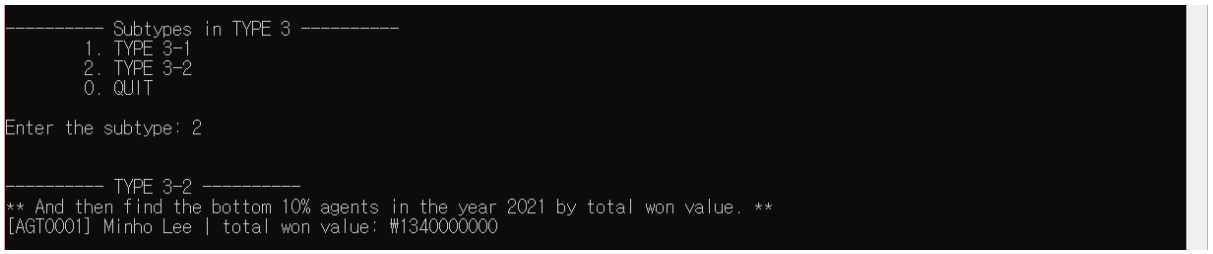
마찬가지로 type 2도 subquery가 존재하기에 쿼리 결과 출력 후 sub query type을 입력하라는 문구가 나온다. sub type으로 1을 입력하면 type 2-1에 해당되는 쿼리와 결과 데이터를 출력한다. 8학군에서 판매 중인 집 중 침실이 4개 이상이고 욕실이 2개인 집의 주소와 침실 수, 욕실 수를 출력하는 것을 확인할 수 있다.



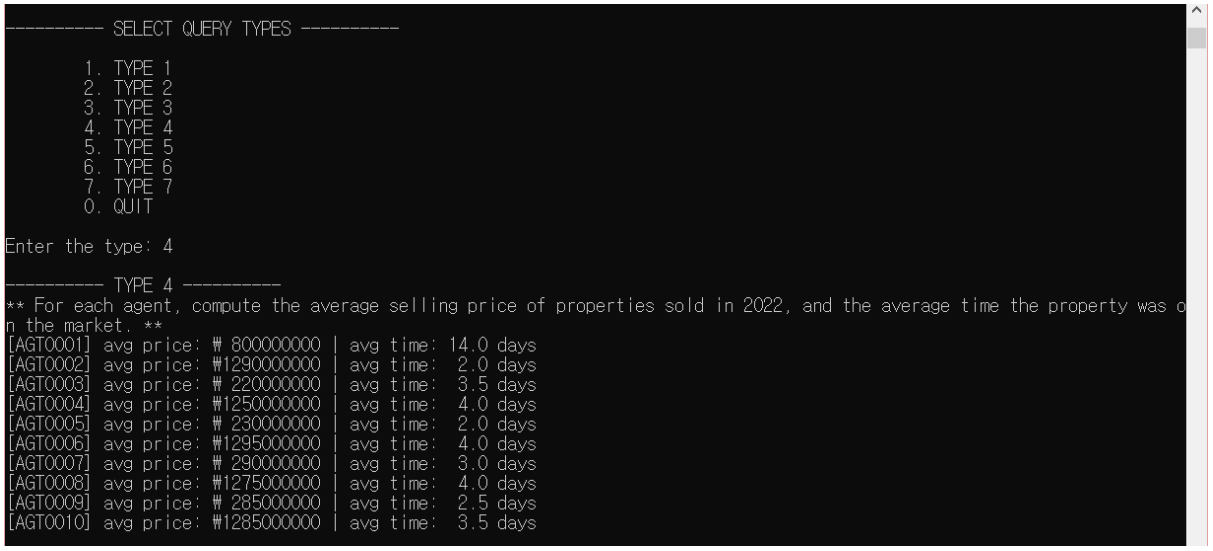
type으로 3을 입력하면 type 3에 해당되는 쿼리와 결과 데이터를 출력한다. 2022년에 가장 많은 부동산을 판매한 에이전트의 이름과 함께 ID, 총 판매 금액을 출력하는 것을 확인할 수 있다.



sub type으로 1을 입력하면 type 3-1에 해당되는 쿼리를 출력한다. 그후 추가적으로 k값을 입력받고 2023년에 총 판매 금액 기준으로 상위 k명의 에이전트의 ID, 이름, 총 판매 금액을 출력한다. 위와 같이 k값으로 5를 입력하면 상위 5명 에이전트의 정보를 출력하는 것을 확인할 수 있다.

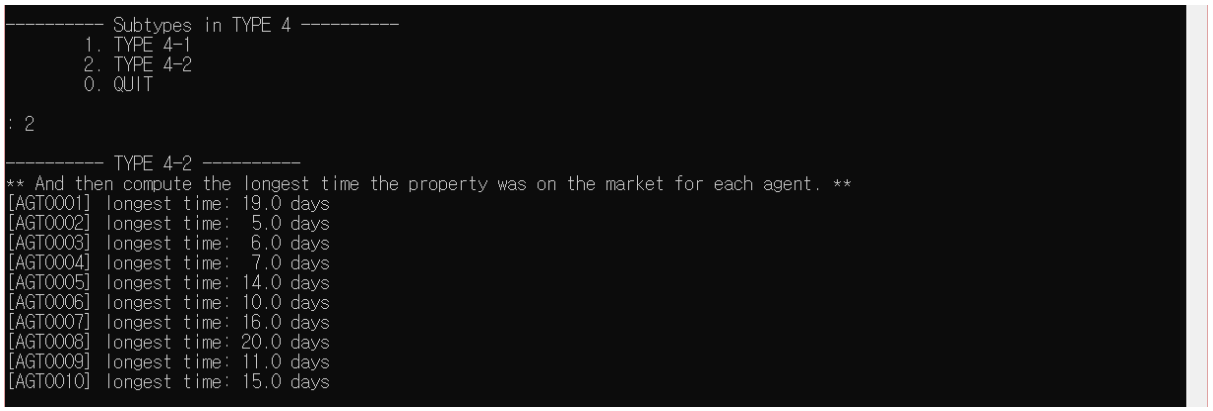


sub type으로 2를 입력하면 type 3-2에 해당되는 쿼리를 출력한다. 2021년에 총 판매 금액 기준으로 하위 10%의 에이전트의 ID, 이름, 총 판매 금액을 출력하는 것을 확인할 수 있다. 이때 데이터베이스 초기화 시 10명의 에이전트를 생성했기에 1명만 출력된다.



type으로 4를 입력하면 type 4에 해당되는 쿼리와 결과 데이터를 출력한다. 각 에이전트에 대해 2022년에 판매된 부동산의 평균 판매 가격과 부동산이 시장에 나와 있던 평균 시간을 출력하는 것을 확인할 수 있다.

sub type으로 1을 입력하면 type 4-1에 해당되는 쿼리를 출력한다. 2023년에 판매된 부동산의 최대 판매 가격을 각 에이전트별로 계산한 결과를 출력하는 것을 확인할 수 있다.

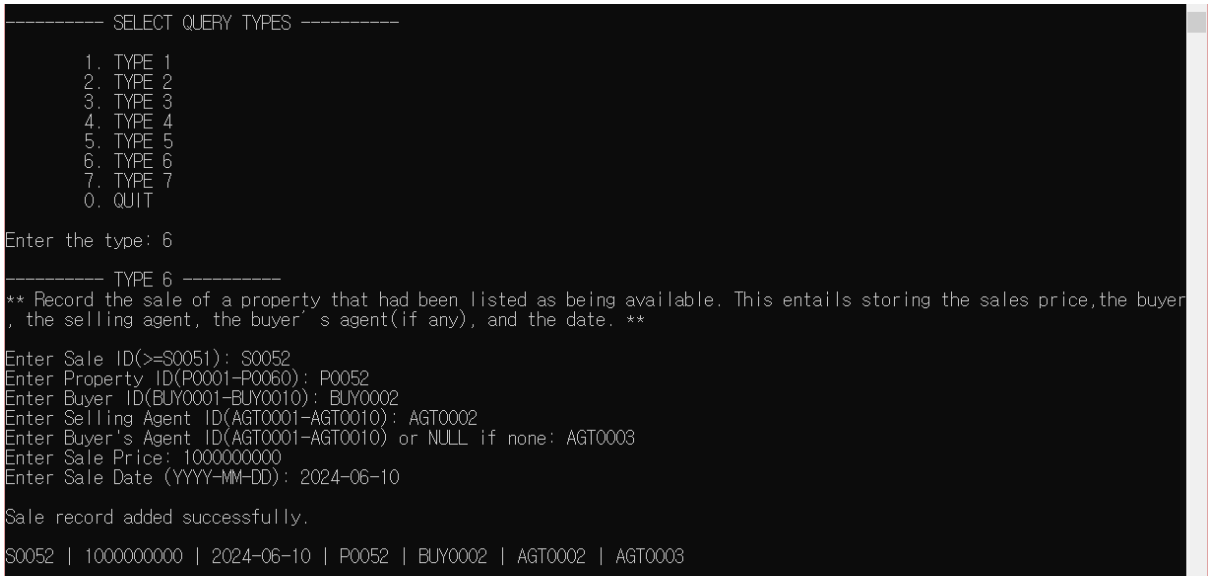


sub type으로 2를 입력하면 type 4-2에 해당되는 쿼리를 출력한다. 각 에이전트 별로 부동산이 시장에 나와 있던 가장 긴 시간을 계산한 결과를 출력하는 것을 확인할 수 있다.

```
---------- 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 the type: 5

---------- TYPE 5 ----------
** Show photos of the most expensive studio, one-bedroom, multi-bedroom apartment(s), and detached house(s), respectivel
y, from the database. **
[studio] interior file path: C:₩Users₩user₩Pictures₩photo8.jpg (P0005, price: 350000000)
[one_bedroom] interior file path: C:₩Users₩user₩Pictures₩photo15.jpg (P0010, price: 1400000000)
[multi_bedroom] interior file path: C:₩Users₩user₩Pictures₩photo41.jpg (P0026, price: 1300000000)
[multi_bedroom] exterior file path: C:₩Users₩user₩Pictures₩photo42.jpg (P0026, price: 1300000000)
[multi_bedroom] floor_plan file path: C:₩Users₩user₩Pictures₩photo43.jpg (P0026, price: 1300000000)
[detached_house] interior file path: C:₩Users₩user₩Pictures₩photo69.jpg (P0040, price: 1350000000)
[detached_house] exterior file path: C:₩Users₩user₩Pictures₩photo70.jpg (P0040, price: 1350000000)
[detached_house] floor_plan file path: C:₩Users₩user₩Pictures₩photo71.jpg (P0040, price: 1350000000)
```

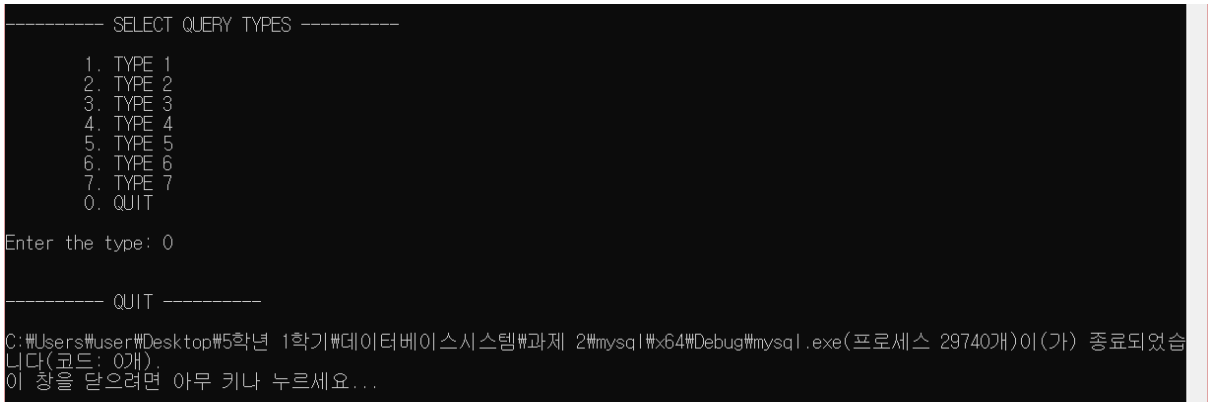
type으로 5를 입력하면 type 5에 해당되는 쿼리와 결과 데이터를 출력한다. 데이터베이스에서 가장 비싼 스튜디오, 원룸, 다중 침실 아파트, 단독 주택의 사진이 저장된 경로와 관련 정보(property type, photo type, property id, price)를 출력하는 것을 확인할 수 있다.

```
---------- 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 the type: 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. **

Enter Sale ID(>=S0051): S0051
Enter Property ID(P0001-P0060): P0051
Enter Buyer ID(BUY0001-BUY0010): BUY0001
Enter Selling Agent ID(AGT0001-AGT0010): AGT0001
Enter Buyer's Agent ID(AGT0001-AGT0010) or NULL if none: NULL
Enter Sale Price: 1000000000
Enter Sale Date (YYYY-MM-DD): 2024-06-10

Sale record added successfully.

S0051 | 1000000000 | 2024-06-10 | P0051 | BUY0001 | AGT0001 | (null)
```



type으로 6을 입력하면 우선 type 6에 해당되는 쿼리를 출력한다. 다음으로 property로 등록된 부동산의 판매를 기록할 수 있도록 사용자로부터 판매 ID, 판매 가격, 판매 날짜, property ID, buyer ID, 판매 에이전트 ID, 구매자 에이전트 ID(존재할 경우)를 각각 입력 받는다. 입력 받은 정보를 토대로 sale table에 값이 정상적으로 insert 되면 Sale record added successfully를 출력한다. 추가적으로 SELECT * FROM sale WHERE sale_id = '사용자가 입력한 sale_id' 쿼리를 실행한 결과를 출력하여 데이터가 잘 저장되었음을 알려준다.

type으로 7을 입력하면 우선 type 7에 해당되는 쿼리를 출력한다. 다음으로 데이터베이스에 새로운 에이전트를 추가할 수 있도록 사용자로부터 에이전트 ID, 이름, 전화번호, 고용 날짜를 각각 입력 받는다. 입력 받은 정보를 토대로 agent table에 값이 정상적으로 insert 되면 Agent added successfully를 출력한다. 추가적으로 SELECT * FROM agent WHERE agent_id = '사용자가 입력한 agent_id' 쿼리를 실행한 결과를 출력하여 데이터가 잘 저장되었음을 알려준다.



마지막으로 type으로 0을 입력하면 데이터베이스에 생성했던 모든 table과 데이터베이스를 drop 한 후 mysql server와 연결을 종료한 후 프로그램을 종료한다.