Database System Project 2

담당 교수 : 정성원

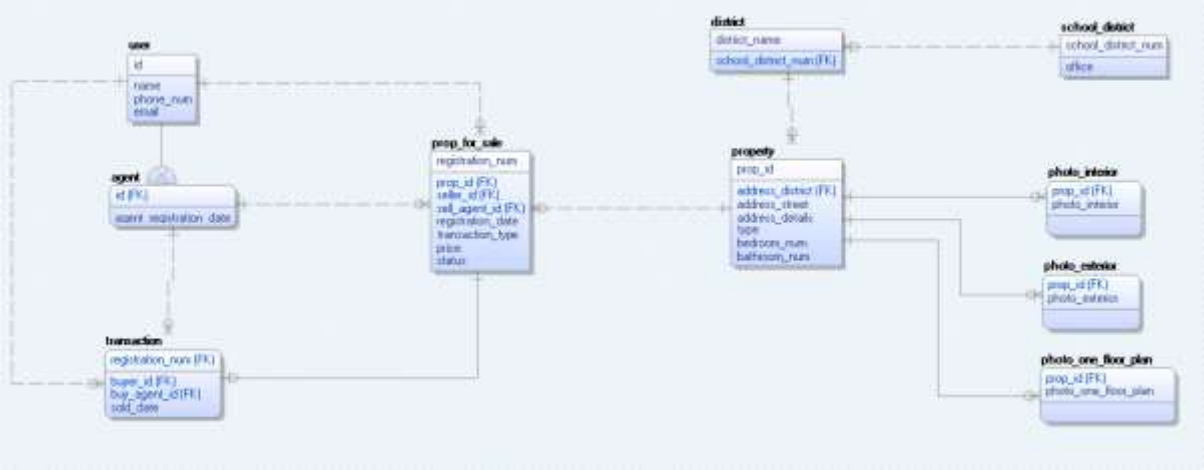
이름 : 김태규

학번 : 20191243

1. **Logical Schema Diagram**

A. BCNF Decomposition

1. Former Logical Schema Diagram



이전의 Logical Schema Diagram은 위와 같다.

2. Decomposition

✓ **User**

Before	User(ID, name, phone_num, e-mail)
After	User(ID, name, phone_num, e-mail) (동일)

F set:

ID -> name, phone_num, e-mail

phone_num -> ID, name, e-mail

e-mail -> ID, name, phone num

모든 function dependency에서 left side가 super key이므로 trivial하게 BCNF를 만족한다.

✓ **Agent**

Before	Agent(ID, registration_date)
After	Agent(ID, registration_date) (동일)

F set:

ID -> registration_date

ID가 Agent table의 super key 역할을 하므로 BCNF를 만족한다.

✓ **Property**

Before	Property(prop_id, address_district, address_street, address_details, type, bedroom_num, bathroom_num)
After	Property(prop_id, address_district, address_details, type, bedroom_num, bathroom_num) (동일)

F set:

prop_id -> address_district, address_street, address_details, type, bedroom_num, bathroom_num

address_district, address_street, address_details -> prop_id, type, bedroom_num, bathroom_num

F set의 모든 functional dependency에서 left side가 super key 역할을 하므로 BCNF condition을 만족한다.

✓ **District**

Before	District(district_name, school_district_num)
After	District(district_name, school_district_num) (동일)

F set:

district_name -> school_district_num

district_name이 primary key이므로 trivial하게 BCNF 조건을 만족한다.

✓ **School_district**

Before	School_district(school_district_num, office)
After	School_district(school_district_num, office) (동일)

F set:

school_district_num -> office

office -> school_district_num

school_district_num과 office 모두 school_district table에서 super key 역할을 하므로 BCNF 조건을 만족한다.

✓ **photo_interior / exterior / one_floor_plan**

Before	photo_* (prop_id, photo_*)
After	photo_* (prop_id, photo_*) (동일)

F set:

prop_id -> photo_interior/exterior/one_floor_plan

photo_interior/exterior/one_floor_plan -> prop_id

F set의 모든 functional dependency에 대해 left side가 super key 역할을 하므로 BCNF를 만족한다.

interior, exterior, one_floor_plan 셋 다 같은 형식이므로 한번에 작성하였다.

✓ **prop_for_sale**

Before	prop_for_sale(registration_num, prop_id, seller_id, sell_agent_id,

	registration_date, transaction_type, price, status)
After	prop_for_sale(registration_num, prop_id, seller_id, sell_agent_id, registration_date, transaction_type, price, status) (동일)

F set:

registration_num -> prop_id, seller_id, sell_agent_id, registration_date, transaction_type, price, status

같은 날에 같은 부동산, 판매자, 판매 대리인, 거래유형이 동일한 매물이 나올 수 있으므로 functional dependency는 위의 하나이다.

registration_num은 해당 table의 primary key이므로 자명하게 BCNF를 만족한다.

✓ **transaction**

Before	transaction(registration_num, buyer_id, buy_agent_id, sold_date)
After	transaction(registration_num, buyer_id, buy_agent_id, sold_date) (동일)

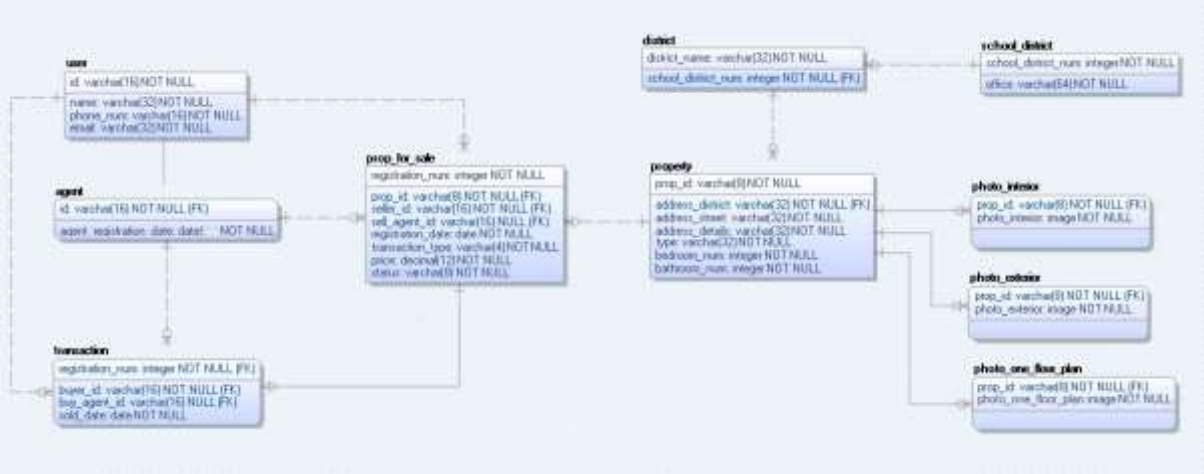
F set:

registration_num -> buyer_id, buy_agent_id, sold_date

transaction과 prop_for_sale entity는 one to one relation을 가지므로 registration이 primary key 역할을 한다. 그러므로 해당 entity는 BCNF를 만족한다.

위의 BCNF decomposition 과정에서 수정된 사항이 없다. 따라서 Logical Schema Diagram은 위의 diagram과 일치한다.

2. **Physical Schema Diagram**



1) user

모든 속성은 varchar type을 가지고 NULL을 허용하지 않는다. Phone_num attribute을 varchar type으로 지정한 이유는 전화번호에 '-'이 포함되어 있어 이를 감안해 numeric이나 integer type을 사용하지 않았다.

2) agent

id는 user table에서 갖고 온 foreign key이나 primary key로 user에서의 특성을 그대로 가져와 varchar type이다.

agent_registration_date는 부동산 매매 대리인으로 certify된 날짜를 의미하며 date type을 가진다.

모든 속성은 NULL을 허용하지 않는다.

3) property

prop_id, address_district/street/detail, type은 varchar type을 가진다. 각각 부동산 고유번호, 주소(구/도로명/상세주소), 집 유형(studio, one_bedroom, multi bedroom, detached house)를 의미한다.

bedroom/bathroom num은 침실/화장실 개수를 의미하며 integer type을 가진다.

모든 속성은 NULL을 허용하지 않는다.

4) district

district name은 구의 이름, school district num은 학군 number를 의미한다. 각각 varchar, integer type을 가진다.

두 속성 모두 NULL을 허용하지 않는다.

5) school_district

school_district_num은 학군 num을, office는 각 학군의 담당 교육청 이름을 의미한다. 각각 integer, varchar type을 지원한다.

두 속성 모두 NULL을 허용하지 않는다.

6) photo_interior/exterior/one_floor_plan

prop_id는 property에서 가져온 foreign key이므로 property table의 prop_id의 특성을 그대로 상속받아 varchar type을 가진다.

photo_interior/exterior/one_floor_plan은 각 부동산의 내부/외부/단면도 사진을 의미하며 physical diagram에서는 image type을 가지도록 했지만 실제 ODBC에서는 VARCHAR type으로 구현하였다.

두 속성 모두 NOT NULL이다.

7) prop_for_sale

registration_num은 매물 등록 번호를 의미한다. 2024-1과 같은 형식을 가지므로 varchar type을 가진다.

prop_id, seller_id, sell_agent_id는 각각 property/user/agent table에서 가져온 foreign key이므로 각 attribute의 특성을 그대로 유지하므로 varchar type을 가진다.

registration_date는 매물 등록 날짜를 의미하며 date type을 가진다.

transaction_type은 거래 유형을 의미하며 sale(매매), rent(임대) 중 하나의 값을 가지도록 하였다. varchar type을 가진다.

price는 decimal(12) type을 가지도록 하였는데 integer는 upper bound가 약 21억이므로 decimal(12)로 설정하였다.

status는 현재 상태를 의미하고 sold, for_sale 중 하나의 값을 가지도록 하였다. varchar type을 가진다.

sell_agent_id을 제외한 모든 attribute이 NULL을 허용하지 않는다. 주택 거래에서 agent가 필수적인 것은 아니므로 sell_agent_id attribute는 NULL을 허용하도록 하였다.

8) transaction

registration_num, buyer_id, buy_agent_id는 각각 prop_for_sale, user, agent table에서 가져온 foreign key이므로 각 특성을 그대로 유지하여 각각 integer, varchar, varchar type을 가진다.

sold_date는 매물이 거래된 날짜를 의미하며 date type을 가진다.

이 table 또한 buy_agent_id는 필수가 아니므로 이를 제외한 모든 attribute가 NULL을 허용하지 않는다.

3. **ODBC Implementation**

1) MySQL, ODBC Setting

프로젝트 공지사항 내용을 따라 workbench와 ODBC 를 설치하고, Visual Studio 2019 와 연동시켰다.

2) Implementation Detail

```
const char* host = "localhost"; // change if necessary
const char* user = "root";      // change if necessary
const char* pw = "taekyu073!";  // change if necessary
const char* db = "project";  // change if necessary

#define MAX_LEN 13000

void Query(MYSQL* connection, char* query);
void type1(MYSQL* connection);
void type1_1(MYSQL* connection);
void type2(MYSQL* connection);
void type2_1(MYSQL* connection);
void type3(MYSQL* connection);
void type3_1(MYSQL* connection);
void type3_2(MYSQL* connection);
void type4(MYSQL* connection);
void type4_1(MYSQL* connection);
void type4_2(MYSQL* connection);
void type5(MYSQL* connection);
void type6(MYSQL* connection);
void type7(MYSQL* connection);
void Choose_type(MYSQL* connection);
```

- MAX_LEN은 CRUD File 인 CRUD.txt 파일을 파일을 open 후 read를 할 때 사용하는 Read Line의 최대 길이이다.

- host, user, pw, db는 MySQL Connection 시 사용하는 파라미터이다. CRUD.txt 에서 CREATE와 USE 명령어로 Schema를 만든다.

3) main 함수

```
int main(void) {
    MYSQL* connection = NULL;
    MYSQL conn;
    FILE* fp = fopen("CRUD.txt", "rt");     // open CRUD file.
    char line[MAX_LEN];
    char query[MAX_LEN * 10] = "";          // Multi-line queries

    if (mysql_init(&conn) == NULL) {
        printf("mysql_init() error!");
        return 1;
    }
```

```
connection = mysql_real_connect(&conn, host, user, pw, NULL, 3306, 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) {
        // Trim newline character from the end of the line
        line[strcspn(line, "₩r₩n")] = 0;
        if (strcmp(line, "$$$") == 0) { // Check for $$$ delimiter
            break;
        }
        if (strlen(line) > 0){
            if (strncmp(line, "--", 2) != 0) {
                strcat(query, line); // Append line to query
              if (line[strlen(line) - 1] == ';') { // If line ends with semicolon(;)
                    printf("Query: %s₩n", query);
                    if (mysql_query(connection, query)) {
                        return 1;
                    }
                    query[0] = '₩0'; // query buffer intialize
                }
            }
        }
    }

   if (mysql_query(connection, "USE project;")) {          // use databasae project
        printf("%d ERROR : %s₩n", mysql_errno(&conn), mysql_error(&conn));
        return 1;
    }

    Choose_type(connection);

    // DELETE & DROP
    query[0] = '₩0'; // query buffer initialize
    while (fgets(line, sizeof(line), fp) != NULL) {
        // Trim newline character from the end of the line
        line[strcspn(line, "₩r₩n")] = 0;
        if (strlen(line) > 0) {
            if (strncmp(line, "--", 2) != 0) {
                strcat(query, line);
              if (line[strlen(line) - 1] == ';') { // If line ends with semicolon(;)
                    printf("Query: %s₩n", query);
                    if (mysql_query(connection, query)) {
                        return 1;
                    }
                    query[0] = '₩0'; // query buffer initialize
                }
            }
        }
    }

    mysql_close(connection);
```

```
}

fclose(fp);
return 0;
    }
```

예시 코드에서 수정하여 ODBC 코드를 작성하였다. 주석으로 설명을 대체한다.

4) type 1

type 1에서 사용한 쿼리는 다음과 같다.

```
SELECT p.address_district, p.address_street, p.address_details
FROM property p
JOIN prop_for_sale ps ON p.prop_id = ps.prop_id
WHERE p.address_district = 'Mapo-Gu' AND ps.status = 'for_sale'
```

property에서 district가 mapo-gu이고, status가 for sale인 property의 주소를 모두 출력하는 쿼리이다.

- type 1-1

```
SELECT p.address_district, p.address_street, p.address_details, ps.price
FROM property p
JOIN prop_for_sale ps ON p.prop_id = ps.prop_id
WHERE p.address_district = 'Mapo-Gu'
AND ps.price BETWEEN 1000000000 AND 1500000000
AND ps.status = 'for_sale'
```

type 1에서 가격 조건이 붙어서 prop_for_sale과 join한 뒤 조건에 맞게 가격을 설정해주고 주소 말고도 가격도 출력하도록 하였다.

5) type 2

```
SELECT p.address_district, p.address_street, p.address_details
FROM property p
JOIN district d ON p.address_district = d.district_name
JOIN prop_for_sale ps ON p.prop_id = ps.prop_id
WHERE d.school_district_num = 8
AND ps.status = 'for_sale'
```

property에서 district와 prop_for_sale을 join하여 8학군이면서 for sale인 prop의 주소를 출력하도록 하는 쿼리이다.

- type 2-1

```
SELECT p.address_district, p.address_street, p.address_details, p.bedroom_num,
p.bathroom_num
FROM property p
JOIN district d ON p.address_district = d.district_name
JOIN prop_for_sale ps ON p.prop_id = ps.prop_id
WHERE d.school_district_num = 8
AND p.bedroom_num >= 4
AND p.bathroom_num >= 2
AND ps.status = 'for_sale'
```

type 2에서 침실과 화장실이 각각 4개, 2개 이상인 부동산을 출력하도록 하는 부분이 추가된 쿼리이다.

6) type 3

```
SELECT u.name, SUM(ps.price) AS total_value
FROM transaction t
JOIN prop_for_sale ps ON t.registration_num = ps.registration_num
JOIN agent a ON ps.sell_agent_id = a.id
JOIN user u ON a.id = u.id
WHERE YEAR(t.sold_date) = 2022
GROUP BY u.name
ORDER BY total_value DESC LIMIT 1
```

group by를 통해서 각 agent가 판매한 부동산들을 묶고 order by와 DESC LIMINT 1을 통해 2022년에 가장 많이 판매한 agent를 출력하도록 하였다.

- type 3-1

```
SELECT u.name, SUM(ps.price) AS total_value
FROM transaction t
JOIN prop_for_sale ps ON t.registration_num = ps.registration_num
JOIN agent a ON ps.sell_agent_id = a.id
JOIN user u ON a.id = u.id
WHERE YEAR(t.sold_date) = 2023
GROUP BY u.name
ORDER BY total_value DESC LIMIT %d(k)
```

type 3와 비슷하지만 top 1이 아니라 top k이므로 k를 입력 받고 %d를 통해 k를 유동적으로 조절할 수 있도록 하였다.

- type 3-2

```
WITH ranked_agents AS (
      SELECT u.name, SUM(ps.price) AS total_value, NTILE(10) OVER(ORDER BY
SUM(ps.price)) AS decile
      FROM transaction t
      JOIN prop_for_sale ps ON t.registration_num = ps.registration_num
      LEFT JOIN agent a ON ps.sell_agent_id = a.id
      JOIN user u ON a.id = u.id
      WHERE YEAR(t.sold_date) = 2021
      GROUP BY u.name )
SELECT name, total_value
FROM ranked_agents
WHERE decile = 1
```

하위 10% agent를 찾기 위해서 ranked agents라는 temporary table을 만들고 NTILE을 통해 10묶음으로 table을 나눈 후 decile을 1로 설정함으로써 하위 10%를 찾는 쿼리가 잘 수행되도록 하였다.

7) type 4

```
SELECT u.name, AVG(ps.price) AS avg_selling_price, AVG(DATEDIFF(t.sold_date,
ps.registration_date)) AS avg_time_on_market
FROM transaction t
JOIN prop_for_sale ps ON t.registration_num = ps.registration_num
JOIN agent a ON ps.sell_agent_id = a.id
JOIN user u ON a.id = u.id
WHERE YEAR(t.sold_date) = 2022
GROUP BY u.name
```

2022년에 판매한 agent들의 average selling price와 average time을 구하기 위해 AVG 함수를 사용하였고 group by를 통해 agent를 기준으로 판매한 부동산들을 분류하여 쿼리를 구현하였다.

- type 4-1

```
SELECT u.name, MAX(ps.price) AS max_selling_price
FROM transaction t
JOIN prop_for_sale ps ON t.registration_num = ps.registration_num
JOIN agent a ON ps.sell_agent_id = a.id
JOIN user u ON a.id = u.id WHERE YEAR(t.sold_date) = 2023
GROUP BY u.name
```

type 4와 비슷하지만 AVG가 아닌 MAX 함수를 사용하여 maximum selling price를 찾을 수 있도록 구현하였다.

- type 4-2

```
SELECT u.name, MAX(DATEDIFF(t.sold_date, ps.registration_date)) AS
longest_time_on_market
FROM transaction t
JOIN prop_for_sale ps ON t.registration_num = ps.registration_num
JOIN agent a ON ps.sell_agent_id = a.id
JOIN user u ON a.id = u.id
GROUP BY u.name
```

이 또한 AVG가 아닌 MAX를 통해 longest time을 구할 수 있도록 하였다.

8) type 5

부동산의 유형마다 쿼리를 하나씩 수행하였다. 네 쿼리 모두 flow는 동일하다.

```
--- Most expensive studio ---
SELECT p.address_district, p.address_street, p.address_details,
pi.photo_interior, pe.photo_exterior, pf.photo_one_floor_plan
FROM property p
JOIN prop_for_sale ps ON p.prop_id = ps.prop_id
LEFT JOIN photo_interior pi ON p.prop_id = pi.prop_id
LEFT JOIN photo_exterior pe ON p.prop_id = pe.prop_id
LEFT JOIN photo_one_floor_plan pf ON p.prop_id = pf.prop_id
WHERE p.type = 'studio'
ORDER BY ps.price DESC LIMIT 1
```

```
--- Most expensive one-bedroom ---
SELECT p.address_district, p.address_street, p.address_details,
pi.photo_interior, pe.photo_exterior, pf.photo_one_floor_plan
FROM property p
JOIN prop_for_sale ps ON p.prop_id = ps.prop_id
LEFT JOIN photo_interior pi ON p.prop_id = pi.prop_id
LEFT JOIN photo_exterior pe ON p.prop_id = pe.prop_id
LEFT JOIN photo_one_floor_plan pf ON p.prop_id = pf.prop_id
WHERE p.type = 'apt_one_bedroom'
ORDER BY ps.price DESC LIMIT 1
```

```
--- Most expensive multi-bedroom ---
```

```
SELECT p.address_district, p.address_street, p.address_details,
pi.photo_interior, pe.photo_exterior, pf.photo_one_floor_plan
FROM property p JOIN prop_for_sale ps ON p.prop_id = ps.prop_id
LEFT JOIN photo_interior pi ON p.prop_id = pi.prop_id
LEFT JOIN photo_exterior pe ON p.prop_id = pe.prop_id
LEFT JOIN photo_one_floor_plan pf ON p.prop_id = pf.prop_id
WHERE p.type = 'apt_multi_bedroom'
ORDER BY ps.price DESC LIMIT 1
```

```
--- Most expensive detached-house ---
SELECT p.address_district, p.address_street, p.address_details,
pi.photo_interior, pe.photo_exterior, pf.photo_one_floor_plan
FROM property p
JOIN prop_for_sale ps ON p.prop_id = ps.prop_id
LEFT JOIN photo_interior pi ON p.prop_id = pi.prop_id
LEFT JOIN photo_exterior pe ON p.prop_id = pe.prop_id
LEFT JOIN photo_one_floor_plan pf ON p.prop_id = pf.prop_id
WHERE p.type = 'detached_house'
ORDER BY ps.price DESC LIMIT 1
```

모든 photo에 관련된 table을 property와 property_for_sale과 join하여 order by를 통해 가장 비싼 부동산을 찾고 해당 부동산의 주소와 사진들을 반환하도록 ㅎ였다.

9) type 6

```
void type6(MYSQL* connection) {
    printf("\n**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");

    printf("\n*** Properties that had been listed as being available ***\n");
    Query(connection,
"SELECT ps.registration_num, p.prop_id, p.address_district, p.address_street,
p.address_details
FROM property p
JOIN prop_for_sale ps ON p.prop_id = ps.prop_id
WHERE ps.status = 'for_sale'
ORDER BY p.prop_id ASC");
```

먼저 현재 부동산 중 for_sale 상태인 부동산을 출력하도록 하는 쿼리를 먼저 수행하였다.

```
    char registration_num[9], buyer_id[17], buy_agent_id[17], sale_date[11];

    printf("\nEnter registration num, buyer id, buy agent id, sale date:\n");
    scanf("%s %s %s %s", registration_num, buyer_id, buy_agent_id, sale_date);
```

그 후, registration num, buyer id, buy agent id, sale date를 입력 받았다.

```
    char query[MAX_LEN];
    snprintf(query, sizeof(query),
"INSERT INTO transaction (registration_num, buyer_id, buy_agent_id, sold_date)
VALUES('%s', '%s', '%s', '%s')",
registration_num, buyer_id, buy_agent_id, sale_date);
    Query(connection, query);

    snprintf(query, sizeof(query),
"UPDATE prop_for_sale
SET status = 'sold'
WHERE registration_num = %s",
registration_num);
    Query(connection, query);

    printf("\n\t\t*** Property for sale List ***\n");
    Query(connection,
"SELECT *
FROM prop_for_sale
WHERE status = 'for_sale'");
}
```

마지막으로 입력 받은 registration num, buyer id, buy agent id, sold date를 transaction table에 INSERT하는 쿼리를 수행하고, prop_for_sale table에서 이에 해당하는 매물에 대해 status를 'sold'로 UPDATE하는 쿼리를 수행하였다.

그리고 나서 현재 남아 있는 매물 명단을 출력하도록 하였다.

10) type7

```
    printf("Enter new agent_id, name, phone num, and email (No space-bar in
name!): \n");
    scanf("%s %s %s %s", new_user_id, new_name, phone_num, email);
```

먼저 새롭게 추가하고자 하는 agent의 id, 이름, 전화번호, email을 입력 받았다.

```
    char query[MAX_LEN];
    snprintf(query, sizeof(query),
"INSERT INTO user (id, name, phone_num, email)
VALUES('%s', '%s', '%s', '%s')",
new_user_id, new_name, phone_num, email);
    Query(connection, query);

    snprintf(query, sizeof(query),
"INSERT INTO agent (id, registration_date)
VALUES('%s', '%s')",
new_user_id, today_date);
    Query(connection, query);

    printf("\n\t\t*** Agent List ***\n");
    Query(connection,
"SELECT *
FROM agent a
JOIN user u ON a.id = u.id
ORDER BY a.registration_date ASC");
```

그 후, user table에 입력 받은 정보를 토대로 새로운 user를 먼저 INSERT하고 나서 agent table에도 INSERT해주었다.

마지막으로 현재 등록되어 있는 agent의 list를 출력하도록 하였다.

```
    time_t t = time(NULL);
    struct tm tm = *localtime(&t);

    if (tm.tm_mon < 9) {
        if (tm.tm_mday < 10) {
            snprintf(today_date, sizeof(today_date), "%d-0%d-0%d", 1900 +
tm.tm_year, 1 + tm.tm_mon, tm.tm_mday);
        }
        else {
            snprintf(today_date, sizeof(today_date), "%d-0%d-%d", 1900 +
tm.tm_year, 1 + tm.tm_mon, tm.tm_mday);
        }
    }
    else {
        if (tm.tm_mday < 10) {
            snprintf(today_date, sizeof(today_date), "%d-%d-0%d", 1900 +
tm.tm_year, 1 + tm.tm_mon, tm.tm_mday);
        }
        else {
            snprintf(today_date, sizeof(today_date), "%d-%d-%d", 1900 +
tm.tm_year, 1 + tm.tm_mon, tm.tm_mday);
```

```
        }
    }
```

위의 코드는 오늘 날짜를 DATE type 형식에 맞게 today_date라는 char* 변수에 문자열로 추가해주는 코드이다.