Project 2
Report

이름: 이예준
학번: 20212022

1. BCNF Decomposition

-특정 Schema 가 BCNF 를 만족시키기 위해서는 해당 Schema 의 모든
Functional Dependency α → β 에 대하여
1. α 가 superkey 이다.
2. trivial 한 α → β 가 존재한다.
를 확인한다.

만약 그런 α → β가 존재하면 해당 Schema는 BCNF를 만족하지만,
존재하지 않는다면 해당 Schema는 BCNF를 만족하기 위한 Decomposition이 필요하다.
기존 Schema가 R이고, R이 BCNF를 위반하게 만드는 Functional Dependency가
α → β일 때, R을 **(α ∪ β)**와 **(R – (β – α))**로 분해한 뒤 BCNF를 만족하는지 확인한다.
이 과정은 BCNF를 만족할 때까지 반복한다.

1. Agent

- Attributes: Agent_id (PK), Name, Address, Email, Phone_num
- FDs:
  - Agent_id → Name, Address, Email, Phone_num

* Agent_id가 primary key이므로 BCNF를 만족한다.

2. Agent_Market

- Attributes: Market_id (FK), Agent_id (FK)
- FDs:
  - Market_id, Agent_id → (Market_id, Agent_id)

* Market_id, Agent_id가 composite priamry key이고, trivial하므로 BCNF를 만족한다.

3. Agent_Sold

- Attributes: Agent_id (FK), Sold_id (FK), Buy, Sell
- FDs:
  - Agent_id, Sold_id → Buy, Sell

* Agent_id, Sold_id가 composite priamry key이므로 BCNF를 만족한다.

4. Buyer

- Attributes: Buyer_id (PK), Name, Address, Email, Phone_num
- FDs:
  - Buyer_id → Name, Address, Email, Phone_num

* Buyer_id가 primary key이므로 BCNF를 만족한다.

5. Buyer_Sold

- Attributes: Sold_id (FK), Buyer_id (FK)
- FDs:
  - Sold_id, Buyer_id → (Sold_id, Buyer_id)

* Sold_id, Buyer_id가 composite priamry key이므로 BCNF를 만족한다.

6. Market

- Attributes: Market_id (PK), Price, Name, Address, Enroll_date, School_district, Type
- FDs:
  - Market_id → Price, Name, Address, Enroll_date, School_district, Type

* Market_id가 primary key이므로 BCNF를 만족한다.

7. Market_Photo

- Attributes: Market_id (FK), Photo_id (FK)
- FDs:
  - Market_id, Photo_id → (Market_id, Photo_id)

* Market_id, Photo_id가 composite priamry key이므로 BCNF를 만족한다.

8. Market_Room

- Attributes: Market_id (FK), Room_id (FK)
- FDs:
  - Market_id, Room_id → (Market_id, Room_id)

* Market_id, Room_id가 composite priamry key이므로 BCNF를 만족한다.

9. Photo

- Attributes: Photo_id (PK), Interior_photo, Exterior_photo, Inspection_photo
- FDs:
  - Photo_id → Interior_photo, Exterior_photo, Inspection_photo

* Photo_id가 primary key이므로 BCNF를 만족한다.

10. Room

- Attributes: Room_id (PK), Bed_room, Bath_room
- FDs:
  - Room_id → Bed_room, Bath_room

* Room_id가 primary key이므로 BCNF를 만족한다.

11. Seller

- Attributes: Seller_id (PK), Name, Address, Phone_num
- FDs:
  - Seller_id → Name, Address, Phone_num

* Seller_id가 primary key이므로 BCNF를 만족한다.

12. Seller_Market

- Attributes: Seller_id (FK), Market_id (FK)
- FDs:
  - {Seller_id, Market_id} is the composite primary key/
  - Seller_id, Market_id → (Seller_id, Market_id)

* Seller_id, Market_id가 composite priamry key이므로 BCNF를 만족한다.

13. Seller_Sold

- Attributes: Seller_id (FK), Sold_id (FK)
- FDs:
  - Seller_id, Sold_id → (Seller_id, Sold_id)

* Seller_id, Sold_id가 composite priamry key이므로 BCNF를 만족한다.

14. Sold

- Attributes: Sold_id (PK), Price, Name, Address, Sold_date, School_district, Type
- FDs:
  - Sold_id → Price, Name, Address, Sold_date, School_district, Type

* Sold_id가 primary key이므로 BCNF를 만족한다.

15. Sold_Photo

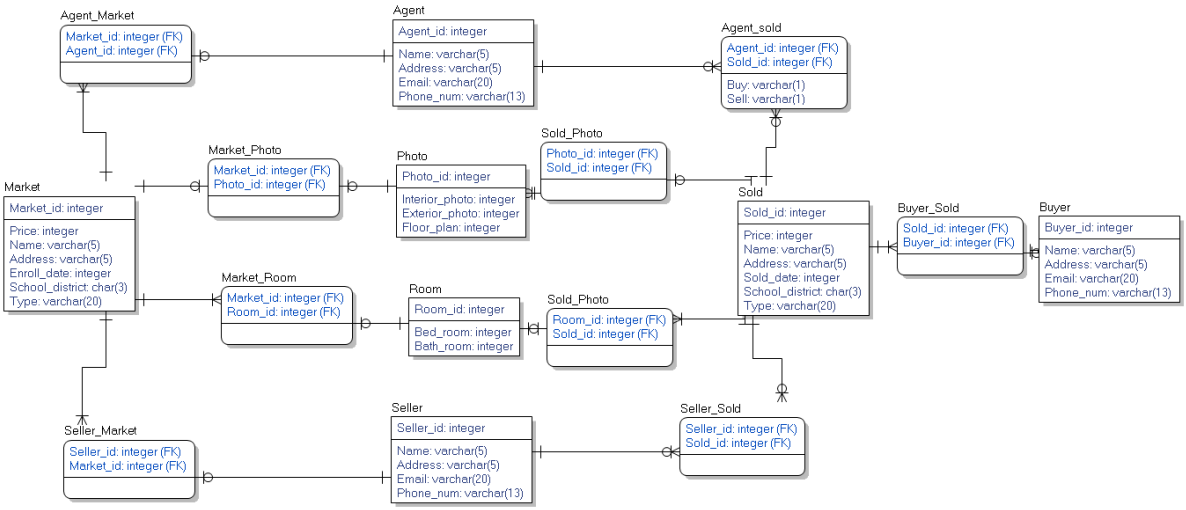
- Attributes: Photo_id (FK), Sold_id (FK)
- FDs:
  - {Photo_id, Sold_id} is the composite primary key, so:
  - Photo_id, Sold_id → (Photo_id, Sold_id)

* Photo_id, Sold_id가 composite priamry key이므로 BCNF를 만족한다.

*** Summary**

모든 Schema가 BCNF를 만족하며, 그 이유는 모든 FD의 LHS에는 candidate key 또는 primary key의 일부가 있고, partial 또는 ransitive dependency가 있기 때문이다.

모든 Decomposition을 마친 Logical Schema Diagram은 아래의 그림과 같다.



2. Physical Schema Diagram

위에서 Decomposition 을 마친 Schema 들에 대하여 각 스키마에 대한 Data Type, Doamin Constraint, Relationship Type 및 NULL 허용 여부를 정의한다.

1. Agent

- Agent_id: INTEGER (PK, NOT NULL)
- Name: VARCHAR(50), NOT NULL
- Address: VARCHAR(100), NOT NULL
- Email: VARCHAR(100), NOT NULL
- Phone_num: VARCHAR(13), NOT NULL

2. Agent_Market

- Market_id: INTEGER (FK, NOT NULL)
- Agent_id: INTEGER (FK, NOT NULL)

3. Agent_Sold

- Agent_id: INTEGER (FK, NOT NULL)
- Sold_id: INTEGER (FK, NOT NULL)
- Buy: VARCHAR(1), NOT NULL
- Sell: VARCHAR(1), NOT NULL

4. Buyer

- Buyer_id: INTEGER (PK, NOT NULL)
- Name: VARCHAR(50), NOT NULL

- Address: VARCHAR(100), NOT NULL
- Email: VARCHAR(100), NOT NULL
- Phone_num: VARCHAR(13), NOT NULL

5. Buyer_Sold

- Sold_id: INTEGER (FK, NOT NULL)
- Buyer_id: INTEGER (FK, NOT NULL)

6. Market

- Market_id: INTEGER (PK, NOT NULL)
- Price: INTEGER, NOT NULL
- Name: VARCHAR(50), NOT NULL
- Address: VARCHAR(100), NOT NULL
- Enroll_date: INTEGER, NOT NULL
- School_district: INTEGER, NOT NULL
- Type: VARCHAR(20), NOT NULL

7. Market_Photo

- Market_id: INTEGER (FK, NOT NULL)
- Photo_id: INTEGER (FK, NOT NULL)

8. Market_Room

- Market_id: INTEGER (FK, NOT NULL)
- Room_id: INTEGER (FK, NOT NULL)

9. Photo

- Photo_id: INTEGER (PK, NOT NULL)
- Interior_photo: INTEGER, NOT NULL
- Exterior_photo: INTEGER, NOT NULL
- Inspection_photo: INTEGER, NOT NULL

10. Room

- Room_id: INTEGER (PK, NOT NULL)
- Bed_room: INTEGER, NOT NULL
- Bath_room: INTEGER, NOT NULL

11. Seller

- Seller_id: INTEGER (PK, NOT NULL)
- Name: VARCHAR(50), NOT NULL
- Address: VARCHAR(100), NOT NULL
- Email: VARCHAR(100), NOT NULL
- Phone_num: VARCHAR(13), NOT NULL

12. Seller_Market

- Seller_id: INTEGER (FK, NOT NULL)
- Market_id: INTEGER (FK, NOT NULL)

13. Seller_Sold

- Seller_id: INTEGER (FK, NOT NULL)
- Sold_id: INTEGER (FK, NOT NULL)

14. Sold

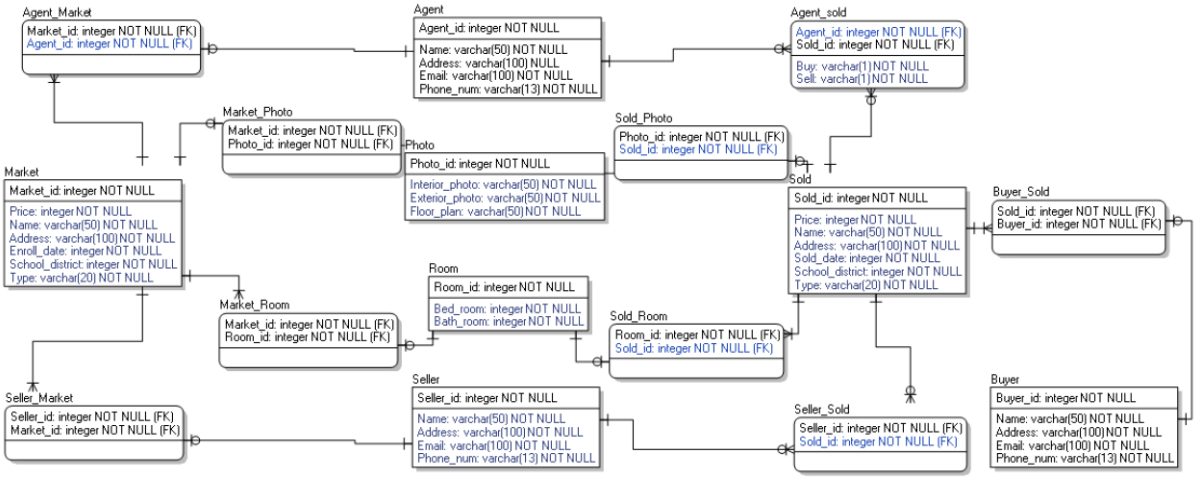
- Sold_id: INTEGER (PK, NOT NULL)
- Price: INTEGER, NOT NULL
- Name: VARCHAR(50), NOT NULL
- Address: VARCHAR(100), NOT NULL
- Sold_date: INTEGER, NOT NULL
- School_district: INTEGER, NOT NULL
- Type: VARCHAR(20), NOT NULL

15. Sold_Photo

- Photo_id: INTEGER (FK, NOT NULL)
- Sold_id: INTEGER (FK, NOT NULL)

ID, 가격 및 개수와 같은 값들은 INTEGER Type 으로 정의했으며,
이름 또는 주소와 같이 경우에 따라서 길이의 차이가 클 수 있는 경우에는
VARCHAR(50), VARCHAR(100)과 같이 가변 String Type 으로 정의했다.

위에서 모든 정의를 마친 Physical Schema Diagram 은 아래의 그림과 같다.



*Physical Schema Diagram

3. Queries

1. CRUD File

먼저, 데이터베이스 초기화를 위해 사용자로부터 쿼리를 입력받기 전에 텍스트 파일을 읽고 ODBC를 통해 데이터베이스에 테이블을 생성하고 데이터를 입력하는 과정이 필요하며, 사용자가 프로그램 종료를 요청하면 생성된 테이블을 모두 제거해야 한다.

이를 위해 두 개의 입력 파일 (20212022_1.txt와 20212022_2.txt)을 사용했다.

```
create table agent(agent_id integer, name varchar(50) not null, address varchar(100) not null, email(100) not null,
phone_num(13) not null, primary key(agent_id));
insert into agent values (1,'Richard', 'Gangnam','sdflkj2340@naver.com' , '123-3424-5235');
insert into agent values (2,'Ruby', 'Gangbuk','mekaka2553@naver.com' , '211-1233-1233');
insert into agent values (3,'Brigida', 'Mapo','dskjwi10@naver.com' , '213-5435-5821');
insert into agent values (4,'Mary', 'Dongdaemun','dslsi023@naver.com' , '492-1239-3249');
create table agent_market(agent_id integer, market_id integer, primary key(agent_id, market_id), foreign key (agent_id),
references agent (agent_id), foreign key (market_id), references market (market_id));
insert into agent_market values (4, 1);
insert into agent_market values (1, 2);
insert into agent_market values (2, 3);
insert into agent_market values (1, 4);
insert into agent_market values (4, 5);
insert into agent_market values (2, 6);
insert into agent_market values (2, 7);
insert into agent_market values (1, 8);
insert into agent_market values (4, 9);
insert into agent_market values (3, 10);
insert into agent_market values (3, 11);
insert into agent_market values (4, 12);
insert into agent_market values (1, 13);
insert into agent_market values (2, 14);
insert into agent_market values (4, 15);
insert into agent_market values (1, 16);
insert into agent_market values (4, 17);
insert into agent_market values (4, 18);
insert into agent_market values (2, 19);
insert into agent_market values (3, 20);
create table agent_sold(agent_id integer, sold_id integer, buy varchar(1) not null, sell varchar(1) not null, primary
key(agent_id, sold_id), foreign key (agent_id), references agent (agent_id), foreign key (sold_id), references sold
(sold_id));
insert into agent_sold values (4, 1, 'O', 'X');
insert into agent_sold values (1, 2, 'O', 'X');
insert into agent_sold values (2, 3, 'X', 'O');
insert into agent_sold values (1, 4, 'X', 'O');
insert into agent_sold values (4, 5, 'X', 'O');
insert into agent_sold values (2, 6, 'O', 'X');
insert into agent_sold values (2, 7, 'O', 'X');
insert into agent_sold values (1, 8, 'X', 'O');
insert into agent_sold values (4, 9, 'X', 'O');
insert into agent_sold values (3, 10, 'X', 'O');
insert into agent_sold values (3, 11, 'X', 'O');
insert into agent_sold values (4, 12, 'O', 'X');
insert into agent_sold values (1, 13, 'O', 'X');
insert into agent_sold values (2, 14, 'X', 'O');
insert into agent_sold values (4, 15, 'X', 'O');
insert into agent_sold values (1, 16, 'O', 'X');
insert into agent_sold values (4, 17, 'X', 'O');
```

*20212022-1.txt 일부분

```
drop table agent;
drop table agent_market;
drop table agent_sold;
drop table buyer;
drop table buyer_sold;
drop table market;
drop table market_photo;
drop table market_room;
drop table photo;
drop table room;
drop table seller;
drop table seller_market;
drop table seller_sold;
drop table sold;
drop table sold_photo;
```

*20212022-2.txt

두 개의 텍스트 파일이 있는데, 첫 번째 파일인 20212022_1.txt에는 테이블을 생성하고 데이터를 삽입하는 명령어가 포함되어 있다. 두 번째 파일인 20212022_2.txt에는 20212022_1.txt에서 생성된 테이블들을 제거하는 명령어가 포함되어 있다.
테이블을 제거할 때는 참조 관계를 고려하여 제거 순서를 유의해야 한다.

2. Query

쿼리를 처리하기 위해서는 서브쿼리를 포함하여 총 13개의 쿼리를 처리해야 한다.
먼저 텍스트 파일을 통해 데이터베이스에 데이터를 입력한 후, 사용자로부터 원하는 쿼리를 입력받는다.

- TYPE 1

```sql
SELECT m.market_id, m.name
FROM market AS m
WHERE m.address = 'Mapo';
```

- TYPE 1-1

```sql
SELECT m.market_id, m.name
FROM market AS m
WHERE m.address = 'Mapo' AND m.price BETWEEN 1000000000 AND 1500000000;
```

- TYPE 2

```sql
SELECT m.market_id, m.name
FROM market AS m
WHERE m.school_district = 8;
```

- **TYPE 2-1**

```
SELECT m.market_id, m.name
FROM market AS m
JOIN market_room AS mr ON m.market_id = mr.market_id
JOIN room AS r ON mr.room_id = r.room_id
WHERE m.school_district = 8
  AND r.bed_room >= 4
  AND r.bath_room = 2;
```

- **TYPE 3**

```
SELECT a.agent_id, a.name
FROM agent AS a
JOIN agent_sold AS ags ON a.agent_id = ags.agent_id
JOIN sold AS s ON ags.sold_id = s.sold_id
WHERE s.sold_date = 2022
GROUP BY a.agent_id, a.name
ORDER BY SUM(s.price) DESC
LIMIT 1;
```

- **TYPE 3-1**

```
SELECT a.agent_id, a.name
FROM agent AS a
JOIN agent_sold AS ags ON a.agent_id = ags.agent_id
JOIN sold AS s ON ags.sold_id = s.sold_id
WHERE s.sold_date = 2023
GROUP BY a.agent_id, a.name
ORDER BY SUM(s.price) DESC
LIMIT %d;
```

- TYPE 3-2

```sql
WITH TotalValue AS (
    SELECT a.agent_id, a.name, SUM(s.price) AS total_value
    FROM agent AS a
    JOIN agent_sold AS ags ON a.agent_id = ags.agent_id
    JOIN sold AS s ON ags.sold_id = s.sold_id
    WHERE s.sold_date = 2021
    GROUP BY a.agent_id, a.name
),
TotalAgents AS (
    SELECT COUNT(*) AS cnt
    FROM TotalValue
),
Bottom10Percent AS (
    SELECT total_value
    FROM TotalValue
    ORDER BY total_value
    LIMIT (SELECT cnt / 10 FROM TotalAgents)
)
SELECT a.agent_id, a.name
FROM TotalValue AS a
JOIN Bottom10Percent AS b ON a.total_value = b.total_value
ORDER BY a.total_value;
```

- TYPE 4

```sql
SELECT a.agent_id, a.name,
       AVG(s.price) AS avg_price,
       AVG(DATEDIFF(s.sold_date, m.enroll_date)) AS avg_days_on_market
FROM agent AS a
JOIN sold AS s ON a.agent_id = s.agent_id
JOIN market AS m ON s.property_id = m.property_id
WHERE s.sale_date = 2022
GROUP BY a.agent_id, a.name;
```

- TYPE 4-1

```sql
SELECT a.agent_id, a.name, MAX(s.price) AS max_price
FROM agent AS a
JOIN sold AS s ON a.agent_id = s.agent_id
WHERE s.sale_date = 2023
GROUP BY a.agent_id, a.name;
```

- TYPE 4-2

```
SELECT a.agent_id, a.name,
       MAX(DATEDIFF(s.sold_date, m.enroll_date)) AS max_days_on_market
FROM agent AS a
JOIN sold AS s ON a.agent_id = s.agent_id
JOIN market AS m ON s.property_id = m.property_id
GROUP BY a.agent_id, a.name;
```

- TYPE 5

```
SELECT m.id, m.name, m.type, m.photo
FROM market AS m
JOIN (
    SELECT m.type, MAX(m.price) AS max_price
    FROM market AS m
    GROUP BY m.type
) AS max_prices
ON m.type = max_prices.type AND m.price = max_prices.max_price
WHERE m.type IN (
    'studio',
    'one-bedroom',
    'multi-bedroom apartment(s)',
    'detached house'
);
```

- TYPE 6

```
INSERT INTO sold (sold_id, price, name, address, sold_date, school_district, type)
VALUES (%d, %d, '%s', '%s', '%s', '%s', '%s');

INSERT INTO seller_sold (seller_id, sold_id)
VALUES (%d, %d);

INSERT INTO agent_sold (agent_id, sold_id, buy, sell)
VALUES (%d, %d, 'O', 'X');

INSERT INTO buyer_sold (buyer_id, sold_id)
VALUES (%d, %d);
```

- TYPE 7

```
INSERT INTO agent (agent_id, name, address, email, phone_num)
VALUES (%d, '%s', '%s', '%s', '%s');
```