**Database System Project 2**



분반 : 1분반

학번 : 20212020

이름 : 박민준

**1. BCNF Decomposition**

텍스트, 스크린샷, 도표, 폰트이(가) 표시된 사진

자동 생성된 설명

- Properties 테이블의 초기 상태

속성: PropertyID (PK), Address, District, Price, PropertyType, AgentID, SellerID, ListedDate, ForSale

- 함수적 종속성

PropertyID → Address, District, Price, PropertyType, AgentID, SellerID, ListedDate, ForSale

District, AgentID → PropertyID (이것은 비정규 함수 종속이다.)

- 이 함수적 종속성 중에서 District, AgentID → PropertyID는 부분 함수 종속성을 나타내며 BCNF 위반이다. BCNF 조건은 모든 비자명 함수적 종속성 X → Y에 대해 X가 슈퍼 키여야 한다는 것이다. 하지만 District, AgentID → PropertyID에서 District와 AgentID는 슈퍼 키가 아니므로 BCNF를 위반한다.

- 즉, District와 AgentID가 부분적으로 결정자 역할을 하고 있으므로 BCNF 위반이다. 부분 함수 종속성이 존재하므로 PropertyID에 대해 모든 속성이 결정되지 않는다. 비정규 함수 종속성을 제거하기 위해 District와 AgentID를 분리한다. 그리고, District, AgentID → PropertyID의 종속성을 제거하기 위해 새로운 테이블을 생성한다.

- AgentDistrict 테이블

- 속성: AgentID, District (PK)

- 함수적 종속성: District, AgentID → PropertyID (PK)

- BCNF로 분해하는 과정에서 District와 AgentID를 사용하여 새로운 테이블을 만들어야 한다. 이를 통해 부분 함수 종속성을 제거하고 테이블을 BCNF 형태로 변환한다. 정규화 후 테이블은 다음과 같다.

- Properties 테이블

속성: PropertyID (PK), Address, Price, PropertyType, SellerID, ListedDate, ForSale

- AgentDistrict 테이블

속성: AgentID (FK), District (PK)

- 해당 테이블은 특정 에이전트가 담당하는 지역 정보를 저장하는 테이블이다. 각 에이전트의 고유 ID와 해당 에이전트가 담당하는 지역을 포함한다.

- 이 과정은 테이블이 BCNF 형태로 변환되었음을 나타낸다. 결과적으로, 모든 함수 종속성이 후보 키에 의해 결정되므로 BCNF를 만족한다.

- 이 정규화 과정은 데이터 무결성을 유지하고 데이터베이스의 효율성을 높이는 데 기여한다. BCNF 정규화는 이러한 이상 현상을 제거하고 데이터베이스의 구조를 개선하는 중요한 과정이다. 전체 테이블을 요약하면 다음과 같다.

1) Agents 테이블

속성: AgentID (PK), AgentName, PhoneNumber

2) Buyers 테이블

속성: BuyerID (PK), BuyerName, ContactInfo

3) Sellers 테이블

속성: SellerID (PK), SellerName, ContactInfo

4) Properties 테이블

속성: PropertyID (PK), Address, Price, PropertyType, SellerID, ListedDate, ForSale

5) Photos 테이블

속성: PhotoID (PK), PropertyID (FK), PhotoType

6) Rooms 테이블

속성: RoomID (PK), PropertyID (FK), BedroomCount, BathroomCount

7) Sales 테이블

속성: SaleID (PK), PropertyID (FK), SalePrice, SaleDate, BuyerID (FK), AgentID (FK)

8) AgentDistrict 테이블

속성: AgentID (FK), District (PK)

- 이로써 모든 테이블이 BCNF를 만족하게 되었다. BCNF 분해는 데이터베이스의 정규화를 통해 데이터 무결성과 일관성을 보장하기 위한 과정이다. Properties 테이블의 AgentID와 District 속성 사이에 부분 함수 종속성이 존재했기 때문에, 이를 해결하기 위해 BCNF 분해를 수행하여 AgentDistrict 테이블을 생성했다. 이로써 각 에이전트가 담당하는 지역 정보를 분리하여 저장하게 되었다.

**2. Physical Schema diagram**

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

1) agents 테이블

agent\_id: 에이전트의 고유 ID (Primary Key, INTEGER, NOT NULL)

name: 에이전트의 이름 (VARCHAR(20), NULL 허용)

phone\_num: 에이전트의 전화번호 (VARCHAR(20), NULL 허용)

- 에이전트의 고유 ID를 Primary Key로 설정하여 각 에이전트를 고유하게 식별한다. 에이전트의 이름과 전화번호는 VARCHAR(20) 형식으로 저장하며, NULL 값을 허용한다.

2) buyers 테이블

buyer\_id: 구매자의 고유 ID (Primary Key, INTEGER, NOT NULL)

name: 구매자의 이름 (VARCHAR(20), NULL 허용)

phone\_num: 구매자의 전화번호 (VARCHAR(20), NULL 허용)

- 구매자의 고유 ID를 Primary Key로 설정하여 각 구매자를 고유하게 식별한다. 구매자의 이름과 전화번호는 VARCHAR(20) 형식으로 저장하며, NULL 값을 허용한다.

3) sellers 테이블

seller\_id: 판매자의 고유 ID (Primary Key, INTEGER, NOT NULL)

name: 판매자의 이름 (VARCHAR(20), NULL 허용)

phone\_num: 판매자의 전화번호 (VARCHAR(20), NULL 허용)

- 판매자의 고유 ID를 Primary Key로 설정하여 각 판매자를 고유하게 식별한다. 판매자의 이름과 전화번호는 VARCHAR(20) 형식으로 저장하며, NULL 값을 허용한다.

4) properties 테이블

property\_id: 부동산의 고유 ID (Primary Key, INTEGER, NOT NULL)

seller\_id: 판매자의 ID (Foreign Key, INTEGER, NOT NULL)

agent\_id: 에이전트의 ID (Foreign Key, INTEGER, NOT NULL)

type: 부동산 유형 (VARCHAR(20), NOT NULL)

address: 부동산 주소 (VARCHAR(50), NULL 허용)

listed\_date: 부동산 목록에 등록된 날짜 (DATETIME, NULL 허용)

price: 부동산 가격 (INTEGER, NULL 허용)

district: 구역 (VARCHAR(20), NULL 허용)

for\_sale: 판매 여부 (BOOLEAN, NULL 허용)

- 부동산의 고유 ID를 Primary Key로 설정하여 각 부동산을 고유하게 식별한다. 외래 키로 seller\_id와 agent\_id를 참조하여 각 부동산이 특정 판매자와 에이전트에 연결되도록 한다. 부동산의 유형, 주소, 가격, 구역 등 다양한 속성들은 각각 적절한 데이터 타입으로 설정된다.

5) photos 테이블

photo\_id: 사진의 고유 ID (Primary Key, CHAR(18), NOT NULL)

property\_id: 부동산의 ID (Foreign Key, INTEGER, NOT NULL)

type: 사진 유형 (VARCHAR(20), NULL 허용)

- 사진의 고유 ID를 Primary Key로 설정하여 각 사진을 고유하게 식별한다. 외래 키로 property\_id를 참조하여 각 사진이 특정 부동산에 연결되도록 한다. 사진의 유형은 VARCHAR(20) 형식으로 저장하며, NULL 값을 허용한다.

6) rooms 테이블

room\_id: 방의 고유 ID (Primary Key, INTEGER, NOT NULL)

property\_id: 부동산의 ID (Foreign Key, INTEGER, NOT NULL)

bed\_num: 침실 수 (INTEGER, NOT NULL)

bath\_num: 욕실 수 (INTEGER, NOT NULL)

- 방의 고유 ID를 Primary Key로 설정하여 각 방을 고유하게 식별한다. 외래 키로 property\_id를 참조하여 각 방이 특정 부동산에 연결되도록 한다. 침실 수와 욕실 수는 INTEGER 형식으로 저장하며, NULL 값을 허용하지 않는다.

7) sales 테이블

sale\_id: 판매의 고유 ID (Primary Key, INTEGER, NOT NULL)

property\_id: 부동산의 ID (Foreign Key, INTEGER, NOT NULL)

sale\_price: 판매 가격 (INTEGER, NULL 허용)

sale\_date: 판매 날짜 (DATETIME, NULL 허용)

buyer\_id: 구매자의 ID (Foreign Key, INTEGER, NOT NULL)

agent\_id: 에이전트의 ID (Foreign Key, INTEGER, NOT NULL)

- 판매의 고유 ID를 Primary Key로 설정하여 각 판매를 고유하게 식별한다. 외래 키로 property\_id, buyer\_id, agent\_id를 참조하여 각 판매가 특정 부동산, 구매자, 에이전트에 연결되도록 한다. 판매 가격과 판매 날짜는 각각 INTEGER와 DATETIME 형식으로 저장되며, NULL 값을 허용한다.

8) agent\_district 테이블

agent\_id: 에이전트의 ID (Foreign Key, INTEGER, NOT NULL)

district: 구역 (VARCHAR(20), NOT NULL)

- 이 테이블은 BCNF 분해를 통해 새로 생성된 테이블로, 에이전트와 구역 간의 관계를 관리한다. 외래 키로 agent\_id를 참조하며, 구역은 VARCHAR(20) 형식으로 저장된다. 에이전트 ID와 구역 모두 NULL값을 허용하지 않는다.

9) seller\_agent 테이블

agent\_id: 에이전트의 ID (Foreign Key, INTEGER, NOT NULL)

seller\_id: 판매자의 ID (Foreign Key, INTEGER, NOT NULL)

- 이 테이블은 에이전트와 판매자 간의 관계를 관리한다. 외래 키로 agent\_id와 seller\_id를 참조하여 각 관계가 특정 에이전트와 판매자에 연결되도록 한다.

10) market\_agent 테이블

agent\_id: 에이전트의 ID (Foreign Key, INTEGER, NOT NULL)

property\_id: 부동산의 ID (Foreign Key, INTEGER, NOT NULL)

- 이 테이블은 에이전트와 부동산 간의 관계를 관리한다. 외래 키로 agent\_id와 property\_id를 참조하여 각 관계가 특정 에이전트와 부동산에 연결되도록 한다.

11) buyer\_agent 테이블

buyer\_id: 구매자의 ID (Foreign Key, INTEGER, NOT NULL)

agent\_id: 에이전트의 ID (Foreign Key, INTEGER, NOT NULL)

- 이 테이블은 구매자와 에이전트 간의 관계를 관리한다. 외래 키로 buyer\_id와 agent\_id를 참조하여 각 관계가 특정 구매자와 에이전트에 연결되도록 한다.

- 해당 물리적 스키마 다이어그램은 데이터베이스 설계의 핵심 요소를 담고 있다. 모든 테이블은 BCNF 정규화 과정을 거쳐 중복과 이상현상을 최소화하였으며, 각 테이블의 속성들은 명확한 데이터 타입과 제약 조건을 가지고 있다. 특히, agent\_district 테이블은 BCNF 분해 과정에서 새롭게 생성된 테이블로, agent\_id를 외래키로 가지며 district와의 관계를 명확히 하고 있다. 이러한 설계를 통해 데이터베이스는 보다 효율적이고 안정적인 데이터 관리가 가능하게 되었다.

**3. ODBC implementation within MySQL**

텍스트, 스크린샷, 폰트, 문서이(가) 표시된 사진

자동 생성된 설명

- CRUD.txt 파일은 데이터베이스와 테이블을 생성하고, 데이터를 삽입한 후, 드롭 명령어를 포함하는 SQL 스크립트를 포함하고 있다. 해당 파일에 대한 예시는 위 사진과 같다.

- 먼저, project라는 데이터베이스를 생성하고, CREATE TABLE query를 통해 각각의 테이블을 생성한다.

- 테이블 생성 후, INSERT INTO query를 사용해 데이터를 삽입하는데, Agents, Buyers, Sellers, Properties, Photos, Rooms, Sales 테이블에 대한 데이터가 삽입된다.

예시: INSERT INTO Agents (AgentID, AgentName, PhoneNumber) VALUES ('A001', 'Alice Johnson', '010-1234-5678');

- 마지막으로 DROP query를 통해 데이터 삭제를 한다. CRUD.txt 파일에서는 사용자의 입력을 받기 전에는 TABLE이 DROP되면 안 되기 때문에 ‘$$$’로 DROP과 나머지 부분을 구분하였다. DROP은 데이터베이스와 테이블을 삭제하는 명령어이다.

예시: DROP TABLE IF EXISTS AgentDistrict; DROP DATABASE project;

- CRUD.txt 파일 생성 후, cpp 파일을 작성하여 프로그램을 실행하게 되는데, 작성한 20212020.cpp 파일은 MySQL 데이터베이스에 연결하고, CRUD.txt 파일을 처리하며, 다양한 유형의 쿼리를 수행하는 C++ 프로그램이다.

- 주요 기능과 각 유형(TYPE)의 처리 방식을 설명하겠다.

1) 주요 기능을 담당하는 함수

(1) connectDatabase() 함수

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

- MySQL 데이터베이스에 연결하는 함수이다. 성공 시 데이터베이스 연결 객체를 반환한다.

(2) processCrudFile() 함수

텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

- CRUD.txt 파일을 읽어 테이블 생성, 데이터 삽입 등의 작업을 수행한다. $$$ 이후의 DROP 명령어는 따로 저장되어 이후 사용자로부터 0을 입력받으면 한꺼번에 수행되도록 코드를 작성하였다.

(3) executeQuery() 함수

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

- 주어진 쿼리를 실행하고 결과를 출력하는 함수이다. printHeader 파라미터를 통해 컬럼 헤더를 출력할지 여부를 설정할 수 있다.

2) 각 유형(TYPE)의 처리 방식

(1) TYPE 1: Mapo 지역의 판매 중인 부동산 조회

- 목적: Mapo 지역에서 판매되고 있는 부동산을 조회한다.

텍스트, 스크린샷, 폰트, 메뉴이(가) 표시된 사진

자동 생성된 설명

- 처음 프로그램을 실행하면 다음과 같이 QUERY TYPE를 SELECT하는 menu가 출력되고, 사용자가 1을 입력하면 Properties의 주소가 Mapo인 지역에서 ForSale=1인 부동산을 조회한다.

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

- 이후 TYPE 1의 Subtypes를 고르는 menu가 출력되고, 마찬가지로 사용자가 1을 입력하면 Properties의 address가 Mapo인 지역에서 ForSale=1이고 가격이 10억에서 15억 사이인 부동산을 조회하게 된다.

- Subtypes 결과가 출력됨과 동시에 다시 초기 QUERY TYPES를 SELECT하는 menu로 돌아오게 된다. 이는 사용자가 0을 누르기 전까지 무한하게 반복된다.

(2) TYPE 2: 특정 구역의 판매 중인 부동산 조회

- 목적: 여러 구역에서 판매 중인 부동산을 조회한다.

텍스트, 스크린샷, 메뉴, 폰트이(가) 표시된 사진

자동 생성된 설명

- 8학군에 해당하는 지역에서 ForSale=1인 부동산을 조회한다.

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

TYPE 2-1은 지정된 구역에서 ForSale=1이고 침실이 4개 이상, 욕실이 2개 이상인 부동산을 조회한다. 이 함수는 사용자가 선택한 옵션에 따라 적절한 쿼리를 실행하고 결과를 출력한다.

(3) TYPE 3: 에이전트의 판매 실적 조회

- 목적: 특정 연도의 에이전트 판매 실적을 조회한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

- 기본 쿼리: 2022년 총 판매 금액 기준 가장 많은 판매를 한 에이전트를 조회한다.

텍스트, 스크린샷, 폰트, 블랙이(가) 표시된 사진

자동 생성된 설명

- TYPE 3-1은 2023년 총 판매 금액 기준 상위 K명의 에이전트를 조회한다. 사용자 입력을 통해 K 값을 받아 쿼리를 동적으로 생성한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

- TYPE 3-2는 2021년 하위 10% 에이전트를 조회한다. 전체 에이전트 수의 10%를 계산한 후, 해당 수 만큼의 하위 에이전트를 조회하는 쿼리를 생성한다. 이는 countQuery 결과를 기반으로 하위 10% 에이전트를 조회하는 쿼리를 이용한다.

(4) TYPE 4: 에이전트의 평균 판매 가격 및 시장에서의 평균 시간 조회

- 목적: 에이전트의 연도별 평균 판매 가격과 시장에 나와 있는 평균 시간을 조회한다.

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

- 기본 쿼리는 2022년 에이전트의 평균 판매 가격과 시장에 나와 있는 평균 시간을 조회한다.

텍스트, 스크린샷, 폰트, 블랙이(가) 표시된 사진

자동 생성된 설명

- TYPE 4-1은 2023년 최대 판매 가격을 조회한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

- TYPE 4-2는 시장에 나와 있는 가장 긴 시간을 조회한다.

(5) TYPE 5: 부동산 유형별 가장 비싼 부동산의 사진 조회

- 목적: 각 부동산 유형에서 가장 비싼 부동산의 사진을 조회한다.

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

- TYPE 5는 가장 비싼 studio 부동산, one-bedroom 부동산, multi-bedroom 부동산, detached 부동산의 사진을 조회한다.

- 즉, 해당 함수는 각 유형별로 가장 비싼 부동산의 사진을 조회하고, 결과를 출력한다. 첫 번째 쿼리에서는 컬럼 헤더를 출력하고, 이후 쿼리에서는 컬럼 헤더를 생략하여 연속적으로 결과를 표시한다.

(6) TYPE 6: 새로운 판매 기록 추가

- 목적: 새로운 판매 기록을 추가한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

- TYPE 6 쿼리 함수는 사용자로부터 판매 기록에 필요한 정보를 입력받는다. 입력된 데이터를 기반으로 Sales 테이블에 삽입하는 쿼리를 생성하고 실행한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

- 만약 동일한 SaleID를 추가한다면 다음과 같이 PRIMARY KEY Error를 출력한다.

(7) TYPE 7: 새로운 에이전트 추가

- 목적: 새로운 에이전트를 추가한다.

텍스트, 스크린샷, 폰트, 디자인이(가) 표시된 사진

자동 생성된 설명

- 해당 TYPE은 사용자로부터 에이전트 정보를 입력받는다. 입력된 데이터를 기반으로 Agents 테이블에 삽입하는 쿼리를 생성하고 실행한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

- 만약 동일한 AgentID를 추가한다면 다음과 같이 PRIMARY KEY Error를 출력한다.

(8) TYPE 0: 프로그램 종료

텍스트, 스크린샷, 폰트, 디자인이(가) 표시된 사진

자동 생성된 설명

- 마지막으로 사용자가 0을 입력하면 CRUD.txt 파일의 DROP TABLE, DROP DATABASE query가 전부 실행되고 난 뒤에 프로그램이 종료된다. MySQL Workbench를 확인해보면 해당 DATABASE와 TABLE들이 전부 DROP된 것을 확인할 수 있다.