DMA_project1_team3 보고서

조원: 김 근, 윤 현, 한창진

<PART1> ER 다이어그램 도식화

<PART2> DB구현 및 데이터 입력

<PART1>

1. 문제 정의

문제를 먼저 간단하게 정의하자. 문제는 멘토와 멘티가 교류하는 사이트 A의 DB에 대한 ER 다이어그램을 도식화하는 것이 목표이다. 그러기 위해서 먼저 R1-1 ~ R1-9에 해당하는 requirement를 따라 entity, relationship, attribute를 설정하고 이를 도식화하여야 한다.

먼저 entity, relationship, attribute의 의미를 정의하고 시작하자.

1) entity란 단독으로 존재하는 객체를 의미하며, 동일한 객체는 존재하지 않는다.

2) attribute란 entity가 갖는 속성을 의미한다.

3) relationship이란 Entity Type간의 관계를 의미한다.

2. ER 다이어그램 도식화 과정

전체의 과정에 대해 들어가기 앞서 문제를 해결하기 위해 접근한 방향성을 알아보자. 먼저 ER 다이어그램에서 최대한 relationship에 attribute를 부여하지 않으려고 했다. 대부분의 attribute는 entity에 부여되었으며, 필요한 상황에만 relationship에 부여하였다. 이러한 방향성을 갖고 ER 다이어그램을 도식화하였다.

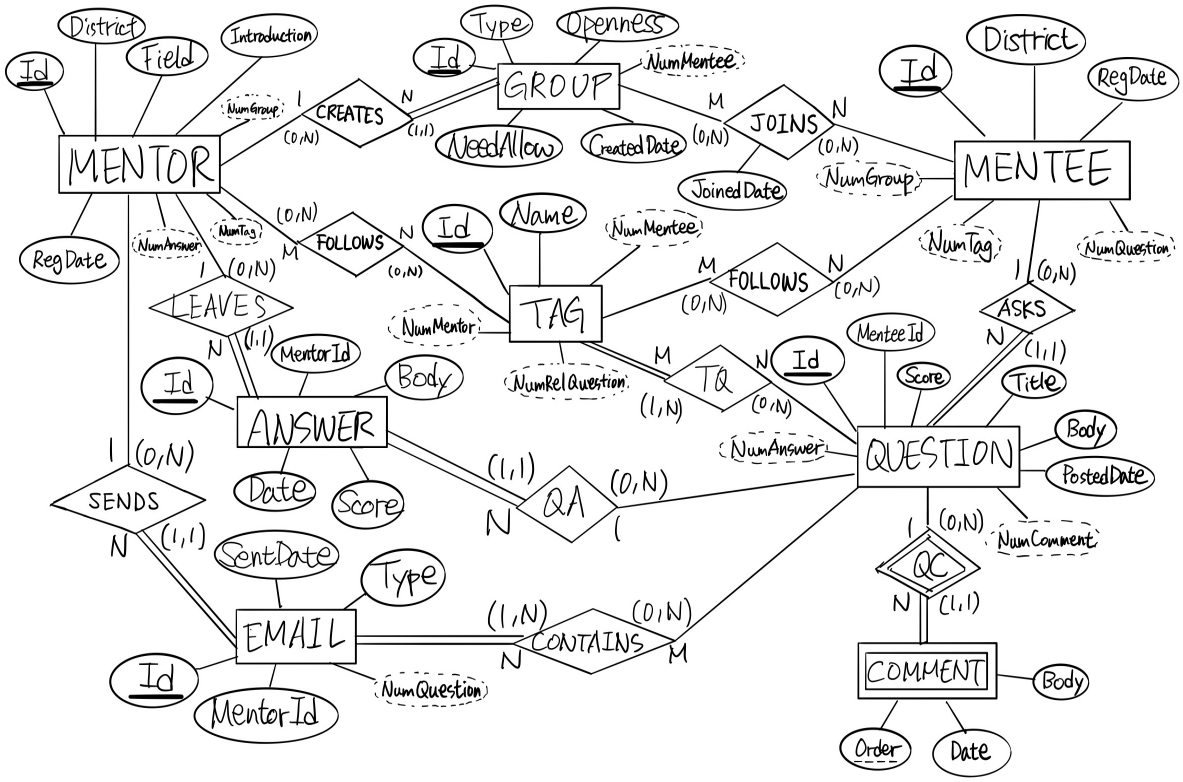
일단 Entity에는 requirement에서 요구 된 mentor, mentee, group, question, answer, tag, email, comment 이렇게 8가지를 정하였다. 그리고 comment를 제외한 entity는 regular entity이다. 하지만 comment는 question이 반드시 있어야 형성되므로 weak entity로 설정하였다. 또한 weak entitiy인 comment에서 order를 partial key로 설정하였다.

그리고 attribute의 경우에는 derived attribute는 가장 적게 하려고 노력하였다. Emil이라는 entity에서 mentor id가 derived attribute인지 아닌지 판별하는 과정이 attribute를 설정하는 데에 있어서 걸림돌이었다. 이를 해결하기 위해 derived attribute의 정의를 확인해 보았다. derived attribute란 '다른 attribute로부터 계산된 특성(주로 사칙연산의 의미로 예시들이 등장함)' 이라는 정의를 가지고 있다. 따라서 프로젝트 시에 고민이 있었던 EMAIL 에서 mentor id는 다른 attribute에서 계산되는 것이 아니고 email이라는 entity 하나를 만들 때 함께 입력해줘야 하는 정보니까 derived attribute가 아닌 것으로 결론을 내릴 수 있었다. 이렇게 하여 attribute의 경우에는 derived attribute는 mentor에서 tag num과 같은 다른 attribute로부터 계산된 특성인 num 부류 attribute만 derived attribute로 처리하였다. 그리고 또한 composite attribute와 multi valued attribute의 경우에는 해당 문제에는 존재하지 않았다.

Relationship의 경우에는 위에서 말하였듯이 attribute를 가능한 한 부여하지 않으려고 하였으며, 그런 노력을 기울인 결과 하나의 relationship에만 attribute가 부여되었다. 1) Group과 mentee의

n:m 관계인 'joins' relationship에서 joined_date라는 attribute를 설정해주었다. 그 이유는 다음과 같다. joined_date라는 속성은 group이라는 것 자체의 속성도 아니고 mentee라는 것 자체의 속성도 아니고 둘이 같이 맺는 관계 속에서만 정의할 수 있으므로 'joins' relationship에 joined_date라는 attribute를 부여하였다. 2) 그 다음 relationship 에서 카디널리티 비율 제약조건을 정하였다. 카디널리티는 1:n, 1:1, n:m 이렇게 존재한다. 먼저 이 문제에서 1:1 관계는 존재하지 않았디. 그 다음 n:m 관계에는 (group<->mentor), (mentor<->tag), (mentee<->tag), (question<->tag), (email <->question) 가 있었고 나머지 관계들에는 모두 1:n관계를 설정하였다. 3) 그리고 나서 전체 또는 부분 참여 조건을 설정하였다. (total or partial) (ex. question-comment 관계에서 comment는 total 참여, question은 부분 참여이다.) 그리고 주어진 조건에 맞춰서 (MIN, MAX)를 입력하였다. (ex. question-answer의 관계를 보면 하나의 answer에 대하여 question은 min=max=1이므로 (1,1)이라 하였고 question에 대하여 answer은 min=0 max의 범위는 지정해 줄 수 없으므로 (0,N)이라 하였다).

최종 ER 다이어그램의 모습은 아래와 같다.



<PART2>

1. 문제 정의

PART2에서는 PART1에서 도식화한 ER 다이어그램을 requirement에 따라 스키마로 바꾸어 사이트 A의 데이터에 적합한 데이터베이스 스키마를 설계하여 데이터베이스 테이블을 실제로 생성한 후 데이터 입력까지를 목표로 한다.

위 스키마를 만들 때 제약조건은 PK(primary key-기본키), FK(외래키)를 고려하여 unique와 not null을 만족하게 구현하였으며, 상세한 설명은 아래에 적혀있다.

2. 스키마 설계 과정

PART 1에서 정의한 ER 다이어그램으로 관계형 스키마를 만들 때 다음과 같은 순서들로 고려하였다.

1) entity들을 먼저 관계형 스키마로 맵핑한다.

2) relationship 중 n:m 관계를 관계형 스키마로 맵핑한다..

3) relationship 중 1:n과 1:1 관계를 FK로 설정시킨다.

4) 존재한다면, multi valued attribute와 3차 관계를 관계형 스키마로 맵핑한다.(이 문제에서는 없으므로 적용하지 않았다.)

따라서 1) ~ 3) 순으로 다음 다이어그램을 관계형 스키마로 만들었다.

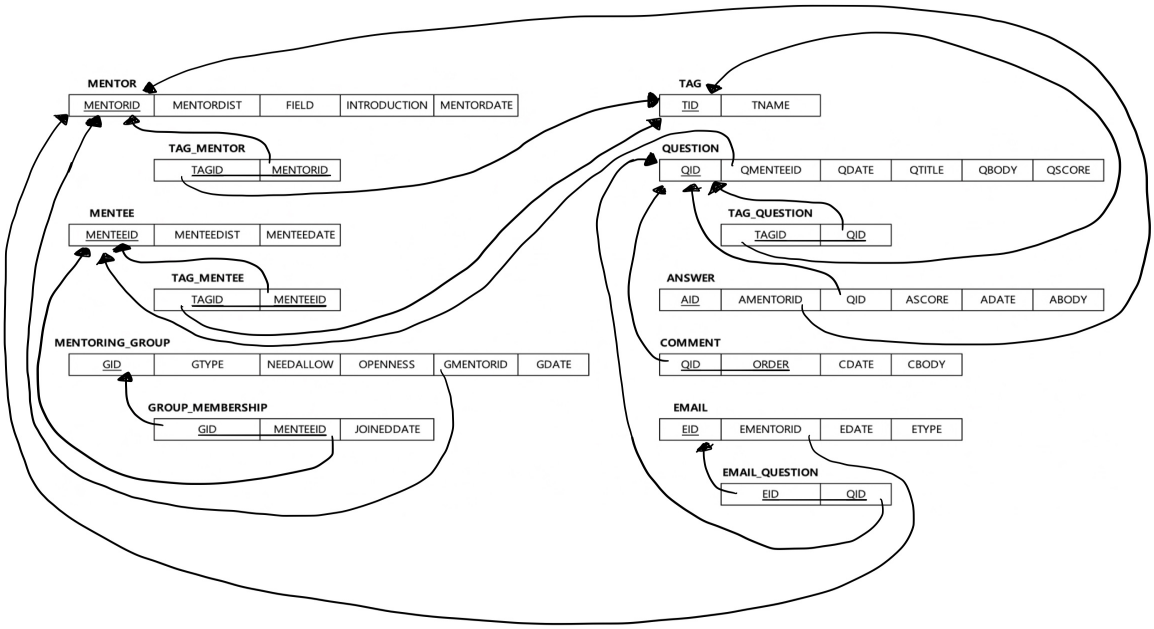
먼저 entity들을 관계형 스키마로 맵핑 시켰다. (mentor, mentee, question, answer, group, tag, email, comment) 여기서 regular entity인 mentor, mentee, question, answer, group, tag, email의 id는 pk(primary key)이다. (not null 하고 unique 하기 때문에) weak entity 타입인 comment 자체에는 pk가 될 수 있는 attribute가 존재하지 않기 때문에 weak entitny 타입의 부분 키(PK1, partial key)(order)와 참조하는 테이블의 기본키(PK)(question_id)를 복합 키로 생성하여 weak entity 타입의 기본키(primary key)를 만들었다. 따라서 comment의 PK는 question_id 와 order의 복합 키로 설정하였다.

그 후 relationship을 보면 관계형 스키마에서 1:1과 1:N 관계 타입은 한 쪽에 foreign key를 부여하지만, n:m 관계에서는 새로운 relation을 생성한다. 그리고 새로 생성된 relation은 두 entity에서 가져온 attribute에서 가져온 PK 2개를 FK(외래 키)로 삼는다. FK가 되는 2개의 attribute가 합쳐진 composite key가 relation의 primary key가 된다. 이 프로젝트에서 n:m관계에는 tag와 mentor, mentee, question과의 관계(tag_mentor, tag_mentee, tag_question), email과 question과의 관계(email_question), group과 mentee(group membership)의 관계가 이에 해당한다.

먼저 1) tag_mentor에서는 mentor 에서의 mentor_id와 tag에서 tag_id를 FK로 삼고 이 두 attribute가 합쳐진 복합키가 tag_mentor의 기본키가 된다. 2) tah_mentee에서는 mentee에서의 mentee_id와 tag에서 tag_id를 FK로 삼고 이 두 attribute가 합쳐진 복합키가 tag_mentee의 기본키가 된다. 3) tag_question에서는 question에서의 question_id와 tag에서 tag_id를 FK로 삼고 이 두 attribute가 합쳐진 복합키가 tag_question의 기본키가 된다. 4) email_question에서는 email에서의 email_id를 question에서의 question_id를 FK로 삼고 이 두 attribute가 합쳐진 복합키가 email_question에서의 기본키(PK)가 된다. 5) group membership 에서는 group에서의 gropu_id와 mentee에서의 mentee_id를 FK로 삼고 이 두 attribute가 합쳐진 복합 키가 group membership의 기본 키가 된다.

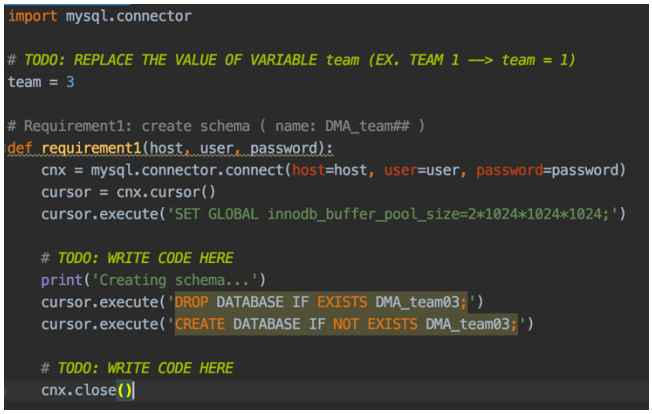
그리고 나머지 관계들 mentor group관계 등등은 FK로 처리함으로 써 설정한다. 예를 들어, mentor group 관계에서 total participate인 group 에게 FK를 준다. generate group mentor id는 이러한 결과물이다. 1:N, 1:1관계에서는 새로운 릴레이션을 만들지 않고 외래 키를 사용하여 처리하였다.

위에서 설명하였듯이 multi valued attribute와 3차 관계 타입은 여기서 고려할 필요가 없었고 최종 관계형 스키마는 아래와 같다.



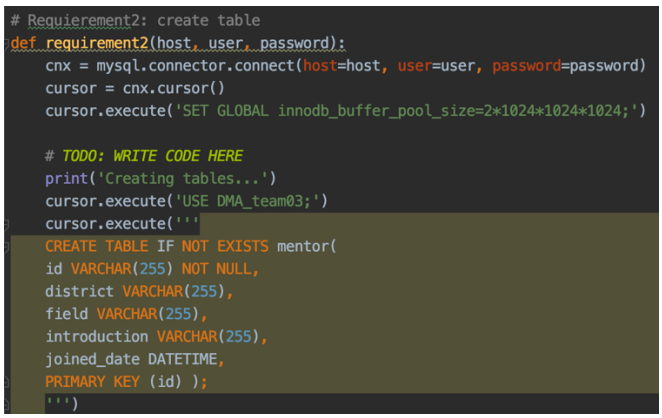
3. 코드 설명

Requirement1에서 CREATE DATABASE ~ IF NOT EXISTS 구문을 활용하여 DMA_team03 database를 만드는 코드를 구현하였다.



```python
import mysql.connector

# TODO: REPLACE THE VALUE OF VARIABLE team (EX. TEAM 1 --> team = 1)
team = 3

# Requirement1: create schema ( name: DMA_team## )
def requirement1(host, user, password):
    cnx = mysql.connector.connect(host=host, user=user, password=password)
    cursor = cnx.cursor()
    cursor.execute('SET GLOBAL innodb_buffer_pool_size=2*1024*1024*1024;')

    # TODO: WRITE CODE HERE
    print('Creating schema...')
    cursor.execute('DROP DATABASE IF EXISTS DMA_team03;')
    cursor.execute('CREATE DATABASE IF NOT EXISTS DMA_team03;')

    # TODO: WRITE CODE HERE
    cnx.close()
```

Requirement2에서는 관계형 스키마에 존재하는 모든 relation을 table로 저장하였고 여기서도 CREATE TABLE ~ IF NOT EXISTS를 사용해 table이 존재할 경우 실행되지 않도록 구현했다. PART 1의 requirement와 실제 데이터를 확인하면서 각 column의 data type과 constraint를 정해줬으며 관계형 스키마를 그릴 때 선택했던 PRIMARY KEY도 설정해주었다. 대부분의 column이 NOT NULL을 constraint로 가지고 있고 requirement에서 언급이 되었던 mentor의 district, field, introduction과 mentee의 district는 NULL을 허용해주었다. 또한 예외적으로 데이터에 NULL이 존재하는 tag의 name은 NULL을 허용해주었다.



```python
# Requierement2: create table
def requirement2(host, user, password):
    cnx = mysql.connector.connect(host=host, user=user, password=password)
    cursor = cnx.cursor()
    cursor.execute('SET GLOBAL innodb_buffer_pool_size=2*1024*1024*1024;')

    # TODO: WRITE CODE HERE
    print('Creating tables...')
    cursor.execute('USE DMA_team03;')
    cursor.execute('''
    CREATE TABLE IF NOT EXISTS mentor(
    id VARCHAR(255) NOT NULL,
    district VARCHAR(255),
    field VARCHAR(255),
    introduction VARCHAR(255),
    joined_date DATETIME,
    PRIMARY KEY (id) );
    ''')
```

Requirement3에서는 for문을 이용해 데이터가 저장된 csv파일을 차례로 불러오고 csv파일을 row단위로 읽어 INSERT INTO ~ VALUES 구문을 활용에 table에 저장해주었다. 이때, row에 빈 값('')이 존재하면 python에서 None으로 처리해서 db 상엔 NULL 값으로 저장되도록 설계하였다.

```
cursor.execute('USE DMA_team03;')
table_name = ['mentor', 'mentee', 'question', 'answer', 'comment', 'tag', 'tag_mentee',
              'tag_mentor', 'tag_question', 'mentoring_group', 'group_membership',
              'email', 'email_question']

for table in table_name:
    filepath = directory + '/' + table + '.csv'
    with open(filepath, 'r') as csv_data:
        next(csv_data, None) #skip the headers
        if table == 'mentor':
            for row in csv_data:
                # Change the null data
                row = row.strip().split(sep=',')
                if '' in row:
                    temp = []
                    for item in row:
                        if item == '':
                            item = None
                        temp.append(item)
                    row = temp
                cursor.execute('INSERT INTO ' + table + ' VALUES (%s,%s,%s,%s,%s)', row)
        elif table == 'mentee':
```

Requirement4에서는 관계형 스키마에서 화살표로 표시한대로 각각의 column이 서로를 참조하도록 FOREIGN KEY CONSTRAINT를 추가해주었다.

```
# Requirement4: add constraint (foreign key)
def requirement4(host, user, password):
    cnx = mysql.connector.connect(host=host, user=user, password=password)
    cursor = cnx.cursor()
    cursor.execute('SET GLOBAL innodb_buffer_pool_size=2*1024*1024*1024;')


    # TODO: WRITE CODE HERE
    print('Adding constraints...')
    cursor.execute('USE DMA_team03;')
    cursor.execute('ALTER TABLE question ADD CONSTRAINT FOREIGN KEY (mentee_id) REFERENCES mentee(id);')
    cursor.execute('ALTER TABLE answer ADD CONSTRAINT FOREIGN KEY (mentor_id) REFERENCES mentor(id);')
    cursor.execute('ALTER TABLE answer ADD CONSTRAINT FOREIGN KEY (question_id) REFERENCES question(id);')
    cursor.execute('ALTER TABLE comment ADD CONSTRAINT FOREIGN KEY (question_id) REFERENCES question(id);')
    cursor.execute('ALTER TABLE tag_mentee ADD CONSTRAINT FOREIGN KEY (tag_id) REFERENCES tag(id);')
    cursor.execute('ALTER TABLE tag_mentee ADD CONSTRAINT FOREIGN KEY (mentee_id) REFERENCES mentee(id);')
    cursor.execute('ALTER TABLE tag_mentor ADD CONSTRAINT FOREIGN KEY (tag_id) REFERENCES tag(id);')
    cursor.execute('ALTER TABLE tag_mentor ADD CONSTRAINT FOREIGN KEY (mentor_id) REFERENCES mentor(id);')
    cursor.execute('ALTER TABLE tag_question ADD CONSTRAINT FOREIGN KEY (tag_id) REFERENCES tag(id);')
    cursor.execute('ALTER TABLE tag_question ADD CONSTRAINT FOREIGN KEY (question_id) REFERENCES question(id);')
    cursor.execute('ALTER TABLE mentoring_group ADD CONSTRAINT FOREIGN KEY (mentor) REFERENCES mentor(id);')
    cursor.execute('ALTER TABLE group_membership ADD CONSTRAINT FOREIGN KEY (group_id) REFERENCES mentoring_group(id);')
    cursor.execute('ALTER TABLE group_membership ADD CONSTRAINT FOREIGN KEY (mentee_id) REFERENCES mentee(id);')
    cursor.execute('ALTER TABLE email ADD CONSTRAINT FOREIGN KEY (recipient_id) REFERENCES mentor(id);')
    cursor.execute('ALTER TABLE email_question ADD CONSTRAINT FOREIGN KEY (email_id) REFERENCES email(id);')
    cursor.execute('ALTER TABLE email_question ADD CONSTRAINT FOREIGN KEY (question_id) REFERENCES question(id);')

    # TODO: WRITE CODE HERE
    cnx.close()
```