Conceptual DB design & DB implementation

팀: 9조

이름: 강명훈 고우성 김원재 김영헌

Part1

1. 문제 정의

Project 1의 part1은 사이트 A가 이용하는 DB에 대한 ER diagram을 도식화하는 것을 목표로 한다. 사이트 A의 DB는 Requirement 7가지를 만족한다. 각 Requirement를 나누어 해석하여 ER diagram을 그리는 것을 목표로 하였다.

A는 entity, relationship에 관한 정보 B는 attribute에 관한 정보, C는 derived attribute에 관한 정보, D는 cardinality에 관한 정보로 크게 문장을 해석하였다. DB implementation에 해당하는 부분은 따로 해석하지 않았다.

R1-1.

A. **사용자**는 서비스에 대한 **리뷰**를 남기거나 **콜렉션**에 추가할 수 있으며 서로 **팔로우**할 수 있다.

B. 사용자에 대한 정보로는 사이트에서 부여한 **고유 id**, 사용자가 설정한 **닉네임**, 사용자가 설정한 **활동 지역의 고유 id**가 있다.

C. 사용자가 작성한 **리뷰의 수**, **팔로워의 수**, **팔로잉 수**, **콜렉션에 추가한 식당 수**, **평균 리뷰 점수**가 파생정보이다.

D. 사용자 활동 지역의 설정은 필수가 아니며 사용자 **한명당 하나의 지역**만을 설정할 수 있다. 팔로우에 정보는 팔로우한 사용자의 고유 id, 팔로우 대상 사용자의 고유 id로 이루어져 있다.

설계 시 초기 디자인: user entity에 review, collection, follow 가 entity혹은 relationship로 존재하며, D에 의해 follow는 recursive한 relationship으로 설계된다, B,C에 의해 파생정보와 stored attribute를 설정할 수 있다

R1-2.

A. **식당**에 대한 정보를 가지고 있다.

B. 식당에 대한 정보로는 **식당 고유 id**, **식당 이름**, **점심 최소 가격**, **저녁 최소 가격**, **저녁 최대 가격**, **식당이 위치한 지역의 고유 id**, **식당이 속한 카테고리의 고유 id**가 있다.

C. **평균 리뷰점수**, **리뷰 수**, 사용자의 **콜렉션에 추가된 횟수**가 파생정보로 관리되어야 한다.

D. 식당은 필수적으로 **하나의 카테고리/지역에 속해야** 하며 하나 이상의 카테고리/지역에 속할 수 없다.

설계 시 초기 디자인: restaurant entity에 collect, review에 대한 entity혹은 relationship이 존재하며

B,C에 의해 파생정보와 stored attribute 설정된다. 지역의 고유 id를 attribute로 관리하기에는 location_id는 key attribute이며 dataset이 다량 존재하므로 user와 restaurant entity에서 참조하는게 맞다고 판단하였다.

R1-3.

A. 사용자들의 **리뷰** 정보를 가지고 있다.

B. 리뷰에 대한 정보로는 **리뷰에 대한 고유 ID, 리뷰 본문, 작성 날짜, 리뷰를 작성한 사용자의 ID,** 종합점수, **맛 점수, 서비스 점수, 분위기 점수, 식당의 고유 ID**가 있다.

C. **종합점수**는 세점수의 평균으로 계산된다. 종합점수는 사용자가 직접 입력하게 된다.

설계 시 초기 디자인: review entity에 user와 restaurant entity가 연결된다. B,C에 의해 파생정보와 stored attribute를 설정하였다. 종합점수는 stored attribute로 판단하였다. 왜냐면 RBD에 stored attribute가 column으로 들어가 있기 때문이다.

R1-4.

A. 식당의 **메뉴정보**를 가지고 있다.

B. 메뉴 정보로는 **메뉴 이름, 최소가격, 최대가격, 메뉴를 판매하는 식당의 고유 id**가 있다.

D. 서로 **다른 식당**에서 **같은 이름을 갖는 메뉴**를 판매할 수 있으나 **하나의 식당**에서는 **중복된 이름을 갖는 여러 메뉴를** 판매하지 않는다.

설계 시 초기 디자인: menu entity에 restaurant entity가 연결된다. 이때 weak entity임을 알 수 있다. 식당 entity에 종속된 entity이며 D에 의해 알 수 있듯이 menu name 단독으로 엔티티를 식별할 수가 없기 때문이다. 또한 B로 stored attribute를 설정하였다.

R1-5.

A. 식당과 메뉴에 대한 **블로그 포스트** 정보를 갖고 있다.

B. **포스트의 고유 id, 포스트의 제목, 포스트의 블로그 URL, 포스트 작성 날짜, 포스트에서 다루고 있는 식당의 고유 id, 포스트에서 다루고 있는 메뉴의 이름**

C. 하나의 포스트에서 여러 식당을 다룰 수 없으나 한 식당의 여러 메뉴를 다룰 수는 있다.

설계 시 초기 디자인: post entity에 restaurant과 menu entity가 연결된다. 이때 B에 의해 post의 stored attribute 설정하였다. D는 cardinality에 관련된 정보이다.

R1-6.

A. **지역정보**를 갖고 있다.

B. **지역의 고유 id, 지역의 이름**

C. **해당지역을 활동지역으로 설정한 유저의 수, 해당지역에 위치한 식당의 수**

설계 시 초기 디자인: 지역 정보 entity는 user entity와 연결하였다. restaurant와 B, C가 각각 stored attribute와 파생정보로 설정되었다.

R1-7

A. **카테고리 정보**를 갖고 있다.

B. **카테고리의 고유 id, 카테고리의 이름**

C. **카테고리에 속하는 식당의 수**

설계 시 초기 디자인: 카테고리 entity는 restaurant entity와 연결된다. B, C가 각각 stored attribute와 파생정보로 설정된다.

위와 같이 대략적으로 설계하여 도식화를 진행한다. 이때 각 entity에 저장되는 stored attribute는 만약 다른 entity에 존재한다면 가장 타당한 entity에 할당하였다.

2. 도식화 과정

먼저 위 초기 디자인을 바탕으로 entity를 설정하였다. 사이트 A의 경우 **user**가 review를 남기고 restaurant를 collect하고, **review**가 restaurant에 관련되어 user에 의해 적히고, **location**이 user와 restaurant의 장소를 표시하고, **menu**가 restaurant에 속해있고, **post**위에 restaurant과 restaurant의 menu가 표시되고 **category**는 해당 restaurant이 어디 속해있는지 표한다. **Restaurant**는 위 entity들에 모두 관련되어있기 7가지 entity가 기본 entity라 할 수 있다. 이중 review는 total_score, survice, taste, mood라는 attribute를 갖는다. 이때 total_score라는 attribute가 composite, derived attribute라는 것에 관하여 다양한 의견이 있었지만 본 project의 제약조건에 의해 이를 single, stored attribute로 설정했다. 그 근거로는 종합 점수가 직접 입력될 수 있다는 점에서 derived라고 할 수 없고 또한 RDB에도 존재하기에 stored라고 판단했다. 그리고 composite는 attribute value의 조합이 unique해야 하기 때문에 composite이 아니라고 판단했다. user, review, location, menu, post, category, restaurant이라는 7개의 entity를 정리하고 stored attribute와 파생정보를 정리하면 아래 그림과 같다.

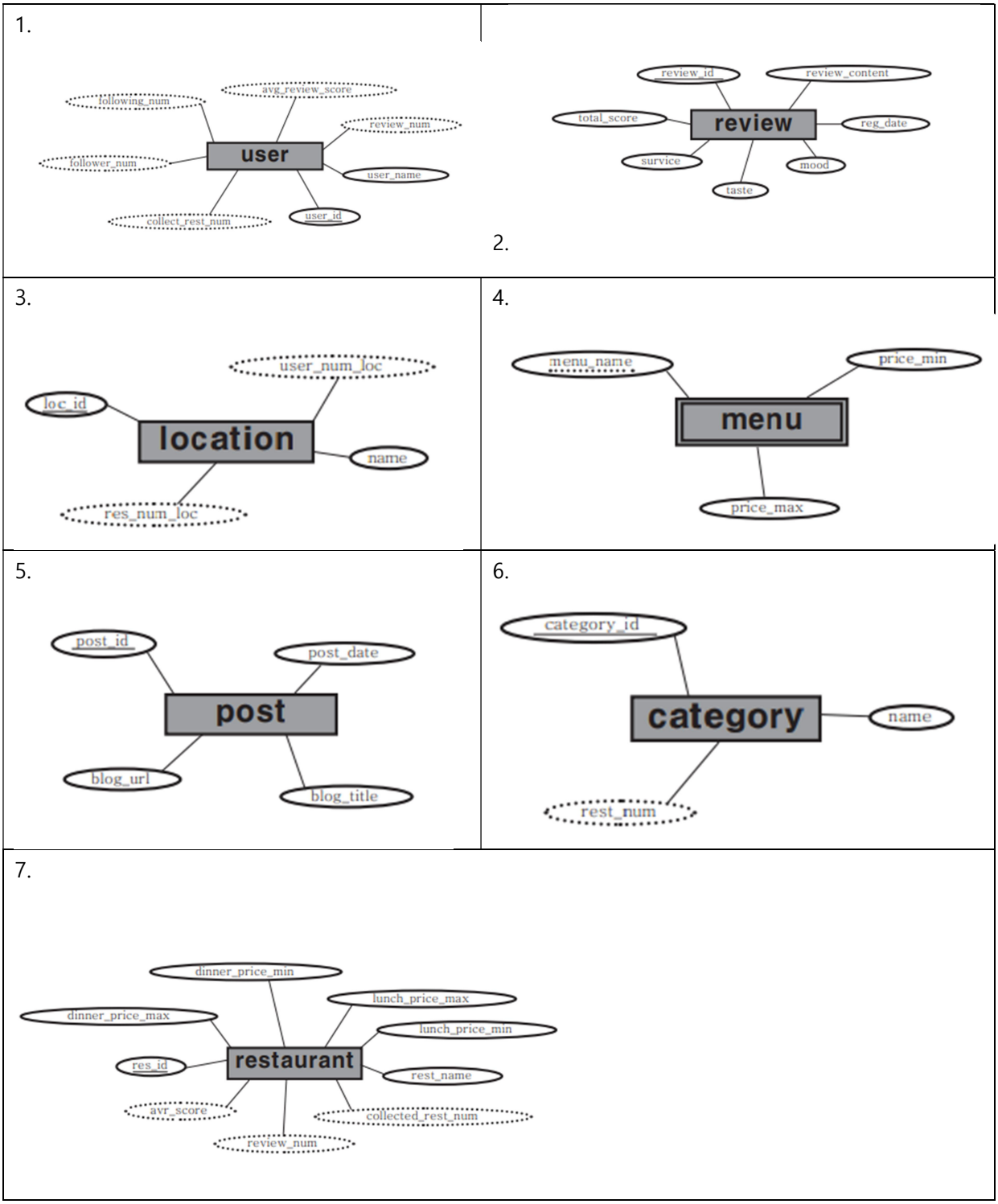

Fig.1. 왼쪽상단부터 순서대로 1.“user 초기 design”, 2.“review 초기 design”, 3. “location 초기 design”, 4. “ menu 초기 design”, 5.“post 초기 design”, 6. “category 초기 design”, 7. “restaurant 초기 design”

3. Relationship 설정, cardinality 설정 및 최종 ER 다이어그램 제시

다음으로는 요구사항에 맞게 entity 기준으로 여러 relationship을 만들고, cardinality를 설정하였다.

User의 경우 Follow relationship을 기준으로 recursive하게 설계되었는데 이는 M:N 관계이다. 한 user가 여러명을 following하고 한 user가 여러명에게 followed 당하기 때문이다. 다음으로 User와 location에 관한 관계는 live라는 relationship을 통해 연결하였고 location 한곳에 여러명의 user가 살수 있고 user는 location 1곳에만 살고 활동 지역설정은 필수가 아니므로 partial participation인 N:1 관계이다. R1-1에서"사용자 활동 지역의 설정은 필수가 아니며 사용자 **한명당 하나의 지역**만을 설정할 수 있다"를 통해 알 수 있다.

User는 review를 적기에 write라는 relationship으로 연결할 수 있고 user는 review를 여러 개 적을 수 있고 review는 user 한 명에 의해 적히기에 1:N 으로 연결 할 수 있다. 이때 review는 작성한 user 가 무조건 있어야하고 user는 review를 적지않을수도 있기에 N만 total participation이다.

다음으로 restaurant를 기준으로 ER을 관찰하였다. 먼저 restaurant와 location의 관계는 locate로 설정하였다. 한 지역에 여러 개의 restaurant가 있을 수 있고 restaurant는 1개의 location에 속한다 또한 restaurant는 무조건 하나의 location에는 속해야 하기에 1:N이고 N은 total participation임을 알 수 있다. "식당은 필수적으로 **하나의 카테고리/지역에 속해야** 하며 하나 이상의 카테고리/지역에 속할 수 없다." 라는 requirement 1-2에 의해 위를 알 수 있다.

다음으로 user가 restaurant를 collection할 수 있기에 collect relationship으로 둘을 연결하였다. 한 user가 여러 restaurant을 collection하고 한 restaurant가 여러 user에게 collected 당하기 때문이다. 그 다음으로 review에 대해 살펴보면 user가 review를 적을 때와 같은 이유로 review와 restaurant의 관계는 N:1, N은 total participation임을 알 수 있다.

다음으로 category에 restaurant이 속하기에 involved라는 관계로 설정할 수 있다. 한 Category에 대해 N개의 restaurant가 속하고 restaurant는 필수적으로 1개만의 Category에 속해야 하므로 N:1임을 알 수 있었다. 이때 N은 total participation이다. "식당은 필수적으로 **하나의 카테고리/지역에 속해야** 하며 하나 이상의 카테고리/지역에 속할 수 없다"라는 requirement 1-2에 의해 위를 알 수 있다.

다음으로 restaurant는 menu를 sell 할 수 있고 이때 sell을 identifying relationship으로 설정했다. 이때 restaurant는 N개의 menu를 팔 수 있고 weak entity이므로 1:N의 cardinality를 가진다. 이때 있는 menu는 모두 등록되었으므로 1:N 에 N이 total participation이다.

마지막으로 post에 대해 relationship을 판단하였다. Menu에 대한 Relationship은 post_menu로 설정했다. post에서 여러 post가 한 menu를 소개할 수 있고 여러 menu들이 한 post에 등록될 수 있다.

또한 post중 menu가 한 개도 없는 post는 없으므로 post 와 post_menu는 total participation으로 이루어져있으며 M:N이다.

그리고 post가 restaurant에 upload되는데 post에 한 개의 restaurant만 등록되고 한 restaurant에 관한 여러 post들이 등록되므로 1:N 임을 알 수 있다. 이때 post는 restaurant를 하나는 포함하므로 N은 total participation이다.

Post와 menu, restaurant의 관계를 ternary relationship으로 설정할 수 있었지만 이를 binary relationship으로 설정했다. 이는 post entity가 RDB에서 relation으로 갔을 때 restaurant column이 포함되는 것을 ternary는 설명할 수 없기 때문이다.

이를 통해 모든 relationship과 cardinality를 알 수 있었고 이를 정리하면 아래 그림과 같다.

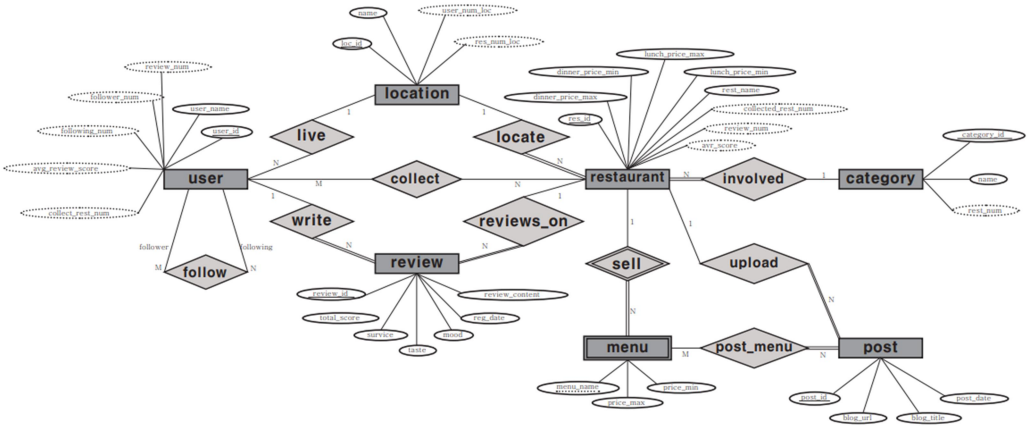

Fig.2. 최종 ER diagram

Part2

1. 문제 정의

위의 Part1에서 제시된 조건을 바탕으로 사이트 A의 데이터에 적합한 데이터베이스 스키마를 설계하여, 데이터베이스 테이블을 실제로 생성한 후 데이터를 입력하고 제약 조건까지 거는 것을 목표로 한다.

해당 프로그램은 Python과 MySQL을 사용하여 구현하고, Python에서 mysql-connector-python, csv 이외의 별도 라이브러리는 사용할 수 없다.

(R2-1) 사이트 A의 데이터를 활용하기에 앞서 이를 MySQL 상에 저장해야 한다. 이를 위해 먼 저 DMA_team##의 이름을 가지는 schema를 생성해야 한다. 예를 들면, 1조의 schema 명은 DMA_team01이다. 이 때 schema가 존재할 경우 생성 과정을 다시 수행하지 않아야 한다.

(R2-2) Schema를 설계한 이후에는 데이터를 저장하기 위한 table을 생성해야 한다. 생성하는 table과 column 이름과 순서는 주어진 데이터셋의 table 및 column과 일치해야 한다. 0 또는 1의 값을 가지는 column은 'TINYINT(1)'로, INTEGER type은 'INT(11)'로, FLOAT type은 'DECIMAL(11)로, STRING type은 'VARCHAR(255)'를 이용하여 생성한다. 255자를 넘는 경우 'LONGTEXT'를 이용하여 생성한다. 그 외 날짜시간은 'DATETIME'을 통해 생성한다. 이 때 table이 존재할 경우 생성 과정 을 다시 수행하지 않아야 한다. (R2-2)에서는 foreign key 조건을 작성하지 않고 (R2-3)에서 데이터 입력 후 foreign key 조건을 추가한다.

(R2-3) 생성된 테이블에 데이터를 저장해야 한다. 데이터는 csv파일로 주어지며 이를 직접 변형 해선 안 된다.

(R2-4) 해당 데이터베이스 스키마에 foreign key 조건들을 반영해주어야 한다.

2. 스키마 및 코드 설명

(1) 라이브러리 설정

```
import mysql.connector
import csv
```

mysql.connector과 csv library만 import하고 이외에는 사용하지 않는다.

(2) Requirement1 - Schema 생성

```
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;')
    print('Creating schema...')

    # TODO: WRITE CODE HERE
    # Check if the schema already exists
    cursor.execute("SHOW DATABASES LIKE 'DMA_team09'")
    # LIKE keyword => Find "DMA_team09" database
    # SHOW DATABASES : Show all databases in the server

    result = cursor.fetchall()
    #fetchall() : fetch all rows of a query result, returning them as a list
of tuples.
    # If no more rows are available, it returns an empty list.

    if result:
        print('Schema already exists')
    else:
        cursor.execute('CREATE DATABASE IF NOT EXISTS DMA_team09;')
        print('Schema created')

    # TODO: WRITE CODE HERE

    cursor.close()
```

SHOW DATABASES LIKE 'DMA_team09' 명령어는 'DMA_team09'라는 이름의 데이터베이스가 있는지 없는지를 구하는 쿼리다. 이 쿼리를 cursor.fetchall()로 받아와서 존재하면 'Schema already exists'를 출력하고, 없으면 'CREATE DATABASE IF NOT EXISTS DMA_team09'를 실행하는 조건문 형식으로 출력되도록 했다.

(3) requirement2 - 데이터 테이블 생성

1) 릴레이션 이름 별로, 데이터 삽입 시 column명, datatype, NOT NULL 여부, Primary key 표시(노란색 음영, 각 릴레이션에 2개 이상 칠해질 경우 각 column이 combination으로 엮여 PK로 설정됨.)

User	Restaurant		Review	
user_id VARCHAR(255) NOT NULL	**restaurant_id** VARCHAR(255) NOT NULL	**dinner_price_min** INT(11)	**review_id** INT(11) NOT NULL	**taste_score** DECIMAL(11)
user_name VARCHAR(255) NOT NULL	**restaurant_name** VARCHAR(255) NOT NULL	**dinner_price_max** INT(11)	**review_content** VARCHAR(255)	**service_score** DECIMAL(11)

region INT(11)	**lunch_price_min** INT(11)	**location** INT(11) NOT NULL	**reg_date** DATETIME	**mood_score** DECIMAL(11)
	lunch_price_max INT(11)	**category** INT(11) NOT NULL	**user_id** VARCHAR(255) NOT NULL	**restaurant** VARCHAR(255) NOT NULL
			total_score DECIMAL(11) NOT NULL	

Menu	Post_Menu	Location	Post	
menu_name VARCHAR(255) NOT NULL	**post_id** INT(11) NOT NULL	**name** VARCHAR(255) NOT NULL	**blog_title** VARCHAR(255)	**restaurant** VARCHAR(255) NOT NULL
price_min INT(11)	**menu_name** VARCHAR(255) NOT NULL	**location_id** INT(11) NOT NULL	**blog_URL** VARCHAR(255) NOT NULL	**post_id** INT(11) NOT NULL
price_max INT(11)	**restaurant** VARCHAR(255) NOT NULL		**post_date** DATETIME NOT NULL	
restaurant VARCHAR(255) NOT NULL				

Follow	Collection	Category
followee_id VARCHAR(255) NOT NULL	**user_id** VARCHAR(255) NOT NULL	**name** VARCHAR(255) NOT NULL
follower_id VARCHAR(255) NOT NULL	**restaurant_id** VARCHAR(255) NOT NULL	**category_id** INT(11) NOT NULL

2) 테이블 별 생성 코드, Datatype 결정 근거, NOT NULL 설정 기준

#1 User relation

```
cursor.execute('''
CREATE TABLE IF NOT EXISTS User(
user_id VARCHAR(255) NOT NULL,
user_name VARCHAR(255) NOT NULL,
region INT(11),
```

```
primary key(user_id));
''')
```

user_id는 Primary Key(이하 PK)로 설정하므로 제약조건에 맞도록 문자형 VARCHAR(255) NOT NULL 조건을 추가하였다. user_name column의 경우, 이 역시 문자형 VARCHAR(255)으로 삽입하며 주어진 User excel csv 파일을 확인한 결과, 빈 값이 없으므로 NOT NULL 조건을 추가하였다. region은 사용자가 임의로 추가할 수 있으므로 NOT NULL 조건을 추가하지 않았다. primary key(user_id)는, user_id column이 PK가 되도록 마지막에 추가해두었다.

#2 Restaurant relation

```
cursor.execute('''
CREATE TABLE IF NOT EXISTS Restaurant(
restaurant_id VARCHAR(255) NOT NULL,
restaurant_name VARCHAR(255) NOT NULL,
lunch_price_min INT(11),
lunch_price_max INT(11),
dinner_price_min INT(11),
dinner_price_max INT(11),
location INT(11) NOT NULL,
category INT(11) NOT NULL,
primary key(restaurant_id));
''')
```

restaurant_id는 PK로 설정하므로 제약조건에 맞도록 문자형 VARCHAR(255), NOT NULL 조건을 추가하였다. restaurand_name은 문자형 VARCHAR(255)로 들어가도록 했고, 이름 없는 식당은 없는 데다가, Restaurant.csv 데이터 파일의 restaurant_name column의 NOT NULL 조건을 확인한 결과 null 값이 없으므로 NOT NULL을 추가하였다.

lunch_price_min, lunch_price_max, dinner_price_min, dinner_price_max column은 각각 판매 가격에 대한 것이다. 가격은 최대가격과 최소가격이 존재하지 않을 수도 있고, 저녁에만 식당을 열 수도, 점심에만 식당을 열수도 있기에 받아들이는 정수형 INT(11)만 설정하고 NOT NULL 조건을 설정하지 않았다.

location과 category는, 식당은 필수적으로 하나의 지역/카테고리에 속해야 한다는 Part I의 R1-3 조건에 따라 NOT NULL로 설정하였다. PK는 restaurant_id가 되도록 마지막에 조건을 추가하였다.

#3 Review relation

```
cursor.execute('''
CREATE TABLE IF NOT EXISTS Review(
review_id INT(11) NOT NULL,
review_content VARCHAR(255),
reg_date DATETIME,
user_id VARCHAR(255) NOT NULL,
total_score DECIMAL(11) NOT NULL,
```

```
taste_score DECIMAL(11),
service_score DECIMAL(11),
mood_score DECIMAL(11),
restaurant VARCHAR(255) NOT NULL,
primary key(review_id));
''')
```

review_id는 PK로 설정하므로 제약조건에 맞도록 정수형 조건 INT(11)로 받아들이고 NOT NULL 조건을 설정하였다.

review_content는 문자형 조건 VARCHAR(255)로 설정했다. 리뷰 내용은 아무런 내용이 있지 않을 수 있다고 판단하여 NOT NULL 조건을 추가하진 않았다.

reg_date는 DATETIME 형식으로 들여오고, 날짜가 적히지 않은 tuple이 존재하여 NOT NULL 조건을 걸지 않았다.

user_id는 User relation의 PK을 외래키(이하 FK)로 쓰고, 문자형 VARCHAR(255)로 설정하였으며 user_id column에 null한 값이 없어 NOT NULL 조건을 걸었다.

total_score는 Part1의 R1-3 조건에 따라서, 반드시 입력해야 하는 값이므로 DECIMAL(11)로 받아들이고 NOT NULL 조건을 설정하였다.
taste_score, service_score, mood_score는 존재하지 않을 수 있는 값이므로, 값들이 실수 형태(float)로 되어 있으므로 DECIMAL(11) 값만 받아온다.

restaurant는 Restaurant relation의 PK을 외래키(이하 FK)로 쓰고, 문자형 VARCHAR(255)로 설정하였으며 restaurant column에 null한 값이 없어 NOT NULL 조건을 걸었다.

#4 Menu relation

```
cursor.execute('''
CREATE TABLE IF NOT EXISTS Menu(
menu_name VARCHAR(255) NOT NULL,
price_min INT(11),
price_max INT(11),
restaurant VARCHAR(255) NOT NULL,
primary key(restaurant, menu_name));
''')
```

원래 Menu relation은 ER Diagram 상에서 weak entity였다. Relation Database model (이하 RDB)로 변환하는 과정에서 Menu relation으로 변환하였는데, menu_name만으로는 각각의 tuple을 식별 불가능하므로, restaurant_id를 갖고 온 것이다. PK는 따라서 restaurant와 menu_name의 combination으로 설정한다.

menu_name은 문자형이므로 VARCHAR(255) 으로 받아들이고, 이름 없는 메뉴는 없으며, 또한 combination 하여 PK로 설정할 것이므로 NOT NULL로 설정하였다.

price_min, price max는, 가격이 정수형이므로 INT(11)으로 설정하였다. 다만 최대가격(시간 및 날짜에 따라 가격이 변동하지 않는 경우)과 최소가격이 존재하지 않을 수 있으므로 (무료) NOT NULL을 추가하지 않았다.

restaurant는 combination으로 PK가 되므로 NOT NULL 형태를 띠므로 NOT NULL을 추가하였다.

#5 Post_Menu relation

```
cursor.execute('''
CREATE TABLE IF NOT EXISTS Post_Menu(
post_id INT(11) NOT NULL,
menu_name VARCHAR(255) NOT NULL,
restaurant VARCHAR(255) NOT NULL,
primary key(post_id,restaurant,menu_name));
''')
```

Post_Menu relation은, 원래 ERD에서는 Post entity와 Menu weak entity를 잇는 카디널리티 M:N의 relationship였다. RDB로의 변환 과정에서 relation으로 바뀌었고, 원래 각각 엔터티에 연결된 PK의 combination이 PK가 된다.

따라서 Post Relation의 PK, Menu Relation의 PK를 넣는 형식 그대로 받아오고, PK는 조합을 (post_id, restaurant, menu_name) 순으로 설정하여 넣는다.

#6 Location relation

```
cursor.execute('''
CREATE TABLE IF NOT EXISTS Location(
name VARCHAR(255) NOT NULL,
location_id INT(11) NOT NULL,
primary key(location_id));
''')
```

Location.csv 데이터를 확인해 보니, name, location_id가 모두 채워져 있는 데다가, 한국의 각 행정구역 이름과 고유 번호가 각각 다 적혀 있어야 Location 정보가 채워졌다고 할 수 있으므로 NOT NULL로 삽입하고, 각각의 형태는 문자형, 정수형이므로 VARCHAR(255), INT(11)로 삽입하도록 한다. Location relation에서는 PK를 location_id로 설정한다.

#7 Post relation

```
cursor.execute('''
CREATE TABLE IF NOT EXISTS Post(
blog_title VARCHAR(255),
blog_URL VARCHAR(255) NOT NULL,
post_date DATETIME NOT NULL,
restaurant VARCHAR(255) NOT NULL,
post_id INT(11) NOT NULL,
primary key(post_id));
''')
```

Post relation은 blog_title이 존재하지 않는 경우가 존재한다. 따라서 NOT NULL 조건 없이 VARCHAR(255) 형으로 삽입될 수 있도록 하였다.

blog_URL, post_date column에는 Post.csv를 확인하니 NULL 값이 없으므로 NOT NULL 조건을 설정하였다. blog_URL는 column에 해당하는 내용이 주소라서 문자형이므로 VARCHAR(255)형, post_date는 DATETIME형으로 삽입한다.

post_id는 Post relation에서 정수형 INT(11)으로 삽입하며, NULL 값이 없으므로 NOT NULL 조건을 설정한다. Post relation에서는 post_id를 PK로 설정하였다.

#8 Follow relation

```
cursor.execute('''
CREATE TABLE IF NOT EXISTS Follow(
followee_id VARCHAR(255) NOT NULL,
follower_id VARCHAR(255) NOT NULL,
primary key(followee_id, follower_id));
''')
```

Follow relation은 어느 User가 어느 User를 follow하고 follow 당하는지 1:1 대응 관계를 튜플로 저장하는 것과 같다. followee_id, follower_id는 문자형, VARCHAR(255)형, combination으로 PK를 짜므로 무결성 제약에 따라 NOT NULL으로 설정하였다.

PK로는 followee_id, follower_id의 combination으로 하였다.

#9 Collection relation

```
cursor.execute('''
CREATE TABLE IF NOT EXISTS Collection(
```

```
user_id VARCHAR(255) NOT NULL,
restaurant_id VARCHAR(255) NOT NULL,
primary key(user_id, restaurant_id));
''')
```

Collection relation은 원래 ER Diagram에서 user entity와 restaurant entity를 잇는 카디널리티 m:n의 relationship이었으나 RDB로 바뀌면서 Relation으로 mapping 된 것이다. 어느 user가 어느 restaurant를 collect 하는지 대응 관계를 튜플로 저장하는 형태이다. 두 column이 combination되어 PK가 되므로 NOT NULL형으로 저장한다. 또한 문자형이므로 모두 VARCHAR(255)로 설정하여야 한다.

PK는 user_id, restaurant_id의 combination으로 저장한다.

#10 Category relation

```
cursor.execute('''
CREATE TABLE IF NOT EXISTS Category(
name VARCHAR(255) NOT NULL,
category_id INT(11) NOT NULL,
primary key(category_id));
''')
```

Category relation은 category별로 name, 각각의 category_id가 저장되어 있다.

name은 문자형이므로 VARCHAR(255)로 하고, 각각의 column은 null 조건이 없는 데다가, 음식을 파는 식당의 카테고리별로 그 이름이 있어야만 하므로 NOT NULL 조건을 부여했다.

category_id는 정수형이므로 INT(11), PK가 되어야 하므로 NOT NULL 조건을 부여했다.

(4) requirement3 - 데이터 삽입

1) 기본 코드 형식

```
cursor.execute('USE DMA_team09')

    # 1. User table insertion
    filepath1 = directory + '/User.csv'
    with open(filepath1, 'r', encoding='utf-8') as User_data:
        for row1 in User_data.readlines()[1:]:
            # create readlines() with [1:2] to output only as many rows as we
want to try / test similarly for smaller files
```

```
            # When running as it is, a lot of data is inserted and it takes
time,
            # set something else to [1:2] when testing code, and run it with
full data afterward
            row1 = row1.strip().split(',')
            for idx, data in enumerate(row1):
                if data == '':
                    row1[idx] = 'null'
                # convert string to int if possible
                try:
                    row1[idx] = int(data)
                except ValueError:
                    pass
            row1 = tuple(row1)
            sql1 = 'INSERT IGNORE INTO User VALUES {}'.format(row1)

           # Reason for IGNORE: If a record doesn't exist, insert it. If it
exists, ignore it.
            # To prevent errors caused by duplicate insertions during
iterations while testing code.

            sql1 = sql1.replace('\'null\'', 'null')
            cursor.execute(sql1)
    cnx.commit()

    print("User tuple insertion is done")
```

Requirement3에선 주어진 dataset을 읽어 데이터베이스에 저장하는 것이 목표이다. csv파일에 접근하여 한 줄씩 읽어서 각 튜플을 저장했고, 콤마 기준으로 나누어 컬럼별로 attribute를 저장하였다. 이 과정에서 Location 테이블의 location_id와 같은 정수형 타입은 string으로 저장하는 것이 아닌 정수형으로 변환하여 저장하였다. 정수형 타입이 아닌 자료의 경우 오류 던지기를(except) 활용하여 해당 코드를 pass하게 된다.

한편, INSERT IGNORE INTO "TABLE명" 이런 식으로 IGNORE을 추가한 이유는, 코드를 실행하고 테스트하는 데 있어서, 삽입 시 충돌이 발생하여 반복적으로 테이블 및 컬럼을 삭제하는 등의 번거로움을 줄이고, csv 파일을 갱신하여 새로 입력하더라도 이를 무시하고 삽입이 가능하고, 중복 튜플이 있더라도 오류 없이 한 번만 삽입이 될 수 있도록 하였. 이는 좀 더 robust한 코드가 될 수 있도록 하는 것이다.

```
                # convert string to int if possible
                # datetime type => str type
```

```
            if idx == 0:
                try:
                    row3[idx] = int(data)
                except ValueError:
                    row3[idx] = 'null'
            elif 4 <= idx <= 7:
                try:
                    row3[idx] = float(data)
                except ValueError:
                    row3[idx] = 'null'
```

한편, Review table의 점수들에 대한 attribute의 경우 정수형이 아닌 float 타입으로 변환하여 값을 입력하였다.

```
        if data == '':
            row4[idx] = 'null'
        # convert string to int if possible
        try:
            row4[idx] = int(float(data))
    # I need to insert as INT, but the csv has prices in decimal,
    # so I convert to float and then int
        except ValueError:
            pass
```

Menu table의 경우 Menu.csv 파일에 저장된 가격 정보들이 소수점을 가지고 있지만 정수값이라고 할 수 있었고, 편의성을 위해 값을 float 타입으로 변환 후, 정수형 타입으로 다시 변환하여 입력하였다.

(4) requirement04

Requirement4에선 foreign key constraint에 대한 내용을 코드로 작성하였다. 각 코드에 대한 설명을 가독성을 위해 다음과 같이 통일하겠다.

Table a(attribute A) -> Table b(attribute B) == Relationship C

테이블 a의 attribute A(foreign key)가 테이블 b의 attribute B를 refer하고 이는 ER diagram에서의

Relationship C에 해당됨.

```
    # 1. User's foreign key
    cursor.execute("ALTER TABLE User ADD CONSTRAINT FOREIGN KEY (region)
REFERENCES Location(location_id)")
    print("1-1 done")
```

User(region) -> Location(location_id) == live

```
    # 2. Restaurant's foreign key
    cursor.execute("ALTER TABLE Restaurant ADD CONSTRAINT FOREIGN KEY
(location) REFERENCES Location(location_id)")
    print("2-1 done")
    cursor.execute("ALTER TABLE Restaurant ADD CONSTRAINT FOREIGN KEY
(category) REFERENCES Category(category_id)")
    print("2-2 done")
```

Restaurant(location) -> Location(location_id) == locate

Restaurant(category) -> Category(category_id) == involved

```
    # 3. Review's foreign key
    cursor.execute("ALTER TABLE Review ADD CONSTRAINT FOREIGN KEY (user_id)
REFERENCES User(user_id)")
    print("3-1 done")
    cursor.execute("ALTER TABLE Review ADD CONSTRAINT FOREIGN KEY (restaurant)
REFERENCES Restaurant(restaurant_id)")
    print("3-2 done")
```

Review(user_id) -> User(user_id) == write

Review(restaurant) -> Restaurant(restaurant_id) == reviews_on

```
    # 4. Post's foreign key
    cursor.execute("ALTER TABLE Post ADD CONSTRAINT FOREIGN KEY (restaurant)
REFERENCES Restaurant(restaurant_id)")
    print("4-1 done")
```

Post(restaurant) -> Restaurant(restaurant_id) == upload

```
    # 5. Menu's foreign key
    cursor.execute("ALTER TABLE Menu ADD CONSTRAINT FOREIGN KEY (restaurant)
REFERENCES Restaurant(restaurant_id)")
    print("5-1 done")
```

Menu(restaurant) -> Restaurant(restaurant_id) == sell

```
    # 6. Collection's foreign key
    cursor.execute("ALTER TABLE Collection ADD CONSTRAINT FOREIGN KEY
(restaurant_id) REFERENCES Restaurant(restaurant_id)")
    print("6-1 done")
    cursor.execute("ALTER TABLE Collection ADD CONSTRAINT FOREIGN KEY (user_id)
REFERENCES User(user_id)")
    print("6-2 done")
```

Collection(restaurant_id) -> Restaurant(restaurant_id) == collect

Collection(user_id) -> User(user_id) == collect

```
    # 7. Post_Menu's foreign key
    cursor.execute("ALTER TABLE Post_Menu ADD CONSTRAINT FOREIGN KEY (post_id)
REFERENCES Post(post_id)")
    print("7-1 done")
    cursor.execute("ALTER TABLE Post_Menu ADD CONSTRAINT FOREIGN KEY
(restaurant, menu_name) REFERENCES Menu(restaurant, menu_name)")
    print("7-2 done")
```

Post_Menu(post_id) -> Post(post_id) == post_menu

Post_Menu(restaurant, menu_name) -> Menu(restaurant, menu_name) == post_menu

Menu (restaurant,menu_name)으로 한 이유는, 참조 무결성 제약 조건을 지키기 위함이다. 만약 restaurant_id를 Restaurant relation에서 refer할 경우 참조 무결성 제약이 어겨지고, 실제로도 코드 오류를 발생시키므로 위와 같이 설정하였다.

```
    # 8. Follow's foreign key
    cursor.execute("ALTER TABLE Follow ADD CONSTRAINT FOREIGN KEY (followee_id)
REFERENCES User(user_id)")
    print("8-1 done")
    cursor.execute("ALTER TABLE Follow ADD CONSTRAINT FOREIGN KEY (follower_id)
REFERENCES User(user_id)")

    print("8-2 done")
```

Follow(followee_id) -> User(user_id) == follow

Follow(follower_id) -> User(user_id) == follow