데이터베이스

프로젝트



컴퓨터공학부

201611189 김민성

신효섭 교수님

2020.06.30

|  |
| --- |
| **201611189 김민성 데이터베이스 프로젝트** |
| 1. **E/R Diagram** |
| 1. **User**    * **User 개체는 user id를 기본키(식별자)로 사용한다**    * **User id를 기본키(식별자)로하며, 그 외에도 age, gender, zip code, occuaption을 추가적인 속성으로 가지고 있다.**    * **각 유저당 최소 20개의 평가를 한다고 가정되어 있기 때문에, User와 Data개체 사이에는 1 : N의 관계를 가지고 있다** 2. **Movie**    * **Movie 개체는 movie id를 기본키(식별자)로 사용한다**    * **Movie id를 기본키(식별자)로하며, 그 외에도 title, IMD URL, video release date, release date를 속성으로 가지고 있다**    * **Movie 개체의 경우 장르 속성을 가져야하는데, 장르 별로 해당 여부에 따라 한 movie 개체가 여러 개의 장르 데이터를 가지기 때문에 Multivalued Attribute로 설정했다**    * **Genre 속성의 경우 기본키(식별자)로 movie id가 있어야만 한다. 따라서 자신의 식별자가 다른 개체, Movie의 식별자이므로 Id dependent 하다**    * **Genre 속성의 경우, movie가 존재하지 않으면, 존재할 수 없기 때문에 genre는 weak entity이다**    * **각 영화당 최소 20개의 평가를 한다고 가정되어 있기 때문에, Movie는 여러 개의 raiting을 받을 수 있기 때문에 Movie 와 Data객체 사이에는 1 : N의 관계를 가지고 있다** 3. **Data**    * **Data 개체는 user 개체와, movie 개체의 Association Relationship을 통해 나타나는 Association Table이다**    * **Data 개체는 user 개체의 기본키(식별자)인 user id와 movie 개체의 기본키(식별자)인 movie id의 쌍(user id, movie id)을 사용한다**    * **기본키(식별자)인 user id, movie id 외에도 각 해당 사용자들이 내린 raiting에 대한 속성과 timestamp 속성을 함께 가지고 있다** |
| **2. Mysql 설계** |
| 1. **코드**   **create database db\_201611189;**  **use db\_201611189;**  **create table User(**  **userid int primary key,**  **age int not null,**  **gender char(2) not null,**  **occupation varchar(20) not null,**  **zipcode varchar(5) not null**  **);**  **create table Item(**  **movieid int primary key,**  **title varchar(100) not null,**  **releasedate varchar(30) not null,**  **v\_releasedate varchar(30),**  **url varchar(1000) not null,**  **genre varchar(37) not null**  **);**  **create table Genre(**  **genrename varchar(30),**  **genrecode int primary key**  **);**  **create table Occupation(**  **occupationname varchar(30) not null unique**  **);**  **create table Data(**  **userid int not null,**  **movieid int not null,**  **rating int not null,**  **timestamp varchar(10) not null,**  **primary key (userid,movieid),**  **foreign key (userid) references User(userid),**  **foreign key (movieid) references Item(movieid)**  **);**  **select \***  **from User;**  **select \***  **from Item;**  **select \***  **from Genre;**  **select \***  **from Occupation;**  **select \***  **from Data;**   1. **설명**    * **조건에 따라서** **create database db\_201611189과 use database db\_201611189 를 통해 db\_201611189 라는 데이터 베이스를 생성 및 사용한다**    * **User 테이블은 u.user의 데이터를 저장하는 테이블이며, python을 통해 데이터를 입력 받을 예정이다**    * **User 테이블은 userid를 기본 키로 사용한다**    * **User 테이블의 애트리뷰트들을 not null로 설정함으로써, 만일 각 애트리뷰트에 데이터를 삽입하는 과정에서 오류 및 오작동으로 인해 null 값이 들어오지 못하도록 한다**    * **Item 테이블은 u.item의 데이터를 저장하는 테이블이며, python을 통해 데이터를 입력 받을 예정이다**    * **Item 테이블은 movieid를 기본 키로 사용한다**    * **Item 테이블의 애트리뷰트들을 not null로 설정함으로써, 만일 각 애트리뷰트에 데이터를 삽입하는 과정에서 오류 및 오작동으로 인해 null 값이 들어오지 못하도록 한다**    * **Movie 테이블의 genre의 경우 데이터 셋에 있는 그대로 받아들이기 위해서 0 , 1, | 의 연속으로 이루어진 문자열로 취급하기 위해 varchar형식으로 지정한다**    * **Occupation 테이블은 u.occupaition의 데이터를 저장하는 테이블이며, python을 통해 데이터를 입력 받을 예정이다**    * **Occupation 테이블에는 u.occupation에 존재하는 직업의 이름들을 애트리뷰트로 가지며, 튜블들이 같지 않게 하기 위해서 unique로 선언했으며, null값이 들들어오지 못하도록 not null로 선언한다**    * **Genre 테이블은 u.genre의 데이터를 저장하는 테이블이며, python을 통해서 데이터를 입력 받을 예정이다**    * **Genre 테이블에는 genrecode와 그에 해당하는 genrename이 저장될 것이며, genrecode에 따라 장르들이 분류되기 때문에, genrecode를 기본키로 사용한다**    * **Data 테이블은 u.data의 데이터들을 저장하는 테이블이며, python을 통해서 데이터를 받을 예정이다**    * **userid와 movieid의 쌍을 통해서 각 튜플들을 구별할 수 있기 때문에 (userid,movieid)를 기본키로 설정한다**    * **userid 와 moiveid를 통해 구별된 각 튜블들은 해당 userid를 가진 User가 평가한 별점을 의미하는 rating과 평가받은 시간 정보를 의미하는 timestamp라는 정보를 추가적으로 가진다.**    * **User 테이블과 moiveid를 통해서 관계를 맺기 때문에 data 테이블에서 userid를 User 테이블에 대한 외래키로 사용한다**    * **Movie 테이블과 movieid를 통해서 관계를 맺기 때문에 data 테이블에서 movieid를 Movie 테이블에 대한 외래키로 사용한다**    * **u.data파일에서 다른 데이터들과 달리 (160 234 5 876861185) / (50 246 3 877052329) 2개의 항목이 userid와 movieid를 구별하는 식별자로 탭문자가 아닌 띄어쓰기로 되어있어, 변경하였습니다** |
| **3. 데이터 베이스 구축** |
| * **User 테이블은 943개의 튜플, Item 테이블은 1682개의 튜플, Genre 테이블은 19개의 튜플, Occupation은 21개의 튜플을 가지고 있기 때문에, 정상적으로 모든 데이터가 들어왔다고 볼 수 있고, Data 테이블의 경우에는 데이터 개수가 너무 많아서 일부분만 출력되었으며, python으로 검색 시 모든 데이터가 정상적으로 저장됨을 확인 할 수 있다(입력 내용은 시연 영상 참고)** |
| **4. 검색 프로그램 작성** |
| 1. **코드**   **import pymysql**  **import csv**  **conn = pymysql.connect(host = 'localhost', user='root', password='3746', db='db\_201611189')**  **curs = conn.cursor(pymysql.cursors.DictCursor)**  **"""**  **# user 데이터 삽입**  **f = open('u.user.tsv','r')**  **rdr = csv.reader(f, delimiter = '|')**  **tsvlist = list(rdr)**  **for row in tsvlist:**  **user\_id = (row[0])**  **user\_age = (row[1])**  **user\_gender = (row[2])**  **user\_occupation = (row[3])**  **user\_zipcode = (row[4])**    **sql = "insert into User (userid,age,gender,occupation,zipcode) values (%s,%s,%s,%s,%s)"**  **user\_list = (user\_id,user\_age,user\_gender,user\_occupation,user\_zipcode)**  **curs.execute(sql,user\_list)**  **#db의 변화 저장**  **conn.commit()**  **row = curs.fetchone()**  **f.close()**  **# item 데이터 삽입**  **f = open('u.item.tsv','r',encoding = "ISO-8859-1")**  **rdr = csv.reader(f,delimiter = '|')**  **tsvlist = list(rdr)**  **for row in tsvlist:**  **movie\_id = (row[0])**  **movie\_title = (row[1])**  **movie\_release = (row[2])**  **movie\_video\_release = (row[3])**  **movie\_url = (row[4])**  **movie\_genre = (row[5])+(row[6])+(row[7])+(row[8])+(row[9])+(row[10])+(row[11])+(row[12])+(row[13])+(row[14])+(row[15])+(row[16])+(row[17])+(row[18])+(row[19])+(row[20])+(row[21])+(row[22])+(row[23])**  **sql = "insert into Item (movieid,title,releasedate,v\_releasedate,url,genre) values (%s,%s,%s,%s,%s,%s)"**  **item\_list = (movie\_id,movie\_title,movie\_release,movie\_video\_release,movie\_url,movie\_genre)**  **curs.execute(sql,item\_list)**  **#db의 변화 저장**  **conn.commit()**  **row = curs.fetchone()**  **f.close()**  **# Data 데이터 삽입**  **f = open('u.data.tsv','r')**  **rdr = csv.reader(f, delimiter = '\t')**  **tsvlist = list(rdr)**  **for row in tsvlist:**  **data\_user\_id = (row[0])**  **data\_movie\_id = (row[1])**  **rating = (row[2])**  **timestamp = (row[3])**  **sql = "insert into Data (userid,movieid,rating,timestamp) values (%s,%s,%s,%s)"**  **data\_list = (data\_user\_id,data\_movie\_id,rating,timestamp)**  **curs.execute(sql,data\_list)**  **#db의 변화 저장**  **conn.commit()**  **row = curs.fetchone()**  **f.close()**  **# Genre 데이터 삽입**  **f = open('u.genre.tsv','r')**  **rdr = csv.reader(f, delimiter = '|')**  **tsvlist = list(rdr)**  **tsvlist = list(filter(None, tsvlist))**  **for row in tsvlist:**  **genre\_name = (row[0])**  **genre\_code = (row[1])**    **sql = "insert into Genre (genrename,genrecode) values (%s,%s)"**  **genre\_list = (genre\_name,genre\_code)**  **curs.execute(sql,genre\_list)**  **#db의 변화 저장**  **conn.commit()**  **row = curs.fetchone()**  **f.close()**  **# Occupation 데이터 삽입**  **f = open('u.occupation.tsv','r')**  **rdr = csv.reader(f, delimiter = '|')**  **tsvlist = list(rdr)**  **for row in tsvlist:**  **occupation\_name = (row[0])**    **sql = "insert into Occupation (occupationname) values (%s)"**  **occupation\_list = (occupation\_name)**  **curs.execute(sql,occupation\_list)**  **#db의 변화 저장**  **conn.commit()**  **row = curs.fetchone()**  **f.close()**  **"""**  **# 검색 기능**  **print("검색할 장르를 입력하세요 : ")**  **s\_genre = input()**  **print("검색할 사용자의 직업을 입력하세요 : ")**  **s\_occupation = input()**  **print("검색할 영화의 최소 평점을 입력하세요 : ")**  **s\_min = input()**  **print("검색할 영화의 최대 평점을 입력하세요 : ")**  **s\_max = input()**  **print("정렬 방식을 선택하세요 1. 영화 제목 오름차순 2. 영화 제목 내림차순 3. 평점 오름차순 4.평점 내림차순 5.vote 수 오름차순 6. vote 수 내림차순: ")**  **s\_sortby = input()**  **if s\_min == '':**  **s\_min = 1**  **if s\_max == '':**  **s\_max = 5**    **#사용자가 입력한 장르가 포함된 movieid를 저장하는 list answer\_genre**  **if s\_genre != '':**  **sql = "select genrecode from Genre where genrename = '%s'"%(s\_genre)**  **curs.execute(sql)**  **row= curs.fetchone()**  **s\_genre\_num = row["genrecode"]**  **answer\_genre = []**  **sql\_genre = "select \* from Item i"**  **curs.execute(sql\_genre)**  **row = curs.fetchone()**  **if s\_genre == '':**  **while row:**  **answer\_genre.append(row["movieid"])**  **row = curs.fetchone()**  **else:**  **while row:**  **original\_genre = list(row["genre"])**  **if original\_genre[s\_genre\_num] == '1':**  **answer\_genre.append(row["movieid"])**  **row = curs.fetchone()**  **#사용자가 입력한 직업을 가진 사람을 저장하는 리스트 answer\_occupation**  **answer\_occupation = []**  **sql\_occupation = "select \* from User u"**  **curs.execute(sql\_occupation)**  **row = curs.fetchone()**  **if s\_occupation == '':**  **while row:**  **answer\_occupation.append(row["userid"])**  **row = curs.fetchone()**  **else:**  **while row:**  **original\_occupation = (row["occupation"])**  **if original\_occupation == s\_occupation:**  **answer\_occupation.append(row["userid"])**  **row = curs.fetchone()**  **#answer\_genre에 있는 moiveid만 data테이블에서 필터링**  **genre\_count = 0**  **filter\_genre = []**  **while genre\_count < len(answer\_genre):**  **sql = "select \* from Data d where movieid = '%s'"%(answer\_genre[genre\_count])**  **curs.execute(sql)**  **row = curs.fetchone()**  **while row:**  **filter\_genre.append(row)**  **row = curs.fetchone()**  **genre\_count = genre\_count + 1**  **#answer\_occupation에 있는 moiveid만 data테이블에서 필터링**  **occupation\_count = 0**  **filter\_occupation = []**  **while occupation\_count < len(answer\_occupation):**  **sql = "select \* from Data d where userid = '%s'"%(answer\_occupation[occupation\_count])**  **curs.execute(sql)**  **row = curs.fetchone()**  **while row:**  **filter\_occupation.append(row)**  **row = curs.fetchone()**  **occupation\_count = occupation\_count + 1**  **#각 필터링의 공통부분 구하기**  **answer\_filter = [x for x in filter\_genre if x in filter\_occupation]**  **#장르와 직업에 관하여 필터링이 완료된 테이블을 집계 함수를 이용해 평점 계산**  **sql = "create table Answer (userid int, movieid int, rating int, timestamp int)"**  **curs.execute(sql)**  **answer\_filter\_count = 0**  **while answer\_filter\_count < len(answer\_filter):**  **sql = "insert into Answer (userid,movieid,rating,timestamp) values (%s,%s,%s,%s)"**  **answer\_list = (answer\_filter[answer\_filter\_count]["userid"],answer\_filter[answer\_filter\_count]["movieid"],answer\_filter[answer\_filter\_count]["rating"],answer\_filter[answer\_filter\_count]["timestamp"])**  **curs.execute(sql,answer\_list)**  **conn.commit()**  **answer\_filter\_count = answer\_filter\_count + 1**  **if s\_sortby == '1':**  **sql = "select i.title, AVG(a.rating), COUNT(a.rating) from Answer a, Item i where a.movieid = i.movieid group by a.movieid having AVG(a.rating) >='%s' and AVG(a.rating) <= '%s' order by i.title"%(s\_min,s\_max)**  **curs.execute(sql)**  **row = curs.fetchone()**  **while row:**  **print("영화제목: %s, 평점 : %s 평가횟수 : %s "%(row['title'],row['AVG(a.rating)'],row['COUNT(a.rating)']))**  **row = curs.fetchone()**  **if s\_sortby == '2':**  **sql = "select i.title, AVG(a.rating), COUNT(a.rating) from Answer a, Item i where a.movieid = i.movieid group by a.movieid having AVG(a.rating) >='%s' and AVG(a.rating) <= '%s' order by i.title desc"%(s\_min,s\_max)**  **curs.execute(sql)**  **row = curs.fetchone()**  **while row:**  **print("영화제목: %s, 평점 : %s 평가횟수 : %s "%(row['title'],row['AVG(a.rating)'],row['COUNT(a.rating)']))**  **row = curs.fetchone()**  **if s\_sortby == '3':**  **sql = "select i.title, AVG(a.rating), COUNT(a.rating) from Answer a, Item i where a.movieid = i.movieid group by a.movieid having AVG(a.rating) >='%s' and AVG(a.rating) <= '%s'order by AVG(a.rating)"%(s\_min,s\_max)**  **curs.execute(sql)**  **row = curs.fetchone()**  **while row:**  **print("영화제목: %s, 평점 : %s 평가횟수 : %s "%(row['title'],row['AVG(a.rating)'],row['COUNT(a.rating)']))**  **row = curs.fetchone()**  **if s\_sortby == '4':**  **sql = "select i.title, AVG(a.rating), COUNT(a.rating) from Answer a, Item i where a.movieid = i.movieid group by a.movieid having AVG(a.rating) >='%s' and AVG(a.rating) <= '%s'order by AVG(a.rating) desc"%(s\_min,s\_max)**  **curs.execute(sql)**  **row = curs.fetchone()**  **while row:**  **print("영화제목: %s, 평점 : %s 평가횟수 : %s "%(row['title'],row['AVG(a.rating)'],row['COUNT(a.rating)']))**  **row = curs.fetchone()**  **if s\_sortby == '5':**  **sql = "select i.title, AVG(a.rating), COUNT(a.rating) from Answer a, Item i where a.movieid = i.movieid group by a.movieid having AVG(a.rating) >='%s' and AVG(a.rating) <= '%s'order by COUNT(a.rating)"%(s\_min,s\_max)**  **curs.execute(sql)**  **row = curs.fetchone()**  **while row:**  **print("영화제목: %s, 평점 : %s 평가횟수 : %s "%(row['title'],row['AVG(a.rating)'],row['COUNT(a.rating)']))**  **row = curs.fetchone()**  **if s\_sortby == '6':**  **sql = "select i.title, AVG(a.rating), COUNT(a.rating) from Answer a, Item i where a.movieid = i.movieid group by a.movieid having AVG(a.rating) >='%s' and AVG(a.rating) <= '%s'order by COUNT(a.rating) desc"%(s\_min,s\_max)**  **curs.execute(sql)**  **row = curs.fetchone()**  **while row:**  **print("영화제목: %s, 평점 : %s 평가횟수 : %s "%(row['title'],row['AVG(a.rating)'],row['COUNT(a.rating)']))**  **row = curs.fetchone()**  **sql = "drop table answer"**  **curs.execute(sql)**  **curs.close()**  **conn.close()**   1. **설명**    * **사용할 데이터베이스의 위치를 conn 과 curs를 통해 지정해준다**    * **U.user, u.item, u.genre, u.occupation, u.data파일들을 읽은 후 insert into sql문을 이용해서 데이터를 삽입해준다**    * **데이터 입력이 완료되었다면, 검색조건들을 입력 받으며, 아무것도 입력하지 않고 넘어간다면, 해당 조건에 대하여 모든 대상을 선택한 것으로 한다**    * **사용자가 입력한 장르가 포함된 movieid를 저장하는 리스트인 answer\_genre를 생성하며, Genre 테이블을 통해서 사용자가 입력한 genrecode를 알아낸 후 Item 테이블에서 genre 애트리뷰트 값의 genrecode 번째에 해당되는 값이 1이라면, 해당 튜블의 movieid를 answer\_genre에 저장한다**    * **사용자가 입력한 직업을 가진 사람을 저장하는 리스트 answer\_occupation를 생성하며, 이는 User 테이블에서 사용자가 입력한 직업인 사람만을 선택하기 위해 while문을 통해 해당되는 튜플들을 찾아내고 해당 튜플의 user id를 answer\_occupation에 저장한다**    * **answer\_genre에 존재하는 movieid 만을 가진 영화들만을 선택하기 위해서 Data 테이블에서 answer\_genre에 있는 movieid를 가진 튜플들만 남겨서 filter\_genre에 저장한다**    * **answer\_occupation에 존재하는 userid 만을 가진 사용자들만을 선택하기 위해서 Data 테이블에서 answer\_occupation에 있는 userid를 가진 튜플들만 남겨서 filter\_occupation에 저장한다**    * **filter\_genre와 file\_occupation의 교집합 부분에 해당되는 리스트를 answer\_filter에 저장하고, Answer 테이블을 sql문을 통해 생성한 후 insert into 문을 통해 answer\_filter의 데이터들을 튜플로 변환한다**    * **정렬기준에 따라서 분기를 실시 한 후 Answer 테이블에서 평점과 평가 횟수를 구하기 위해 avg(rating) 과 counting(rating)을 sql문을 통해 실행한다**    * **Avg(rating)을 통해서 구한 영화 별 평점에서 사용자가 입력한 평점 조건을 만족 시키기 위해 출력 sql 문에 where 절의 조건을 추가한다** |