406.426B Data Management and Analysis

2017 Fall

**Project #2**

**DB implementation & query processing**

팀 #12

채현승 (2013 – 11297)

강예진 (2016 – 15058)

권순빈 (2016 – 17330)

최하경 (2016 – 16871)

<목차>

1. 관계 스키마 구성

1.1. 1NF 충족

1.2. 3NF 충족

1.3. Null Redundancy 최소화

1.4. 전체 관계 스키마

1.5 상세 attribute 및 constraints

2. 데이터베이스 생성

2.1. 기본적인 Database생성 방법

2.2. QuestionPosts와 answerPosts의 foreign key 상호 참조

2.3. QuestionPost에 대해 생기는 multivalues를 가지는Tags

2.4. votes, votesBookMark, votesBountyAmount

3. SQL Query 작성

3.1. R6

3.2. R7

3.3. R8

3.4. R9

3.5. R10

4. 데이터 분석

4.1. R6

4.2. R7

4.3. R8

4.4. R9

4.5. R10

1. 관계 스키마 구성

기본적으로 주어진 dataset에서 스키마 형성을 시작했다. 제시되어 있는 table은 userInfo, posts, questionPosts, answerPosts, postHistory, postId, postLinks, badges, comments, tags, votes 11개 이며, 각 table의 attr 또한 제시된 바와 같다. 기본적으로 만들어진 관계 스키마에서 3NF를 충족시킬 수 있도록 스키마를 split 하였다.

1.1. 1NF 충족 : Multivalued 제외

1NF를 충족시키기 위하여 기본 스키마에서 보이는 multivalued atrribute를 제거하는 방향으로 스키마를 split하였다. questionPosts의 Tags는 제시된 dataset에 보면 multivalue이다. 1NF를 만족시키기 위해 기존의 questionPosts에서 questionTags를 파생한다. questionTags에서 Id와 Tags가 composite key를 이룬다. 이때 questionTags에서 Id는 questionPosts Id의 foreign key이다. question Tags는 Id와 Tags 두 가지 attr을 갖는다.

1.2. 3NF 충족

제시된 dataset에 보면, tags.csv에서 ExcerptPostId와 WikiPostId는 항상 동시에 Null값을 갖고, 그렇지 않을때는 ExcerptPostId와 WikiPostId는 Excel 각각 고유한 값을 가진다. 실제 그렇지 않더라도 그렇다고 가정 하겠다. 그렇게 가정하면 3NF를 만족하지 못하게 된다. 이를 Decompose한다.

tagsPosts의 primary key는 Id로 정의한다. 이때 Id, ExcerptPostId, WikiPostId 셋 모두 candidate key이나 Id를 primary key로 정의한다. tags2는 Id, ExcerptPostId, WikiPostId 3개의 attr을 가진다.

1.3. Null redundancy 최소화

DataDescription에서 제시 되어있듯 votes에서 ‘즐겨찾기’ 게시물만, 즉 VoteTypeId = 5 일 때 만 userInfo attr이 값을 갖고, 나머지 경우는 Null이다. 또 votes에서 VoteTypeId = 9일 때만 BountyAmount attr이 값을 갖고, 나머지는 Null이다.

이때, votesBookMark의 primary key는 Id이며, foreign key이고, UserInfoId attr을 가지며 이는 foreign key이다. votesBountyAmount의 primary key는 Id이며, attr로 BountyAmount를 갖는다.

1.4. 전체 관계 스키마

각 relation의 primary key와 foreign key는 아래 표와 같다.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| userInfo | Id | Reputation | DisplayName | Age | CreationDate | LastAccessDate | WebsiteUrl | Location | AboutMe |
| posts | Id | CreationDate | Body | OwnerUserId | LastActivityDate |  |  |  |  |
| questionPosts | Id | PostId | AcceptedAnswerId | ViewCount | Title |  |  |  |  |
| questionTags | Id | Tags |  |  |  |  |  |  |  |
| answerPosts | Id | PostId | Accepted | ParentId |  |  |  |  |  |
| postHistory | Id | PostHistoryTypeId | PostId | CreationDate | UserInfoId | Text | Comment |  |  |
| postLinks | Id | CreationDate | PostId | RelatedPostId | LinkTypeId |  |  |  |  |
| badges | Id | UserInfoId | Name | Date |  |  |  |  |  |
| comments | Id | PostId | Score | CreationDate | UserInfoId |  |  |  |  |
| tags | Id | TagName |  |  |  |  |  |  |  |
| tagsPosts | Id | ExcerptPostId | WikiPostId |  |  |  |  |  |  |
| votes | Id | PostId | VoteTypeId | CreationDate |  |  |  |  |  |
| votesBookMark | Id | UserInfoId |  |  |  |  |  |  |  |
| votesBounty  Amount | Id | BountyAmount |  |  |  |  |  |  |  |
|  |  |  |  |  | foreign key | primary key |  |  |  |

1.5. 상세 attribute 및 constraints

각 relation attribute들의 Domain Constraints, Null Availability, Integrity Constraints는 다음과 같다.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| userInfo | Id | Reputation | DisplayName | Age | CreationDate | LastAccessDate | WebsiteUrl | Location | AboutMe |
| Domain Constraints | Int(11) | Int(11) | Varchar(255) | Int(11) | Timestamp | Timestamp | Varchar(255) | varchar(255) | Longtext |
| Null Availability | Not Null | Not Null | Not Null |  | Not Null | Not Null |  |  |  |
| Integrity Constraints | Primary Key |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
| posts | Id | CreationDate | Body | OwnerUserId | LastActivityDate |  |  |  |  |
| Domain Constraints | Int(11) | Timestamp | Longtext | Int(11) | Timestamp |  |  |  |  |
| Null Availability | Not Null |  |  |  |  |  |  |  |  |
| Integrity Constraints | Primary Key |  |  | Foreign Key references userInfo(Id) |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
| questionPosts | Id | PostId | AcceptedAnswerId | ViewCount | Title |  |  |  |  |
| Domain Constraints | Int(11) | Int(11) | Int(11) | Varchar(255) | Varchar(255) |  |  |  |  |
| Null Availability |  |  |  |  |  |  |  |  |  |
| Integrity Constraints | Primary Key | Foreign Key references posts(Id) | Foreign Key references posts(Id) |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
| questionTags | Id | Tags |  | answerPosts | Id | PostId | Accepted | ParentId |  |
| Domain Constraints | Int(11) | Varchar(255) |  | Domain Constraints | Int(11) | Int(11) | Bit | Int(11) |  |
| Null Availability |  |  |  | Null Availability | Not Null |  |  |  |  |
| Integrity Constraints | Primary Key, Foreign Key references posts(Id) | Primary Key |  | Integrity Constraints | Primary Key | Foreign Key references posts(Id) |  | Foreign Key references questionPosts(Id) |  |
|  |  |  |  |  |  |  |  |  |  |
| postHistory | Id | PostHistoryTypeId | PostId | CreationDate | UserInfoId | Text | Comment |  |  |
| Domain Constraints | Int(11) | Int(11) | Int(11) | Timestamp | Int(11) | Varchar(255) | Varchar(255) |  |  |
| Null Availability | Not Null |  |  |  |  |  |  |  |  |
| Integrity Constraints | Primary Key |  | Foreign Key references posts(Id) |  | Foreign Key references userInfo(Id) |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
| postLinks | Id | CreationDate | PostId | RelatedPostId | LinkTypeId |  |  |  |  |
| Domain Constraints | Int(11) | Timestamp | Int(11) | Int(11) | Int(11) |  |  |  |  |
| Null Availability | Not Null |  |  |  |  |  |  |  |  |
| Integrity Constraints | Primary Key |  | Foreign Key references posts(Id) | Foreign Key references posts(Id) |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
| badges | Id | UserInfoId | Name | Date |  |  |  |  |  |
| Domain Constraints | Int(11) | Int(11) | Varchar(255) | Timestamp |  |  |  |  |  |
| Null Availability | Not Null |  |  |  |  |  |  |  |  |
| Integrity Constraints | Primary Key | Foreign Key references userInfo(Id) |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
| comments | Id | PostId | Score | CreationDate | UserInfoId |  | tags | Id | TagName |
| Domain Constraints | Int(11) | Int(11) | Int(11) | Timestamp | Int(11) |  | Domain Constraints | Int(11) | Varchar(255) |
| Null Availability | Not Null |  |  |  |  |  | Null Availability | Not Null |  |
| Integrity Constraints | Primary Key | Foreign Key references posts(Id) |  |  | Foreign Key references userInfo(Id) |  | Integrity Constraints | Primary Key |  |
|  |  |  |  |  |  |  |  |  |  |
| tagPosts | Id | ExcerptPostId | WikiPostId |  | votes | Id | PostId | VoteTypeId | CreationDate |
| Domain Constraints | Int(11) | Int(11) | Int(11) |  | Domain Constraints | Int(11) | Int(11) | Int(11) | Timestamp |
| Null Availability | Not Null |  |  |  | Null Availability | Not Null |  |  |  |
| Integrity Constraints | Primary Key | Foreign Key references posts(Id) | Foreign Key references posts(Id) |  | Integrity Constraints |  | Foreign Key references posts(Id) |  |  |
|  |  |  |  |  |  |  |  |  |  |
| votesBookMark | Id | UserInfoId |  | votesBountyAmount | Id | BountyAmount |  |  |  |
| Domain Constraints | Int(11) | Int(11) |  | Domain Constraints | Int(11) | Int(11) |  |  |  |
| Null Availability | Not Null |  |  | Null Availability | Not Null |  |  |  |  |
| Integrity Constraints | Primary Key | Foreign Key references votes(Id) |  | Integrity Constraints | Primary Key |  |  |  |  |

2. 데이터베이스 생성

2.1. 기본적인 Database생성 방법

Database 생성은 크게 세 가지로 나눌 수 있다. 데이터베이스를 생성하고, 대부분의 데이터에 대해서는 그 틀 안에 각각의 엑셀 데이터를 담는 표를 생성한 뒤, 생성된 표에 대해 constraints들을 명시한다. 하지만 보고서의 2. 스키마 에서 언급하였듯, tags와 votes에 대해서는 추가적인 table들을 생성한다. UserInfo를 예시로 들어 대체적인 방법을 설명하겠다.

**import** csv  
**import** pymysql  
conn **=** pymysql.connect**(**host**='localhost'**, user**='root'**, password**='1234'**, charset**='utf8')**curs**=**conn.cursor**()**sql **= 'CREATE DATABASE IF NOT EXISTS db2017\_12'**curs.execute**(**sql**)**conn **=** pymysql.connect**(**host**='localhost'**, user**='root'**, password**='1234'**, db**='db2017\_12'**, charset**='utf8')**curs**=**conn.cursor**()**

제일 먼저, csv와 pymysql을 연결한 뒤, db2017\_12라는 이름의 database를 생성한다. 그 다음 table 생성은 다음과 같다.

sql **= (  
 'CREATE TABLE IF NOT EXISTS userInfo ('  
 'Id int(11) NOT NULL,'  
 'Reputation int(11) NOT NULL,'  
 'DisplayName varchar(255),'  
 'Age int(11),'  
 'CreationDate TIMESTAMP,'  
 'LastAccessDate TIMESTAMP,'  
 'WebsiteUrl varchar(255),'  
 'Location varchar(255),'  
 'AboutMe LONGTEXT,'  
 'Primary Key (Id))'  
 )**curs.execute**(**sql**)**

스키마를 따르도록 userInfo라는 table에 Id, Reputation 등의 attribute를 생성한다. 이 때, 제약 조건에 알맞도록 int, varchar, TIMESTAMP, LONGTEXT 등을 적는다. 자세한 constraints 들은 보고서 2번의 스키마를 참고하면 된다. 이렇게 데이터를 담을 수 있는 틀인 userInfo table을 생성한 뒤, userInfo data를 불러온다.

f **=** open**('C://Users\강예진\Downloads//dataset\_revised//userInfo.csv'**, **'r'**, encoding**='utf-8'**,  
 errors**='replace')**rdr **=** csv.reader**(**f**)**next**(**rdr, **None)**userInfo **= []**

그 다음, int 값을 불러들여오는 table의 columns와 그렇지 않은 columns에 대해서 다음과 같은 query를 짠다.

**for** line **in** rdr**:  
 for** i **in (**0, 1, 3**):  
 if** line**[**i**] != "":** line**[**i**] =** int**(**line**[**i**])  
 else:** line**[**i**] = None  
 for** j **in (**2, 4, 5, 6, 7, 8**):  
 if** line**[**j**] == "":** line**[**j**] = None** userInfo.append**(**line**)**f.close**()**

데이터를 다 불러왔으므로, table의 row별로 data의 row들을 대입해주면 된다. 최종 단계는 다음과 같다.

**try:  
 with** conn.cursor**() as** cursor**:** cursor.executemany**("insert into userInfo(Id, Reputation, DisplayName, Age, CreationDate, LastAccessDate, WebsiteUrl, Location, AboutMe) values(%s,%s,%s,%s,%s,%s,%s,%s,%s)"**, userInfo**)** conn.commit**()  
finally:** conn.close**()**

위와 같은 과정으로 userInfo, posts, questionPosts, answerPosts, postHistory, postId, postLinks, badges, comments, tags, votes 중 questionPosts, answerPosts, tags를 제외하고 동일하게 table을 생성한다. 자세한 코드는 파일을 첨부한다.

2.2. QuestionPosts와 answerPosts의 foreign key 상호 참조

QuestionPosts와 answerPosts에 대해서는, 거의 동일한 과정을 거쳐서 생성하되, primary key와 foreign key가 서로의 table을 참조하므로 table 생성은 동일하게 하되, foreign key 부분만 맨 마지막에 따로 작성한다. 코드는 다음과 같다.

conn **=** pymysql.connect**(**host**='localhost'**, user**='root'**, password**='1234'**, db**='db2017\_12'**, charset**='utf8')  
  
try:  
 with** conn.cursor**() as** cursor**:** sql1 **= (  
 'ALTER TABLE questionPosts ADD Foreign Key (AcceptedAnswerId) references answerPosts(Id)'  
 )** sql2 **= (  
 'ALTER TABLE answerPosts ADD Foreign Key (ParentId) references questionPosts(Id)'  
 )** curs.execute**(**sql1**)** curs.execute**(**sql2**)** conn.commit**()  
finally:** conn.close**()**

여기까지는 모든 data에 대해 code가 비슷했으나, tag와 vote의 경우code가 달라진다.

2.3. QuestionPost에 대해 생기는 multivalues를 가지는Tags

Tag의 경우, QuestionPost에 대해서만 생기기 때문에, pass문을 이용하여 questionTags에 대한 code를 생성한다. Tag에 대한 code의 뒷부분은 다음과 같다.

**try:  
 with** conn.cursor**() as** cursor**:** cursor.executemany**("insert into questionPosts(Id, PostId, AcceptedAnswerId, ViewCount, Title) values(%s,%s,%s,%s,%s)"**, questionPosts**)** conn.commit**()  
finally:  
 pass  
  
try:  
 with** conn.cursor**() as** cursor**:** cursor.executemany**(  
 "insert into questionTags(Id, TagName) values(%s,%s)"**, questionTags**)** conn.commit**()  
finally:** conn.close**()**

tag의 경우, 보고서 앞부분에서 언급하였듯 tagname이 multivalued이다. 따라서 이를 simple attributes로 쪼개주어야 한다. 쪼개는 code는 split를 이용하여 구현하였고, 다음과 같다. 값들이 “<”와 “>”를 이용하여 나누어져 있었으므로, 이를 이용하여 하나씩 분리한다.

**for** tagname **in** line**[**5**]**.split**('<')[**1**:]:** line2 **= []** line2.append**(**line**[**0**])** line2.append**(**tagname.replace**('>'**,**''))** questionTags.append**(**line2**)** questionPosts.append**(**line**)  
 del** line**[**5**]**f.close**()**

그리고 tags와 tagsPost에 대해선 스키마를 참고하여 다시 동일한 과정으로 table들을 생성하되, 역시 pass문을 이용하여 code를 생성한다. 자세한 code는 다음과 같다.

conn **=** pymysql.connect**(**host**='localhost'**, user**='root'**, password**='1234'**, db**='db2017\_12'**, charset**='utf8')**curs**=**conn.cursor**()**sql **= (  
 'CREATE TABLE IF NOT EXISTS tags ('  
 'Id int(11) NOT NULL,'  
 'TagName varchar(255),'  
 'Primary Key (Id))'  
 )**curs.execute**(**sql**)**sql **= (  
 'CREATE TABLE IF NOT EXISTS tagsPosts ('  
 'Id int(11) NOT NULL,'  
 'ExcerptPostId int(11),'  
 'WikiPostId int(11),'  
 'Primary Key (Id),'  
 'Foreign Key (WikiPostId) references posts(Id),'  
 'Foreign Key (ExcerptPostId) references posts(Id)'  
 ')'  
 )**curs.execute**(**sql**)**f **=** open**('C://Users\강예진\Downloads//dataset\_revised//tags.csv'**, **'r'**, encoding**='utf-8'**,  
 errors**='replace')**rdr **=** csv.reader**(**f**)**next**(**rdr, **None)**tags **= []**tagsPosts **= []  
  
for** line **in** rdr**:** line2 **= []  
 for** i **in (**0, 2, 3**):  
 if** line**[**i**] != "":** line**[**i**] =** int**(**line**[**i**])  
 else:** line**[**i**] = None  
 if** line**[**1**] == "" :** line**[**1**] = None** line2.append**(**line**[**0**])** line2.append**(**line**[**2**])** line2.append**(**line**[**3**])  
 del** line**[**2**]  
 del** line**[**2**]** tags.append**(**line**)  
 if** line2**[**1**] != None :** tagsPosts.append**(**line2**)**f.close**()  
  
try:  
 with** conn.cursor**() as** cursor**:** cursor.executemany**("insert into tags values(%s,%s)"**, tags**)** conn.commit**()  
finally:  
 pass  
  
try:  
 with** conn.cursor**() as** cursor**:** cursor.executemany**("insert into tagsPosts values(%s ,%s,%s)"**, tagsPosts**)** conn.commit**()  
finally:** conn.close**()**

2.4. votes, votesBookMark, votesBountyAmount

마지막으로, vote행위에 대해선, votes, votesBookMark, votesBountyAmount의 3개 table을 생성하고, pass문을 이용하여 3개의 datasheet를 한 번에 삽입한다. Tags와 tagsPost를 구현하는 과정과 비슷하므로, 자세한 code는 첨부된 파일에서 보기로 한다.

3. SQL Query 작성

3.1. R6

문제에 제시된 나이 대 별 최고 조회수 회원을 tuple로 가지는 table을 ageBest로 정의하고, 생성하는데 다음과 같다.

ageBest =

SELECT U.Id, SUM(Q.ViewCount) AS Totalhits

FROM userInfo AS U, posts AS P, questionPosts AS Q

WHERE U.Id=P.OwnerUserId AND P.Id=Q.PostId AND (U.Age BETWEEN 10 AND 19)

GROUP BY U.Id

ORDER BY Totalhits DESC

LIMIT 1

위와 같이 ViewCount의 총합을 Totalhits로 정의하고, 10대 중에서 Totalhits가 가장 높은 한명을 tuple로 뽑는다.

같은 방식으로 20대, 30대, 40대, 50대를 정의하고 UNION ALL을 사용하여 각 나이대 별로 뽑아진 최대 Totalhits 회원들을 UNION하여 반환한 Table을 ageBest라 정의한다.

이 ageBest Table에 대해 U로 정의한 userInfo table의 U.ID와 ageBest table의 ageBest.Id에 대해 natural join하고 그 결과를 Reputaion에 대해 내림차순으로 반환하면 결과를 얻을 수 있다.

SELECT \*

FROM userInfo AS U, AgeBest

WHERE U.ID = AgeBest.Id

ORDER BY U.Reputation DESC

자세한 코드는 첨부한다.

3.2. R7

R7에서는 년도 별 계정 생성자의 수를 5 columns로 나타내라고 하였다. 따라서, column을 뽑기 위해 CROSS JOIN을 이용하여 각 계정의 생성 년도로 GROUP BY를 해서 계정 수를 세었다. 년도 별로 같은 쿼리 문을 반복하였으며, 쿼리의 일부인 2010년의 예시는 다음과 같다.

SELECT COUNT(\*) AS "2010"

FROM userInfo AS U

WHERE Year(U.CreationDate)=2010

GROUP BY Year(U.CreationDate) ) AS y2010

이렇게 2010년, 2011년, 2012년, 2013년, 2014년에 대해 5개의 query를 cross join했다. 자세한 sql문은 첨부된 파일에 있다.

3.3 R8

R8을 구성하는 쿼리는 두 개 테이블의 INNER JOIN 형태로 구성되어 있다. 첫 번째 테이블은 댓글 수가 10개 이상인 게시물에 대하여 고유번호와 댓글 수를 반환하는 COMMENTCOUNT 테이블이다. 두 번째 테이블은 좋아요 수가 1개 이상인 게시물에 대하여 고유번호, 좋아요 수와 싫어요 수를 반환하는 LIKEDISLIKE 테이블이다.

COMMENTCOUNT 테이블은 posts와 comments 테이블을 INNER JOIN하여 만들었다. 게시글의 Id와 댓글의 PostId가 같다는 조건을 만들고 게시글의 Id를 기준으로 GROUP BY문을 사용하여 그룹화 하였다. 댓글 수가 10개 이상이어야 한다는 조건을 HAVING 문을 사용하여 ‘댓글 수가 10개 이상인 게시물’에 대하여 고유번호와 댓글 수를 반환하는 테이블을 생성하였다.

LIKEDISLIKE 테이블은 좋아요 수가 1개 이상인 게시물을 보여주는 LIKEPOST와 게시물의 싫어요 수를 보여주는 DISLIKEPOST 테이블의 INNER JOIN 형태로 구성되어 있다.

LIKEPOST 테이블은 posts와 votes 테이블을 INNER JOIN하여 만들었다. 게시글의 Id와 votes의 PostId가 같다는 조건과 VoteTypeId이 좋아요를 나타내는 2어야 한다는 조건을 부여하였고, 게시글의 Id를 기준으로 GROUP BY문을 사용하여 그룹화 하였다. 좋아요 수가 1개 이상이어야 한다는 조건을 HAVING 문을 사용하여 ‘좋아요 수가 1개 이상인 게시물’에 대하여 고유번호와 좋아요 수를 반환하는 테이블을 생성하였다.

DISLIKEPOST 테이블은 posts와 votes 테이블을 INNER JOIN하여 만들었다. 게시글의 Id와 votes의 PostId가 같다는 조건과 VoteTypeId이 싫어요를 나타내는 3이어야 한다는 조건을 부여하였고, 게시글의 Id를 기준으로 GROUP BY문을 사용하여 그룹화 하였다. 이 테이블은 게시글의 고유번호와 싫어요 수를 반환한다.

이렇게 생성된 두 테이블의 고유번호가 같다는 조건을 부여하여 INNER JOIN하고, 게시글 고유번호와 좋아요 수, 싫어요 수, 점수를 SELECT문을 이용하여 출력한다. 이때 ORDER BY문을 사용하여 점수에 대해 내림차순으로 결과물을 출력하였다.

3.4 R9

사용자 고유번호는 userInfo테이블의 Id를 뜻하는 것으로 첨부한 코드처럼 userInfo를 U라는 축약어로 한 다음, U.Id로 표현하였다. 획득한 뱃지 수는 cbadge로 명명하였으며, 구하는 방법은 userInfo테이블과 badges테이블을 서로 연계되는 attribute인 사용자 고유번호를 통해 LEFT OUTER JOIN과 RIGHT OUTER JOIN을 하여 UNION하였고, 사용자 고유번호로 그룹화 및 COUNT(\*)를 하면 각 사용자 고유번호가 나타나는 개수가 곧, 해당 사용자가 얻은 뱃지수가 되는 것이다.

FROM

(SELECT U.Id, COUNT(\*) AS cbadge

FROM userInfo AS U

LEFT OUTER JOIN badges AS B

ON U.Id = B.UserInfoId

UNION

SELECT U.Id, COUNT(\*) AS cbadge

FROM userInfo AS U

RIGHT OUTER JOIN badges AS B

ON U.Id = B.UserInfoId

GROUP BY U.Id) AS getbadge

평균 게시물 작성 수는 avPost로 명명하였고, 작성 게시물 개수를 구하는 방법은 위 방법과 유사하다. 다만, R9의 핵심조건인 ‘1년마다’를 반영하기 위해 R7에서 정의된 년도를 토대로 2010년부터 2014년까지 총 5개 년도로 분리하여 각각을 LEFT OUTER JOIN과 RIGHT OUTER JOIN을 하여 UNION을 하였다. 그렇게 하면, R9에서 요구하는 기능을 정확히 수행할 수 있다. 정확한 요구사항이란, 예를 들어 사이트 계정 생성을 2012년에 만든 사용자의 경우 2010년, 2011년에는 게시물 작성 기록이 없을 것이다. 따라서 그 사용자 입장에서 1년마다 평균적으로 작성한 게시물은 2012년 이후인 3개 년도에만 해당되는 것으로 평균을 구할 때 나누기 3이 되어야 하는 것이다. 또, 계정 생성이후의 년도에 게시물을 하나도 작성하지 않았을 경우에도 그 년도를 제외하고 평균을 내야한다. 아래의 코드는 이 2개의 상황을 모두 충족한다. 게시물 작성이 없는 년도에는 그 사용자의 정보가 나타나지 않기 때문이다. 그렇게 각 연도별 게시물 작성 개수를 cPost로 명명하고, AVG(cPost)를 위에서 언급한 avPost로 하여 구하고자 하는 평균 게시물 작성 수를 얻을 수 있다.

FROM (

SELECT U.Id, COUNT(\*) AS cPost

FROM userInfo AS U

LEFT OUTER JOIN posts AS P

ON U.Id = P.OwnerUserId

UNION

SELECT U.Id, COUNT(\*) AS cPost

FROM userInfo AS U

RIGHT OUTER JOIN posts AS P

ON U.Id = P.OwnerUserId

WHERE YEAR(P.CreationDate) =2010

GROUP BY U.Id

UNION ALL

SELECT U.Id, COUNT(\*) AS cPost

FROM userInfo AS U

LEFT OUTER JOIN posts AS P

ON U.Id = P.OwnerUserId

UNION

SELECT U.Id, COUNT(\*) AS cPost

FROM userInfo AS U

RIGHT OUTER JOIN posts AS P

ON U.Id = P.OwnerUserId

WHERE YEAR(P.CreationDate) =2011

GROUP BY U.Id

코드의 마지막 줄에는 WHERE문과 ORDER BY문을 통해 R9에 명시된 제약조건들을 나타내주었다. 뱃지가 50개 이상이어야 하고, 등록된 사용자여야 함을(익명의 사용자는 뱃지를 받을 수 없기 때문이다) WHERE문으로 표현하였고, 뱃지 수의 내림차순으로 정렬하기 위해 ORDER BY문을 사용하였다.

ON getbadge.Id = postaverage.Id

WHERE (cbadge >= 50) AND NOT (getbadge.Id = 0)

ORDER BY cbadge DESC

3.5 R10

R10의 자유토론 주제는 ‘월별 댓글이 가장 많이 달린 게시물’에 관한 정보이다. 이 정보가 온라인 쇼핑몰에 적용되었을 경우, 월별 가장 인기가 많았던 상품을 추출하여 홍보하는데 사용할 수 있을 것이다. 이 정보가 일반 커뮤티니 사이트에 적용되었을 경우 연말에 ‘월별 가장 인기가 많았던 게시물’과 같은 한 해에 대한 사이트 정산을 할 수도 있을 것이라고 기대된다.

게시물 테이블에서 게시물 작성날짜와 게시물의 고유번호, 게시물의 내용, 게시물의 최대 댓글 수로 최종 출력물 테이블을 만들었다.

최종 출력물 테이블은 큰 하나의 쿼리 안에 두 개의 서브쿼리를 넣어 구성되었다.

heidisql에서는 FULL OUTER JOIN이 따로 지원되지 않아, LEFT OUTER JOIN과 RIGHT OUTER JOIN의 합(UNION)의 형태로 SQL문을 짠 commentcount 테이블은 게시물 별 댓글을 게시물 단위로 그룹화하여, 게시물 별 댓글 수를 내림차순으로 나타낸 테이블이다. 이 commentcount 테이블과 게시물 테이블을 서로의 고유번호가 같다는 조건으로 INNER JOIN하고, 월별로 그룹화하면 최종 출력물이 나오게 된다. (단, 게시물의 고유번호가 0이면 안 된다는 조건이 추가 부여 된다.)

4. 결과분석

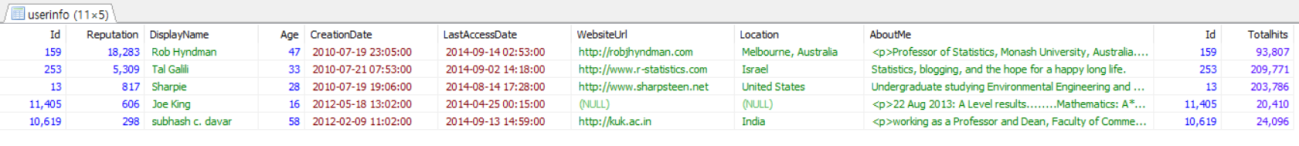
4.1 R6

R6에 해당하는 QUERY 실행 결과 다음과 같은 결과를 얻었다.

결과는 10대, 20대, 30대, 40대, 50대 이후의 나이 대 별로 가장 조회수가 높은 사람 1명씩을 뽑고,

이를 reputation에 따라 내림차순 정렬 한 것이다.

30대 그룹에서 선정된 33세 Tal Galili가 조회수가 209,771건으로 가장 높았고, 20 대 그룹의 28Sharpie가 조회수 203,786회로 그 뒤를 이었다.



4.2 R7

R7에 해당하는 QUERY 실행 결과 다음과 같은 결과를 얻었다.

결과는 사용자 계정 생성 날짜시간을 2010년, 2011년, 2012년, 2013년, 2014년으로 나눴을 때 순서대로 각 년도 별로 생성 된 계정 수이다.

2014년에 계정 생성 수가 14,442회로 가장 높았고, 2013년이 12,232회로 그 뒤를 이었다. 뿐만 아니라 2010년 이후로 계정 생성 수가 꾸준히 증가하였다. 이는 사이트가 점점 더 번성하고 있음을 알 수 있다.

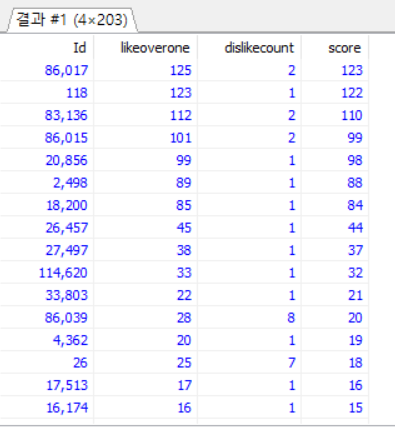


4.3 R8

R8에 해당하는 QUERY의 실행 결과 다음과 같은 결과를 얻었다.

결과는 댓글 수가 10개 이상, 좋아요가 1개 이상인 게시물을 게시물 점수의 내림차순으로 정렬하 여 뽑은 것이며, 게시물 고유번호, 게시물이 획득한 좋아요 수, 게시물이 획득한 싫어요 수, 게시물이 획득한 점수(좋아요- 싫어요)로 구성되어있다.

이들 대부분은 공통적으로 싫어요 수가 한자리 수(최소 1, 최대 8) 임을 알 수 있다. 즉 그 말은 이들 대부분의 score는 싫어요 수보다는 좋아요 수로 결정된다는 의미이며 게시물 ID 33,803과 ID 86,039를 제외하고는 모두 좋아요 수와 점수가 비례한다.



4.4 R9

R9에 해당하는 QUERY를 실행하여 다음과 같은 결과를 얻었다.

사용자별로 가장 badge 획득 개수가 많은 순으로 결과값을 도출하였다. Id 919가 획득 badge수 456개로 가장 많았고, 그 뒤로 300개 대의 badge획득 사용자 1명, 200개 대의 badge 획득 사용자 3명, 그 뒤로 100개 대의 badge 획득 사용자들과 0개 대의 badge획득 사용자들이 줄을 이었다. 이를 보았을 때, badge획득이 200대를 진입할 경우 거의 사용자가 없음을 알 수 있다.

4.5 R10

R10에 해당하는 QUERY를 실행한 결과 다음과 같은 결과를 얻었다.

월 별로 가장 댓글이 많았던, 혹은 사람들의 관심도가 높았던 게시물들의 주제 사이의 연관성이나 경향성은 크게 나타나지 않았다. 이는 애초에 dataset이 같은 주제에 대한 게시물들의 집합이 아니기 때문에 경향성을 발견하기 힘든 것이다. 그러나 다른 dataset, 예를 들어 쇼핑몰 등의 사이트에 적용한다면, 사람들이 어떤 주제, 혹은 제품에 관심이 많았는지 그 경향성을 쉽게 분석할 수 있을 것이다. 따라서 본 쿼리는 통계적으로 유용한 쿼리라고 볼 수 있다.

