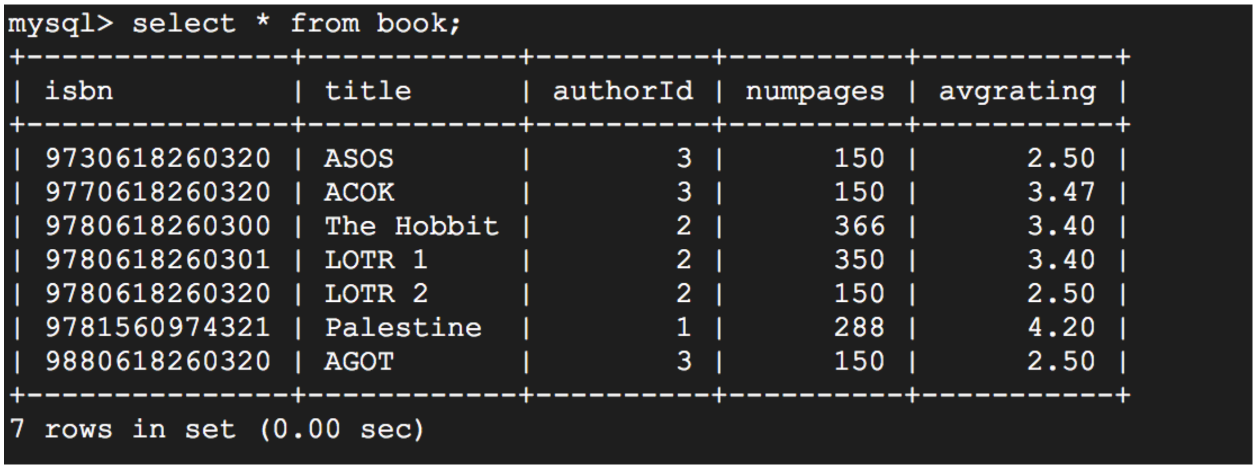
CSCI 585 HW2 Report

Name: Hao Wu
USC ID:1699530173
E-mail: hwu638@usc.edu
Database: MySQL
Consider Set for bonus: Set 3

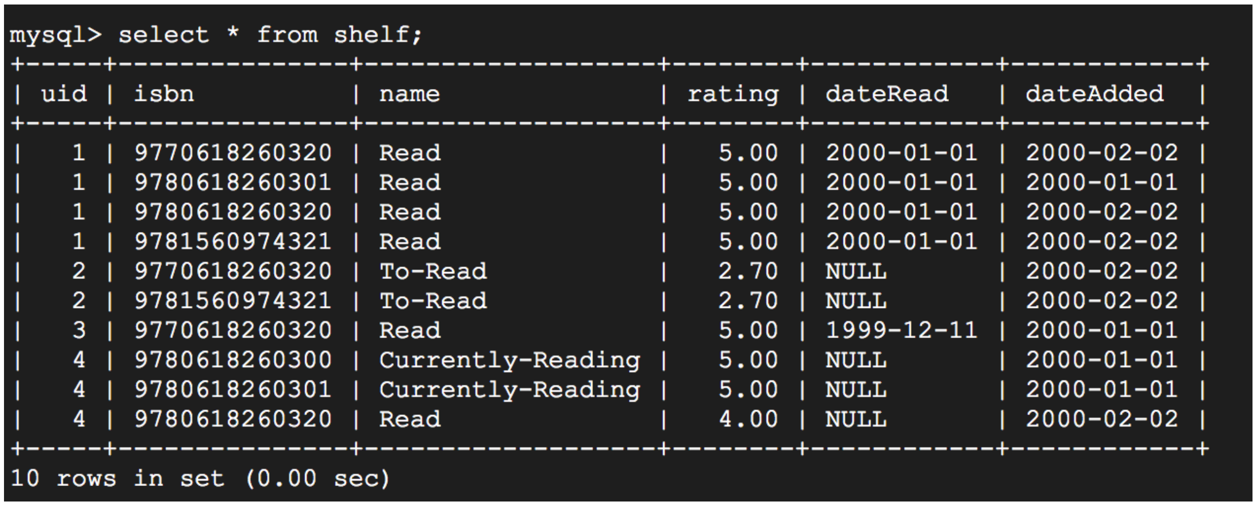
1. GoodReads database

Q1 User adds a new book to his shelf with a rating. Update the average rating of that book.
Query: UPDATE book SET avgrating = (SELECT AVG(rating) FROM shelf WHERE isbn = '9780618260301') WHERE isbn = '9780618260301';
Explanation: The assumption is that user add a new book and the isbn of the book is '9780618260301'. To realize this function, we should firstly calculate the average rating of the book than update its value.

Screen shot



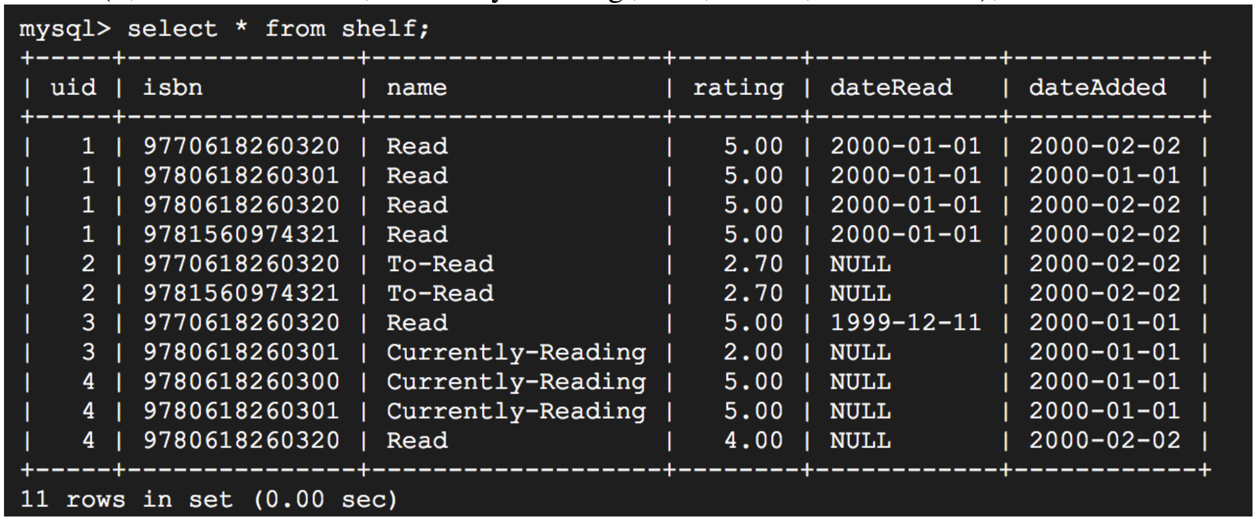
```
mysql> select * from book;
+---------------+------------+----------+----------+-----------+
| isbn          | title      | authorId | numpages | avgrating |
+---------------+------------+----------+----------+-----------+
| 9730618260320 | ASOS       |        3 |      150 |      2.50 |
| 9770618260320 | ACOK       |        3 |      150 |      3.47 |
| 9780618260300 | The Hobbit |        2 |      366 |      3.40 |
| 9780618260301 | LOTR 1     |        2 |      350 |      3.40 |
| 9780618260320 | LOTR 2     |        2 |      150 |      2.50 |
| 9781560974321 | Palestine  |        1 |      288 |      4.20 |
| 9880618260320 | AGOT       |        3 |      150 |      2.50 |
+---------------+------------+----------+----------+-----------+
7 rows in set (0.00 sec)
```



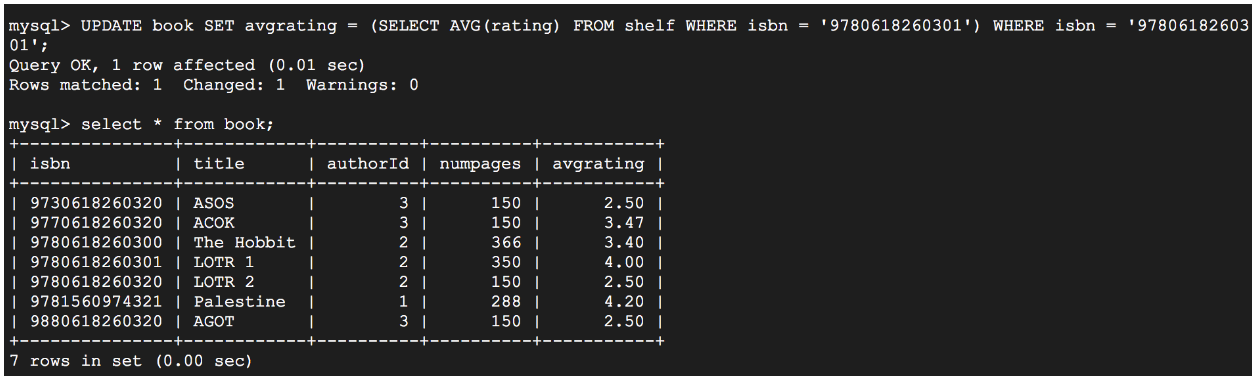
```
mysql> select * from shelf;
+-----+---------------+-------------------+--------+------------+------------+
| uid | isbn          | name              | rating | dateRead   | dateAdded  |
+-----+---------------+-------------------+--------+------------+------------+
|   1 | 9770618260320 | Read              |   5.00 | 2000-01-01 | 2000-02-02 |
|   1 | 9780618260301 | Read              |   5.00 | 2000-01-01 | 2000-01-01 |
|   1 | 9780618260320 | Read              |   5.00 | 2000-01-01 | 2000-02-02 |
|   1 | 9781560974321 | Read              |   5.00 | 2000-01-01 | 2000-02-02 |
|   2 | 9770618260320 | To-Read           |   2.70 | NULL       | 2000-02-02 |
|   2 | 9781560974321 | To-Read           |   2.70 | NULL       | 2000-02-02 |
|   3 | 9770618260320 | Read              |   5.00 | 1999-12-11 | 2000-01-01 |
|   4 | 9780618260300 | Currently-Reading |   5.00 | NULL       | 2000-01-01 |
|   4 | 9780618260301 | Currently-Reading |   5.00 | NULL       | 2000-01-01 |
|   4 | 9780618260320 | Read              |   4.00 | NULL       | 2000-02-02 |
+-----+---------------+-------------------+--------+------------+------------+
10 rows in set (0.00 sec)
```

1. The original table data.

INSERT INTO `shelf` (`uid`, `isbn`, `name`, `rating`, `dateRead`, `dateAdded`)
VALUES
(3, '9780618260301', 'Currently-Reading', 2.00, NULL, '2000-01-01');



```
mysql> select * from shelf;
+-----+---------------+-------------------+--------+------------+------------+
| uid | isbn          | name              | rating | dateRead   | dateAdded  |
+-----+---------------+-------------------+--------+------------+------------+
|   1 | 9770618260320 | Read              |   5.00 | 2000-01-01 | 2000-02-02 |
|   1 | 9780618260301 | Read              |   5.00 | 2000-01-01 | 2000-01-01 |
|   1 | 9780618260320 | Read              |   5.00 | 2000-01-01 | 2000-02-02 |
|   1 | 9781560974321 | Read              |   5.00 | 2000-01-01 | 2000-02-02 |
|   2 | 9770618260320 | To-Read           |   2.70 | NULL       | 2000-02-02 |
|   2 | 9781560974321 | To-Read           |   2.70 | NULL       | 2000-02-02 |
|   3 | 9770618260320 | Read              |   5.00 | 1999-12-11 | 2000-01-01 |
|   3 | 9780618260301 | Currently-Reading |   2.00 | NULL       | 2000-01-01 |
|   4 | 9780618260300 | Currently-Reading |   5.00 | NULL       | 2000-01-01 |
|   4 | 9780618260301 | Currently-Reading |   5.00 | NULL       | 2000-01-01 |
|   4 | 9780618260320 | Read              |   4.00 | NULL       | 2000-02-02 |
+-----+---------------+-------------------+--------+------------+------------+
11 rows in set (0.00 sec)
```

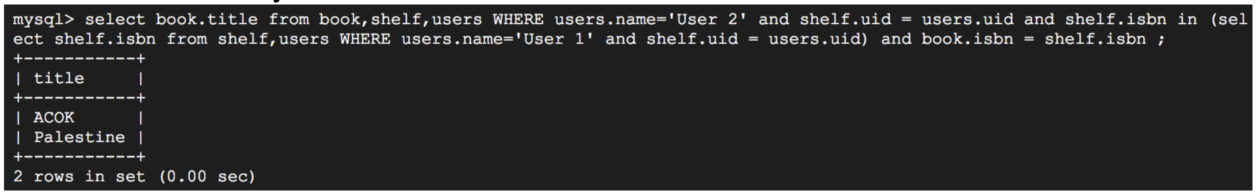


```
mysql> UPDATE book SET avgrating = (SELECT AVG(rating) FROM shelf WHERE isbn = '9780618260301') WHERE isbn = '97806182603
01';
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from book;
+---------------+------------+----------+----------+-----------+
| isbn          | title      | authorId | numpages | avgrating |
+---------------+------------+----------+----------+-----------+
| 9730618260320 | ASOS       |        3 |      150 |      2.50 |
| 9770618260320 | ACOK       |        3 |      150 |      3.47 |
| 9780618260300 | The Hobbit |        2 |      366 |      3.40 |
| 9780618260301 | LOTR 1     |        2 |      350 |      4.00 |
| 9780618260320 | LOTR 2     |        2 |      150 |      2.50 |
| 9781560974321 | Palestine  |        1 |      288 |      4.20 |
| 9880618260320 | AGOT       |        3 |      150 |      2.50 |
+---------------+------------+----------+----------+-----------+
7 rows in set (0.00 sec)
```

After executing query, the arvgraing is updated.

Q2 Find the names of the common books that were read by any two users X and Y.
Query: select book.title from book,shelf,users WHERE users.name='User 2' and shelf.uid = users.uid and shelf.isbn in (select shelf.isbn from shelf,users WHERE users.name='User 1' and shelf.uid = users.uid) and book.isbn = shelf.isbn ;

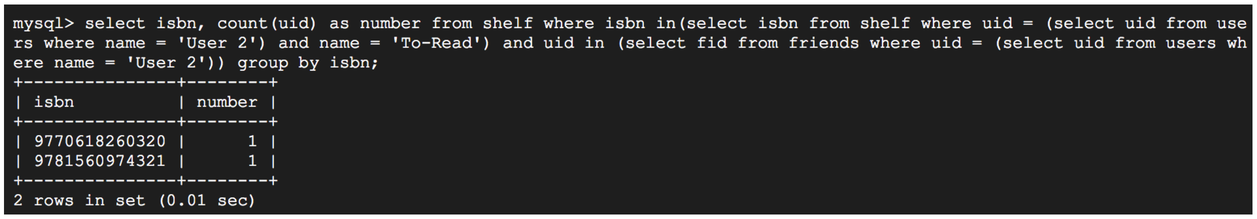
Explanation: We assume that the user X's user name is 'User 1' and user Y's user name is 'User 2'. We Firstly find the book read by user X. Then find the book which read by Y from the book read by X.



```
mysql> select book.title from book,shelf,users WHERE users.name='User 2' and shelf.uid = users.uid and shelf.isbn in (sel
ect shelf.isbn from shelf,users WHERE users.name='User 1' and shelf.uid = users.uid) and book.isbn = shelf.isbn ;
+-----------+
| title     |
+-----------+
| ACOK      |
| Palestine |
+-----------+
2 rows in set (0.00 sec)
```

Q3 For each book that user X has in his to-read list, find the number of friends that have read the book.
Query: select isbn, count(uid) as number from shelf where isbn in(select isbn from shelf where uid = (select uid from users where name = 'User 2') and name = 'To-Read') and uid in (select fid from friends where uid = (select uid from users where name = 'User 2')) group by isbn

Explanation: We assume that the user X's user name is 'User 2'. Firstly, find the user X's to-read book's isbn then search in shelf table to count the item have same isbn with X's book and uid with X's friends. Then, display the data by count users number who read the same book.

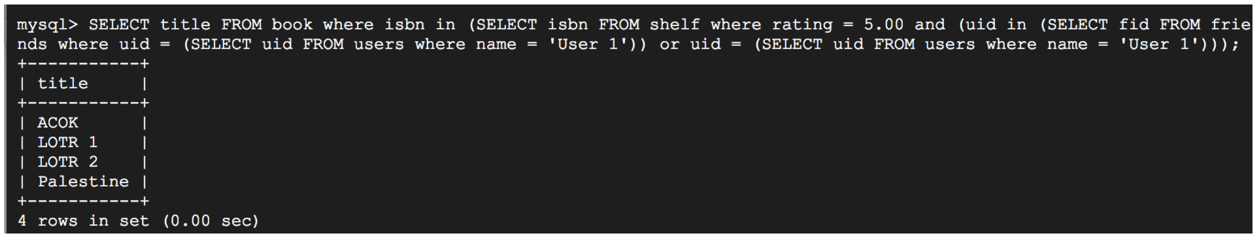


```
mysql> select isbn, count(uid) as number from shelf where isbn in(select isbn from shelf where uid = (select uid from use
rs where name = 'User 2') and name = 'To-Read') and uid in (select fid from friends where uid = (select uid from users wh
ere name = 'User 2')) group by isbn;
+---------------+--------+
| isbn          | number |
+---------------+--------+
| 9770618260320 |      1 |
| 9781560974321 |      1 |
+---------------+--------+
2 rows in set (0.01 sec)
```

Q4 Find the names of all books where user X and any of his/her friends X gave the book a rating of 5 stars.(e.g.,ifXandYgave5starstobook1,andXandZgave5starstobook2,bothbooks 1 and 2 should be included in the output.)

Query: SELECT title FROM book where isbn in (SELECT isbn FROM shelf where rating = 5.00 and (uid in (SELECT fid FROM friends where uid = (SELECT uid FROM users where name = 'User 1')) or uid = (SELECT uid FROM users where name = 'User 1')));

Explanation: We assume that the user X's user name is 'User 1'. First, find the specific isbns from the shelf table. The filter condition is that the rating should be equal to 5 and uid should belong to X or X's friend. Then find the books title from book table according to the isbns.

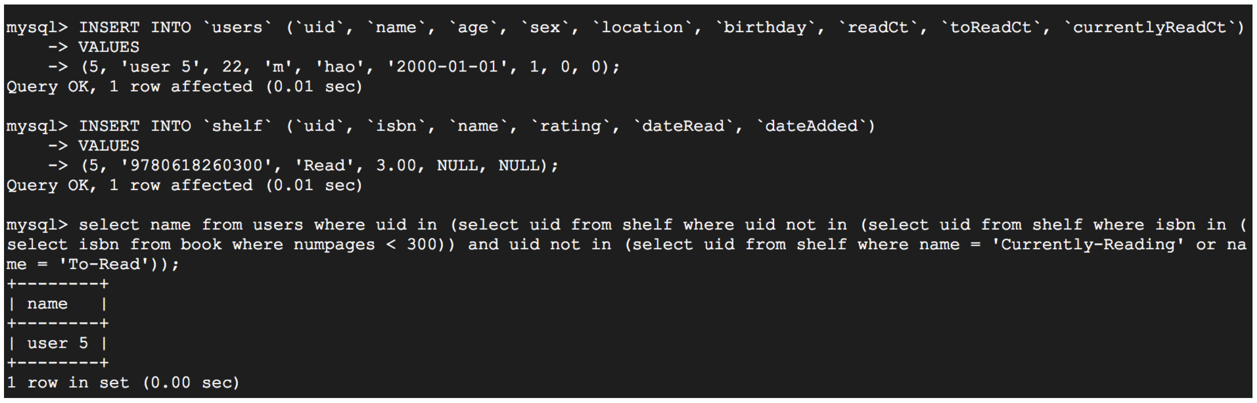


```
mysql> SELECT title FROM book where isbn in (SELECT isbn FROM shelf where rating = 5.00 and (uid in (SELECT fid FROM frie
nds where uid = (SELECT uid FROM users where name = 'User 1')) or uid = (SELECT uid FROM users where name = 'User 1')));
+-----------+
| title     |
+-----------+
| ACOK      |
| LOTR 1    |
| LOTR 2    |
| Palestine |
+-----------+
4 rows in set (0.00 sec)
```

Q5 Find users who have only read books that have more than 300 pages.

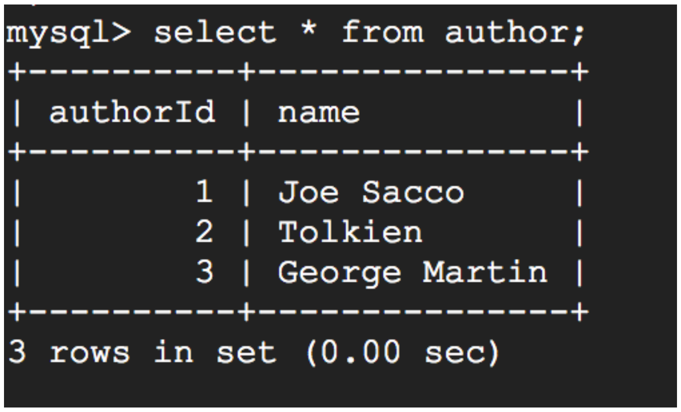
Query: select name from users where uid in (select uid from shelf where uid not in (select uid from shelf where isbn in (select isbn from book where numpages < 300)) and uid not in (select uid from shelf where name = 'Currently-Reading' or name = 'To-Read'));

Explanation: First, find the the isbn of book which have 300 less pages. Second find the users who don't read such book. Finally, take the rest of the user in shelf table with the exception of the user which read 300 less page book and have other book which the 'name' is not 'read'.



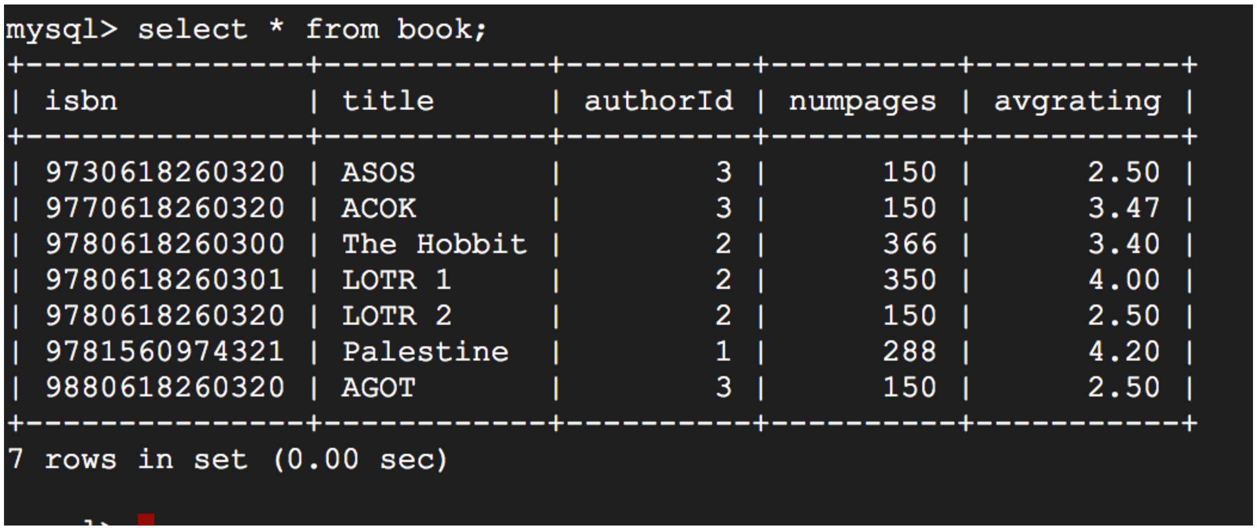
```
mysql> INSERT INTO `users` (`uid`, `name`, `age`, `sex`, `location`, `birthday`, `readCt`, `toReadCt`, `currentlyReadCt`)
    -> VALUES
    -> (5, 'user 5', 22, 'm', 'hao', '2000-01-01', 1, 0, 0);
Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO `shelf` (`uid`, `isbn`, `name`, `rating`, `dateRead`, `dateAdded`)
    -> VALUES
    -> (5, '9780618260300', 'Read', 3.00, NULL, NULL);
Query OK, 1 row affected (0.01 sec)

mysql> select name from users where uid in (select uid from shelf where uid not in (select uid from shelf where isbn in (
select isbn from book where numpages < 300)) and uid not in (select uid from shelf where name = 'Currently-Reading' or na
me = 'To-Read'));
+--------+
| name   |
+--------+
| user 5 |
+--------+
1 row in set (0.00 sec)
```

GoodReads Table Content:



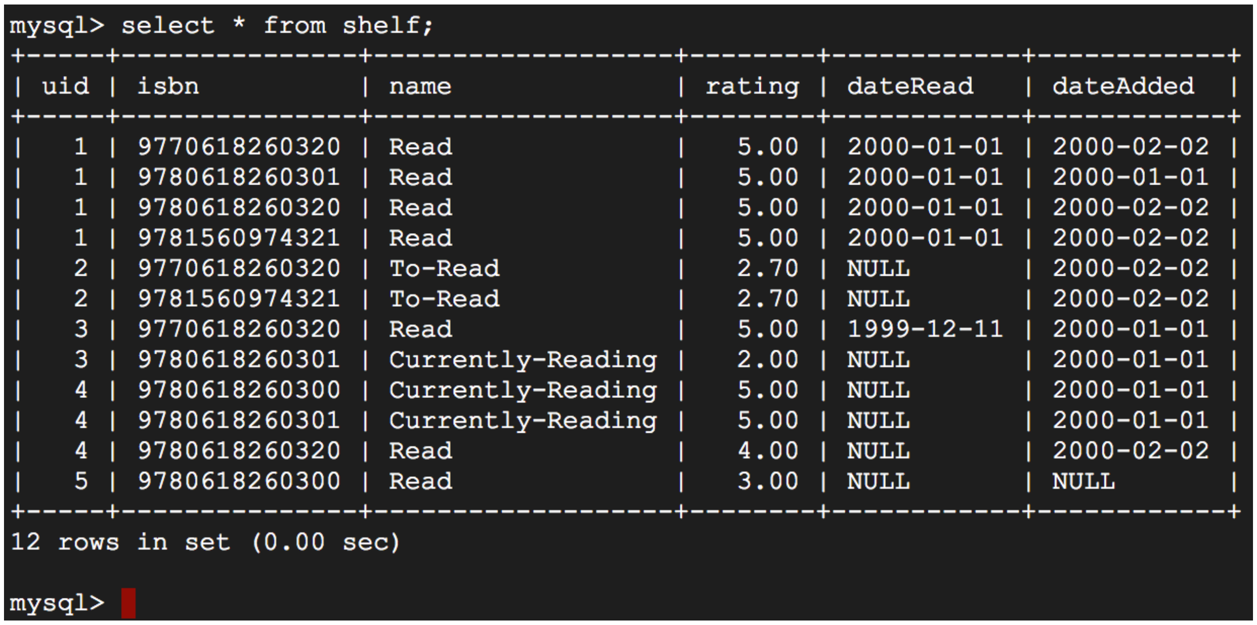
```
mysql> select * from author;
+----------+----------------+
| authorId | name           |
+----------+----------------+
|        1 | Joe Sacco      |
|        2 | Tolkien        |
|        3 | George Martin  |
+----------+----------------+
3 rows in set (0.00 sec)
```

1. Author Table



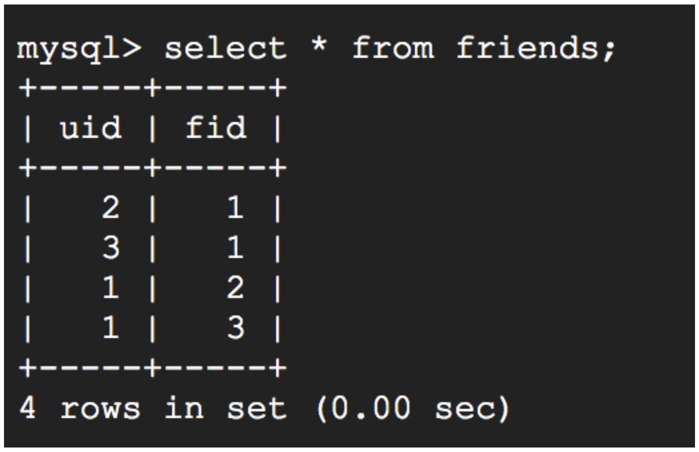
```
mysql> select * from book;
+---------------+------------+----------+----------+-----------+
| isbn          | title      | authorId | numpages | avgrating |
+---------------+------------+----------+----------+-----------+
| 9730618260320 | ASOS       |        3 |      150 |      2.50 |
| 9770618260320 | ACOK       |        3 |      150 |      3.47 |
| 9780618260300 | The Hobbit |        2 |      366 |      3.40 |
| 9780618260301 | LOTR 1     |        2 |      350 |      4.00 |
| 9780618260320 | LOTR 2     |        2 |      150 |      2.50 |
| 9781560974321 | Palestine  |        1 |      288 |      4.20 |
| 9880618260320 | AGOT       |        3 |      150 |      2.50 |
+---------------+------------+----------+----------+-----------+
7 rows in set (0.00 sec)
```

2. Book Table



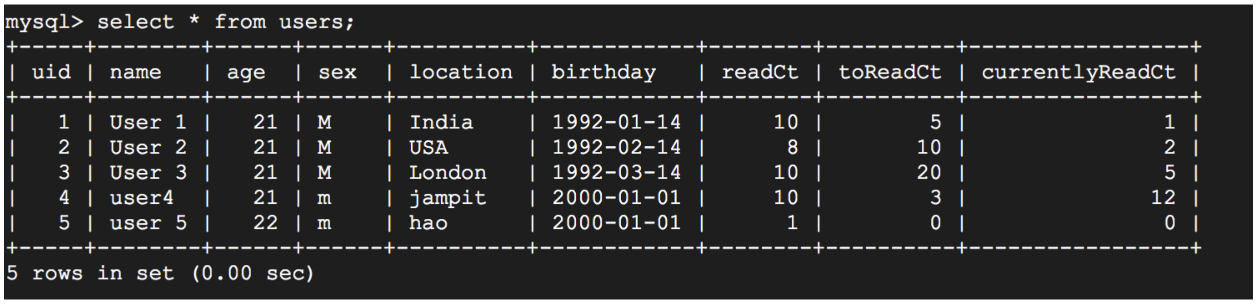
```
mysql> select * from shelf;
+-----+---------------+-------------------+--------+------------+------------+
| uid | isbn          | name              | rating | dateRead   | dateAdded  |
+-----+---------------+-------------------+--------+------------+------------+
|   1 | 9770618260320 | Read              |   5.00 | 2000-01-01 | 2000-02-02 |
|   1 | 9780618260301 | Read              |   5.00 | 2000-01-01 | 2000-01-01 |
|   1 | 9780618260320 | Read              |   5.00 | 2000-01-01 | 2000-02-02 |
|   1 | 9781560974321 | Read              |   5.00 | 2000-01-01 | 2000-02-02 |
|   2 | 9770618260320 | To-Read           |   2.70 | NULL       | 2000-02-02 |
|   2 | 9781560974321 | To-Read           |   2.70 | NULL       | 2000-02-02 |
|   3 | 9770618260320 | Read              |   5.00 | 1999-12-11 | 2000-01-01 |
|   3 | 9780618260301 | Currently-Reading |   2.00 | NULL       | 2000-01-01 |
|   4 | 9780618260300 | Currently-Reading |   5.00 | NULL       | 2000-01-01 |
|   4 | 9780618260301 | Currently-Reading |   5.00 | NULL       | 2000-01-01 |
|   4 | 9780618260320 | Read              |   4.00 | NULL       | 2000-02-02 |
|   5 | 9780618260300 | Read              |   3.00 | NULL       | NULL       |
+-----+---------------+-------------------+--------+------------+------------+
12 rows in set (0.00 sec)

mysql>
```

3. Shelf Table



```
mysql> select * from friends;
+-----+-----+
| uid | fid |
+-----+-----+
|   2 |   1 |
|   3 |   1 |
|   1 |   2 |
|   1 |   3 |
+-----+-----+
4 rows in set (0.00 sec)
```

4. Friends Table



```
mysql> select * from users;
+-----+--------+------+------+----------+------------+--------+----------+-----------------+
| uid | name   | age  | sex  | location | birthday   | readCt | toReadCt | currentlyReadCt |
+-----+--------+------+------+----------+------------+--------+----------+-----------------+
|   1 | User 1 |   21 | M    | India    | 1992-01-14 |     10 |        5 |               1 |
|   2 | User 2 |   21 | M    | USA      | 1992-02-14 |      8 |       10 |               2 |
|   3 | User 3 |   21 | M    | London   | 1992-03-14 |     10 |       20 |               5 |
|   4 | user4  |   21 | m    | jampit   | 2000-01-01 |     10 |        3 |              12 |
|   5 | user 5 |   22 | m    | hao      | 2000-01-01 |      1 |        0 |               0 |
+-----+--------+------+------+----------+------------+--------+----------+-----------------+
5 rows in set (0.00 sec)
```

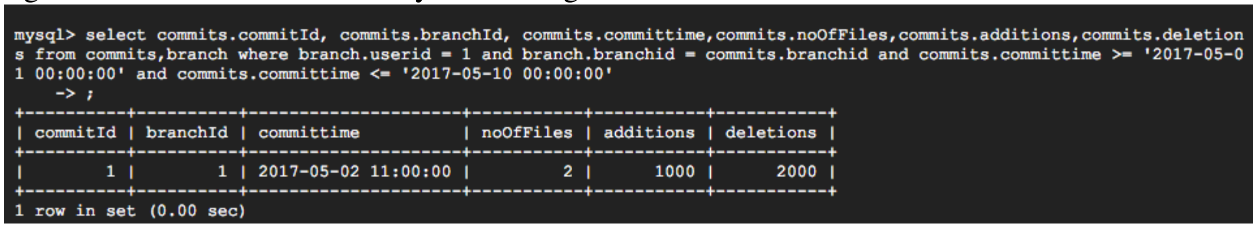
5. Users Table

2. GitHub database

Q1 1. Find the commits made by user X in all branches of a repository between 1st May and 10th May.

Query: select commits.commitId, commits.branchId, commits.committime,commits.noOfFiles,commits.additions,commits.deletions from commits,branch where branch.userid = 1 and branch.branchid = commits.branchid and commits.committime >= '2017-05-01 00:00:00' and commits.committime <= '2017-05-10 00:00:00';

Explanation: In this question, we assume that the uid of user X is 1. We find the right commit record for user x by connecting commit table and branch table.

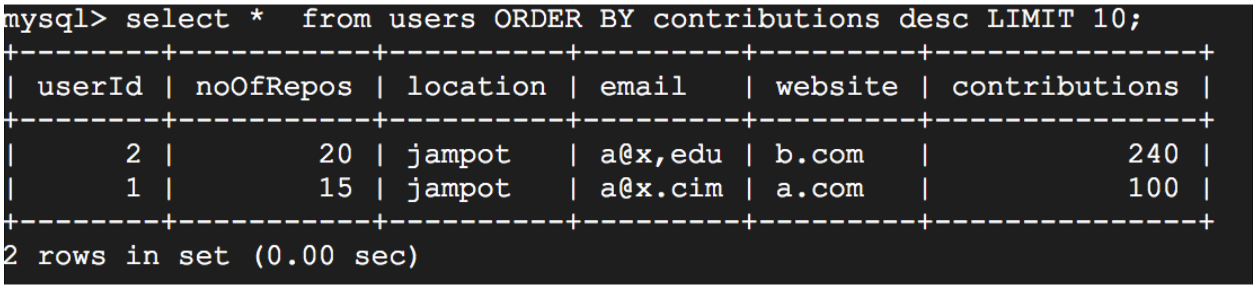


```
mysql> select commits.commitId, commits.branchId, commits.committime,commits.noOfFiles,commits.additions,commits.deletion
s from commits,branch where branch.userid = 1 and branch.branchid = commits.branchid and commits.committime >= '2017-05-0
1 00:00:00' and commits.committime <= '2017-05-10 00:00:00'
    -> ;
+----------+----------+---------------------+-----------+-----------+-----------+
| commitId | branchId | committime          | noOfFiles | additions | deletions |
+----------+----------+---------------------+-----------+-----------+-----------+
|        1 |        1 | 2017-05-02 11:00:00 |         2 |      1000 |      2000 |
+----------+----------+---------------------+-----------+-----------+-----------+
1 row in set (0.00 sec)
```

Q2 2. Display the top ten users who made the most contributions.

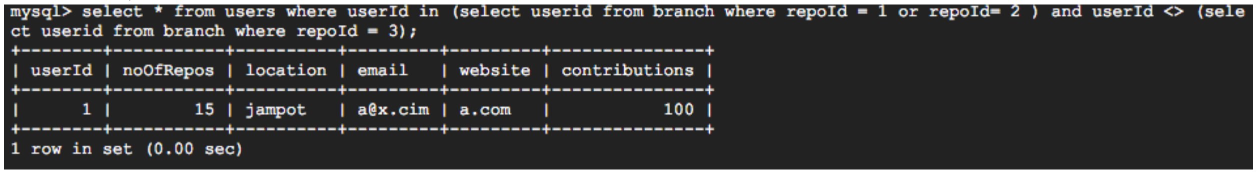
Query: select * from users ORDER BY contributions desc LIMIT 10;

Explanation: This sql sentences is pretty straight



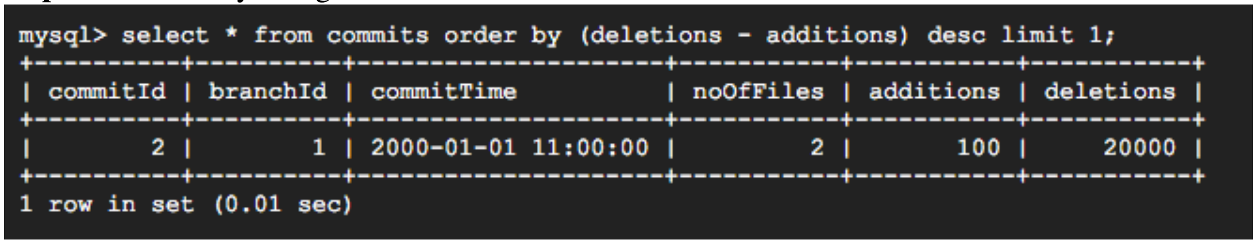
```
mysql> select *  from users ORDER BY contributions desc LIMIT 10;
+--------+-----------+----------+---------+---------+---------------+
| userId | noOfRepos | location | email   | website | contributions |
+--------+-----------+----------+---------+---------+---------------+
|      2 |        20 | jampot   | a@x,edu | b.com   |           240 |
|      1 |        15 | jampot   | a@x.cim | a.com   |           100 |
+--------+-----------+----------+---------+---------+---------------+
2 rows in set (0.00 sec)
```

Q3 3. Find the users who made branches of either of repositories X or Y but not of a repository Z.
Query: select * from users where userId in (select userid from branch where repoId = 1 or repoId= 2) and userId <> (select userid from branch where repoId = 3);
Explanation: For this question, we assume that the ids of repositories X,Y and Z are given and they are 1,2 and 3. First search the user's id from branch table which satisfy the condition, then take user information from the users table based on the user's IDs.



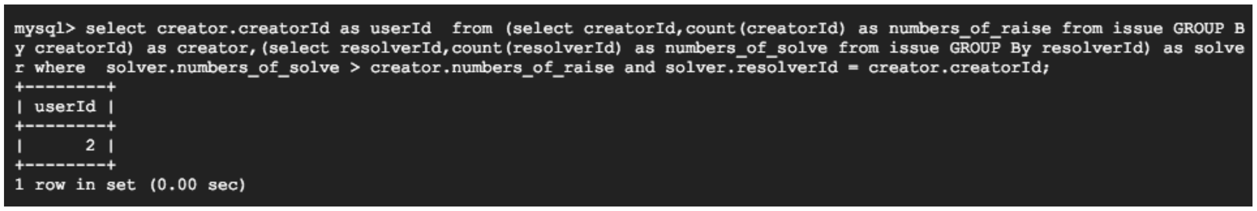
```
mysql> select * from users where userId in (select userid from branch where repoId = 1 or repoId= 2 ) and userId <> (sele
ct userid from branch where repoId = 3);
+--------+-----------+----------+---------+---------+---------------+
| userId | noOfRepos | location | email   | website | contributions |
+--------+-----------+----------+---------+---------+---------------+
|      1 |        15 | jampot   | a@x.cim | a.com   |           100 |
+--------+-----------+----------+---------+---------+---------------+
1 row in set (0.00 sec)
```

Q4 4. Find the top commit with the highest lines of code reduced. (Hint: We need to find the maximized value of: number of deletions - number of additions in each commit).
Query: select * from commits order by (deletions - additions) desc limit 1;
Explanation: Very straight forward.



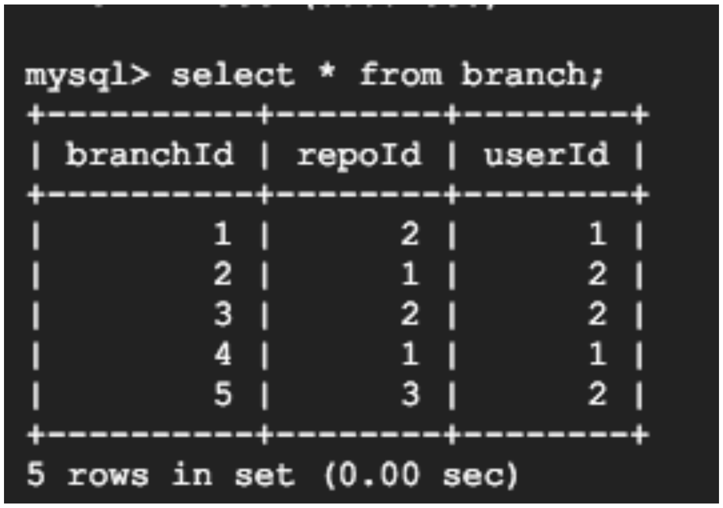
```
mysql> select * from commits order by (deletions - additions) desc limit 1;
+----------+----------+---------------------+-----------+-----------+-----------+
| commitId | branchId | commitTime          | noOfFiles | additions | deletions |
+----------+----------+---------------------+-----------+-----------+-----------+
|        2 |        1 | 2000-01-01 11:00:00 |         2 |       100 |     20000 |
+----------+----------+---------------------+-----------+-----------+-----------+
1 row in set (0.01 sec)
```

Q5 5.List the users who solved more issues than they raised. (i.e. number of issues in which they were the resolver is greater than the number of issues where they were the creator.)
Query: select creator.creatorId as userId from (select creatorId,count(creatorId) as numbers_of_raise from issue GROUP By creatorId) as creator,(select resolverId,count(resolverId) as numbers_of_solve from issue GROUP By resolverId) as solver where solver.numbers_of_solve > creator.numbers_of_raise and solver.resolverId = creator.creatorId;
Explanation: First, count the number of every user's raised issues and count the number of every user's solved issues. Then calculate the subtraction between them and take the record which the number of sloved issues are bigger than number of raised issues.



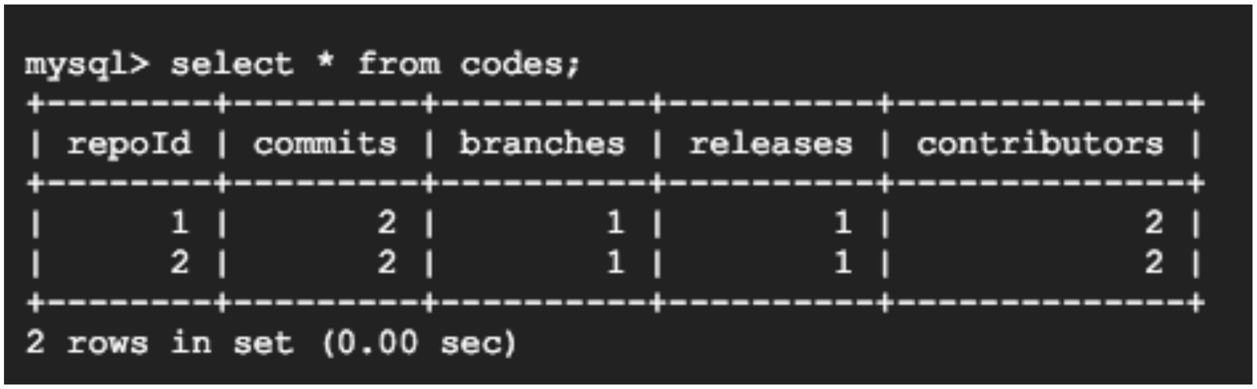
```
mysql> select creator.creatorId as userId  from (select creatorId,count(creatorId) as numbers_of_raise from issue GROUP B
y creatorId) as creator,(select resolverId,count(resolverId) as numbers_of_solve from issue GROUP By resolverId) as solve
r where  solver.numbers_of_solve > creator.numbers_of_raise and solver.resolverId = creator.creatorId;
+--------+
| userId |
+--------+
|      2 |
+--------+
1 row in set (0.00 sec)
```

GitHub Table Content:



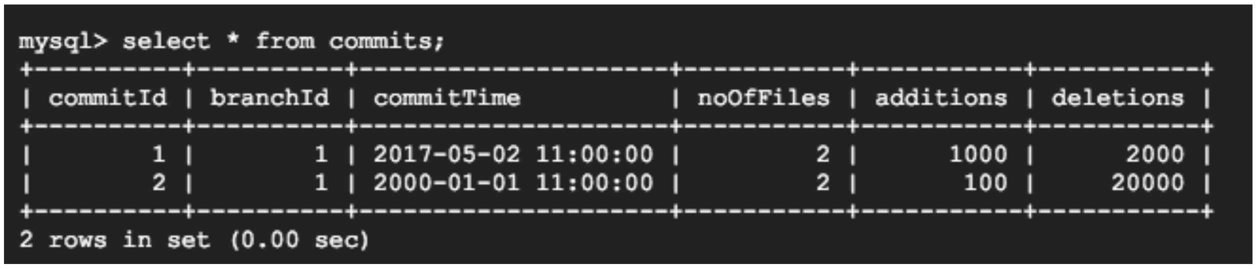
```
mysql> select * from branch;
+----------+--------+--------+
| branchId | repoId | userId |
+----------+--------+--------+
|        1 |      2 |      1 |
|        2 |      1 |      2 |
|        3 |      2 |      2 |
|        4 |      1 |      1 |
|        5 |      3 |      2 |
+----------+--------+--------+
5 rows in set (0.00 sec)
```

1. **Branch Table**



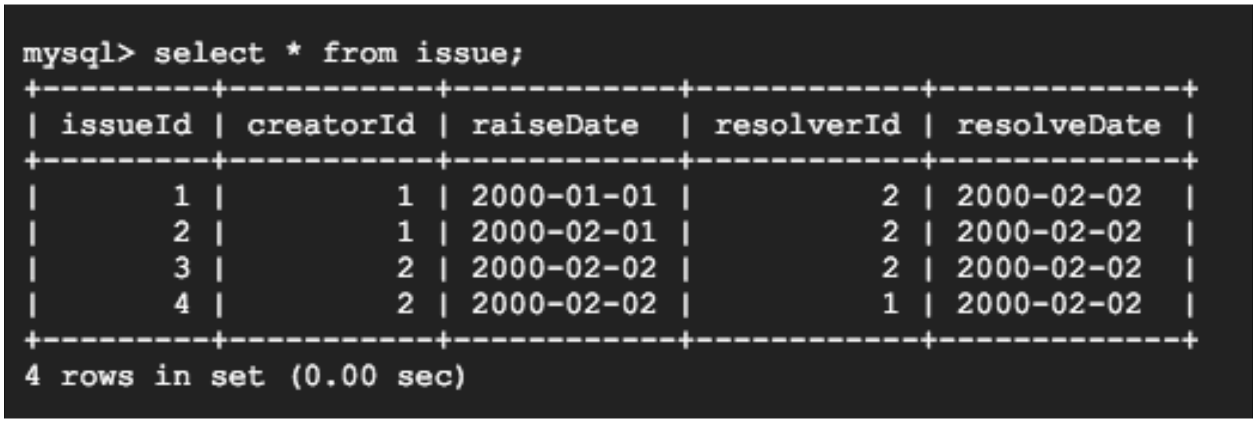
```
mysql> select * from codes;
+--------+---------+----------+----------+--------------+
| repoId | commits | branches | releases | contributors |
+--------+---------+----------+----------+--------------+
|      1 |       2 |        1 |        1 |            2 |
|      2 |       2 |        1 |        1 |            2 |
+--------+---------+----------+----------+--------------+
2 rows in set (0.00 sec)
```

2. **Codes Table**



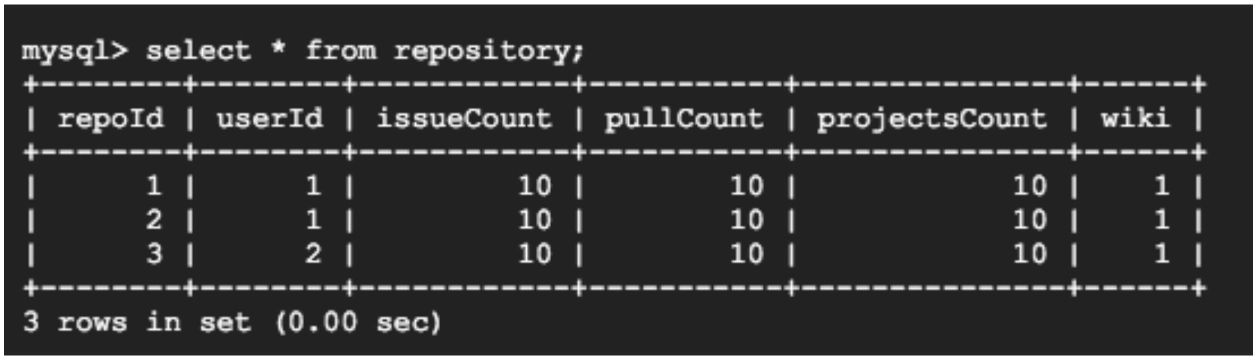
```
mysql> select * from commits;
+----------+----------+---------------------+-----------+-----------+-----------+
| commitId | branchId | commitTime          | noOfFiles | additions | deletions |
+----------+----------+---------------------+-----------+-----------+-----------+
|        1 |        1 | 2017-05-02 11:00:00 |         2 |      1000 |      2000 |
|        2 |        1 | 2000-01-01 11:00:00 |         2 |       100 |     20000 |
+----------+----------+---------------------+-----------+-----------+-----------+
2 rows in set (0.00 sec)
```

3. **Commits Table**



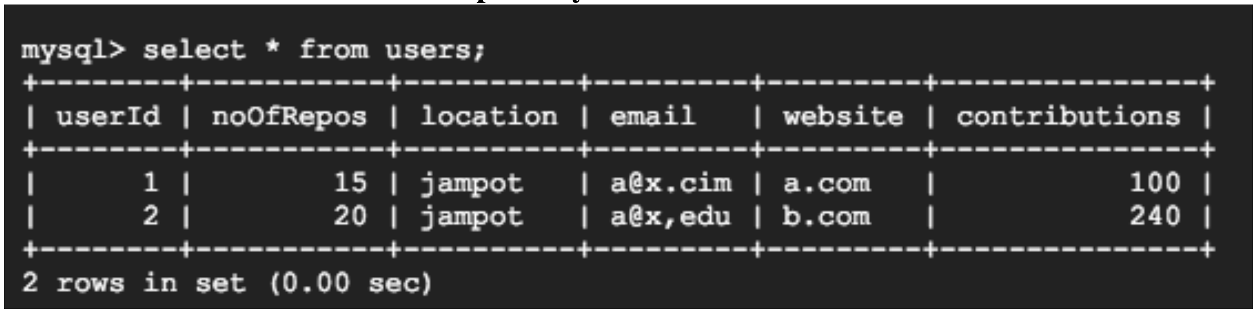
```
mysql> select * from issue;
+---------+-----------+------------+------------+-------------+
| issueId | creatorId | raiseDate  | resolverId | resolveDate |
+---------+-----------+------------+------------+-------------+
|       1 |         1 | 2000-01-01 |          2 | 2000-02-02  |
|       2 |         1 | 2000-02-01 |          2 | 2000-02-02  |
|       3 |         2 | 2000-02-02 |          2 | 2000-02-02  |
|       4 |         2 | 2000-02-02 |          1 | 2000-02-02  |
+---------+-----------+------------+------------+-------------+
4 rows in set (0.00 sec)
```

4. Issue Table



```
mysql> select * from repository;
+--------+--------+------------+-----------+---------------+------+
| repoId | userId | issueCount | pullCount | projectsCount | wiki |
+--------+--------+------------+-----------+---------------+------+
|      1 |      1 |         10 |        10 |            10 |    1 |
|      2 |      1 |         10 |        10 |            10 |    1 |
|      3 |      2 |         10 |        10 |            10 |    1 |
+--------+--------+------------+-----------+---------------+------+
3 rows in set (0.00 sec)
```

5. Repository Table



```
mysql> select * from users;
+--------+-----------+----------+---------+---------+---------------+
| userId | noOfRepos | location | email   | website | contributions |
+--------+-----------+----------+---------+---------+---------------+
|      1 |        15 | jampot   | a@x.cim | a.com   |           100 |
|      2 |        20 | jampot   | a@x,edu | b.com   |           240 |
+--------+-----------+----------+---------+---------+---------------+
2 rows in set (0.00 sec)
```

6. Users Table

3.Ad-targeting on YouTube using LinkedIn

1 For LinkedIn Database:

```
# ************************************************************
# Sequel Pro SQL dump
# Version 4541
#
# http://www.sequelpro.com/
# https://github.com/sequelpro/sequelpro
#
# Host: 127.0.0.1 (MySQL 5.7.18)
# Database: 585HWDB_LinkedIn
# Generation Time: 2017-06-11 23:59:39 +0000
# ************************************************************
```

```
/*!40101 SET
@OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET
@OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET
@OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8 */;
/*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,
FOREIGN_KEY_CHECKS=0 */;
/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;
/*!40111 SET @OLD_SQL_NOTES=@@SQL_NOTES, SQL_NOTES=0 */;


# Dump of table application
# ------------------------------------------------------------

DROP TABLE IF EXISTS `application`;

CREATE TABLE `application` (
  `userid` int(11) NOT NULL,
  `jobid` int(11) NOT NULL,
  PRIMARY KEY (`userid`,`jobid`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

LOCK TABLES `application` WRITE;
/*!40000 ALTER TABLE `application` DISABLE KEYS */;

INSERT INTO `application` (`userid`, `jobid`)
VALUES
	(1,1),
	(1,2),
	(1,3),
	(2,1),
	(2,2),
	(3,5),
	(5,5);

/*!40000 ALTER TABLE `application` ENABLE KEYS */;
UNLOCK TABLES;


# Dump of table experience
# ------------------------------------------------------------
```

```
DROP TABLE IF EXISTS `experience`;

CREATE TABLE `experience` (
  `userid` int(11) NOT NULL,
  `experienceid` int(11) NOT NULL,
  `title` varchar(50) DEFAULT NULL,
  `organizationid` int(11) NOT NULL,
  `from_where` varchar(50) DEFAULT NULL,
  `to_where` varchar(50) DEFAULT NULL,
  `location` varchar(50) DEFAULT NULL,
  `description` varchar(100) DEFAULT NULL,
  PRIMARY KEY (`experienceid`),
  KEY `userid` (`userid`),
  KEY `organizationid` (`organizationid`),
  CONSTRAINT `experience_ibfk_1` FOREIGN KEY (`userid`) REFERENCES `user`
(`userid`),
  CONSTRAINT `experience_ibfk_2` FOREIGN KEY (`organizationid`) REFERENCES
`organization` (`organizationid`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

LOCK TABLES `experience` WRITE;
/*!40000 ALTER TABLE `experience` DISABLE KEYS */;

INSERT INTO `experience` (`userid`, `experienceid`, `title`, `organizationid`,
`from_where`, `to_where`, `location`, `description`)
VALUES
	(1,1,'eg',1,NULL,NULL,NULL,NULL),
	(1,2,'eg',2,NULL,NULL,NULL,NULL),
	(2,3,'eg',1,NULL,NULL,NULL,NULL),
	(3,4,'eg',3,NULL,NULL,NULL,NULL),
	(4,5,'mg',4,NULL,NULL,NULL,NULL),
	(5,6,'mg',4,NULL,NULL,NULL,NULL);

/*!40000 ALTER TABLE `experience` ENABLE KEYS */;
UNLOCK TABLES;


# Dump of table jobs
# ------------------------------------------------------------

DROP TABLE IF EXISTS `jobs`;

CREATE TABLE `jobs` (
  `jobid` int(11) NOT NULL,
  `dateposted` date DEFAULT NULL,
  `organizationid` int(11) NOT NULL,
  `genre` varchar(50) DEFAULT NULL,
  PRIMARY KEY (`jobid`),
  KEY `organizationid` (`organizationid`),
  CONSTRAINT `jobs_ibfk_1` FOREIGN KEY (`organizationid`) REFERENCES
`organization` (`organizationid`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

LOCK TABLES `jobs` WRITE;
/*!40000 ALTER TABLE `jobs` DISABLE KEYS */;

INSERT INTO `jobs` (`jobid`, `dateposted`, `organizationid`, `genre`)
VALUES
	(1,'2017-06-06',1,'a'),
	(2,'2017-06-06',2,'b'),
	(3,'2017-06-06',2,'c'),
	(4,'2017-06-06',3,'d'),
	(5,'2017-07-07',5,'cc');

/*!40000 ALTER TABLE `jobs` ENABLE KEYS */;
UNLOCK TABLES;


# Dump of table network
# ------------------------------------------------------------

DROP TABLE IF EXISTS `network`;

CREATE TABLE `network` (
  `userid` int(11) NOT NULL,
  `friendid` int(11) NOT NULL,
  `dateAdded` date DEFAULT NULL,
  `note` varchar(100) DEFAULT NULL,
  PRIMARY KEY (`userid`,`friendid`),
  KEY `friendid` (`friendid`),
  CONSTRAINT `network_ibfk_1` FOREIGN KEY (`userid`) REFERENCES `user`
(`userid`),
  CONSTRAINT `network_ibfk_2` FOREIGN KEY (`friendid`) REFERENCES `user`
(`userid`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

LOCK TABLES `network` WRITE;
/*!40000 ALTER TABLE `network` DISABLE KEYS */;

INSERT INTO `network` (`userid`, `friendid`, `dateAdded`, `note`)
VALUES
```

```
	(1,2,'2017-06-06',NULL),
	(1,3,'2017-06-06',NULL),
	(2,3,'2017-06-06',NULL),
	(3,5,'2017-06-06',NULL),
	(4,3,'2017-06-06',NULL),
	(5,1,'2017-06-06',NULL);

/*!40000 ALTER TABLE `network` ENABLE KEYS */;
UNLOCK TABLES;


# Dump of table organization
# ------------------------------------------------------------

DROP TABLE IF EXISTS `organization`;

CREATE TABLE `organization` (
  `organizationid` int(11) NOT NULL,
  `name` varchar(50) DEFAULT NULL,
  PRIMARY KEY (`organizationid`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

LOCK TABLES `organization` WRITE;
/*!40000 ALTER TABLE `organization` DISABLE KEYS */;

INSERT INTO `organization` (`organizationid`, `name`)
VALUES
	(1,'O1'),
	(2,'O2'),
	(3,'O3'),
	(4,'O4'),
	(5,'O5'),
	(6,'O6');

/*!40000 ALTER TABLE `organization` ENABLE KEYS */;
UNLOCK TABLES;


# Dump of table User
# ------------------------------------------------------------

DROP TABLE IF EXISTS `User`;

CREATE TABLE `User` (
  `userid` int(11) NOT NULL,
  `email` varchar(50) DEFAULT NULL,
  `handle` varchar(50) DEFAULT NULL,
  `headline` varchar(50) DEFAULT NULL,
  `location` varchar(50) DEFAULT NULL,
  PRIMARY KEY (`userid`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

LOCK TABLES `User` WRITE;
/*!40000 ALTER TABLE `User` DISABLE KEYS */;

INSERT INTO `User` (`userid`, `email`, `handle`, `headline`, `location`)
VALUES
	(1,'1@gmail.com','?','?','LA'),
	(2,'2@gmail.com',NULL,NULL,'BA'),
	(3,'3@gmail.com',NULL,NULL,'MA'),
	(4,'4@gmail.com',NULL,NULL,'CA'),
	(5,'5@gmail.com',NULL,NULL,'SF'),
	(6,'5@gmail.com',NULL,NULL,'SD');

/*!40000 ALTER TABLE `User` ENABLE KEYS */;
UNLOCK TABLES;



/*!40111 SET SQL_NOTES=@OLD_SQL_NOTES */;
/*!40101 SET SQL_MODE=@OLD_SQL_MODE */;
/*!40014 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS */;
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT
*/;
/*!40101 SET
CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION
*/;
```

2 For YouTube Table:

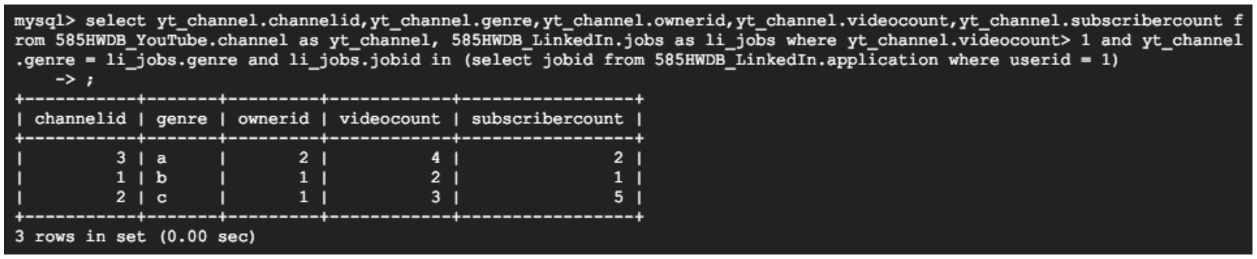
```
# ************************************************************
# Sequel Pro SQL dump
# Version 4541
#
# http://www.sequelpro.com/
# https://github.com/sequelpro/sequelpro
#
# Host: 127.0.0.1 (MySQL 5.7.18)
# Database: 585HWDB_YouTube
# Generation Time: 2017-06-11 23:59:47 +0000
# ************************************************************
```

```
/*!40101 SET
@OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET
@OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET
@OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8 */;
/*!40014 SET
@OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,
FOREIGN_KEY_CHECKS=0 */;
/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;
/*!40111 SET @OLD_SQL_NOTES=@@SQL_NOTES, SQL_NOTES=0 */;


# Dump of table channel
# ------------------------------------------------------------

DROP TABLE IF EXISTS `channel`;

CREATE TABLE `channel` (
  `channelid` int(11) NOT NULL,
  `genre` varchar(50) DEFAULT NULL,
  `ownerid` int(11) NOT NULL,
  `videocount` int(11) DEFAULT NULL,
  `subscribercount` int(11) DEFAULT NULL,
  PRIMARY KEY (`channelid`),
  KEY `ownerid` (`ownerid`),
  CONSTRAINT `channel_ibfk_1` FOREIGN KEY (`ownerid`) REFERENCES
`user` (`userid`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

LOCK TABLES `channel` WRITE;
/*!40000 ALTER TABLE `channel` DISABLE KEYS */;

INSERT INTO `channel` (`channelid`, `genre`, `ownerid`, `videocount`,
`subscribercount`)
VALUES
      (1,'b',1,2,1),
      (2,'c',1,3,5),
      (3,'a',2,4,2),
      (4,'cc',3,5,1);

/*!40000 ALTER TABLE `channel` ENABLE KEYS */;
UNLOCK TABLES;
```

```
# Dump of table subscription
# ------------------------------------------------------------

DROP TABLE IF EXISTS `subscription`;

CREATE TABLE `subscription` (
  `channelid` int(11) NOT NULL,
  `userid` int(11) NOT NULL,
  PRIMARY KEY (`channelid`,`userid`),
  KEY `userid` (`userid`),
  CONSTRAINT `subscription_ibfk_1` FOREIGN KEY (`channelid`)
REFERENCES `channel` (`channelid`),
  CONSTRAINT `subscription_ibfk_2` FOREIGN KEY (`userid`) REFERENCES
`user` (`userid`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;



# Dump of table user
# ------------------------------------------------------------

DROP TABLE IF EXISTS `user`;

CREATE TABLE `user` (
  `userid` int(11) NOT NULL,
  `email` varchar(50) DEFAULT NULL,
  `phone` int(11) DEFAULT NULL,
  `handle` varchar(50) DEFAULT NULL,
  `name` varchar(50) DEFAULT NULL,
  `favoriteGenre` varchar(50) DEFAULT NULL,
  `location` varchar(100) DEFAULT NULL,
  PRIMARY KEY (`userid`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

LOCK TABLES `user` WRITE;
/*!40000 ALTER TABLE `user` DISABLE KEYS */;

INSERT INTO `user` (`userid`, `email`, `phone`, `handle`, `name`, `favoriteGenre`,
`location`)
VALUES
	(1,'1@gmail.com',213,NULL,NULL,'a','LA'),
	(2,'2@gmail.com',213,NULL,NULL,'b','BA'),
	(3,'3@gmail.com',213,NULL,NULL,'c','MA'),
	(4,'4@gmail.com',213,NULL,NULL,'d','CA'),
     (5,'5@gmail.com',213,NULL,NULL,'a','SF'),
     (6,'6@gmail.com',213,NULL,NULL,'cc','SD');

/*!40000 ALTER TABLE `user` ENABLE KEYS */;
UNLOCK TABLES;


# Dump of table video
# ------------------------------------------------------------

DROP TABLE IF EXISTS `video`;

CREATE TABLE `video` (
  `videoid` int(11) NOT NULL,
  `channelid` int(11) NOT NULL,
  PRIMARY KEY (`videoid`),
  KEY `channelid` (`channelid`),
  CONSTRAINT `video_ibfk_1` FOREIGN KEY (`channelid`) REFERENCES
`channel` (`channelid`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

LOCK TABLES `video` WRITE;
/*!40000 ALTER TABLE `video` DISABLE KEYS */;

INSERT INTO `video` (`videoid`, `channelid`)
VALUES
     (1,1),
     (3,1),
     (2,2),
     (4,2),
     (5,2),
     (6,3),
     (7,3),
     (8,3),
     (9,3),
     (10,4),
     (11,4),
     (12,4),
     (13,4),
     (14,4);

/*!40000 ALTER TABLE `video` ENABLE KEYS */;
UNLOCK TABLES;
```

```
/*!40111 SET SQL_NOTES=@OLD_SQL_NOTES */;
/*!40101 SET SQL_MODE=@OLD_SQL_MODE */;
/*!40014 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS */;
/*!40101 SET
CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET
CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET
COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
```

Q1 1. List all channels that have more than 50 videos and have the same genres as jobs searched on LinkedIn by user X.
Query: select yt_channel.channelid,yt_channel.genre,yt_channel.ownerid,yt_channel.videocount,yt_channel.subscribercount from 585HWDB_YouTube.channel as yt_channel, 585HWDB_LinkedIn.jobs as li_jobs where yt_channel.videocount> 1 and yt_channel.genre = li_jobs.genre and li_jobs.jobid in (select jobid from 585HWDB_LinkedIn.application where userid = 1)
Explanation: For this question, we assume that the user X's id is 1. Because it's too hard to make half hundred fake record, I change the condition 'more than 50 videos' to 'more than 2 videos'. First find all the jobs that user x applied. Then list all the channel information that matches search condition.



```
mysql> select yt_channel.channelid,yt_channel.genre,yt_channel.ownerid,yt_channel.videocount,yt_channel.subscribercount f
rom 585HWDB_YouTube.channel as yt_channel, 585HWDB_LinkedIn.jobs as li_jobs where yt_channel.videocount> 1 and yt_channel
.genre = li_jobs.genre and li_jobs.jobid in (select jobid from 585HWDB_LinkedIn.application where userid = 1)
    -> ;
+-----------+-------+---------+------------+-----------------+
| channelid | genre | ownerid | videocount | subscribercount |
+-----------+-------+---------+------------+-----------------+
|         3 | a     |       2 |          4 |               2 |
|         1 | b     |       1 |          2 |               1 |
|         2 | c     |       1 |          3 |               5 |
+-----------+-------+---------+------------+-----------------+
3 rows in set (0.00 sec)
```

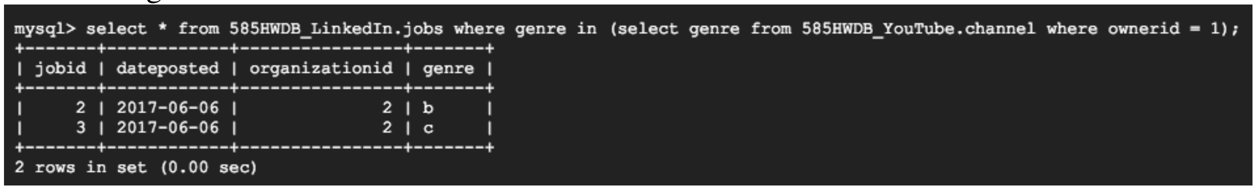
Q2 2. Delete all channels that have less than 50 subscribers but move their videos to the channel with highest number of subscribers in the same genre.
Query:
1. update 585HWDB_YouTube.video as video,585HWDB_YouTube.channel as channel set video.channelid = (select channelid from channel where genre = (select genre from channel where subscribercount < 50 order by subscribercount limit 1) order by subscribercount desc limit 1) where video.channelid = (select channelid from channel where channel.subscribercount < 50 order by channel.subscribercount limit 1;

2. delete from 585HWDB_YouTube.subscription as subscription where channelid = (select channelid from channel where channel.subscribercount < 50 order by channel.subscribercount limit 1)

3. delete from 585HWDB_YouTube.channel as channel where channelid = (select channelid from channel where channel.subscribercount < 50 order by channel.subscribercount limit 1)

Explanation: These three sentences can only delete one channel which have less than 50 subscribers at one time. So if we want to delete all the eligible channels, we can run these sentences multiple times until empty set appeared. Another thing needs to mention is that every video can be moved to other channel.
First move the video to other channel. Secondly, delete subscription information. Finally, delete channel information.

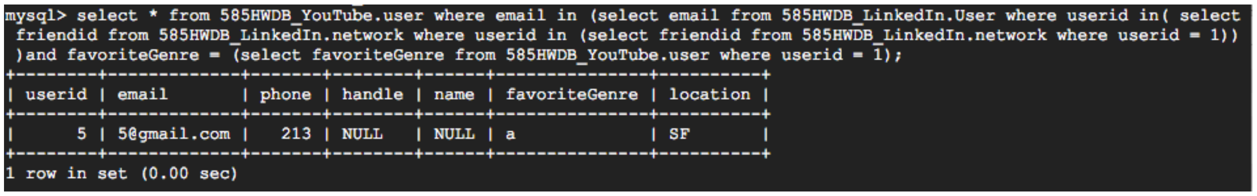
Q3 3. Find all jobs on LinkedIn whose genre matches Owner-ID X's channel's genre.
Query: select * from 585HWDB_LinkedIn.jobs where genre in (select genre from 585HWDB_YouTube.channel where ownerid = 1);
Explanation: For this question, we assume that the owner x's id is 1. First we have to get X's channel genre. Then find the jobs from LinkedIn database and its jobs table which have same genre.



```
mysql> select * from 585HWDB_LinkedIn.jobs where genre in (select genre from 585HWDB_YouTube.channel where ownerid = 1);
+-------+------------+----------------+-------+
| jobid | dateposted | organizationid | genre |
+-------+------------+----------------+-------+
|     2 | 2017-06-06 |              2 | b     |
|     3 | 2017-06-06 |              2 | c     |
+-------+------------+----------------+-------+
2 rows in set (0.00 sec)
```

Q4 4.Find friends of friends of User X on LinkedIn who have subscribed to channels with the genre that matches user X's favorite genre on YouTube. (This can be used to suggest friends.)
Query: select * from 585HWDB_YouTube.user where email in (select email from 585HWDB_LinkedIn.User where userid in(select friendid from 585HWDB_LinkedIn.network where userid in (select friendid from 585HWDB_LinkedIn.network where userid = 1)))and favoriteGenre = (select favoriteGenre from 585HWDB_YouTube.user where userid = 1);

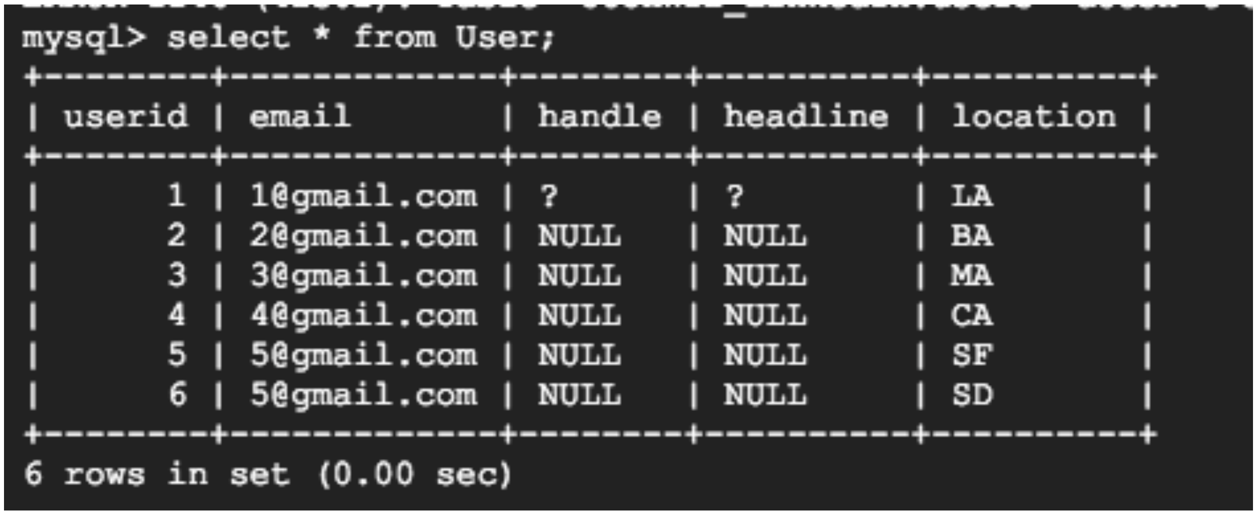
Explanation: For this question, we assume that user X's id is 1. First search X's favorite genre. Then search friends of friends of User X which subscribed the same genre channels.



```
mysql> select * from 585HWDB_YouTube.user where email in (select email from 585HWDB_LinkedIn.User where userid in( select
 friendid from 585HWDB_LinkedIn.network where userid in (select friendid from 585HWDB_LinkedIn.network where userid = 1))
 )and favoriteGenre = (select favoriteGenre from 585HWDB_YouTube.user where userid = 1);
+--------+-------------+-------+--------+------+---------------+----------+
| userid | email       | phone | handle | name | favoriteGenre | location |
+--------+-------------+-------+--------+------+---------------+----------+
|      5 | 5@gmail.com |   213 | NULL   | NULL | a             | SF       |
+--------+-------------+-------+--------+------+---------------+----------+
1 row in set (0.00 sec)
```

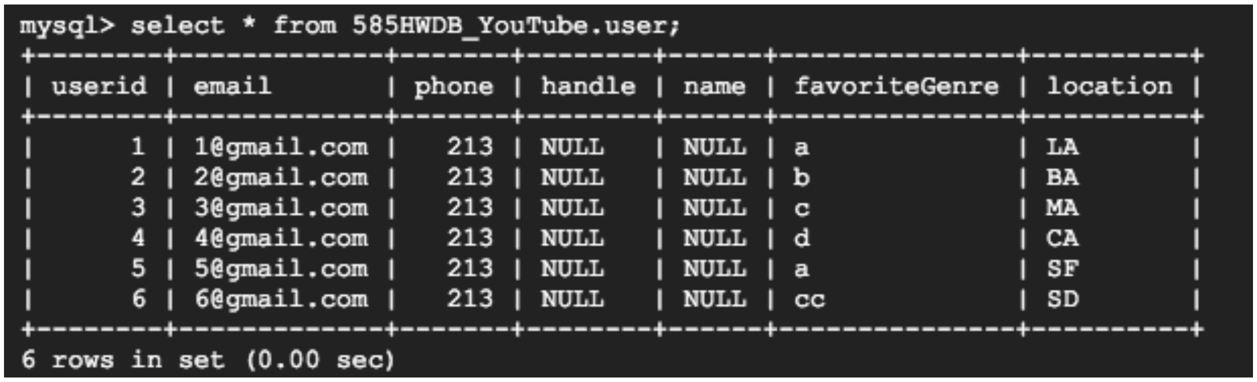
Q5 Add a column "location" to YouTube's User table based on the location associated with their LinkedIn account.
Query:
1. alter table 585HWDB_YouTube.user add location varchar(100);
2. update 585HWDB_YouTube.user,585HWDB_LinkedIn.user set 585HWDB_YouTube.user.location=585HWDB_LinkedIn.user.location where 585HWDB_YouTube.user.email=585HWDB_LinkedIn.user.email;

Explanation: First add a new column to YouTube's user table and named it 'location'. Then copy location information from LinkedIn's user table to YouTube's user table.



```
mysql> select * from User;
+--------+------------+--------+----------+----------+
| userid | email      | handle | headline | location |
+--------+------------+--------+----------+----------+
|      1 | 1@gmail.com | ?      | ?        | LA       |
|      2 | 2@gmail.com | NULL   | NULL     | BA       |
|      3 | 3@gmail.com | NULL   | NULL     | MA       |
|      4 | 4@gmail.com | NULL   | NULL     | CA       |
|      5 | 5@gmail.com | NULL   | NULL     | SF       |
|      6 | 5@gmail.com | NULL   | NULL     | SD       |
+--------+------------+--------+----------+----------+
6 rows in set (0.00 sec)
```

LinkedIn user table



```
mysql> select * from 585HWDB_YouTube.user;
+--------+------------+-------+--------+------+---------------+----------+
| userid | email      | phone | handle | name | favoriteGenre | location |
+--------+------------+-------+--------+------+---------------+----------+
|      1 | 1@gmail.com |   213 | NULL   | NULL | a             | LA       |
|      2 | 2@gmail.com |   213 | NULL   | NULL | b             | BA       |
|      3 | 3@gmail.com |   213 | NULL   | NULL | c             | MA       |
|      4 | 4@gmail.com |   213 | NULL   | NULL | d             | CA       |
|      5 | 5@gmail.com |   213 | NULL   | NULL | a             | SF       |
|      6 | 6@gmail.com |   213 | NULL   | NULL | cc            | SD       |
+--------+------------+-------+--------+------+---------------+----------+
6 rows in set (0.00 sec)
```

YouTube user table