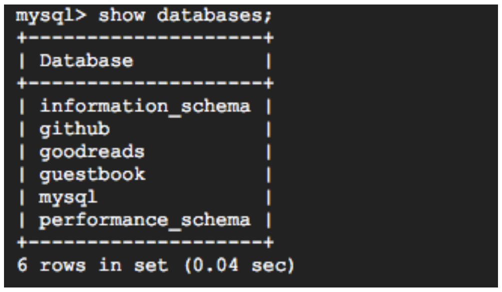
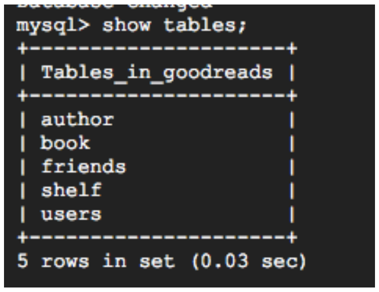
1. **Set 1 - GoodReads Database**

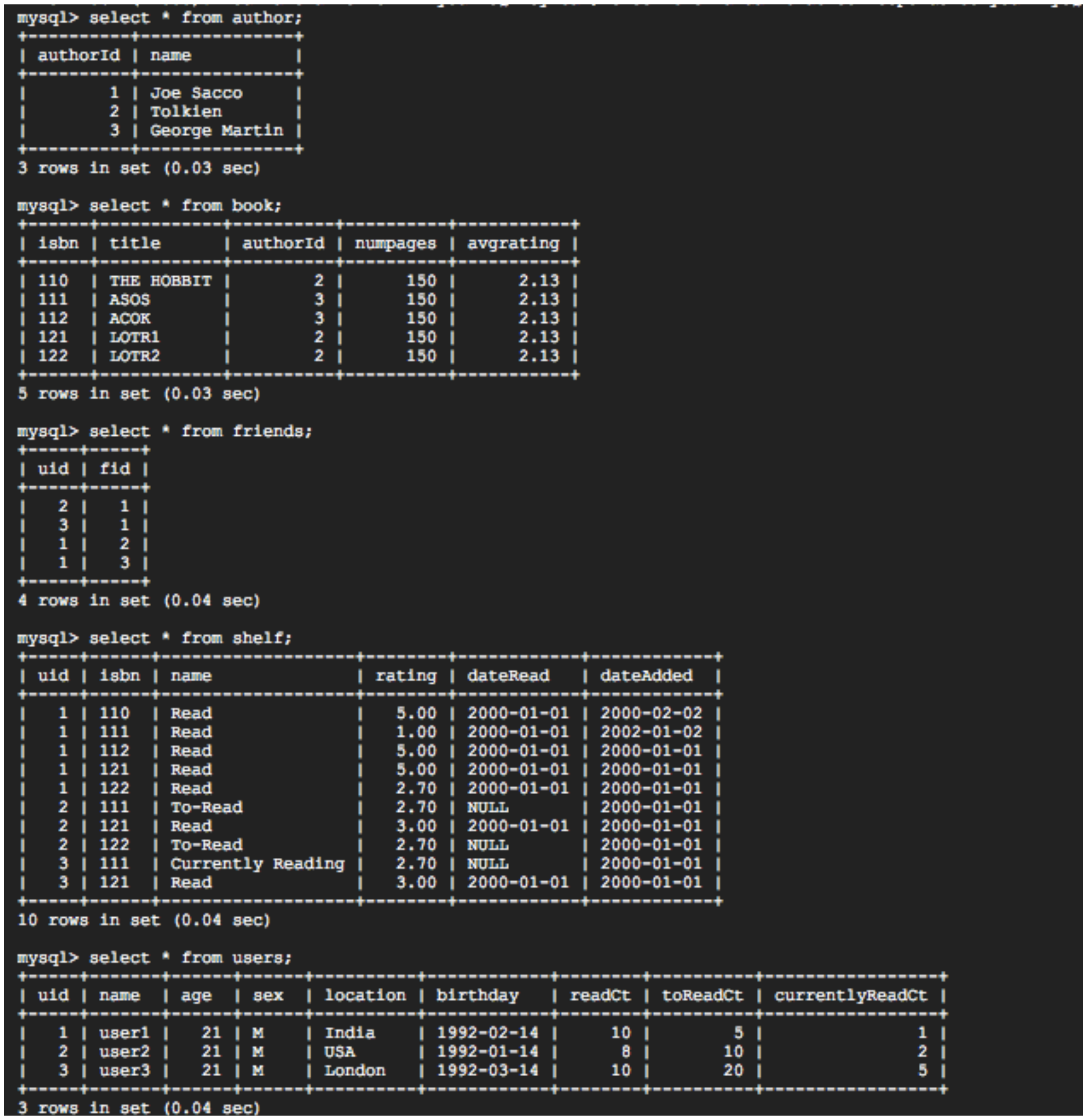
Starter Code is used to create tables in the cloudDB and data is inserted into the database using insert statement.



```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| github             |
| goodreads          |
| guestbook          |
| mysql              |
| performance_schema |
+--------------------+
6 rows in set (0.04 sec)
```



```
mysql> show tables;
+---------------------+
| Tables_in_goodreads |
+---------------------+
| author              |
| book                |
| friends             |
| shelf               |
| users               |
+---------------------+
5 rows in set (0.03 sec)
```

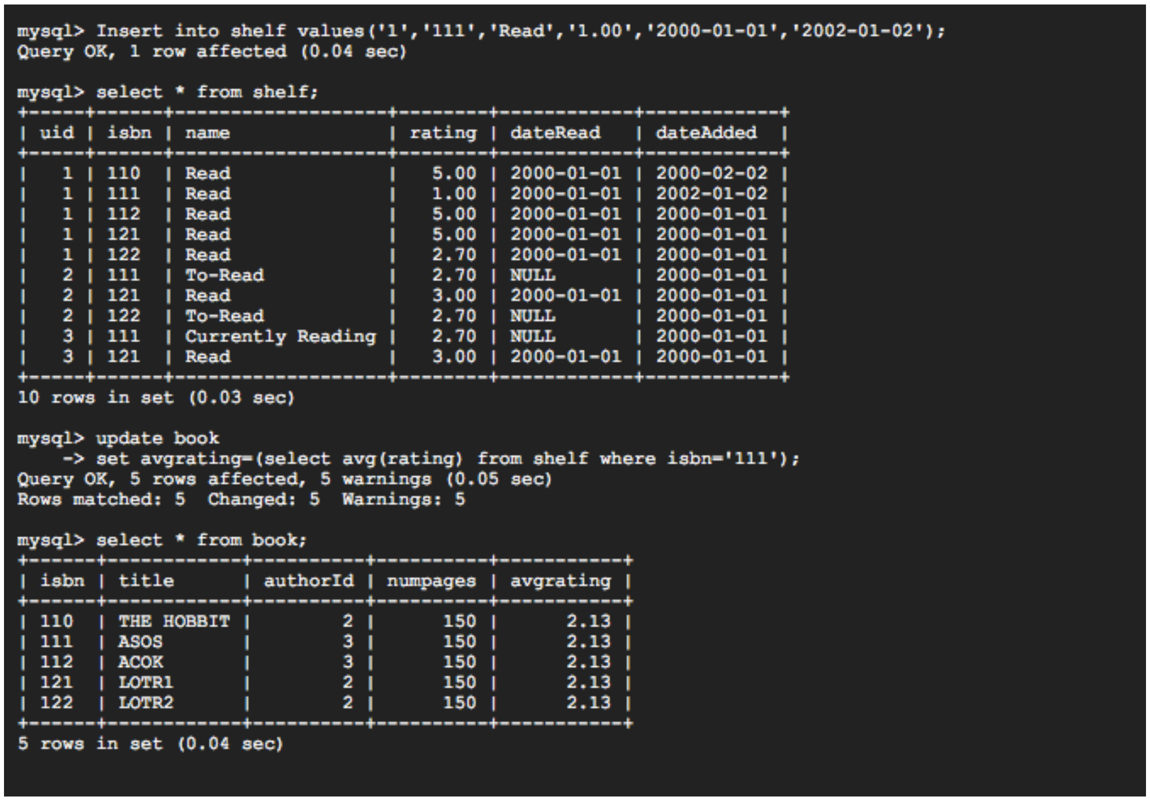


```
mysql> select * from author;
+----------+---------------+
| authorId | name          |
+----------+---------------+
|        1 | Joe Sacco     |
|        2 | Tolkien       |
|        3 | George Martin |
+----------+---------------+
3 rows in set (0.03 sec)

mysql> select * from book;
+------+------------+----------+----------+-----------+
| isbn | title      | authorId | numpages | avgrating |
+------+------------+----------+----------+-----------+
| 110  | THE HOBBIT |        2 |      150 |      2.13 |
| 111  | ASOS       |        3 |      150 |      2.13 |
| 112  | ACOK       |        3 |      150 |      2.13 |
| 121  | LOTR1      |        2 |      150 |      2.13 |
| 122  | LOTR2      |        2 |      150 |      2.13 |
+------+------------+----------+----------+-----------+
5 rows in set (0.03 sec)

mysql> select * from friends;
+-----+-----+
| uid | fid |
+-----+-----+
|   2 |   1 |
|   3 |   1 |
|   1 |   2 |
|   1 |   3 |
+-----+-----+
4 rows in set (0.04 sec)

mysql> select * from shelf;
+-----+------+-------------------+--------+------------+------------+
| uid | isbn | name              | rating | dateRead   | dateAdded  |
+-----+------+-------------------+--------+------------+------------+
|   1 | 110  | Read              |   5.00 | 2000-01-01 | 2000-02-02 |
|   1 | 111  | Read              |   1.00 | 2000-01-01 | 2002-01-02 |
|   1 | 112  | Read              |   5.00 | 2000-01-01 | 2000-01-01 |
|   1 | 121  | Read              |   5.00 | 2000-01-01 | 2000-01-01 |
|   1 | 122  | Read              |   2.70 | 2000-01-01 | 2000-01-01 |
|   2 | 111  | To-Read           |   2.70 | NULL       | 2000-01-01 |
|   2 | 121  | Read              |   3.00 | 2000-01-01 | 2000-01-01 |
|   2 | 122  | To-Read           |   2.70 | NULL       | 2000-01-01 |
|   3 | 111  | Currently Reading |   2.70 | NULL       | 2000-01-01 |
|   3 | 121  | Read              |   3.00 | 2000-01-01 | 2000-01-01 |
+-----+------+-------------------+--------+------------+------------+
10 rows in set (0.04 sec)

mysql> select * from users;
+-----+-------+------+------+----------+------------+--------+----------+----------------+
| uid | name  | age  | sex  | location | birthday   | readCt | toReadCt | currentlyReadCt |
+-----+-------+------+------+----------+------------+--------+----------+----------------+
|   1 | user1 |   21 | M    | India    | 1992-02-14 |     10 |        5 |               1 |
|   2 | user2 |   21 | M    | USA      | 1992-01-14 |      8 |       10 |               2 |
|   3 | user3 |   21 | M    | London   | 1992-03-14 |     10 |       20 |               5 |
+-----+-------+------+------+----------+------------+--------+----------+----------------+
3 rows in set (0.04 sec)
```

1.User adds a new book to his shelf with a rating. Update the average rating of that book.
Query:
Insert into shelf values(‘1’,’111’,’Read’,’1.00’,’2000-01-01’,’2002-01-02’);

update book
set avgrating=(select avg(rating) from shelf where isbn=’111’);

Screenshot (o/p):



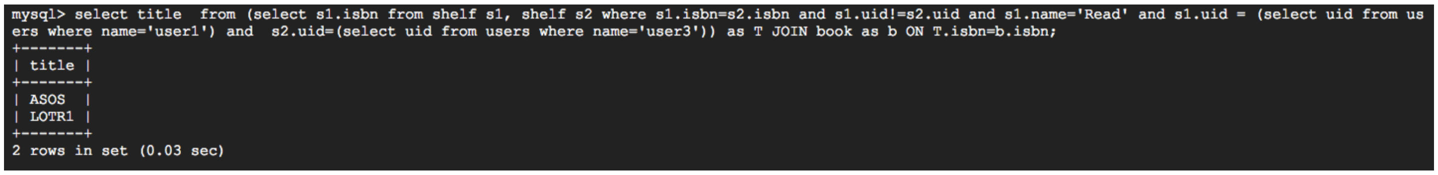
```
mysql> Insert into shelf values('1','111','Read','1.00','2000-01-01','2002-01-02');
Query OK, 1 row affected (0.04 sec)

mysql> select * from shelf;
+-----+------+-------------------+--------+------------+------------+
| uid | isbn | name              | rating | dateRead   | dateAdded  |
+-----+------+-------------------+--------+------------+------------+
|   1 | 110  | Read              |   5.00 | 2000-01-01 | 2000-02-02 |
|   1 | 111  | Read              |   1.00 | 2000-01-01 | 2002-01-02 |
|   1 | 112  | Read              |   5.00 | 2000-01-01 | 2000-01-01 |
|   1 | 121  | Read              |   5.00 | 2000-01-01 | 2000-01-01 |
|   1 | 122  | Read              |   2.70 | 2000-01-01 | 2000-01-01 |
|   2 | 111  | To-Read           |   2.70 | NULL       | 2000-01-01 |
|   2 | 121  | Read              |   3.00 | 2000-01-01 | 2000-01-01 |
|   2 | 122  | To-Read           |   2.70 | NULL       | 2000-01-01 |
|   3 | 111  | Currently Reading |   2.70 | NULL       | 2000-01-01 |
|   3 | 121  | Read              |   3.00 | 2000-01-01 | 2000-01-01 |
+-----+------+-------------------+--------+------------+------------+
10 rows in set (0.03 sec)

mysql> update book
    -> set avgrating=(select avg(rating) from shelf where isbn='111');
Query OK, 5 rows affected, 5 warnings (0.05 sec)
Rows matched: 5  Changed: 5  Warnings: 5

mysql> select * from book;
+------+------------+----------+----------+-----------+
| isbn | title      | authorId | numpages | avgrating |
+------+------------+----------+----------+-----------+
| 110  | THE HOBBIT |        2 |      150 |      2.13 |
| 111  | ASOS       |        3 |      150 |      2.13 |
| 112  | ACOK       |        3 |      150 |      2.13 |
| 121  | LOTR1      |        2 |      150 |      2.13 |
| 122  | LOTR2      |        2 |      150 |      2.13 |
+------+------------+----------+----------+-----------+
5 rows in set (0.04 sec)
```

Explanation:
The insert query inserts the value of the rating of a particular user into the shelf.
The update query updates the average rating of the book in the book table by computing the average of a particular book hardcoded in the code (variable) .
The average gets updated whenever update query is executed.

2. Find the names of the common books that were read by any two users X and Y.

Query:
```
select title
from (select s1.isbn from shelf s1, shelf s2
        where s1.isbn=s2.isbn and s1.uid!=s2.uid and s1.name=‘Read’
        and s1.uid =(select uid from users where name=‘user1’) and
        s2.uid=(select uid from users where name=‘user3’) ) as T
        JOIN book as b on T.isbn=b.isbn;
```

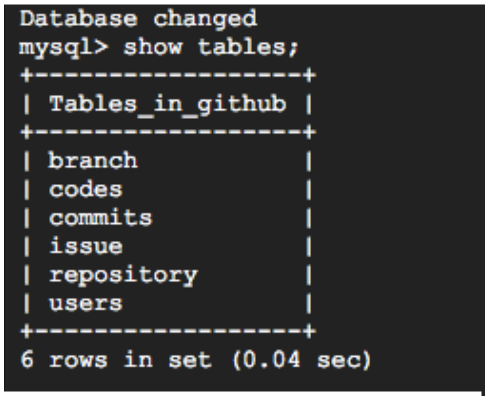
Screenshot(O/P)



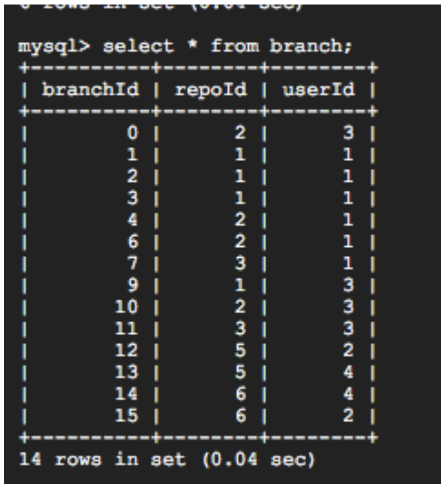
```
mysql> select title  from (select s1.isbn from shelf s1, shelf s2 where s1.isbn=s2.isbn and s1.uid!=s2.uid and s1.name='Read' and s1.uid = (select uid from us
ers where name='user1') and  s2.uid=(select uid from users where name='user3')) as T JOIN book as b ON T.isbn=b.isbn;
+-------+
| title |
+-------+
| ASOS  |
| LOTR1 |
+-------+
2 rows in set (0.03 sec)
```

Explanation:
The query selects the similar isbn of the uid's returned from the users table in comparison with ‘user1’(X) and ‘user 2’(Y) and joins the isbn with the book table to get the name of the book. The query returns the books which are read but not currently read or to read.

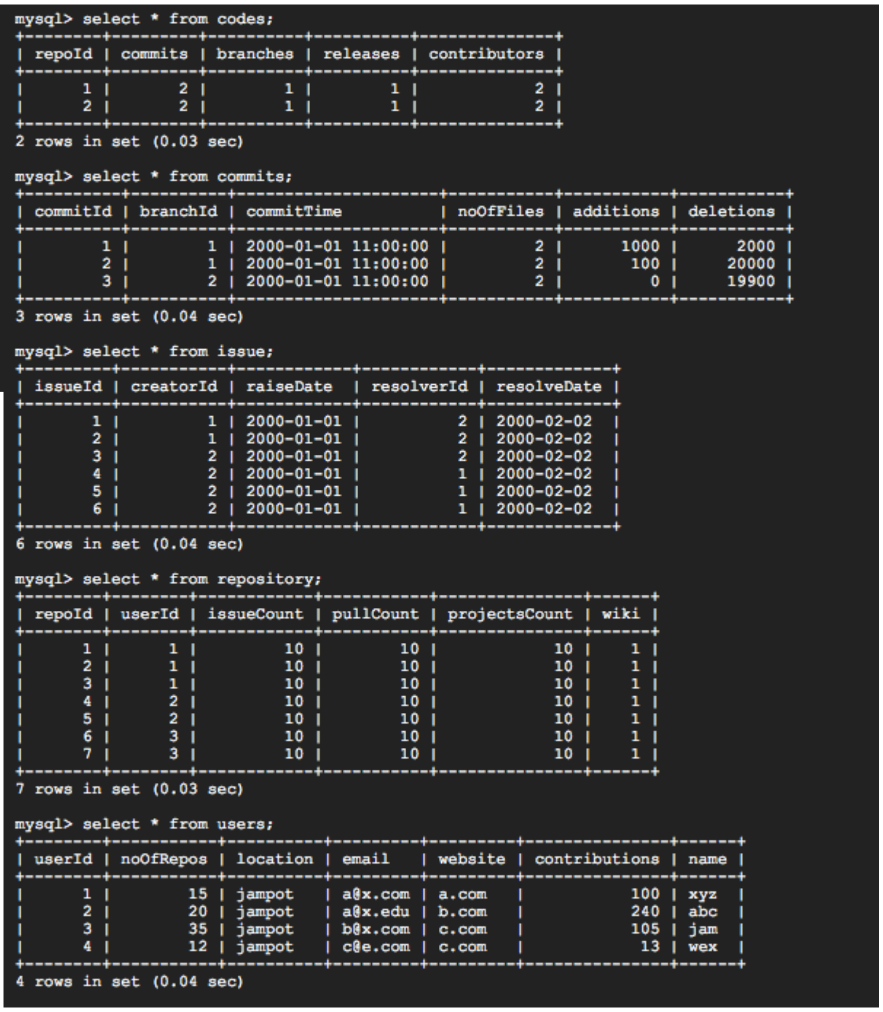
2.Set 2 - GitHub Database
Started code is used to create the database and the sample tables are implemented.
Apart from the given data in the tables the branch table is inserted using insert query.
Eg: Insert into branch values (‘1’,’1’,’1’);



```
Database changed
mysql> show tables;
+------------------+
| Tables_in_github |
+------------------+
| branch           |
| codes            |
| commits          |
| issue            |
| repository       |
| users            |
+------------------+
6 rows in set (0.04 sec)
```



```
mysql> select * from branch;
+----------+--------+--------+
| branchId | repoId | userId |
+----------+--------+--------+
|        0 |      2 |      3 |
|        1 |      1 |      1 |
|        2 |      1 |      1 |
|        3 |      1 |      1 |
|        4 |      2 |      1 |
|        6 |      2 |      1 |
|        7 |      3 |      1 |
|        9 |      1 |      3 |
|       10 |      2 |      3 |
|       11 |      3 |      3 |
|       12 |      5 |      2 |
|       13 |      5 |      4 |
|       14 |      6 |      4 |
|       15 |      6 |      2 |
+----------+--------+--------+
14 rows in set (0.04 sec)
```



```
mysql> select * from codes;
+--------+---------+----------+----------+--------------+
| repoId | commits | branches | releases | contributors |
+--------+---------+----------+----------+--------------+
|      1 |       2 |        1 |        1 |            2 |
|      2 |       2 |        1 |        1 |            2 |
+--------+---------+----------+----------+--------------+
2 rows in set (0.03 sec)

mysql> select * from commits;
+----------+----------+---------------------+-----------+-----------+-----------+
| commitId | branchId | commitTime          | noOfFiles | additions | deletions |
+----------+----------+---------------------+-----------+-----------+-----------+
|        1 |        1 | 2000-01-01 11:00:00 |         2 |      1000 |      2000 |
|        2 |        1 | 2000-01-01 11:00:00 |         2 |       100 |     20000 |
|        3 |        2 | 2000-01-01 11:00:00 |         2 |         0 |     19900 |
+----------+----------+---------------------+-----------+-----------+-----------+
3 rows in set (0.04 sec)

mysql> select * from issue;
+---------+-----------+------------+------------+-------------+
| issueId | creatorId | raiseDate  | resolverId | resolveDate |
+---------+-----------+------------+------------+-------------+
|       1 |         1 | 2000-01-01 |          2 | 2000-02-02  |
|       2 |         1 | 2000-01-01 |          2 | 2000-02-02  |
|       3 |         2 | 2000-01-01 |          2 | 2000-02-02  |
|       4 |         2 | 2000-01-01 |          1 | 2000-02-02  |
|       5 |         2 | 2000-01-01 |          1 | 2000-02-02  |
|       6 |         2 | 2000-01-01 |          1 | 2000-02-02  |
+---------+-----------+------------+------------+-------------+
6 rows in set (0.04 sec)

mysql> select * from repository;
+--------+--------+------------+-----------+---------------+------+
| repoId | userId | issueCount | pullCount | projectsCount | wiki |
+--------+--------+------------+-----------+---------------+------+
|      1 |      1 |         10 |        10 |            10 |    1 |
|      2 |      1 |         10 |        10 |            10 |    1 |
|      3 |      1 |         10 |        10 |            10 |    1 |
|      4 |      2 |         10 |        10 |            10 |    1 |
|      5 |      2 |         10 |        10 |            10 |    1 |
|      6 |      3 |         10 |        10 |            10 |    1 |
|      7 |      3 |         10 |        10 |            10 |    1 |
+--------+--------+------------+-----------+---------------+------+
7 rows in set (0.03 sec)

mysql> select * from users;
+--------+-----------+----------+---------+---------+---------------+------+
| userId | noOfRepos | location | email   | website | contributions | name |
+--------+-----------+----------+---------+---------+---------------+------+
|      1 |        15 | jampot   | a@x.com | a.com   |           100 | xyz  |
|      2 |        20 | jampot   | a@x.edu | b.com   |           240 | abc  |
|      3 |        35 | jampot   | b@x.com | c.com   |           105 | jam  |
|      4 |        12 | jampot   | c@e.com | c.com   |            13 | wex  |
+--------+-----------+----------+---------+---------+---------------+------+
4 rows in set (0.04 sec)
```

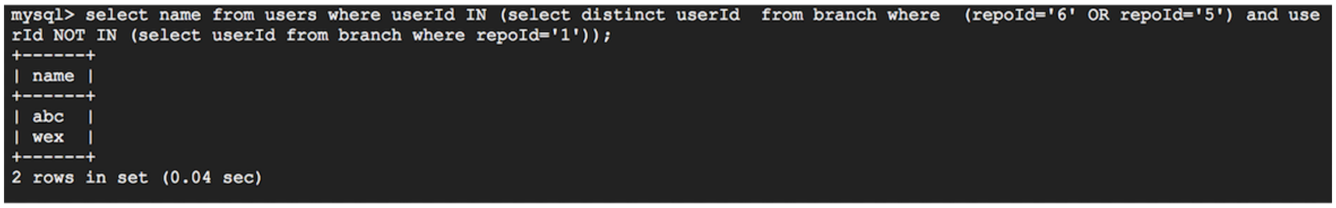
1. Find the users who made branches of either of repositories X or Y but not of a repository Z.

Assumptions:
1.Assumed that the query asked to display the name of the user. Hence a column called name is added in the users table using ‘Alter’ command.

Alter table users add name varchar2(30);
2. Assumed that a user can have any number of branches in any number of repositories i,e though a repository is created by a single user, a single repository can have different branches by different users .

Query:
```
select name from users
where userId IN ( select distinct userId from branch
                  where  (repoId=‘6’ OR repoId=‘5’)
                  and userId NOT IN (select userId from branch where repoId=‘1’));
```

screenshot(o/p):



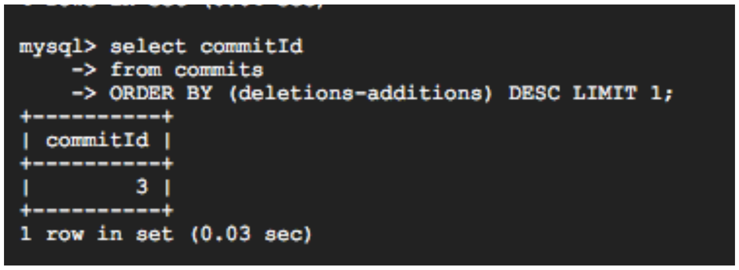
```
mysql> select name from users where userId IN (select distinct userId  from branch where  (repoId='6' OR repoId='5') and use
rId NOT IN (select userId from branch where repoId='1'));
+------+
| name |
+------+
| abc  |
| wex  |
+------+
2 rows in set (0.04 sec)
```

Explanation:
The query selects the distinct (as there can be more than one) userId’s from branches which have repository either ‘6’(X) or ‘7’(Y) and then subtracts the users who have repoId=‘1’(Z) and displays the name of those userId’s from users table.

2. 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).

Any random commitId of highest value is displayed if the highest is greater than 1.
Query:
```
select commitId
from commits ORDER BY (deletions-additions) DESC LIMIT1;
```

Screenshot(I/O)



```
mysql> select commitId
    -> from commits
    -> ORDER BY (deletions-additions) DESC LIMIT 1;
+----------+
| commitId |
+----------+
|        3 |
+----------+
1 row in set (0.03 sec)
```

Explanation: The query selects one commitId(Limits value to 1) which have the highest difference of deletions and additions as they are grouped as per the difference in descending order.

3. 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.)

Assumption: Assumed that a resolver must have risen at least a single issue.

Issue table is used in the execution of the query.

Creation of Issue table: (Starter Code)
create table issue (issueId int, creatorId int not null, raiseDate date, resolverId int, resolveDate date, primary key (issueId), constraint fk2 foreign key(creatorId) references users(userId), constraint fk3 foreign key(resolverId) references users(userId));

Insertion into the table :

insert into issue values(‘1’,’1’,’2000-01-01’, ‘2’, ‘2000-02-02’);

insert into issue values(‘2’,’1’,’2000-01-01’, ‘2’, ‘2000-02-02’);

insert into issue values(‘3’,’2’,’2000-01-01’, ‘2’, ‘2000-02-02’);

insert into issue values(‘4’,’2’,’2000-01-01’, ‘1’, ‘2000-02-02’);

insert into issue values(‘5’,’2’,’2000-01-01’, ‘1’, ‘2000-02-02’);

insert into issue values(‘6’,’2’,’2000-01-01’, ‘1’, ‘2000-02-02’);

Query:

select from creatorId from

(select creatorId, count(creatorId) as raised from issue

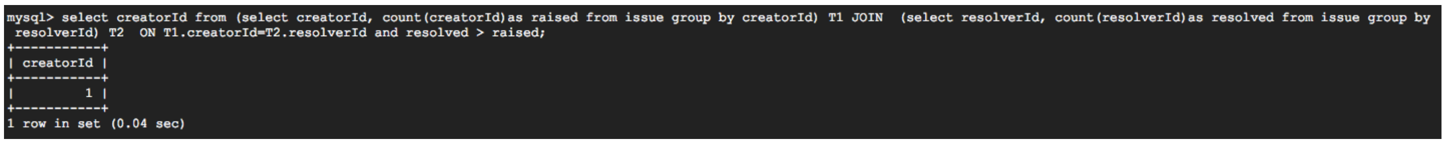
group by creatorId) T1 JOIN

(select resolverId, count(resolverId) as resolved from issue

group by resolverId) T2

ON T1.creatorId=T2.resolverId and resolved > raised;

Screenshot (o/p):



```
mysql> select creatorId from (select creatorId, count(creatorId)as raised from issue group by creatorId) T1 JOIN  (select resolverId, count(resolverId)as resolved from issue group by
 resolverId) T2  ON T1.creatorId=T2.resolverId and resolved > raised;
+-----------+
| creatorId |
+-----------+
|         1 |
+-----------+
1 row in set (0.04 sec)
```

Explanation:

The query does not link to the users table since the creatorId already references to the userId.

The query counts the creatorId and groups by the creator Id as well as counts the resolverId and groups the resolverId, joins both the outputs with condition that creator Id is equal to resolver Id. Hence, the output from the above condition displays the userId’s with counts of creatorId’s and resolverId’s. Using alias names raised and resolved, the query then checks the condition that resolved is greater than the raised(issue) and displays corresponding creatorId.