Database Systems:

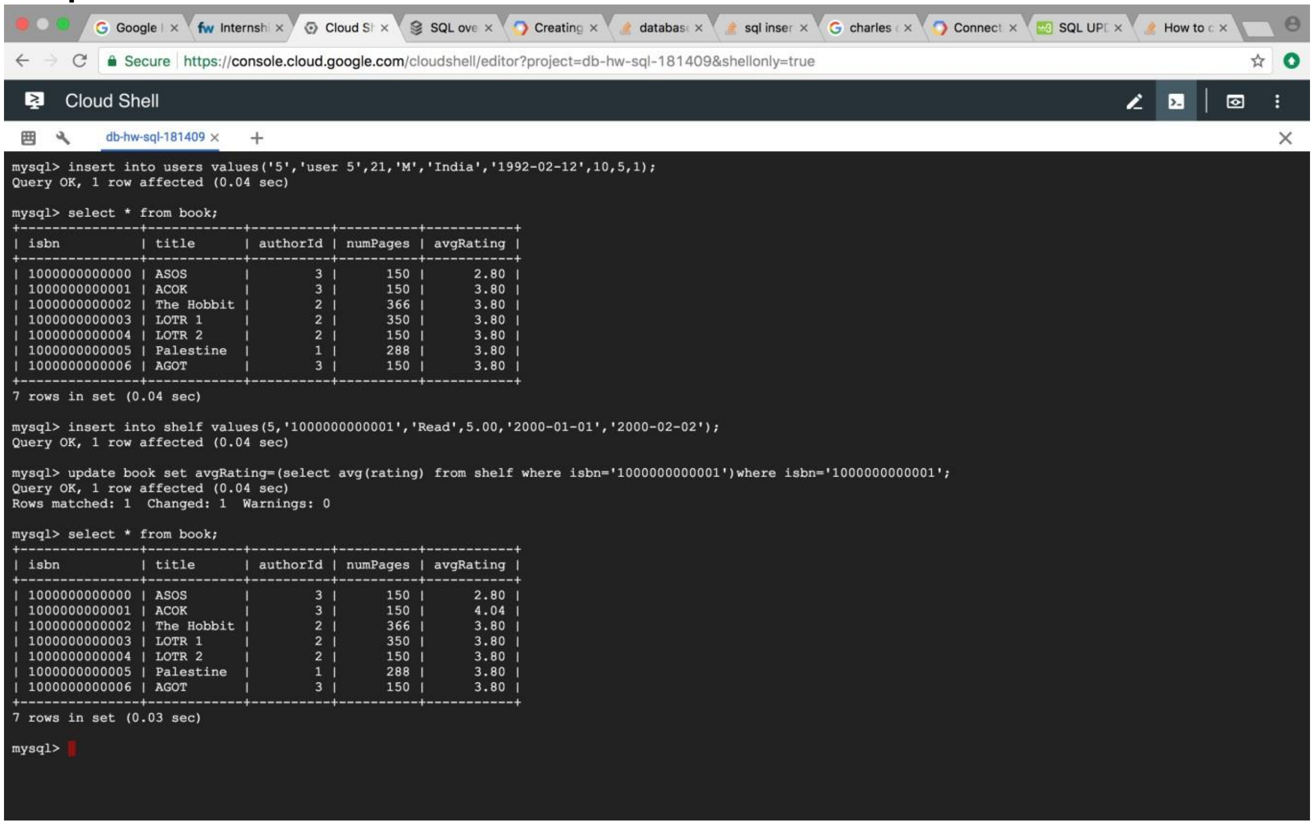
Database Used: **MySQL**

## GoodReads database:

1. User adds a new book to his shelf with a rating.Update the average rating of that book.

**SQL queries:** 1) insert into shelf values(5,‘1000000000001’,‘Read’,5.00,‘2000-01-01’,‘2000-02-02’);
2)update book set avgRating = (select avg(rating) from shelf where isbn = ‘1000000000001’) where isbn = ‘1000000000001’;

## Output:



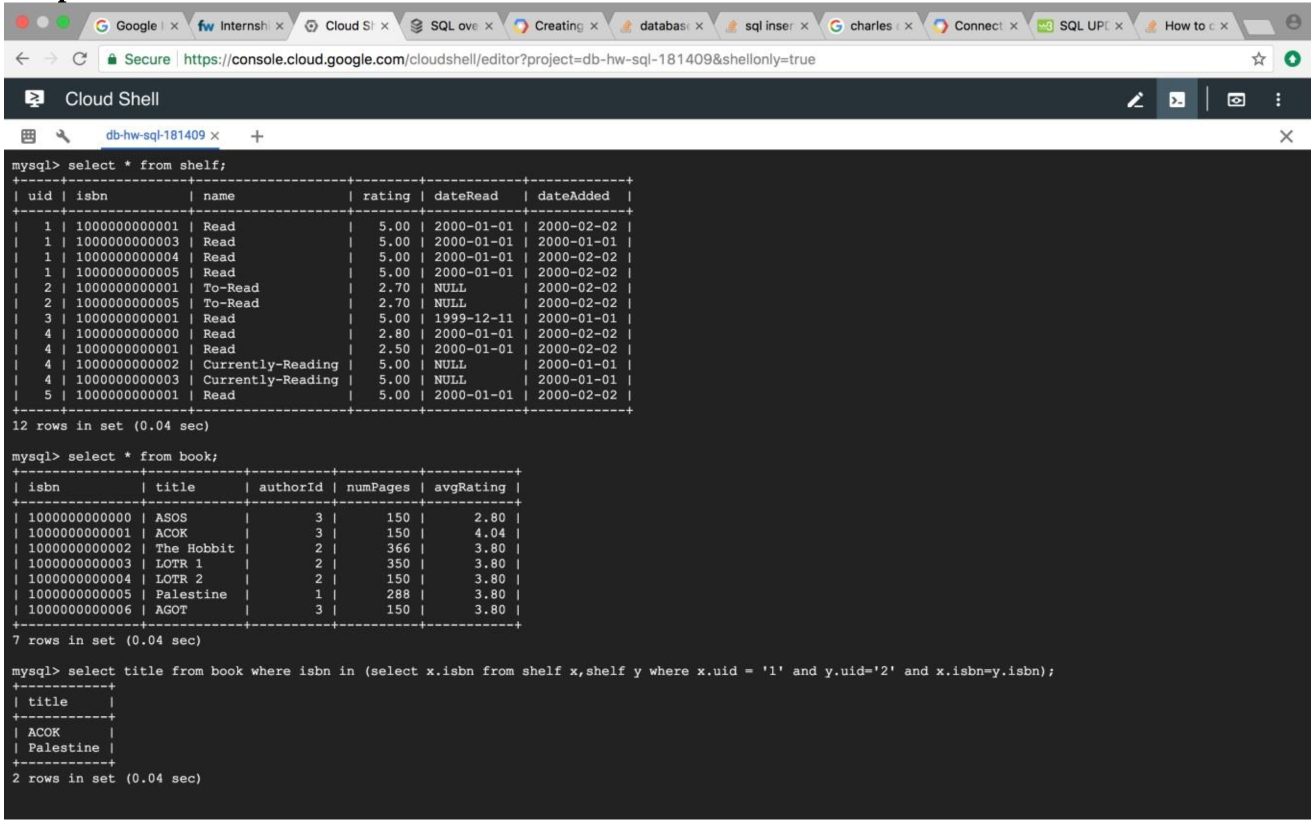
Secure https://console.cloud.google.com/cloudshell/editor?project=db-hw-sql-181409&shellonly=true
Cloud Shell
db-hw-sql-181409

```
mysql> insert into users values('5','user 5',21,'M','India','1992-02-12',10,5,1);
Query OK, 1 row affected (0.04 sec)

mysql> select * from book;
+---------------+------------+----------+----------+-----------+
| isbn          | title      | authorId | numPages | avgRating |
+---------------+------------+----------+----------+-----------+
| 1000000000000 | ASOS       |        3 |      150 |      2.80 |
| 1000000000001 | ACOK       |        3 |      150 |      3.80 |
| 1000000000002 | The Hobbit |        2 |      366 |      3.80 |
| 1000000000003 | LOTR 1     |        2 |      350 |      3.80 |
| 1000000000004 | LOTR 2     |        2 |      150 |      3.80 |
| 1000000000005 | Palestine  |        1 |      288 |      3.80 |
| 1000000000006 | AGOT       |        3 |      150 |      3.80 |
+---------------+------------+----------+----------+-----------+
7 rows in set (0.04 sec)

mysql> insert into shelf values(5,'1000000000001','Read',5.00,'2000-01-01','2000-02-02');
Query OK, 1 row affected (0.04 sec)

mysql> update book set avgRating=(select avg(rating) from shelf where isbn='1000000000001')where isbn='1000000000001';
Query OK, 1 row affected (0.04 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from book;
+---------------+------------+----------+----------+-----------+
| isbn          | title      | authorId | numPages | avgRating |
+---------------+------------+----------+----------+-----------+
| 1000000000000 | ASOS       |        3 |      150 |      2.80 |
| 1000000000001 | ACOK       |        3 |      150 |      4.04 |
| 1000000000002 | The Hobbit |        2 |      366 |      3.80 |
| 1000000000003 | LOTR 1     |        2 |      350 |      3.80 |
| 1000000000004 | LOTR 2     |        2 |      150 |      3.80 |
| 1000000000005 | Palestine  |        1 |      288 |      3.80 |
| 1000000000006 | AGOT       |        3 |      150 |      3.80 |
+---------------+------------+----------+----------+-----------+
7 rows in set (0.03 sec)

mysql>
```

**Explanation:** I have 2 queries to perform the operation given in question. In first query user with Id 5 inserts a book with isbn 1000000000001 into the shelf which is ACOK book. As we can see in the screenshot, before updating the average rating for this book is 3.80 in book table. My 2nd query updates the avgrating of book with isbn 1000000000001 i.e ACOK book in book table to 4.04.

2. Find the names of the common books that were read by any two users X and Y.
**SQL query:** select title from book where isbn in (select x.isbn from shelf x, shelf y where x.uid = 1 and y.uid = 2 and x.isbn = y.isbn and x.name='Read' and y.name='Read');

**Output:**



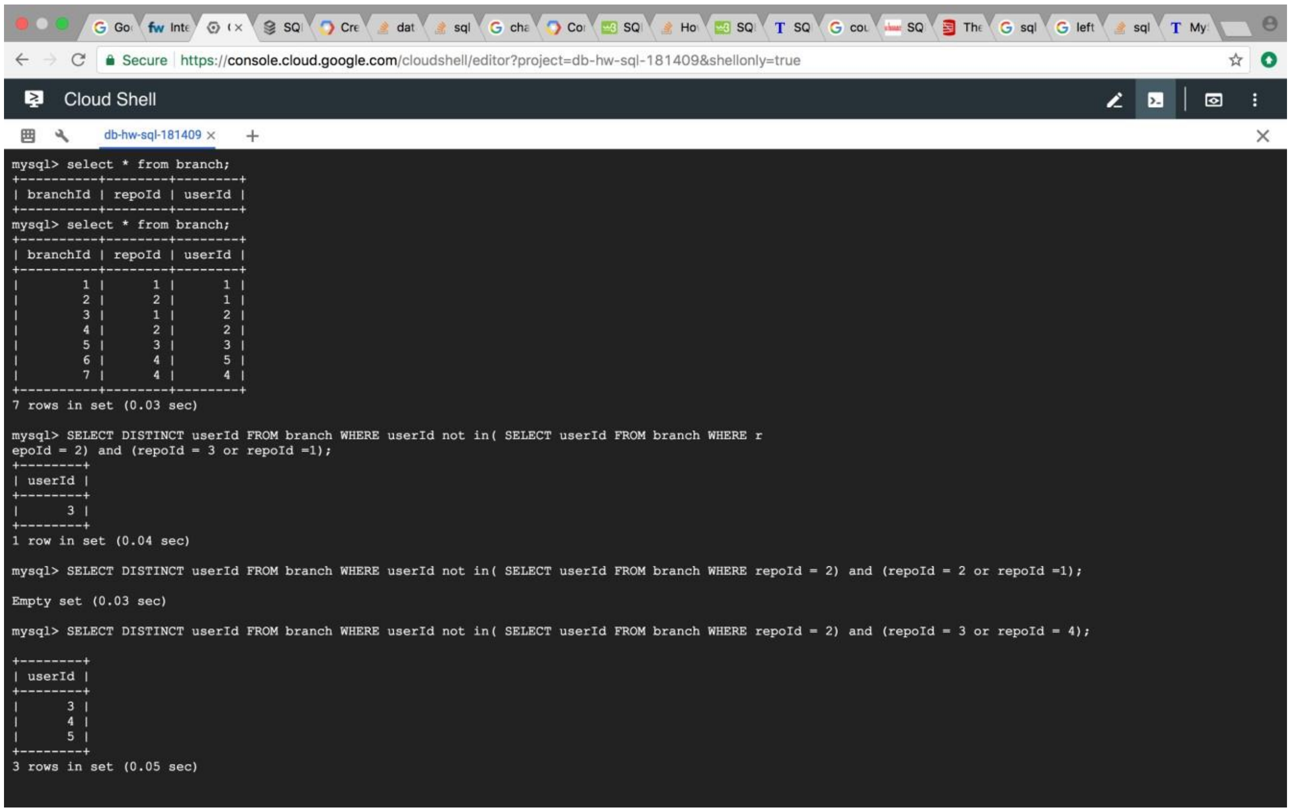
```
mysql> select * from shelf;
+-----+---------------+------------------+--------+------------+------------+
| uid | isbn          | name             | rating | dateRead   | dateAdded  |
+-----+---------------+------------------+--------+------------+------------+
|   1 | 1000000000001 | Read             |   5.00 | 2000-01-01 | 2000-02-02 |
|   1 | 1000000000003 | Read             |   5.00 | 2000-01-01 | 2000-01-01 |
|   1 | 1000000000004 | Read             |   5.00 | 2000-01-01 | 2000-02-02 |
|   1 | 1000000000005 | Read             |   5.00 | 2000-01-01 | 2000-02-02 |
|   2 | 1000000000001 | To-Read          |   2.70 | NULL       | 2000-02-02 |
|   2 | 1000000000005 | To-Read          |   2.70 | NULL       | 2000-02-02 |
|   3 | 1000000000001 | Read             |   5.00 | 1999-12-11 | 2000-01-01 |
|   4 | 1000000000000 | Read             |   2.80 | 2000-01-01 | 2000-02-02 |
|   4 | 1000000000001 | Read             |   2.50 | 2000-01-01 | 2000-02-02 |
|   4 | 1000000000002 | Currently-Reading |   5.00 | NULL       | 2000-01-01 |
|   4 | 1000000000003 | Currently-Reading |   5.00 | NULL       | 2000-01-01 |
|   5 | 1000000000001 | Read             |   5.00 | 2000-01-01 | 2000-02-02 |
+-----+---------------+------------------+--------+------------+------------+
12 rows in set (0.04 sec)

mysql> select * from book;
+---------------+------------+----------+----------+-----------+
| isbn          | title      | authorId | numPages | avgRating |
+---------------+------------+----------+----------+-----------+
| 1000000000000 | ASOS       |        3 |      150 |      2.80 |
| 1000000000001 | ACOK       |        3 |      150 |      4.04 |
| 1000000000002 | The Hobbit |        2 |      366 |      3.80 |
| 1000000000003 | LOTR 1     |        2 |      350 |      3.80 |
| 1000000000004 | LOTR 2     |        2 |      150 |      3.80 |
| 1000000000005 | Palestine  |        1 |      288 |      3.80 |
| 1000000000006 | AGOT       |        3 |      150 |      3.80 |
+---------------+------------+----------+----------+-----------+
7 rows in set (0.04 sec)

mysql> select title from book where isbn in (select x.isbn from shelf x,shelf y where x.uid = '1' and y.uid='2' and x.isbn=y.isbn);
+-----------+
| title     |
+-----------+
| ACOK      |
| Palestine |
+-----------+
2 rows in set (0.04 sec)
```

**Explanation:** Here I am displaying titles of common books from book table which have been read by two users with uids/userIds 1 and 2. In the above example the output is 2 books ‘ACOK’ with isbn ‘1000000000001’ and ‘Palestine’ with isbn ‘1000000000005’. The output can be verified by looking into shelf table.

## GitHub database:

1. Find the users who made branches of either of repositories X or Y but not of a repository Z.
**SQL query:** select distinct userId from branch where userId not in (select userId from branch where repoId = 2) and (repoId = 3 or repoId = 1);

**Output:**



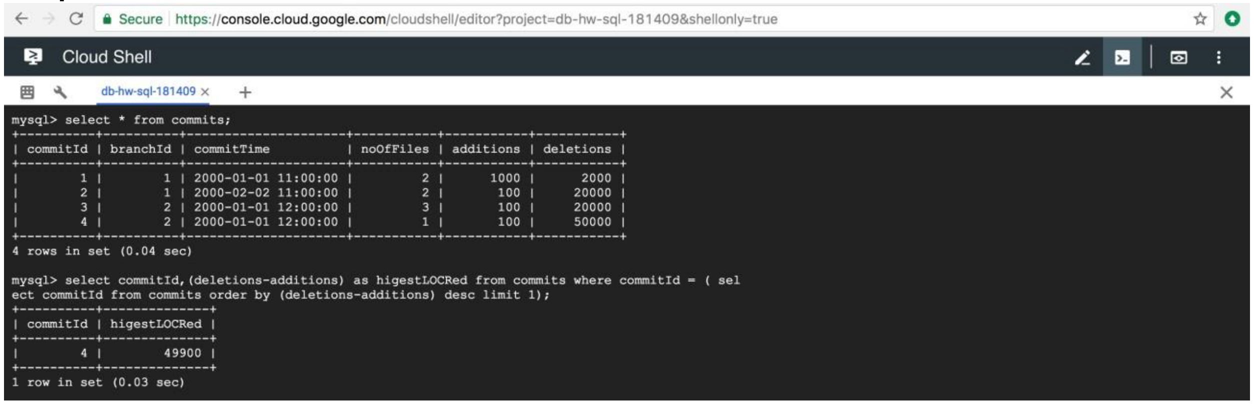
```
mysql> select * from branch;
+----------+--------+--------+
| branchId | repoId | userId |
+----------+--------+--------+
mysql> select * from branch;
+----------+--------+--------+
| branchId | repoId | userId |
+----------+--------+--------+
|        1 |      1 |      1 |
|        2 |      2 |      1 |
|        3 |      1 |      2 |
|        4 |      2 |      2 |
|        5 |      3 |      3 |
|        6 |      4 |      5 |
|        7 |      4 |      4 |
+----------+--------+--------+
7 rows in set (0.03 sec)

mysql> SELECT DISTINCT userId FROM branch WHERE userId not in( SELECT userId FROM branch WHERE r
epoId = 2) and (repoId = 3 or repoId =1);
+--------+
| userId |
+--------+
|      3 |
+--------+
1 row in set (0.04 sec)

mysql> SELECT DISTINCT userId FROM branch WHERE userId not in( SELECT userId FROM branch WHERE repoId = 2) and (repoId = 2 or repoId =1);

Empty set (0.03 sec)

mysql> SELECT DISTINCT userId FROM branch WHERE userId not in( SELECT userId FROM branch WHERE repoId = 2) and (repoId = 3 or repoId = 4);

+--------+
| userId |
+--------+
|      3 |
|      4 |
|      5 |
+--------+
3 rows in set (0.05 sec)
```

**Explanation:** Here my first query in output displays all the users which have branches in repository X=1 or Y=3 but not in repository Z=2. Similarly I have used different values for X,Y and Z to show different outputs.

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).

**SQL Query:** select commitId from commits where commitId = ( select commitId from commits order by (deletions-additions) desc limit 1);

**Output:**



```
mysql> select * from commits;
+----------+----------+---------------------+-----------+-----------+-----------+
| commitId | branchId | commitTime          | noOfFiles | additions | deletions |
+----------+----------+---------------------+-----------+-----------+-----------+
|        1 |        1 | 2000-01-01 11:00:00 |         2 |      1000 |      2000 |
|        2 |        1 | 2000-02-02 11:00:00 |         2 |       100 |     20000 |
|        3 |        2 | 2000-01-01 12:00:00 |         3 |       100 |     20000 |
|        4 |        2 | 2000-01-01 12:00:00 |         1 |       100 |     50000 |
+----------+----------+---------------------+-----------+-----------+-----------+
4 rows in set (0.04 sec)

mysql> select commitId,(deletions-additions) as higestLOCRed from commits where commitId = ( sel
ect commitId from commits order by (deletions-additions) desc limit 1);
+----------+--------------+
| commitId | higestLOCRed |
+----------+--------------+
|        4 |        49900 |
+----------+--------------+
1 row in set (0.03 sec)
```

**Explanation:** Here, I am calculating highest lines of code reduced for all the commits in commits table using sub query, arranging them in descending order and displaying only the top commit by limiting the output of sub query to return only 1 row.

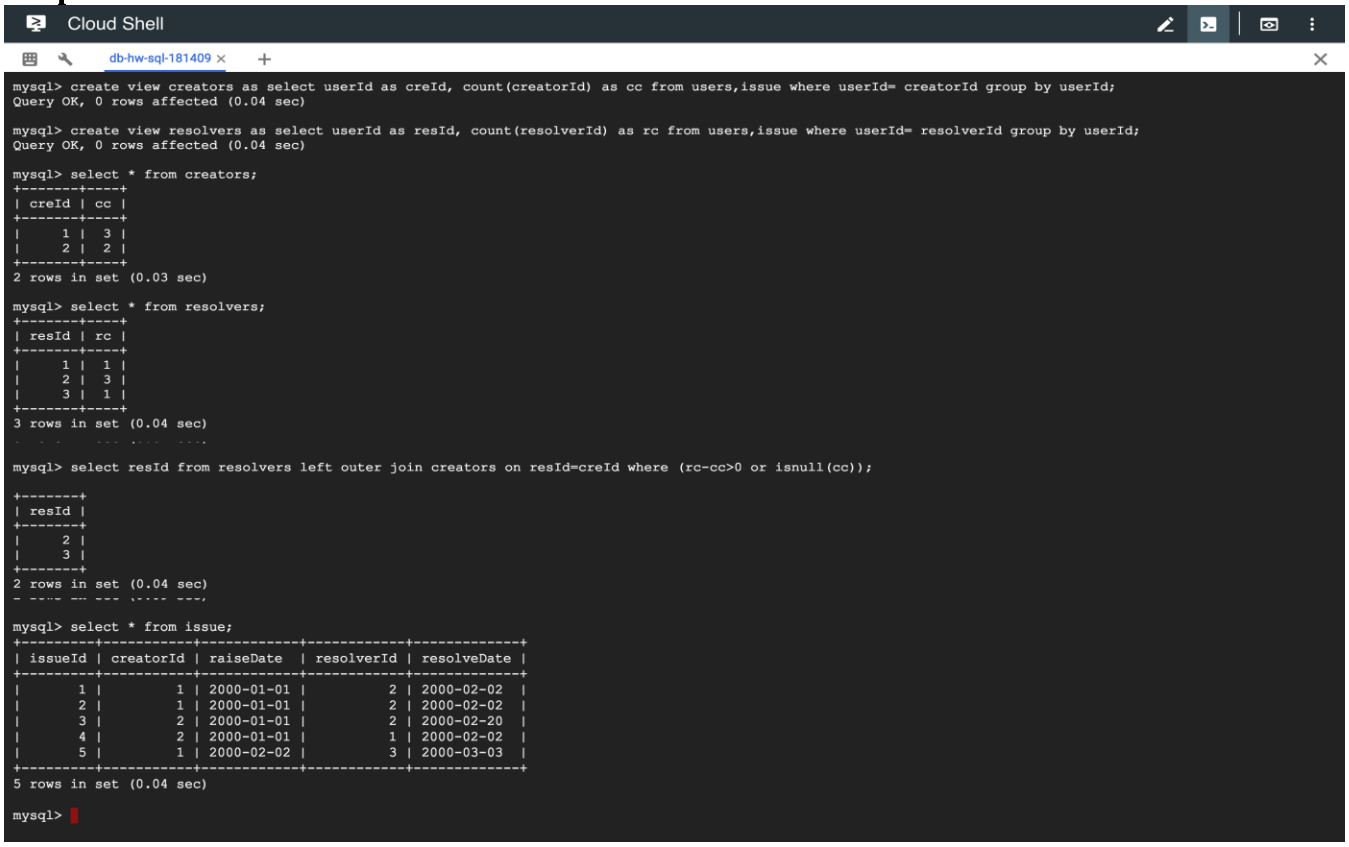
3. (BONUS question) List the users who solved more issues than they raised. (i.e. number of issues in which they were the resolver is greater than the number of issues where they were the creator.)

**SQL Query:** Creating views for creator issue count and resolver issue count.

1) create view creators as select userId as creId, count(creatorId) as cc from users,issue where userId= creatorId group by userId;
2) create view resolvers as select userId as resId, count(resolverId) as rc from users,issue where userId= resolverId group by userId;

**Main query:** select resId from resolvers left outer join creators on resId=creId where (rc-cc>0 or isnull(rc));

**Output:**



```
mysql> create view creators as select userId as creId, count(creatorId) as cc from users,issue where userId= creatorId group by userId;
Query OK, 0 rows affected (0.04 sec)

mysql> create view resolvers as select userId as resId, count(resolverId) as rc from users,issue where userId= resolverId group by userId;
Query OK, 0 rows affected (0.04 sec)

mysql> select * from creators;
+-------+----+
| creId | cc |
+-------+----+
|     1 |  3 |
|     2 |  2 |
+-------+----+
2 rows in set (0.03 sec)

mysql> select * from resolvers;
+-------+----+
| resId | rc |
+-------+----+
|     1 |  1 |
|     2 |  3 |
|     3 |  1 |
+-------+----+
3 rows in set (0.04 sec)

mysql> select resId from resolvers left outer join creators on resId=creId where (rc-cc>0 or isnull(cc));

+-------+
| resId |
+-------+
|     2 |
|     3 |
+-------+
2 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 | 2000-02-20  |
|       4 |         2 | 2000-01-01 |          1 | 2000-02-02  |
|       5 |         1 | 2000-02-02 |          3 | 2000-03-03  |
+---------+-----------+------------+------------+-------------+
5 rows in set (0.04 sec)

mysql>
```

**Explanation:**

In this query, I am creating 2 views called creators which contains issues created count(cc) for each user, and resolvers which contains issues resolved count(rc) for each user from users and issue table. Finally, I am performing left outer join on resolvers and creators with the condition issues resolved count > issues created count and where cc is null. The second condition is necessary because we also want to display the users who have never created an issue but resolved issues.