HOMEWORK – 2

NAME: *Ruchin Patel*
STUDENT ID: *9520665364*
LANGUAGE USED: *MySQL*

GoodReads database:

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

SOLUTION: The query that I am going to write for this section is the one which triggers After Insert. The example that I will be providing here would be for the book title ASOS with ISBN 9730618260320.

BOOK TABLE

```
mysql> SELECT * FROM book;
+---------------+------------+----------+----------+-----------+
| isbn          | title      | authorId | numpages | avgrating |
+---------------+------------+----------+----------+-----------+
| 9730618260320 | ASOS       |        3 |      150 |      2.50 |
| 9770618260320 | ACOK       |        3 |      150 |      4.24 |
| 9780618260300 | The Hobbit |        2 |      366 |      4.20 |
| 9780618260301 | LOTR 1     |        2 |      350 |      4.20 |
| 9780618260320 | LOTR 2     |        2 |      150 |      3.75 |
| 9781560974321 | Palestine  |        1 |      288 |      3.45 |
| 9880618260320 | AGOT       |        3 |      150 |      2.50 |
+---------------+------------+----------+----------+-----------+
7 rows in set (0.04 sec)
```

QUERY

```
mysql>  CREATE TRIGGER update_rating AFTER INSERT ON shelf
   ->    FOR EACH ROW
   ->      UPDATE book
   ->        SET avgrating = (SELECT SUM(rating) FROM shelf WHERE isbn = NEW.isbn)/(SELECT COUNT(isbn) FROM shelf WHERE isbn = NEW.isbn)
   ->      WHERE (book.isbn = NEW.isbn);
Query OK, 0 rows affected (0.04 sec)
```

EXPLANATION: This query creates a trigger named update_rating after a book is added to a shelf by any user. We have assumed that the shelf_name can be anything i.e it does not necessarily have to be in 'Read' section.
Thus whenever a new Insert is made in the shelf table, after the insert is made our query will update the 'avgrating' attribute in the book table by counting the sum of all the rating with 'isbn' number of the newly added Book in shelf.

For example consider book with title 'ASOS' with ISBN 9730618260320 and its value in the shelf table is shown as follows:

```
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 | 9730618260320 | Read             |   2.50 | 2000-01-01 | 2000-02-02 |
|   4 | 9780618260300 | Currently-Reading |   5.00 | NULL       | 2000-01-01 |
|   4 | 9780618260301 | Currently-Reading |   5.00 | NULL       | 2000-01-01 |
+-----+---------------+------------------+--------+------------+------------+
10 rows in set (0.03 sec)
```

Thus if users with user id 2 and 3 enter the book ASOS with their respective rating the avgrating in book should change and it is as shown below:

OUTPUT:

- When 3 Inserts ASOS with rating 3.5 in shelf avgrating should be (2.5+3.5)/2 = 3.

```
mysql> INSERT into shelf VALUES(3,"9730618260320","Read",3.5,'2000-01-01','2000-02-02');
Query OK, 1 row affected (0.04 sec)

mysql> SELECT * FROM book;
+---------------+------------+----------+----------+-----------+
| isbn          | title      | authorId | numpages | avgrating |
+---------------+------------+----------+----------+-----------+
| 9730618260320 | ASOS       |        3 |      150 |      3.00 |
| 9770618260320 | ACOK       |        3 |      150 |      4.24 |
| 9780618260300 | The Hobbit |        2 |      366 |      4.20 |
| 9780618260301 | LOTR 1     |        2 |      350 |      4.20 |
| 9780618260320 | LOTR 2     |        2 |      150 |      3.75 |
| 9781560974321 | Palestine  |        1 |      288 |      3.45 |
| 9880618260320 | AGOT       |        3 |      150 |      2.50 |
+---------------+------------+----------+----------+-----------+
7 rows in set (0.04 sec)
```

- When 2 Inserts ASOS with rating 5 in shelf avgrating should be (2.5+3.5+5)/3 = 3.67

```
mysql> INSERT into shelf VALUES(2,"9730618260320","Read",5.0,'2000-01-01','2000-02-02');
Query OK, 1 row affected (0.04 sec)

mysql> SELECT * FROM book;
+---------------+------------+----------+----------+-----------+
| isbn          | title      | authorId | numpages | avgrating |
+---------------+------------+----------+----------+-----------+
| 9730618260320 | ASOS       |        3 |      150 |      3.67 |
| 9770618260320 | ACOK       |        3 |      150 |      4.24 |
| 9780618260300 | The Hobbit |        2 |      366 |      4.20 |
| 9780618260301 | LOTR 1     |        2 |      350 |      4.20 |
| 9780618260320 | LOTR 2     |        2 |      150 |      3.75 |
| 9781560974321 | Palestine  |        1 |      288 |      3.45 |
| 9880618260320 | AGOT       |        3 |      150 |      2.50 |
+---------------+------------+----------+----------+-----------+
7 rows in set (0.04 sec)
```

QUESTION 2: *Find the names of the common books that were read bytwo users X and Y.*

SOLUTION: Here I will first declare 2 variables with the help of SET command ,then substitute some hardcoded values in those variables and then perform the query on those variables so that the code remains generic.

SHELF 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 | 9730618260320 | 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 | 9730618260320 | Read             |   3.50 | 2000-01-01 | 2000-02-02 |
|   3 | 9770618260320 | Read             |   5.00 | 1999-12-11 | 2000-01-01 |
|   4 | 9730618260320 | Read             |   2.50 | 2000-01-01 | 2000-02-02 |
|   4 | 9780618260300 | Currently-Reading |   5.00 | NULL       | 2000-01-01 |
|   4 | 9780618260301 | Currently-Reading |   5.00 | NULL       | 2000-01-01 |
+-----+---------------+------------------+--------+------------+------------+
12 rows in set (0.03 sec)
```

Common Books

BOOK TABLE

```
mysql> SELECT * FROM book;
+---------------+------------+----------+----------+-----------+
| isbn          | title      | authorId | numpages | avgrating |
+---------------+------------+----------+----------+-----------+
| 9730618260320 | ASOS       |        3 |      150 |      3.67 |
| 9770618260320 | ACOK       |        3 |      150 |      4.24 |
| 9780618260300 | The Hobbit |        2 |      366 |      4.20 |
| 9780618260301 | LOTR 1     |        2 |      350 |      4.20 |
| 9780618260320 | LOTR 2     |        2 |      150 |      3.75 |
| 9781560974321 | Palestine  |        1 |      288 |      3.45 |
| 9880618260320 | AGOT       |        3 |      150 |      2.50 |
+---------------+------------+----------+----------+-----------+
7 rows in set (0.04 sec)
```

I am showing the shelf table because in the following query I will be using the user ids 1 and 3 and finding books that read by both of them.

QUERY TABLE

```
mysql> SET @X = 1, @Y = 3;
Query OK, 0 rows affected (0.04 sec)

mysql> SELECT title
    -> FROM book
    -> WHERE book.isbn IN (SELECT shelf.isbn
    -> FROM shelf
    -> WHERE shelf.uid = @X
    -> AND shelf.NAME = "READ"
    -> AND shelf.isbn IN
    ->    (SELECT shelf.isbn
    ->     FROM shelf
    ->     WHERE shelf.uid = @Y));
+-------+
| title |
+-------+
| ACOK  |
+-------+
1 row in set (0.05 sec)
```

EXPLANATION:
Here the query will select title from book table whose isbn are common isbn codes of users with user id 1 and 3.
In other words if you look at the shelf table, our query selects the intersection between two sets where the two sets are isbn codes corresponding to user ids 1 and 3 in the shelf table.

OUTPUT:
As we can see from the shelf table only ACOK is common to both 1 and 3.

GOODS READS OVER

GitHub Database:

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

SOLUTION: Here I will first declare THREE variables with the help of SET command ,then substitute some hardcoded values in those variables and then perform the query on those variables so that the code remains generic.

REPOSITORY 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 |         11 |        12 |            10 |    1 |
|      4 |      3 |         14 |        16 |            10 |    1 |
|      5 |      2 |         14 |        12 |            10 |    1 |
|      6 |      3 |         15 |        17 |            10 |    1 |
+--------+--------+------------+-----------+---------------+------+
6 rows in set (0.03 sec)
```

The repository id that I am going to choose as X Y and Z in this question are as follows:
X=4, Y=5, Z=6.

Consider the branch table with following values:

BRANCH TABLE

```
mysql> SELECT * FROM branch;
+----------+--------+--------+
| branchId | repoId | userId |
+----------+--------+--------+
|        1 |      2 |      1 |
|        2 |      1 |      1 |
|        3 |      3 |      2 |
|        4 |      2 |      2 |
|        5 |      1 |      2 |
|        6 |      4 |      1 |
|        7 |      4 |      3 |
|        8 |      5 |      3 |
+----------+--------+--------+
8 rows in set (0.04 sec)
```

Which repoId we are Considering

Thus as per the question we need to find the users who made branches of repository with repo id 4 OR 5 but not with repo id 6. If we look closely at the branch table we can see that User 3 has branches made for both repository 4 and 5 but not 6 an the same is true for user with user id 1.
Thus we can say that both 3 and 1 should be selected and our query will do exactly the same.

QUERY

```
mysql> SET @X=4, @Y=5, @Z=6;
Query OK, 0 rows affected (0.03 sec)

mysql> SELECT *
    -> FROM users
    -> WHERE userId IN (SELECT DISTINCT userId
    ->                  FROM branch
    ->                  WHERE ((repoId IN (@X,@Y)) AND (repoId NOT IN (@Z))));
+--------+-----------+----------+---------+---------+---------------+
| userId | noOfRepos | location | email   | website | contributions |
+--------+-----------+----------+---------+---------+---------------+
|      1 |        15 | jampot   | a@x.com | a.com   |           100 |
|      3 |        25 | india    | b@x.edu | c.com   |           340 |
+--------+-----------+----------+---------+---------+---------------+
2 rows in set (0.04 sec)
```

EXPLANATION : This query will select everything from the user table corresponding to the user ids which create branches for repo id 4 or 5 but not for repo id 6.

This query selects only those distinct user ids from the branch table depending on the condition shown in the WHERE clause which is nothing “all user ids in branch which have repo Id 4 or 5 but not 6).
After selecting those distinct user Ids we select all the attributed from user table where the user id belong to the list specified by the SELECT DISTINCT userId sub query.

OUTPUT: If we consider the branch table we see that user 1 and 3 are the only users ho branch (4 OR 5) but Not 6

QUESTION 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).*

SOLUTION: The solution is pretty straight forward in which we just have to find the MAX value of the number of deletions – number of additions.

COMMIT TABLE

```
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 |
+----------+----------+---------------------+-----------+-----------+-----------+
2 rows in set (0.04 sec)
```

QUERY

```
mysql> SELECT *,(deletions-additions) AS high_line_code_red
    -> FROM commits
    -> WHERE (deletions-additions)=(SELECT MAX(deletions-additions) AS D
    ->                              FROM commits);
+----------+----------+---------------------+-----------+-----------+-----------+-------------------+
| commitId | branchId | commitTime          | noOfFiles | additions | deletions | high_line_code_red |
+----------+----------+---------------------+-----------+-----------+-----------+-------------------+
|        2 |        1 | 2000-01-01 11:00:00 |         2 |       100 |     20000 |             19900 |
+----------+----------+---------------------+-----------+-----------+-----------+-------------------+
1 row in set (0.03 sec)
```

EXPLANATION: If we look at the query we have added a new attribute to the table and labelled ‘high_line_code_red’. This is nothing but the attribute representing the highest lines of code reduced.

Thus Our query selects every attribute from the commits table and also adds a new attribute high_line_code_red in such a way where the (deletions-additions) value of each commitId is checked and only the Max amount from them is considered.

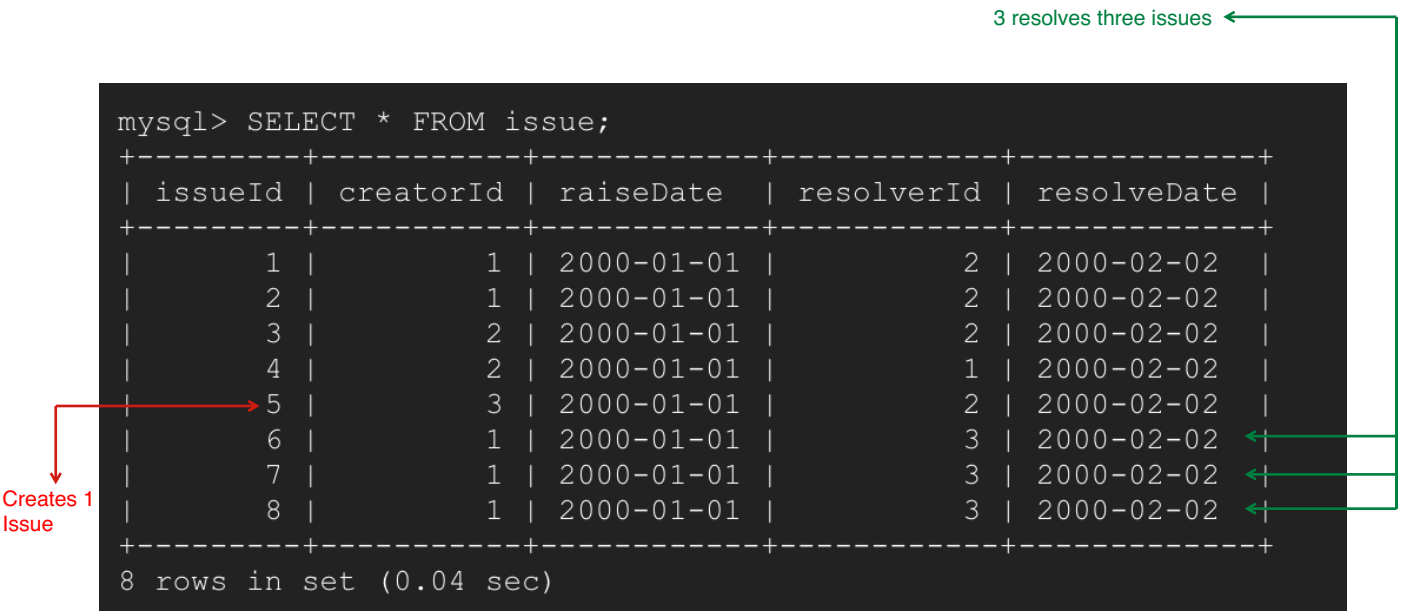
Finally our query shows the attributes of the commit id which has the highest lines of code reduced.

OUTPUT: If we look at the commit table the output that we generated indeed has the highest lines of code reduced.

QUESTION 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.)*

SOLUTION: SO here we are basically supposed to check if a particular user solved more issues than he created. This can easily be achieved from the issue table. Consider the issue table:

ISSUE TABLE




```
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 |         3 | 2000-01-01 |          2 | 2000-02-02  |
|       6 |         1 | 2000-01-01 |          3 | 2000-02-02  |
|       7 |         1 | 2000-01-01 |          3 | 2000-02-02  |
|       8 |         1 | 2000-01-01 |          3 | 2000-02-02  |
+---------+-----------+------------+------------+-------------+
8 rows in set (0.04 sec)
```

The main characteristic of the issue table is that the attributes ‘creatorid’ and ‘resolverId’ are both foreign keys which reference to userid in user table. This gives us a very good insight about what the creatorId and resolverId are.

They are essentially the user Id which are just given a different name.

Thus if for a particular userId we have the resolverId count corresponding that is greater than the corresponding creatorId count we can say that that specific user has solved more issues than he has resolved.

QUERY

```
mysql> SELECT *
    -> FROM users
    -> WHERE userId IN (SELECT creatorId
    ->   FROM
    ->      (SELECT creatorId,issue_created,issue_resolved,(issue_resolved - issue_created) AS winner
    ->       FROM
    ->           (SELECT creatorId,COUNT(creatorID) AS issue_created
    ->           FROM issue
    ->           GROUP BY creatorID) AS T1
    ->           JOIN
    ->           (SELECT resolverId,COUNT(resolverID) AS issue_resolved
    ->           FROM issue
    ->           GROUP BY resolverID) AS T2
    ->           ON (T1.creatorId = T2.resolverID)) AS T3
    ->    WHERE (T3.winner > 0));
+--------+-----------+----------+---------+---------+---------------+
| userId | noOfRepos | location | email   | website | contributions |
+--------+-----------+----------+---------+---------+---------------+
|      2 |        20 | jampot   | a@x.edu | b.com   |           240 |
|      3 |        25 | india    | b@x.edu | c.com   |           340 |
+--------+-----------+----------+---------+---------+---------------+
2 rows in set (0.03 sec)
```

EXPLANATION:

- Here the first query tells us that we are going to select all attributes from user table where the UserId in sub_query2.
- Sub_query2: Select creatorId from sub_query3.
- Sub_query3: Selects creatorId ,issue_created,issue_resolved ,(issue_resolved-issue_created) AS winner from sub_query4.
- Sub_query4: This query basically joins 2 tables T1 and T2 and aliases it T3.
    - T1: It is a table where the attributes are creatorId and counts of the particular creatorId which is grouped by the creatorId. This table essentially tells us which user created how many issues
    - T2: This table is almost same as T1 only that it contains which user resolved how many issues.
    - T3: This is a join between T1 and T2 in such a way that it lists all the issues resolved by the creator. This table also contains a new attribute ‘winner’ which tells us the if the issues

resolved by that particular creator are more or less than the ones created.

Finally CreatorId is selected from userId in sub_query3 with a Where condition which says that if a particular creator has a value of winner greater than 0, select that creator as he has solved more issues than he has created. And finally the userId values are selected from these creatorIds.

OUTPUT: As we can see from the output user 2 and 3 are selected.
If we see in the issue table user 2 create 2 issues but solved 3 and user 3 created 1 issue but solved 3. Thus our answer is correct.

GITHUB DATABASE OVER