

**USC ID: 3085084700**

**Name: Sushant Ramesh**

**Language: MySQL**

**Set 1: GoodReads Database:**

The database is created with the help of starter code. Following screenshots are the results of the queries along with their values.

**Book: ISBN, Title, AuthorID, NumPages, AverageRating.**

Input : create table book (isbn varchar(255), title varchar(20), authorId int, numpages int not null, avgrating decimal(3,2), primary key(isbn), constraint fk1 foreign key(authorId) references author(authorId));

Output: Select \* from book;

This will display the book table.

```
mysql> select * from book;
```

isbn	title	authorId	numpages	avgrating
9731	ASOS	3	150	4.60
9732	ACOK	3	150	3.47
9733	The Hobbit	2	366	3.30
9734	LOTR 1	2	350	3.35
9735	LOTR 2	2	150	3.40
9736	PALESTINE	1	288	2.50
9737	AGOT	3	150	4.20

**Users: UID, Name, Age, Sex, Location, Birthday, ReadCount, CurrentlyReadingCount, ToReadCount**

Input: create table users (uid int, name varchar(20), age int, sex char(1), location varchar(20), birthday date, readCt int, toReadCt int, currentlyReadCt int, primary key(uid));

Output: Select \* from users; This will display users 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	user 4	20	m	jampit	2000-01-01	10	3	12

**Shelf: UID, ISBN, Rating, ShelfName, DateRead, DateAdded**

Input: create table shelf(uid int, isbn varchar(255), name varchar(20), rating decimal(3,2), dateRead date, dateAdded date, primary key(uid, isbn), constraint fk2 foreign key(uid) references users(uid), constraint fk3 foreign key(isbn) references book(isbn));

Output: select \* from shelf This will display shelf table

```
mysql> select * from shelf;
```

uid	isbn	name	rating	dateRead	dateAdded
1	9731	read	5.00	2000-01-01	2000-02-02
1	9732	read	5.00	2000-01-01	2000-01-01
1	9733	read	5.00	2000-01-01	2000-02-02
1	9734	read	4.00	2000-01-01	2001-01-01
1	9735	to-read	4.00	NULL	2002-02-02
1	9736	read	5.00	2000-01-01	2000-02-02
2	9734	to-read	2.70	NULL	2000-02-02
2	9736	to-read	2.70	NULL	2000-02-02
3	9731	to-read	4.20	NULL	2000-02-02
3	9736	read	5.00	1999-12-11	2000-01-01
4	9733	currently reading	5.00	2000-01-01	2000-02-02
4	9735	currently reading	5.00	2000-01-01	2000-02-02

### **Friends: UID, FriendID**

Input: create table friends(uid int, fid int, primary key(uid, fid), constraint fk4 foreign

key(uid) references users(uid), constraint fk5 foreign key(fid) references users(uid));

Output: Select \* from friends; This will display friends table

```
mysql> select * from friends;
+-----+-----+
| uid | fid |
+-----+-----+
| 2 | 1 |
| 3 | 1 |
| 1 | 2 |
| 1 | 3 |
+-----+-----+
```

### **Author: AuthorID, Name.**

```
mysql> select * from author;
+-----+-----+
| authorId | name |
+-----+-----+
| 1 | Joe Sacco |
| 2 | Talkien |
| 3 | George Martin |
+-----+-----+
```

## Questions:

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

### Solution 1:

**Input:** insert into shelf values (3,9732,"read",3.50, '2000-01-01','2000-01-01');  
In this query user will insert a new book with a rating to his shelf. Insert query with values will work here.

**Output:** select \* from shelf;

Gives updated table with new inserted book with isbn 9732 uid 3.

```
mysql> insert into shelf values (3,9732,"read",3.50, '2000-01-01','2000-01-01');
Query OK, 1 row affected (0.04 sec)
```

```
mysql> select * from shelf;
```

uid	isbn	name	rating	dateRead	dateAdded
1	9731	read	5.00	2000-01-01	2000-02-02
1	9732	read	5.00	2000-01-01	2000-01-01
1	9733	read	5.00	2000-01-01	2000-02-02
1	9734	read	4.00	2000-01-01	2001-01-01
1	9735	to-read	4.00	NULL	2002-02-02
1	9736	read	5.00	2000-01-01	2000-02-02
2	9734	to-read	2.70	NULL	2000-02-02
2	9736	to-read	2.70	NULL	2000-02-02
3	9731	to-read	4.20	NULL	2000-02-02
3	9732	read	3.50	2000-01-01	2001-01-01
3	9736	read	5.00	1999-12-11	2000-01-01
4	9733	currently reading	5.00	2000-01-01	2000-02-02
4	9735	currently reading	5.00	2000-01-01	2000-02-02

In second part of question the avg rating of book is to be updated in book table.

**Input:** update book set avgrating = (select avg(rating) from shelf group by isbn having isbn=9732) where isbn=9732;

I applied nested queries where the first avg rating is calculated from shelf table. Then, I updated each particular rating in book table with the help of update query.

**Ouput:** select \* from book;

Updated rating column in book will be displayed. First in book column rating of 9732 was 3.47 now it's changed to 4.25.

```
mysql> select isbn, count(uid), avg(rating)
-> from shelf
-> group by isbn
-> having isbn=9732;
```

isbn	count(uid)	avg(rating)
9732	2	4.250000

```
mysql> update book set avgrating= (select avg(rating) from shelf group by isbn having isbn=9732) where isbn=
-> 9732;
```

Query OK, 1 row affected (0.04 sec)

Rows matched: 1 Changed: 1 Warnings: 0

```
mysql> select * from book;
```

isbn	title	authorId	numpages	avgrating
9731	ASOS	3	150	4.60
9732	ACOK	3	150	4.25
9733	The Hobbit	2	366	3.30
9734	LOTR 1	2	350	3.35
9735	LOTR 2	2	150	3.40
9736	PALESTINE	1	288	2.50
9737	AGOT	3	150	4.20

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

**Input:** select a.isbn from shelf a inner join shelf b on a.name="read" and b.name="read" and a.isbn=b.isbn where a.uid=1 and b.uid=3;

**Output:** It will display common books isbn of two users x=1 and y=3;

```
mysql> select a.isbn from shelf a inner join shelf b on a.name="read" and b.name="read" and a.isbn=b.isbn where a.uid=1 and b.uid=3;
```

isbn
9732
9736

**Input:** select title from book where isbn IN(select a.isbn from shelf a inner join shelf b on a.name="read" and b.name="read" and a.isbn=b.isbn where a.uid=1 and b.uid=3;)

**Output:** It will display title of common books read by two users.

Explanation: Inner join will join two tables in such a way that it will display only common elements. In the above query two users x=1 and y=3 who read the common book(isbn) will be displayed.

```
mysql> select title from book where isbn IN (select a.isbn from shelf a inner join shelf b on a.name="read" and b.name="read" and a.isbn=b.isbn where a.uid=1 and b.uid=3);
```

title
BOOK
PALESTINE

## QUESTION 2

### Set 2: Github Database

**User:** UserID, NumRepos, Location, Email, Website, ContributionCount

**Input:** create table users (userId int, noOfRepos int, location varchar (50), email varchar (50), website varchar (50), contributions int, primary key(userId));

**Output:** list of users will be displayed

```
mysql> select * from users;
```

userId	noOfRepos	location	email	website	contributions
1	15	jampot	a@x.com	a.com	100
2	10	kampota	b@x.com	b.com	200
3	19	nampota	c@x.com	c.com	300
4	21	pampota	d@x.com	d.com	400
5	25	rampota	e@x.com	e.com	500

**Repository: RepoID, UserID, IssueCount, PullRequestCount, ProjectCount, Wiki (yes/no)**

**Input:** create table repository (repoId int, userId int not null, issueCount int, pullCount int, projectsCount int, wiki boolean primary key (repoId), constraint fk1 foreign key(userId) references users(userId));

**Output:** list of all repository by users will be displayed

```
mysql> select * from repository;
```

repoId	userId	issueCount	pullCount	projectsCount	wiki
1	1	10	10	10	0
2	2	12	10	10	1
3	1	12	10	10	1
4	3	12	10	10	1
5	4	12	10	10	0
6	5	12	10	10	0
7	2	12	10	10	0
8	1	12	10	10	0

**Issues: IssueID, CreatorID, RaiseDate, ResolverID, ResolveDate.**

**Input:** 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));

**Output:** List of all issues will be displayed



```
mysql> select * from issue;
```

issueId	creatorId	raiseDate	resolverId	resolveDate
1	1	2000-01-01	2	2000-02-02
2	2	2000-01-01	1	2000-02-02
3	1	2000-01-01	3	2000-02-02
4	3	2000-01-01	5	2000-02-02
5	4	2000-01-01	5	2000-02-02
6	3	2000-01-01	4	2000-02-02
7	5	2000-01-01	3	2000-02-02

**Codes: RepoID, CommitCount, BranchCount, ReleaseCount, ContributorCount**

**Input:** create table codes (repoId int, commits int not null, branches int not null, releases int, contributors int, primary key(repoId), constraint fk4 foreign key(repoId) references repository(repoId));

**Output:** List of all codes by the users will be displayed

```
mysql> select * from codes;
```

repoId	commits	branches	releases	contributors
1	2	1	1	2
2	2	1	1	2
3	2	1	1	2
4	2	4	3	2

### Branch: UserID, BranchID, RepoID

**Input:** create table branch (branchId int, repoId int not null, userId int not null, primary key(branchId), constraint fk5 foreign key(repoId) references repository(repoId), constraint fk6 foreign key(userId) references users(userId));

**Output:** List of all branches will be displayed

```
mysql> select * from branch;
```

branchId	repoId	userId
1	1	1
2	2	1
3	2	2
4	1	2
5	3	4
6	4	5
7	1	3
8	2	5

### Commit: CommitID, BranchID, Timestamp, NumFiles, Additions, Deletions

**Input:** create table commits (commitId int, branchId int not null, commitTime datetime, noOfFiles int, additions int, deletions int, primary key (commitId), constraint fk7 foreign key(branchId) references branch(branchId));

**Output:** commit table will be displayed

```
mysql> select * from commits;
```

commitId	branchId	commitTime	noOfFiles	additions	deletions
1	1	2000-01-01 11:00:00	2	100	200
2	3	2000-01-01 11:00:00	2	1000	2300
3	4	2000-01-01 11:00:00	2	1000	2300
4	7	2000-01-01 11:00:00	2	2000	100
5	8	2000-01-01 11:00:00	2	200	1200
6	4	2000-01-01 11:00:00	2	200	1200

## QUESTION:

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

**Input:** select distinct(userid) from branch where (repoid=1 or repoid=2) and userid not in (select userid from branch where repoid=3);

**Output:** List of userid who created branch in repository x=1 and y=2 but not in z=3;

First, I created a list of all users who created a branch in either repository 1 or 2. Then a list of users created a branch in repository=3. Selected distinct userid who have created in 1 or 2 repository but not in 3. Distinct will remove repeating values in the userid column.

```
mysql> select distinct(userid) from branch where (repoid=1 or repoid=2) and userid not in(select userid from branch where repoid=3);
```

userid
1
2
5

**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).**

**Input:** select commitid, (deletions-additions) from commits where (deletions-additions)=(select (max(deletions-additions)) from commits);

**Output:** Display userid with maximum lines of code reduced. If there are two maximum lines of code reduced then two will be displayed.

Here, (deletions-additions) will be equal lines of code reduced. In inner query max of (deletions-additions) is calculated and displayed along with commitid to show which commit have highest lines of code reduced.

```
mysql> select commitid, (deletions-additions) from commits where (deletions-additions)=(select(max(deletions-additions)) from commits);
```

commitid	(deletions-additions)
2	1300
3	1300

One more example to check whether query is working perfectly. This time commits table is updated.

**Input:** Here commitid 7 is added.

```
mysql> select * from commits;
```

commitId	branchId	commitTime	noOfFiles	additions	deletions
1	1	2000-01-01 11:00:00	2	100	200
2	3	2000-01-01 11:00:00	2	1000	2300
3	4	2000-01-01 11:00:00	2	1000	2300
4	7	2000-01-01 11:00:00	2	2000	2100
5	8	2000-01-01 11:00:00	2	200	1200
6	4	2000-01-01 11:00:00	2	200	1200
7	4	2000-01-01 11:00:00	2	100	2000

## Output:

```
mysql> select commitid,(deletions-additions) from commits where (deletions-additions)=(select(max(deletions-additions)) from commits);
```

commitid	(deletions-additions)
7	1900

Hence, the query is working perfectly.

## Bonus Question

**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.)**

**Input:** Creation of issue table:

```
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));
```

```

Insert into issue values(1,1,'2000-01-01',2,'2000-02-02'); Insert into issue
values(2,2,'2000-01-01',1,'2000-02-02'); Insert into issue
values(3,1,'2000-01-01',3,'2000-02-02'); Insert into issue
values(4,3,'2000-01-01',5,'2000-02-02'); Insert into issue
values(5,4,'2000-01-01',5,'2000-02-02'); Insert into issue
values(6,3,'2000-01-01',4,'2000-02-02'); Insert into issue
values(7,5,'2000-01-01',3,'2000-02-02');

```

```

Select * from issue;

```

```
mysql> select * from issue;
```

issueId	creatorId	raiseDate	resolverId	resolveDate
1	1	2000-01-01	2	2000-02-02
2	2	2000-01-01	1	2000-02-02
3	1	2000-01-01	3	2000-02-02
4	3	2000-01-01	5	2000-02-02
5	4	2000-01-01	5	2000-02-02
6	3	2000-01-01	4	2000-02-02
7	5	2000-01-01	3	2000-02-02

**User Table:**

```
mysql> select * from users;
```

userId	noOfRepos	location	email	website	contributions
1	15	jampot	a@x.com	a.com	100
2	10	kampota	b@x.com	b.com	200
3	19	nampota	c@x.com	c.com	300
4	21	pampota	d@x.com	d.com	400
5	25	rampota	e@x.com	e.com	500

**Output:** select userid from users where ( select count(creatorid) from issue where users.userid=issue.creatorid group by userid)< ( select count(resolverid) from issue where users.userid=issue.resolverid group by userid);

```
mysql> select userid from users where ( select count(creatorid) from issue where users.userid=issue.creatorid group by userid)< ( select count(resolverid) from issue where users.userid=issue.resolverid group by userid);
+-----+
| userid |
+-----+
|      5 |
+-----+
```

In order to test the query, I modified the issue table.

Modified Table:

Insert into issue values (8,1,'2000-01-01',3,'2000-02-02'); Select \* from issue;

```
mysql> select * from issue;
+-----+-----+-----+-----+-----+
| issueId | creatorId | raiseDate | resolverId | resolveDate |
+-----+-----+-----+-----+-----+
|      1 |      1 | 2000-01-01 |      2 | 2000-02-02 |
|      2 |      2 | 2000-01-01 |      1 | 2000-02-02 |
|      3 |      1 | 2000-01-01 |      3 | 2000-02-02 |
|      4 |      3 | 2000-01-01 |      5 | 2000-02-02 |
|      5 |      4 | 2000-01-01 |      5 | 2000-02-02 |
|      6 |      3 | 2000-01-01 |      4 | 2000-02-02 |
|      7 |      5 | 2000-01-01 |      3 | 2000-02-02 |
|      8 |      1 | 2000-01-01 |      3 | 2000-02-02 |
+-----+-----+-----+-----+-----+
```

Executing query on the above table result will be:

```
mysql> select userid from users where ( select count(creatorid) from issue where users.userid=issue.creatorid group by userid)< ( select count(resolverid) from issue where users.userid=issue.resolverid group by userid);
+-----+
| userid |
+-----+
|      3 |
|      5 |
+-----+
```

In this query, I simply applied EquiJoin in both queries and compare them to give the result. Join is applied on users table and issue table. In first subquery users who raised the issues are counted and second subquery the users who resolved issues are counted. Both are compared, users with more issues resolved than created are displayed using select query.