Query the two cities in **STATION** with the shortest and longest *CITY* names, as well as their respective lengths (i.e.: number of characters in the name). If there is more than one smallest or largest city, choose the one that comes first when ordered alphabetically.  
The **STATION** table is described as follows:



where **LAT\_N** is the northern latitude and **LONG\_W** is the western longitude.

**Sample Input**

For example, **CITY** has four entries: **DEF, ABC, PQRS** and **WXY**.

**Sample Output**

ABC 3

PQRS 4

**Explanation**

When ordered alphabetically, the **CITY** names are listed as **ABC, DEF, PQRS,** and **WXY**, with lengths  and . The longest name is **PQRS**, but there are  options for shortest named city. Choose **ABC**, because it comes first alphabetically.

**Note**  
You can write two separate queries to get the desired output. It need not be a single query.

(SELECT CITY, LENGTH(CITY) FROM STATION ORDER BY LENGTH(CITY) ASC, CITY ASC limit 1) UNION ALL (SELECT CITY, LENGTH(CITY) FROM STATION ORDER BY LENGTH(CITY) DESC, CITY ASC limit 1);

Query the list of CITY names starting with vowels (i.e., a, e, i, o, or u) from **STATION**. Your result cannot contain duplicates.

**Input Format**

The **STATION** table is described as follows:



where LAT\_N is the northern latitude and LONG\_W is the western longitude.

SELECT CITY

FROM STATION

WHERE CITY REGEXP '^[AEIOU]'

Query the list of CITY names ending with vowels (a, e, i, o, u) from **STATION**. Your result cannot contain duplicates.

**Input Format**

The **STATION** table is described as follows:



where LAT\_N is the northern latitude and LONG\_W is the western longitude.

SELECT DISTINCT CITY

FROM STATION

WHERE CITY REGEXP '[aeiou]$'

Query the list of CITY names from **STATION** which have vowels (i.e., a, e, i, o, and u) as both their first and last characters. Your result cannot contain duplicates.

**Input Format**

The **STATION** table is described as follows:



where LAT\_N is the northern latitude and LONG\_W is the western longitude.

SELECT DISTINCT CITY

FROM STATION

WHERE CITY REGEXP '^[AEIOU]' AND CITY REGEXP '[aeiou]$'

Query the list of CITY names from **STATION** that do not start with vowels. Your result cannot contain duplicates.

**Input Format**

The **STATION** table is described as follows:



where LAT\_N is the northern latitude and LONG\_W is the western longitude.

SELECT DISTINCT CITY

FROM STATION

WHERE CITY NOT REGEXP '^[AEIOU]'

Query the list of CITY names from **STATION** that do not end with vowels. Your result cannot contain duplicates.

**Input Format**

The **STATION** table is described as follows:



where LAT\_N is the northern latitude and LONG\_W is the western longitude.

SELECT DISTINCT CITY

FROM STATION

WHERE CITY NOT REGEXP '[aeiou]$'

Query the list of CITY names from **STATION** that either do not start with vowels or do not end with vowels. Your result cannot contain duplicates.

**Input Format**

The **STATION** table is described as follows:



where LAT\_N is the northern latitude and LONG\_W is the western longitude.

SELECT DISTINCT CITY

FROM STATION

WHERE CITY NOT REGEXP '^[AEIOU]' OR CITY NOT REGEXP '[aeiou]$'

Query the list of CITY names from **STATION** that do not start with vowels and do not end with vowels. Your result cannot contain duplicates.

**Input Format**

The **STATION** table is described as follows:



where LAT\_N is the northern latitude and LONG\_W is the western longitude.

SELECT DISTINCT CITY

FROM STATION

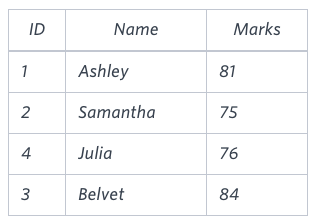
WHERE CITY NOT REGEXP '^[AEIOU]' AND CITY NOT REGEXP '[aeiou]$'

Query the Name of any student in **STUDENTS** who scored higher than  Marks. Order your output by the last three characters of each name. If two or more students both have names ending in the same last three characters (i.e.: Bobby, Robby, etc.), secondary sort them by ascending ID.

**Input Format**

The **STUDENTS** table is described as follows:The Name column only contains uppercase (A-Z) and lowercase (a-z) letters.

**Sample Input**



**Sample Output**

Ashley

Julia

Belvet

**Explanation**

Only Ashley, Julia, and Belvet have Marks > . If you look at the last three characters of each of their names, there are no duplicates and 'ley' < 'lia' < 'vet'.

SELECT NAME

FROM STUDENTS

WHERE Marks > 75

ORDER BY RIGHT(Name,3), ID ASC

Write a query that prints a list of employee names (i.e.: the name attribute) from the **Employee** table in alphabetical order.

**Input Format**

The **Employee** table containing employee data for a company is described as follows:



where employee\_id is an employee's ID number, name is their name, months is the total number of months they've been working for the company, and salary is their monthly salary.

**Sample Input**



**Sample Output**

Angela

Bonnie

Frank

Joe

Kimberly

Lisa

Michael

Patrick

Rose

Todd

SELECT name

FROM Employee

ORDER BY name

Write a query that prints a list of employee names (i.e.: the name attribute) for employees in **Employee** having a salary greater than  per month who have been employees for less than  months. Sort your result by ascending employee\_id.

**Input Format**

The **Employee** table containing employee data for a company is described as follows:



where employee\_id is an employee's ID number, name is their name, months is the total number of months they've been working for the company, and salary is the their monthly salary.

**Sample Input**



**Sample Output**

Angela

Michael

Todd

Joe

**Explanation**

Angela has been an employee for  month and earns  per month.

Michael has been an employee for  months and earns  per month.

Todd has been an employee for  months and earns  per month.

Joe has been an employee for  months and earns  per month.

We order our output by ascending employee\_id.

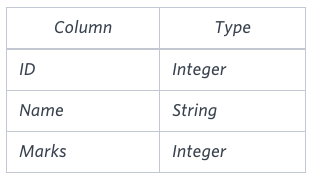
SELECT name

FROM Employee

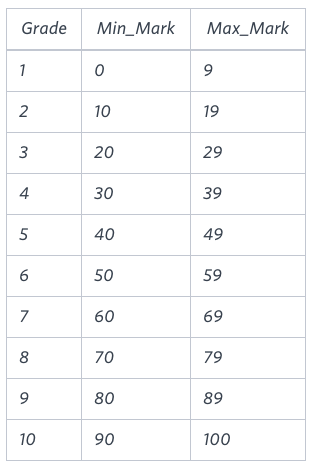
WHERE salary > 2000 AND months < 10

ORDER BY employee\_id

You are given two tables: *Students* and *Grades*. *Students* contains three columns *ID*, *Name* and *Marks*.



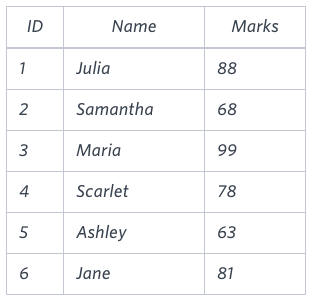
*Grades* contains the following data:



*Ketty* gives *Eve* a task to generate a report containing three columns: *Name*, *Grade* and *Mark*. *Ketty* doesn't want the NAMES of those students who received a grade lower than *8*. The report must be in descending order by grade -- i.e. higher grades are entered first. If there is more than one student with the same grade (8-10) assigned to them, order those particular students by their name alphabetically. Finally, if the grade is lower than 8, use "NULL" as their name and list them by their grades in descending order. If there is more than one student with the same grade (1-7) assigned to them, order those particular students by their marks in ascending order.

Write a query to help Eve.

**Sample Input**



**Sample Output**

Maria 10 99

Jane 9 81

Julia 9 88

Scarlet 8 78

NULL 7 63

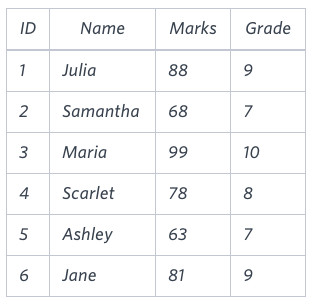
NULL 7 68

**Note**

Print "NULL"  as the name if the grade is less than 8.

**Explanation**

Consider the following table with the grades assigned to the students:



So, the following students got *8*, *9* or *10* grades:

* *Maria (grade 10)*
* *Jane (grade 9)*
* *Julia (grade 9)*
* *Scarlet (grade 8)*

select if(g.grade>7,s.name,null), g.grade, s.marks

from students s

inner join grades g

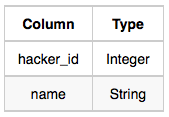
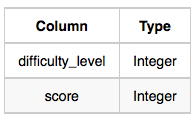
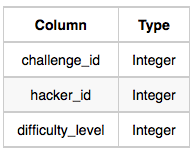
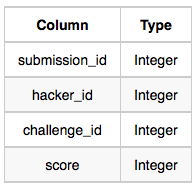
on s.marks >= g.min\_mark and s.marks <= g.max\_mark

ORDER BY g.grade DESC, Name ASC, s.marks ASC

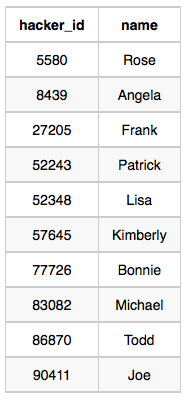
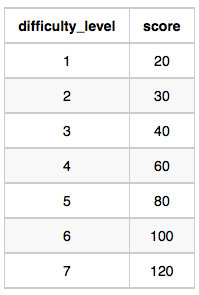
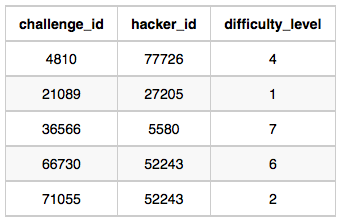
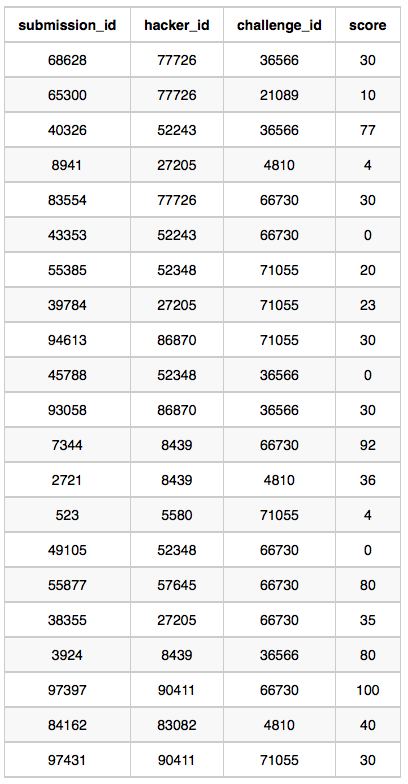
Julia just finished conducting a coding contest, and she needs your help assembling the leaderboard! Write a query to print the respective *hacker\_id* and *name* of hackers who achieved full scores for *more than one* challenge. Order your output in descending order by the total number of challenges in which the hacker earned a full score. If more than one hacker received full scores in same number of challenges, then sort them by ascending *hacker\_id*.

**Input Format**

The following tables contain contest data:

* *Hackers:* The *hacker\_id* is the id of the hacker, and *name* is the name of the hacker.
* *Difficulty:* The *difficult\_level* is the level of difficulty of the challenge, and *score* is the score of the challenge for the difficulty level.
* *Challenges:* The *challenge\_id* is the id of the challenge, the *hacker\_id* is the id of the hacker who created the challenge, and *difficulty\_level* is the level of difficulty of the challenge.
* *Submissions:* The *submission\_id* is the id of the submission, *hacker\_id* is the id of the hacker who made the submission, *challenge\_id* is the id of the challenge that the submission belongs to, and *score* is the score of the submission.

**Sample Input**

*Hackers* Table:*Difficulty* Table:*Challenges* Table:*Submissions* Table:

**Sample Output**

90411 Joe

**Explanation**

Hacker *86870* got a score of *30* for challenge *71055* with a difficulty level of *2*, so *86870* earned a full score for this challenge.

Hacker *90411* got a score of *30* for challenge *71055* with a difficulty level of *2*, so *90411* earned a full score for this challenge.

Hacker *90411* got a score of *100* for challenge *66730* with a difficulty level of *6*, so *90411* earned a full score for this challenge.

Only hacker *90411* managed to earn a full score for more than one challenge, so we print the their *hacker\_id* and *name* as  space-separated values.

SELECT s.hacker\_id, h.name

FROM

Submissions s JOIN Challenges c ON s.challenge\_id=c.challenge\_id

JOIN Difficulty d ON c.difficulty\_level=d.difficulty\_level

JOIN Hackers h ON h.hacker\_id=s.hacker\_id

WHERE d.score=s.score

GROUP BY s.hacker\_id, h.name

HAVING COUNT(c.challenge\_id)>1

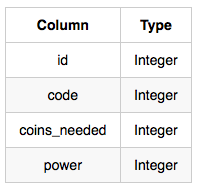
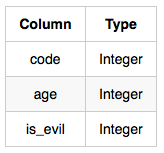
ORDER BY COUNT(c.challenge\_id) DESC, s.hacker\_id

Harry Potter and his friends are at Ollivander's with Ron, finally replacing Charlie's old broken wand.

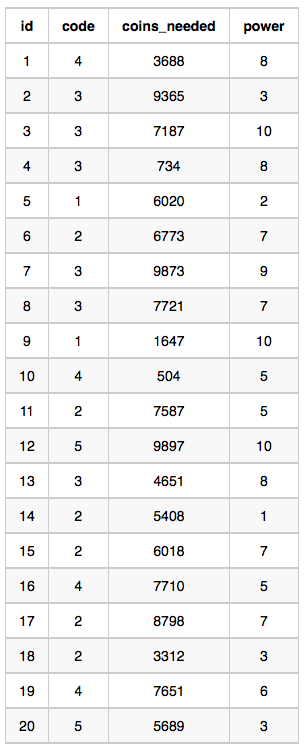
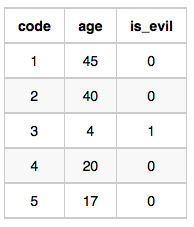
Hermione decides the best way to choose is by determining the minimum number of gold galleons needed to buy each *non-evil* wand of high power and age. Write a query to print the *id*, *age*, *coins\_needed*, and *power* of the wands that Ron's interested in, sorted in order of descending *power*. If more than one wand has same power, sort the result in order of descending *age*.

**Input Format**

The following tables contain data on the wands in Ollivander's inventory:

* *Wands:* The *id* is the id of the wand, *code* is the code of the wand, *coins\_needed* is the total number of gold galleons needed to buy the wand, and *power* denotes the quality of the wand (the higher the power, the better the wand is).
* *Wands\_Property:* The *code* is the code of the wand, *age* is the age of the wand, and *is\_evil* denotes whether the wand is good for the dark arts. If the value of *is\_evil* is *0*, it means that the wand is not evil. The mapping between *code* and *age* is one-one, meaning that if there are two pairs,  and , then  and .

**Sample Input**

*Wands* Table:*Wands\_Property* Table:

**Sample Output**

9 45 1647 10

12 17 9897 10

1 20 3688 8

15 40 6018 7

19 20 7651 6

11 40 7587 5

10 20 504 5

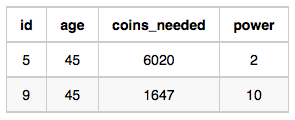
18 40 3312 3

20 17 5689 3

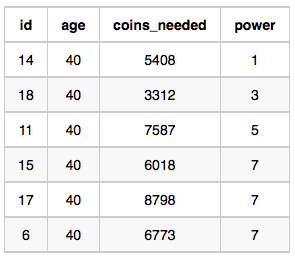
5 45 6020 2

14 40 5408 1

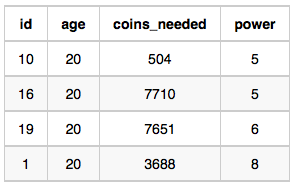
**Explanation**

The data for wands of *age 45* (code 1):

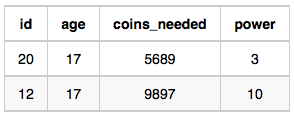
* The minimum number of galleons needed for
* The minimum number of galleons needed for

The data for wands of *age 40* (code 2):

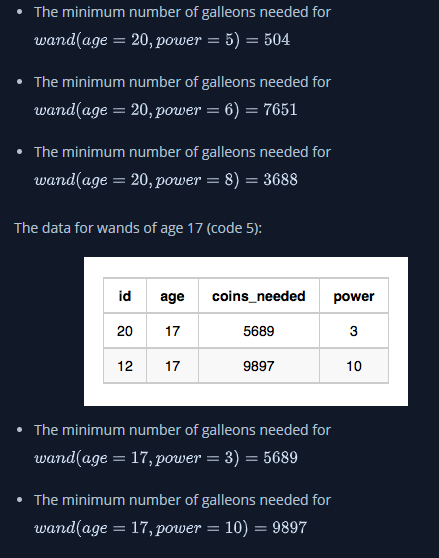
* The minimum number of galleons needed for
* The minimum number of galleons needed for
* The minimum number of galleons needed for
* The minimum number of galleons needed for

The data for wands of *age 20* (code 4):

* The minimum number of galleons needed for
* The minimum number of galleons needed for
* The minimum number of galleons needed for

The data for wands of *age 17* (code 5):

* The minimum number of galleons needed for
* The minimum number of galleons needed for



select

(select id from wands where code=w.code and coins\_needed=min(w.coins\_needed) and power=w.power) as id,

wp.age,

min(w.coins\_needed) as coins\_needed,

w.power

from

wands as w

LEFT JOIN

wands\_property as wp

ON

wp.code = w.code

where

wp.is\_evil = 0

group by

w.code, wp.age, w.power

order by

w.power desc, wp.age desc

;

Given the **CITY** and **COUNTRY** tables, query the sum of the populations of all cities where the *CONTINENT* is *'Asia'*.

**Note:** *CITY.CountryCode* and *COUNTRY.Code* are matching key columns.

**Input Format**

The **CITY** and **COUNTRY** tables are described as follows:



select sum(city.population) from city,country where CITY.CountryCode=COUNTRY.Code and country.continent='Asia';

Given the **CITY** and **COUNTRY** tables, query the names of all cities where the *CONTINENT* is *'Africa'*.

**Note:** *CITY.CountryCode* and *COUNTRY.Code* are matching key columns.

**Input Format**

The **CITY** and **COUNTRY** tables are described as follows:



Select CITY.NAME from CITY join COUNTRY on CITY.CountryCode= COUNTRY.Code where COUNTRY.CONTINENT='africa' ;

Given the **CITY** and **COUNTRY** tables, query the names of all the continents (*COUNTRY.Continent*) and their respective average city populations (*CITY.Population*) rounded *down* to the nearest integer.

**Note:** *CITY.CountryCode* and *COUNTRY.Code* are matching key columns.

**Input Format**

The **CITY** and **COUNTRY** tables are described as follows:



SELECT COUNTRY.Continent, FLOOR(AVG(CITY.population)) AS AVG\_CITY\_POPULATION

FROM COUNTRY

INNER JOIN CITY

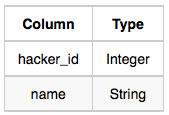
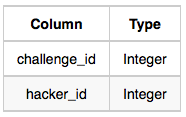
ON COUNTRY.CODE = CITY.COUNTRYCODE

GROUP BY COUNTRY.Continent

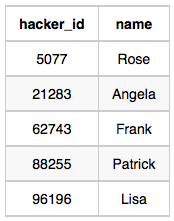
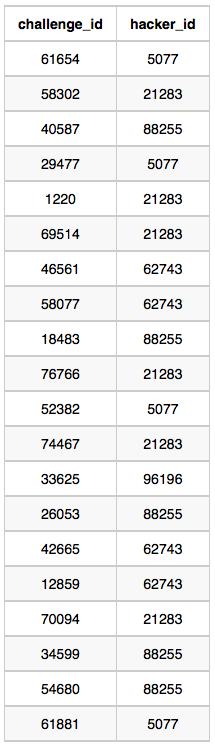
Julia asked her students to create some coding challenges. Write a query to print the *hacker\_id*, *name*, and the total number of challenges created by each student. Sort your results by the total number of challenges in descending order. If more than one student created the same number of challenges, then sort the result by *hacker\_id*. If more than one student created the same number of challenges and the count is less than the maximum number of challenges created, then exclude those students from the result.

**Input Format**

The following tables contain challenge data:

* *Hackers:* The *hacker\_id* is the id of the hacker, and *name* is the name of the hacker.
* *Challenges:* The *challenge\_id* is the id of the challenge, and *hacker\_id* is the id of the student who created the challenge.

**Sample Input 0**

*Hackers* Table:*Challenges* Table:

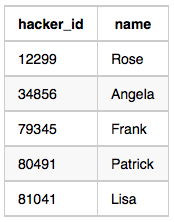
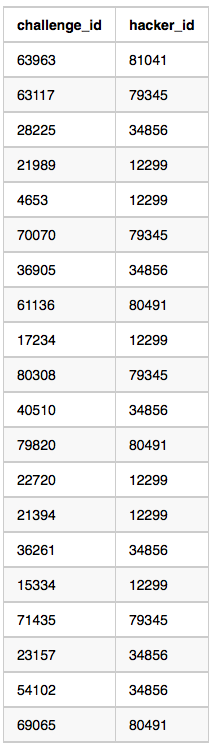
**Sample Output 0**

21283 Angela 6

88255 Patrick 5

96196 Lisa 1

**Sample Input 1**

*Hackers* Table:*Challenges* Table:

**Sample Output 1**

12299 Rose 6

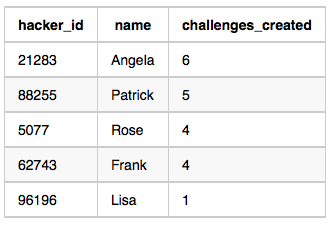
34856 Angela 6

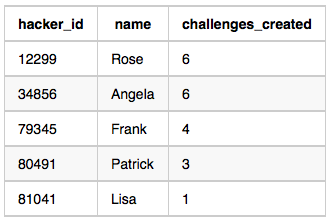
79345 Frank 4

80491 Patrick 3

81041 Lisa 1

**Explanation**

For *Sample Case 0*, we can get the following details:  
  
Students  and  both created  challenges, but the maximum number of challenges created is  so these students are excluded from the result.

For *Sample Case 1*, we can get the following details:  
  
Students  and  both created  challenges. Because  is the maximum number of challenges created, these students are included in the result.

with

cte1 as

(

Select h.hacker\_id,h.name,count(c.challenge\_id) as counter from Hackers h

INNER JOIN Challenges c on h.hacker\_id = c.hacker\_id

GROUP BY h.hacker\_id,h.name

ORDER BY COUNT(\*) DESC, h.hacker\_id

),

cte2 as (

SELECT counter,count(\*) FROM cte1

group by counter

HAVING count(\*) = 1)

SELECT

cte1.hacker\_id,

cte1.name,

cte1.counter

from cte1

where cte1.counter = (SELECT MAX(counter) from cte1 inner\_cte)

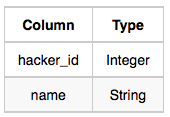
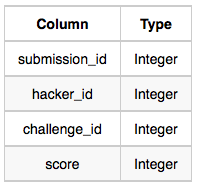
or cte1.counter in (select counter from cte2);

You did such a great job helping Julia with her last coding contest challenge that she wants you to work on this one, too!

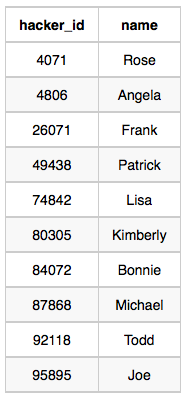
The total score of a hacker is the sum of their maximum scores for all of the challenges. Write a query to print the *hacker\_id*, *name*, and total score of the hackers ordered by the descending score. If more than one hacker achieved the same total score, then sort the result by ascending *hacker\_id*. Exclude all hackers with a total score of  from your result.

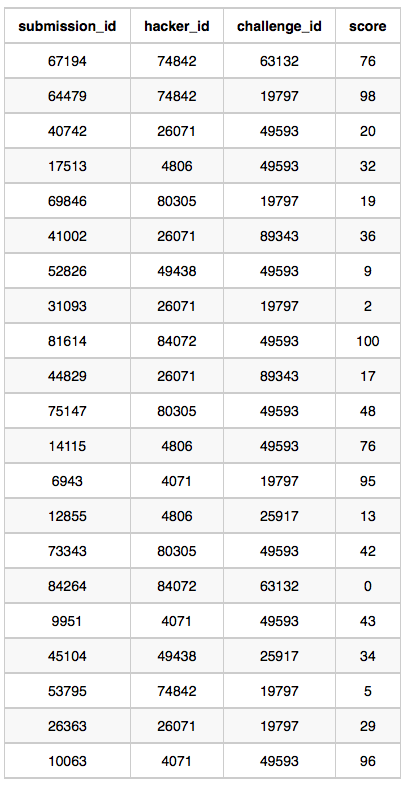
**Input Format**

The following tables contain contest data:

* *Hackers:* The *hacker\_id* is the id of the hacker, and *name* is the name of the hacker.
* *Submissions:* The *submission\_id* is the id of the submission, *hacker\_id* is the id of the hacker who made the submission, *challenge\_id* is the id of the challenge for which the submission belongs to, and *score* is the score of the submission.

**Sample Input**

*Hackers* Table:

*Submissions* Table:

**Sample Output**

4071 Rose 191

74842 Lisa 174

84072 Bonnie 100

4806 Angela 89

26071 Frank 85

80305 Kimberly 67

49438 Patrick 43

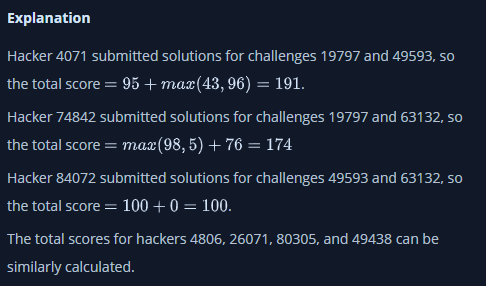
**Explanation**

Hacker *4071* submitted solutions for challenges *19797* and *49593*, so the total score .

Hacker *74842* submitted solutions for challenges *19797* and *63132*, so the total score

Hacker *84072* submitted solutions for challenges *49593* and *63132*, so the total score .

The total scores for hackers *4806*, *26071*, *80305*, and *49438* can be similarly calculated.



SELECT hackers.hacker\_id AS hacker\_id, hackers.name, Score\_card.totalScore AS totalScore

FROM hackers

INNER JOIN

(

SELECT hacker\_id, SUM(ChallengeScore) AS totalScore FROM

(

SELECT hacker\_id,challenge\_id,MAX(score) AS ChallengeScore

FROM submissions GROUP BY hacker\_id,challenge\_id

) AS Hackers\_score

GROUP BY hacker\_id

)

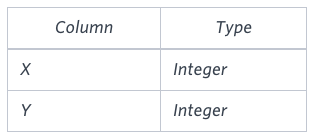
AS Score\_card

USING(hacker\_id)

WHERE totalScore>0

ORDER BY totalScore DESC, hacker\_id;

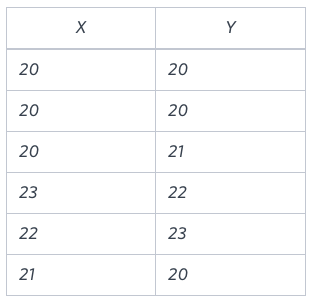
You are given a table, *Functions*, containing two columns: *X*and *Y*.



Two pairs *(X1, Y1)* and *(X2, Y2)* are said to be *symmetric* *pairs* if *X1 = Y2* and *X2 = Y1*.

Write a query to output all such *symmetric* *pairs* in ascending order by the value of *X*. List the rows such that *X1 ≤ Y1*.

**Sample Input**



**Sample Output**

20 20

20 21

22 23

with temp1 as(

select x as x1, y as y1, row\_number() over(order by x, y) num1

from functions

),

temp2 as(

select x1, y1, num1, x2, y2, num2, case when x2=y1 then 1 else 0 end as flag

from temp1

join (select x as x2, y as y2,row\_number() over(order by x, y) num2 from functions) m on x1=y2

)

select max(x1), max(y1)

from(

select x1, y1 , flag, num1, num2

from temp2

where x1<=y1 and flag=1 and num1<>num2) a

group by x1, y1

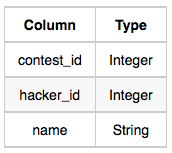
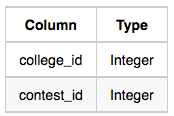
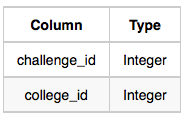
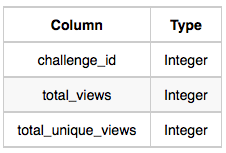
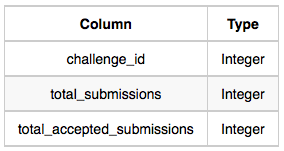
order by x1

Samantha interviews many candidates from different colleges using coding challenges and contests. Write a query to print the *contest\_id*, *hacker\_id*, *name*, and the sums of *total\_submissions*, *total\_accepted\_submissions*, *total\_views*, and *total\_unique\_views* for each contest sorted by *contest\_id*. Exclude the contest from the result if all four sums are .

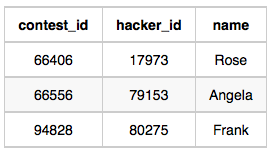
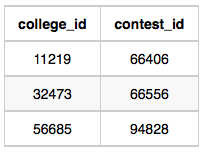
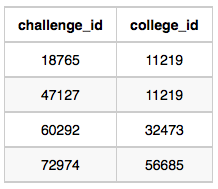
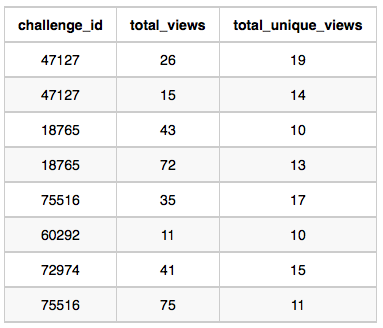
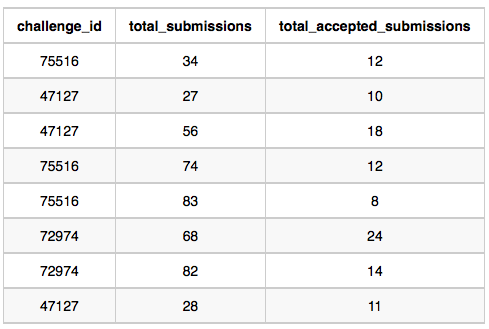
**Note:** A specific contest can be used to screen candidates at more than one college, but each college only holds  screening contest.

**Input Format**

The following tables hold interview data:

* *Contests:* The *contest\_id* is the id of the contest, *hacker\_id* is the id of the hacker who created the contest, and *name* is the name of the hacker.
* *Colleges:* The *college\_id* is the id of the college, and *contest\_id* is the id of the contest that Samantha used to screen the candidates.
* *Challenges:* The *challenge\_id* is the id of the challenge that belongs to one of the contests whose contest\_id Samantha forgot, and *college\_id* is the id of the college where the challenge was given to candidates.
* *View\_Stats:* The *challenge\_id* is the id of the challenge, *total\_views* is the number of times the challenge was viewed by candidates, and *total\_unique\_views* is the number of times the challenge was viewed by unique candidates.
* *Submission\_Stats:* The *challenge\_id* is the id of the challenge, *total\_submissions* is the number of submissions for the challenge, and *total\_accepted\_submission* is the number of submissions that achieved full scores.

**Sample Input**

*Contests* Table:*Colleges* Table:*Challenges* Table:*View\_Stats* Table:*Submission\_Stats* Table:

**Sample Output**

66406 17973 Rose 111 39 156 56

66556 79153 Angela 0 0 11 10

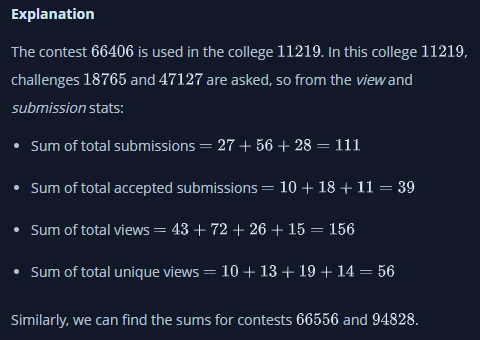
94828 80275 Frank 150 38 41 15

**Explanation**

The contest  is used in the college . In this college , challenges  and  are asked, so from the *view* and *submission* stats:

* Sum of total submissions
* Sum of total accepted submissions
* Sum of total views
* Sum of total unique views

Similarly, we can find the sums for contests  and .



select distinct

c.contest\_id,

c.hacker\_id,

c.name,

sum(case when t.rn\_st\_challenge\_id = 1 then t.total\_submissions else 0 end) over(partition by c.contest\_id, c.name) as sum\_total\_submissions,

sum(case when t.rn\_st\_challenge\_id = 1 then t.total\_accepted\_submissions else 0 end) over(partition by c.contest\_id, c.name) as sum\_total\_accepted\_submissions,

sum(case when t.rn\_vs\_challenge\_id = 1 then t.total\_views else 0 end) over(partition by c.contest\_id, c.name) as sum\_total\_views,

sum(case when t.rn\_vs\_challenge\_id = 1 then t.total\_unique\_views else 0 end) over(partition by c.contest\_id, c.name) as sum\_total\_unique\_views

from contests c

join colleges co on c.contest\_id = co.contest\_id

join challenges ch on co.college\_id = ch.college\_id

join (

(select

cha.college\_id,

cha.challenge\_id,

sum(st.total\_submissions) over (partition by cha.challenge\_id order by cha.challenge\_id) as total\_submissions,

sum(st.total\_accepted\_submissions) over (partition by cha.challenge\_id order by cha.challenge\_id) as total\_accepted\_submissions,

row\_number() over (partition by cha.challenge\_id order by cha.challenge\_id) as rn\_st\_challenge\_id,

null as total\_views,

null as total\_unique\_views,

null as rn\_vs\_challenge\_id

from challenges cha

join submission\_stats st on cha.challenge\_id = st.challenge\_id)

union all

(select

cha.college\_id,

cha.challenge\_id,

null as total\_submissions,

null total\_accepted\_submissions,

null as rn\_st\_challenge\_id,

sum(vs.total\_views) over (partition by cha.challenge\_id order by cha.challenge\_id) as total\_views,

sum(vs.total\_unique\_views) over (partition by cha.challenge\_id order by cha.challenge\_id) as total\_unique\_views,

row\_number() over (partition by cha.challenge\_id order by cha.challenge\_id) as rn\_vs\_challenge\_id

from challenges cha

join view\_stats vs on cha.challenge\_id = vs.challenge\_id)

) t on ch.challenge\_id = t.challenge\_id

order by c.contest\_id;

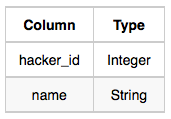
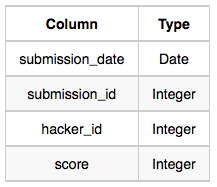
[HackerRank SQL challenge - Interviews (advanced joins) · GitHub](https://gist.github.com/eareese/5e962fec0ef866f4fa0c2c003c2794be#file-solution-sql-L1)

Julia conducted a  days of learning SQL contest. The start date of the contest was March 01, 2016 and the end date was March 15, 2016.

Write a query to print total number of unique hackers who made at least  submission each day (starting on the first day of the contest), and find the hacker\_id and name of the hacker who made maximum number of submissions each day. If more than one such hacker has a maximum number of submissions, print the lowest hacker\_id. The query should print this information for each day of the contest, sorted by the date.

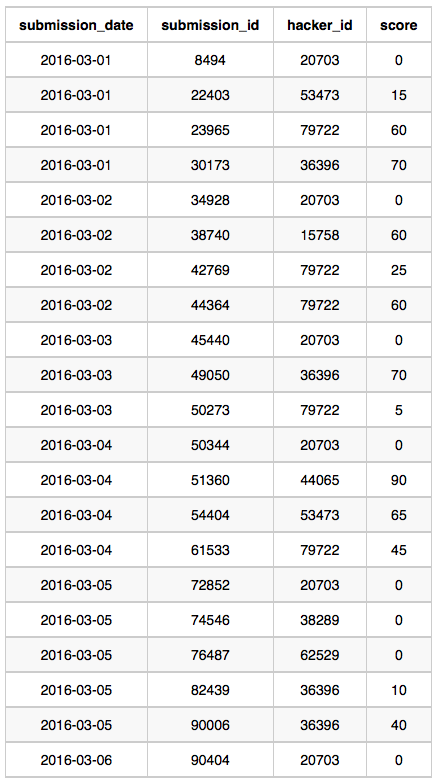
**Input Format**

The following tables hold contest data:

* Hackers: The hacker\_id is the id of the hacker, and name is the name of the hacker.
* Submissions: The submission\_date is the date of the submission, submission\_id is the id of the submission, hacker\_id is the id of the hacker who made the submission, and score is the score of the submission.

**Sample Input**

For the following sample input, assume that the end date of the contest was March 06, 2016.

Hackers Table:Submissions Table:

**Sample Output**

2016-03-01 4 20703 Angela

2016-03-02 2 79722 Michael

2016-03-03 2 20703 Angela

2016-03-04 2 20703 Angela

2016-03-05 1 36396 Frank

2016-03-06 1 20703 Angela

**Explanation**

On March 01, 2016 hackers , , , and  made submissions. There are  unique hackers who made at least one submission each day. As each hacker made one submission,  is considered to be the hacker who made maximum number of submissions on this day. The name of the hacker is Angela.

On March 02, 2016 hackers , , and  made submissions. Now  and  were the only ones to submit every day, so there are  unique hackers who made at least one submission each day.  made  submissions, and name of the hacker is Michael.

On March 03, 2016 hackers , , and  made submissions. Now  and  were the only ones, so there are  unique hackers who made at least one submission each day. As each hacker made one submission so  is considered to be the hacker who made maximum number of submissions on this day. The name of the hacker is Angela.

On March 04, 2016 hackers , , , and  made submissions. Now  and  only submitted each day, so there are  unique hackers who made at least one submission each day. As each hacker made one submission so  is considered to be the hacker who made maximum number of submissions on this day. The name of the hacker is Angela.

On March 05, 2016 hackers , ,  and  made submissions. Now  only submitted each day, so there is only  unique hacker who made at least one submission each day.  made  submissions and name of the hacker is Frank.

On March 06, 2016 only  made submission, so there is only  unique hacker who made at least one submission each day.  made  submission and name of the hacker is Angela.

WITH A AS

(

SELECT submission\_date, hacker\_id, COUNT(\*) AS num\_sm

FROM Submissions

GROUP BY submission\_date, hacker\_id

),

B AS

(

SELECT A.submission\_date, A.hacker\_id,

RANK() OVER(

PARTITION BY A.submission\_date

ORDER BY A.num\_sm DESC, A.hacker\_id) AS rank

FROM A

),

C AS

(

SELECT B.submission\_date, B.hacker\_id

FROM B

WHERE B.rank=1

),

D AS

(

SELECT s1.submission\_date, COUNT(DISTINCT s1.hacker\_id) AS num\_hacker

FROM Submissions s1

WHERE hacker\_id IN (

SELECT s2.hacker\_id

FROM Submissions s2

WHERE s2.submission\_date >= "2016-03-01"

AND s2.submission\_date <= s1.submission\_date

GROUP BY s2.hacker\_id

HAVING

COUNT(DISTINCT s2.submission\_date)=(

SELECT COUNT(DISTINCT s3.submission\_date)

FROM Submissions s3

WHERE s3.submission\_date <= s1.submission\_date

)

)

GROUP BY s1.submission\_date

)

SELECT D.submission\_date, D.num\_hacker, h.hacker\_id, h.name

FROM D JOIN C ON D.submission\_date=C.submission\_date

JOIN Hackers h ON C.hacker\_id=h.hacker\_id

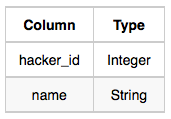
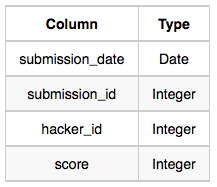
ORDER BY D.submission\_date

Julia conducted a  days of learning SQL contest. The start date of the contest was March 01, 2016 and the end date was March 15, 2016.

Write a query to print total number of unique hackers who made at least  submission each day (starting on the first day of the contest), and find the hacker\_id and name of the hacker who made maximum number of submissions each day. If more than one such hacker has a maximum number of submissions, print the lowest hacker\_id. The query should print this information for each day of the contest, sorted by the date.

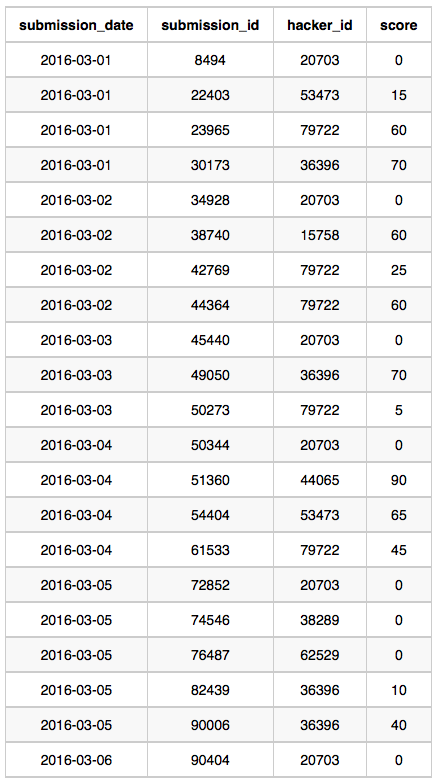
**Input Format**

The following tables hold contest data:

* Hackers: The hacker\_id is the id of the hacker, and name is the name of the hacker.
* Submissions: The submission\_date is the date of the submission, submission\_id is the id of the submission, hacker\_id is the id of the hacker who made the submission, and score is the score of the submission.

**Sample Input**

For the following sample input, assume that the end date of the contest was March 06, 2016.

Hackers Table:Submissions Table:

**Sample Output**

2016-03-01 4 20703 Angela

2016-03-02 2 79722 Michael

2016-03-03 2 20703 Angela

2016-03-04 2 20703 Angela

2016-03-05 1 36396 Frank

2016-03-06 1 20703 Angela

**Explanation**

On March 01, 2016 hackers , , , and  made submissions. There are  unique hackers who made at least one submission each day. As each hacker made one submission,  is considered to be the hacker who made maximum number of submissions on this day. The name of the hacker is Angela.

On March 02, 2016 hackers , , and  made submissions. Now  and  were the only ones to submit every day, so there are  unique hackers who made at least one submission each day.  made  submissions, and name of the hacker is Michael.

On March 03, 2016 hackers , , and  made submissions. Now  and  were the only ones, so there are  unique hackers who made at least one submission each day. As each hacker made one submission so  is considered to be the hacker who made maximum number of submissions on this day. The name of the hacker is Angela.

On March 04, 2016 hackers , , , and  made submissions. Now  and  only submitted each day, so there are  unique hackers who made at least one submission each day. As each hacker made one submission so  is considered to be the hacker who made maximum number of submissions on this day. The name of the hacker is Angela.

On March 05, 2016 hackers , ,  and  made submissions. Now  only submitted each day, so there is only  unique hacker who made at least one submission each day.  made  submissions and name of the hacker is Frank.

On March 06, 2016 only  made submission, so there is only  unique hacker who made at least one submission each day.  made  submission and name of the hacker is Angela.

In MS SQL servers.

-- Get the total sub count of a hacker for each date. Will be used to find the the hacker with the most subs later.

with hackersubs (submission\_date, subcount, hacker\_id) as

(

select submission\_date, COUNT(submission\_date) as subcount, hacker\_id FROM submissions

GROUP BY submission\_date,hacker\_id

),

-- This was gnarly. Gets the count of unique hackers that are keeping up the streak from day 1.

-- Works a bit like a for loop

-- For each distinct submission date of a hacker, get the count of dates which are earlier

-- If this count is the same as the number of days since challenge start, then the hacker has kept their streak for that date.

everydaysubbers (submission\_date, countz) as

(

select s1.submission\_date, COUNT(DISTINCT hacker\_id) FROM submissions s1

WHERE

(select count(distinct s2.submission\_date) FROM submissions s2 WHERE s1.hacker\_id = s2.hacker\_id AND s2.submission\_date < s1.submission\_date) = datediff(day, '2016-03-01', s1.submission\_date) GROUP BY s1.submission\_date

),

-- Collect it all together and add a row number. The row number will be used to select the lower hacker ID when there is a tie.

results as (SELECT sbms.submission\_date as subdate, es.countz as unique\_subs, hs1.hacker\_id as hackerid, hs.name as hackername, ROW\_NUMBER() OVER (PARTITION BY sbms.submission\_date ORDER BY hs1.hacker\_id) as rn

FROM submissions sbms

INNER JOIN hackersubs hs1

ON sbms.submission\_date = hs1.submission\_date

INNER JOIN hackers hs

ON hs1.hacker\_id = hs.hacker\_id

INNER JOIN everydaysubbers es

ON sbms.submission\_date = es.submission\_date

WHERE hs1.subcount = (SELECT MAX(subcount) FROM hackersubs hs2 WHERE hs2.submission\_date = sbms.submission\_date)

GROUP BY sbms.submission\_date,es.countz, hs1.hacker\_id, hs.name)

SELECT subdate, unique\_subs, hackerid, hackername FROM results where rn = 1 ORDER BY subdate, hackerid;

*P(R)* represents a pattern drawn by Julia in *R* rows. The following pattern represents *P(5)*:

\*

\* \*

\* \* \*

\* \* \* \*

\* \* \* \* \*

Write a query to print the pattern *P(20)*.

WITH RECURSIVE numbers AS (

select 1 as n

UNION ALL

select numbers.n + 1

from numbers

where numbers.n <= 19

)

select REPEAT("\* ", numbers.n) from numbers order by numbers.n asc

Write a query to print all *prime numbers* less than or equal to . Print your result on a single line, and use the ampersand () character as your separator (instead of a space).

For example, the output for all prime numbers  would be:

2&3&5&7

DELIMITER //

CREATE PROCEDURE PRIMEFUNC(OUT foutput VARCHAR(1200))

BEGIN

DECLARE num INT;

DECLARE counter INT;

DECLARE isprime INT;

SET num = 3;

SET foutput = "2";

outerlabel : LOOP

SET counter = 2;

SET isprime = 1;

innerlabel : LOOP

IF counter <= sqrt(num) THEN

IF num mod counter = 0

THEN SET isprime =0;

LEAVE innerlabel;

END IF;

SET counter = counter + 1;

ITERATE innerlabel;

ELSE

LEAVE innerlabel;

END IF;

END LOOP innerlabel;

IF isprime = 1

THEN SET foutput = CONCAT(foutput,"&",num);

END IF;

SET num = num + 1;

IF num <= 1000 THEN ITERATE outerlabel;

ELSE LEAVE outerlabel;

END IF;

END LOOP outerlabel;

END//

CALL PRIMEFUNC(@fop);

SELECT @fop

P(R) represents a pattern drawn by Julia in R rows. The following pattern represents P(5):

\* \* \* \* \*

\* \* \* \*

\* \* \*

\* \*

\*

Write a query to print the pattern P(20).

WITH RECURSIVE numbers AS (

select 1 as n

UNION ALL

select numbers.n + 1

from numbers

where numbers.n <= 19

)

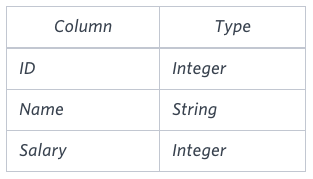
select REPEAT("\* ", numbers.n) from numbers order by numbers.n desc

Samantha was tasked with calculating the average monthly salaries for all employees in the **EMPLOYEES** table, but did not realize her keyboard's  key was broken until after completing the calculation. She wants your help finding the difference between her miscalculation (using salaries with any zeros removed), and the actual average salary.

Write a query calculating the amount of error (i.e.:  average monthly salaries), and round it up to the next integer.

**Input Format**

The **EMPLOYEES** table is described as follows:

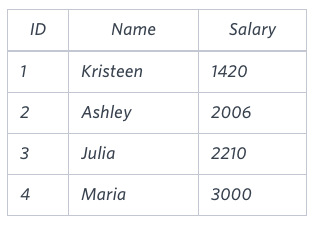


**Note:** Salary is per month.

**Constraints**

.

**Sample Input**

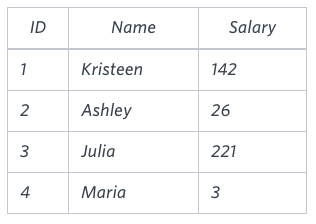


**Sample Output**

2061

**Explanation**

The table below shows the salaries without zeros as they were entered by Samantha:



Samantha computes an average salary of . The actual average salary is .

The resulting error between the two calculations is . Since it is equal to the integer , it does not get rounded up.

SELECT CEILING(AVG(SALARY) - AVG(REPLACE(SALARY,'0','')))

FROM EMPLOYEES