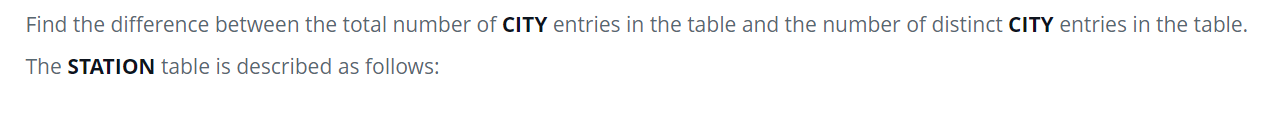


SELECT DISTINCT CITY FROM STATION WHERE MOD(ID,2)=0 ;

select COUNT(CITY) - COUNT(DISTINCT(CITY)) FROM STATION;

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.

SELECT CITY,LENGTH(CITY)AS CITYLENGTH FROM STATION ORDER BY CITY,CITYLENGTH LIMIT 1 ;

SELECT CITY,LENGTH(CITY)AS CITYLENGTH FROM STATION ORDER BY CITY,CITYLENGTH DESC 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.

REGEXP\_LIKE used in where clause to filter rows based on a specific regular expression pattern.

SELECT DISTINCT CITY FROM STATION WHERE REGEXP\_LIKE(CITY,'^[aeiou]') ;

Or

SELECT DISTINCT City FROM Station

WHERE City LIKE 'a%' or City LIKE 'e%' or City LIKE 'i%' or City LIKE 'o%' or City LIKE 'u%';

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

SELECT DISTINCT CITY FROM STATION WHERE CITY LIKE '%a' or CITY LIKE '%e' or CITY LIKE '%i' or CITY LIKE '%o' or CITY LIKE '%u' ;

Or SELECT DISTINCT CITY FROM STATION WHERE REGEXP\_LIKE(CITY,'[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.

SELECT DISTINCT CITY FROM STATION WHERE REGEXP\_LIKE(CITY,'^[aeiou].\*[aeiou]$') ;

This query uses the REGEXP\_LIKE function to filter cities where the CITY starts and ends with any vowel. The regular expression ^[aeiouAEIOU].\*[aeiouAEIOU]$ specifies:

^[aeiouAEIOU]: The string should start with any vowel.

.\*: There can be any characters in between.

[aeiouAEIOU]$: The string should end with any vowel.

Or SELECT DISTINCT CITY FROM STATION WHERE (CITY LIKE 'a%' OR CITY LIKE 'e%' OR CITY LIKE 'i%' OR CITY LIKE 'o%' OR CITY LIKE 'u%') AND (CITY LIKE '%a' OR CITY LIKE '%e' OR CITY LIKE '%i' OR CITY LIKE '%o' OR CITY LIKE '%u') ;

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

select DISTINCT CITY FROM STATION WHERE NOT REGEXP\_LIKE(CITY,'^[aeiou]');

or select DISTINCT CITY FROM STATION WHERE lower(CITY) NOT LIKE 'a%' AND lower(CITY) NOT LIKE 'e%' AND lower(CITY) NOT LIKE 'i%' AND lower(CITY) NOT LIKE 'o%' AND lower(CITY) NOT LIKE 'u%' ;

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

SELECT DISTINCT CITY FROM STATION WHERE CITY NOT LIKE '%a' AND CITY NOT LIKE '%e' AND CITY NOT LIKE '%i' AND CITY NOT LIKE '%o' AND CITY NOT LIKE '%u';

Or SELECT DISTINCT CITY FROM STATION WHERE NOT REGEXP\_LIKE(CITY,'[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.

SELECT DISTINCT CITY FROM STATION WHERE NOT REGEXP\_LIKE(lower(CITY),'^[aeiou].\*[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.

SELECT DISTINCT CITY FROM STATION WHERE NOT REGEXP\_LIKE(lower(CITY),'^[aeiou]') AND

NOT REGEXP\_LIKE(lower(CITY),'[aeiou]$');

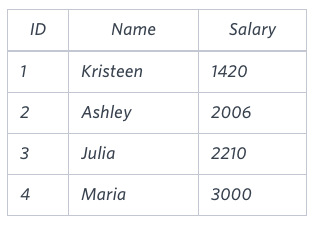
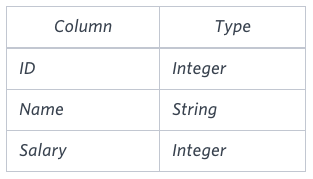
Or SELECT DISTINCT CITY FROM STATION WHERE CITY NOT LIKE 'a%' AND CITY NOT LIKE 'e%' AND CITY NOT LIKE 'i%' AND CITY NOT LIKE 'O%' AND CITY NOT LIKE 'i%' AND CITY NOT LIKE '%a' AND CITY NOT LIKE '%e' AND CITY NOT LIKE '%i' AND CITY NOT LIKE '%o' AND CITY NOT LIKE '%u' AND CITY NOT LIKE 'A%' AND CITY NOT LIKE 'E%' AND CITY NOT LIKE 'I%' AND CITY NOT LIKE 'O%' AND CITY NOT LIKE 'U%' AND CITY NOT LIKE '%A' AND CITY NOT LIKE '%E' AND CITY NOT LIKE '%I' AND CITY NOT LIKE '%O' AND CITY NOT LIKE '%U' ;

Query the Name of any student in **STUDENTS** who scored higher than 75 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.

SELECT Name FROM STUDENTS WHERE Marks>75 ORDER BY SUBSTR(Name,Length(Name)-2,3),ID ;

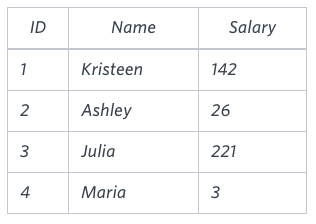
Samantha was tasked with calculating the average monthly salaries for all employees in the **EMPLOYEES** table but did not realize her keyboard's  key was broken until after completing the calculation. She wants your help finding the difference between her miscalculation (using salaries with any zeros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



Sample output : 2061

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



Samantha computes an average salary of 98.00. The actual average salary is  2159.00.

The resulting error between the two calculations is 2159.00-98.00 = 2061.00 . Since it is equal to the integer 2061 , it does not get rounded up

Op:

SELECT CEIL(AVG(SALARY)- AVG(TO\_NUMBER(REPLACE(TO\_CHAR(SALARY),'0','')))) FROM EMPLOYEES ;

We define an employee's total earnings to be their monthly  SALARY\*MONTHS worked, and the maximum total earnings to be the maximum total earnings for any employee in the **Employee** table. Write a query to find the maximum total earnings for all employees as well as the total number of employees who have maximum total earnings. Then print these values as  2 space-separated integers.

SELECT MAX(MONTHS\*SALARY) AS EARNINGS , COUNT(\*) AS TOTAL\_NO\_OF\_EMPLOYEES FROM EMPLOYEE WHERE MONTHS\*SALARY =(SELECT MAX(MONTHS\*SALARY) FROM EMPLOYEE);

Query the Western Longitude (LONG\_W) for the largest Northern Latitude (LAT\_N) in **STATION** that is less than 137.2345 . Round your answer to 4  decimal places.



SELECT ROUND(LONG\_W,4) FROM STATION WHERE LAT\_N =(SELECT MAX(LAT\_N) FROM STATION WHERE LAT\_N<137.2345) ;

Query the smallest Northern Latitude (LAT\_N) from **STATION** that is greater than 38.7780 . Round your answer to 4  decimal places.

SELECT ROUND(LAT\_N,4) FROM STATION WHERE LAT\_N =(SELECT MIN(LAT\_N) FROM STATION WHERE LAT\_N>38.7780);

Query the Western Longitude (LONG\_W)where the smallest Northern Latitude (LAT\_N) in **STATION** is greater than 38.7780 . Round your answer to  4 decimal places.

SELECT ROUND(LONG\_W,4) FROM STATION WHERE LAT\_N=(SELECT MIN(LAT\_N) FROM STATION WHERE LAT\_N>38.7780);

Consider P1(a,b)  and P2(c,d)  to be two points on a *2D* plane.

* a happens to equal the minimum value in *Northern Latitude* (*LAT\_N* in **STATION**).
* b happens to equal the minimum value in *Western Longitude* (*LONG\_W* in **STATION**).
* c happens to equal the maximum value in *Northern Latitude* (*LAT\_N* in **STATION**).
* d happens to equal the maximum value in *Western Longitude* (*LONG\_W* in **STATION**).

Query the [Manhattan Distance](https://xlinux.nist.gov/dads/HTML/manhattanDistance.html) between points P1 and P2 and round it to a scale of 4 decimal places.

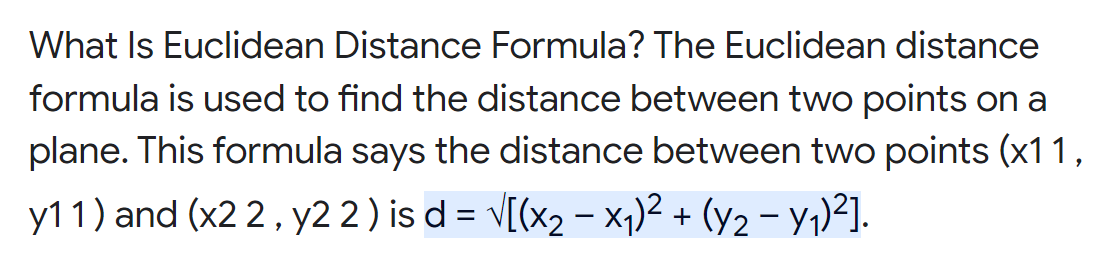
*p*1 at coordinates (*x*1, *y*1) and its nearest neighbor *p*2 at coordinates (*x*2, *y*2) as

Manhattan Distance = |x1-x2| + |y1-y2|

Op: SELECT ROUND(ABS(MIN(LAT\_N)-MAX(LAT\_N))+ ABS(MIN(LONG\_W)-MAX(LONG\_W)),4) FROM STATION;

Consider P1(a,c) and P2(b,d) to be two points on a 2D plane where (a,b) are the respective minimum and maximum values of *Northern Latitude* (*LAT\_N*) and  (c,d) are the respective minimum and maximum values of *Western Longitude* (*LONG\_W*) in **STATION**.

Query the [Euclidean Distance](https://en.wikipedia.org/wiki/Euclidean_distance) between points P1 and P2  and *format your answer* to display 4 decimal digits.



Op: SELECT ROUND(SQRT(POWER(MAX(LAT\_N)-MIN(LAT\_N),2) + POWER(MAX(LONG\_W)-MIN(LONG\_W),2)),4) FROM STATION ;

A [*median*](https://en.wikipedia.org/wiki/Median) is defined as a number separating the higher half of a data set from the lower half. Query the median of the Northern Latitudes (LAT\_N) from **STATION** and round your answer to 4 decimal places.

SELECT ROUND(PERCENTILE\_CONT(0.5) WITHIN GROUP (ORDER BY LAT\_N) , 4) AS median\_latitude FROM STATION;

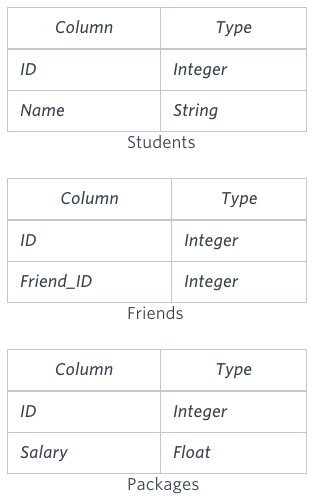
Or

SELECT ROUND(LAT\_N,4) FROM (SELECT ROW\_NUMBER() OVER(ORDER BY LAT\_N ASC) AS RNK,LAT\_N FROM STATION) WHERE RNK =(select round((count(\*)/2)) from station);

Query the total population of all cities in **CITY** where District is **California**.

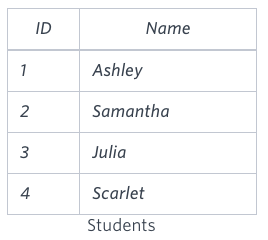
SELECT SUM(POPULATION) FROM CITY where District = 'California' ;

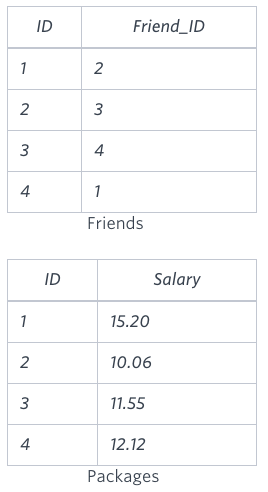
You are given three tables: Students, Friends and Packages. Students contains two columns: ID and Name. Friends contains two columns: ID and Friend\_ID (ID of the ONLY best friend). Packages contains two columns: ID and Salary (offered salary in $ thousands per month).



Write a query to output the names of those students whose best friends got offered a higher salary than them. Names must be ordered by the salary amount offered to the best friends. It is guaranteed that no two students got same salary offer.

Sample input:

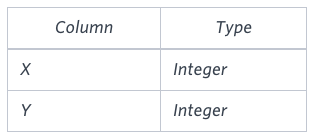




select s.Name from Students s inner join Friends f on s.ID=f.ID inner join Packages p on f.ID=p.ID inner join Packages ps on ps.ID=f.Friend\_ID

WHERE ps.Salary>p.Salary order by ps.Salary ;

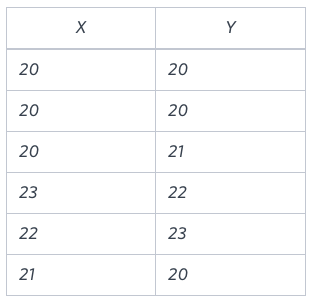
You are given a table, Functions, containing two columns: X and Y.



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:



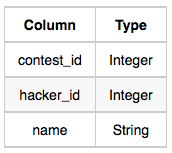
Samantha interviews many candidates from different colleges using coding challenges and contests. Write a query to print the *contest\_id*, *hacker\_id*, *name*, and the sums of *total\_submissions*, *total\_accepted\_submissions*, *total\_views*, and *total\_unique\_views* for each contest sorted by *contest\_id*. Exclude the contest from the result if all four sums are 0.

**Note:** A specific contest can be used to screen candidates at more than one college, but each college only holds 1  screening contest.

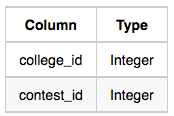
**Input Format**

The following tables hold interview data:

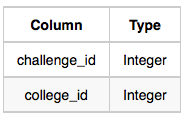
* *Contests:* The *contest\_id* is the id of the contest, *hacker\_id* is the id of the hacker who created the contest, and *name* is the name of the hacker.



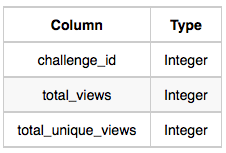
* *Colleges:* The *college\_id* is the id of the college, and *contest\_id* is the id of the contest that Samantha used to screen the candidates.



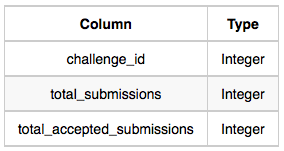
* *Challenges:* The *challenge\_id* is the id of the challenge that belongs to one of the contests whose contest\_id Samantha forgot, and *college\_id* is the id of the college where the challenge was given to candidates.



* *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.

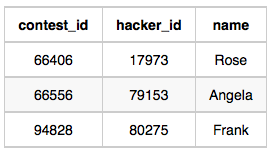
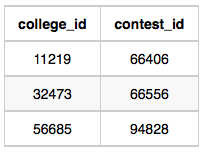


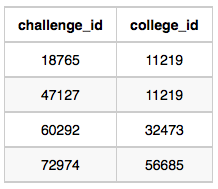
* *Submission\_Stats:* The *challenge\_id* is the id of the challenge, *total\_submissions* is the number of submissions for the challenge, and *total\_accepted\_submission* is the number of submissions that achieved full scores.



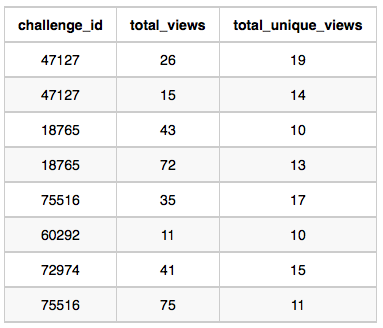
**Sample Input**

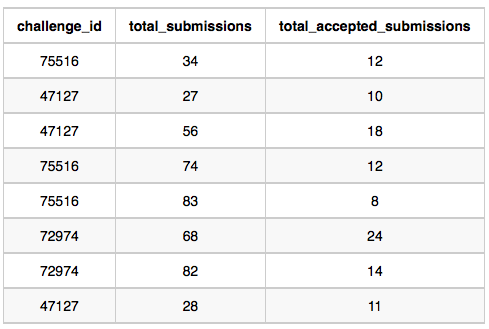
*Contests* Table: *Colleges* Table:

Challenges Table:

*View\_Stats* Table:



*Submission\_Stats* Table: 

SELECT \* FROM(

SELECT Con.contest\_id,Con.hacker\_id,Con.name,SUM(b.ts) ts,SUM(b.tas) tas,SUM(a.tv) tv,SUM(a.tuv) tuv FROM Contests Con , Colleges Col ,Challenges Cha,(SELECT challenge\_id,SUM(total\_views) tv ,SUM(total\_unique\_views) tuv FROM View\_Stats GROUP BY challenge\_id)a,(SELECT challenge\_id,SUM(total\_submissions) ts ,SUM(total\_accepted\_submissions) tas FROM Submission\_Stats GROUP BY challenge\_id)b WHERE Con.contest\_id=Col.contest\_id(+) AND

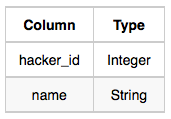
Col.college\_id=Cha.college\_id(+) AND Cha.challenge\_id=a.challenge\_id(+) and Cha.challenge\_id=b.challenge\_id(+) GROUP BY Con.contest\_id,Con.hacker\_id,Con.name ORDER BY contest\_id) WHERE (ts+tas+tv+tuv)<>0;

Julia conducted a 15 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.

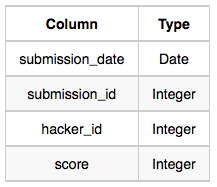
**Input Format**

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.



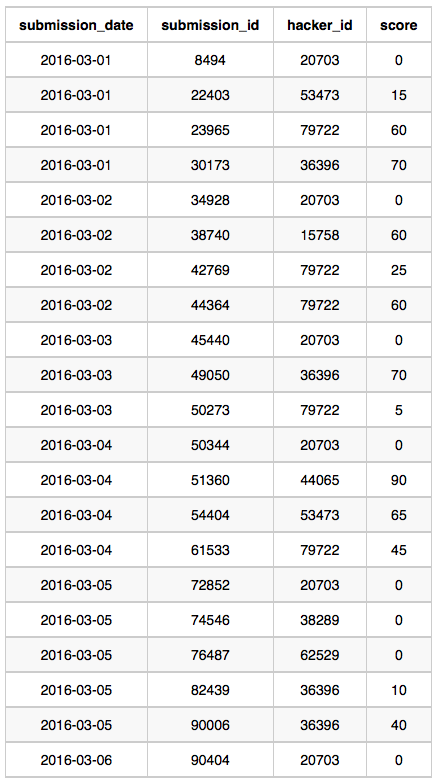
**Sample Input**

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

Op:

Amber's conglomerate corporation just acquired some new companies. Each of the companies follows this hierarchy:



Given the table schemas below, write a query to print the *company\_code*, *founder* name, total number of *lead* managers, total number of *senior* managers, total number of *managers*, and total number of *employees*. Order your output by ascending *company\_code*.

**Note:**

* The tables may contain duplicate records.
* The *company\_code* is string, so the sorting should not be **numeric**. For example, if the *company\_codes* are *C\_1*, *C\_2*, and *C\_10*, then the ascending *company\_codes* will be *C\_1*, *C\_10*, and *C\_2*.

**Input Format**

The following tables contain company data:

* *Company:* The *company\_code* is the code of the company and *founder* is the founder of the company.



* *Lead\_Manager:* The *lead\_manager\_code* is the code of the lead manager, and the *company\_code* is the code of the working company. 
* *Senior\_Manager:* The *senior\_manager\_code* is the code of the senior manager, the *lead\_manager\_code* is the code of its lead manager, and the *company\_code* is the code of the working company. 
* *Manager:* The *manager\_code* is the code of the manager, the *senior\_manager\_code* is the code of its senior manager, the *lead\_manager\_code* is the code of its lead manager, and the *company\_code* is the code of the working company. 
* *Employee:* The *employee\_code* is the code of the employee, the *manager\_code* is the code of its manager, the *senior\_manager\_code* is the code of its senior manager, the *lead\_manager\_code* is the code of its lead manager, and the *company\_code* is the code of the working company. 

**Sample Input**

*Company* Table: 

*Lead\_Manager* Table: 

*Senior\_Manager* Table: 

*Manager* Table: 

*Employee* Table: 

**Sample Output**

C1 Monika 1 2 1 2

C2 Samantha 1 1 2

Op: select c.company\_code,c.founder,count(distinct lm.lead\_manager\_code),count(distinct sm.senior\_manager\_code),count(distinct m.manager\_code),count(distinct e.employee\_code) from Company c inner join lead\_manager lm on c.company\_code =lm.company\_code

inner join senior\_manager sm on sm.lead\_manager\_code=lm.lead\_manager\_code

inner join Manager m on m.senior\_manager\_code =sm.senior\_manager\_code

inner join Employee e on e.manager\_code =m.manager\_code

inner join company on c.company\_code=e.company\_code

GROUP BY c.company\_code,c.founder order by c.company\_code asc;

Or

select c.company\_code,

c.founder,

count(distinct e.lead\_manager\_code),

count(distinct e.senior\_manager\_code),

count(distinct e.manager\_code),

count(distinct e.employee\_code)

from company c

inner join employee e on e.company\_code = c.company\_code

group by c.company\_code,c.founder

order by c.company\_code;

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:



SELECT \* FROM (SELECT CITY,LENGTH(CITY) FROM STATION WHERE LENGTH(CITY)=(SELECT MIN(LENGTH(CITY)) FROM STATION)

ORDER BY CITY ASC) WHERE ROWNUM<2

UNION

SELECT \* FROM (SELECT CITY,LENGTH(CITY) FROM STATION WHERE LENGTH(CITY)=(SELECT MAX(LENGTH(CITY)) FROM STATION)

ORDER BY CITY ASC) WHERE ROWNUM<2 ;

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.

select distinct city from station where LOWER(SUBSTR(CITY,1,1)) IN('a','e','i','o','u') and LOWER(SUBSTR(CITY,LENGTH(CITY),1)) IN('a','e','i','o','u');

or can also use LEFT and RIGHT functions.

Write a query identifying the *type* of each record in the **TRIANGLES** table using its three side lengths. Output one of the following statements for each record in the table:

* **Equilateral**: It's a triangle with  sides of equal length.
* **Isosceles**: It's a triangle with  sides of equal length.
* **Scalene**: It's a triangle with  sides of differing lengths.
* **Not A Triangle**: The given values of *A*, *B*, and *C* don't form a triangle.
* **Input Format**
* The **TRIANGLES** table is described as follows:



* Each row in the table denotes the lengths of each of a triangle's three sides.
* **Sample Input**



**Sample Output**

Isosceles

Equilateral

Scalene

Not A Triangle

Op: SELECT CASE

WHEN A + B <= C OR A + C <= B OR B + C <= A THEN 'Not A Triangle'

WHEN A = B AND B = C THEN 'Equilateral'

WHEN A = B OR B = C OR A = C THEN 'Isosceles'

ELSE 'Scalene'

END

FROM TRIANGLES;

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.





SELECT c.NAME FROM CITY c INNER JOIN COUNTRY co

ON co.code=c.countrycode WHERE co.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. (Same tables as above).

Op:

select co.CONTINENT ,

FLOOR(AVG(c.POPULATION))

from

city c inner join country co on c.countrycode =co.code GROUP BY co.CONTINENT;

*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)*.

Op: SELECT SYS\_CONNECT\_BY\_PATH(NULL, '\* ') AS a FROM DUAL CONNECT BY ROWNUM <= 20 ORDER BY a DESC;

*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)*.

select sys\_connect\_by\_path(NULL,'\* ') as a from dual connect by rownum<=20 order by a ;