HackerRank SQL Problem Solving Questions With Solutions

Updated 5 months ago · 54 min read

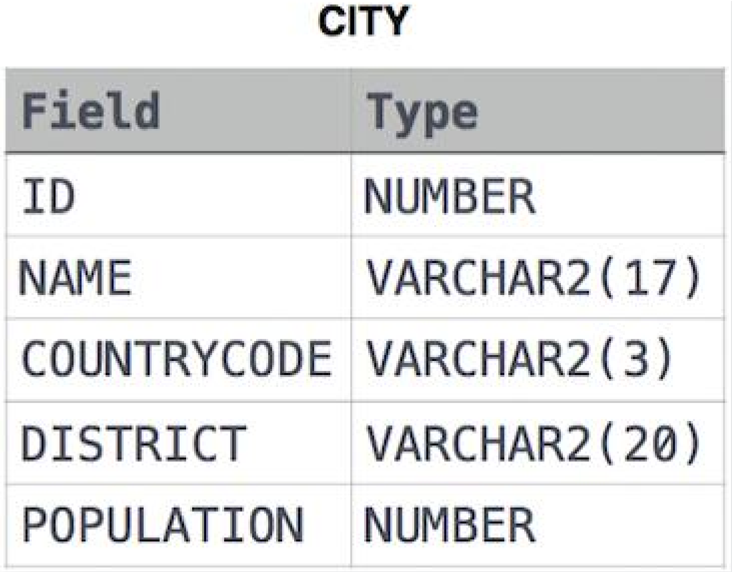
#SQL #SQL Problem Solving #HackerRank

▶ Table of contents

1. Revising the Select Query I | Easy | HackerRank

Query all columns for all American cities in the **CITY** table with populations larger than 100000. The **CountryCode** for America is USA.

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



CITY

Field	Type
ID	NUMBER
NAME	VARCHAR2(17)
COUNTRYCODE	VARCHAR2(3)
DISTRICT	VARCHAR2(20)
POPULATION	NUMBER

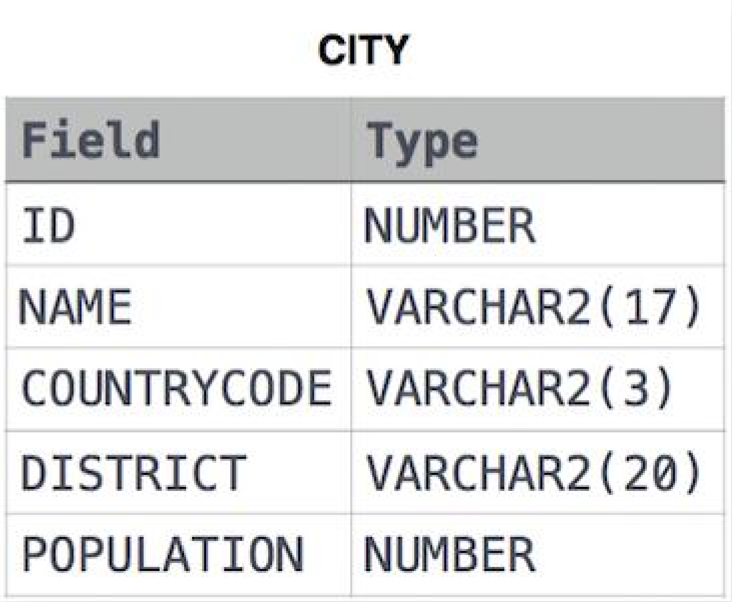
Solution

```sql
SELECT * FROM CITY
WHERE COUNTRYCODE='USA' AND POPULATION > 100000;
```

2. Revising the Select Query II | Easy | HackerRank

Query the NAME field for all American cities in the CITY table with populations larger than 120000. The CountryCode for America is USA.

The CITY table is described as follows:



CITY

Field	Type
ID	NUMBER
NAME	VARCHAR2(17)
COUNTRYCODE	VARCHAR2(3)
DISTRICT	VARCHAR2(20)
POPULATION	NUMBER

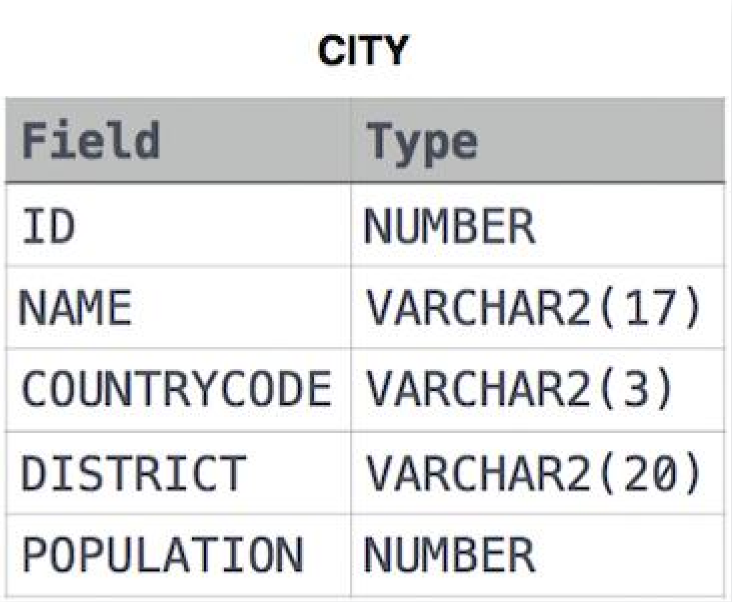
Solution

```sql
SELECT NAME FROM CITY
WHERE COUNTRYCODE ='USA' AND POPULATION > 120000;
```

3. Select All | Easy | HackerRank

Query all columns (attributes) for every row in the CITY table.

The CITY table is described as follows:



CITY

Field	Type
ID	NUMBER
NAME	VARCHAR2(17)
COUNTRYCODE	VARCHAR2(3)
DISTRICT	VARCHAR2(20)
POPULATION	NUMBER

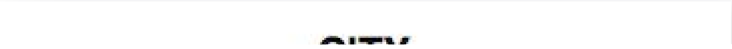
Solution

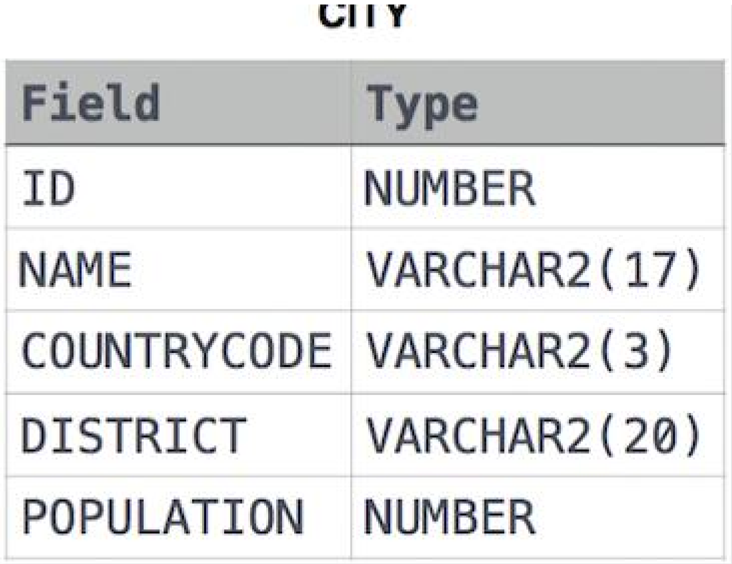
```sql
SELECT * FROM CITY;
```

4. Select By ID | Easy | HackerRank

Query all columns for a city in CITY with the ID 1661.

The CITY table is described as follows:





CITY

Field	Type
ID	NUMBER
NAME	VARCHAR2(17)
COUNTRYCODE	VARCHAR2(3)
DISTRICT	VARCHAR2(20)
POPULATION	NUMBER

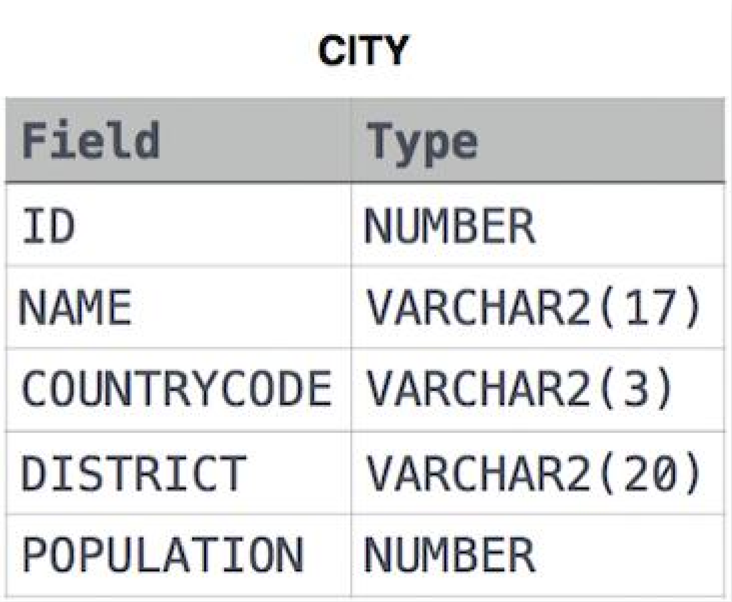
Solution

```sql
SELECT * FROM CITY WHERE ID = 1661;
```

5. Japanese Cities' Attributes | Easy | HackerRank

Query all attributes of every Japanese city in the **CITY** table. The **COUNTRYCODE** for Japan is JPN.

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



CITY

Field	Type
ID	NUMBER
NAME	VARCHAR2(17)
COUNTRYCODE	VARCHAR2(3)
DISTRICT	VARCHAR2(20)
POPULATION	NUMBER

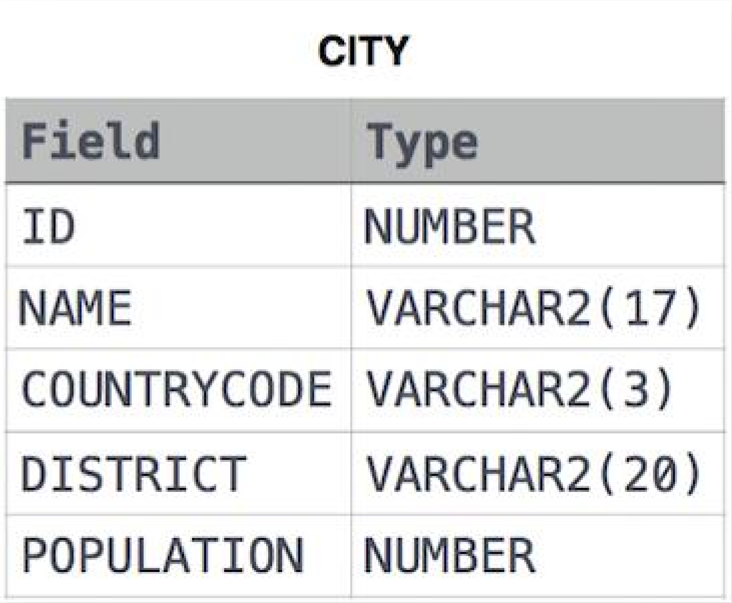
Solution

```sql
SELECT * FROM CITY
WHERE COUNTRYCODE='JPN';
```

6. Japanese Cities' Names | Easy | HackerRank

Query the names of all the Japanese cities in the **CITY** table. The **COUNTRYCODE** for Japan is JPN.

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



CITY

Field	Type
ID	NUMBER
NAME	VARCHAR2(17)
COUNTRYCODE	VARCHAR2(3)
DISTRICT	VARCHAR2(20)
POPULATION	NUMBER

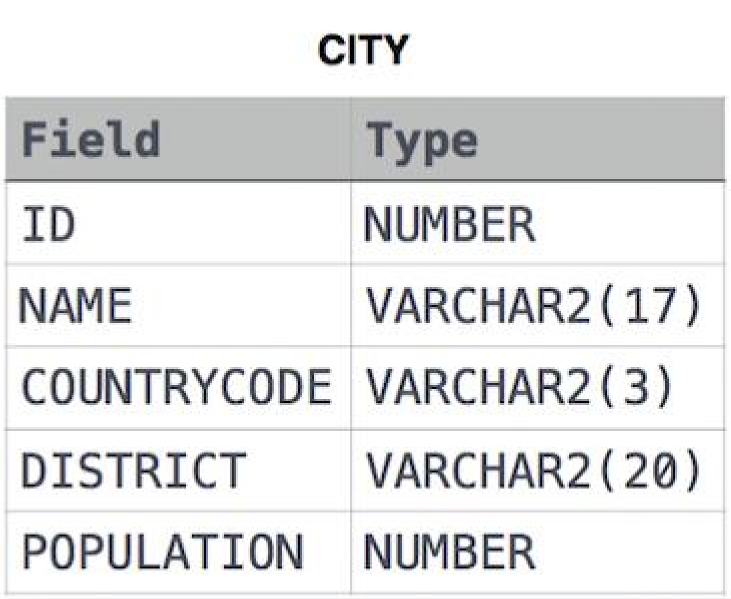
Solution

```sql
SELECT NAME FROM CITY
WHERE COUNTRYCODE ='JPN';
```

7. Average Population | Easy | HackerRank

Query the average population for all cities in **CITY**, rounded down to the nearest integer.

Input Format
The **CITY** table is described as follows:



CITY

Field	Type
ID	NUMBER
NAME	VARCHAR2(17)
COUNTRYCODE	VARCHAR2(3)
DISTRICT	VARCHAR2(20)
POPULATION	NUMBER

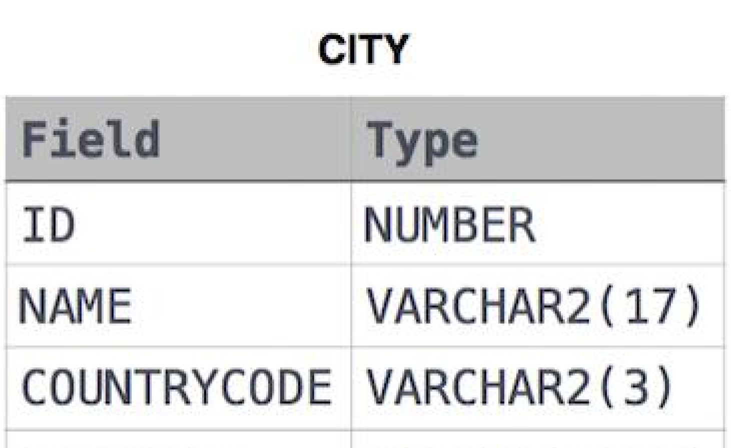
Solution

```sql
SELECT ROUND(AVG(POPULATION))
FROM CITY;
```

8. Japan Population | Easy | HackerRank

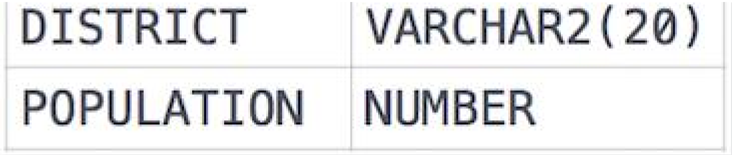
Query the sum of the populations for all Japanese cities in **CITY**. The COUNTRYCODE for Japan is **JPN**.

Input Format
The **CITY** table is described as follows:



CITY

Field	Type
ID	NUMBER
NAME	VARCHAR2(17)
COUNTRYCODE	VARCHAR2(3)



DISTRICT	VARCHAR2(20)
POPULATION	NUMBER

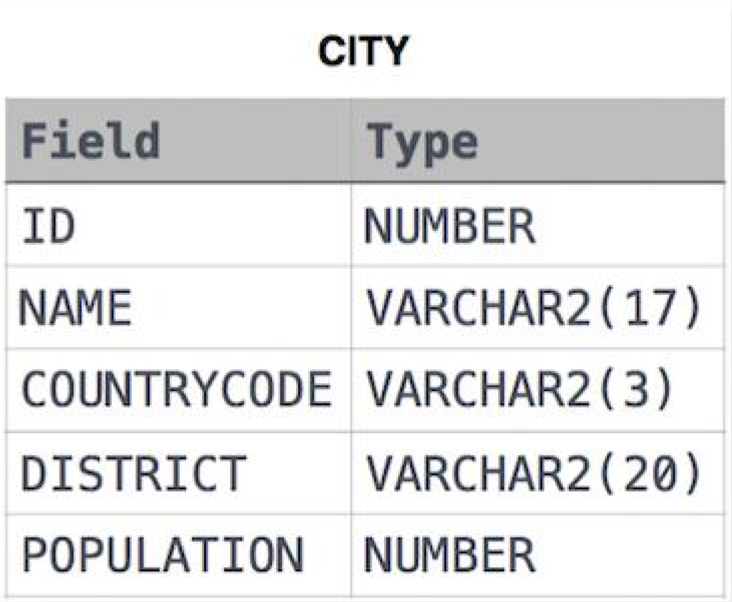
Solution

```sql
SELECT SUM(POPULATION)
FROM CITY
WHERE COUNTRYCODE = 'JPN';
```

9. Revising Aggregations - The Count Function | Easy | HackerRank

Query a count of the number of cities in **CITY** having a Population larger than **100,000**.

Input Format
The **CITY** table is described as follows:



CITY

Field	Type
ID	NUMBER
NAME	VARCHAR2(17)
COUNTRYCODE	VARCHAR2(3)
DISTRICT	VARCHAR2(20)
POPULATION	NUMBER

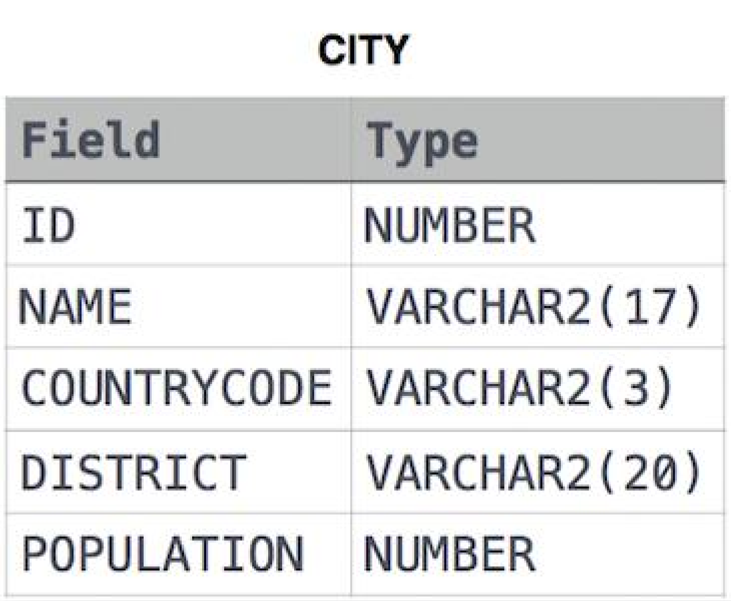
Solution

```sql
SELECT COUNT(ID) FROM CITY
WHERE POPULATION > 100000;
```

10. Revising Aggregations - The Sum Function | Easy | [HackerRank]

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

Input Format
The **CITY** table is described as follows:



CITY

Field	Type
ID	NUMBER
NAME	VARCHAR2(17)
COUNTRYCODE	VARCHAR2(3)
DISTRICT	VARCHAR2(20)
POPULATION	NUMBER

Solution

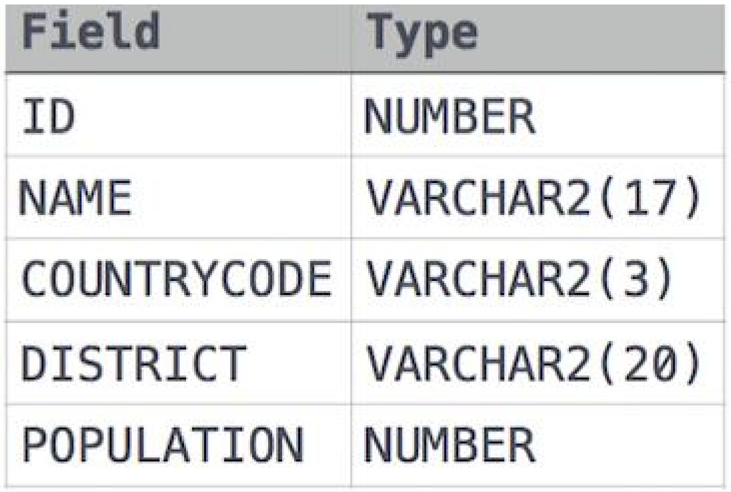
```sql
SELECT SUM(POPULATION) FROM CITY
WHERE DISTRICT = 'California';
```

11. Revising Aggregations - Averages | Easy | [HackerRank]

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

Input Format
The **CITY** table is described as follows:

CITY



Field	Type
ID	NUMBER
NAME	VARCHAR2(17)
COUNTRYCODE	VARCHAR2(3)
DISTRICT	VARCHAR2(20)
POPULATION	NUMBER

Solution

```sql
SELECT AVG(POPULATION) FROM CITY
WHERE DISTRICT = 'California';
```

12. Population Density Difference | Easy | HackerRank

Query the difference between the maximum and minimum populations in CITY.

Input Format
The **CITY** table is described as follows:

Solution

```sql
SELECT MAX(POPULATION) - MIN(POPULATION)
FROM CITY;
```

13. African Cities | Easy | HackerRank

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:

Solution

SQL

```sql
SELECT ci.Name
FROM CITY ci
JOIN COUNTRY co
ON co.code = ci.countrycode
WHERE CONTINENT ='Africa'
```

14. Asian Population | Easy | HackerRank

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:

Solution

SQL

```sql
SELECT SUM(ci.POPULATION)
FROM CITY AS ci
JOIN COUNTRY AS co
ON ci.COUNTRYCODE=co.CODE
WHERE co.CONTINENT='Asia';
```

15. Average Population of Each Continent | Easy | HackerRank

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:

Solution

```sql
SELECT co.continent, FLOOR(AVG(ci.population))
FROM CITY ci
JOIN COUNTRY co
ON co.code = ci.countrycode
GROUP BY co.continent;
```

16. Weather Observation Station 1 | Easy | HackerRank

Query a list of **CITY** and **STATE** from the **STATION** table. The **STATION** table is described as follows:

where **LAT_N** is the northern latitude and **LONG_W** is the western longitude.

Solution

```SQL
SELECT CITY, STATE FROM STATION;
```

17. Weather Observation Station 2 | Easy | HackerRank

Query the following two values from the STATION table:

1. The sum of all values in LAT_N rounded to a scale of **2** decimal places.
2. The sum of all values in LONG_W rounded to a scale of **2** decimal places.

Input Format

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

where LATN *is the northern latitude and LONG*W is the western longitude.

Output Format

Your results must be in the form:

```TEXT
lat lon
```

where ***lat*** is the sum of all values in LATN *and **lon** is the sum of all values in LONG*W. Both results must be rounded to a scale of **2** decimal places.

Solution

```SQL
SELECT ROUND(SUM(LAT_N), 2), ROUND(SUM(LONG_W), 2)
FROM STATION;
```

18. Weather Observation Station 3 | Easy | HackerRank

Query a list of **CITY** names from **STATION** for cities that have an even **ID** number. Print the results in any order, but exclude duplicates from the answer. The **STATION** table is described as follows

where **LAT_N** is the northern latitude and **LONG_W** is the western longitude.

Solution

```sql
#Solution 1:
SELECT DISTINCT CITY FROM STATION
WHERE ID % 2 = 0;

#Solution 2:
SELECT DISTINCT CITY FROM STATION
WHERE MOD(ID, 2) = 0;
```

19. Weather Observation Station 4 | Easy | HackerRank

Find the difference between the total number of **CITY** entries in the table and the number of distinct **CITY** entries in the table. The **STATION** table is described as follows:

where **LAT_N** is the northern latitude and **LONG_W** is the western longitude.

For example, if there are three records in the table with **CITY** values 'New York', 'New York', 'Bengalaru', there are 2 different city names: 'New York' and 'Bengalaru'. The query returns 1, because

`total number of records - number of unique city names = 3 - 2 = 1`

Solution

```sql
SELECT COUNT(CITY)- COUNT(DISTINCT CITY)
FROM STATION;
```

20. Weather Observation Station 5 | Easy | HackerRank

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

```text
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.

Solution

```sql
SELECT CITY, LENGTH(CITY) FROM STATION ORDER BY LENGTH(CITY), CITY LIMIT 1;
SELECT CITY, LENGTH(CITY) FROM STATION ORDER BY LENGTH(CITY) DESC, CITY LIMIT 1;
```

21. Weather Observation Station 6 | Easy | HackerRank

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.

Solution

```sql
#Solution 1:
SELECT CITY FROM STATION
WHERE LEFT(UPPER(CITY),1) IN ('A','E','I','O','U');

#Solution 2:
SELECT DISTINCT CITY FROM STATION
WHERE CITY REGEXP '^[aeiou]';
```

22. Weather Observation Station 7 | Easy | HackerRank

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 LATN *is the northern latitude and LONG*W is the western longitude.

Solution

```sql
#Solution 1:
SELECT DISTINCT CITY FROM STATION
WHERE RIGHT(UPPER(CITY),1) IN('A','E','I','O','U');

#Solution 2:
SELECT DISTINCT CITY FROM STATION
WHERE CITY REGEXP '[aeiou]$';
```

23. Weather Observation Station 8 | Easy | HackerRank

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 LATN *is the northern latitude and LONG*W is the western longitude.

Solution

```sql
#Solution 1:
SELECT DISTINCT CITY
FROM STATION
WHERE LEFT(UPPER(CITY),1) IN('A','E','I','O','U') AND
RIGHT(UPPER(CITY),1) IN('A','E','I','O','U');

#Solution 2:
SELECT DISTINCT CITY FROM STATION
WHERE CITY REGEXP '^[aeiou].*[aeiou]$';
```

24. Weather Observation Station 9 | Easy | HackerRank

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 LATN *is the northern latitude and LONG*W is the western longitude.

Solution

```sql
#Solution 1:
SELECT DISTINCT CITY
FROM STATION
WHERE LEFT(UPPER(CITY),1) NOT IN('A','E','I','O','U');

#Solution 2:
SELECT DISTINCT CITY FROM STATION
WHERE CITY REGEXP '^[^aeiou]';
```

25. Weather Observation Station 10 | Easy | HackerRank

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 LATN *is the northern latitude and LONG*W is the western longitude.

Solution

```sql
#Solution 1:
SELECT DISTINCT CITY
FROM STATION
WHERE RIGHT(UPPER(CITY),1) NOT IN('A','E','I','O','U');

#Solution 2:

SELECT DISTINCT CITY FROM STATION
WHERE CITY REGEXP '[^aeiou]$';
```

26. Weather Observation Station 11 | Easy | HackerRank

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 LATN *is the northern latitude and LONG*W is the western longitude.

Solution

```sql
#Solution 1:
SELECT DISTINCT CITY
```

```
FROM STATION
WHERE LEFT(UPPER(CITY),1) NOT IN('A','E','I','O','U')
OR RIGHT(UPPER(CITY),1) NOT IN('A','E','I','O','U');

#Solution 2:
SELECT DISTINCT CITY FROM STATION
WHERE CITY REGEXP '^[^aeiou]|[^aeiou]$';
```

27. Weather Observation Station 12 | Easy | HackerRank

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.

Solution

```sql
#Solution 1:
SELECT DISTINCT CITY
FROM STATION
WHERE LEFT(UPPER(CITY),1) NOT IN('A','E','I','O','U')
AND RIGHT(UPPER(CITY),1) NOT IN('A','E','I','O','U');
```

```
#Solution 2:
SELECT DISTINCT CITY FROM STATION
WHERE CITY REGEXP '^[^aeiou].*[^aeiou]$';
```

28. Weather Observation Station 13 | Easy | HackerRank

Query the sum of Northern Latitudes (LAT_N) from **STATION** having values greater than **38.7880** and less than **137.2345**. Truncate your answer to **4** decimal places.

Input Format

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

where LATN *is the northern latitude and LONG*W is the western longitude.

Solution

```sql
SELECT ROUND(SUM(LAT_N),4) AS sum_lat
FROM STATION
WHERE LAT_N > 38.7880 AND LAT_N < 137.2345;
```

29. Weather Observation Station 14 | Easy | HackerRank

Query the greatest value of the Northern Latitudes (LAT_N) from STATION that is less than **137.2345**. Truncate your answer to **4** decimal places.

Input Format

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

where LAT*N is the northern latitude and LONG*W is the western longitude.

Solution

SQL

```
SELECT ROUND(MAX(LAT_N),4) AS max_lat_n
FROM STATION
WHERE LAT_N < 137.2345;
```

30. Weather Observation Station 15 | Easy | HackerRank

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.

Input Format

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

where LAT*N is the northern latitude and LONG*W is the western longitude.

Solution

```sql
SELECT ROUND(LONG_W, 4)
FROM STATION
WHERE LAT_N < 137.2345
ORDER BY LAT_N DESC
LIMIT 1;
```

31. Weather Observation Station 16 | Easy | HackerRank

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

Input Format

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

where LAT*N is the northern latitude and LONG*W is the western longitude.

Solution

```sql
#Solution 1:
SELECT ROUND(MIN(LAT_N),4)
FROM STATION
WHERE LAT_N > 38.7780 ;

#Solution 2:
SELECT ROUND(LAT_N, 4)
FROM STATION
WHERE LAT_N > 38.7780
ORDER BY LAT_N
LIMIT 1;
```

32. Weather Observation Station 17 | Easy | HackerRank

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.

Input Format

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

where LAT*N is the northern latitude and LONG*W is the western longitude.

Solution

```sql
SELECT ROUND(LONG_W, 4)
FROM STATION
WHERE LAT_N > 38.7780
ORDER BY LAT_N
LIMIT 1;
```

33. Weather Observation Station 18 | Medium | HackerRank

Consider **P_1(a, b)** and **P_2(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 between points **P_1** and **P_2** and round it to a scale of 4 decimal places

Input Format

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

where LAT*N is the northern latitude and LONG*W is the western longitude.

Solution

```sql
SELECT ROUND((ABS(MIN(LAT_N)-MAX(LAT_N)) + ABS(MIN(LONG_W)-MAX(LONG_W))),4)
FROM STATION;
```

34. Weather Observation Station 19 | Medium | HackerRank

Consider **P$_1$(a, c)** and **P$_2$(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 between points **P$_1$** and **P$_2$** and format your answer to display **4** decimal digits.

Input Format

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

where LAT*N is the northern latitude and LONG*W is the western longitude.

Solution

```sql
SELECT ROUND(SQRT(POW(MIN(LAT_N)-MAX(LAT_N),2) + POW(MIN(LONG_W)-MAX(LONG_W),2)),
FROM STATION;
```

35. Weather Observation Station 20 | Medium | HackerRank

A 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.

Input Format

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

where LAT*N is the northern latitude and LONG*W is the western longitude.

Solution

```sql
#Solution 1:
SELECT ROUND(S1.LAT_N, 4)
```

```
FROM STATION AS S1
WHERE (SELECT ROUND(COUNT(S1.ID)/2) - 1
       FROM STATION) =
      (SELECT COUNT(S2.ID)
       FROM STATION AS S2
       WHERE S2.LAT_N > S1.LAT_N);

#Solution 2:
SELECT ROUND(MEDIAN(LAT_N),4)
FROM STATION;
```

36. Higher Than 75 Marks | Easy | HackerRank

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.

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

```text
Ashley
Julia
Belvet
```

Explanation

Only Ashley, Julia, and Belvet have Marks > **75**. If you look at the last three characters of each of their names, there are no duplicates and ‘ley’ < ‘lia’ < ‘vet’.

Solution

```sql
#Solution 1:
SELECT Name
FROM STUDENTS
WHERE MARKS > 75
ORDER BY RIGHT(NAME, 3) ASC, ID;

#Solution 2:
SELECT name FROM students
WHERE marks > 75
ORDER BY SUBSTR(name, LENGTH(name)-2, 3), id;
```

37. Employee Names | Easy | HackerRank

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

```text
Angela
Bonnie
Frank
Joe
Kimberly
Lisa
Michael
```

```
Patrick
Rose
Todd
```

Solution

```sql
SELECT name FROM Employee ORDER BY name;
```

38. Employee Salaries | Easy | HackerRank

Write a query that prints a list of employee names (i.e.: the name attribute) for employees in Employee having a salary greater than **$2000** per month who have been employees for less than **10** 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

```text
Angela
Michael
Todd
Joe
```

Explanation

Angela has been an employee for **1** month and earns **$3443** per month.

Michael has been an employee for **6** months and earns **$2017** per month.

Todd has been an employee for **5** months and earns **$3396** per month.

Joe has been an employee for **9** months and earns **$3573** per month.

We order our output by ascending employee_id.

Solution

```sql
SELECT name
FROM Employee
WHERE salary > 2000 AND months < 10
ORDER BY employee_id;
```

39. Top Earners | Easy | HackerRank

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.

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

```
69952 1
```

Explanation

The table and earnings data is depicted in the following diagram:

The maximum earnings value is **69952**. The only employee with earnings = **69952** is Kimberly, so we print the maximum earnings value (**69952**) and a count of the number of employees who have earned **$69952** (which is **1**) as two space-separated values.

Solution

```sql
SELECT (months*salary) as earnings, COUNT(*)
FROM Employee
GROUP BY earnings
ORDER BY earnings DESC
LIMIT 1;
```

40. The Blunder | Easy | HackerRank

Samantha was tasked with calculating the average monthly salaries for all employees in the **EMPLOYEES** table, but did not realize her keyboard's **0** 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.: ***actual - miscalculated*** 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

1000 < Salary < 10^5.

Sample Input

Sample Output

```text
2061
```

Explanation

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.

Solution

```sql
SELECT CEIL(AVG(Salary) - AVG(REPLACE(Salary, '0', '')))
FROM EMPLOYEES;
```

41. Type of Triangle | Easy | HackerRank

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 **3** sides of equal length.
- **Isosceles**: It's a triangle with **2** sides of equal length.
- **Scalene**: It's a triangle with **3** 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

```text
Isosceles
Equilateral
Scalene
Not A Triangle
```

Explanation

Values in the tuple **(20, 20, 23)**form an Isosceles triangle, because **A ≡ B**. Values in the tuple **(20, 20, 20)** form an Equilateral triangle, because **A ≡ B ≡ C**. Values in the tuple **(20, 21, 22)** form a Scalene triangle, because **A ≠ B ≠ C**. Values in the tuple **(13, 14, 30)** cannot form a triangle because the combined value of sides **A** and **B** is not larger than that of side **C**.

Solution

```sql
SELECT IF(A+B>C AND A+C>B AND B+C>A, IF(A=B AND B=C, 'Equilateral', IF(A=B OR B=C
FROM TRIANGLES;
```

42. The PADS | Medium | HackerRank

Generate the following two result sets:

1. Query an alphabetically ordered list of all names in **OCCUPATIONS**, immediately followed by the first letter of each profession as a parenthetical (i.e.: enclosed in parentheses). For example: AnActorName(A), ADoctorName(D), AProfessorName(P), and ASingerName(S).
2. Query the number of ocurrences of each occupation in **OCCUPATIONS**. Sort the occurrences in ascending order, and output them in the following format:

`There are a total of [occupation_count] [occupation]s.`

where [occupation_count] is the number of occurrences of an occupation in **OCCUPATIONS** and [occupation] is the lowercase occupation name. If more than one Occupation has the same [occupation_count], they should be ordered alphabetically.

Note: There will be at least two entries in the table for each type of occupation.

Input Format

The **OCCUPATIONS** table is described as follows: Occupation will only contain one of the following values: **Doctor**, **Professor**, **Singer** or **Actor**.

Sample Input

An **OCCUPATIONS** table that contains the following records:

Sample Output

TEXT

```
Ashely(P)
Christeen(P)
Jane(A)
Jenny(D)
Julia(A)
Ketty(P)
Maria(A)
Meera(S)
Priya(S)
Samantha(D)
There are a total of 2 doctors.
There are a total of 2 singers.
There are a total of 3 actors.
There are a total of 3 professors.
```

Explanation

The results of the first query are formatted to the problem description's specifications. The results of the second query are ascendingly ordered first by number of names corresponding to each profession (2 ≤ 2 ≤ 3 ≤ 3), and then alphabetically by profession (doctor ≤ singer, and actor ≤ professor).

Solution

SQL

```sql
#Solution 1:
SELECT CONCAT(NAME,CONCAT("(",CONCAT(substr(OCCUPATION,1,1),")"))) FROM OCCUPATIO
SELECT "There are a total of ", count(OCCUPATION), CONCAT(LOWER(occupation),"s.")

#Solution 2:
SELECT NAME || '(' || SUBSTR(OCCUPATION, 0, 1) || ')'
FROM OCCUPATIONS
ORDER BY NAME;

SELECT 'There are a total of ' || COUNT(*) || ' ' || LOWER(OCCUPATION) || 's.'
FROM OCCUPATIONS
GROUP BY OCCUPATION
ORDER BY COUNT(*), OCCUPATION;
```

43. The Report | Medium | HackerRank

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

```text
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)

Solution

```sql
SELECT IF(g.Grade<8, NULL, s.Name), g.Grade, s.Marks
FROM Students AS s
JOIN Grades AS g
ON s.Marks
BETWEEN g.Min_Mark AND g.Max_Mark
ORDER BY g.Grade DESC, s.Name, s.Marks;
```

44. Top Competitors | Medium | HackerRank

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

```text
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 **2** space-separated values.

Solution

```sql
SELECT h.hacker_id, h.name
FROM Submissions AS s
JOIN Hackers AS h
ON s.hacker_id = h.hacker_id
JOIN Challenges AS c
ON s.challenge_id = c.challenge_id
JOIN Difficulty AS d
ON c.difficulty_level = d.difficulty_level
WHERE s.score = d.score
GROUP BY h.hacker_id, h.name
HAVING COUNT(*)>1
ORDER BY COUNT(*) DESC, h.hacker_id;
```

45. Challenges | Medium | HackerRank

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

```TEXT
21283 Angela 6
88255 Patrick 5
96196 Lisa 1
```

Sample Input 1

Hackers Table:

Challenges Table:

Sample Output 1

TEXT

```
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 **5077** and **62743** both created **4** challenges, but the maximum number of challenges created is **6** so these students are excluded from the result.

For Sample Case 1, we can get the following details:

Students **12299** and **34856** both created **6** challenges. Because **6** is the maximum number of challenges created, these students are included in the result.

Solution

```sql
SELECT c.hacker_id, h.name, COUNT(c.challenge_id) AS cnt
FROM Hackers AS h JOIN Challenges AS c ON h.hacker_id = c.hacker_id
GROUP BY c.hacker_id, h.name HAVING
cnt = (SELECT COUNT(c1.challenge_id) FROM Challenges AS c1 GROUP BY c1.hacker_id (
cnt NOT IN (SELECT COUNT(c2.challenge_id) FROM Challenges AS c2 GROUP BY c2.hacke
ORDER BY cnt DESC, c.hacker_id;
```

46. Contest Leaderboard | Medium | HackerRank

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 **0** 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

```text
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 = **95 + max(43, 96) = 191**.

Hacker 74842 submitted solutions for challenges 19797 and 63132, so the total score = **max(98, 5) + 76 = 174**.

Hacker 84072 submitted solutions for challenges 49593 and 63132, so the total score = **100 + 0 = 100**.

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

Solution

```sql
SELECT m.hacker_id, h.name, SUM(m.score) AS total_score FROM
(SELECT hacker_id, challenge_id, MAX(score) AS score FROM Submissions GROUP BY ha
JOIN Hackers AS h ON m.hacker_id = h.hacker_id
GROUP By m.hacker_id, h.name
HAVING total_score > 0
ORDER BY total_score DESC, m.hacker_id;
```

47. 15 Days of Learning SQL | Hard | HackerRank

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 **1** 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

```text
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 **20703**, **36396**, **53473**, and **79722** made submissions. There are **4** unique hackers who made at least one submission each day. As each hacker made one submission, **20703** 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 **15758**, **20703**, and **79722** made submissions. Now **20703** and **79722** were the only ones to submit every day, so there are **2** unique hackers who made at least one submission each day. **79722** made **2** submissions, and name of the hacker is Michael.

On March 03, 2016 hackers **20703**, **36396**, and **79722** made submissions. Now **20703** and **79722** were the only ones, so there are **2** unique hackers who made at least one submission each day. As each hacker made one submission so **20703** 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 **20703**, **44065**, **53473**, and **79722** made submissions. Now **20703** and **79722** only submitted each day, so there are unique **2** hackers who made at least one submission each day. As each hacker made one submission so **20703** 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 **20703**, **36396**, **38289** and **62529** made submissions. Now **20703** only submitted each day, so there is only **1** unique hacker who made at least one submission each day. **36396** made **2** submissions and name of the hacker is Frank.

On March 06, 2016 only **20703** made submission, so there is only **1** unique hacker who made at least one submission each day. **20703** made **1** submission and name of the hacker is Angela.

Solution

```sql
SELECT SUBMISSION_DATE,
(SELECT COUNT(DISTINCT HACKER_ID)
 FROM SUBMISSIONS S2
 WHERE S2.SUBMISSION_DATE = S1.SUBMISSION_DATE AND
(SELECT COUNT(DISTINCT S3.SUBMISSION_DATE)
 FROM SUBMISSIONS S3 WHERE S3.HACKER_ID = S2.HACKER_ID AND S3.SUBMISSION_DATE < S
(SELECT HACKER_ID FROM SUBMISSIONS S2 WHERE S2.SUBMISSION_DATE = S1.SUBMISSION_DA
GROUP BY HACKER_ID ORDER BY COUNT(SUBMISSION_ID) DESC, HACKER_ID LIMIT 1) AS TMP,
(SELECT NAME FROM HACKERS WHERE HACKER_ID = TMP)
FROM
(SELECT DISTINCT SUBMISSION_DATE FROM SUBMISSIONS) S1
GROUP BY SUBMISSION_DATE;
```

48. Binary Tree Nodes | Medium | HackerRank

You are given a table, BST, containing two columns: N and P, where N represents the value of a node in Binary Tree, and P is the parent of N.

Write a query to find the node type of Binary Tree ordered by the value of the node. Output one of the following for each node:

- Root: If node is root node.
- Leaf: If node is leaf node.
- Inner: If node is neither root nor leaf node.

Sample Input

Sample Output

```text
1 Leaf
2 Inner
3 Leaf
5 Root
6 Leaf
8 Inner
9 Leaf
```

Explanation

The Binary Tree below illustrates the sample:

Solution

```sql
#Solution 1:
SELECT N,
IF(P IS NULL, 'Root', IF((SELECT COUNT(*) FROM BST WHERE P=B.N)>0, 'Inner', 'Leaf
FROM BST AS B ORDER BY N;

#Solution 2:
SELECT N,
IF(P IS NULL, 'Root', IF(B.N IN (SELECT P FROM BST), 'Inner', 'Leaf'))
FROM BST AS B ORDER BY N;
```

49. New Companies | Medium | HackerRank

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 C1, *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

TEXT

```
C1 Monika 1 2 1 2
C2 Samantha 1 1 2 2
Explanation
```

In company C1, the only lead manager is LM1. There are two senior managers, SM1 and SM2, under LM1. There is one manager, M1, under senior manager SM1. There are two employees, E1 and E2, under manager M1.

In company C2, the only lead manager is LM2. There is one senior manager, SM3, under LM2. There are two managers, M2 and M3, under senior manager SM3. There is one employee, E3, under manager M2, and another employee, E4, under manager, M3.

Solution

SQL

```sql
#Solution 1:
SELECT c.company_code, c.founder,
       COUNT(DISTINCT l.lead_manager_code), COUNT(DISTINCT s.senior_manager_code)
       COUNT(DISTINCT m.manager_code), COUNT(DISTINCT e.employee_code)
FROM Company c, Lead_Manager l, Senior_Manager s, Manager m, Employee e
WHERE c.company_code = l.company_code AND
      l.lead_manager_code = s.lead_manager_code AND
      s.senior_manager_code = m.senior_manager_code AND
      m.manager_code = e.manager_code
GROUP BY c.company_code, c.founder ORDER BY c.company_code;

#Solution 2:
SELECT c.company_code, c.founder,
       COUNT(DISTINCT l.lead_manager_code), COUNT(DISTINCT s.senior_manager_code)
       COUNT(DISTINCT m.manager_code), COUNT(DISTINCT e.employee_code)
FROM Company c JOIN Lead_Manager l ON c.company_code = l.company_code JOIN
     Senior_Manager s ON l.lead_manager_code = s.lead_manager_code JOIN
     Manager m ON s.senior_manager_code = m.senior_manager_code JOIN
     Employee e ON m.manager_code = e.manager_code
GROUP BY c.company_code, c.founder ORDER BY c.company_code;
```

50. Draw The Triangle 1 | Easy | HackerRank

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

TEXT

```
* * * * *
* * * *
* * *
* *
*
```

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

Solution

SQL

```
#Solution 1:
SET @number = 21;
SELECT REPEAT('* ', @number := @number - 1)
FROM information_schema.tables
LIMIT 20;

#Solution 2:
SET @number = 21;
SELECT REPEAT('* ', @number := @number - 1)
FROM information_schema.tables
WHERE @number > 0;
```

51. Draw The Triangle 2 | Easy | HackerRank

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

TEXT

```
*
* *
* * *
* * * *
* * * * *
```

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

Solution

SQL

```
#Solution 1:
SET @number = 0;
SELECT REPEAT('* ', @number := @number+1)
FROM information_schema.tables
LIMIT 20;

#Solution 2:
SET @number = 0;
```

```sql
SELECT REPEAT('* ', @number := @number+1)
FROM information_schema.tables
WHERE @number < 20;
```

52. Print Prime Numbers | Medium | HackerRank

Write a query to print all prime numbers less than or equal to **1000**. 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 **≤ 10** would be :

```text
2&3&5&7
```

Solution

```sql
#Solution 1: MS SQL
DECLARE @table TABLE (PrimeNumber INT)
DECLARE @final AS VARCHAR(1500)
SET @final = ''
DECLARE @counter INT
SET @counter = 2
WHILE @counter <= 1000
BEGIN
    IF NOT EXISTS (
            SELECT PrimeNumber
            FROM @table
            WHERE @counter % PrimeNumber = 0)
        BEGIN
            INSERT INTO @table SELECT @counter
            SET @final = @final + CAST(@counter AS VARCHAR(20))+'&'
        END
    SET @counter = @counter + 1
END
SELECT SUBSTRING(@final,0,LEN(@final))

#Solution 2:MySQL
```

```
SELECT GROUP_CONCAT(NUMB SEPARATOR '&')
FROM (
    SELECT @num:=@num+1 as NUMB FROM
    information_schema.tables t1,
    information_schema.tables t2,
    (SELECT @num:=1) tmp
) tempNum
WHERE NUMB<=1000 AND NOT EXISTS(
        SELECT * FROM (
            SELECT @nu:=@nu+1 as NUMA FROM
                information_schema.tables t1,
                information_schema.tables t2,
                (SELECT @nu:=1) tmp1
                LIMIT 1000
            ) tatata
        WHERE FLOOR(NUMB/NUMA)=(NUMB/NUMA) AND NUMA<NUMB AND NUMA>1
    )
```

53. Ollivander's Inventory | Medium | HackerRank

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 $\boldsymbol{(code_1, age_1)}$ and $\boldsymbol{(code_2, age_2)}$, then $\boldsymbol{code_1} \neq \boldsymbol{code_2}$ and $\boldsymbol{age_1} \neq \boldsymbol{age_2}$

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 ***wand(age = 45, power = 2) = 6020***
- The minimum number of galleons needed for ***wand(age = 45, power = 10) = 1647***

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

- The minimum number of galleons needed for ***wand(age = 40, power = 1) = 5408***
- The minimum number of galleons needed for ***wand(age = 40, power = 3) = 3312***
- The minimum number of galleons needed for ***wand(age = 40, power = 5) = 7587***
- The minimum number of galleons needed for ***wand(age = 40, power = 7) = 6018***

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

- The minimum number of galleons needed for ***wand(age = 20, power = 5) = 504***
- The minimum number of galleons needed for ***wand(age = 20, power = 6) = 7651***

- The minimum number of galleons needed for ***wand(age = 20, power = 8) = 3688***

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

- The minimum number of galleons needed for ***wand(age = 17, power = 3) = 5689***
- The minimum number of galleons needed for ***wand(age = 17, power = 10) = 9897***

Solution

```
SELECT id, age, m.coins_needed, m.power FROM
(SELECT code, power, MIN(coins_needed) AS coins_needed FROM Wands GROUP BY code,
JOIN Wands AS w ON m.code = w.code AND m.power = w.power AND m.coins_needed = w.c
JOIN Wands_Property AS p ON m.code = p.code
WHERE p.is_evil = 0
ORDER BY m.power DESC, age DESC;
```

54. Symmetric Pairs | Medium | HackerRank

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

Two pairs (X_1, Y_1) and (X_2, Y_2) are said to be symmetric pairs if X_1 = Y_2 and X_2 = Y_1.

Write a query to output all such symmetric pairs in ascending order by the value of X. List the rows such that $X_1 \leq Y_1$.

Sample Input

Sample Output

```
20 20
20 21
22 23
```

Solution

```sql
#Solution 1:
SELECT f1.X, f1.Y FROM Functions AS f1
WHERE f1.X = f1.Y AND
(SELECT COUNT(*) FROM Functions WHERE X = f1.X AND Y = f1.X) > 1
UNION
SELECT f1.X, f1.Y FROM Functions AS f1, Functions AS f2
WHERE f1.X <> f1.Y AND f1.X = f2.Y AND f1.Y = f2.X AND f1.X < f2.X
ORDER BY X;

#Solution 2:
SELECT f1.X, f1.Y FROM Functions AS f1
WHERE f1.X = f1.Y AND
(SELECT COUNT(*) FROM Functions WHERE X = f1.X AND Y = f1.X) > 1
UNION
SELECT f1.X, f1.Y FROM Functions AS f1
```

```
WHERE f1.X <> f1.Y AND EXISTS(SELECT X, Y FROM Functions WHERE f1.X = Y AND f1.Y
ORDER BY X;

#Solution 3:
SELECT f1.X, f1.Y FROM Functions AS f1
WHERE f1.X = f1.Y AND
(SELECT COUNT(*) FROM Functions WHERE X = f1.X AND Y = f1.X) > 1
UNION
SELECT f1.X, f1.Y FROM Functions AS f1
WHERE EXISTS(SELECT X, Y FROM Functions WHERE f1.X = Y AND f1.Y = X AND f1.X < X)
ORDER BY X;

#Solution 4:
(SELECT f1.X, f1.Y FROM Functions AS f1
WHERE f1.X = f1.Y GROUP BY f1.X, f1.Y HAVING COUNT(*) > 1)
UNION
(SELECT f1.X, f1.Y FROM Functions AS f1
WHERE EXISTS(SELECT X, Y FROM Functions WHERE f1.X = Y AND f1.Y = X AND f1.X < X)
ORDER BY X;
```

55. Interviews | Hard | HackerRank

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. ![](../../images/articles/sql-questions/hackerrank-sql-interviews*5.png)

Contests Table:

Colleges Table:

Challenges Table:

View_Stats Table:

Submission_Stats Table:

Sample Output

```text
66406 17973 Rose 111 39 156 56
66556 79153 Angela 0 0 11 10
94828 80275 Frank 150 38 41 15
```

Explanation

The contest **66406** is used in the college **11219**. In this college **11219**, challenges **18765** and **47127** are asked, so from the view and submission stats:

- Sum of total submissions = **27 + 56 + 28 = 111**
- Sum of total accepted submissions = **10 + 18 + 11 = 39**
- Sum of total views = **43 + 72 + 26 + 15 = 156**
- Sum of total unique views = **10 + 13 + 19 + 14 = 56**

Similarly, we can find the sums for contests **66556** and **94828**.

Solution

```sql
SELECT con.contest_id, con.hacker_id, con.name,
SUM(sg.total_submissions), SUM(sg.total_accepted_submissions),
SUM(vg.total_views), SUM(vg.total_unique_views)
FROM Contests AS con
JOIN Colleges AS col ON con.contest_id = col.contest_id
JOIN Challenges AS cha ON cha.college_id = col.college_id
LEFT JOIN
(SELECT ss.challenge_id, SUM(ss.total_submissions) AS total_submissions, SUM(ss.t
ON cha.challenge_id = sg.challenge_id
LEFT JOIN
(SELECT vs.challenge_id, SUM(vs.total_views) AS total_views, SUM(vs.total_unique_
FROM View_Stats AS vs GROUP BY vs.challenge_id) AS vg
ON cha.challenge_id = vg.challenge_id
GROUP BY con.contest_id, con.hacker_id, con.name
HAVING SUM(sg.total_submissions) +
       SUM(sg.total_accepted_submissions) +
       SUM(vg.total_views) +
       SUM(vg.total_unique_views) > 0
ORDER BY con.contest_id;
```

56. SQL Project Planning | Medium | HackerRank

You are given a table, Projects, containing three columns: Task*ID, Start*Date and End*Date. It is guaranteed that the difference between the End*Date and the Start_Date is equal to 1 day for each row in the table.

If the End_Date of the tasks are consecutive, then they are part of the same project. Samantha is interested in finding the total number of different projects completed.

Write a query to output the start and end dates of projects listed by the number of days it took to complete the project in ascending order. If there is more than one project that have the same number of completion days, then order by the start date of the project.

Sample Input

Sample Output

```text
2015-10-28 2015-10-29
2015-10-30 2015-10-31
2015-10-13 2015-10-15
2015-10-01 2015-10-04
```

Explanation

The example describes following four projects:

- Project 1: Tasks 1, 2 and 3 are completed on consecutive days, so these are part of the project. Thus start date of project is 2015-10-01 and end date is 2015-10-04, so it took 3 days to complete the project.
- Project 2: Tasks 4 and 5 are completed on consecutive days, so these are part of the project. Thus, the start date of project is 2015-10-13 and end date is 2015-10-15, so it took 2 days to complete the project.

- Project 3: Only task 6 is part of the project. Thus, the start date of project is 2015-10-28 and end date is 2015-10-29, so it took 1 day to complete the project.
- Project 4: Only task 7 is part of the project. Thus, the start date of project is 2015-10-30 and end date is 2015-10-31, so it took 1 day to complete the project.

Solution

```sql
SELECT Start_Date, MIN(End_Date) FROM
(SELECT Start_Date FROM Projects WHERE Start_Date NOT IN (SELECT End_Date FROM Pr
(SELECT End_Date FROM Projects WHERE End_Date NOT IN (SELECT Start_Date FROM Proj
WHERE Start_Date < End_Date
GROUP BY Start_Date
ORDER BY DATEDIFF(MIN(End_Date), Start_Date), Start_Date;
```

57. Placements | Medium | HackerRank

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

Sample Output

```text
Samantha
Julia
Scarlet
```

Explanation

See the following table:

Now,

- Samantha’s best friend got offered a higher salary than her at 11.55
- Julia’s best friend got offered a higher salary than her at 12.12
- Scarlet’s best friend got offered a higher salary than her at 15.2
- Ashley’s best friend did NOT get offered a higher salary than her

The name output, when ordered by the salary offered to their friends, will be:

- Samantha
- Julia
- Scarlet

Solution

```sql
SELECT s.Name FROM Students AS s
JOIN Packages AS sp ON s.ID = sp.ID
JOIN Friends AS f ON s.ID = f.ID
JOIN Packages AS fp ON f.Friend_ID = fp.ID
WHERE sp.Salary < fp.Salary
ORDER BY fp.Salary;
```

58. Occupations | Medium | HackerRank

Pivot the Occupation column in **OCCUPATIONS** so that each Name is sorted alphabetically and displayed underneath its corresponding Occupation. The output column headers should be Doctor, Professor, Singer, and Actor, respectively.

Note: Print **NULL** when there are no more names corresponding to an occupation.

Input Format

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

Occupation will only contain one of the following values: **Doctor**, **Professor**, **Singer** or **Actor**.

Sample Input

Sample Output

```text
Jenny    Ashley     Meera  Jane
Samantha Christeen  Priya  Julia
NULL     Ketty      NULL   Maria
```

Explanation

The first column is an alphabetically ordered list of Doctor names. The second column is an alphabetically ordered list of Professor names. The third column is an alphabetically ordered list of Singer names. The fourth column is an alphabetically ordered list of Actor names. The empty cell data for columns with less than the maximum number of names per occupation (in this case, the Professor and Actor columns) are filled with **NULL** values.

Solution

```sql
SET @r1=0, @r2=0, @r3 =0, @r4=0;
SELECT MIN(Doctor), MIN(Professor), MIN(Singer), MIN(Actor) FROM
(SELECT CASE Occupation WHEN 'Doctor' THEN @r1:=@r1+1
                       WHEN 'Professor' THEN @r2:=@r2+1
                       WHEN 'Singer' THEN @r3:=@r3+1
                       WHEN 'Actor' THEN @r4:=@r4+1 END
       AS RowLine,
       CASE WHEN Occupation = 'Doctor' THEN Name END AS Doctor,
       CASE WHEN Occupation = 'Professor' THEN Name END AS Professor,
       CASE WHEN Occupation = 'Singer' THEN Name END AS Singer,
       CASE WHEN Occupation = 'Actor' THEN Name END AS Actor
       FROM OCCUPATIONS ORDER BY Name) AS t
GROUP BY RowLine;
```

No comments

← JavaScript frequently asked problem solving question with solution

LeetCode SQL Problem Solving Questions With Solutions →