HACKER RANK

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

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

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

Select NAME from CITY

WHERE POPULATION > 120000 and COUNTRYCODE="USA";

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

SELECT * FROM CITY;

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

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

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

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

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

```
SELECT NAME FROM CITY
WHERE COUNTRYCODE = "JPN";
```

Query a list of **CITY** and **STATE** from the **STATION** table.

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

STATION

Field	Type
ID	NUMBER
CITY	VARCHAR2(21)
STATE	VARCHAR2(2)
LAT_N	NUMBER
LONG_W	NUMBER

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

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

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:

STATION

Field	Type
ID	NUMBER
CITY	VARCHAR2(21)
STATE	VARCHAR2(2)
LAT_N	NUMBER
LONG_W	NUMBER

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

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

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:

STATION

Field	Type
ID	NUMBER
CITY	VARCHAR2(21)
STATE	VARCHAR2(2)
LAT_N	NUMBER
LONG_W	NUMBER

```
select count(*)-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.

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

STATION

Field	Type
ID	NUMBER
CITY	VARCHAR2(21)
STATE	VARCHAR2(2)
LAT_N	NUMBER
LONG_W	NUMBER

```
select city,length(city) from station

where length(city)=(select MIN(length(city))

                from station)

                order by CITY

                limit 1;



select city,length(city) from station

where length(city)=(select MAX(length(city))

                from station)

                order by CITY

                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:

STATION

Field	Type
ID	NUMBER
CITY	VARCHAR2(21)
STATE	VARCHAR2(2)
LAT_N	NUMBER
LONG_W	NUMBER

```
select DISTINCT(CITY) from STATION

WHERE CITY REGEXP '^[aeiouAEIOU]'

ORDER BY CITY;
```

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:

STATION

Field	Type
ID	NUMBER
CITY	VARCHAR2(21)
STATE	VARCHAR2(2)
LAT_N	NUMBER
LONG_W	NUMBER

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

```
SELECT DISTINCT(CITY) FROM STATION

WHERE CITY REGEXP '[aeiouAEIOU]$';
```

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:

STATION

Field	Type
ID	NUMBER
CITY	VARCHAR2(21)
STATE	VARCHAR2(2)
LAT_N	NUMBER
LONG_W	NUMBER

```
SELECT DISTINCT(CITY)

FROM STATION

WHERE CITY REGEXP '^[aeiouAEIOU].*[aeiouAEIOU]$';
```

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:

STATION

Field	Type
ID	NUMBER
CITY	VARCHAR2(21)
STATE	VARCHAR2(2)
LAT_N	NUMBER
LONG_W	NUMBER

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

```
SELECT DISTINCT(CITY)

FROM STATION

WHERE CITY NOT REGEXP '^[AEIOUaeiou]';
```

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:

STATION

Field	Type
ID	NUMBER
CITY	VARCHAR2(21)
STATE	VARCHAR2(2)
LAT_N	NUMBER
LONG_W	NUMBER

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

```
SELECT DISTINCT(CITY) FROM STATION

WHERE CITY NOT REGEXP '[aieouAEIOU]$';
```

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:

STATION

Field	Type
ID	NUMBER
CITY	VARCHAR2(21)
STATE	VARCHAR2(2)
LAT_N	NUMBER
LONG_W	NUMBER

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

```
SELECT DISTINCT(CITY) from station

where city not regexp '^[aeiouAEIOU].*[aeiouAEIOU]$';
```

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:

STATION

Field	Type
ID	NUMBER
CITY	VARCHAR2(21)
STATE	VARCHAR2(2)
LAT_N	NUMBER
LONG_W	NUMBER

```
SELECT DISTINCT(CITY) from STATION

WHERE CITY REGEXP '^[^aeiouAEIOU].*[^aeiouAEIOU]$';
```

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

Column	Type
ID	Integer
Name	String
Marks	Integer

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

```
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:

Column	Type
employee_id	Integer
name	String
months	Integer
salary	Integer

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.

```
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 $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:

Column	Type
employee_id	Integer
name	String
months	Integer
salary	Integer

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.

```
Select name from Employee

where salary > 2000 and months < 10

order by employee_id;
```

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:

Column	Type
A	Integer
B	Integer
C	Integer

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

```
SELECT CASE WHEN (A = B AND B = C) OR (A = C AND C = B) THEN 'Equilateral'

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

    WHEN (A = B AND B != C) OR (A = C AND C != B) THEN 'Isosceles'

    WHEN (A != B AND B != C) OR (A != C AND C != B) THEN 'Scalene'

  END

FROM triangles;
```

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:
3. 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:

Column	Type
Name	String
Occupation	String

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

```
SELECT CONCAT(Name,CONCAT("(",CONCAT(SUBSTR(OCCUPATION,1,1),")")))

FROM OCCUPATIONS

ORDER BY NAME;



SELECT "There are a total of ",count(OCCUPATION),concat(lower(occupation),"s.")

FROM OCCUPATIONS

GROUP BY OCCUPATION

ORDER BY COUNT(OCCUPATION);
```

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:

Column	Type
Name	String
Occupation	String

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

Sample Input

Name	Occupation
Samantha	Doctor
Julia	Actor
Maria	Actor
Meera	Singer
Ashely	Professor
Ketty	Professor
Christeen	Professor
Jane	Actor
Jenny	Doctor
Priya	Singer

Sample Output

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

```
WITH temp AS (
  SELECT
    CASE WHEN Occupation='Doctor' THEN Name END AS doctor,
    CASE WHEN Occupation='Actor' THEN Name END AS actor,
    CASE WHEN Occupation='Singer' THEN Name END AS singer,
    CASE WHEN Occupation='Professor' THEN Name END AS professor,
```

```
      ROW_NUMBER() OVER(PARTITION BY Occupation ORDER BY Name) as rn
   FROM OCCUPATIONS
)
SELECT MAX(doctor),MAX(professor),MAX(singer),MAX(actor)
FROM temp
GROUP BY rn;
```
