Weather Observation Station 5

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.

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.

Sample Input

Let's say that *CITY* only has four entries: *DEF*, *ABC*, *PQRS* and *WXY*

Sample Output

```
ABC 3
PQRS 4
```

Explanation

When ordered alphabetically, the *CITY* names are listed as *ABC, DEF, PQRS,* and *WXY*, with the respective lengths $3, 3, 4,$ and 3. The longest-named city is obviously *PQRS*, but there are 3 options for shortest-named city; we 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.

Weather Observation Station 6

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

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

Weather Observation Station 10

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.

The Blunder

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 zeroes 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:

Column	Type
ID	Integer
Name	String
Salary	Integer

Note: *Salary* is measured in dollars per month and its value is $< 10^5$.

Sample Input

ID	Name	Salary
1	Kristeen	1420
2	Ashley	2006
3	Julia	2210
4	Maria	3000

Sample Output

```
2061
```

Explanation

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

ID	Name	Salary
1	Kristeen	142
2	Ashley	26
3	Julia	221
4	Maria	3

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

The resulting error between the two calculations is $2159.00 - 98.00 = 2061.00$ which, when rounded to the next integer, is 2061.

Type of Triangle

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:

Column	Type
A	Integer
B	Integer
C	Integer

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

Sample Input

A	B	C
20	20	23
20	20	20
20	21	22
13	14	30

Sample Output

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

Explanation

Values in the tuple $(20, 20, 23)$ form an Isosceles triangle, because $A \equiv B$.
Values in the tuple $(20, 20, 20)$ form an Equilateral triangle, because $A \equiv B \equiv C$. Values in the tuple $(20, 21, 22)$ form a Scalene triangle, because $A \neq B \neq C$.
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.

Draw The Triangle 2

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)*.

Placements

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).

Column	Type
ID	Integer
Name	String

Students

Column	Type
ID	Integer
Friend_ID	Integer

Friends

Column	Type
ID	Integer
Salary	Float

Packages

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

ID	Name
1	Ashley
2	Samantha
3	Julia
4	Scarlet

Students

ID	Friend_ID
1	2
2	3
3	4
4	1

Friends

ID	Salary
1	15.20
2	10.06
3	11.55
4	12.12

Packages

Sample Output

```
Samantha
Julia
Scarlet
```

Explanation

See the following table:

ID	1	2	3	4
Name	Ashley	Samantha	Julia	Scarlet
Salary	15.20	10.06	11.55	12.12
Friend ID	2	3	4	1
Friend Salary	10.06	11.55	12.12	15.20

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*

The PADS

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:

Column	Type
Name	String
Occupation	String

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

Sample Input

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

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

Sample Output

```
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 ($2 \leq 2 \leq 3 \leq 3$), and then alphabetically by profession ($doctor \leq singer$, and $actor \leq professor$).

New Companies

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

Founder
⇓
Lead Manager
⇓
Senior Manager
⇓
Manager
⇓
Employee

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.

Column	Type
company_code	String
founder	String

- *Lead_Manager:* The *lead_manager_code* is the code of the lead manager, and the *company_code* is the code of the working company.

Column	Type
lead_manager_code	String
company_code	String

- *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.

Column	Type
senior_manager_code	String
lead_manager_code	String
company_code	String

- *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.

Column	Type
manager_code	String
senior_manager_code	String
lead_manager_code	String
company_code	String

- *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.

Column	Type
employee_code	String
manager_code	String
senior_manager_code	String
lead_manager_code	String
company_code	String

Sample Input

Company Table:

company_code	founder
C1	Monika
C2	Samantha

Lead_Manager Table:

lead_manager_code	company_code
LM1	C1
LM2	C2

Senior_Manager Table:

senior_manager_code	lead_manager_code	company_code
SM1	LM1	C1
SM2	LM1	C1
SM3	LM2	C2

Manager Table:

manager_code	senior_manager_code	lead_manager_code	company_code
M1	SM1	LM1	C1
M2	SM3	LM2	C2
M3	SM3	LM2	C2

Employee Table:

employee_code	manager_code	senior_manager_code	lead_manager_code	company_code
E1	M1	SM1	LM1	C1
E2	M1	SM1	LM1	C1
E3	M2	SM3	LM2	C2
E4	M3	SM3	LM2	C2

Sample Output

```
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*.

The Report

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

Column	Type
ID	Integer
Name	String
Marks	Integer

Grades contains the following data:

Grade	Min_Mark	Max_Mark
1	0	9
2	10	19
3	20	29
4	30	39
5	40	49
6	50	59
7	60	69
8	70	79
9	80	89
10	90	100

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

ID	Name	Marks
1	Julia	88
2	Samantha	68
3	Maria	99
4	Scarlet	78
5	Ashley	63
6	Jane	81

Sample Output

```
Maria 10 99
Jane 9 81
Julia 9 88
Scarlet 8 78
NULL 7 63
NULL 7 68
```

Note

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

Explanation

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

ID	*Name*	*Marks*	*Grade*
1	*Julia*	*88*	*9*
2	*Samantha*	*68*	*7*
3	*Maria*	*99*	*10*
4	*Scarlet*	*78*	*8*
5	*Ashley*	*63*	*7*
6	*Jane*	*81*	*9*

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

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

Ollivander's Inventory

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).

Column	Type
id	Integer
code	Integer
coins_needed	Integer
power	Integer

- *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, $(code_1, age_1)$ and $(code_2, age_2)$, then $code_1 \neq code_2$ and $age_1 \neq age_2$.

Column	Type
code	Integer
age	Integer
is_evil	Integer

Sample Input

Wands Table:

id	code	coins_needed	power
1	4	3688	8
2	3	9365	3
3	3	7187	10
4	3	734	8
5	1	6020	2
6	2	6773	7
7	3	9873	9
8	3	7721	7
9	1	1647	10
10	4	504	5
11	2	7587	5
12	5	9897	10
13	3	4651	8
14	2	5408	1
15	2	6018	7
16	4	7710	5
17	2	8798	7
18	2	3312	3
19	4	7651	6
20	5	5689	3

Wands_Property Table:

code	age	is_evil
1	45	0
2	40	0
3	4	1
4	20	0
5	17	0

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):

id	age	coins_needed	power
5	45	6020	2
9	45	1647	10

- 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):

id	age	coins_needed	power
14	40	5408	1
18	40	3312	3
11	40	7587	5
15	40	6018	7
17	40	8798	7
6	40	6773	7

- 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):

id	age	coins_needed	power
10	20	504	5
16	20	7710	5
19	20	7651	6
1	20	3688	8

- 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):

id	age	coins_needed	power
20	17	5689	3
12	17	9897	10

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

Binary Tree Nodes

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*.

Column	Type
N	Integer
P	Integer

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

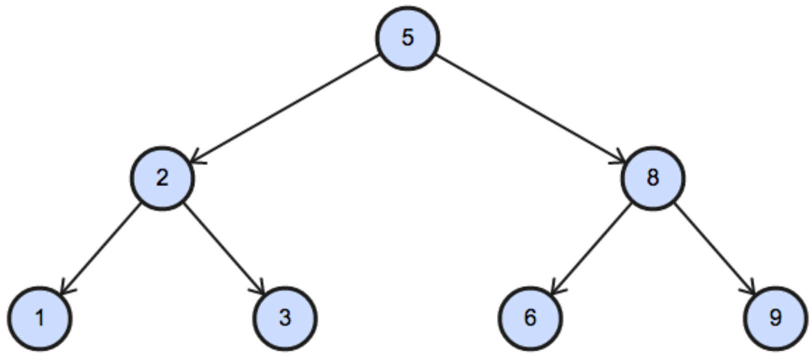
N	P
1	2
3	2
6	8
9	8
2	5
8	5
5	null

Sample Output

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

Explanation

The *Binary Tree* below illustrates the sample:


5
2
8
1
3
6
9

SQL Project Planning

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.

Column	Type
Task_ID	*Integer*
Start_Date	*Date*
End_Date	*Date*

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

Task_ID	Start_Date	End_Date
1	2015-10-01	2015-10-02
2	2015-10-02	2015-10-03
3	2015-10-03	2015-10-04
4	2015-10-13	2015-10-14
5	2015-10-14	2015-10-15
6	2015-10-28	2015-10-29
7	2015-10-30	2015-10-31

Sample Output

```
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.

Symmetric Pairs

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

Column	*Type*
X	*Integer*
Y	*Integer*

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*.

Sample Input

X	*Y*
20	*20*
20	*20*
20	*21*
23	*22*
22	*23*
21	*20*

Sample Output

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

Print Prime Numbers

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:

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

Weather Observation Station 20

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.

Note: Oracle solutions are not permitted for this challenge.

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.

15 Days of Learning SQL

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.

Column	Type
hacker_id	Integer
name	String

- *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.

Column	Type
submission_date	Date
submission_id	Integer
hacker_id	Integer
score	Integer

Sample Input

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

Hackers Table:

hacker_id	name
15758	Rose
20703	Angela
36396	Frank
38289	Patrick
44065	Lisa
53473	Kimberly
62529	Bonnie
79722	Michael

Submissions Table:

submission_date	submission_id	hacker_id	score
2016-03-01	8494	20703	0
2016-03-01	22403	53473	15
2016-03-01	23965	79722	60
2016-03-01	30173	36396	70
2016-03-02	34928	20703	0
2016-03-02	38740	15758	60
2016-03-02	42769	79722	25
2016-03-02	44364	79722	60
2016-03-03	45440	20703	0
2016-03-03	49050	36396	70
2016-03-03	50273	79722	5
2016-03-04	50344	20703	0
2016-03-04	51360	44065	90
2016-03-04	54404	53473	65
2016-03-04	61533	79722	45
2016-03-05	72852	20703	0
2016-03-05	74546	38289	0
2016-03-05	76487	62529	0
2016-03-05	82439	36396	10
2016-03-05	90006	36396	40
2016-03-06	90404	20703	0

Sample Output

```
2016-03-01 4 20703 Angela
2016-03-02 2 79722 Michael
2016-03-03 2 20703 Angela
2016-03-04 2 20703 Angela
2016-03-05 1 36396 Frank
2016-03-06 1 20703 Angela
```

Explanation

On *March 01, 2016* hackers 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 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 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*.

Occupations

[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
```

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.