SQL HACKER RANK SOLUTION

1. 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';
```

2. 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;
```

3. 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;

4. 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';

5. 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';

6. 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;

7. 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 ID%2 = 0 order by CITY asc;

or

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

8. 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

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 no. of records - no. of unique records = 3-2 = 1

```
Select count(city) - count(distinct city) as cnt_city from station;
```

9. 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

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

Sample Input

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

Sample Output

ABC 3
PQRS 4

Explanation

When ordered alphabetically, the **CITY** names are listed as **ABC, DEF, PQRS,** and **WXY**, with lengths 3,3,4,and 3. The longest name is **PQRS**, but there are 3 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.

Mysql:

select city,length(city) as len_city from station order by len_city asc, city asc limit 1;

select city,length(city) as len_city from station order by len_city desc,city asc limit 1;

DB2:

select city, length(city) from station order by length(city) DESC,city ASC fetch first row only;

select city, length(city) from station order by length(city) asc ,city asc fetch first row only;

10. 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.

Select distinct city from station where city like 'a%' or city like 'e%' or city like 'i%' or city like 'o%' or city like 'u%';

11. 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 like '%a' or city like '%e' or city like '%i' or city like '%o' or city like '%u';

12. 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

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

Select distinct city from station where (city like 'a%' or city like 'e%' or city like 'i%' or city like 'o%' or city like 'u%') and (city like '%a' or city like '%e' or city like '%i' or city like '%o' or city like '%u');

13. 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 upper(SUBSTR(CITY,1,1)) NOT IN
('A','E','I','O','U') AND lower(SUBSTR(CITY,1,1)) NOT IN
('a','e','i','o','u');
```

14. 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 upper(right(CITY,1)) NOT IN
('A','E','I','O','U') AND lower(right(CITY,1)) NOT IN

('a','e','i','o','u');

or

SELECT DISTINCT CITY FROM STATION WHERE UPPER(SUBSTR(CITY, LENGTH(CITY), 1))
NOT IN ('A','E','I','O','U') AND LOWER(SUBSTR(CITY, LENGTH(CITY),1)) NOT IN
('a','e','i','o','u');
```

15. 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 (upper(SUBSTR(CITY,1,1)) NOT IN
('A','E','I','O','U') AND lower(SUBSTR(CITY,1,1)) NOT IN
('a','e','i','o','u')) or (upper(right(CITY,1)) NOT IN ('A','E','I','O','U') AND lower(right(CITY,1))
NOT IN
('a','e','i','o','u'));
```

16. 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

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

```
SELECT DISTINCT CITY FROM STATION WHERE (upper(SUBSTR(CITY,1,1)) NOT IN
('A','E','I','O','U') AND lower(SUBSTR(CITY,1,1)) NOT IN ('a','e','i','o','u')) and
(upper(right(CITY,1)) NOT IN ('A','E','I','O','U') AND lower(right(CITY,1)) NOT IN
('a','e','i','o','u'));
```

17. 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.

Sample Input

employee_id	name	months	salary
12228	Rose	15	1968
33645	Angela	1	3443
45692	Frank	17	1608
56118	Patrick	7	1345
59725	Lisa	11	2330
74197	Kimberly	16	4372
78454	Bonnie	8	1771
83565	Michael	6	2017
98607	Todd	5	3396
99989	Joe	9	3573

Sample Output

Angela
Bonnie
Frank
Joe
Kimberly
Lisa
Michael
Patrick
Rose
Todd

Select name from employee order by name asc;

18. 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:

Column	Type
ID	Integer
Name	String
Marks	Integer

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

Sample Input

ID	Name	Marks
1	Ashley	81
2	Samantha	75
4	Julia	76
3	Belvet	84

Sample Output

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'.

```
Select name from students where marks>75 order by right(name,3) asc, id asc;
```

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

Input Format

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

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.

Sample Input

employee_id	name	months	salary
12228	Rose	15	1968
33645	Angela	1	3443
45692	Frank	17	1608
56118	Patrick	7	1345
59725	Lisa	11	2330
74197	Kimberly	16	4372
78454	Bonnie	8	1771
83565	Michael	6	2017
98607	Todd	5	3396
99989	Joe	9	3573

Sample Output

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*.

Select name from employee where salary >2000 and months<10;

20.

```
Given a table TRIANGLES that holds data for three fields namely A, B, C.
+-------------+------------+
| Column | Type |
+-------------+------------+
| A | INTEGER |
```

```
| B | INTEGER |
| C | INTEGER |
+-------------+------------+
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.
*/
--Solution
```

```
SELECT CASE
WHEN A + B <= C OR A + C <= B OR B + C <= A THEN 'Not A Triangle'
WHEN A = B AND B = C THEN 'Equilateral'
WHEN A = B OR B = C OR A = C THEN 'Isosceles'
ELSE 'Scalene'
END
FROM TRIANGLES;
```

21. 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
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.

```
SELECT (name || '(' || SUBSTR(occupation,1,1) || ')') FROM occupations ORDER BY name;
SELECT ('There are a total of ' || COUNT(occupation) || ' ' || LOWER(occupation) || 's' ||
'.') FROM occupations GROUP BY occupation ORDER BY COUNT(occupation), occupation ASC;
```

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:

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

Sample Input

Sample Output

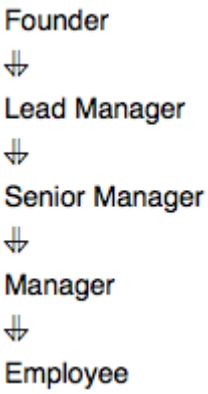
```
Jenny     Ashley      Meera  Jane
Samantha  Christeen   Priya  Julia
NULL      Ketty       NULL   Maria
```

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.

SELECT CONCAT(name,'(',ucase(left(occupation,1)),')') AS name_occupatn FROM occupations order by name;
select concat('There are a total of ',count(occupation),' ', lower(occupation),'s.') from occupations GROUP BY occupation ORDER BY COUNT(occupation) asc, occupation ASC;

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



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

Note:

- The tables may contain duplicate records.
- The *company_code* is string, so the sorting should not be **numeric**. For example, if the *company_codes* are *C_1*, *C_2*, and *C_10*, then the ascending *company_codes* will be *C_1*, *C_10*, and *C_2*.

Input Format

The following tables contain company data:

- *Company:* The *company_code* is the code of the company and *founder* is the founder of the company.

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

Column	Type
manager_code	String
senior_manager_code	String
lead_manager_code	String
company_code	String

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

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

company.

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

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

Table:

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*.

```
Select a.company_code,a.founder,count(Distinct b.lead_manager_code),

count (Distinct b.senior_manager_code),count(Distinct b.manager_code),

count (Distinct b.employee_code)

From company a join employee b

on a.company_code=b.company_code

GROUP BY a.Company_Code, a.Founder

Order by a.company_code asc;
```

or

```
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 as c
join Lead_Manager as l
on c.company_code = l.company_code
join Senior_Manager as s
on l.lead_manager_code = s.lead_manager_code
join Manager as m
on m.senior_manager_code = s.senior_manager_code
join Employee as e
on e.manager_code = m.manager_code
group by c.company_code, c.founder
order by c.company_code;
```

23. 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.

Column	Type
hacker_id	Integer
name	String

- *Difficulty:* The *difficult_level* is the level of difficulty of the challenge, and *score* is the score of the challenge for the difficulty level.

Column	Type
difficulty_level	Integer
score	Integer

- *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.

Column	Type
challenge_id	Integer
hacker_id	Integer
difficulty_level	Integer

- *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.

Column	Type
submission_id	Integer
hacker_id	Integer
challenge_id	Integer
score	Integer

Sample Input

Hackers Table:

hacker_id	name
5580	Rose
8439	Angela
27205	Frank
52243	Patrick
52348	Lisa
57645	Kimberly
77726	Bonnie
83082	Michael
86870	Todd
90411	Joe

Difficulty Table:

difficulty_level	score
1	20
2	30
3	40
4	60
5	80
6	100
7	120

Challenges Table:

challenge_id	hacker_id	difficulty_level
4810	77726	4
21089	27205	1
36566	5580	7
66730	52243	6
71055	52243	2

Submissions Table:

submission_id	hacker_id	challenge_id	score
68628	77726	36566	30
65300	77726	21089	10
40326	52243	36566	77
8941	27205	4810	4
83554	77726	66730	30
43353	52243	66730	0
55385	52348	71055	20
39784	27205	71055	23
94613	86870	71055	30
45788	52348	36566	0
93058	86870	36566	30
7344	8439	66730	92
2721	8439	4810	36
523	5580	71055	4
49105	52348	66730	0
55877	57645	66730	80
38355	27205	66730	35
3924	8439	36566	80
97397	90411	66730	100
84162	83082	4810	40
97431	90411	71055	30

Sample Output

```
90411 Joe
```

Explanation

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

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

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

Only hacker *90411* managed to earn a full score for more than one challenge, so we print the their *hacker_id* and *name* as 2 space-separated values.

```
SELECT S.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.name, S.hacker_id
HAVING count(S.challenge_id) > 1
ORDER BY count(S.challenge_id) DESC, S.hacker_id asc;
```

24. 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

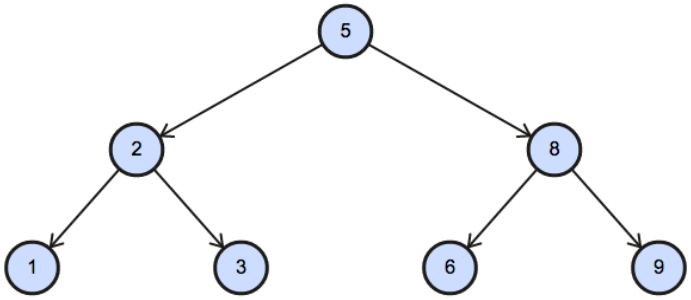
N	P
1	2
3	2
6	8
9	8
2	5
8	5
5	null

Sample Output

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

Explanation

The *Binary Tree* below illustrates the sample:




```
SELECT N,
(CASE
WHEN P is NULL THEN 'Root'
WHEN N in (SELECT P FROM BST) THEN 'Inner'
ELSE 'Leaf'
END) FROM BST
ORDER by N;
```

25. 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

```
Select avg(population) as avg_pop from city where District='California';
```

26. 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

```
Select sum(population) from city where district='california';
```

27. 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

```
Select floor(avg(population)) from city;
```

28. 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

```
Select sum(population) from city where COUNTRYCODE = 'JPN';
```

29. 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.: 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 per month.

Constraints

$1000 < \text{salary} < 10^5$.

Sample Input

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

Sample Output

```
2061
```

```
SELECT CEIL(AVG(salary) - AVG(REPLACE(salary, '0', '')))
FROM employees;
```

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

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

```
Select (max(population)-min(population)) from city;
```

31. 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 space-separated integers.

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.

Sample Input

employee_id	name	months	salary
12228	Rose	15	1968
33645	Angela	1	3443
45692	Frank	17	1608
56118	Patrick	7	1345
59725	Lisa	11	2330
74197	Kimberly	16	4372
78454	Bonnie	8	1771
83565	Michael	6	2017
98607	Todd	5	3396
99989	Joe	9	3573

Sample Output

```
69952 1
```

Explanation

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

employee_id	name	months	salary	earnings
12228	Rose	15	1968	29520
33645	Angela	1	3443	3443
45692	Frank	17	1608	27336
56118	Patrick	7	1345	9415
59725	Lisa	11	2330	25630
74197	Kimberly	16	4372	69952
78454	Bonnie	8	1771	14168
83565	Michael	6	2017	12102
98607	Todd	5	3396	16980
99989	Joe	9	3573	32157

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.

```
select months*salary, count(*) from employee

group by months*salary

order by months*salary desc

limit 1;
```

or

```
select MAX(salary*months), COUNT(*) from Employee where (salary * months) >= (select
MAX(salary * months) from employee);
```

32. Query the following two values from the **STATION** table:

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

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.

Output Format

Your results must be in the form:

lat lon
where lat is the sum of all values in *LAT_N* and lan is the sum of all values in *LONG_W*. Both results must be rounded to a scale of decimal places.

```
Select round(sum(lat_n),2) as lat , round(sum(long_w),2) as lon from station;
```

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

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 round(sum(lat_n),4) from station
Where lat_n>38.7880 and lat_n<137.2345;
```

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

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 round (max(lat_n),4) from station
Where lat_n <137.2345;
```

35. 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 decimal places.

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 ROUND(LONG_W,4) FROM STATION
WHERE LAT_N = (SELECT MAX(LAT_N) FROM STATION WHERE LAT_N <
137.2345);
```

36. 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:

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 round(min(lat_n),4) from station where lat_n>38.7780;
```

37. 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 CountryCode='USA' and population>100000;
```

38. 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:

CITY

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

COUNTRY

Field	Type
CODE	VARCHAR2(3)
NAME	VARCHAR2(44)
CONTINENT	VARCHAR2(13)
REGION	VARCHAR2(25)
SURFACEAREA	NUMBER
INDEPYEAR	VARCHAR2(5)
POPULATION	NUMBER
LIFEEXPECTANCY	VARCHAR2(4)
GNP	NUMBER
GNPOLD	VARCHAR2(9)
LOCALNAME	VARCHAR2(44)
GOVERNMENTFORM	VARCHAR2(44)
HEADOFSTATE	VARCHAR2(32)
CAPITAL	VARCHAR2(4)
CODE2	VARCHAR2(2)

Select sum (a.population) from city a join country b

On a.countrycode = b.code

Where CONTINENT = 'ASIA';

39. 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:

CITY

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

COUNTRY

Field	Type
CODE	VARCHAR2(3)
NAME	VARCHAR2(44)
CONTINENT	VARCHAR2(13)
REGION	VARCHAR2(25)
SURFACEAREA	NUMBER
INDEPYEAR	VARCHAR2(5)
POPULATION	NUMBER
LIFEEXPECTANCY	VARCHAR2(4)
GNP	NUMBER
GNPOLD	VARCHAR2(9)
LOCALNAME	VARCHAR2(44)
GOVERNMENTFORM	VARCHAR2(44)
HEADOFSTATE	VARCHAR2(32)
CAPITAL	VARCHAR2(4)
CODE2	VARCHAR2(2)

Select a.name from city a join country b

On a.countrycode = b.code

Where CONTINENT='Africa';

40. 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:

CITY

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

COUNTRY

Field	Type
CODE	VARCHAR2(3)
NAME	VARCHAR2(44)
CONTINENT	VARCHAR2(13)
REGION	VARCHAR2(25)
SURFACEAREA	NUMBER
INDEPYEAR	VARCHAR2(5)
POPULATION	NUMBER
LIFEEXPECTANCY	VARCHAR2(4)
GNP	NUMBER
GNPOLD	VARCHAR2(9)
LOCALNAME	VARCHAR2(44)
GOVERNMENTFORM	VARCHAR2(44)
HEADOFSTATE	VARCHAR2(32)
CAPITAL	VARCHAR2(4)
CODE2	VARCHAR2(2)

Select b.continent,floor(avg(a.population))

from city a join country b

on a.countrycode = b.code

group by b.continent;

41. 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)*

```
SELECT CASE
      WHEN Grades.Grade < 8 THEN 'NULL'
      ELSE Students.Name
      END
, Grades.Grade, Students.Marks
FROM Students, Grades
WHERE Students.Marks >= Grades.Min_mark AND Students.Marks <= Grades.Max_mark
ORDER BY Grades.Grade DESC, Students.Name;
```