SQL HACKER RANK

BASIC SELECT

1. Revising the Select Query I

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

Ans:

```
select * from city where population>100000 and countrycode='USA'
```

2. Revising the Select Query II

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

Ans:

```
select name from city where population >120000 and countrycode='USA'
```

3. Select ALL

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

Ans:

```
select * from city ;
```

4. Select By ID

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

Ans:

```
select * from city where ID=1661 ;
```

5. Japanese Cities Attributes

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

Ans:

```
select * from city where COUNTRYCODE = 'JPN' ;
```

6. Japanese Cities Names

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

Ans:

```
select name from city where countrycode='JPN';
```

7. Weather Observation Station 1

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.

Ans: `select city,state from station ;`

8. Weather Observation Station 3

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.

Ans:

```
select distinct city from station where ID%2 = 0 ;
```

9. Weather Observation Station 4

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 – number of unique city names = 3-2 =1

Ans:

```
select count(city) - count(distinct city) from station ;
```

10.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.

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.

Ans:

```
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;
```

11.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.

Ans:

```
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%' ORDER BY CITY ASC ;
```

12.Weather Observation Station 7

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.

Ans:

```
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';
```

13.Weather Observation Station 8

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.

Ans:

```
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';
```

14.Weather Observation Station 9

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.

Ans:

```
select distinct city from station where city not like 'a%' or city not like 'e%'  or city not like 'i%' or city not like 'o%' or city not like 'u%'
```

15.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

Ans:

```
select distinct city from station where city not like '%a' and city not like '%e' and city not like '%i' and city not like '%o' and city not like '%u'
```

16.Weather Observation Station 11

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.

Ans:

```
select distinct city from station where (city not like '%a' and city not like '%e' and city not like '%i' and city not like '%o' and city not like '%u') or (city not like 'a%' and city not like 'e%' and city not like 'i%' and city not like 'o%' and city not like 'u%')
```

17.Weather Observation Station 12

Query the list of *CITY* names from **STATION** that either 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.

Ans:

```
select distinct city from station where (city not like '%a' and city not like '%e' and city not like '%i' and city not like '%o' and city not like '%u') and  (city not like 'a%' and city not like 'e%' and city not like 'i%' and city not like 'o%' and city not like 'u%')
```

18. Higher than 75 Marks

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'.

Ans:

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

19. Employee Names

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

Ans:

```
select name from employee order by name;
```

20.Employee Salaries

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.

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.

Ans:

```
select name from employee where salary > 2000 and months < 10
order by employee_id asc;
```

ADVANCE SELECT

21. 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 form an Isosceles triangle, because .

Values in the tuple form an Equilateral triangle, because . Values in the tuple form a Scalene triangle, because .

Values in the tuple cannot form a triangle because the combined value of sides and is not larger than that of side .

Ans:

MYSQL

```
SELECT
  CASE
    WHEN (A + B <= C) OR (B + C <= A) OR (A + C <= B) THEN 'Not A Triangle'
    WHEN (A = B) AND (B = C) THEN 'Equilateral'
    WHEN ((A = B) &(A != C)) OR ((B = C) &(B != A)) OR ((A = C) &(A != B)) THEN 'Isosceles'
    WHEN (A != B) AND (B != C) AND (A != C) THEN 'Scalene'
    END AS Triangle_Type
    FROM
   TRIANGLES;
```

MS SQL SERVER

```
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;
```

22.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:

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.

Ans:

MYSQL

```
select concat(name,"(",left(occupation,1),")") from occupations
order by name asc;

select concat("There are a total of ",count(occupation)," ",lower(occupation), "s.") from
occupations
group by occupation
order by count(occupation) asc,occupation;
```

23.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.

Ans:

MYSQL

```
CREATE VIEW pq AS (
   SELECT
```

```
        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',
        ROW_NUMBER() OVER (PARTITION BY occupation ORDER BY name) as cr
    FROM occupations
);

SELECT MAX(Doctor),MAX(Professor),MAX(Singer),MAX(Actor) FROM pq
GROUP BY cr;
```

OR

```
Select Doctor, Professor, Singer, Actor from ( select
NameOrder,
max(case Occupation when 'Doctor' then Name end) as Doctor,
max(case Occupation when 'Professor' then Name end) as Professor,
max(case Occupation when 'Singer' then Name end) as Singer,
max(case Occupation when 'Actor' then Name end) as Actor
from ( select Occupation, Name, row_number() over(partition by Occupation order by
Name ASC) as NameOrder from Occupations ) as NameLists group by NameOrder
) as Names
```

24.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

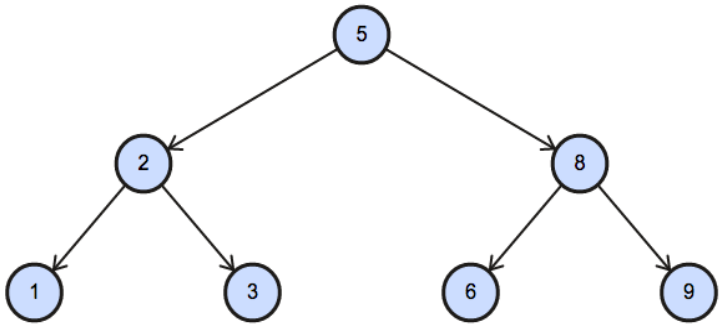
N	P
1	2
3	2
6	8
9	8
2	5
8	5
5	null

Sample Output

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

Explanation

The *Binary Tree* below illustrates the sample:



Ans:

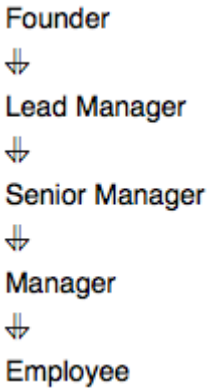
MYSQL

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

ORDER by N;

25.New Companies

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 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*.

Ans:

MYSQL

```
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 as a join employee as 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;
```

AGGREGATION

26. Revising aggregation – The Count Function

Query a *count* of the number of cities in **CITY** having a *Population* larger than .100000

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

Ans:

MYSQL

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

27. Revising aggregation – The Sum Function

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

Ans:

```
select sum(population) from city where district ='California';
```

28. Revising aggregation – Average

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

Ans:

```
select avg (population) from city where district = 'California';
```

29. Average Population

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

Ans:

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

30. Japan Population

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

Ans:

```
select sum(population) from city where countrycode='JPN';
```

31. Population Density Difference

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

Ans:

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

32. 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 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

.

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 zeros* 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 . The *actual* average salary is .

The resulting error between the two calculations is . Since it is equal to the integer , it does not get rounded up.

Ans:

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

33. Top Earners

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.

Ans:

```
select months*salary, count(*) from employee
group by months*salary
order by months*salary desc
limit 1;
```

34. Weather Observation Station 2

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:

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 lon is the sum of all values in *LONG_W*. Both results must be rounded to a scale of 2 decimal places.

Ans:

```
select round(sum(LAT_N),2) AS lat , round(sum(LONG_W),2) as lon from STATION;
```

35. Weather Observation Station 13

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:

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.

Ans:

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

36. Weather Observation Station 14

Query the greatest value of the *Northern Latitudes* (*LAT_N*) from **STATION** that is less than 137.345. Truncate 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.

Ans:

```
SELECT ROUND(MAX(LAT_N),4) FROM STATION WHERE LAT_N < 137.345;
```

37. Weather Observation Station 15

Query the *Western Longitude* (*LONG_W*) for the largest *Northern Latitude* (*LAT_N*) in **STATION** that is less than 137.345.

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.

Ans:

```
SELECT ROUND(LONG_W,4) FROM STATION WHERE LAT_N < 137.345
ORDER BY LAT_N DESC LIMIT 1;
```

38. Weather Observation Station 16

Query the smallest *Northern Latitude* (*LAT_N*) from **STATION** that is greater than . 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.

Ans:

```
SELECT ROUND(MIN(LAT_N),4) FROM STATION WHERE LAT_N > 38.7780;
```

39. Weather Observation Station 17

Query the *Western Longitude* (*LONG_W*)where the smallest *Northern Latitude* (*LAT_N*) in **STATION** is greater than . 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.

Ans:

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

40. Weather Observation Station 18

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

- a happens to equal the **minimum** value in Northern Latitude (LAT_N in **STATION**).
- b happens to equal the **minimum** value in Western Longitude (LONG_W in **STATION**).
- c happens to equal the **maximum** value in Northern Latitude (LAT_N in **STATION**).
- c happens to equal the **maximum** value in Western Longitude (LONG_W in **STATION**).

Query the Manhattan Distance between points and and round it 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.

Ans:

```
/*
Manhattan distance
Definition: The distance between two points measured along axes at right angles. In a plane with p1 at (x1, y1) and p2 at (x2, y2), it is |x1 - x2| + |y1 - y2|.
*/
```

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

OR

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

41. Weather Observation Station 19

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

Query the Euclidean Distance between points and and format your answer to display decimal digits.

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.

Ans:

```
SELECT ROUND(SQRT(POWER(MAX(LAT_N)-MIN(LAT_N),2) + POWER(MAX(LONG_W)-MIN(LONG_W),2)),4) FROM STATION;
```

42. 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 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.

Ans:

```
SET @r = 0;

SELECT ROUND(AVG(Lat_N), 4)

FROM (SELECT (@r := @r + 1) AS r, Lat_N FROM Station ORDER BY Lat_N) Temp

WHERE

   r = (SELECT CEIL(COUNT(*) / 2) FROM Station) OR

   r = (SELECT FLOOR((COUNT(*) / 2) + 1) FROM Station);
```

OR

```
SELECT ROUND(t.lat_n,4) FROM (SELECT lat_n, NTILE(2) OVER (ORDER BY lat_n) as median
FROM station) t WHERE median = 1 ORDER BY lat_n DESC LIMIT 1;
```

BASIC JOIN

43. POPULATION SENSUS

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)

Ans:

```
SELECT SUM(A.POPULATION) FROM
CITY AS A INNER JOIN COUNTRY AS B
ON A.COUNTRYCODE=B.CODE
WHERE B.CONTINENT='ASIA';
```

44. AFRICAN CITIES

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)

Ans:

```
SELECT A.NAME FROM

CITY AS A INNER JOIN COUNTRY AS B

ON A.COUNTRYCODE=B.CODE

WHERE CONTINENT='AFRICA';
```

45. AVERAGE POPULATION OF EACH CONTINENT

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)

Ans:

```
SELECT B.CONTINENT,FLOOR(AVG(A.POPULATION)) FROM
CITY AS A INNER JOIN COUNTRY AS B
ON A.COUNTRYCODE=B.CODE
GROUP BY CONTINENT;
```

46. TOP COMPETITORS

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.

Ans:

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

47. 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, and , then and .

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
```

Ans:

```
select w.id, p.age, w.coins_needed, w.power from Wands as w
join Wands_Property as p
on w.code = p.code
where w.coins_needed = (select min(coins_needed)
                        from Wands w2 inner join Wands_Property p2
                        on w2.code = p2.code
                        where p2.is_evil = 0 and p.age = p2.age and w.power = w2.power)
order by w.power desc, p.age desc;
```

48. CHALLENGES

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.

Column	Type
hacker_id	Integer
name	String

- *Challenges:* The *challenge_id* is the id of the challenge, and *hacker_id* is the id of the student who created the challenge.

Column	Type
challenge_id	Integer
hacker_id	Integer

Sample Input 0

Hackers Table:

hacker_id	name
5077	Rose
21283	Angela
62743	Frank
88255	Patrick
96196	Lisa

Challenges Table:

challenge_id	hacker_id
61654	5077
58302	21283
40587	88255
29477	5077
1220	21283
69514	21283
46561	62743
58077	62743
18483	88255
76766	21283
52382	5077
74467	21283
33625	96196
26053	88255
42665	62743
12859	62743
70094	21283
34599	88255
54680	88255
61881	5077

Sample Output 0

21283 Angela 6
88255 Patrick 5
96196 Lisa 1

Sample Input 1

Hackers Table:

hacker_id	name
12299	Rose
34856	Angela
79345	Frank
80491	Patrick
81041	Lisa

Challenges Table:

challenge_id	hacker_id
63963	81041
63117	79345
28225	34856
21989	12299
4653	12299
70070	79345
36905	34856
61136	80491
17234	12299
80308	79345
40510	34856
79820	80491
22720	12299
21394	12299
36261	34856
15334	12299
71435	79345
23157	34856
54102	34856
69065	80491

Sample Output 1

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:

hacker_id	name	challenges_created
21283	Angela	6
88255	Patrick	5
5077	Rose	4
62743	Frank	4
96196	Lisa	1

Students and both created challenges, but the maximum number of challenges created is so these students are excluded from the result.

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

hacker_id	name	challenges_created
12299	Rose	6
34856	Angela	6
79345	Frank	4
80491	Patrick	3
81041	Lisa	1

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

Ans:

```
SELECT H.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 H.hacker_id, H.name
HAVING CNT=(SELECT COUNT(C1.challenge_id) FROM CHALLENGES AS C1 GROUP BY
c1.hacker_id ORDER BY count(*) desc limit 1)
OR
CNT NOT IN (SELECT COUNT(C2.challenge_id) FROM CHALLENGES AS C2 GROUP BY
c2.hacker_id HAVING c2.hacker_id <> H.hacker_id)
ORDER BY COUNT(C.challenge_id) DESC,H.HACKER_ID ASC;
```

49. CONTEST LEADERBOARD

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.

Column	Type
hacker_id	Integer
name	String

- *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.

Column	Type
submission_id	Integer
hacker_id	Integer
challenge_id	Integer
score	Integer

Sample Input

Hackers Table:

hacker_id	name
4071	Rose
4806	Angela
26071	Frank
49438	Patrick
74842	Lisa
80305	Kimberly
84072	Bonnie
87868	Michael
92118	Todd
95895	Joe

Submissions Table:

submission_id	hacker_id	challenge_id	score
67194	74842	63132	76
64479	74842	19797	98
40742	26071	49593	20
17513	4806	49593	32
69846	80305	19797	19
41002	26071	89343	36
52826	49438	49593	9
31093	26071	19797	2
81614	84072	49593	100
44829	26071	89343	17
75147	80305	49593	48
14115	4806	49593	76
6943	4071	19797	95
12855	4806	25917	13
73343	80305	49593	42
84264	84072	63132	0
9951	4071	49593	43
45104	49438	25917	34
53795	74842	19797	5
26363	26071	19797	29
10063	4071	49593	96

Sample Output

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 .

Hacker *74842* submitted solutions for challenges *19797* and *63132*, so the total score

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

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

Ans:

```
select m.hacker_id, h.name, sum(score) as total_score from
(select hacker_id, challenge_id, max(score) as score
from Submissions group by hacker_id, challenge_id) as m
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;
```

50. 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)*

Ans:

```
SELECT
  CASE
    WHEN GRADES.GRADE >= 8 THEN STUDENTS.NAME
    WHEN GRADES.GRADE < 8 THEN NULL
  END AS NAME,
  GRADES.GRADE, STUDENTS.MARKS
FROM STUDENTS
  LEFT JOIN GRADES
```

```
  ON STUDENTS.MARKS >= MIN_MARK AND STUDENTS.MARKS <=
MAX_MARK

ORDER BY

  GRADES.GRADE DESC, STUDENTS.NAME ASC, STUDENTS.MARKS ASC;
```

OR

```
SELECT (CASE WHEN GRADE <8 THEN 'NULL' ELSE NAME END) AS
NAME, GRADE, MARKS
FROM (
SELECT ID, NAME, MARKS,(CASE
   WHEN MARKS >= 0 AND MARKS <= 9 THEN 1
   WHEN MARKS >= 10 AND MARKS <= 19 THEN 2
   WHEN MARKS >= 20 AND MARKS <= 29 THEN 3
   WHEN MARKS >= 30 AND MARKS <= 39 THEN 4
   WHEN MARKS >= 40 AND MARKS <= 49 THEN 5
   WHEN MARKS >= 50 AND MARKS <= 59 THEN 6
   WHEN MARKS >= 60 AND MARKS <= 69 THEN 7
   WHEN MARKS >= 70 AND MARKS <= 79 THEN 8
   WHEN MARKS >= 80 AND MARKS <= 89 THEN 9
   ELSE 10 END) AS GRADE
FROM STUDENTS)
ORDER BY GRADE DESC,
   CASE WHEN GRADE >= 8 AND GRADE <= 10 THEN GRADE END DESC,
   CASE WHEN GRADE >= 8 AND GRADE <= 10 THEN NAME END ASC,
   CASE WHEN GRADE >= 1 AND GRADE <= 7 THEN MARKS END ASC;
```

51. SQL Project Planning

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.

Ans:

```
SELECT START_DATE, MIN(END_DATE)
FROM
```

```
(SELECT START_DATE
 FROM PROJECTS
 WHERE START_DATE NOT IN
    (SELECT END_DATE
     FROM PROJECTS)) A,
(SELECT END_DATE
 FROM PROJECTS
 WHERE END_DATE NOT IN
    (SELECT START_DATE
     FROM PROJECTS)) B
WHERE START_DATE < END_DATE
GROUP BY START_DATE
ORDER BY (MIN(END_DATE) - START_DATE), START_DATE;
```

52. SQL Project Planning

53. SQL Project Planning

54. SQL Project Planning

ALTERNATIVE QUERIES

55. 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

Ans:

```
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
   ) ;
```

56. Draw the Triangle 1

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)*.

Ans:

```
set @number = 20+1;
select repeat('* ', @number := @number - 1)
from information_schema.tables ;
```

57. 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)*.

Ans:

```
set @number = 0;
select repeat('* ', @number := @number + 1)
from information_schema.tables
where @number < 20 ;
```

58. Placement

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	Friend_ID
1	2
2	3
3	4
4	1

Friends

ID	Name
1	Ashley
2	Samantha
3	Julia
4	Scarlet

Students

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*

Ans:

```
SELECT t.Name
FROM (
   SELECT s1.ID, s1.Name, p1.Salary, f.Friend_ID, s2.name as friend_name, p2.Salary as
friend_salary
   FROM Students s1
   JOIN Packages p1 ON s1.ID = p1.ID
   JOIN Friends f ON s1.ID = f.ID
   JOIN Students s2 ON f.Friend_ID = s2.ID
   JOIN Packages p2 ON f.Friend_ID = p2.ID
   ) t
WHERE t.friend_salary > t.Salary
ORDER BY friend_salary;
```

59. Placement