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

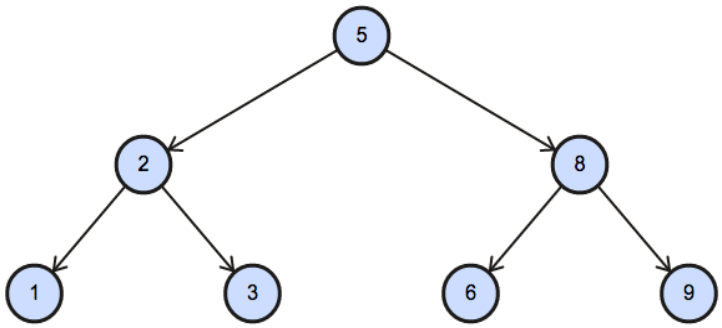
N	P
1	2
3	2
6	8
9	8
2	5
8	5
5	null

Sample Output

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

Explanation

The *Binary Tree* below illustrates the sample:



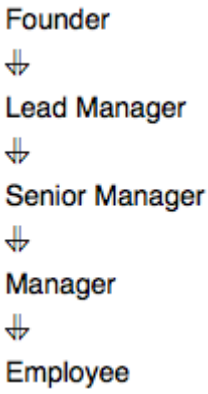
Ans:

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

CITY

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

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

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

CITY

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

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

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

CITY

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

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

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 . 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.

Ans:

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