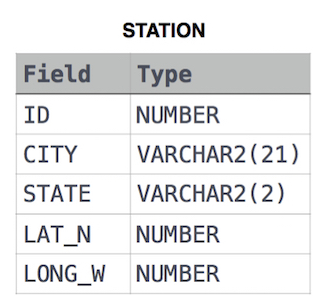


1. Query the two cities in **STATION** with the shortest and longest CITY names, as well as their respective lengths (i.e.: number of characters in the name). If there is more than one smallest or largest city, choose the one that comes first when ordered alphabetically.  
   The **STATION** table is described as follows:



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

**SOLUTION**

SELECT TOP 1 city, LEN(city) AS name\_length

FROM STATION

ORDER BY name\_length, city;

SELECT TOP 1 city, LEN(city) AS name\_length

FROM STATION

ORDER BY name\_length DESC, city;

==========================================================================================================================================================================

1. Query the list of *CITY* names starting with vowels (i.e., a, e, i, o, or u) from **STATION**. Your result *cannot* contain duplicates.

**Input Format**

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

A table with text and numbers

Description automatically generated

where LAT\_N is the northern latitude and LONG\_W is the western longitude.

**SOLUTION**

SELECT DISTINCT CITY

FROM STATION

WHERE CITY LIKE '[AEIOUaeiou]%';

**OR (SOLUTION 2)**

WHERE LEFT(CITY, 1) IN ('A', 'E', 'I', 'O', 'U', 'a', 'e', 'i', 'o', 'u');

==========================================================================================================================================================================

1. 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:

A table with text and numbers

Description automatically generated

where LAT\_N is the northern latitude and LONG\_W is the western longitude.

**SOLUTION**

SELECT DISTINCT CITY

FROM STATION

WHERE CITY LIKE '%[aeiou]';

**OR (SOLUTION 2)**

WHERE RIGHT(CITY, 1) IN ('a', 'e', 'i', 'o', 'u');

==========================================================================================================================================================================

1. 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:



where LAT\_N is the northern latitude and LONG\_W is the western longitude.

**SOLUTION**

SELECT DISTINCT CITY

FROM STATION

WHERE CITY LIKE '[AEIOUaeiou]%' and CITY LIKE '%[aeiou]';

**OR (SOLUTION 2)**

WHERE left(city, 1) IN ('A', 'E', 'I', 'O', 'U', 'a', 'e', 'i', 'o', 'u')

AND right(city, 1) IN ('a', 'e', 'i', 'o', 'u');

==========================================================================================================================================================================

1. 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:



where LAT\_N is the northern latitude and LONG\_W is the western longitude.

**SOLUTION**

SELECT DISTINCT CITY

FROM STATION

WHERE CITY NOT LIKE '[AEIOUaeiou]%';

==========================================================================================================================================================================

1. 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:

A table with text and numbers

Description automatically generated

where LAT\_N is the northern latitude and LONG\_W is the western longitude.

**SOLUTION**

SELECT DISTINCT CITY

FROM STATION

WHERE right(city,1) not in ('a', 'e', 'i', 'o', 'u');

==========================================================================================================================================================================

1. 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:

A table with text and numbers

Description automatically generated

where LAT\_N is the northern latitude and LONG\_W is the western longitude.

**SOLUTION**

SELECT DISTINCT CITY

FROM STATION

WHERE LEFT(city, 1) NOT IN ('A', 'E', 'I', 'O', 'U', 'a', 'e', 'i', 'o', 'u')

OR RIGHT(city, 1) NOT IN ('a', 'e', 'i', 'o', 'u');

==========================================================================================================================================================================

1. Query the *Name* of any student in **STUDENTS** who scored higher than

*Marks*. Order your output by the *last three characters* of each name. If two or more students both have names ending in the same last three characters (i.e.: Bobby, Robby, etc.), secondary sort them by ascending *ID*.

**Input Format**

A screenshot of a computer

Description automatically generated

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

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

**Sample Input**

A screenshot of a computer

Description automatically generated

**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'.

**SOLUTION**

SELECT name

FROM students

WHERE marks > 75

ORDER BY SUBSTRING(name, LEN(name) - 2, 3) ASC, id ASC;

==========================================================================================================================================================================

1. 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:

A table of numbers with black text

Description automatically generated

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**



**Sample Output**

Angela

Bonnie

Frank

Joe

Kimberly

Lisa

Michael

Patrick

Rose

Todd

**SOLUTION**

SELECT name

FROM employee

ORDER BY name ASC;

==========================================================================================================================================================================

1. 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:

A table of numbers with black text

Description automatically generated

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.

**SOLUTION**

SELECT name AS employee\_name

FROM employee

WHERE salary > 2000

AND months < 10

ORDER BY employee\_id;

==========================================================================================================================================================================

1. 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 4 decimal places.

**Input Format**

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



where *LAT\_N* is the northern latitude and *LONG\_W* is the western longitude.

**SOLUTION**

SELECT TOP 1 CAST(long\_w AS DECIMAL(10, 4)) AS western\_longitude

FROM Station

WHERE lat\_n < 137.2345

ORDER BY lat\_n DESC;

==========================================================================================================================================================================

1. 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:

A table of numbers with text

Description automatically generated with medium confidence

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

**Sample Input**

A grid of numbers and letters

Description automatically generated

**Sample Output**

Isosceles

Equilateral

Scalene

Not 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 AS TriangleType

FROM TRIANGLES;

==========================================================================================================================================================================

1. Generate the following two result sets:
2. 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).
3. 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**

A white rectangular box with black text

Description automatically generated

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:

A screenshot of a cellphone

Description automatically generated

**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.

**SOLUTION**

SELECT

NAME + '(' + LEFT(OCCUPATION, 1) + ')'

FROM OCCUPATIONS

ORDER BY NAME;

SELECT

'There are a total of ' + CAST(COUNT(OCCUPATION) AS VARCHAR) + ' ' + LOWER(OCCUPATION) + 's.' AS OccupationCount

FROM OCCUPATIONS

GROUP BY OCCUPATION

ORDER BY COUNT(OCCUPATION), OCCUPATION;

==========================================================================================================================================================================

1. [Pivot](https://en.wikipedia.org/wiki/Pivot_table) 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:

A white rectangular box with black text

Description automatically generated

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

**Sample Input**

A screenshot of a phone

Description automatically generated

**Sample Output**

Jenny Ashley Meera Jane

Samantha Christeen Priya Julia

NULL Ketty NULL Maria

**SOLUTION**

SELECT

MAX(CASE WHEN Occupation = 'Doctor' THEN Name END) AS Doctor,

MAX(CASE WHEN Occupation = 'Professor' THEN Name END) AS Professor,

MAX(CASE WHEN Occupation = 'Singer' THEN Name END) AS Singer,

MAX(CASE WHEN Occupation = 'Actor' THEN Name END) AS Actor

FROM (

SELECT

Name,

Occupation,

ROW\_NUMBER() OVER (PARTITION BY Occupation ORDER BY Name) AS rn

FROM OCCUPATIONS

) AS ranked

GROUP BY rn;

================================================================================================================================================================

1. 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.

A screenshot of a computer

Description automatically generated

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**

A screenshot of a cell phone

Description automatically generated

**Sample Output**

1 Leaf

2 Inner

3 Leaf

5 Root

6 Leaf

8 Inner

9 Leaf

**SOLUTION**

SELECT

N AS Node,

CASE

WHEN P IS NULL THEN 'Root'

WHEN N NOT IN (SELECT DISTINCT P FROM BST WHERE P IS NOT NULL) THEN 'Leaf'

ELSE 'Inner'

END AS NodeType

FROM BST

ORDER BY N;

==========================================================================================================================================================================

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

A group of black text

Description automatically generated

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.

A screenshot of a computer code

Description automatically generated

* *Lead\_Manager:* The *lead\_manager\_code* is the code of the lead manager, and the *company\_code* is the code of the working company.

A screenshot of a computer code

Description automatically generated

* *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.



* *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.

A table of code

Description automatically generated

* *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.

A table of code

Description automatically generated

**SOLUTION**

SELECT

C.company\_code,

C.founder,

COUNT(DISTINCT LM.lead\_manager\_code) AS num\_lead\_managers,

COUNT(DISTINCT SM.senior\_manager\_code) AS num\_senior\_managers,

COUNT(DISTINCT M.manager\_code) AS num\_managers,

COUNT(DISTINCT E.employee\_code) AS num\_employees

FROM Company C

LEFT JOIN Lead\_Manager LM ON C.company\_code = LM.company\_code

LEFT JOIN Senior\_Manager SM ON LM.lead\_manager\_code = SM.lead\_manager\_code

LEFT JOIN Manager M ON SM.senior\_manager\_code = M.senior\_manager\_code

LEFT JOIN Employee E ON M.manager\_code = E.manager\_code

GROUP BY C.company\_code, C.founder

ORDER BY C.company\_code;

==========================================================================================================================================================================

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

**Input Format**

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

A screenshot of a computer

Description automatically generated

**SOLUTION**

SELECT

MAX(population) - MIN(population) AS population\_difference

FROM city;

==========================================================================================================================================================================

1. 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:

A table of numbers with text

Description automatically generated with medium confidence

**SOLUTION**

SELECT CAST(CEILING(AVG(CAST(salary AS FLOAT)) - AVG(CAST(REPLACE(salary, '0', '') AS FLOAT))) AS INT)

FROM EMPLOYEES;

==========================================================================================================================================================================

1. We define an employee's *total earnings* to be their monthly  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:

A table of numbers with black text

Description automatically generated

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**

A table of numbers and letters

Description automatically generated

**Sample Output**

69952 1

**SOLUTION**

SELECT

MAX(months \* salary) AS max\_earnings,

COUNT(\*) AS num\_employees\_with\_max\_earnings

FROM Employee

WHERE months \* salary = (SELECT MAX(months \* salary) FROM Employee);

==========================================================================================================================================================================