1- 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.  
for ORACLE

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

for MySQL

SELECT DISTINCT(CITY) FROM STATION WHERE (ID%2)=0 ;

2-Find the difference between the total number of **CITY** entries in the table and the number of distinct **CITY** entries in the table.

for MySQL

SELECT COUNT(CITY) - COUNT(DISTINCT CITY) FROM STATION;

3- 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.

**Explanation**

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

for ORACLE

1-SELECT \* FROM (SELECT CITY, LENGTH(CITY) FROM STATION ORDER BY LENGTH(CITY), CITY)

WHERE ROWNUM = 1

UNION

SELECT \* FROM (SELECT CITY, LENGTH(CITY) FROM STATION ORDER BY LENGTH(CITY) DESC, CITY) WHERE ROWNUM = 1;

for MySQL

2-select city, length(city) from station order by length(city),city asc limit 1;

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

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

for MySQL

SELECT DISTINCT CITY FROM STATION

WHERE CITY REGEXP '[aeiou]$';

for ORACLE

SELECT DISTINCT CITY

FROM STATION

WHERE REGEXP\_LIKE(City, '[aeiou]$');

5- 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.

for MySQL

select DISTINCT CITY from station where city regexp '[aeiou]$' and city in (SELECT CITY FROM STATION WHERE CITY REGEXP '^[aeiou]');

for MySQL

select distinct city from station

where left(city,1) in ('a','e','i','o','u')

and right(city, 1) in ('a','e','i','o','u')

for ORACLE

SELECT DISTINCT CITY FROM STATION WHERE REGEXP\_LIKE(LOWER(CITY), '^[aeiou]') intersect SELECT DISTINCT CITY FROM STATION WHERE REGEXP\_LIKE(LOWER(CITY), '[aeiou]$');

6- Query the list of CITY names from **STATION** that do not start with vowels. Your result cannot contain duplicates.

for MySQL

SELECT DISTINCT(CITY) FROM STATION WHERE LEFT(CITY,1) NOT IN ('a','e','i','o','u');

for MySQL

SELECT distinct CITY FROM STATION WHERE CITY REGEXP '^[^aeiou]';

for ORACLE

SELECT DISTINCT CITY FROM STATION WHERE REGEXP\_LIKE(LOWER(CITY), '^[^aeiou]');

7 Query the list of CITY names from **STATION** that do not end with vowels. Your result cannot contain duplicates.

for MySQL

SELECT DISTINCT(CITY) FROM STATION WHERE right(CITY,1) NOT IN ('a','e','i','o','u');

for MySQL

SELECT distinct CITY FROM STATION WHERE CITY REGEXP '[^aeiou]$';

for ORACLE

SELECT DISTINCT CITY FROM STATION WHERE REGEXP\_LIKE(LOWER(CITY), '[^aeiou]$');

8- 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.

for MySQL

SELECT distinct CITY FROM STATION WHERE CITY REGEXP '^[^aeiou]|[^aeiou]$' ;

for MySQL

SELECT DISTINCT CITY FROM STATION WHERE CITY NOT REGEXP '^[AEIOU]' OR CITY NOT REGEXP '[AEIOU]$';

for MySQL

SELECT DISTINCT CITY FROM STATION WHERE RIGHT (CITY, 1) NOT IN ('A','E', 'I', 'O', 'U') OR LEFT (CITY,1) NOT IN ('A','E', 'I', 'O', 'U');

9- Query the list of CITY names from **STATION** that do not start with vowels and do not end with vowels. Your result cannot contain duplicates.

for MySQL

SELECT distinct CITY FROM STATION WHERE CITY REGEXP '^[^aeiou].\*[^aeiou]$' ;

for MySQL

SELECT DISTINCT CITY FROM STATION WHERE CITY NOT REGEXP '^[AEIOU]' and CITY NOT REGEXP '[AEIOU]$';

for MySQL

SELECT DISTINCT CITY FROM STATION WHERE RIGHT (CITY, 1) NOT IN ('A','E', 'I', 'O', 'U') and LEFT (CITY,1) NOT IN ('A','E', 'I', 'O', 'U');

10-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.

for MySQL

SELECT NAME

FROM STUDENTS

WHERE MARKS > 75

ORDER BY SUBSTRING(NAME, -3), ID ASC;

for MySQL

SELECT NAME FROM STUDENTS WHERE MARKS > 75 ORDER BY RIGHT(NAME, 3), ID ASC;

11- Write a query that prints a list of employee names (i.e.: the name attribute) from the **Employee** table in alphabetical order.

for MySQL

SELECT NAME FROM EMPLOYEE ORDER BY NAME ASC;

12-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.

for MySQL

SELECT NAME FROM EMPLOYEE WHERE SALARY>2000 AND MONTHS<10 ORDER BY EMPLOYEE\_ID;

13- Write a query identifying the *type* of each record in the **TRIANGLES** table using its three side lengths. Output one of the following statements for each record in the table:

* **Equilateral**: It's a triangle with  sides of equal length.
* **Isosceles**: It's a triangle with  sides of equal length.
* **Scalene**: It's a triangle with  sides of differing lengths.
* **Not A Triangle**: The given values of *A*, *B*, and *C* don't form a triangle.

**Input Format**

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

Table

Description automatically generated

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

**Sample Input**

Table

Description automatically generated

**Sample Output**

Isosceles

Equilateral

Scalene

Not A Triangle

for MySQL

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 A=C OR B=C THEN 'Isosceles'

ELSE 'Scalene'

END

FROM TRIANGLES;

for MySQL

SELECT

CASE

WHEN A+B>C AND B+C>A AND C+A>B THEN

CASE

WHEN A=B AND B=C THEN "Equilateral"

WHEN A=B OR B=C OR A=C THEN "Isosceles"

WHEN A<>B AND B<>C AND C<>A THEN "Scalene"

END

ELSE "Not A Triangle"

END

FROM TRIANGLES

14- 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:
3. 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: Table

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

**Sample Input**

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

Table

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.

for MySQL

SELECT CONCAT(NAME,'(',SUBSTR(OCCUPATION,1,1),')') AS A1 FROM OCCUPATIONS ORDER BY NAME;

SELECT CONCAT('There are a total of',' ',COUNT(OCCUPATION),' ',LOWER(OCCUPATION),'s.') AS B1 FROM OCCUPATIONS GROUP BY OCCUPATION ORDER BY B1;

14- Query the average population of all cities in **CITY** where District is **California**.

for MySQL

SELECT SUM(POPULATION)/COUNT(POPULATION) FROM CITY WHERE DISTRICT ="California"

15- Query the average population for all cities in **CITY**, rounded down to the nearest integer.

for MySQL

SELECT ROUND(SUM(POPULATION)/COUNT(ID)) FROM CITY

16- Query the sum of the populations for all Japanese cities in **CITY**. The COUNTRYCODE for Japan is **JPN**.

for MySQL

SELECT SUM(POPULATION) FROM CITY WHERE COUNTRYCODE='JPN'

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

for MySQL

SELECT MAX(POPULATION)-MIN(POPULATION) FROM CITY

18-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:

Table

Description automatically generated

**Note:** Salary is per month.

**Constraints**

.

**Sample Input**

Table

Description automatically generated

**Sample Output**

2061

**Explanation**

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

Table

Description automatically generated

for MySQL

SELECT CEIL(AVG(Salary)-AVG(REPLACE(Salary,'0',''))) FROM EMPLOYEES;

19-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:

Table

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**

Table

Description automatically generated

for MySQL

SELECT salary \* months AS earnings, COUNT(\*)

FROM Employee

GROUP BY earnings

ORDER BY earnings DESC

LIMIT 1;

select max(salary \* months), count(\*) from employee

where salary \* months = (select max(salary \* months) from employee);

20- Query the following two values from the **STATION** table:

1. The sum of all values in *LAT\_N* rounded to a scale of  decimal places.
2. The sum of all values in *LONG\_W* rounded to a scale of  decimal places.

for MySQL

SELECT ROUND(SUM(LAT\_N),2) , ROUND(SUM(LONG\_W),2) FROM STATION

21- 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  decimal places.

**Input Format**

for MySQL

SELECT ROUND(SUM(LAT\_N),4) FROM STATION WHERE LAT\_N > 38.7880 AND LAT\_N < 137.2345

22- Query the greatest value of the Northern Latitudes (LAT\_N) from **STATION** that is less than . 137.2345 Truncate your answer to 4  decimal places.

**Input Format**

for MySQL

SELECT ROUND(MAX(LAT\_N),4) FROM STATION WHERE LAT\_N < 137.2345

23- Query the Western Longitude (LONG\_W) for the largest Northern Latitude (LAT\_N) in **STATION** that is less than . Round your answer to  decimal places.

for MySQL

SELECT ROUND(LONG\_W,4) FROM STATION WHERE LAT\_N = (SELECT MAX(LAT\_N) FROM STATION WHERE LAT\_N < 137.2345)

24- Query the Western Longitude (LONG\_W)where the smallest Northern Latitude (LAT\_N) in **STATION** is greater than . Round your answer to  decimal places.

for MySQL

SELECT ROUND(LONG\_W,4) FROM STATION WHERE LAT\_N = (SELECT MIN(LAT\_N) FROM STATION WHERE LAT\_N >38.778)

25- Consider (a,b)  and (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**).
* d happens to equal the maximum value in *Western Longitude* (*LONG\_W* in **STATION**).

Query the [Manhattan Distance](https://xlinux.nist.gov/dads/HTML/manhattanDistance.html) between points  and  and round it to a scale of  4 decimal places.

for MySQL

SELECT ROUND( MAX(lat\_n)-MIN(lat\_n) + MAX(long\_w)-MIN(long\_w), 4) FROM Station;

26-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:

for MySQL

SELECT SUM(CITY.POPULATION)

FROM CITY , COUNTRY

WHERE CITY.COUNTRYCODE = COUNTRY.CODE

AND COUNTRY.CONTINENT='Asia'

27-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.

for MySQL

SELECT COUNTRY.CONTINENT, FLOOR(AVG(CITY.POPULATION))

FROM CITY INNER JOIN COUNTRY

ON CITY.COUNTRYCODE = COUNTRY.CODE

GROUP BY COUNTRY.CONTINENT;

**28 -Weather Observation Station 19**

for MySQL

SELECT ROUND(

SQRT(

POWER(MAX(LAT\_N) - MIN(LAT\_N), 2) +

POWER(MAX(LONG\_W) - MIN(LONG\_W), 2)

), 4)

FROM STATION;

for MySQL

for MySQL