SQL Practice

1)

SELECT , FROM , ORDER BY , TOP ,

-- Return unique countries and use an alias

SELECT

  DISTINCT country AS unique\_country

FROM

  eurovision;

2)

-- Return all columns, restricting the percent of rows returned

SELECT

  TOP (50) PERCENT \*

FROM

  eurovision;

3)

-- Return all columns, restricting the percent of rows returned

SELECT

  TOP (50) PERCENT \*

FROM

  eurovision;

4)

-- Select the top 20 rows from description, nerc\_region and event\_date

SELECT

  TOP (20) description,

  nerc\_region,

  event\_date

FROM

  grid

  -- Order by nerc\_region, affected\_customers & event\_date

  -- Event\_date should be in descending order

ORDER BY

  nerc\_region,

  affected\_customers,

  event\_date DESC;

5)

-- Select description and event\_year

SELECT

  description,

  event\_year

FROM

  grid

  -- Filter the results

where

  description= 'Vandalism';

6

-- Select nerc\_region and demand\_loss\_mw

SELECT

  nerc\_region,

  demand\_loss\_mw

FROM

  grid

-- Retrieve rows where affected\_customers is >= 500000  (500,000)

where

  affected\_customers >= 500000;

7

-- Select description and affected customers

SELECT

  description,

  affected\_customers

FROM

  grid

  -- Retrieve rows where the event\_date was the 22nd December, 2013

WHERE

  event\_date = '2013-12-22';

8

-- Select description, affected\_customers and event date

SELECT

  description,

  affected\_customers,

  event\_date

FROM

  grid

  -- The affected\_customers column should be >= 50000 and <=150000

WHERE

  affected\_customers BETWEEN 50000

  AND 150000

   -- Define the order

order by

  event\_date desc;

9

-- Retrieve all columns

SELECT

  \*

FROM

  grid

  -- Return only rows where demand\_loss\_mw is missing or unknown

WHERE

  demand\_loss\_mw is NULL;

10

-- Retrieve all columns

SELECT

  \*

FROM

  grid

  -- Return rows where demand\_loss\_mw is not missing or unknown

WHERE

  demand\_loss\_mw IS not NULL;

11

-- Retrieve all columns

SELECT

  \*

FROM

  grid

  -- Return rows where demand\_loss\_mw is not missing or unknown

WHERE

  demand\_loss\_mw IS not NULL;

12

-- Retrieve the song,artist and release\_year columns

SELECT

  song,

  artist,

  release\_year

FROM

  songlist

  -- Ensure there are no missing or unknown values in the release\_year column

WHERE

  release\_year IS NOT NULL

  -- Arrange the results by the artist and release\_year columns

order by

  artist,

  release\_year;

13

SELECT

  song,

  artist,

  release\_year

FROM

  songlist

WHERE

  -- Retrieve records greater than and including 1980

  release\_year >= 1980

  -- Replace AND with OR

  OR release\_year <= 1990

ORDER BY

  artist,

  release\_year;

14

SELECT

  artist,

  release\_year,

  song

FROM

  songlist

  -- Choose the correct artist and specify the release year

WHERE

  (

    artist LIKE 'B%'

    AND release\_year = 1986

  )

  -- Or return all songs released after 1990

  or release\_year > 1990

  -- Order the results

ORDER BY

  release\_year,

  artist,

  song;

15

MIN , MAX , COUNT , AVERAGE , DISTINCT , SUM

-- Obtain a count of 'grid\_id'

SELECT

  COUNT(grid\_id) AS RFC\_count

FROM

  grid

-- Restrict to rows where the nerc\_region is 'RFC'

where

  nerc\_region = 'RFC';

16

-- Find the average number of affected customers

SELECT

  AVG(affected\_customers) AS avg\_affected\_customers

FROM

  grid

-- Only retrieve rows where demand\_loss\_mw has a value

WHERE

  demand\_loss\_mw IS NOT NULL;

17

Strings, left, Right, char index, substring.

-- Calculate the length of the description column

SELECT

  LEN(description) AS description\_length

FROM

  grid;

18

-- Amend the query to select 25 characters from the  right of the description column

SELECT

  right(description, 25) AS last\_25\_right

FROM

  grid;

19

This is important .

-- Complete the substring function to begin extracting from the correct character in the description column

SELECT TOP (10)

  description,

  CHARINDEX('Weather', description) AS start\_of\_string,

  LEN ('Weather') AS length\_of\_string,

  SUBSTRING(

    description,

    15,

    LEN(description)

  ) AS additional\_description

FROM

  grid

WHERE description LIKE '%Weather%';

20 )

-- Select the region column

SELECT

  nerc\_region,

  -- Sum the demand\_loss\_mw column

  SUM(demand\_loss\_mw) AS demand\_loss

FROM

  grid

  -- Exclude NULL values of demand\_loss

WHERE

  demand\_loss\_mw IS NOT NULL

  -- Group the results by nerc\_region

Group BY

  nerc\_region

  -- Order the results in descending order of demand\_loss

ORDER BY

  demand\_loss desc;

21 )

SELECT

  nerc\_region,

  SUM (demand\_loss\_mw) AS demand\_loss

FROM

  grid

  -- Remove the WHERE clause

GROUP BY

  nerc\_region

  -- Enter a new HAVING clause so that the sum of demand\_loss\_mw is greater than 10000

HAVING

  SUM(demand\_loss\_mw) > 10000

ORDER BY

  demand\_loss DESC;

22)

-- Obtain a count for each country

SELECT

  count(country) AS country\_count,

  -- Retrieve the country column

  country,

  -- Return the average of the Place column

  AVG(place) AS average\_place,

  AVG(points) AS avg\_points,

  MIN(points) AS min\_points,

  MAX(points) AS max\_points

FROM

  eurovision

GROUP BY

  country;

23 )

select count(city) + count(distinct city) from station ;

>> WE can do addition and subtraction with count variable .

Select \* from country where mod(id,2)=0

>> get rows with id = even integer .

select CITY,LENGTH(CITY) from STATION order by Length(CITY) asc, CITY limit 1;

select CITY,LENGTH(CITY) from STATION order by Length(CITY) DESC, CITY limit 1;

24)

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

Table

Description automatically generated

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' ;

25)

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.

Select DISTINCT CITY

FROM STATION

WHERE (CITY LIKE '%A' OR CITY LIKE '%E' OR CITY LIKE '%I' OR CITY LIKE '%O' OR CITY LIKE '%U')AND

(CITY LIKE 'A%' OR CITY LIKE 'E%' OR CITY LIKE 'I%' OR CITY LIKE 'O%' OR CITY LIKE 'U%')

26) ///// **NOT LIKE**

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

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%';

\*/ String match / mismatch is case sensitive . /\*

27 )

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

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';

28)

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.

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%') ;

29)

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.

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%') ;

30)

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.

Table

Description automatically generated

SELECT NAME

FROM STUDENTS

WHERE MARKS > 75

ORDER BY RIGHT(NAME,3), ID;

31)

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

Table

Description automatically generated

SELECT NAME

FROM EMPLOYEE

order by Name ;

32)

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.

Table

Description automatically generated

SELECT NAME

FROM EMPLOYEE

WHERE SALARY > '2000' AND months <'10'

ORDER BY employee\_id;

33)

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:

Table

Description automatically generated

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  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.

>>>

select CITY,LENGTH(CITY) from STATION order by Length(CITY) asc, CITY limit 1;

select CITY,LENGTH(CITY) from STATION order by Length(CITY) desc, CITY limit 1;

.>>>

select top 1 city, len(city) from station order by len(city) ASC, city ASC;

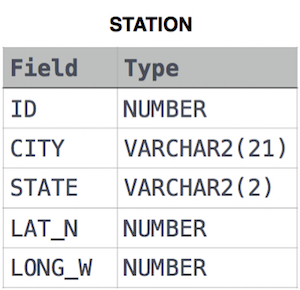
select top 1 city, len(city) from station order by len(city) DESC, city ASC;

34 ))

Query the smallest Northern Latitude (LAT\_N) from **STATION** that is greater than . Round your answer to  decimal places.

**Input Format**

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



>>

select cast(min(lat\_n) as decimal(6,4)) from station where lat\_n>38.7780;

>>

select cast(lat\_n as decimal(6,4)) from station where lat\_n>38.7780 order by lat\_n ,LAT\_N limit 1;

35)

Query the Western Longitude (LONG\_W)where the smallest Northern Latitude (LAT\_N) 38 .7790 in **STATION** is greater than .

>>>>>>>

select cast(long\_w as decimal(6,4)) from station where lat\_n > 38.7780 order by lat\_n , long\_w limit 1;

36)

Consider  p1(a,b)and p2(c,d)  to be two points on a *2D* plane.

* happens to equal the minimum value in *Northern Latitude* (*LAT\_N* in **STATION**).
* happens to equal the minimum value in *Western Longitude* (*LONG\_W* in **STATION**).
* happens to equal the maximum value in *Northern Latitude* (*LAT\_N* in **STATION**).
* 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 p1 and p2 and round it to a scale of 4  decimal places.

>>>

select cast(abs(max(long\_w)-min(long\_w))+abs(max(lat\_n)-min(lat\_n)) as decimal(10,4)) from station;

37 )

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

**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 .

​

select case

          when A + B > C and B + C > A and A + C >B THEN

                CASE WHEN A=B and B=C  THEN 'Equilateral'

                     WHEN A=B or B=C  or A=C then 'Isosceles'

                     else 'Scalene'

                END

                     else 'Not A Triangle '

          END

from triangles

​

38 )

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, application

Description automatically generated

select CONCAT(Name,'(',left(Occupation,1),')')

from Occupations

order by NAME asc;

select Concat( "There are a total of ",COUNT(Occupation)," ",lower(occupation),"s.") as T

from occupations

group by occupation

order by T asc, occupation

39)

**JOINS ::**

We can join 3 tables using join statement .

**Select**

Table\_A.columnX,

Table\_A.columnX,

Table\_A.columnX,

**From** table\_A

**Inner Join** table\_b **ON** table\_B.foreign\_key= table\_A.primary\_key

**Inner Join** table\_C **ON** table\_C.foreign\_key=table\_B.primary\_key

Graphical user interface, text, application, email

Description automatically generated

SELECT track.track\_id,

-- Enter the correct table name prefix when retrieving the name column from the track table

  track.name AS track\_name,

  album.title as album\_title,

  -- Enter the correct table name prefix when retrieving the name column from the artist table

  artist.name AS artist\_name

FROM track

  -- Complete the matching columns to join album with track, and artist with album

INNER JOIN album on track.album\_id = album.album\_id

INNER JOIN artist on album.artist\_id = artist.artist\_id;

A picture containing table

Description automatically generated

# UNION & UNION ALL

Graphical user interface

Description automatically generated

Graphical user interface

Description automatically generated

Graphical user interface, text

Description automatically generated

Graphical user interface, text

Description automatically generated

Graphical user interface, text, application

Description automatically generated

SELECT

  album\_id AS ID,

  title AS description,

  'Album' AS Source

  -- Complete the FROM statement

FROM Album

 -- Combine the result set using the relevant keyword

union

SELECT

  artist\_id AS ID,

  name AS description,

  'Artist'  AS Source

  -- Complete the FROM statement

from  Artist;

-- Create the table

CREATE TABLE results (

  -- Create track column

  track VARCHAR(200),

    -- Create artist column

  artist VARCHAR(120),

    -- Create album column

  album VARCHAR(160),

  -- Create track\_length\_mins

  track\_length\_mins INT,

  );

-- Select all columns from the table

SELECT

  track,

  artist,

  album,

  track\_length\_mins

FROM

  results;

Insert with conditions in SQL :::: IT IS POSSIBLE , This can be used to fill empty values according to my knowledge .

Graphical user interface, text, application

Description automatically generated

UPDATE A COLUMN VALUE USING WHERE CLAUSE .

Graphical user interface, application

Description automatically generated

DELETE WITH CONDITIONS.

Graphical user interface, application

Description automatically generated

-- Create the table

CREATE TABLE tracks(

  -- Create track column

  track VARCHAR(200),

  -- Create album column

  album VARCHAR(160),

  -- Create track\_length\_mins column

  track\_length\_mins INT

);

-- Complete the statement to enter the data to the table

Insert INTO tracks

-- Specify the destination columns

(track, album, track\_length\_mins)

-- Insert the appropriate values for track, album and track length

VALUES

  ('Basket Case', 'Dookie', 3);

-- Select all columns from the new table

SELECT

  \*

FROM

  tracks;

UPDATE Example :

-- Select the album

SELECT

  title

FROM

  album

WHERE

  album\_id = 213;

-- UPDATE the title of the album

UPDATE

  album

SET

  title = 'Pure Cult: The Best Of The Cult'

WHERE

  album\_id = 213;

-- Run the query again

SELECT

  title

FROM

  album

WHERE

  album\_id = 213;

**DELETE EXAMPLE :**

-- Run the query

SELECT

  \*

FROM

  album

  -- DELETE the record

DELETE FROM

  album

WHERE

  album\_id = 1

  -- Run the query again

SELECT

  \*

FROM

  album;

**Use of Variables in SQL with @ .**Graphical user interface, application

Description automatically generated

Creating Temporary Tables :

Graphical user interface, application

Description automatically generated

-- Declare the variable @region

DECLARE @region VARCHAR(10)

-- Update the variable value

SET @region = 'RFC'

SELECT description,

       nerc\_region,

       demand\_loss\_mw,

       affected\_customers

FROM grid

WHERE nerc\_region = @region;

-- Declare your variables

DECLARE @start DATE

DECLARE @stop DATE

DECLARE @affected INT;

-- SET the relevant values for each variable

SET @start = '2014-01-24'

SET @stop  = '2014-07-02'

SET @affected =  5000 ;

SELECT

  description,

  nerc\_region,

  demand\_loss\_mw,

  affected\_customers

FROM

  grid

-- Specify the date range of the event\_date and the value for @affected

WHERE event\_date between @start AND @stop

AND affected\_customers >= @affected;

SELECT  album.title AS album\_title,

  artist.name as artist,

  MAX(track.milliseconds / (1000 \* 60) % 60 ) AS max\_track\_length\_mins

-- Name the temp table #maxtracks

INTO #maxtracks

FROM album

-- Join album to artist using artist\_id

INNER JOIN artist ON album.artist\_id = artist.artist\_id

-- Join track to album using album\_id

INNER JOIN track ON album.album\_id = track.album\_id

GROUP BY artist.artist\_id, album.title, artist.name,album.album\_id

-- Run the final SELECT query to retrieve the results from the temporary table

SELECT album\_title, artist, max\_track\_length\_mins

FROM  #maxtracks

ORDER BY max\_track\_length\_mins DESC, artist.

Text

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

**------**

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

**Input Format**

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

**select floor(avg(population))**

**from city**

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

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.

**SELECT CAST(CEILING((AVG(CAST(Salary AS Float)) - AVG(CAST(REPLACE(Salary, 0, '')AS Float)))) AS INT)**

**FROM EMPLOYEES;**

**==== ALWAYS USE CAST FOR CONVERSION – FROM INT TO DECIMAL , DECIMAL TO INT ETC .**

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

**Sample Output**

69952 1

**Explanation**

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

Description automatically generated

The maximum *earnings* value is 69952 . The only employee with *earnings* = 69952  is *Kimberly*, so we print the maximum *earnings* value (69952 ) and a count of the number of employees who have earned  69952  (which is 1 ) as two space-separated values.

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

**from employee**

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