Hacker rank queries

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

SQL:

```
Select distinct city from station where lower(substr(city,1,1)) in ('a','e','i','o','u') ;
```

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

SQL:

```
Select distinct city from station where lower(substr(city,-1)) in ('a','e','i','o','u') ;
```

Q. 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.

SQL:

```
Select distinct city from station where regexp_like(lower(city), '^[aeiou]') and  regexp_like(lower(city), '[aeiou]$');
```

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

SQL:

```
Select distinct city from station where lower(substr(city,1,1)) not in ('a','e','i','o','u') ;
```

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

SQL:

```
Select distinct city from station where lower(substr(city,-1)) not in ('a','e','i','o','u') ;
```

Q. 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.

SQL:

```
SELECT DISTINCT city

FROM station

WHERE regexp_like (city, '^[^aeiouAEIOU].*')

OR regexp_like (city, '.*[^aeiouAEIOU]$');
```

Q. 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.

SQL:

```
select distinct city from station where city not like '%a'

and city not like '%e'

and city not like '%u'

and city not like '%i'

and city not like '%o'

and city not like 'a%'

and city not like 'e%'

and city not like 'i%'

and city not like 'u%'

and city not like 'o%'

and city not like '%A'

and city not like '%E'

and city not like '%U'

and city not like '%I'

and city not like '%O'

and city not like 'A%'

and city not like 'E%'

and city not like 'I%'

and city not like 'U%'

and city not like 'O%';
```

Q. 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*.

SQL:

```
SELECT Name FROM Students WHERE Marks > 75 ORDER BY substr(Name, -3), ID;
```

Q. Julia conducted a 15 days of learning SQL contest. The start date of the contest was *March 01, 2016* and the end date was *March 15, 2016*.

Write a query to print total number of unique hackers who made at least submission each day (starting on the first day of the contest), and find the *hacker_id* and *name* of the hacker who made maximum number of submissions each day. If more than one such hacker has a maximum number of submissions, print the lowest *hacker_id*. The query should print this information for each day of the contest, sorted by the date.

Sql:

```
Select big_1.submission_date, big_1.hkr_cnt, big_2.hacker_id, h.name
From
(select submission_date, count(distinct hacker_id) as hkr_cnt
From
(select s.*, dense_rank() over(order by submission_date) as date_rank,
Dense_rank() over(partition by hacker_id order by submission_date) as hacker_rank
From submissions s ) a
Where date_rank = hacker_rank
Group by submission_date) big_1
Join
(select submission_date,hacker_id,
Rank() over(partition by submission_date order by sub_cnt desc, hacker_id) as max_rank
From (select submission_date, hacker_id, count(*) as sub_cnt
From submissions
Group by submission_date, hacker_id) b ) big_2
On big_1.submission_date = big_2.submission_date and big_2.max_rank = 1
Join hackers h on h.hacker_id = big_2.hacker_id
Order by 1;
```

Q. 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.

SQL:

```
SELECT months*salary,COUNT(*) FROM Employee GROUP BY months*salary having count(*) = 7
ORDER BY months*salary DESC;
```

Q. 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.

SQL:

```
select round(sum(lat_n),2) as lat ,round(sum(long_w),2) as lon from station;
```

Q. 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.

SQL:

```
select trunc(sum(lat_n),4) from station where lat_n between 38.7880 and 137.2345;
```

Q. Query the *Western Longitude* (*LONG_W*) for the largest *Northern Latitude* (*LAT_N*) in **STATION** that is less than . Round your answer to 4 decimal places.

SQL:

```
select round(long_w,4) from station where lat_n= (select max(lat_n) from station where lat_n
<137.2345);
```

Q. 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**).
- *d h*appens to equal the maximum value in *Western Longitude* (*LONG_W* in **STATION**).

Query the Manhattan Distance between points *P1* and *P2* and round it to a scale of 4 decimal places.

SQL:

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

FROM STATION;
```

Q. 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.

SQL:

```
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'
    WHEN A <> B and B <> C THEN 'Scalene'
  END tuple
FROM TRIANGLES;
```

Q. 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.

SQL:

```
1 .select name||'('||substr(occupation,1,1)||')' from occupations order by name;
select 'there are total of '||count(occupation)||' '||occupation||'s'
from occupations
group by occupation;                                        --my solution/wrong
2 .SELECT Name||'('|| SUBSTR(Occupation,1,1)||')' FROM OCCUPATIONS ORDER BY Name;
SELECT 'There are a total of '|| COUNT(Occupation)|| ' '||LOWER(Occupation)|| 's.' FROM
OCCUPATIONS
GROUP BY Occupation
ORDER BY COUNT(Occupation), Occupation;          -- google solution/correct
```

Q. 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.

SQL:

```
1 .select nvl(doctor.name,'NULL'), nvl(professor.name,'NULL'), nvl(singer.name,'NULL'),
nvl(actor.name,'NULL') from

(

   (select row_number() over(order by Name) as rn, Name from Occupations where
Occupation='Doctor' order by Name) doctor

full outer join

   (select row_number() over(order by Name) as rn, Name from Occupations where
Occupation='Professor' order by Name) professor on (doctor.rn=professor.rn)

full outer join

   (select row_number() over(order by Name) as rn, Name from Occupations where
Occupation='Singer' order by Name) singer on (professor.rn=singer.rn)

full outer join

   (select row_number() over(order by Name) as rn, Name from Occupations where
Occupation='Actor' order by Name) actor on (singer.rn=actor.rn) );
```

```
2 .SELECT * FROM

(SELECT Doctor,Professor,Singer,Actor FROM

(SELECT NAME,OCCUPATION, ROW_NUMBER() OVER (PARTITION BY OCCUPATION ORDER BY NAME)

FROM Occupations)

PIVOT

( MAX(Name) FOR (Occupation) IN ('Doctor' as Doctor,'Professor' as Professor,'Singer' as
Singer,'Actor' as Actor))

)

ORDER BY Doctor ASC NULLS LAST,Professor ASC NULLS LAST,Singer ASC NULLS LAST,Actor ASC
NULLS LAST ;
```

Q. 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.

SQL:

```
Select distinct c.n,
       case when c.P is null then 'Root'
           when b1.N is null then 'Leaf'
           else 'Inner' end
from BST c
left join BST b1
on c.N = b1.P
order by c.n;
```

Q. 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*.

SQL:

```
1 .  select c.company_code, c.founder,              --my try
count(distinct l.lead_manager_code),
count(distinct s.senior_manager_code),
count(distinct m.manager_code),
count(distinct e.employee_code)
from company c join lead_manager l
on (l.company_code=c.company_code)
join
senior_manager s
on (l.lead_manager_code=s.lead_manager_code)
join
manager m
```

on(s.senior_manager_code=m.senior_manager_code)

join

employee e

on (m.manager_code=e.manager_code)

group by c.company_code,c.founder

order by c.company_code;

2 . select c.company_code,c.founder,   --online search

count(distinct e.lead_manager_code),

count(distinct e.senior_manager_code),

count(distinct e.manager_code),

count(distinct e.employee_code)

from company c join employee e

on (c.company_code=e.company_code)

group by c.company_code,c.founder

order by c.company_code;

SQL:

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

/ CEIL is used to get next integer in case of decimal, it is same as round function /

Q. Query the Euclidean Distance between points P1 and P2 and *format your answer* to display decimal digits.

SQL:

Select round(sqrt(power((min(lat_n)-max(lat_n)),2) + power((min(long_w)-max(long_w)),2)),4) From station;

*/ sqrt : square root

Power : square

Euclidean Distance formula link: https://en.wikipedia.org/wiki/Euclidean_distance#:~:text=In%20mathematics%2C%20the%20Euclidean%20distance,occasionally%20called%20the%20Pythagorean%20distance.

/*

Q: Get median of Lat_n.

SQL:

```
select round( median(lat_n),4) from station;
```

Q. Samantha interviews many candidates from different colleges using coding challenges and contests. Write a query to print the *contest_id*, *hacker_id*, *name*, and the sums of *total_submissions*, *total_accepted_submissions*, *total_views*, and *total_unique_views* for each contest sorted by *contest_id*. Exclude the contest from the result if all four sums are .

SQL: --my try

```
select c.contest_id,c.hacker_id,c.name,sum(s.total_submissions),
sum(s.total_accepted_submissions),sum(v.total_views),
sum(v.total_unique_views)
from contests c join colleges cl
on (c.contest_id = cl.contest_id)
join challenges ch
on (cl.college_id =ch.college_id)
join view_stats v
on (ch.challenge_id=v.challenge_id)
join submission_stats s
on (v.challenge_id = s.challenge_id)
group by c.contest_id,c.hacker_id,c.name
having
(sum(s.total_submissions)+
sum(s.total_accepted_submissions)+ sum(v.total_views)+
sum(v.total_unique_views))>0;
```

--Online search

```
SELECT CON.CONTEST_ID,

       CON.HACKER_ID,

       CON.NAME,

       SUM(TOTAL_SUBMISSIONS),

       SUM(TOTAL_ACCEPTED_SUBMISSIONS),

       SUM(TOTAL_VIEWS),

       SUM(TOTAL_UNIQUE_VIEWS)
```

```

FROM CONTESTS CON

JOIN COLLEGES COL ON CON.CONTEST_ID = COL.CONTEST_ID

JOIN CHALLENGES CHA ON COL.COLLEGE_ID = CHA.COLLEGE_ID

LEFT JOIN

  (SELECT CHALLENGE_ID,

          SUM(TOTAL_VIEWS) AS TOTAL_VIEWS,

          SUM(TOTAL_UNIQUE_VIEWS) AS TOTAL_UNIQUE_VIEWS

   FROM VIEW_STATS

   GROUP BY CHALLENGE_ID) VS ON CHA.CHALLENGE_ID = VS.CHALLENGE_ID

LEFT JOIN

  (SELECT CHALLENGE_ID,

          SUM(TOTAL_SUBMISSIONS) AS TOTAL_SUBMISSIONS,

          SUM(TOTAL_ACCEPTED_SUBMISSIONS) AS TOTAL_ACCEPTED_SUBMISSIONS

   FROM SUBMISSION_STATS

   GROUP BY CHALLENGE_ID) SS ON CHA.CHALLENGE_ID = SS.CHALLENGE_ID

GROUP BY CON.CONTEST_ID,

         CON.HACKER_ID,

         CON.NAME

HAVING SUM(TOTAL_SUBMISSIONS) != 0

OR SUM(TOTAL_ACCEPTED_SUBMISSIONS) != 0

OR SUM(TOTAL_VIEWS) != 0

OR SUM(TOTAL_UNIQUE_VIEWS) != 0

ORDER BY CONTEST_ID;
```

Q. 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.

SQL:

1. select cn.continent,ceil(avg(c.population)) from city c join country cn   --my try

on (c.countrycode = cn.code)

group by cn.continent;

2. select cn.continent,floor(avg(c.population)) from city c join country cn   --online search
   on (c.countrycode = cn.code)
   group by cn.continent;

Q. *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.

SQL:

1. select grade,name,marks from (select grade,name,marks   --my try

from students join grades

on marks between min_mark and max_mark

order by grade desc,name,marks)

where (case

    when grade > 8 then name

    when grade <= 8 then null

    else name

   end)=name;

2. select case when (s.marks < 70) then 'null' else s.name end, --my try

    g.grade,s.marks

from students s, grades g

where s.marks >= g.min_mark and s.marks <= g.max_mark

order by g.grade desc,s.name asc;

```
3. SELECT CASE WHEN (STUDENTS.MARKS < 70) THEN 'NULL' ELSE STUDENTS.NAME END,    --online

       GRADES.GRADE, STUDENTS.MARKS

FROM STUDENTS, GRADES

WHERE STUDENTS.MARKS >= GRADES.MIN_MARK AND STUDENTS.MARKS <= GRADES.MAX_MARK

ORDER BY GRADES.GRADE DESC, STUDENTS.NAME ASC;
```

Q. 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*.SQL: -- online search

```
SELECT h.hacker_id, h.name FROM Submissions s JOIN Hackers h ON s.hacker_id = h.hacker_id

                    JOIN Challenges c ON s.challenge_id = c.challenge_id

                    JOIN Difficulty d ON c.difficulty_level = d.difficulty_level

WHERE s.score = d.score

GROUP BY h.hacker_id, h.name HAVING COUNT(*)>1 ORDER BY COUNT(*) DESC, h.hacker_id;
```

Q. 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*.

SQL: --my try not working

```
1 .select w.id,max(wp.age),min(w.coins_needed),max(w.power) from  wands w join wands_property wp

on w.code = wp.code

where is_evil <> 1

group by wp.age,w.power,w.id

order by w.power asc,wp.age desc;
```

--online search

2 . SELECT aa.id, bb.age, aa.coins_needed, aa.power

FROM WANDS aa

JOIN WANDS_PROPERTY bb ON aa.CODE = bb.CODE

JOIN (SELECT age AG, MIN(coins_needed) MCN, power PW

FROM WANDS A

JOIN WANDS_PROPERTY B ON A.CODE = B.CODE

WHERE IS_EVIL = 0

GROUP BY power, age) Q ON bb.age = AG AND aa.coins_needed = MCN AND aa.power = PW

ORDER BY aa.power DESC, bb.age DESC;

Q. 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.

SQL: --my try

1. select h.hacker_id,name,count(challenge_id) from hackers h join challenges c

on (h.hacker_id=c.hacker_id)

group by h.hacker_id,name

order by count(challenge_id)desc,hacker_id desc;

2. --from submissions

```
SELECT HACKER_ID,NAME,TOT_CHAL

FROM

(

  SELECT HACKER_ID,NAME,COUNT(DISTINCT CHALLENGE_ID) AS TOT_CHAL

  FROM

  (

    SELECT A.HACKER_ID,NAME,CHALLENGE_ID

    FROM HACKERS A,CHALLENGES B

    WHERE A.HACKER_ID=B.HACKER_ID
  )
  GROUP BY HACKER_ID,NAME
)
WHERE TOT_CHAL NOT IN
(
  SELECT TOT_CHAL
  FROM
  (
    SELECT TOT_CHAL
    FROM
    (
      SELECT HACKER_ID,NAME,COUNT(DISTINCT CHALLENGE_ID) AS TOT_CHAL
      FROM
      (
      SELECT A.HACKER_ID,NAME,CHALLENGE_ID
      FROM HACKERS A,CHALLENGES B
      WHERE A.HACKER_ID=B.HACKER_ID
      )
      GROUP BY HACKER_ID,NAME
    )
    GROUP BY TOT_CHAL
    HAVING COUNT(HACKER_ID) > 1
  )
  WHERE
  TOT_CHAL <
  (
    SELECT MAX(TOT_CHAL) AS MAX_CHAL FROM
    (
      SELECT HACKER_ID,NAME,COUNT(DISTINCT CHALLENGE_ID) AS TOT_CHAL
      FROM
      (
            SELECT A.HACKER_ID,NAME,CHALLENGE_ID
            FROM HACKERS A,CHALLENGES B
            WHERE A.HACKER_ID=B.HACKER_ID
            )
            GROUP BY HACKER_ID,NAME
        )
    )
)
ORDER BY TOT_CHAL DESC,HACKER_ID;
```

Q.

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 from your result.

SQL: --my try didn’t worked.

1. select h.hacker_id,h.name,sum(s.score) from hackers h join submissions s on h.hacker_id = s.hacker_id

group by h.hacker_id,h.name

having sum(s.score) <> 0

order by sum(s.score) desc;

2. --my try

select h.hacker_id,h.name,max(s.score) from hackers h

join submissions s

on h.hacker_id=s.hacker_id

group by h.hacker_id,h.name

order by max(s.score) desc,h.hacker_id asc;

3. --online search

SELECT m.hacker_id, h.name, SUM(m.score) FROM

(SELECT hacker_id, challenge_id, MAX(score) score FROM Submissions GROUP BY hacker_id, challenge_id) m

```
JOIN Hackers h ON m.hacker_id = h.hacker_id

GROUP By m.hacker_id, h.name

HAVING sum(m.score) > 0

ORDER BY sum(m.score) DESC, m.hacker_id;
```

Q.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.

SQL:

```
SELECT s.Name FROM Students s

JOIN Packages sp ON s.ID = sp.ID    /* get names from students */

JOIN Friends f ON s.ID = f.ID

JOIN Packages fp ON f.Friend_ID = fp.ID /* */

WHERE sp.Salary < fp.Salary

ORDER BY fp.Salary;
```

Q. 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.

SQL:

```
select a,b from

(select min(start_date) a ,max(end_date) b,max(end_date) - min(start_date) c from

 (select a.start_date,a.end_date,row_number() over(order by start_date) rn from

  (select * from

   (select start_date,end_date,lag(end_date) over(order by start_date) led from Projects)a

  where  nvl(led,sysdate+10000) <> start_date)a

union all

  select a.start_date,a.end_date,row_number() over(order by start_date) rn from

  (select * from

     (select start_date,end_date,lead(start_date) over(order by start_date) lsd from Projects)a

where nvl(lsd,sysdate+10000) <> end_date)a ) group by rn )

order by c,1;
```

Q. Two pairs *(X_1, Y_1)* and *(X_2, Y_2)* are said to be *symmetric pairs* if $X_1 = Y_2$ and $X_2 = Y_1$.

Write a query to output all such *symmetric pairs* in ascending order by the value of *X*. List the rows such that $X_1 \leq Y_1$.

SQL:

```
SELECT f1.X, f1.Y FROM Functions f1
WHERE f1.X = f1.Y GROUP BY f1.X, f1.Y HAVING COUNT(*) > 1
UNION
SELECT f1.X, f1.Y FROM Functions f1
WHERE EXISTS(SELECT X, Y FROM Functions WHERE f1.X = Y AND f1.Y = X AND f1.X < X)
ORDER BY X;
```

Q. Write a query to print the *contest_id*, *hacker_id*, *name*, and the sums of *total_submissions*, *total_accepted_submissions*, *total_views*, and *total_unique_views* for each contest sorted by *contest_id*. Exclude the contest from the result if all four sums are 0 .

Note: A specific contest can be used to screen candidates at more than one college, but each college only holds 1 screening contest.

SQL: -- my try

```
select c.contest_id,c.hacker_id,c.name,
   sum(ss.total_submissions),
   sum(ss.total_accepted_submissions),
   sum(vs.total_views),
   sum(vs.total_unique_views)
   from contests c
join colleges cl on c.contest_id=cl.contest_id
join challenges ch on cl.college_id=ch.college_id
join view_stats vs on ch.challenge_id=vs.challenge_id
join submission_stats ss on ss.challenge_id = ch.challenge_id
group by c.contest_id,c.hacker_id,c.name
having
```

```
    sum(ss.total_submissions)<>0 or
    sum(ss.total_accepted_submissions)<>0 or
    sum(vs.total_views)<>0 or
    sum(vs.total_unique_views) <>0
order by c.contest_id;


--answer from submissions
 WITH views (contest_id, total_views, total_unique_views) AS (
    SELECT contest_id, SUM(total_views), SUM(total_unique_views)
    FROM Contests
    JOIN Colleges USING(contest_id)
    JOIN Challenges USING(college_id)
    JOIN View_Stats USING(challenge_id)
    GROUP BY contest_id
), subs (contest_id, total_submissions, total_accepted_submissions) AS (
    SELECT contest_id, SUM(total_submissions), SUM(total_accepted_submissions)
    FROM Contests
    JOIN Colleges USING(contest_id)
    JOIN Challenges USING(college_id)
    JOIN Submission_Stats USING(challenge_id)
    GROUP BY contest_id
)
SELECT contest_id, hacker_id, name, total_submissions, total_accepted_submissions,
total_views, total_unique_views   FROM Contests
LEFT JOIN views USING(contest_id)
LEFT JOIN subs USING(contest_id)
WHERE
total_submissions + total_accepted_submissions + total_views + total_unique_views > 0
ORDER BY contest_id ASC;
```

Q. Write a query to print the pattern *P(20)*

SQL: -- my try got output but test case failed

```
create or replace procedure rep_str (str1 in varchar2)
is
str varchar2(100);
begin
  for i in reverse 1..20 loop
    for j in reverse 1..i loop
     str:=str||' '||str1;
     end loop;
     dbms_output.put_line(str);
     str:= null;
     end loop;
end rep_str;

exec rep_str('*');
```

--online

```
SELECT SUBSTR(RPAD('* ', 40, '* '), 1, (42 - LEVEL * 2))

FROM DUAL

CONNECT BY LEVEL <= 20;
```

Q. Write a query to print the pattern *P(20)*

SQL:

```
select rpad('* ', level * 2, '* ')

  from dual connect by

    level <= 20;
```

Q. 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).

SQL:

```
WITH NUM AS (
   SELECT LEVEL N
   FROM DUAL CONNECT BY LEVEL <= 1000
)
SELECT LISTAGG(B.N,'&') WITHIN GROUP(ORDER BY B.N) AS PRIMES
FROM (
   SELECT  N,
         CASE WHEN EXISTS (
                        SELECT NULL
                        FROM NUM N_INNER
                        WHERE N_INNER .N > 1
                        AND N_INNER.N < NUM.N
                        AND MOD(NUM.N, N_INNER.N)=0
                     ) THEN
            'NO PRIME'
         ELSE
            'PRIME'
         END IS_PRIME
      FROM NUM
   ) B
WHERE B.IS_PRIME='PRIME'
AND B.N!=1;
```