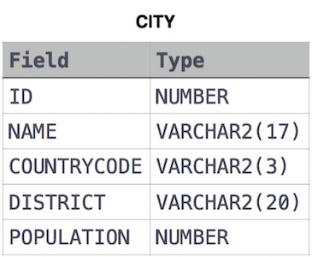
SQL Problems

1. 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:



**Sol.**

**SELECT \***

**FROM CITY**

**WHERE POPULATION > 100000 AND COUNTRYCODE = "USA";**

2. 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:



**Sol.**

**SELECT name FROM CITY WHERE COUNTRYCODE = 'USA' AND POPULATION > 120000;**

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.  
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

**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.

**Sol.**

**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;**

4. 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.

**Sol.**

**SELECT DISTINCT(city) FROM station WHERE city LIKE '[a,e,i,o,u]%[a,e,i,o,u]';**

5. 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:



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

**Sol.**

**SELECT NAME FROM EMPLOYEE ORDER BY NAME;**

6. 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:



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**



**Sample Output**

Angela

Michael

Todd

Joe

**Explanation**

*Angela* has been an employee for  month and earns  per month.

*Michael* has been an employee for  months and earns  per month.

*Todd* has been an employee for  months and earns  per month.

*Joe* has been an employee for  months and earns  per month.

We order our output by ascending *employee\_id*.

**Sol.**

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

7. 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:



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

**Sample Input**



**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 .

**Sol.**

**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;**

8. 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:



*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.

**Explanation**

The results of the first query are formatted to the problem description's specifications.  
The results of the second query are ascendingly ordered first by number of names corresponding to each profession (), and then alphabetically by profession (, and ).

**Sol.**

**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 ASC;**

9. [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:



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

**Sample Input**



**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.

**Sol.**

**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 RankedOccupations GROUP BY rn;**

10. 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*.



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**



**Sample Output**

1 Leaf

2 Inner

3 Leaf

5 Root

6 Leaf

8 Inner

9 Leaf

**Explanation**

The *Binary Tree* below illustrates the sample:



**Sol.**

**SELECT N,**

**CASE**

**WHEN P IS NULL THEN 'Root'**

**WHEN N IN (SELECT P FROM BST) THEN 'Inner'**

**ELSE 'Leaf'**

**END**

**FROM BST**

**ORDER by N;**

11. 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. 
* *Lead\_Manager:* The *lead\_manager\_code* is the code of the lead manager, and the *company\_code* is the code of the working company.

* *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

. 

* *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. 

**Sample Input**

*Company* Table:



*Lead\_Manager* Table:



*Senior\_Manager* Table:



*Manager* Table:



*Employee* Table:



**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*.

**Sol.**

**SELECT c.company\_code, c.founder, 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;**

12. Consider  and  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  and  and round it to a scale of  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.

**Sol.**

**SELECT c.company\_code, c.founder, 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;**

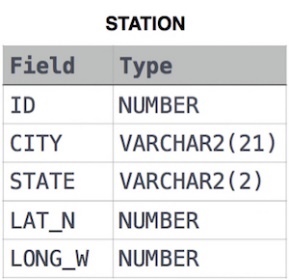
12. Consider  and  to be two points on a 2D plane.

* 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  and  and round it to a scale of  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.

**Sol.**

**SELECT ROUND((MAX(LAT\_N) + MAX(LONG\_W)) - (MIN(LAT\_N) + MIN(LONG\_W)), 4) FROM STATION;**

13. Consider  and  to be two points on a 2D plane where  are the respective minimum and maximum values of Northern Latitude (LAT\_N) and  are the respective minimum and maximum values of Western Longitude (LONG\_W) in **STATION**.

Query the [Euclidean Distance](https://en.wikipedia.org/wiki/Euclidean_distance) between points  and  and format your answer to display  decimal digits.

**Input Format**

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



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

**Sol.**

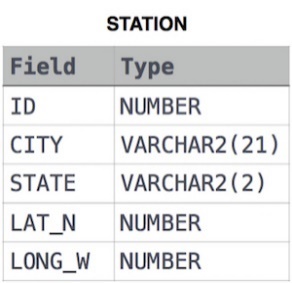
**select cast(sqrt(square(min(lat\_n)-max(lat\_n)) + square(min(long\_w)-max(long\_w))) as decimal(18,4))**

**from station;**

14. A [*median*](https://en.wikipedia.org/wiki/Median) is defined as a number separating the higher half of a data set from the lower half. Query the median of the Northern Latitudes (LAT\_N) from **STATION** and round your answer to  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.

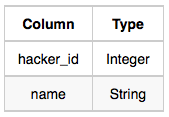
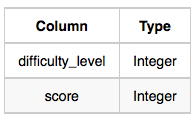
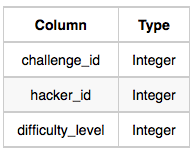
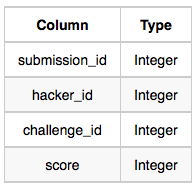
**Sol.**

**SELECT ROUND(s.lat\_n, 4) FROM station s WHERE ( SELECT COUNT(lat\_n) FROM station WHERE s.lat\_n > lat\_n ) = ( SELECT COUNT(lat\_n) FROM station WHERE s.lat\_n < lat\_n );**

15. 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.

**Input Format**

The following tables contain contest data:

* Hackers: The hacker\_id is the id of the hacker, and name is the name of the hacker. 
* Difficulty: The difficult\_level is the level of difficulty of the challenge, and score is the score of the challenge for the difficulty level. 
* Challenges: The challenge\_id is the id of the challenge, the hacker\_id is the id of the hacker who created the challenge, and difficulty\_level is the level of difficulty of the challenge. 
* Submissions: The submission\_id is the id of the submission, hacker\_id is the id of the hacker who made the submission, challenge\_id is the id of the challenge that the submission belongs to, and score is the score of the submission. 

**Sol.**

**SELECT HA.HACKER\_ID, HA.NAME**

**FROM SUBMISSIONS SU**

**INNER JOIN CHALLENGES CH**

**ON SU.CHALLENGE\_ID = CH.CHALLENGE\_ID**

**INNER JOIN DIFFICULTY DI**

**ON CH.DIFFICULTY\_LEVEL = DI.DIFFICULTY\_LEVEL**

**JOIN HACKERS HA**

**ON SU.HACKER\_ID = HA.HACKER\_ID**

**WHERE SU.SCORE = DI.SCORE**

**GROUP BY HA.HACKER\_ID, HA.NAME, SU.HACKER\_ID**

**HAVING COUNT(SU.HACKER\_ID) > 1**

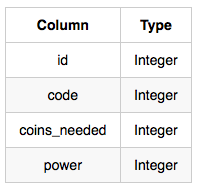
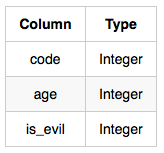
**ORDER BY COUNT(SU.HACKER\_ID) DESC, SU.HACKER\_ID ASC;**

16. 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*.

**Input Format**

The following tables contain data on the wands in Ollivander's inventory:

* *Wands:* The *id* is the id of the wand, *code* is the code of the wand, *coins\_needed* is the total number of gold galleons needed to buy the wand, and *power* denotes the quality of the wand (the higher the power, the better the wand is). 
* *Wands\_Property:* The *code* is the code of the wand, *age* is the age of the wand, and *is\_evil* denotes whether the wand is good for the dark arts. If the value of *is\_evil* is *0*, it means that the wand is not evil. The mapping between *code* and *age* is one-one, meaning that if there are two pairs,  and , then  and .

**Sol.**

**SELECT W.ID, WP.AGE, W.COINS\_NEEDED, W.POWER**

**FROM WANDS W**

**INNER JOIN WANDS\_PROPERTY WP**

**ON W.CODE = WP.CODE**

**WHERE WP.IS\_EVIL = 0**

**AND W.COINS\_NEEDED =**

**(**

**SELECT MIN(W2.COINS\_NEEDED)**

**FROM WANDS W2**

**INNER JOIN WANDS\_PROPERTY WP2**

**ON W2.CODE = WP2.CODE**

**WHERE WP.AGE = WP2.AGE**

**AND W.POWER = W2.POWER**

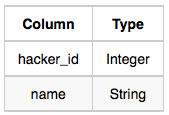
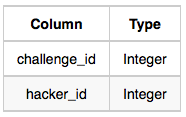
**)**

**ORDER BY W.POWER DESC, WP.AGE DESC;**

17. 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.

**Input Format**

The following tables contain challenge data:

* *Hackers:* The *hacker\_id* is the id of the hacker, and *name* is the name of the hacker. 
* *Challenges:* The *challenge\_id* is the id of the challenge, and *hacker\_id* is the id of the student who created the challenge. 

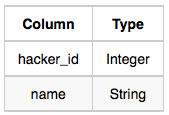
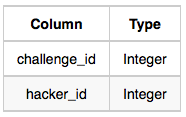
**Sol.**

**select challenges.hacker\_id,hackers.name,count(challenges.challenge\_id) from challenges inner join hackers on challenges.hacker\_id=hackers.hacker\_id group by challenges.hacker\_id,hackers.name having count(challenges.challenge\_id)>=8 and count(challenges.challenge\_id)!=12 order by count(challenges.challenge\_id) desc, hackers.hacker\_id;**

17. 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.

**Input Format**

The following tables contain challenge data:

* *Hackers:* The *hacker\_id* is the id of the hacker, and *name* is the name of the hacker. 
* *Challenges:* The *challenge\_id* is the id of the challenge, and *hacker\_id* is the id of the student who created the challenge. 

**Sol.**

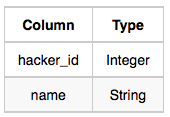
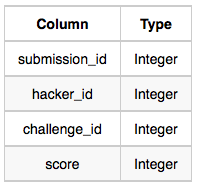
**select challenges.hacker\_id,hackers.name,count(challenges.challenge\_id) from challenges inner join hackers on challenges.hacker\_id=hackers.hacker\_id group by challenges.hacker\_id,hackers.name having count(challenges.challenge\_id)>=8 and count(challenges.challenge\_id)!=12 order by count(challenges.challenge\_id) desc, hackers.hacker\_id;**

18. You did such a great job helping Julia with her last coding contest challenge that she wants you to work on this one, too!

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.

**Input Format**

The following tables contain contest data:

* Hackers: The hacker\_id is the id of the hacker, and name is the name of the hacker. 
* Submissions: The submission\_id is the id of the submission, hacker\_id is the id of the hacker who made the submission, challenge\_id is the id of the challenge for which the submission belongs to, and score is the score of the submission. 

**Sol.**

**with ctes as (**

**select \*, Row\_Number() Over(PARTITION BY hacker\_id, challenge\_id order by score desc ) [Row\_num]**

**from Submissions**

**) SELECT h.hacker\_Id, h.name, Sum(c.Score)**

**from Hackers h inner join ctes c on h.hacker\_id = c.hacker\_id**

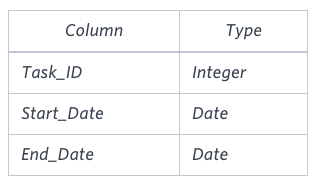
**where [Row\_num] = 1**

**group by h.hacker\_Id, h.name**

**having Sum(c.Score) > 0**

**order by Sum(c.Score) desc, hacker\_id**

19. You are given a table, Projects, containing three columns: Task\_ID, Start\_Date and End\_Date. It is guaranteed that the difference between the End\_Date and the Start\_Date is equal to 1 day for each row in the table.



If the End\_Date of the tasks are consecutive, then they are part of the same project. Samantha is interested in finding the total number of different projects completed.

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.

**Sol.**

**SELECT START\_DATE, X**

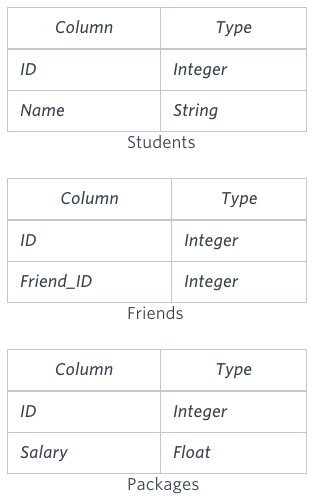
**FROM**

**(SELECT A.START\_DATE, MIN(B.END\_DATE) AS X FROM (SELECT START\_DATE FROM PROJECTS WHERE START\_DATE NOT IN (SELECT END\_DATE FROM PROJECTS)) A,**

**(SELECT END\_DATE FROM PROJECTS WHERE END\_DATE NOT IN (SELECT START\_DATE FROM PROJECTS)) B WHERE START\_DATE < END\_DATE GROUP BY A.START\_DATE ) P**

**ORDER BY DATEDIFF(X, START\_DATE), START\_DATE;**

20. You are given three tables: Students, Friends and Packages. Students contains two columns: ID and Name. Friends contains two columns: ID and Friend\_ID (ID of the ONLY best friend). Packages contains two columns: ID and Salary (offered salary in $ thousands per month).

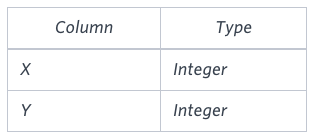


Write a query to output the names of those students whose best friends got offered a higher salary than them. Names must be ordered by the salary amount offered to the best friends. It is guaranteed that no two students got same salary offer.

**Sol.**

**SELECT TBL.Name FROM (SELECT S1.ID,S1.Name,P1.Salary,F.Friend\_ID,S2.Name AS FriendsName,P2.Salary AS FriendsSalary FROM Students S1 INNER JOIN Packages P1 ON S1.ID=P1.ID INNER JOIN Friends F ON S1.ID=F.ID INNER JOIN Students S2 ON F.Friend\_ID=S2.ID INNER JOIN Packages P2 ON F.Friend\_ID=P2.ID ) AS TBL WHERE TBL.FriendsSalary>TBL.Salary ORDER BY TBL.FriendsSalary;**

21. You are given a table, Functions, containing two columns: X and Y.



Two pairs (X1, Y1) and (X2, Y2) are said to be symmetric pairs if X1 = Y2 and X2 = Y1.

Write a query to output all such symmetric pairs in ascending order by the value of X. List the rows such that X1 ≤ Y1.

**Sol. SELECT f1.x,f1.y**

**FROM**

**(**

**SELECT x,y,row\_number() over(ORDER BY x) rw**

**FROM functions**

**) f1**

**JOIN**

**(**

**SELECT x,y,row\_number() over(ORDER BY x) rw**

**FROM functions**

**) f2**

**ON f1.rw<f2.rw**

**WHERE f1.x=f2.y AND f1.y=f2.x**

22. 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 .

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

**Sol.**

**SELECT**

**t1.contest\_id,**

**t1.hacker\_id,**

**t1.name,**

**SUM(t2.total\_submissions) AS ts,**

**SUM(t2.total\_accepted\_submissions) AS tas,**

**SUM(t3.total\_views) AS tv,**

**SUM(t3.total\_unique\_views) AS tuv**

**FROM (**

**SELECT**

**Con.contest\_id AS contest\_id,**

**Con.hacker\_id AS hacker\_id,**

**Con.name AS name,**

**Col.college\_id AS college\_id,**

**Cha.challenge\_id AS challenge\_id**

**FROM Contests Con**

**JOIN Colleges Col ON Col.contest\_id = Con.contest\_id**

**JOIN Challenges Cha ON Cha.college\_id = Col.college\_id**

**) AS t1**

**LEFT JOIN (**

**SELECT**

**Sub.challenge\_id AS challenge\_id,**

**COALESCE(SUM(Sub.total\_submissions), 0) AS total\_submissions,**

**COALESCE(SUM(Sub.total\_accepted\_submissions), 0) AS total\_accepted\_submissions**

**FROM Submission\_Stats Sub**

**GROUP BY challenge\_id**

**) AS t2 ON t1.challenge\_id = t2.challenge\_id**

**LEFT JOIN (**

**SELECT**

**Vie.challenge\_id AS challenge\_id,**

**COALESCE(SUM(Vie.total\_views), 0) AS total\_views,**

**COALESCE(SUM(Vie.total\_unique\_views), 0) AS total\_unique\_views**

**FROM View\_Stats Vie**

**GROUP BY challenge\_id**

**) AS t3 ON t1.challenge\_id = t3.challenge\_id**

**GROUP BY t1.contest\_id, t1.hacker\_id, t1.name**

**HAVING (ts + tas + tv + tuv) > 0**

**ORDER BY t1.contest\_id;**

**0|Add CommentPermalink**

**saytoabrarkhan 3 days ago**

**SELECT c.contest\_id, c.hacker\_id, c.name,**

**SUM(s.total\_submissions) AS total\_submissions,**

**SUM(s.total\_accepted\_submissions) AS total\_accepted\_submissions,**

**SUM(v.total\_views) AS total\_views,**

**SUM(v.total\_unique\_views) AS total\_unique\_views**

**FROM Contests c**

**LEFT JOIN Colleges cl ON c.contest\_id = cl.contest\_id**

**LEFT JOIN Challenges ch ON cl.college\_id = ch.college\_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**

**) s ON ch.challenge\_id = s.challenge\_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**

**) v ON ch.challenge\_id = v.challenge\_id**

**GROUP BY c.contest\_id, c.hacker\_id, c.name**

**HAVING SUM(s.total\_submissions) > 0**

**OR SUM(s.total\_accepted\_submissions) > 0**

**OR SUM(v.total\_views) > 0**

**OR SUM(v.total\_unique\_views) > 0**

**ORDER BY c.contest\_id;``**

23. Julia conducted a  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.

**Sol.**

**SELECT SUBMISSION\_DATE,**

**(SELECT Count(DISTINCT hacker\_id) as no\_of\_unique\_hacker\_id FROM submissions S2**

**WHERE S2.submission\_date = S1.submission\_date**

**AND (SELECT Count(DISTINCT S3.submission\_date)**

**FROM submissions S3**

**WHERE S3.hacker\_id = S2.hacker\_id**

**AND S3.submission\_date < S1.submission\_date**

**) = Datediff(S1.submission\_date, '2016-03-01')**

**) AS NO\_OF\_UNIQUE\_HACKERS,**

**(SELECT hacker\_id FROM submissions S2**

**WHERE S2.submission\_date = S1.submission\_date**

**GROUP BY hacker\_id**

**ORDER BY Count(submission\_id) DESC, hacker\_id ASC LIMIT 1**

**) AS MAX\_SUB\_HACKER\_ID,**

**(SELECT name FROM hackers**

**WHERE hacker\_id = MAX\_SUB\_HACKER\_ID**

**) AS NAME**

**FROM (SELECT DISTINCT submission\_date FROM submissions) S1**

**GROUP BY submission\_date;**