**NAME: ABHIJIT BARIK abro45172018@gmail.com**

**Table creation and value insertion:**

create database Assignment;

use  Assignment;

create table CITY

(

ID integer,

NAME VARCHAR(17),

COUNTRYCODE VARCHAR(3),

DISTRICT  VARCHAR(20),

 POPULATION  integer

);

insert into CITY values(6,'Rotterdam ','NLD','Zuid-Holland', 593321),

(3878, 'Scottsdale', 'USA', 'Arizona', 202705),(

3965, 'Corona', 'USA', 'California', 124966),

(3973, 'Concord', 'USA', 'California', 121780),

(3977,'Cedar Rapids', 'USA', 'Iowa', 120758),

(3982, 'Coral Springs', 'USA', 'Florida', 117549),

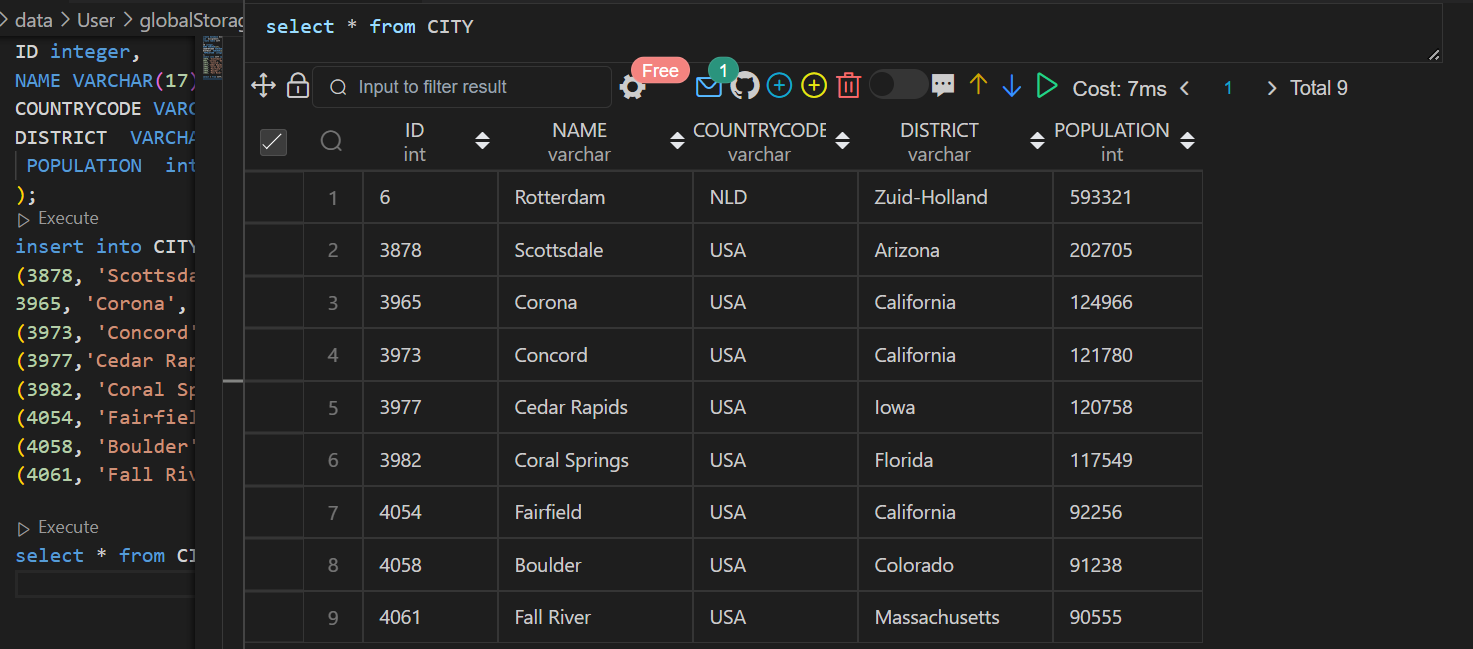
(4054, 'Fairfield', 'USA', 'California', 92256),

(4058, 'Boulder', 'USA', 'Colorado', 91238),

(4061, 'Fall River', 'USA','Massachusetts', 90555);

select \* from CITY;

output:



**Q1.**

**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.**

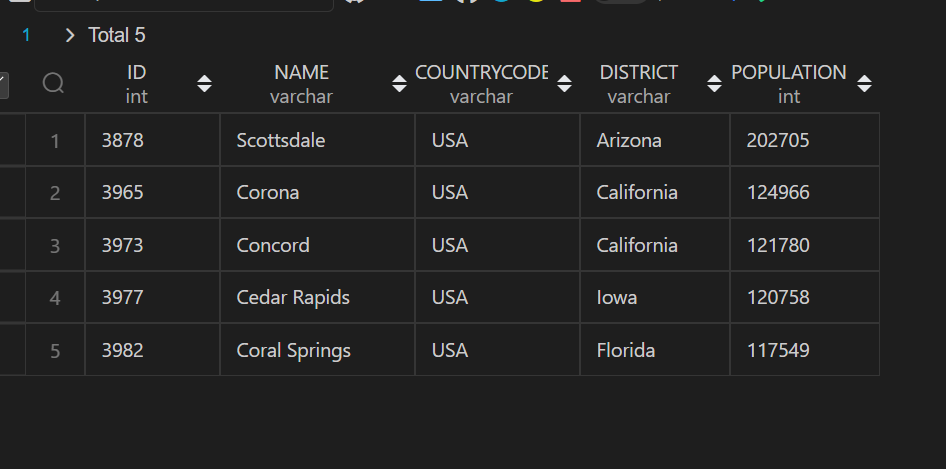
SQL CODE:

SELECT \*

FROM CITY

WHERE POPULATION > 100000 AND COUNTRYCODE = "USA";

OUTPUT:



**Q2. Query the NAME field for all American cities in the CITY table with populations larger than 120000. The CountryCode for America is USA**

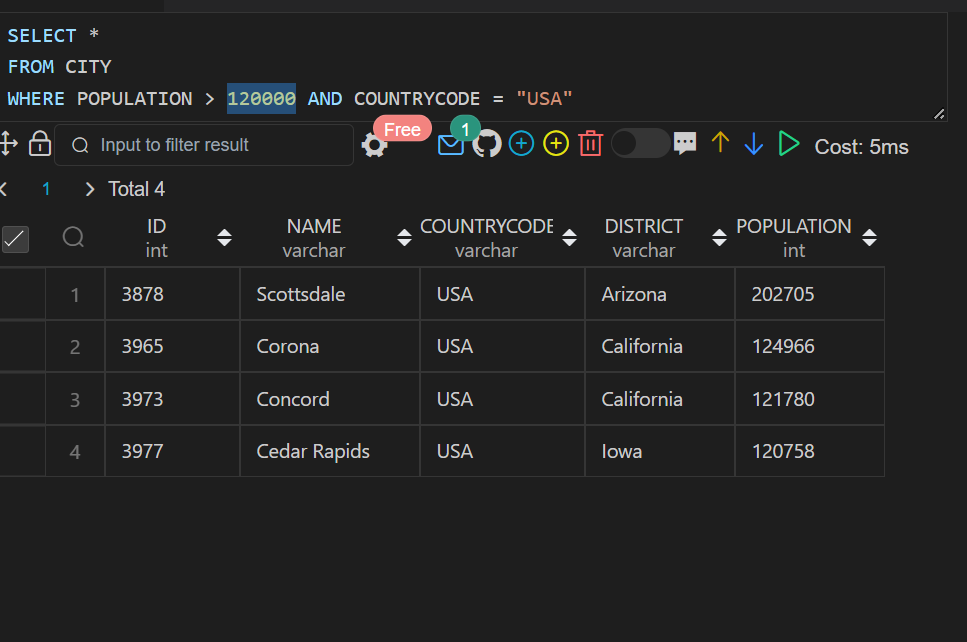
**SQL CODE:**

SELECT \*

FROM CITY

WHERE POPULATION > 120000 AND COUNTRYCODE = "USA";

**OUTPUT:**

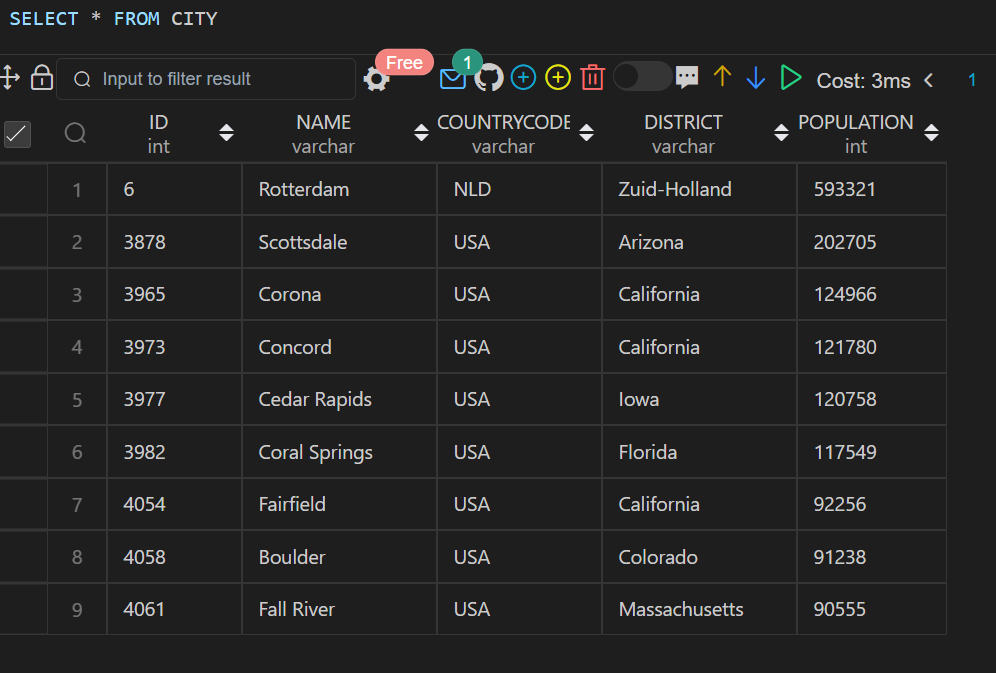
****

**Q3. Query all columns (attributes) for every row in the CITY table.**

**SQL CODE:**

SELECT \* FROM City;

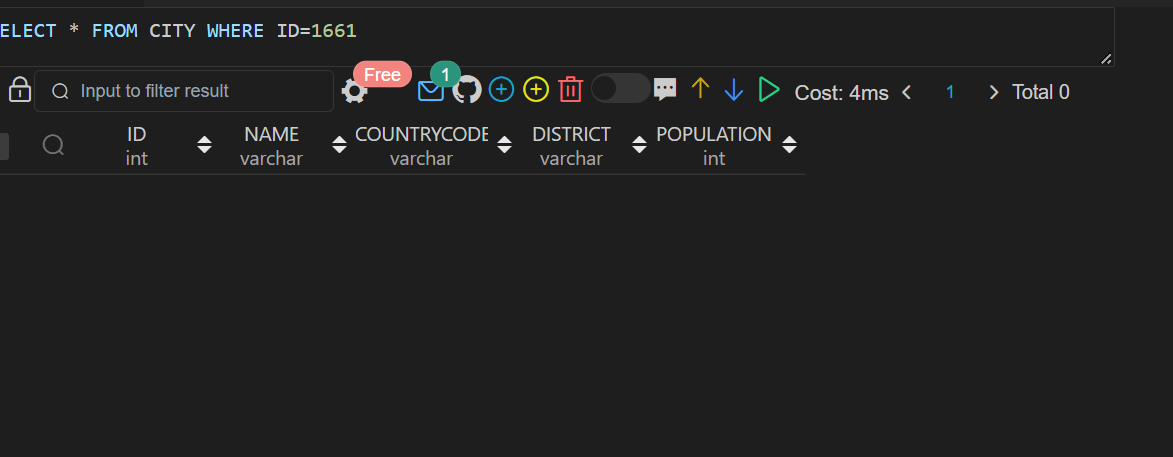
**OUTPUT:**

****

**Q4. Query all columns for a city in CITY with the ID 1661.**

**SQL CODE:**

SELECT \* FROM CITY WHERE ID=1661;

**OUTPUT:** ****

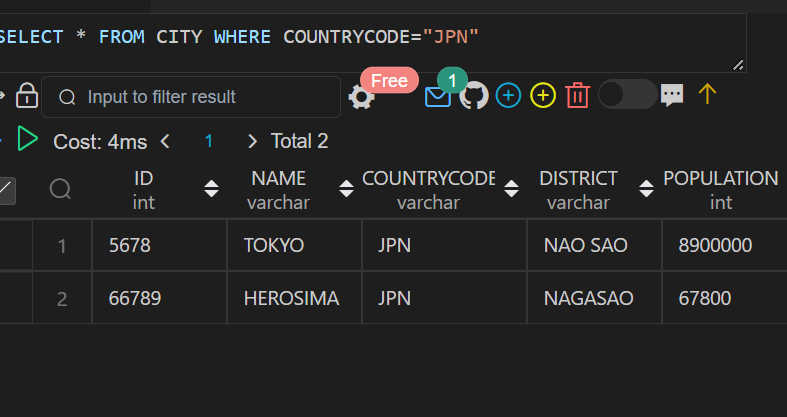
**Q5. Query all attributes of every Japanese city in the CITY table. The COUNTRYCODE for Japan is JPN.**

**Q6. Query the names of all the Japanese cities in the CITY table. The COUNTRYCODE for Japan is JPN.**

**SQL CODE:**

SELECT \* FROM CITY WHERE COUNTRYCODE="JPN";

**OUTPUT:**

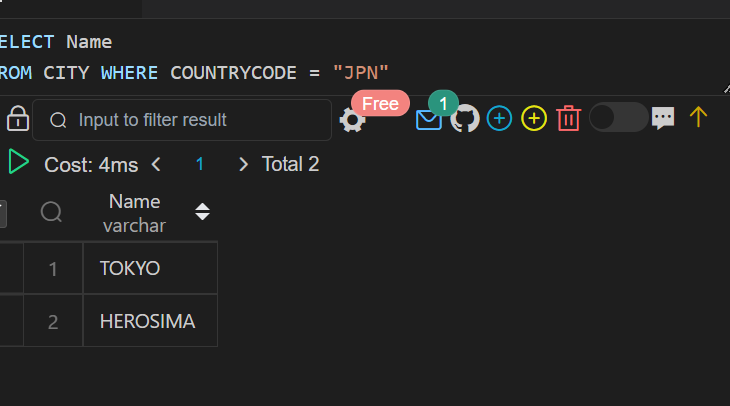
****

**SQL CODE:**

SELECT Name

FROM CITY WHERE COUNTRYCODE = "JPN";

**OUTPUT:**

****

**Sample dataset-2**

**Table creation and value insertion:**

CREATE DATABASE ASSIGNMENT;

USE ASSIGNMENT;

create table  STATION

(

ID integer,

CITY VARCHAR(17),

STATE VARCHAR(3),

LAT\_N  integer,

LONG\_W integer

);

insert into STATION values(794,'Kissee Mills','MO',139,73),

(824,'Loma Mar','CA',48,130),

(603,'Sandy Hook','CT',72,148),

(478,'Tipton','IN',33,97),

(619,'Arlington','CO',75,92),

(711,'Turner','AR',50,101),

(839,'Slidell','LA',85,151),

(411,'Negreet','LA',98,105),

(588,'Glencoe','KY',46,136),

(665,'Chelsea','IA', 98,59),

(342,'Chignik Lagoon','AK',103,153),

(733,'Pelahatchie','MS',38,28),

(441,'Hanna City','IL',50,136),

(811,'Dorrance','KS',102,121),

(698,'Albany','CA','49',80),

(325,'Monument','KS',70,141),

(414,'Manchester','MD',73,37),

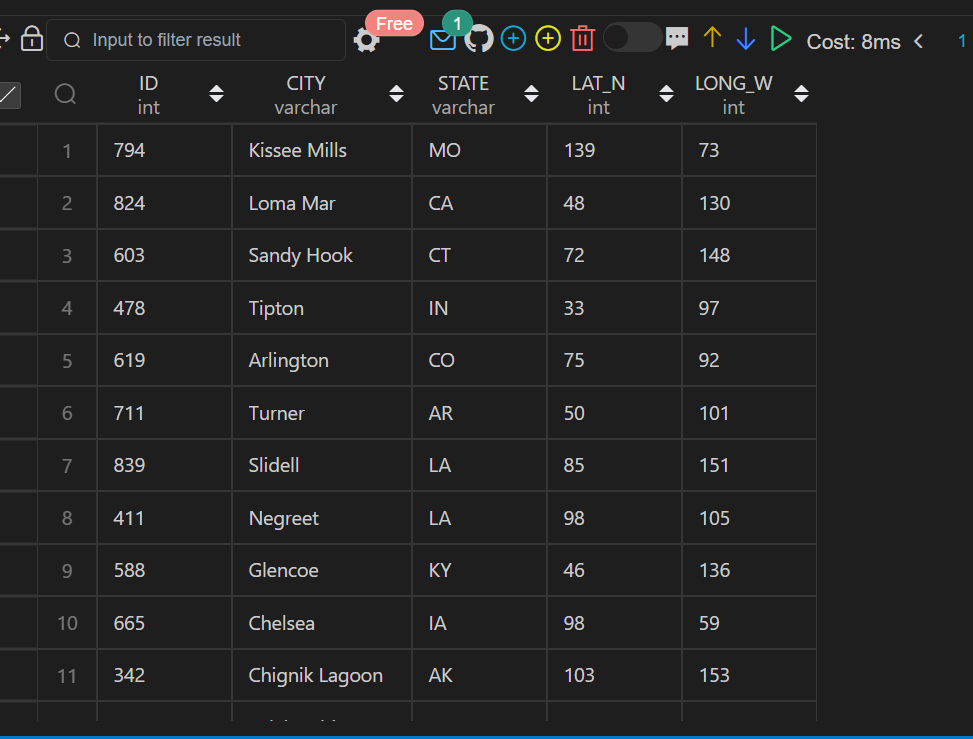
(113,'Prescott','IA',39,65),

(971,'Graettinger','IA',94,150),

(266,'Cahone','CO',116,127);

SELECT \* FROM STATION;

**OUTPUT:**

****

**Q7. Query a list of CITY and STATE from the STATION table.**

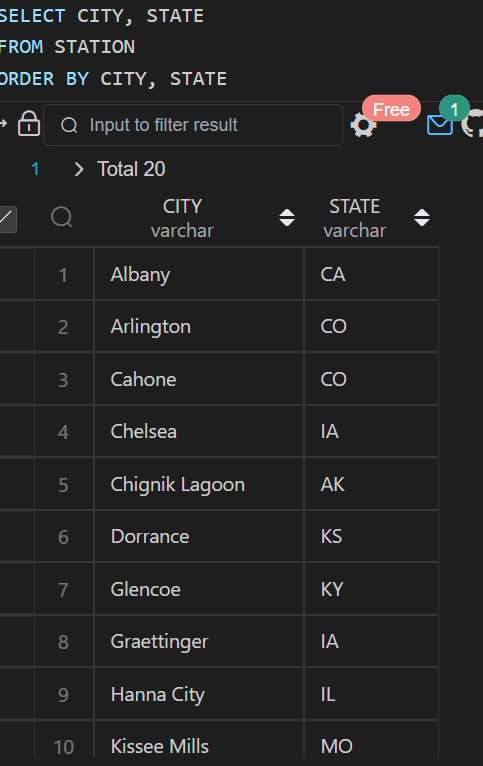
**SQL QUERY:**

SELECT CITY, STATE

FROM STATION

ORDER BY CITY, STATE;

**OUTPUT:**

****

**Q8. Query a list of CITY names from STATION for cities that have an even ID number. Print the results**

**SQL QUERY:**

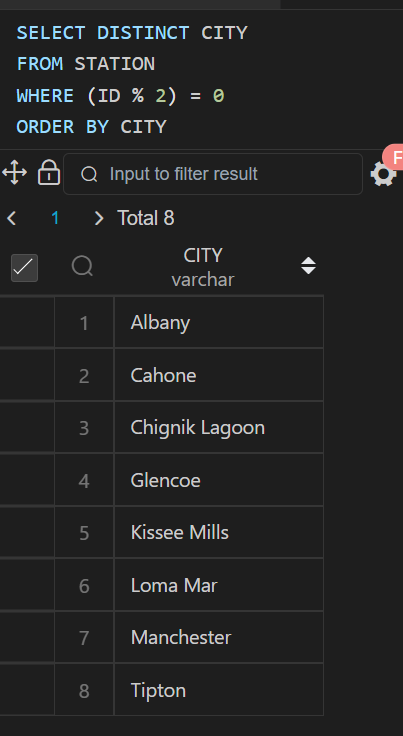
SELECT DISTINCT CITY

FROM STATION

WHERE (ID % 2) = 0

ORDER BY CITY;

**OUTPUT:**

****

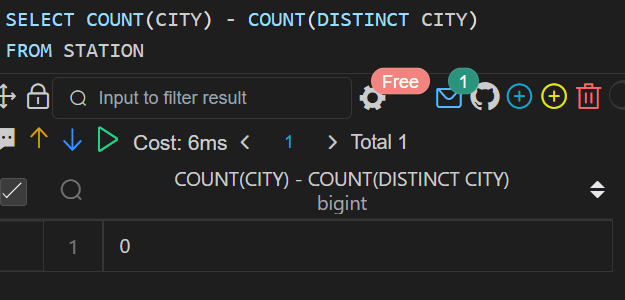
**Q9. Find the difference between the total number of CITY entries in the table and the number of distinct CITY entries in the table**

**SQL QUERY:**

SELECT COUNT(CITY) - COUNT(DISTINCT CITY)

FROM STATION;

**OUTPUT:**

****

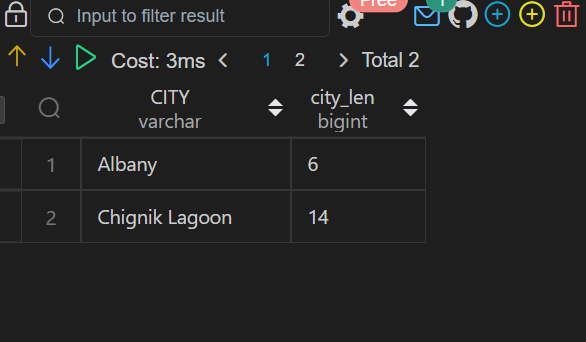
**Q10. Query the two cities in STATION with the shortest and longest CITY names, as well as their respective lengths (i.e.: number of characters in the name). If there is more than one smallest or largest city, choose the one that comes first when ordered alphabetically.**

**SQL QUERY:**

(select CITY, length(CITY) as city\_len from STATION order by city\_len asc, CITY asc limit 1) union

(select CITY, length(city) as city\_len from STATION order by city\_len desc, CITY asc limit 1);

**OUTPUT:**

****

**Q11.Query the list of CITY names starting with vowels (i.e., a, e, i, o, or u) from STATION. Your result cannot contain duplicates.**

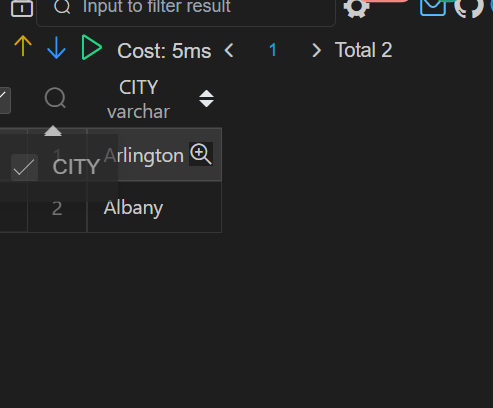
**SQL QUERY:**

SELECT  DISTINCT CITY

FROM STATION

WHERE CITY REGEXP'^[aeiouAEIOU]';

**OUTPUT:**

****

**Q12. Query the list of CITY names ending with vowels (a, e, i, o, u) from STATION. Your result cannot contain duplicates.**

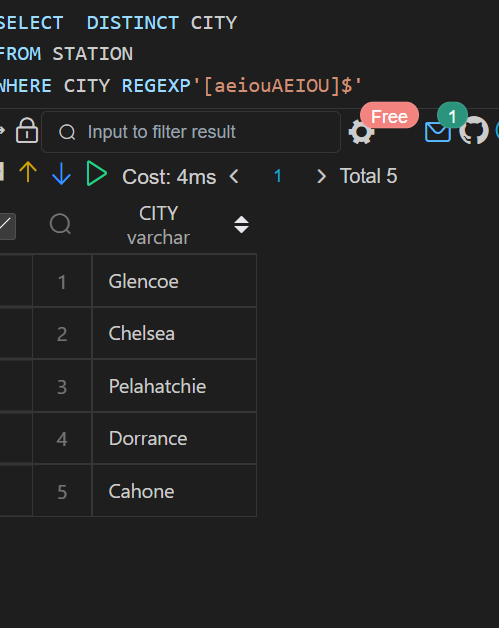
**SQL QUERY:**

SELECT  DISTINCT CITY

FROM STATION

WHERE CITY REGEXP'[aeiouAEIOU]$';

**OUTPUT:**

****

**Q13. Query the list of CITY names from STATION that do not start with vowels. Your result cannot contain duplicates.**

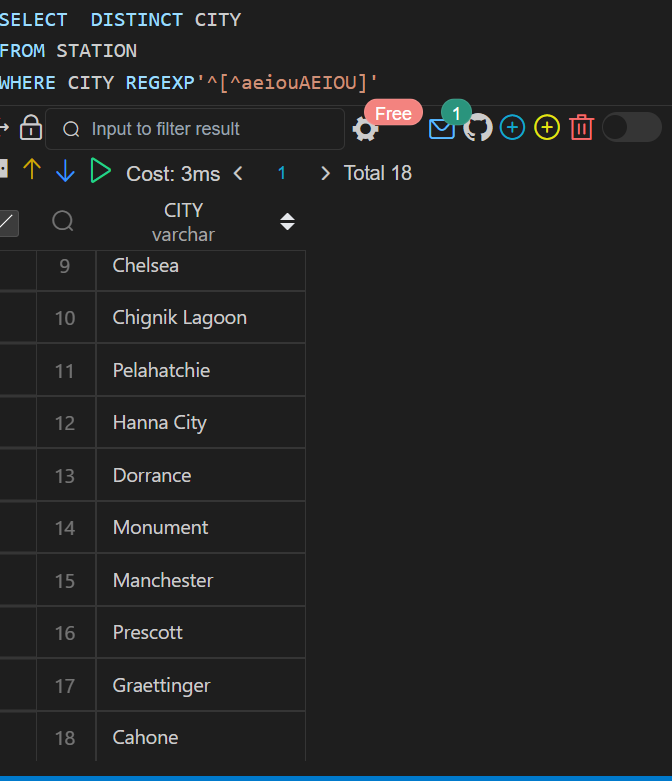
**SQL QUERY:**

SELECT  DISTINCT CITY

FROM STATION

WHERE CITY REGEXP'^[^aeiouAEIOU]';

**OUTPUT:**

****

**Q14. Query the list of CITY names from STATION that do not end with vowels. Your result cannot contain duplicates.**

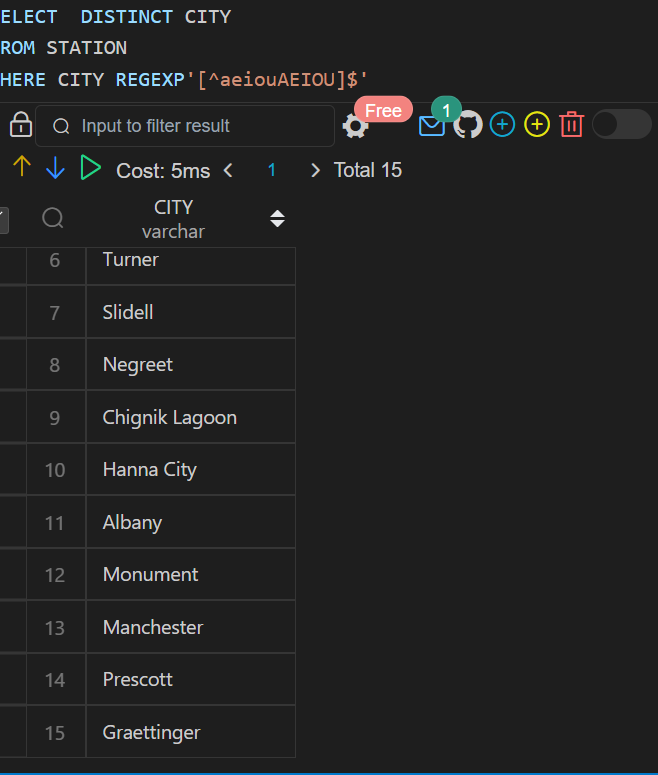
**SQL QUERY:**

SELECT  DISTINCT CITY

FROM STATION

WHERE CITY REGEXP'[^aeiouAEIOU]$';

**OUTPUT:**

****

**Q15. 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 QUERY:**

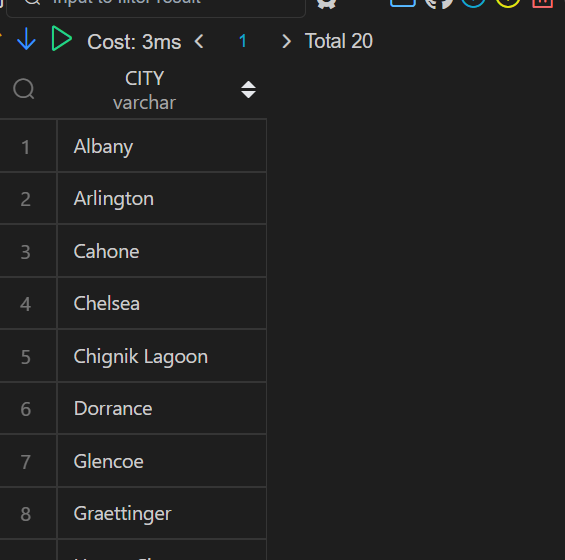
SELECT DISTINCT(CITY)

FROM STATION

WHERE CITY REGEXP "^[^aeiou].+" OR CITY REGEXP ".+[^aeiou]$"

ORDER BY CITY;

**OUTPUT:**

****

**Q16. 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 QUERY:**

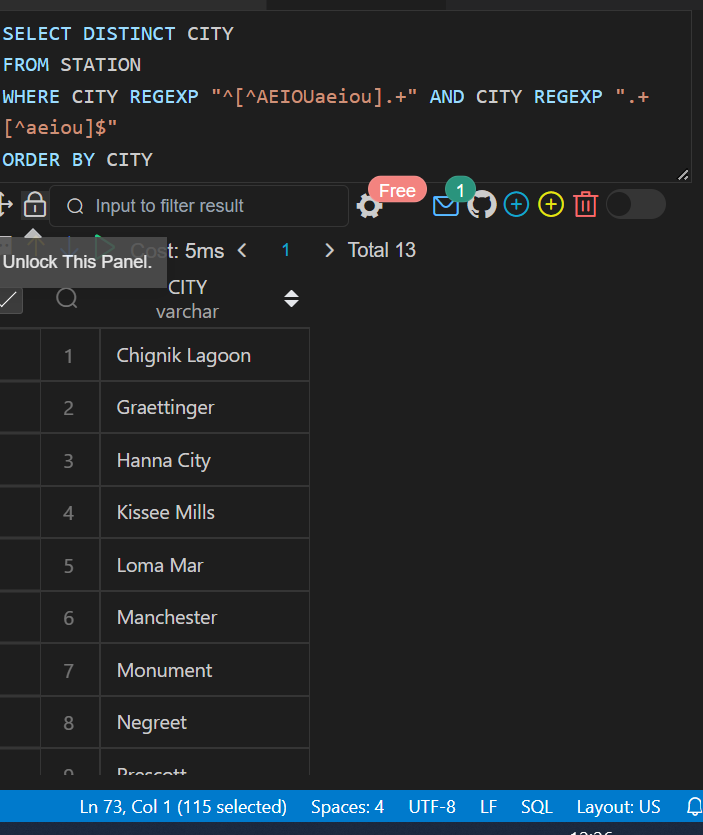
SELECT DISTINCT CITY

FROM STATION

WHERE CITY REGEXP "^[^AEIOUaeiou].+" AND CITY REGEXP ".+[^aeiou]$"

ORDER BY CITY;

**OUTPUT:**

****

Q17.

CREATE TABLE PRODUCT(

product\_id int PRIMARY KEY,

product\_name varchar(32),

unit\_price int

);

CREATE TABLE SALES(

seller\_id int,

product\_id int,

buyer\_id int,

sale\_date date,

quantity int,

price int,

constraint fk Foreign Key (product\_id) REFERENCES PRODUCT(product\_id)

);

INSERT INTO PRODUCT VALUES(1,'S8',1000),(2,'G4',800),(3,'iPhone',1400);

INSERT INTO SALES VALUES

(1,1,1,'2019-01-21',2,2000),

(1,2,2,'2019-02-17',1,800),

(2,2,3,'2019-06-02',1,800),

(3,3,4,'2019-05-13',2,2800);

**Write an SQL query that reports the products that were only sold in the first quarter of 2019. That is, between 2019-01-01 and 2019-03-31 inclusive.**

**SQL QUERY:**

SELECT product\_id,

       product\_name

FROM   PRODUCT

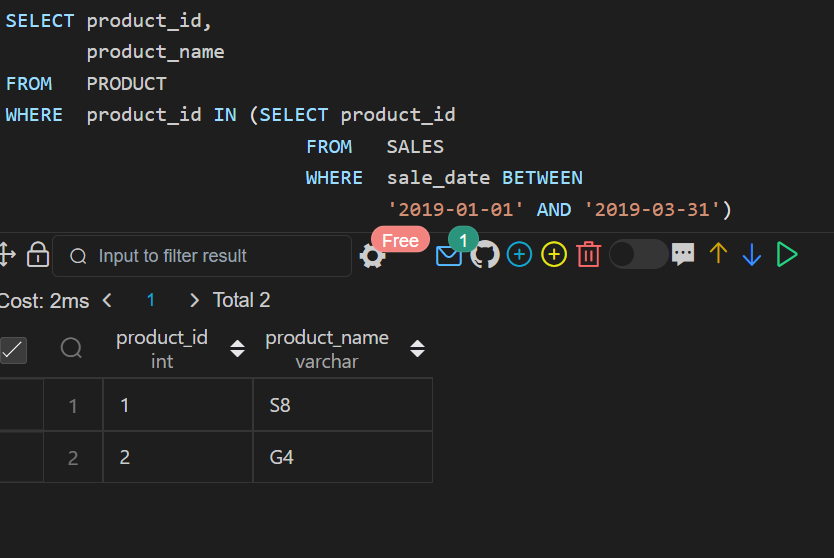
WHERE  product\_id IN (SELECT product\_id

                          FROM   SALES

                          WHERE  sale\_date BETWEEN

                                 '2019-01-01' AND '2019-03-31')

**OUTPUT:**

****

Q18.

create TABLE Views

(article\_id int,

author\_id int,

viewer\_id int,

view\_date date

);

INSERT into Views VALUES(1,3,5,'2019-08-01'),

(1,3,6,'2019-08-02'),

(2,7,7,'2019-08-01'),

(2,7,6,'2019-08-02'),

(4,7,1,'2019-07-22'),

(3,4,4,'2019-07-21'),

(3,4,4,'2019-07-21');

**Write an SQL query to find all the authors that viewed at least one of their own articles. Return the result table sorted by id in ascending order**

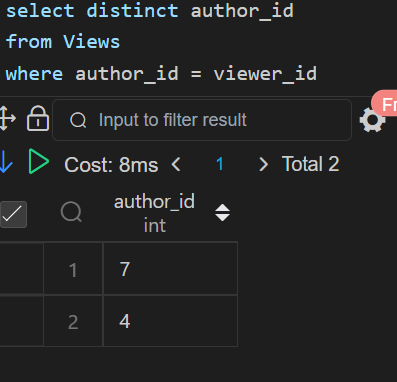
**SQL QUERY:**

select distinct author\_id

from Views

where author\_id = viewer\_id;

**OUTPUT:**

****

**Q19.**

CREATE  TABLE Delivery(

delivery\_id int PRIMARY key,

customer\_id int,

order\_date date,

customer\_pref\_delivery\_date date

);

INSERT into Delivery VALUES(1,1,'2019-08-01','2019-08-02'),

(2,5,'2019-08-02','2019-08-02'),

(3,1,'2019-08-11','2019-08-11'),

(4,3,'2019-08-24', '2019-08-26'),

(5,4,'2019-08-21', '2019-08-22'),

(6,2,'2019-08-11','2019-08-13');

**Q19.Write an SQL query to find the percentage of immediate orders in the table, rounded to 2 decimal places.**

**SQL QUERY:**

SELECT ROUND(

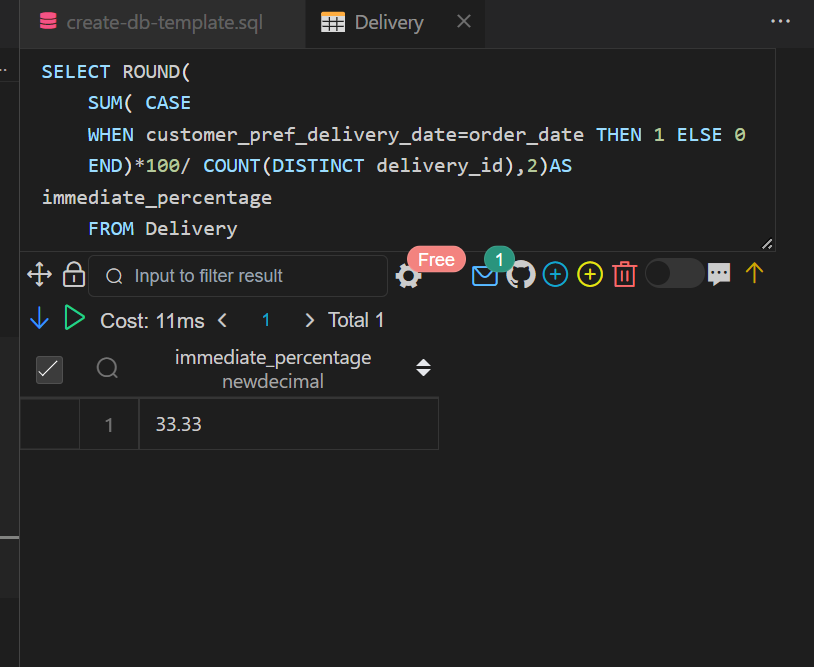
    SUM( CASE

    WHEN customer\_pref\_delivery\_date=order\_date THEN 1 ELSE 0

    END)\*100/ COUNT(DISTINCT delivery\_id),2)AS immediate\_percentage

    FROM Delivery;

**OUTPUT:**

****

**Q20.**

CREATE TABLE Ads(

ad\_id int,

user\_id int,

action enum('Clicked','Viewed','Ignored'),

 constraint pk Primary Key(ad\_id,user\_id)

);

INSERT INTO Ads VALUES(1,1,'Clicked'),

(2,2,'Clicked'),

(3,3,'Viewed'),

(5,5,'Ignored'),

(1,7,'Ignored'),

(2,7,'Viewed'),

(3,5,'Clicked'),

(1,4,'Viewed'),

(2,11,'Viewed'),

(1,2,'Clicked');

**Write an SQL query to find the ctr of each Ad. Round ctr to two decimal points.**

**SQL QUERY:**

select ad\_id,

    (case when clicks+views = 0 then 0 else round(clicks/(clicks+views)\*100, 2) end) as ctr

from

    (select ad\_id,

        sum(case when action='Clicked' then 1 else 0 end) as clicks,

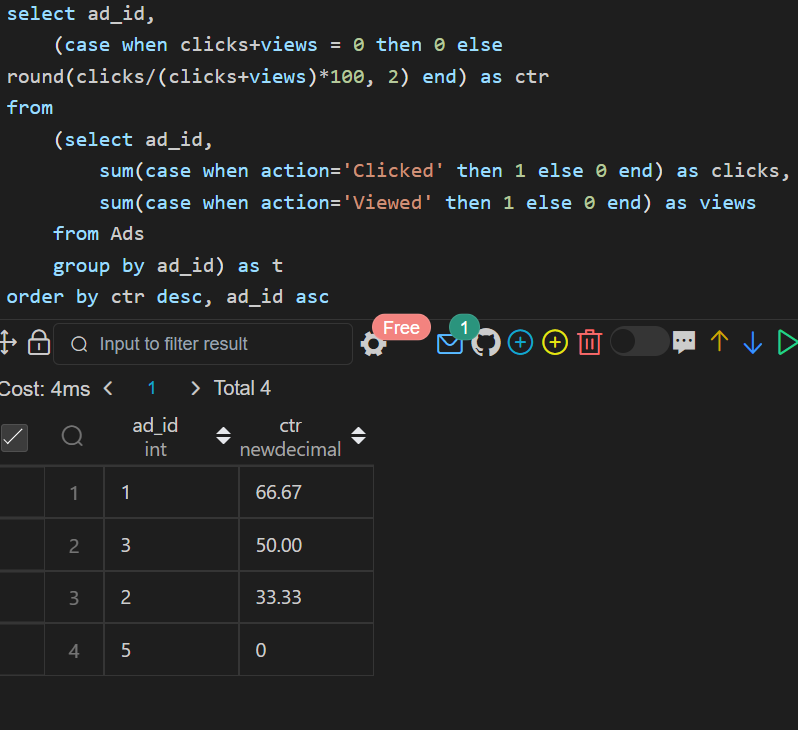
        sum(case when action='Viewed' then 1 else 0 end) as views

    from Ads

    group by ad\_id) as t

order by ctr desc, ad\_id asc;

**OUTPUT:**

****

Q21.

CREATE TABLE Employee(

employee\_id int,

 team\_id INT

);

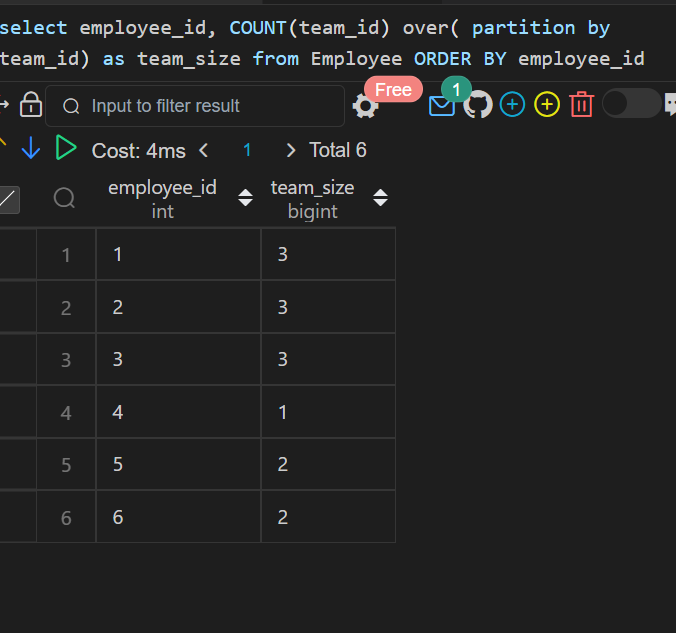
insert into Employee values(1,8),(2,8),(3,8),(4,7),(5,9),(6,9);

**Write an SQL query to find the team size of each of the employees.**

**SQL QUERY:**

select employee\_id, COUNT(team\_id) over( partition by team\_id) as team\_size from Employee ORDER BY employee\_id;

**OUTPUT:**

****

Q22.

CREATE Table Countries(

country\_id int PRIMARY key,

country\_name varchar(20)

 );

 CREATE Table Weather(

country\_id int,

weather\_state int,

day date,

CONSTRAINT  PK PRIMARY KEY(country\_id, day)

 );

insert into Countries values(2,'USA'),

(3,'Australia'),

(7,'Peru'),

(5,'China'),

(8,'Morocco'),

(9,'Spain');

insert into Weather values(2,15,'2019-11-01'),

(2,12,'2019-10-28'),

(2,12,'2019-10-27'),

(3,-2,'2019-11-10'),

(3,0,'2019-11-11'),

(3,3,'2019-11-12'),

(5,16,'2019-11-07'),

(5,18,'2019-11-09'),

(5,21,'2019-11-23'),

(7,25,'2019-11-28'),

(7,22,'2019-12-01'),

(7,20,'2019-12-02'),

(8,25,'2019-11-05'),

(8,27,'2019-11-15'),

(8,31,'2019-11-25'),

(9,7,'2019-10-23'),

(9,3,'2019-12-23');

Q22)**Write an SQL query to find the type of weather in each country for November 2019.**

**SQL QUERY:**

select country\_name, case when avg(weather\_state) <= 15 then "Cold"

                          when avg(weather\_state) >= 25 then "Hot"

                          else "Warm" end as weather\_type

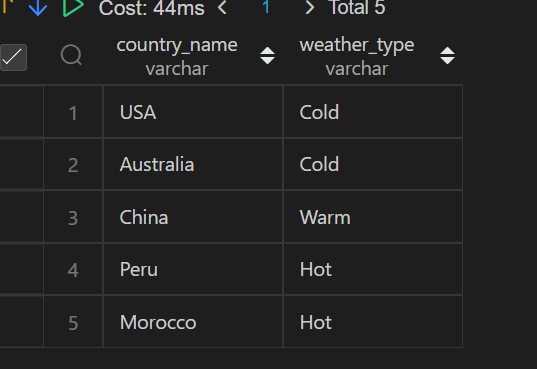
from Countries inner join Weather

on Countries.country\_id = Weather.country\_id

where left(day, 7) = '2019-11'

group by country\_name

**OUTPUT:**



**Q23.**

**Q24.**

CREATE TABLE Activity(

player\_id int,

device\_id int,

event\_date date,

games\_played int

);

INSERT INTO Activity VALUES(1,2,'2016-03-01',5),(1,2,'2016-05-02',6),(2,3,'2017-06-25',1),

(3,1,'2016-03-02',0),(3,4,'2018-07-03',5);

**Write an SQL query to report the first login date for each player.**

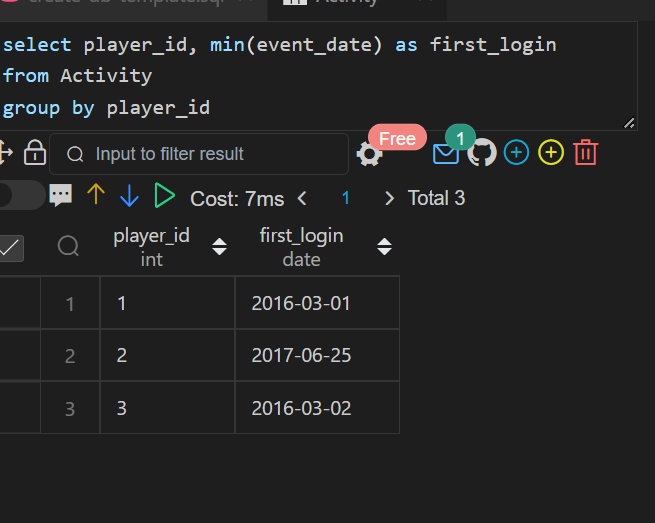
**SQL QUERY:**

select player\_id, min(event\_date) as first\_login

from Activity

group by player\_id;

**OUTPUT:**

****

Q25.

**Write an SQL query to report the device that is first logged in for each player**

**SQL QUERY:**

 SELECT player\_id, device\_id  FROM (SELECT

        player\_id,

        device\_id,

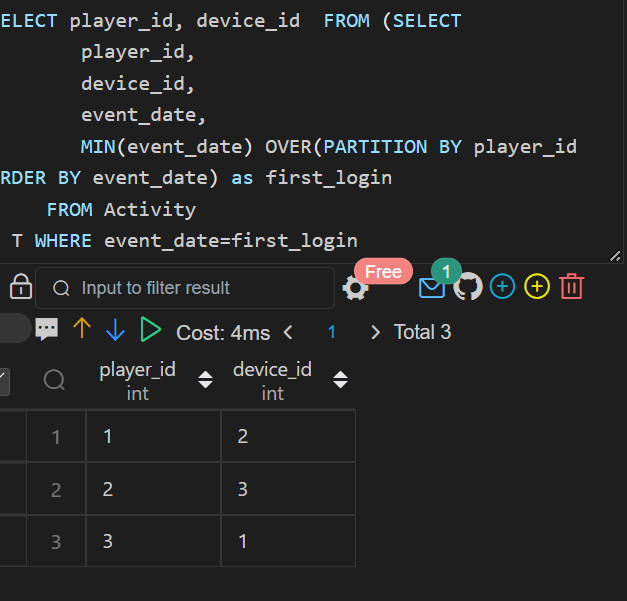
        event\_date,

        MIN(event\_date) OVER(PARTITION BY player\_id ORDER BY event\_date) as first\_login

     FROM Activity

) T WHERE event\_date=first\_login;

**OUTPUT:**

****

**Q26**

CREATE TABLE  Products(

product\_id int,

product\_name varchar(29),

product\_category varchar(19)

);

CREATE TABLE ORDERS(

product\_id int,

order\_date date,

unit int

);

INSERT INTO Products VALUES(1,'Leetcode Solutions','Book'),(2,'Jewels of Stringology',' Book'),

(3,'HP','Laptop'),

(4,'Lenovo','Laptop'),

(5,'Leetcode Kit','T-shir');

INSERT INTO ORDERS VALUES(1,'2020-02-05',60),

(1,'2020-02-10', 70),

(2,'2020-01-18',30),

(2,'2020-02-11',80),

(3,'2020-02-17',2),

(3,'2020-02-24',3),

(4,'2020-03-01',20),

(4,'2020-03-04',30),

(4,'2020-03-04',60),

(5,'2020-02-25',50),

(5,'2020-02-27',50),

(5,'2020-03-01',50);

**Write an SQL query to get the names of products that have at least 100 units ordered in February 2020**

**SQL QUERY:**

select p.product\_name as product\_name, o.sum\_unit as unit from Products p

join

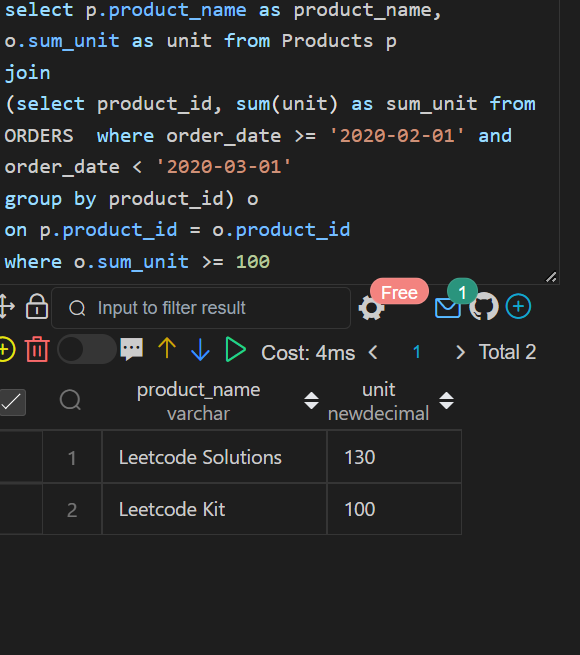
(select product\_id, sum(unit) as sum\_unit from ORDERS  where order\_date >= '2020-02-01' and order\_date < '2020-03-01'

group by product\_id) o

on p.product\_id = o.product\_id

where o.sum\_unit >= 100;

**OUTPUT:**

****

**Q27.**

CREATE TABLE Users(

user\_id int,

name varchar(33),

mail varchar(42)

);

insert into Users VALUES(1,'Winston','winston@leetcode.com'),

(2,'Jonathan','jonathanisgreat'),

(3,'Annabelle','bella-@leetcode.com'),

(4,'Sally','sally.come@leetcode.com'),

(5,'Marwan','quarz#2020@leetcode.com'),

(6,'David','david69@gmail.com'),

(7,'Shapiro','.shapo@leetcode.com');

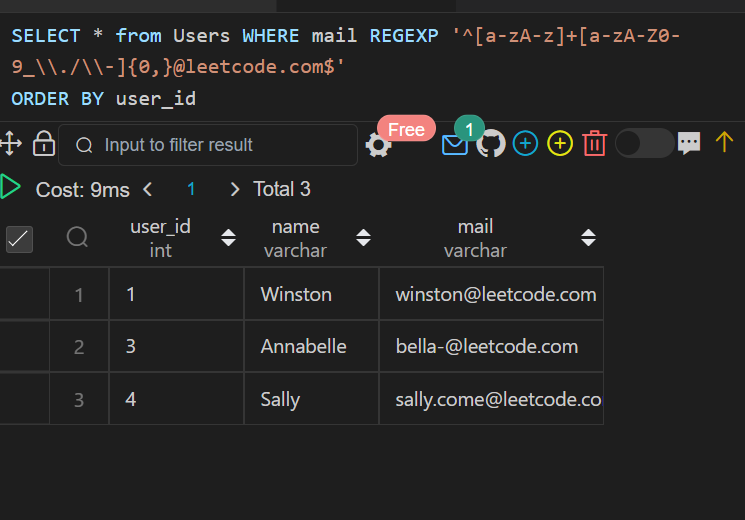
**Write an SQL query to find the users who have valid emails.**

**SQL QUERY:**

SELECT \* from Users WHERE mail REGEXP '^[a-zA-z]+[a-zA-Z0-9\_\\./\\-]{0,}@leetcode.com$'

ORDER BY user\_id;

**OUTPUT:**

****

**Q28)**

CREATE TABLE Customers(

customer\_id int,

name varchar(33),

country varchar(33)

);

DROP TABLE Product;

CREATE TABLE Product(

product\_id int PRIMARY KEY,

name varchar(33),

price int

);

CREATE TABLE ORDERSS(

order\_id int PRIMARY KEY,

customer\_id int,

product\_id int,

order\_date date,

quantity int

);

INSERT INTO Customers VALUES(1,'Winston','USA'),(2,'Jonathan','Peru'),

(3,'Moustafa','Egypt');

INSERT INTO Product VALUES(10,'LC Phone', 300),

(20,'LC T-Shirt',10),

(30,'LC Book',45),

(40,'LC Keychain',22);

INSERT INTO ORDERSS VALUES(1,1,10,'2020-06-10',1),

(2,1,20,'2020-07-01',1),

(3,1,30,'2020-07-08', 2),

(4,2,10,'2020-06-15',2),

(5,2,40,'2020-07-01',10),

(6,3,20,'2020-06-24',2),

(7,3,30,'2020-06-25', 2),

(9,3,30,'2020-05-08',3);

**Write an SQL query to report the customer\_id and customer\_name of customers who have spent at least $100 in each month of June and July 2020.**

**SQL QUERY:**

select o.customer\_id, c.name

from

ORDERSS  o

join

Product p

on o.product\_id = p.product\_id

join

Customers c

on o.customer\_id = c.customer\_id

group by 1, 2

having sum(case when date\_format(order\_date, '%Y-%m')='2020-06'

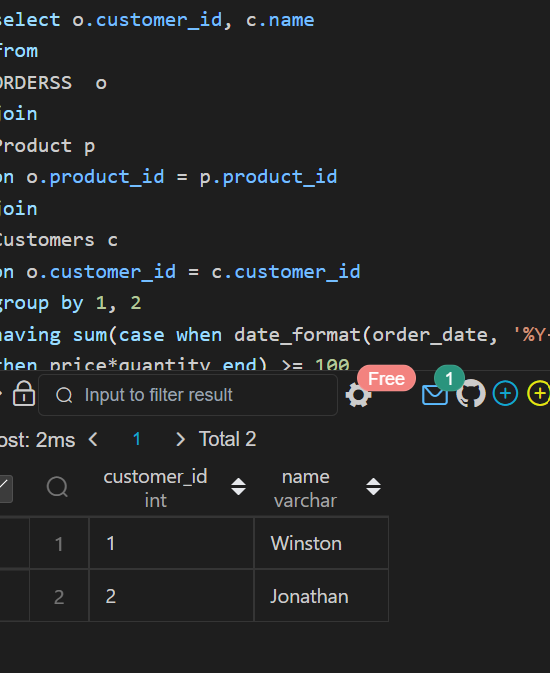
then price\*quantity end) >= 100

and

sum(case when date\_format(order\_date, '%Y-%m')='2020-07'

then price\*quantity end) >= 100

**OUTPUT:**

****

**Q29)**

CREATE TABLE TV(

program\_date date,

content\_id int,

channel varchar(30),

CONSTRAINT PRIMARY KEY(program\_date,content\_id )

);

CREATE TABLE Content(

content\_id varchar(30) PRIMARY KEY,

title varchar(30),

Kids\_content enum('Y','N'),

content\_type varchar(30)

);

INSERT INTO TV VALUES('2020-06-10 08:00',1,'LC-Channel'),

('2020-05-11 12:00',2,'LC-Channel'),

('2020-05-12 12:00',3,'LC-Channel'),

('2020-05-13 14:00', 4 ,'Disney Ch'),

('2020-06-18 14:00',4,'Disney Ch'),

('2020-07-15 16:00',5,'Disney Ch');

INSERT INTO Content VALUES

(1,'Leetcode Movie', 'N','Movies'),

(2, 'Alg for Kids', 'Y','Series'),

(3,'Database Sols','N','Series'),

(4,'Aladdin','Y','Movies'),

(5,'Cinderella','Y','Movies');

**Write an SQL query to report the distinct titles of the kid-friendly movies streamed in June 2020. Return the result table in any order**

**SQL QUERY:**

SELECT DISTINCT title

FROM Content ctt

INNER JOIN TV t

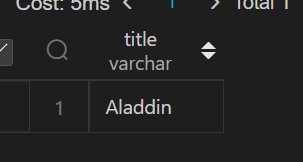
ON ctt.content\_id = t.content\_id

WHERE content\_type = 'Movies'

AND Kids\_content = 'Y'

AND program\_date BETWEEN '2020-06-01' AND '2020-06-30';

**OUTPUT:**

****

**Q30.**

CREATE TABLE NPV(

id int,

year int,

npv int,

 constraint primary key(id, year)

);

CREATE TABLE Queries(

id int,

year int,

constraint primary key(id, year)

);

insert into NPV VALUES (1,2018,100),

(7,2020,30),

(13, 2019, 40),

(1 ,2019 ,113),

(2 ,2008 ,121),

(3, 2009 ,12),

(11,2020,99),

(7,2019,0);

INSERT INTO Queries VALUES(1 ,2019),

(2,2008),

(3,2009),

(7,2018),

(7,2019),

(7,2020),

(13,2019);

**Write an SQL query to find the npv of each query of the Queries table**

**SQL QUERY:**

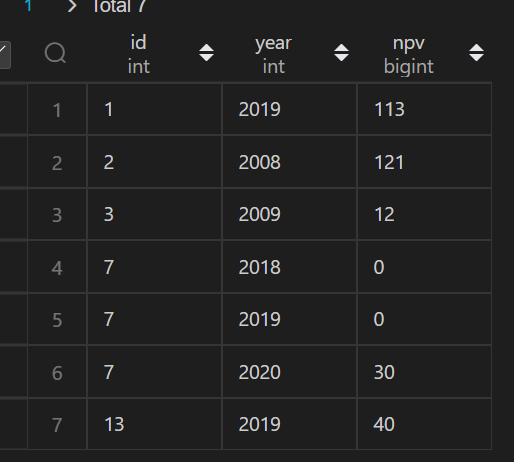
select q.id, q.year, ifnull(n.npv,0) as npv

from Queries as q

left join NPV as n

on q.id = n.id AND  q.year = n.year;

**OUTPUT:**

****

**Q31.**

CREATE TABLE NPV(

id int,

year int,

npv int,

 constraint primary key(id, year)

);

CREATE TABLE Queries(

id int,

year int,

constraint primary key(id, year)

);

insert into NPV VALUES (1,2018,100),

(7,2020,30),

(13, 2019, 40),

(1 ,2019 ,113),

(2 ,2008 ,121),

(3, 2009 ,12),

(11,2020,99),

(7,2019,0);

INSERT INTO Queries VALUES(1 ,2019),

(2,2008),

(3,2009),

(7,2018),

(7,2019),

(7,2020),

(13,2019);

**Write an SQL query to find the npv of each query of the Queries table**

**SQL QUERY:**

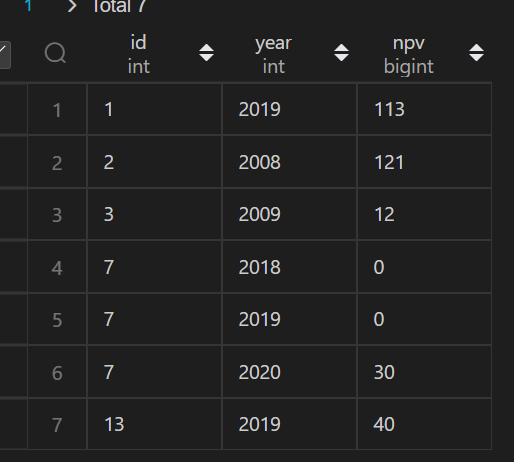
select q.id, q.year, ifnull(n.npv,0) as npv

from Queries as q

left join NPV as n

on q.id = n.id AND  q.year = n.year;

**OUTPUT:**

****

**Q32.**

create TABLE Employees(

id int PRIMARY  KEY,

name varchar(40)

 );

  create TABLE EmployeeUNI(

id int,

unique\_id int,

 constraint PRIMARY KEY(id, unique\_id)

);

INSERT INTO Employees VALUES(1,'Alice'),

(7, 'Bob'),

(11, 'Meir'),

(90,'Winston'),

(3,'Jonathan');

INSERT INTO EmployeeUNI VALUES(3,1),

(11, 2),

(90,3);

**Write an SQL query to show the unique ID of each user, If a user does not have a unique ID replace just show null.**

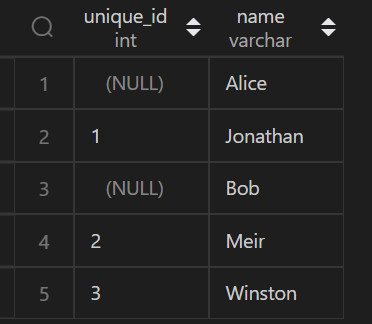
**SQL QUERY:**

select unique\_id, name

from Employees left join EmployeeUNI

on Employees.id = EmployeeUNI.id

**OUTPUT:**

****

**Q33)**

CREATE TABLE Users(

id int PRIMARY KEY,

name varchar(30)

);

CREATE TABLE Rides(

id int PRIMARY KEY,

user\_id int,

distance int

);

insert into Users VALUES(1,'Alice'),

(2,'Bob'),

(3,'Alex'),

(4,'Donald'),

(7,'Lee'),

(13,'Jonathan'),

(19,'Elvis');

insert into Rides VALUES

(1,1,120),

(2,2,317),

(3,3,222),

(4,7,100),

(5,13,312),

(6,19,50),

(7,7,120),

(8,19,400),

(9,7,230);

**Write an SQL query to report the distance travelled by each user. Return the result table ordered by travelled\_distance in descending order, if two or more users travelled the same distance, order them by their name in ascending order.**

**SQL QUERY:**

select name, sum(ifnull(distance, 0)) as travelled\_distance

from Rides r

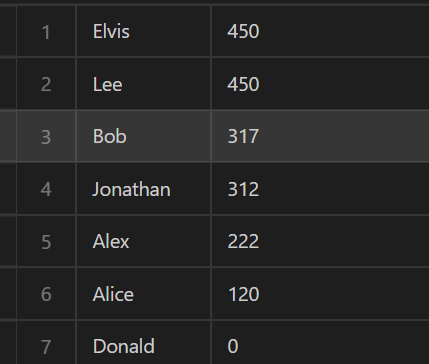
right join Users u

on r.user\_id = u.id

group by name

order by 2 desc,1 asc;

**OUTPUT:**

****

**Q34)**

create TABLE Products(

product\_id int PRIMARY KEY,

product\_name varchar(30),

product\_category varchar(30)

 );

  create TABLE Orders(product\_id int,

order\_date date,

unit int);

INSERT INTO Products VALUES(1,'Leetcode Solutions','Book'),(2,

'Jewels of Stringology', 'Book'),

(3,'HP','Laptop'),

(4,'Lenovo','Laptop'),

(5,'Leetcode Kit','T-shirt');

INSERT INTO Orders VALUES (1,'2020-02-05',60),

(1,'2020-02-10',70),

(2,'2020-01-18',30),

(2,'2020-02-11',80),

(3,'2020-02-17',2),

(3,'2020-02-24',3),

(4,'2020-03-01',20),

(4,'2020-03-04',30),

(4,'2020-03-04',60);

**Write an SQL query to get the names of products that have at least 100 units ordered in February 2020 and their amount.**

**SQL QUERY:**

select product\_name, sum(unit) as unit

from Products inner join Orders

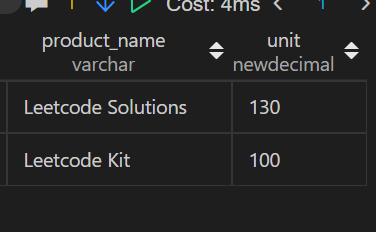
on Products.product\_id = Orders.product\_id

where left(order\_date, 7) = "2020-02"

group by Products.product\_id

having sum(unit)>=100;

**OUTPUT:**

****

**Q35**

**SQL QUERY:**

 create TABLE Movies(

movie\_id int PRIMARY KEY,

title varchar(30)

  );

 create TABLE Users(

user\_id int PRIMARY KEY,

name varchar(40)

 );

 create TABLE MovieRating(

movie\_id int,

user\_id int,

rating int,

created\_at date

 );

 INSERT INTO Movies VALUES(1,'Avengers'),

(2,'Frozen 2'),

(3,'Joker');

 INSERT INTO Users VALUES(1,'Daniel'),

(2,'Monica'),

(3,'Maria'),

(4,'James');

 INSERT INTO MovieRating VALUES(1, 1, 3 ,'2020-01-12'),

(1 ,2 ,4, '2020-02-11'),

(1 ,3 ,2, '2020-02-12'),

(1 ,4 ,1 ,'2020-01-01'),

(2 ,1 ,5, '2020-02-17'),

(2 ,2, 2,'2020-02-01'),

(2,3,2,'2020-03-01'),

(3, 1, 3,'2020-02-22'),

(3,2,4,'2020-02-25');

SELECT user\_name AS results FROM

(

SELECT a.name AS user\_name, COUNT(\*) AS counts FROM MovieRating AS b

    JOIN Users AS a

    on a.user\_id = b.user\_id

    GROUP BY b.user\_id

    ORDER BY counts DESC, user\_name ASC LIMIT 1

) first\_query

UNION

SELECT movie\_name AS results FROM

(

SELECT c.title AS movie\_name, AVG(d.rating) AS rate FROM MovieRating AS d

    JOIN Movies AS c

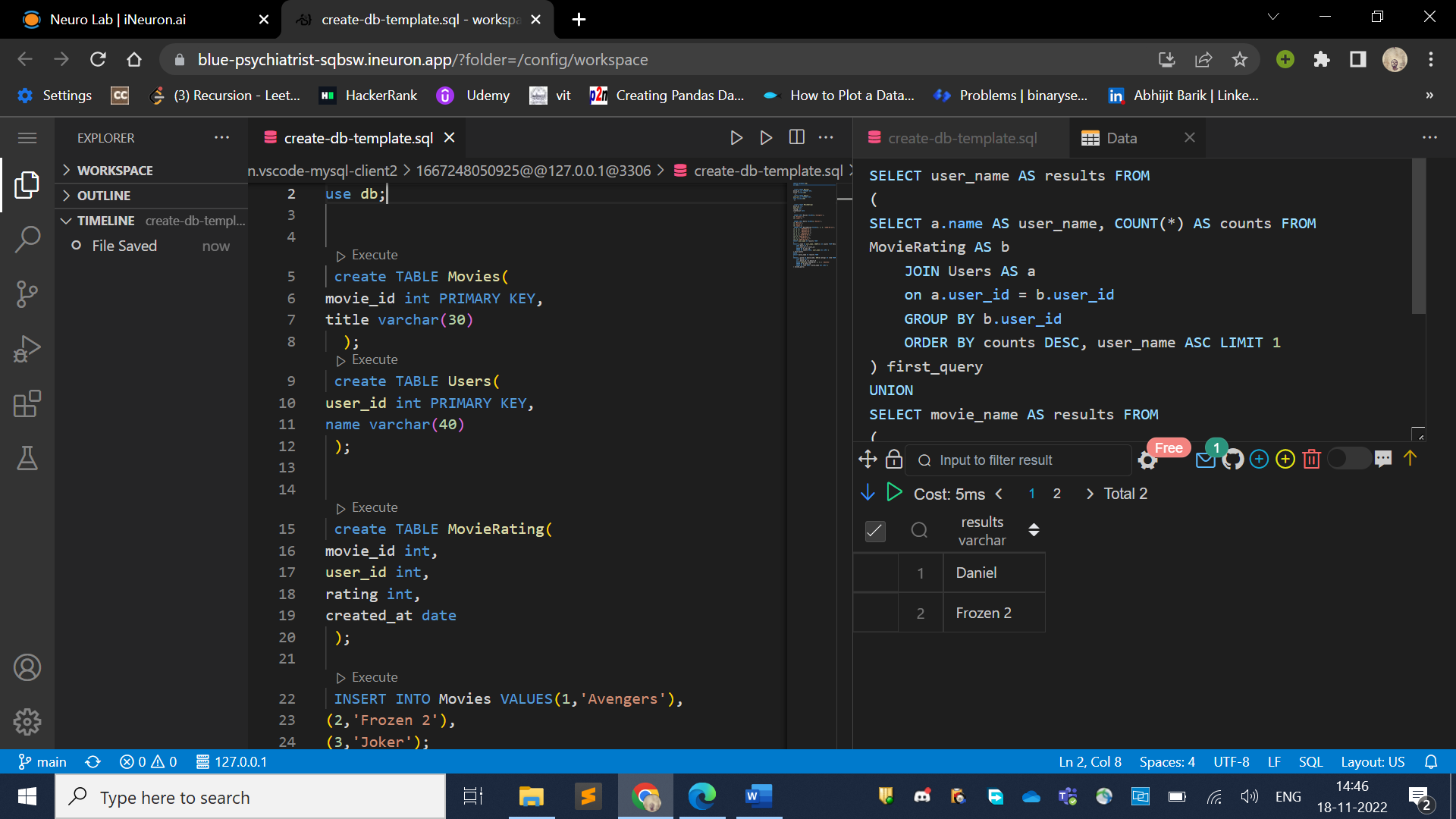
    on c.movie\_id = d.movie\_id

    WHERE substr(d.created\_at, 1, 7) = '2020-02'

    GROUP BY d.movie\_id

    ORDER BY rate DESC, movie\_name ASC LIMIT 1

) second\_query;

**OUTPUT:** 

**Q36)**

CREATE TABLE Users(

id int PRIMARY KEY,

name varchar(30)

);

CREATE TABLE Rides(

id int PRIMARY KEY,

user\_id int,

distance int

);

insert into Users VALUES(1,'Alice'),

(2,'Bob'),

(3,'Alex'),

(4,'Donald'),

(7,'Lee'),

(13,'Jonathan'),

(19,'Elvis');

insert into Rides VALUES

(1,1,120),

(2,2,317),

(3,3,222),

(4,7,100),

(5,13,312),

(6,19,50),

(7,7,120),

(8,19,400),

(9,7,230);

**Write an SQL query to report the distance travelled by each user. Return the result table ordered by travelled\_distance in descending order, if two or more users travelled the same distance, order them by their name in ascending order.**

**SQL QUERY:**

select name, sum(ifnull(distance, 0)) as travelled\_distance

from Rides r

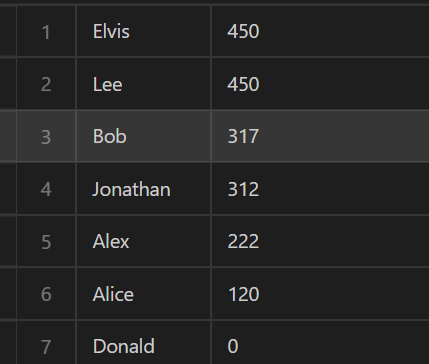
right join Users u

on r.user\_id = u.id

group by name

order by 2 desc,1 asc;

**OUTPUT:**

****

**Q37)**

create TABLE Employees(

id int PRIMARY  KEY,

name varchar(40)

 );

  create TABLE EmployeeUNI(

id int,

unique\_id int,

 constraint PRIMARY KEY(id, unique\_id)

);

INSERT INTO Employees VALUES(1,'Alice'),

(7, 'Bob'),

(11, 'Meir'),

(90,'Winston'),

(3,'Jonathan');

INSERT INTO EmployeeUNI VALUES(3,1),

(11, 2),

(90,3);

**Write an SQL query to show the unique ID of each user, If a user does not have a unique ID replace just show null.**

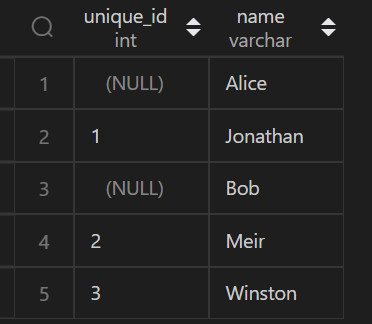
**SQL QUERY:**

select unique\_id, name

from Employees left join EmployeeUNI

on Employees.id = EmployeeUNI.id

**OUTPUT:**

****

**Q38.**

CREATE TABLE  Departments(

id int PRIMARY KEY,

name varchar(30)

);

CREATE TABLE Students(

id int PRIMARY KEY,

name varchar(30),

department\_id int

);

INSERT INTO  Departments VALUES(1,'Electrical Engineering'),

(7,'Computer Engineering'),

(13,'Business Administration');

INSERT INTO Students VALUES(23,'Alice',1),

(1,'Bob',7),

(5,'Jennifer',13),

(2,'John',14),

(4,'Jasmine',77),

(3,'Steve',74),

(6,'Luis',1),

(8,'Jonathan',7),

(7,'Daiana',33),

(11,'Madelynn',1);

**Write an SQL query to find the id and the name of all students who are enrolled in departments that no longer exist.**

**SQL QUERY:**

select s.id, s.name

from Students s

left join Departments d

on s.department\_id = d.id

where d.id is null;

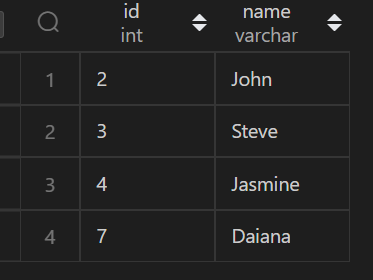
--Other way

SELECT id, name

FROM Students

WHERE department\_id not in (SELECT id from Departments);

**OUTPUT:**

****

**Q39.**

 CREATE TABLE Calls(

from\_id int,

to\_id int,

duration int

 );

 INSERT INTO Calls VALUES(1,2,59),

(2,1,11),

(1,3,20),

(3,4,100),

(3,4,200),

(3,4,200),

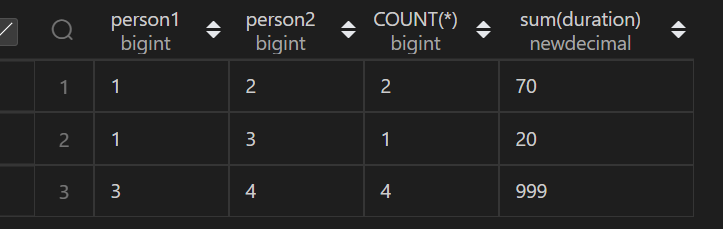
(4,3,499);

**Write an SQL query to report the number of calls and the total call duration between each pair of distinct persons (person1, person2) where person1 < person2**

**SQL QUERY:**

SELECT LEAST(from\_id,to\_id) as person1, Greatest(from\_id,to\_id) as person2,COUNT(\*),sum(duration) from Calls group by person1,person2;

**OUTPUT:**

****

**Q40.**

CREATE TABLE Pricess(

product\_id int,

start\_date date,

end\_date date,

price int,

constraint pk PRIMARY key (product\_id,start\_date,end\_date)

);

CREATE TABLE  UnitsSold(

product\_id int,

purchase\_date date,

units int

);

insert into Pricess VALUES(1,'2019-02-17','2019-02-28',5),

(1,'2019-03-01','2019-03-22',20),

(2,'2019-02-01','2019-02-20',15),

(2,'2019-02-21','2019-03-31',30);

INSERT INTO UnitsSold VALUES(1,'2019-02-25',100),

(1,'2019-03-01',15),

(2,'2019-02-10',200),

(2,'2019-03-22',30);

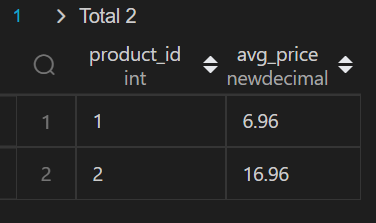
**Write an SQL query to find the average selling price for each product. average\_price should be rounded to 2 decimal places**

**SQL QUERY:**

select p.product\_id, round(sum(p.price\*u.units)/sum(u.units) ,2) as avg\_price

from Pricess p INNER JOIN UnitsSold u on p.product\_id=u.product\_id where u.purchase\_date  BETWEEN p.start\_date and p.end\_date GROUP BY p.product\_id ;

**OUTPUT:**

****

**Q41.**

CREATE TABLE Warehouse(

name varchar(128),

product\_id int,

units int,

CONSTRAINT PK PRIMARY KEY(name,product\_id)

);

CREATE TABLE Products(

product\_id int PRIMARY KEY,

product\_name varchar(128),

Width int,

Length int,

Height int

);

INSERT INTO Warehouse VALUES ('LCHouse1',1,1),

('LCHouse1',2,10),

('LCHouse1',3,5),

('LCHouse2', 1,2),

('LCHouse2',2,2),

('LCHouse3',4,1);

INSERT INTO Products VALUES(1 ,'LC-TV', 5 ,50, 40),

(2, 'LC-KeyChain', 5, 5, 5),

(3,'LC-Phone', 2, 10 ,10),

(4 ,'LC-T-Shirt',4,10 ,20);

**Write an SQL query to report the number of cubic feet of volume the inventory occupies in each warehouse.**

**SQL QUERY:**

select warehouse\_name, sum(volume) as volume from (

    select w.name as warehouse\_name, w.product\_id, w.units \* Width \* Length \* Height as volume

    from Warehouse w INNER join Products p on w.product\_id = p.product\_id

) t group by warehouse\_name;

**OUTPUT:**

****

**Q42.**

CREATE Table Sales(

sale\_date date,

fruit enum("apples","oranges"),

sold\_num int,

CONSTRAINT PK PRIMARY KEY(sale\_date, fruit)

);

INSERT INTO Sales VALUES ('2020-05-01','apples', 10),

('2020-05-01', 'oranges',8),

('2020-05-02','apples', 15),

('2020-05-02','oranges',15),

('2020-05-03','apples',20),

('2020-05-03', 'oranges',0),

('2020-05-04','apples',15),

('2020-05-04','oranges',16);

**Write an SQL query to report the difference between the number of apples and oranges sold each day. Return the result table ordered by sale\_date.**

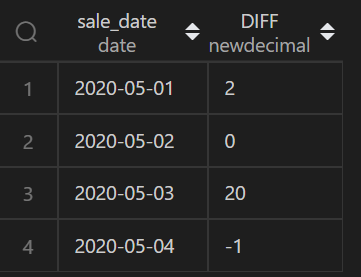
**SQL QUERY:**

SELECT sale\_date, SUM(CASE WHEN fruit='apples' THEN sold\_num

                           WHEN fruit='oranges' THEN -sold\_num end ) AS DIFF

 FROM Sales GROUP BY sale\_date ;

**OUTPUT:**

****

**Q43**

CREATE TABLE Activity(

player\_id int,

device\_id int,

event\_date date,

games\_played int,

CONSTRAINT PK PRIMARY KEY(player\_id, event\_date)

 );

 INSERT INTO Activity VALUES(1,2,'2016-03-01',5),

(1,2,'2016-03-02',6),

(2,3,'2017-06-25',1),

(3,1,'2016-03-02',0),

(3,4,'2018-07-03',5);

**Write an SQL query to report the fraction of players that logged in again on the day after the day they first logged in, rounded to 2 decimal places.**

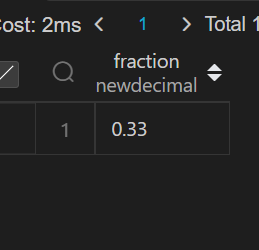
**SQL QUERY:**

select  round(count(cte.player\_id)/(select count(distinct player\_id) from Activity) ,2)as fraction

from (SELECT player\_id,min(event\_date) as start\_date from Activity GROUP BY player\_id) as cte  inner join Activity a

on cte.player\_id=a.player\_id  and datediff(cte.start\_date,a.event\_date)=-1;

**OUTPUT:**

****

**Q44.**

create TABLE Employee(

id int PRIMARY key,

name varchar(30),

department varchar(30),

managerId int

 );

INSERT into Employee VALUES(101,'John','A',null),

(102,'Dan','A',101),

(103,'James','A',101),

(104,'Amy','A',101),

(105, 'Anne','A', 101),

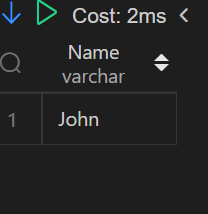
(106,'Ron','B', 101);

**Write an SQL query to report the managers with at least five direct reports.**

**SQL QUERY:**

select Name From  Employee WHERE Id in( select managerId FROM Employee GROUP BY managerId HAVING  COUNT(\*) >=5);

**OUTPUT:**

****

**Q45.**

create TABLE Student(

student\_id int,

student\_name varchar(30),

gender varchar(30),

dept\_id int,

constraint  PRIMARY KEY(student\_id),

constraint  Foreign Key(dept\_id) REFERENCES Department(dept\_id)

);

CREATE TABLE  Department(

dept\_id int PRIMARY KEY,

dept\_name varchar(30)

);

INSERT INTO Student values(1,'Jack','M',1),

(2,'Jane','F',1),

(3,'Mark','M',2);

INSERT INTO Department VALUES(1,'Engineering'),

(2,'Science'),

(3,'Law');

**Write an SQL query to report the respective department name and number of students majoring in each department for all departments in the Department table**

**SQL QUERY:**

SELECT

    d.DEPT\_NAME,

    COUNT(s.STUDENT\_ID)

FROM

    Department d

    LEFT JOIN Student s ON d.dept\_id = s.dept\_id

GROUP by

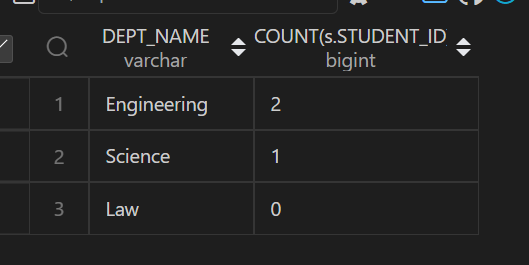
    d.dept\_id

ORDER by

    COUNT(s.STUDENT\_ID) DESC,

    d.DEPT\_NAME ASC

**OUTPUT:**

****

**Q46.**

create table Customer(

customer\_id int,

product\_key int,

constraint Foreign Key(product\_key) REFERENCES Product(product\_key)

);

create table Product(

product\_key int PRIMARY KEY

);

INSERT INTO  Customer  VALUES( 1,5),

(2,6),

(3,5),

(3,6),

(1,6);

INSERT INTO  Product VALUES(5),(6);

**Write an SQL query to report the customer ids from the Customer table that bought all the products in the Product table**

**SQL QUERY:**

SELECT

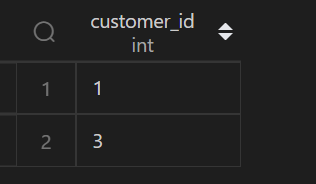
    customer\_id

FROM Customer

GROUP BY customer\_id

HAVING COUNT( DISTINCT product\_key) = (SELECT COUNT(\*) FROM Product)

**OUTPUT:**

****

**Q47**

CREATE TABLE Project(

project\_id int,

employee\_id int,

constraint pk PRIMARY KEY(project\_id, employee\_id),

constraint fk FOREIGN KEY(employee\_id) REFERENCES (employee\_id)

);

CREATE TABLE Employees(

employee\_id int PRIMARY KEY,

name varchar(40),

experience\_years int

);

INSERT INTO  Project VALUES(1,1),

(1,2),

(1,3),

(2,1),

(2,4);

INSERT INTO Employees VALUES(1,'Khaled',3),

(2,'Ali',2),

(3,'John',3),

(4,'Doe',2);

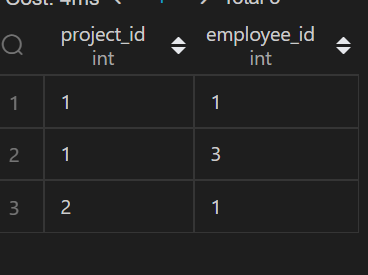
**Write an SQL query that reports the most experienced employees in each project. In case of a tie, report all employees with the maximum number of experience years**

**SQL QUERY:**

SELECT project\_id ,employee\_id from (SELECT p.project\_id ,p.employee\_id, dense\_rank()  over(partition by p.project\_id order by E.experience\_years desc ) as rank1

FROM Project p INNER JOIN Employees E on p.employee\_id=E.employee\_id) temp where rank1=1;

**OUTPUT:**

****

**Q48.**

create table Books(

book\_id int PRIMARY KEY,

name varchar(30),

available\_from date

);

CREATE TABLE Orders(

order\_id int PRIMARY KEY,

book\_id int,

quantity int,

dispatch\_date date,

CONSTRAINT FK FOREIGN KEY(book\_id)REFERENCES Books(book\_id)

);

INSERT INTO Books VALUES(1,"Kalila And Demna",'2010-01-01'),

(2 ,"28 Letters",'2012-05-12'),(3,"The Hobbit",'2019-06-10'),

(4 ,"13 Reasons Why",'2019-06-01'),(5,"The Hunger Games",'2008-09-21' );

INSERT INTO Orders VALUES(1,1,2,'2018-07-26'),(2,1,1,'2018-11-05'),(3,3,8,'2019-06-11'),

(4,4,6,'2019-06-05'),

(5,4,5,'2019-06-20'),

(6,5,9,'2009-02-02'),

(7,5,8,'2010-04-13');

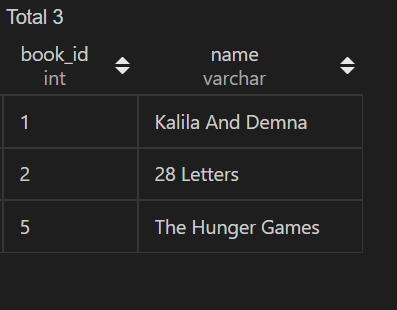
**Write an SQL query that reports the books that have sold less than 10 copies in the last year, excluding books that have been available for less than one month from today. Assume today is 2019-06-23. Data of this question is missing, for full data visit** [**https://code.dennyzhang.com/unpopular-books**](https://code.dennyzhang.com/unpopular-books)

**SQL QUERY:**

select book\_id, name

from Books

where book\_id not in (select book\_id from Orders where dispatch\_date >='2018-06-23' and dispatch\_date <= '2019-06-22' GROUP BY book\_id  having sum(quantity) >=10) and available\_from < '2019-05-23';

**OUTPUT:  
**

**Q49.**

CREATE TABLE Enrollments(

student\_id int,

course\_id int,

grade int,

CONSTRAINT PK PRIMARY KEY(student\_id, course\_id)

);

INSERT INTO Enrollments(2,2,95),

(2,3,95),

(1,1,90),

(1,2,99),

(3,1,80),

(3,2,75),

(3,3,82);

**Write a SQL query to find the highest grade with its corresponding course for each student. In case of a tie, you should find the course with the smallest course\_id.**

**SQL QUERY:**

select student\_id, min(course\_id) as course\_id, grade

from Enrollments

where (student\_id, grade) in

    (select student\_id, max(grade)

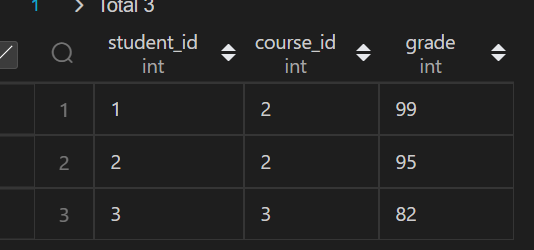
    from Enrollments

    group by student\_id)

group by student\_id,grade

order by student\_id asc

**OUTPUT:**

****

**Q50**

CREATE TABLE Players(

player\_id int PRIMARY KEY,

group\_id varchar(30)

);

 CREATE TABLE Matches(

match\_id int primary KEY,

first\_player int,

second\_player int,

first\_score int,

second\_score int

 );

 insert into Players VALUES(15,1),

(25,1),

(30,1),

(45,1),

(10,2),

(35,2),

(50,2),

(20,3),

(40,3);

insert into Matches VALUES(1,15,45,3,0),

(2,30,25,1,2),(3,30,25,1,2),

(4,40,20,5,2),(5,35,50,1,1);

**The winner in each group is the player who scored the maximum total points within the group. In the case of a tie, the lowest player\_id wins. Write an SQL query to find the winner in each group.**

**Query:**

select group\_id,player\_id

from (

    select sc.group\_id group\_id, sc.player\_id player\_id,

       rank() over (partition by sc.group\_id order by sum(sc.score) desc, sc.player\_id asc) as rnk

    from(

        select p.group\_id group\_id,

         p.player\_id player\_id ,

         sum(m.first\_score) as score

        from Players p

        inner join Matches m

        on p.player\_id = m.first\_player

        group by p.group\_id,p.player\_id

        union all

        select p.group\_id group\_id,

         p.player\_id player\_id ,

        sum(second\_score) as score

        from Players p

        inner join Matches m

        on p.player\_id = m.second\_player

        group by p.group\_id,p.player\_id

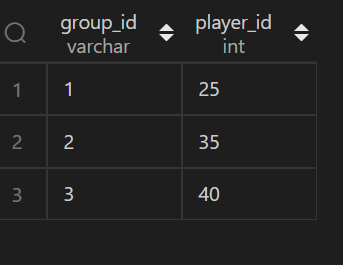
    ) sc

    group by sc.group\_id,sc.player\_id

) A

where rnk = 1;

**OUTPUT:**

****

**Q51.**

create TABLE world(

name varchar(30) PRIMARY KEY,

continent varchar(30),

area bigint,

population bigint,

gdp bigint

);

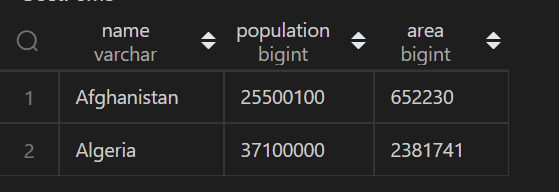
**Write an SQL query to report the name, population, and area of the big countries. Return the result table in any order**

**Query:**

SELECT name,population,area FROM  world where

area >=3000000 or population >= 25000000;

**OUTPUT:**

****

**Q52.**

CREATE TABLE customer(

id int PRIMARY KEY,

name varchar(30),

referee\_id int

);

insert into customer values(1, "Will", null);

 insert into customer values(2, "Jane", null);

 insert into customer values(3, "Alex", 2);

  insert into customer values(4, "Bill", null);

   insert into customer values(5, "Zack", 1);

   insert into customer values(6, "Mark", 2);

**Write an SQL query to report the names of the customer that are not referred by the customer with id = 2.**

**Query:**

select name FROM customer where referee\_id !=2 or referee\_id  is null ;

**OUTPUT:**

****

**Q53.**

create TABLE Customers(

id int PRIMARY KEY,

name varchar(30)

  );

insert into Customers values(1,"Joe"),(2, "Henry"),(3, "Sam"),(4,"Max");

 create TABLE Orders(

id int PRIMARY KEY,

customerId INT

 );

 insert  into Orders VALUES(1, 3),(2, 1);

**Write an SQL query to report all customers who never order anything. Return the result table in any order**

**Query:**

SELECT name from Customers where id not in( select customerId  from Orders );

**OUTPUT:**

****

**Q54.**

create table Employee(

employee\_id int,

team\_id int

 );

 INSERT INTO Employee values(1,8),

(2,8),(3,8),

(4,7),

(5,9),

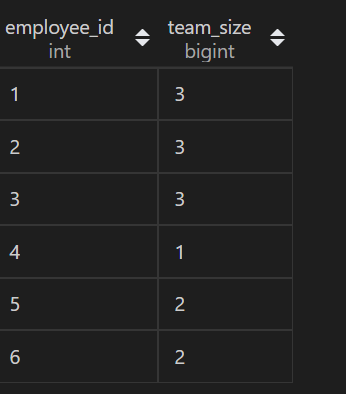
(6,9);

**Write an SQL query to find the team size of each of the employees.**

**Query:**

SELECT employee\_id,count(\*) over( partition by team\_id ) as team\_size FROM  Employee ORDER BY employee\_id ;

**OUTPUT:**

****

**Q55**

CREATE TABLE person(

id int PRIMARY KEY,

name varchar(30),

phone\_number varchar(30)

);

 CREATE TABLE country(

name varchar(30),

country\_code varchar(30) PRIMARY KEY

 );

  CREATE TABLE calls(

 caller\_id int,

callee\_id int,

duration int

  );

insert into person values(3 , "Jonathan", "051-1234567"),(12, "Elvis", "051-7654321"),(1 , "Moncef", "212-1234567"),

(2 , "Maroua", "212-6523651"),(7 , "Meir", "972-1234567"),(9 , "Rachel", "972-0011100");

insert into country  values("Peru", '051'),("Israel", '972'),("Morocco", '212'),("Germany", '049'),("Ethiopia", '251');

insert into calls values (1, 9, 33),(2, 9, 4),(1, 2, 59),(3, 12, 102),(3, 12, 330),(12, 3, 5),(7, 9, 13),(7, 1, 3),(9, 7, 1),(1, 7, 7);

**Write an SQL query to find the countries where this company can invest. Return the result table in any order**

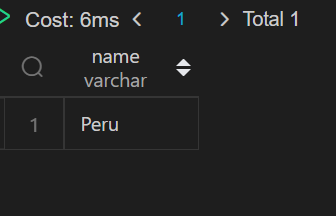
**Query:**

SELECT cc.name from person p inner join  calls c on p.id=c.caller\_id or p.id=c.callee\_id

 inner join country cc on cc.country\_code=left(p.phone\_number,3) group by cc.name having avg(c.duration) >

(select avg(duration) from calls);

**OUTPUT:**

****

**Q56.**

CREATE TABLE Activity(

player\_id int,

device\_id int,

event\_date date,

games\_played int

);

INSERT INTO Activity VALUES(1,2,'2016-03-01',5),(1,2,'2016-05-02',6),(2,3,'2017-06-25',1),

(3,1,'2016-03-02',0),(3,4,'2018-07-03',5);

**Write an SQL query to report the first login date for each player.**

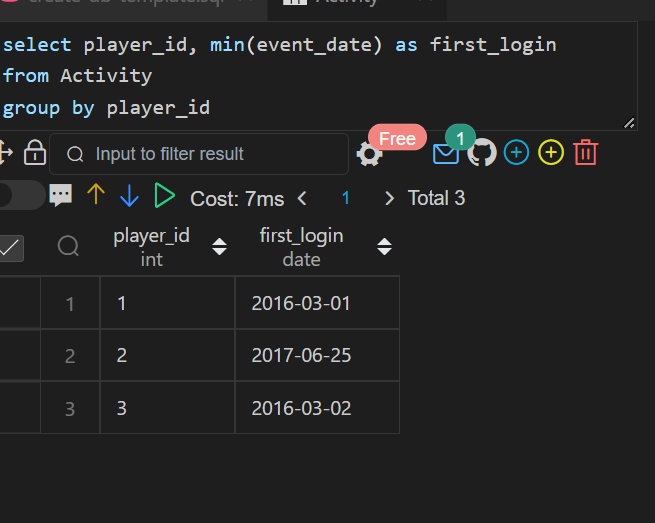
**SQL QUERY:**

select player\_id, min(event\_date) as first\_login

from Activity

group by player\_id;

**OUTPUT:**

****

**Q57.**

reate table Orders(

order\_number int PRIMARY key,

customer\_number int

);

INSERT into Orders VALUES(1,1),(2,2),(3,3),(4,3);

**Write an SQL query to find the customer\_number for the customer who has placed the largest number of orders.**

**SQL QUERY:**

select

 a.customer\_number

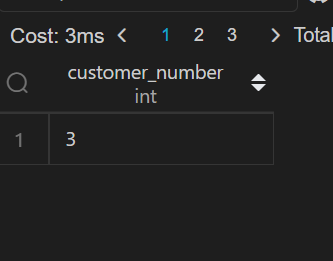
from

 (select customer\_number, count(order\_number) order\_count

  from Orders group by customer\_number) a

order by a.order\_count desc limit 1;

**OUTPUT:**

****

**Q58.**

CREATE TABLE  Cinema(

seat\_id int  PRIMARY KEY AUTO\_INCREMENT,

free bool

);

INSERT INTO Cinema VALUES(1,1),

(2,0),

(3,1),

(4,1),

(5,1);

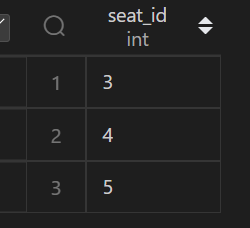
**Write an SQL query to report all the consecutive available seats in the cinema.**

**SQL QUERY:**

SELECT distinct c1.seat\_id from Cinema c1 INNER JOIN Cinema  c2 on

(c1.seat\_id = c2.seat\_id+1) or (c1.seat\_id=c2.seat\_id-1) WHERE c1.free=1 and c1.free=c2.free order by c1.seat\_id asc;

**OUTPUT:**

****

**Q59.**

create TABLE SalesPerson(

sales\_id int PRIMARY KEY,

name varchar(30),

salary int,

commission\_rate int,

hire\_date date

);

CREATE  table Company(

com\_id int PRIMARY KEY,

name varchar(30),

city varchar(30)

);

create Table Orders(

order\_id int PRIMARY KEY,

order\_date date,

com\_id int,

sales\_id int,

amount int,

constraint  fk FOREIGN KEY(com\_id) REFERENCES Company(com\_id)

);

insert into SalesPerson values(1, "John",  100000, 6, STR\_TO\_DATE("4/1/2006","%m/%d/%Y")),(2, "Amy ", 12000, 5,  STR\_TO\_DATE("5/1/2010","%m/%d/%Y")),(3, "Mark",  65000, 12, STR\_TO\_DATE("12/25/2008","%m/%d/%Y")),

(4, "Pam ", 25000, 25,  STR\_TO\_DATE("1/1/2005","%m/%d/%Y")),(5, "Alex",  5000, 10, STR\_TO\_DATE("2/3/2007" ,"%m/%d/%Y"));

insert into Company values(1, "RED", "Boston"),(2, "ORANGE", "New York"),(3, "YELLOW", "Boston"),(4, "GREEN", "Austin");

insert into Orders values(1, STR\_TO\_DATE("1/1/2014","%m/%d/%Y"), 3, 4, 10000),(2, STR\_TO\_DATE("2/1/2014","%m/%d/%Y"), 4, 5, 5000),

(3,STR\_TO\_DATE("3/1/2014","%m/%d/%Y") , 1, 1, 50000),(4, STR\_TO\_DATE("4/1/2014","%m/%d/%Y"), 1, 4, 25000);

**Write an SQL query to report the names of all the salespersons who did not have any orders related to the company with the name "RED".**

**Query:**

SELECT name

FROM SalesPerson

WHERE sales\_id

NOT IN (

    SELECT s.sales\_id FROM Orders o

    INNER JOIN SalesPerson s ON o.sales\_id = s.sales\_id

    INNER JOIN Company c ON o.com\_id = c.com\_id

    WHERE c.name = 'RED'

);

**OUTPUT:**

****

**Q60.**

create DATABASE db;

use db;

create TABLE Triangle (

x int,

y int,

z int,

constraint pk PRIMARY KEY(x,y,z)

);

INSERT INTO Triangle values(13,15,30),(10,20,15);

**Write an SQL query to report for every three line segments whether they can form a triangle.**

**Query:**

SELECT

    x,

    y,

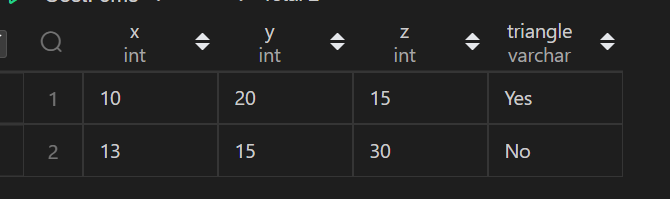
    z,

    IF(x + y > z AND y + z > x AND z + x > y, 'Yes', 'No') triangle

FROM

   Triangle ;

**OUTPUT:**

****

**Q61.**

create table  Point(

x int

);

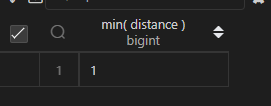
insert into Point  values (-1),(0),(2);

**Write an SQL query to report the shortest distance between any two points from the Point table. The query result format is in the following example.**

**Query:**

SELECT min( distance ) from (select abs(p1.x -p2.x) as distance from Point p1 cross join Point p2 WHERE p1.x <> p2.x ) tmp;

**OUTPUT:**

****

**Q62.**

CREATE TABLE  ActorDirector(

actor\_id int,

director\_id int,

timestamp int primary key

);

insert into ActorDirector  values(1,1,0),(1,1,1),(1,1,2),(1,2,3),(1,2,4),(2,1,5),(2,1,6);

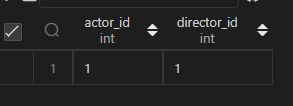
**Write a SQL query for a report that provides the pairs (actor\_id, director\_id) where the actor has cooperated with the director at least three times.**

**Query:**

 select actor\_id,director\_id from (select actor\_id, director\_id,count(\*) as frequent from ActorDirector

 group by actor\_id,director\_id) temp where temp.frequent >=3;

**OUTPUT:**

****

**Q63.**

create Table Sales(

sale\_id int,

product\_id int,

year int,

quantity int,

price int,

constraint pk primary key(sale\_id,year),

constraint fk foreign key(product\_id) references Product(product\_id)

);

 create table Product(

product\_id int primary key,

product\_name varchar(30)

);

insert into Product values(100,"Nokia"),(200,"Apple"),(300,"Samsung");

insert into Sales  values(1,100,2008,10,5000),

(2,100,2009,12,5000),

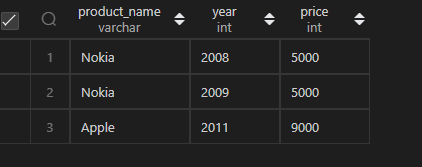
(7,200,2011,15,9000);

**Write an SQL query that reports the product\_name, year, and price for each sale\_id in the Sales table. Return the resulting table in any order.**

**Query:**

SELECT  product\_name,year,price FROM  Product P INNER JOIN Sales S ON P.product\_id = S.product\_id ;

**OUTPUT:**

****

**Q64.**

create Table Project(

project\_id int,

employee\_id int,

constraint pk primary key(project\_id, employee\_id)

);

INSERT INTO Project VALUES(1,1),(1,2),(1,3),(2,1),(2,4);

CREATE TABLE Employee(

employee\_id int PRIMARY KEY,

name varchar(30),

experience\_years int

);

insert into  Employee VALUES(1,'Khaled',3),(2,'Ali',2),(3,'John',1),(4,'Doe',2);

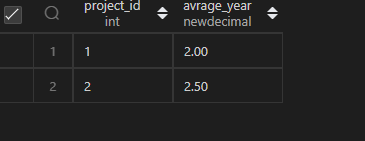
**Write an SQL query that reports the average experience years of all the employees for each project, rounded to 2 digits.**

**Query:**

SELECT  project\_id,round(avg(experience\_years),2) as avrage\_year from Project p inner join  Employee e on p.employee\_id =e.employee\_id

  group by project\_id;

**OUTPUT:**

****

**Q65.**

CREATE Table Sales(

seller\_id int,

product\_id int,

buyer\_id int,

sale\_date date,

quantity int,

price int,

constraint fk FOREIGN KEY(product\_id) REFERENCES Product(product\_id)

);

insert into Product values (1,'S8',1000),(2,'G4',800),(3,'iPhone',1400);

insert into Sales  values (1,1,1,'2019-01-21',2,2000),(1,2,2,'2019-02-17',1,800),(2,2,3,'2019-06-02',1,800),(3,3,4,'2019-05-13',2,2800);

**Write an SQL query that reports the best seller by total sales price, If there is a tie, report them all. Return the result table in any order.**

**Query:**

select seller\_id from Sales group by seller\_id

    having sum(price)=(select max(price) FROM  Sales );

Another way

select seller\_id from (select seller\_id, rank() over(order by sum(price) DESC) as rk from Sales group by seller\_id)tmp where tmp.rk=1

**OUTPUT:**

****

**Q66.**

CREATE Table Product(

product\_id int PRIMARY KEY,

product\_name varchar(30),

unit\_price int

);

CREATE Table Sales(

seller\_id int,

product\_id int,

buyer\_id int,

sale\_date date,

quantity int,

price int,

constraint fk FOREIGN KEY(product\_id) REFERENCES Product(product\_id)

);

insert into Product values (1,'S8',1000),(2,'G4',800),(3,'iPhone',1400);

insert into Sales  values (1,1,1,'2019-01-21',2,2000),(1,2,2,'2019-02-17',1,800),(2,2,3,'2019-06-02',1,800),(3,3,3,'2019-05-13',2,2800);

**Write an SQL query that reports the buyers who have bought S8 but not iPhone. Note that S8 and iPhone are products present in the Product table.**

**Query:**

select s.buyer\_id

from Product p

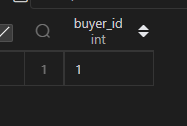
join Sales s

on p.product\_id=s.product\_id

group by buyer\_id

having sum(p.product\_name='S8') >=1  and sum(p.product\_name = 'iPhone') =0 ;

**OUTPUT:**

****

**Q67.**

create Table Customer(

customer\_id int,

name varchar(30),

visited\_on date,

amount int,

constraint pk PRIMARY KEY(customer\_id, visited\_on)

);

INSERT INTO Customer VALUES(1,'Jhon','2019-01-01',100),(2,'Daniel','2019-01-02',110),

(3,'Jade','2019-01-03',120),(4,'Khaled','2019-01-04',130),(5,'Winston','2019-01-05',110),

(6,'Elvis','2019-01-06',140),(7,'Anna','2019-01-07',150),(8,'Maria','2019-01-08',80),

(9,'Jaze','2019-01-09',110),(1,'Jhon','2019-01-10',130),(3,'Jade','2019-01-10',150);

**You are the restaurant owner and you want to analyse a possible expansion (there will be at least one customer every day)**

**Query:**

select c1.visited\_on, sum(c2.amount) as amount,

    round(avg(c2.amount), 2) as average\_amount

from (select visited\_on, sum(amount) as amount

      from Customer group by visited\_on) c1

join (select visited\_on, sum(amount) as amount

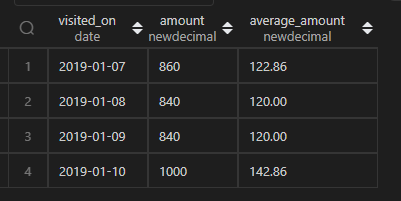
      from Customer group by visited\_on) c2

on datediff(c1.visited\_on, c2.visited\_on) between 0 and 6

group by c1.visited\_on

having count(c2.amount) = 7 ORDER BY c1.visited\_on;

**OUTPUT:**

****

**Q68.**

CREATE TABLE Scores(

player\_name varchar(30),

gender varchar(30),

day date,

score\_points int,

constraint pk PRIMARY KEY((gender, day)

);

insert into Scores values('Aron','F','2020-01-01', 17),

('Alice','F','2020-01-07',23),

('Bajrang','M','2020-01-07',7),

('Khali','M','2019-12-25',11),

('Slaman','M','2019-12-30', 13),

('Joe','M','2019-12-31', 3),

('Jose','M','2019-12-18',2),

('Priya','F','2019-12-31',23),

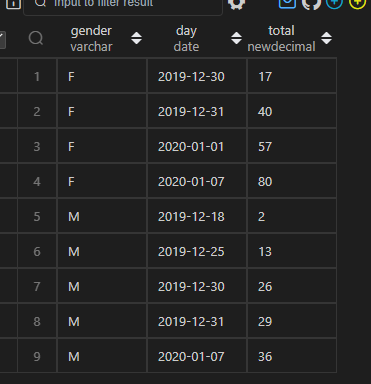
('Priyanka','F','2019-12-30',17);

**Write an SQL query to find the total score for each gender on each day.**

**Query:**

select gender,day, sum(score\_points) over(partition by gender order by gender,day  rows BETWEEN unbounded preceding and current row) as total from  Scores;

**OUTPUT:**

****

**Q69.**

select gender,day, sum(score\_points) over(partition by gender order by gender,day  rows BETWEEN unbounded preceding and current row) as total from  Scores;

create Table Logs(

log\_id int

);

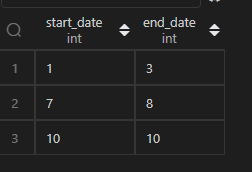
insert into Logs VALUES(1),(2),(3),(7),(8),(10);

**Write an SQL query to find the total score for each gender on each day.**

**Query:**

select min(log\_id) as start\_date ,max(log\_id)as end\_date from (SELECT log\_id,log\_id-row\_number() over(order by log\_id ) as diff from Logs) cte GROUP BY cte.diff ORDER BY start\_date;

**OUTPUT:**

****

**70)**

CREATE Table Students(

student\_id int PRIMARY KEY,

student\_name varchar(30)

);

CREATE Table Subjects(

subject\_name varchar(30) PRIMARY KEY

);

 CREATE Table Examinations(

student\_id int,

subject\_name varchar(30)

 );

 INSERT INTO Students VALUES(1,'Alice'),(2,'Bob'),(13,'John'),(6,'Alex');

insert into Subjects VALUES('Math'),('Physics'),('Programming');

INSERT INTO Examinations VALUES (1,'Math'),(1,'Physics'),(1,'Programming'),

(2,'Programming'),(1,'Physics'),(1,'Math'),(13,'Math'),(13,'Programming'),

(13,'Physics'),(2,'Math'),(1,'Math');

**Write an SQL query to find the number of times each student attended each exam.**

**Query:**

select cte1.student\_id,cte1.subject\_name,

CASE WHEN cnt IS NOT NULL THEN cnt

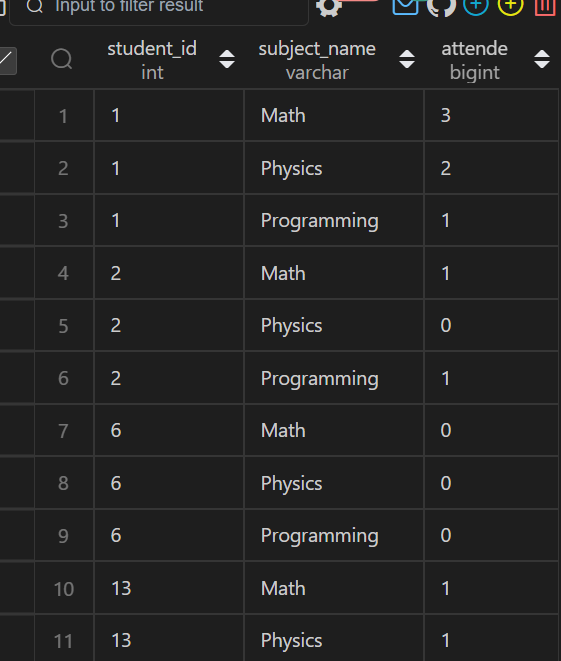
ELSE 0 END AS attende

 from (SELECT  student\_id, subject\_name,student\_name from  Students cross JOIN Subjects)cte1 left JOIN

(SELECT student\_id ,subject\_name,count(student\_id) as cnt from Examinations GROUP BY student\_id ,subject\_name)cte2 on cte1.student\_id=cte2.student\_id and cte1.subject\_name=cte2.subject\_name

order by cte1.student\_id,cte1.subject\_name;

**Output:**

****

**Q71.**

create  TABLE Employees(

employee\_id int ,

employee\_name varchar(30),

manager\_id int

);

INSERT INTO  Employees VALUES(1,'Boss',1),(3,'Alice',3),

(2,'Bob',1),

(4,'Daniel',2),

(7,'Luis',4),

(8,'Jhon',3),

(9,'Angela',8),

(77,'Robert',1);

**Write an SQL query to find employee\_id of all employees that directly or indirectly report their work to the head of the company**

**Query:**

select a.employee\_id as EMPLOYEE\_ID

from

    Employees as a

    left join

    Employees as b on a.manager\_id = b.employee\_id

    left join

    Employees as c on b.manager\_id = c.employee\_id

    left join

    Employees as d on c.manager\_id = d.employee\_id

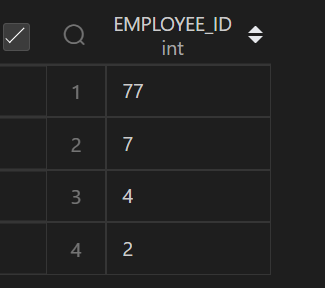
where

    a.employee\_id != 1

    and

    d.employee\_id = 1;

**Output:**

****

**Q72.**

create Table Transactions(

id int PRIMARY KEY,

country varchar(30),

state enum("approved", "declined"),

amount int,

trans\_date date

);

INSERT INTO  Transactions VALUES (121,'US', 'approved',1000,'2018-12-18'),

(122,'US','declined',2000,'2018-12-19'),

(123,'US','approved',2000,'2019-01-01'),

(124,'DE','approved',2000,'2019-01-07');

**Write an SQL query to find for each month and country, the number of transactions and their total amount, the number of approved transactions and their total amount.**

**Query:**

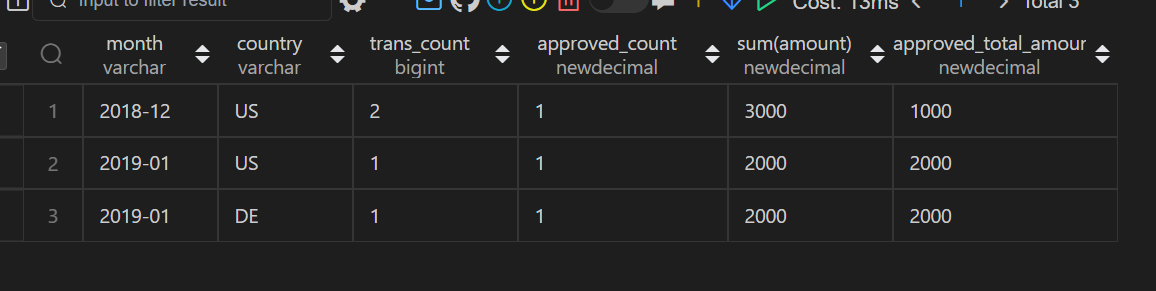
select date\_format(trans\_date,'%Y-%m') as month,country,count(\*) as trans\_count,sum(if(state='approved',1,0))as approved\_count,sum(amount),

sum(if(state = 'approved', amount, 0)) as approved\_total\_amount

    from Transactions

GROUP BY date\_format(trans\_date,'%Y-%m'),country;

**Output:**

****

**Q73.**

create Table Actions(

user\_id int,

post\_id int,

action\_date date,

action enum('view', 'like', 'reaction', 'comment', 'report', 'share'),

extra varchar(30)

 );

 create Table Removals(

post\_id int,

remove\_date date

 );

insert into Actions values(1,1,'2019-07-01', 'view','null'),

(1,1,'2019-07-01','like','null'),

(1,1,'2019-07-01','share','null'),

(2,2,'2019-07-04','view','null'),

(2,2,'2019-07-04','report','spam'),

(3,4,'2019-07-04','view','null'),

(3,4,'2019-07-04','report','spam'),

(4,3,'2019-07-02','view','null'),

(4,3,'2019-07-02','report','spam'),

(5,2,'2019-07-03','view','null'),

(5,2,'2019-07-03','report','racism'),

(5,5,'2019-07-03','view','null'),

(5,5,'2019-07-03','report','racism');

insert into Removals values(2,'2019-07-20'),(3,'2019-07-18');

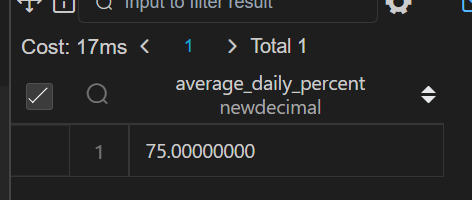
**Write an SQL query to find the average daily percentage of posts that got removed after being reported as spam, rounded to 2 decimal places.**

**Query:**

select avg(daily\_percentage)as average\_daily\_percent from

(select count(distinct b.post\_id)/count(distinct a.post\_id)\*100 as daily\_percentage from  Actions a  left join Removals b  on a.post\_id= b.post\_id where a.extra='spam' GROUP BY action\_date)temp;

**Output:**

****

**Q74.**

CREATE TABLE Activity(

player\_id int,

device\_id int,

event\_date date,

games\_played int

);

INSERT INTO Activity VALUES(1,2,'2016-03-01',5),(1,2,'2016-05-02',6),(2,3,'2017-06-25',1),

(3,1,'2016-03-02',0),(3,4,'2018-07-03',5);

**Write an SQL query to report the fraction of players that logged in again on the day after the day they first logged in, rounded to 2 decimal places.**

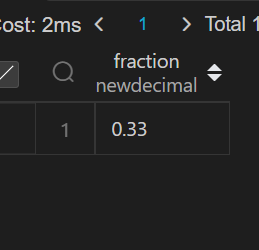
**SQL QUERY:**

select  round(count(cte.player\_id)/(select count(distinct player\_id) from Activity) ,2)as fraction

from (SELECT player\_id,min(event\_date) as start\_date from Activity GROUP BY player\_id) as cte  inner join Activity a

on cte.player\_id=a.player\_id  and datediff(cte.start\_date,a.event\_date)=-1;

**OUTPUT:**

****

**Q75.**

CREATE TABLE Activity(

player\_id int,

device\_id int,

event\_date date,

games\_played int

);

INSERT INTO Activity VALUES(1,2,'2016-03-01',5),(1,2,'2016-05-02',6),(2,3,'2017-06-25',1),

(3,1,'2016-03-02',0),(3,4,'2018-07-03',5);

**Write an SQL query to report the fraction of players that logged in again on the day after the day they first logged in, rounded to 2 decimal places.**

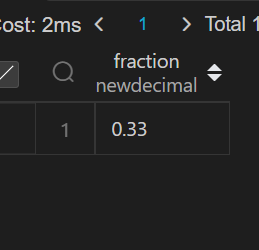
**SQL QUERY:**

select  round(count(cte.player\_id)/(select count(distinct player\_id) from Activity) ,2)as fraction

from (SELECT player\_id,min(event\_date) as start\_date from Activity GROUP BY player\_id) as cte  inner join Activity a

on cte.player\_id=a.player\_id  and datediff(cte.start\_date,a.event\_date)=-1;

**OUTPUT:**

****

**Q76.**

create Table Salaries(

company\_id int,

employee\_id int,

employee\_name varchar(30),

salary int,

constraint primary key(company\_id, employee\_id)

);

insert into Salaries values(1,1,'Tony',2000),

(1,2,'Pronub',21300),

(1,3,'Tyrrox',10800),

(2,1,'Pam',300),

(2,7,'Bassem',450),

(2,9,'Hermione',700),

(3,7,'Bocaben',100),

(3,2,'Ognjen',2200),

(3,13,'Nyan Cat',3300),

(3,15,'Morning Cat',7777);

**Write an SQL query to find the salaries of the employees after applying taxes. Round the salary to the nearest integer.**

**Query:**

select company\_id, employee\_id, employee\_name, round(salary - salary\*tax, 0) as salary

from

(

  select \*,

  case when max(salary) over(partition by company\_id) < 1000 then 0

       when max(salary) over(partition by company\_id) between 1000

       and 10000 then 0.24

       else 0.49 end as tax

  from Salaries

) x ;

**Output:**

**Q77.**

CREATE Table Sales(

sale\_date date,

fruit enum("apples","oranges"),

sold\_num int,

CONSTRAINT PK PRIMARY KEY(sale\_date, fruit)

);

INSERT INTO Sales VALUES ('2020-05-01','apples', 10),

('2020-05-01', 'oranges',8),

('2020-05-02','apples', 15),

('2020-05-02','oranges',15),

('2020-05-03','apples',20),

('2020-05-03', 'oranges',0),

('2020-05-04','apples',15),

('2020-05-04','oranges',16);

**Write an SQL query to report the difference between the number of apples and oranges sold each day. Return the result table ordered by sale\_date.**

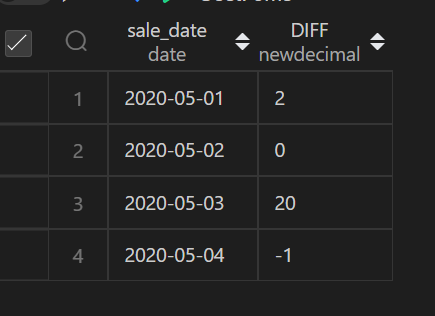
**SQL QUERY:**

SELECT sale\_date, SUM(CASE WHEN fruit='apples' THEN sold\_num

                           WHEN fruit='oranges' THEN -sold\_num end ) AS DIFF

 FROM Sales GROUP BY sale\_date ;

**OUTPUT:**

****

**Q78.**

create Table Variables(

name varchar(30) primary key ,

value int

);

create Table Expressions(

left\_operand varchar(30),

operator enum('<', '>', '='),

right\_operand varchar(30),

constraint fk primary key(left\_operand, operator, right\_operand)

);

insert into Variables values('x',66),('y',77);

**Write an SQL query to evaluate the boolean expressions in Expressions table. Return the result table in any order**

**SQL QUERY:**

SELECT e.\*,case when operator = '=' and v1.value=v2.value then 'true'

                when operator = '<' and v1.value < v2.value then 'true'

                 when operator = '>' and v1.value > v2.value then 'true'

                else 'false'

                end as value

from Expressions e left join Variables  v1 on e.left\_operand = v1.name

left join  Variables v2  on e.right\_operand = v2.name;

**OUTPUT:**

****

**Q79.**

create Table Movies(

movie\_id int  primary key,

title varchar(30)

);

CREATE Table Users(

user\_id int primary key,

name varchar(30)

);

create Table MovieRating(

movie\_id int,

user\_id int,

rating int,

created\_at date,

constraint pk PRIMARY KEY(movie\_id, user\_id)

);

INSERT INTO Movies  VALUES(1,'Avengers'),(2,'Frozen2'),(3,'Joker');

INSERT INTO Users values(1,'Daniel'),(2,'Monica'),(3,'Maria'),(4,'James');

insert into MovieRating values  (1,1,3,'2020-01-12'),

(1,2,4,'2020-02-11'),

(1, 3, 2, '2020-02-12'),

(1, 4, 1,'2020-01-01'),

(2,1,5,'2020-02-17'),

(2,2,2,'2020-02-01'),

(2,3,2,'2020-03-01'),

(3,1,3,'2020-02-22'),

(3,2,4,'2020-02-25');

**Write an SQL query to:**

**● Find the name of the user who has rated the greatest number of movies. In case of a tie, return the lexicographically smaller user name.**

**● Find the movie name with the highest average rating in February 2020. In case of a tie, return the lexicographically smaller movie name**

**Query**

(select name from

 MovieRating m left join Users u  on  m.user\_id=u.user\_id

  GROUP BY name order by count(\*) desc, name LIMIT 1

)

UNION

  (SELECT

    m.title

FROM MovieRating as mr JOIN Movies as m

ON mr.movie\_id = m.movie\_id

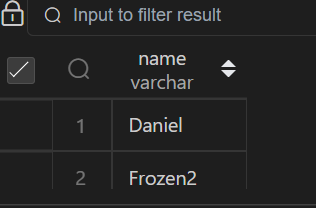
WHERE DATE\_FORMAT(created\_at, '%Y-%m') = '2020-02'

GROUP BY 1

ORDER BY AVG(rating) DESC, 1

LIMIT 1)

**OUTPUT**

****

**80)**

CREATE TABLE person(

id int PRIMARY KEY,

name varchar(30),

phone\_number varchar(30)

);

 CREATE TABLE country(

name varchar(30),

country\_code varchar(30) PRIMARY KEY

 );

  CREATE TABLE calls(

 caller\_id int,

callee\_id int,

duration int

  );

insert into person values(3 , "Jonathan", "051-1234567"),(12, "Elvis", "051-7654321"),(1 , "Moncef", "212-1234567"),

(2 , "Maroua", "212-6523651"),(7 , "Meir", "972-1234567"),(9 , "Rachel", "972-0011100");

insert into country  values("Peru", '051'),("Israel", '972'),("Morocco", '212'),("Germany", '049'),("Ethiopia", '251');

insert into calls values (1, 9, 33),(2, 9, 4),(1, 2, 59),(3, 12, 102),(3, 12, 330),(12, 3, 5),(7, 9, 13),(7, 1, 3),(9, 7, 1),(1, 7, 7);

**Write an SQL query to find the countries where this company can invest. Return the result table in any order**

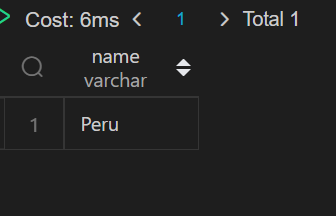
**Query:**

SELECT cc.name from person p inner join  calls c on p.id=c.caller\_id or p.id=c.callee\_id

 inner join country cc on cc.country\_code=left(p.phone\_number,3) group by cc.name having avg(c.duration) >

(select avg(duration) from calls);

**OUTPUT:**

****

**Q81.**

CREATE TABLE STUDENTS (

    ID INTEGER,

    NAME VARCHAR(30),

    MARKS VARCHAR(30)

);

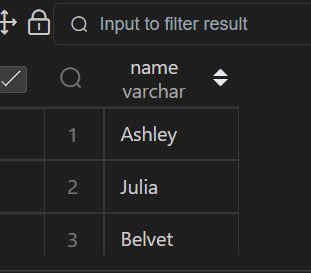
insert into STUDENTS  VALUES(1,'Ashley',81),(2,'Samantha',75),(4,'Julia',76),(3,'Belvet',84);

**Query the Name of any student in STUDENTS who scored higher than 75 Marks.**

**Query:**

SELECT name from STUDENTS  where MARKS >75 order by right(name,3);

**OUTPUT:**

****

**Q82.**

CREATE TABLE EMPLOYEE(

    employee\_id INTEGER,

    name varchar(30),

    months INTEGER,

    salary INTEGER

);

insert into EMPLOYEE VALUES(1228,'Rose',15,1968),(33645,'Angela',1,3443),(45692,'Frank',17,1608),

(56118,'Pratick',7,1345),(59725,'Lisa',11,2330),

(74197,'Kimberly',16,4372);

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

**Query:**

select \* from EMPLOYEE order by name;

**OUTPUT:**

****

**Q83.**

CREATE TABLE EMPLOYEE(

    employee\_id INTEGER,

    name varchar(30),

    months INTEGER,

    salary INTEGER

);

insert into EMPLOYEE VALUES(1228,'Rose',15,1968),(33645,'Angela',1,3443),(45692,'Frank',17,1608),

(56118,'Pratick',7,1345),(59725,'Lisa',11,2330),

(74197,'Kimberly',16,4372),(78454,'Bonnie',8,1771),

(83565,'Michele',6,2017),(98607,'Todd',5,3396),

(99989,'Joe',9,3573);

**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**.

**Query:**

select \* from EMPLOYEE where salary > 2000 and months < 10 order by  employee\_id  ;

**OUTPUT:**

****

**Q84.**

create table TRIANGLES(

    A integer,

    B integer,

    C integer

);

insert into TRIANGLES VALUES(20,20,23),(20,20,20),(20,21,22),(13,14,30);

Write a query identifying the type of each record in the TRIANGLES table using its three side lengths.

**Query:**

SELECT CASE

WHEN A + B <= C OR A + C <= B OR B + C <= A THEN 'Not A Triangle'

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

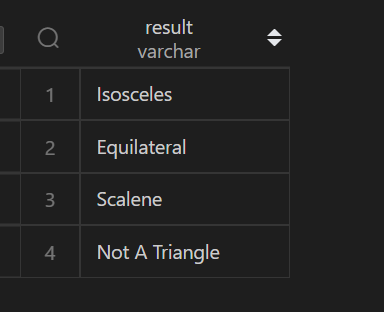
WHEN A = B OR B = C OR A = C THEN 'Isosceles'

ELSE 'Scalene'

END as result

FROM TRIANGLES;

**OUTPUT:**

****

**85)**

create table user\_transactions(

transaction\_id integer,

product\_id integer,

spend decimal,

transaction\_date datetime

);

;

insert into user\_transactions VALUES(1341,123424,1500.60,STR\_TO\_DATE("12/31/2019 12:00:00",'%m/%d/%Y %T')),

(1423,123424,1000.20,STR\_TO\_DATE('12/31/2020 12:00:00','%m/%d/%Y %T')),

(1623,123424,1246.44,STR\_TO\_DATE('12/31/2021 12:00:00','%m/%d/%Y %T')),

(1322,123424,2145.32,STR\_TO\_DATE('12/31/2022 12:00:00','%m/%d/%Y %T'));

**Write a query to obtain the year-on-year growth rate for the total spend of each product for each year.**

**Query:**

WITH yearsum AS

(

SELECT EXTRACT(YEAR FROM transaction\_date) as year, product\_id, SUM(spend) as spend

FROM user\_transactions

GROUP BY 1,2

)

SELECT a.year, a.product\_id,

a.spend as curr\_year\_spend,

b.spend as prev\_year\_spend,

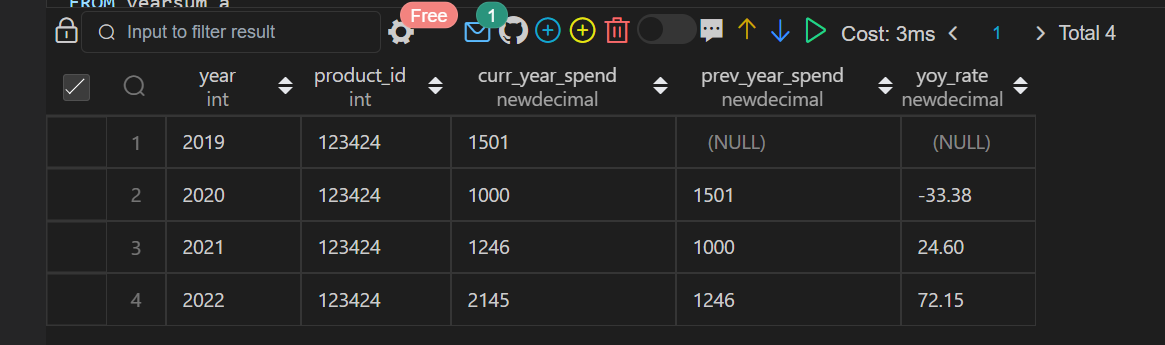
ROUND(100.00\*(a.spend - b.spend)/b.spend,2) as yoy\_rate

FROM yearsum a

LEFT JOIN yearsum b ON a.year-1=b.year AND a.product\_id = b.product\_id

ORDER BY 2,1;

**Output:**

****

**86)**

 create table inventory(

item\_id integer,

item\_type varchar(30),

item\_category varchar(30),

square\_footage decimal

 );

insert into inventory VALUES (1374,'prime\_eligible' ,'mini refrigerator',68.00),

(4245,'not\_prime standing','lamp',26.40),

(2452,'prime\_eligible','television',85.00),

(3255,'not\_prime','side table',22.60),

(1672,'prime\_eligible','laptop',8.50);

**Write a SQL query to find the number of prime and non-prime items that can be stored in the 500,000 square feet warehouse. Output the item type and number of items to be stocked.**

**Query:**

SELECT item\_type,

case

when item\_type = 'prime\_eligible'

then Floor(500000/sum(square\_footage))\*count(item\_type)

else floor((500000 -(select(floor(500000/sum(square\_footage)))\*sum(square\_footage) from inventory where item\_type = 'prime\_eligible'))/sum(square\_footage))\*Count(item\_type)

end

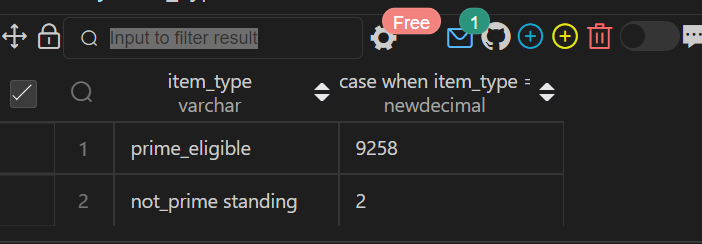
from

inventory

group by item\_type

order by item\_type desc;

**Output:**

****

**Q87)**

Create Table user\_actions (

users\_id int,

event\_id int,

event\_type varchar(50),

event\_date datetime );

nsert into user\_actions Values(445, 7765 ,'sign-in',STR\_TO\_DATE('05/31/2022 12:00:00','%m/%d/%Y %T'));

Insert into user\_actions Values(445, 3634 ,'like',STR\_TO\_DATE('06/05/2022 12:00:00','%m/%d/%Y %T'));

Insert into user\_actions Values(742, 6458 ,'sign-in', STR\_TO\_DATE('07/03/2022 12:00:00','%m/%d/%Y %T'));

Insert into user\_actions Values(742, 1374 ,'comment', STR\_TO\_DATE('07/19/2022 12:00:00','%m/%d/%Y %T'));

**Assume you have the table below containing information on Facebook user actions. Write a query to obtain the active user retention in July 2022. Output the month (in numerical format 1, 2, 3) and the number of monthly active users (MAUs).**

**Query:**

SELECT EXTRACT(MONTH FROM a1.event\_date) as month, COUNT(DISTINCT a1.users\_id) as monthly\_active\_users

from user\_actions as a1,user\_actions as a2

where a1.users\_id = a2.users\_id AND

EXTRACT(MONTH FROM a1.event\_date) = 7

AND EXTRACT(MONTH FROM a2.event\_date) =6

AND EXTRACT(YEAR FROM a1.event\_date) = 2022

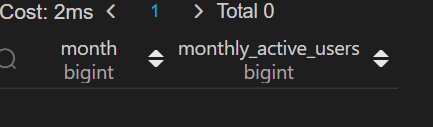
AND EXTRACT(YEAR FROM a2.event\_date) =2022

AND a1.event\_type in ( 'sign-in', 'like', 'comment') AND

a2.event\_type in ('sign-in', 'like', 'comment')

GROUP BY month;

**Output:**

****

**Q88)**

Google's marketing team is making a Superbowl commercial and needs a simple statistic to put on their TV ad: the median number of searches a person made last year. However, at Google scale, querying the 2 trillion searches is too costly. Luckily, you have access to the summary table which tells you the number of searches made last year and how many Google users fall into that bucket.

Query

WITH expanded AS(

  SELECT searches

  FROM search\_frequency

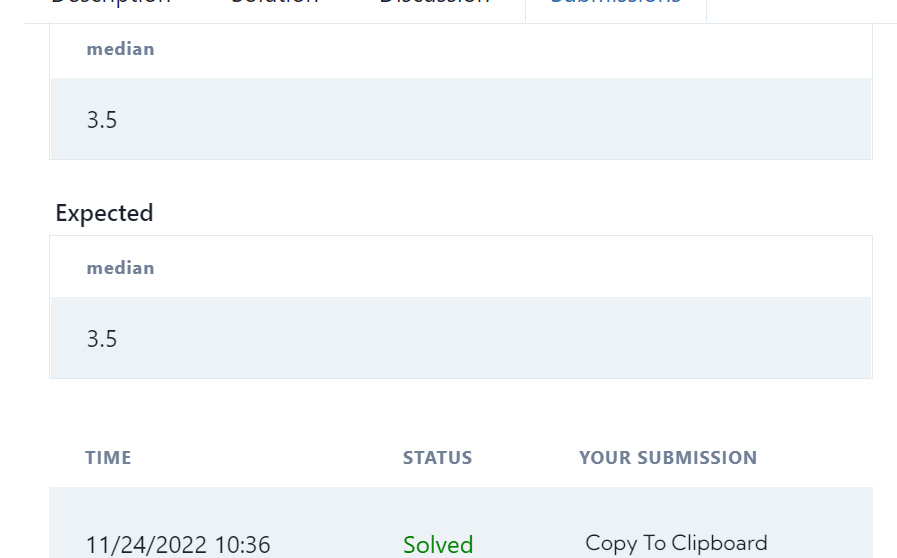
  GROUP BY searches, GENERATE\_SERIES(1,num\_users)

)

SELECT PERCENTILE\_CONT(0.5) WITHIN GROUP (ORDER BY searches) AS median

FROM expanded

**Output**

****

**Q91.**

**Sometimes, payment transactions are repeated by accident; it could be due to user error, API failure or a retry error that causes a credit card to be charged twice. Using the transactions table, identify any payments made at the same merchant with the same credit card for the same amount within 10 minutes of each other. Count such repeated payments. Level - Hard Hint- Use Partition and order by**

Query

select count(t1.merchant\_id) as payment\_count

from transactions t1

join transactions t2

on t1.merchant\_id=t2.merchant\_id

and t1.credit\_card\_id=t2.credit\_card\_id

and t1.amount=t2.amount

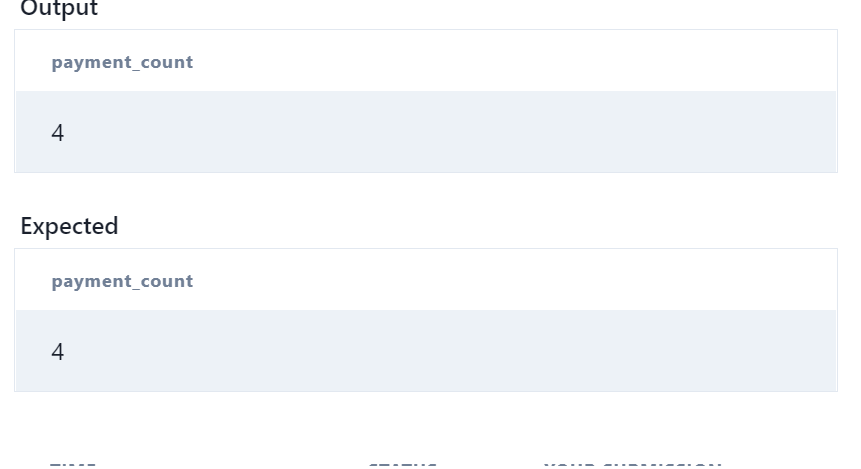
and t1.transaction\_id<t2.transaction\_id

where (date\_part('minute',t2.transaction\_timestamp)-date\_part('minute',t1.transaction\_timestamp))<=10

and  date\_part('hour',t1.transaction\_timestamp)=date\_part('hour',t2.transaction\_timestamp)

;;

**OUTPUT**

****

create Table Scores(

player\_name varchar(30),

gender varchar(30),

day date,

score\_points int,

 constraint pk PRIMARY key(gender, day)

);

insert into Scores VALUES('Aron','F','2020-01-01',17),

('Alice','F','2020-01-07',23),

('Bajrang','M','2020-01-07',7),

('Khali','M','2019-12-25',11),

('Slaman','M','2019-12-30',13),

('Joe','M','2019-12-31',3),

('Jose','M','2019-12-18',2),

('Priya','F','2019-12-31',23),

('Priyanka','F','2019-12-30',1);

**Write an SQL query to find the total score for each gender on each day. Return the result table ordered by gender and day in ascending order.**

**Query**

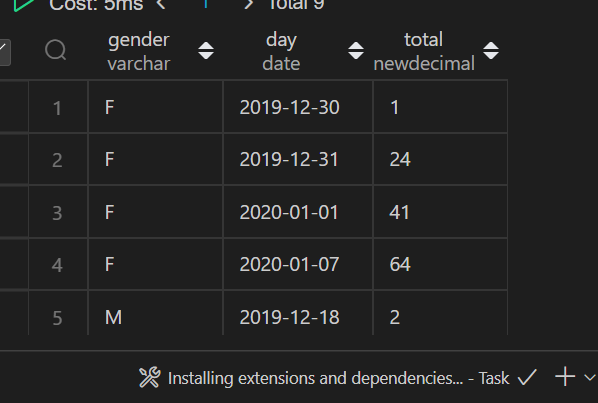
select s.gender, s.day, (select sum(score\_points) from Scores where gender = s.gender and day <= s.day) as total

    from Scores s

    group by gender, day

    order by gender, day;

**Output:**

****

**Q94.**

CREATE TABLE person(

id int PRIMARY KEY,

name varchar(30),

phone\_number varchar(30)

);

 CREATE TABLE country(

name varchar(30),

country\_code varchar(30) PRIMARY KEY

 );

  CREATE TABLE calls(

 caller\_id int,

callee\_id int,

duration int

  );

insert into person values(3 , "Jonathan", "051-1234567"),(12, "Elvis", "051-7654321"),(1 , "Moncef", "212-1234567"),

(2 , "Maroua", "212-6523651"),(7 , "Meir", "972-1234567"),(9 , "Rachel", "972-0011100");

insert into country  values("Peru", '051'),("Israel", '972'),("Morocco", '212'),("Germany", '049'),("Ethiopia", '251');

insert into calls values (1, 9, 33),(2, 9, 4),(1, 2, 59),(3, 12, 102),(3, 12, 330),(12, 3, 5),(7, 9, 13),(7, 1, 3),(9, 7, 1),(1, 7, 7);

**Write an SQL query to find the countries where this company can invest. Return the result table in any order**

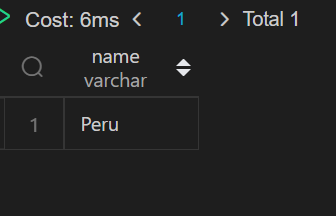
**Query:**

SELECT cc.name from person p inner join  calls c on p.id=c.caller\_id or p.id=c.callee\_id

 inner join country cc on cc.country\_code=left(p.phone\_number,3) group by cc.name having avg(c.duration) >

(select avg(duration) from calls);

**OUTPUT:**

****

Q95.

Create table If Not Exists Numbers (

       Number int,

       Frequency int);

insert into Numbers (Number, Frequency) values ('0', '7');

insert into Numbers (Number, Frequency) values ('1', '1');

insert into Numbers (Number, Frequency) values ('2', '3');

insert into Numbers (Number, Frequency) values ('3', '1');

**Write an SQL query to report the median of all the numbers in the database after decompressing the Numbers table. Round the median to one decimal point**.

**Query:**

select

    avg(number) median

from

   Numbers n

where

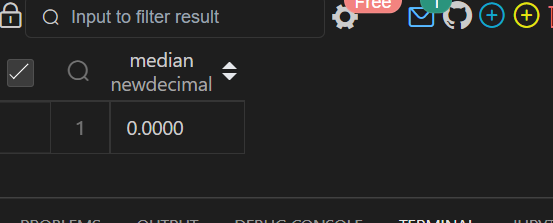
    n.frequency >= abs(

                        (select sum(Frequency) from Numbers where Number<=n.number)

                        -

                        (select sum(Frequency) from Numbers where Number>=n.number));

**OUTPUT:**

****

**Q96**

Create table If Not Exists salary (

    id int,

    employee\_id int,

    amount int,

    pay\_date date);

Create table If Not Exists employee (

    employee\_id int,

    department\_id int);

Truncate table salary;

insert into salary

    (id, employee\_id, amount, pay\_date)

values

    ('1', '1', '9000', '2017/03/31');

insert into salary

    (id, employee\_id, amount, pay\_date)

values

    ('2', '2', '6000', '2017/03/31');

insert into salary

    (id, employee\_id, amount, pay\_date)

values

    ('3', '3', '10000', '2017/03/31');

insert into salary

    (id, employee\_id, amount, pay\_date)

values

    ('4', '1', '7000', '2017/02/28');

insert into salary

    (id, employee\_id, amount, pay\_date)

values

    ('5', '2', '6000', '2017/02/28');

insert into salary

    (id, employee\_id, amount, pay\_date)

values

    ('6', '3', '8000', '2017/02/28');

Truncate table employee;

insert into employee

    (employee\_id, department\_id)

values

    ('1', '1');

insert into employee

    (employee\_id, department\_id)

values

    ('2', '2');

insert into employee

    (employee\_id, department\_id)

values

    ('3', '2');

**Write an SQL query to report the comparison result (higher/lower/same) of the average salary of employees in a department to the company's average salary.**

**Query:**

select

    pay\_month,

    department\_id,

    case when dept\_avg > comp\_avg then 'higher' when dept\_avg < comp\_avg then 'lower' else 'same' end comparison

from (

        select  date\_format(b.pay\_date, '%Y-%m') pay\_month, a.department\_id, avg(b.amount) dept\_avg,  d.comp\_avg

        from employee a

        inner join salary b

            on (a.employee\_id = b.employee\_id)

        inner join (select date\_format(c.pay\_date, '%Y-%m') pay\_month, avg(c.amount) comp\_avg

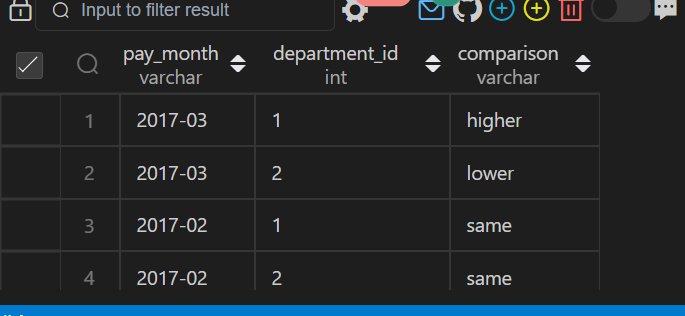
                    from salary c

                    group by date\_format(c.pay\_date, '%Y-%m')) d

            on ( date\_format(b.pay\_date, '%Y-%m') = d.pay\_month)

group by date\_format(b.pay\_date, '%Y-%m'), department\_id, d.comp\_avg) final

**OUTPUT:**

****

**Q97.**

CREATE TABLE Activity(

player\_id int,

device\_id int,

event\_date date,

games\_played int

);

INSERT INTO Activity VALUES(1,2,'2016-03-01',5),(1,2,'2016-05-02',6),(2,3,'2017-06-25',1),

(3,1,'2016-03-02',0),(3,4,'2018-07-03',5);

**Write an SQL query to report the first login date for each player.**

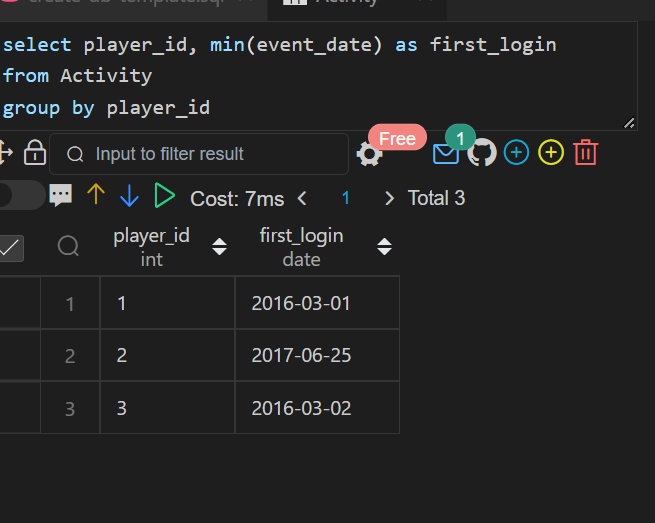
**SQL QUERY:**

select player\_id, min(event\_date) as first\_login

from Activity

group by player\_id;

**OUTPUT:**

****

**Q98.**

CREATE TABLE Players(

player\_id int PRIMARY KEY,

group\_id varchar(30)

);

 CREATE TABLE Matches(

match\_id int primary KEY,

first\_player int,

second\_player int,

first\_score int,

second\_score int

 );

 insert into Players VALUES(15,1),

(25,1),

(30,1),

(45,1),

(10,2),

(35,2),

(50,2),

(20,3),

(40,3);

insert into Matches VALUES(1,15,45,3,0),

(2,30,25,1,2),(3,30,25,1,2),

(4,40,20,5,2),(5,35,50,1,1);

**The winner in each group is the player who scored the maximum total points within the group. In the case of a tie, the lowest player\_id wins. Write an SQL query to find the winner in each group.**

**Query:**

select group\_id,player\_id

from (

    select sc.group\_id group\_id, sc.player\_id player\_id,

       rank() over (partition by sc.group\_id order by sum(sc.score) desc, sc.player\_id asc) as rnk

    from(

        select p.group\_id group\_id,

         p.player\_id player\_id ,

         sum(m.first\_score) as score

        from Players p

        inner join Matches m

        on p.player\_id = m.first\_player

        group by p.group\_id,p.player\_id

        union all

        select p.group\_id group\_id,

         p.player\_id player\_id ,

        sum(second\_score) as score

        from Players p

        inner join Matches m

        on p.player\_id = m.second\_player

        group by p.group\_id,p.player\_id

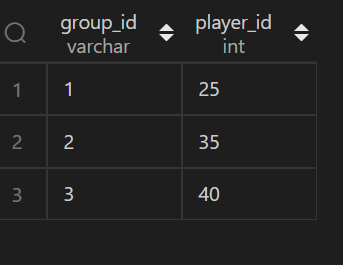
    ) sc

    group by sc.group\_id,sc.player\_id

) A

where rnk = 1;

**OUTPUT:**

****

**Q99)**

CREATE TABLE Student

(student\_id INT,

 student\_name VARCHAR(32));

INSERT INTO Student

VALUES

(1, 'Daniel'),

(2, 'Jade'),

(3, 'Stella'),

(4, 'Jonathan'),

(5, 'Will');

CREATE TABLE Exam

(exam\_id INT,

 student\_id INT,

 score INT);

 INSERT INTO Exam

 VALUES

 (10, 1, 70),

 (10, 2, 80),

 (10, 3, 90),

 (20, 1, 80),

 (30, 1, 70),

 (30, 3, 80),

 (30, 4, 90),

 (40, 1, 60),

 (40, 2, 70),

 (40, 4, 80);

**A quiet student is the one who took at least one exam and did not score the high or the low score. Write an SQL query to report the students (student\_id, student\_name) being quiet in all exams. Do not return the student who has never taken any exam**

**Query:**

WITH TMP AS

 (SELECT DISTINCT(student\_id) AS student\_id

 FROM (SELECT student\_id,

     RANK() OVER(PARTITION BY exam\_id

           ORDER BY Score) AS r1,

     RANK() OVER(PARTITION BY exam\_id

           ORDER BY Score DESC) AS r2

 FROM Exam) AS T

 WHERE r1 = 1 OR r2 = 1),

 TMP1 AS

 (SELECT DISTINCT(student\_id) AS student\_id

  FROM Exam

  WHERE student\_id NOT IN (SELECT student\_id FROM TMP))

SELECT A.student\_id, B.student\_name

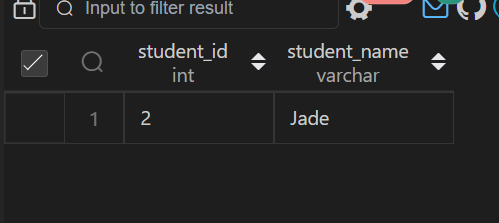
FROM TMP1 AS A

LEFT OUTER JOIN Student AS B

ON A.student\_id = B.student\_id

ORDER BY student\_id;

**OUPUT:**

****

**Q100)**

CREATE TABLE Student

(student\_id INT,

 student\_name VARCHAR(32));

INSERT INTO Student

VALUES

(1, 'Daniel'),

(2, 'Jade'),

(3, 'Stella'),

(4, 'Jonathan'),

(5, 'Will');

CREATE TABLE Exam

(exam\_id INT,

 student\_id INT,

 score INT);

 INSERT INTO Exam

 VALUES

 (10, 1, 70),

 (10, 2, 80),

 (10, 3, 90),

 (20, 1, 80),

 (30, 1, 70),

 (30, 3, 80),

 (30, 4, 90),

 (40, 1, 60),

 (40, 2, 70),

 (40, 4, 80);

**A quiet student is the one who took at least one exam and did not score the high or the low score. Write an SQL query to report the students (student\_id, student\_name) being quiet in all exams. Do not return the student who has never taken any exam**

**Query:**

WITH TMP AS

 (SELECT DISTINCT(student\_id) AS student\_id

 FROM (SELECT student\_id,

     RANK() OVER(PARTITION BY exam\_id

           ORDER BY Score) AS r1,

     RANK() OVER(PARTITION BY exam\_id

           ORDER BY Score DESC) AS r2

 FROM Exam) AS T

 WHERE r1 = 1 OR r2 = 1),

 TMP1 AS

 (SELECT DISTINCT(student\_id) AS student\_id

  FROM Exam

  WHERE student\_id NOT IN (SELECT student\_id FROM TMP))

SELECT A.student\_id, B.student\_name

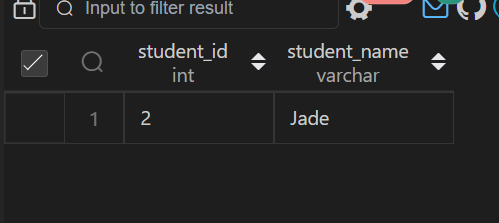
FROM TMP1 AS A

LEFT OUTER JOIN Student AS B

ON A.student\_id = B.student\_id

ORDER BY student\_id;

**OUPUT:**

****

**Q101)**

create table UserActivity(

username varchar(30),

activity varchar(30),

startDate Date,

endDate Date

);

insert into UserActivity VALUES('Alice','Travel', '2020-02-12','2020-02-20'),

('Alice','Dancing','2020-02-21','2020-02-23'),

('Alice','Travel','2020-02-24','2020-02-28'),

('Bob','Travel','2020-02-11','2020-02-18');

**Write an SQL query to show the second most recent activity of each user. If the user only has one activity, return that one. A user cannot perform more than one activity at the same time.**

**Query:**

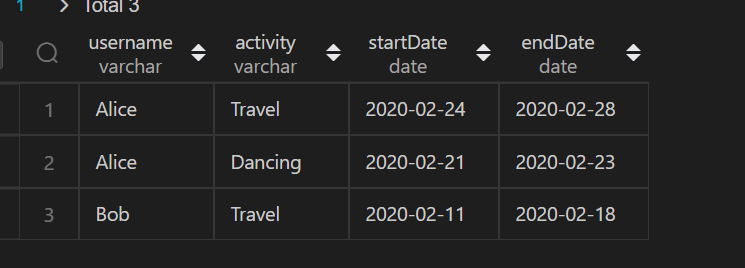
select username,activity,startDate,endDate from  (

select \* , rank()over(partition by username order by startDate desc) as rnk,

count(username) over( partition by username order by startDate desc) as cnt

from UserActivity)tmp  where rnk=2 or cnt=1;

**OUPUT:**

****

**Q102)**

create table UserActivity(

username varchar(30),

activity varchar(30),

startDate Date,

endDate Date

);

insert into UserActivity VALUES('Alice','Travel', '2020-02-12','2020-02-20'),

('Alice','Dancing','2020-02-21','2020-02-23'),

('Alice','Travel','2020-02-24','2020-02-28'),

('Bob','Travel','2020-02-11','2020-02-18');

**Write an SQL query to show the second most recent activity of each user. If the user only has one activity, return that one. A user cannot perform more than one activity at the same time.**

**Query:**

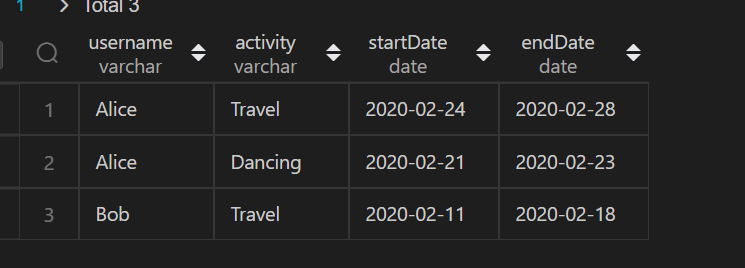
select username,activity,startDate,endDate from  (

select \* , rank()over(partition by username order by startDate desc) as rnk,

count(username) over( partition by username order by startDate desc) as cnt

from UserActivity)tmp  where rnk=2 or cnt=1;

**OUPUT:**

****

**Q103)**

CREATE TABLE STUDENTS (

    ID INTEGER,

    NAME VARCHAR(30),

    MARKS VARCHAR(30)

);

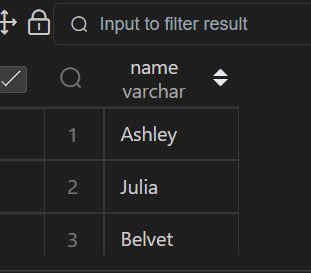
insert into STUDENTS  VALUES(1,'Ashley',81),(2,'Samantha',75),(4,'Julia',76),(3,'Belvet',84);

**Query the Name of any student in STUDENTS who scored higher than 75 Marks.**

**Query:**

SELECT name from STUDENTS  where MARKS >75 order by right(name,3);

**OUTPUT:**

****

**104)**

CREATE TABLE EMPLOYEE(

    employee\_id INTEGER,

    name varchar(30),

    months INTEGER,

    salary INTEGER

);

insert into EMPLOYEE VALUES(1228,'Rose',15,1968),(33645,'Angela',1,3443),(45692,'Frank',17,1608),

(56118,'Pratick',7,1345),(59725,'Lisa',11,2330),

(74197,'Kimberly',16,4372);

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

**Query:**

select \* from EMPLOYEE order by name;

**OUTPUT:**

****

**Q105.**

CREATE TABLE EMPLOYEE(

    employee\_id INTEGER,

    name varchar(30),

    months INTEGER,

    salary INTEGER

);

insert into EMPLOYEE VALUES(1228,'Rose',15,1968),(33645,'Angela',1,3443),(45692,'Frank',17,1608),

(56118,'Pratick',7,1345),(59725,'Lisa',11,2330),

(74197,'Kimberly',16,4372),(78454,'Bonnie',8,1771),

(83565,'Michele',6,2017),(98607,'Todd',5,3396),

(99989,'Joe',9,3573);

**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**.

**Query:**

select \* from EMPLOYEE where salary > 2000 and months < 10 order by  employee\_id  ;

**OUTPUT:**

****

**Q106)**

create table TRIANGLES(

    A integer,

    B integer,

    C integer

);

insert into TRIANGLES VALUES(20,20,23),(20,20,20),(20,21,22),(13,14,30);

Write a query identifying the type of each record in the TRIANGLES table using its three side lengths.

**Query:**

SELECT CASE

WHEN A + B <= C OR A + C <= B OR B + C <= A THEN 'Not A Triangle'

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

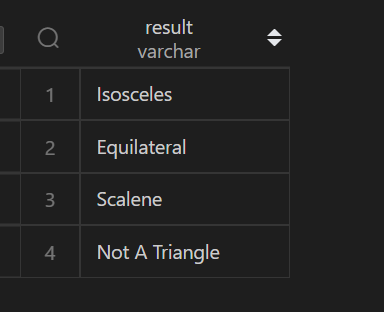
WHEN A = B OR B = C OR A = C THEN 'Isosceles'

ELSE 'Scalene'

END as result

FROM TRIANGLES;

**OUTPUT:**

****

**Q107.**

create table  EMployees(ID int, Name Varchar(20), Salary int);

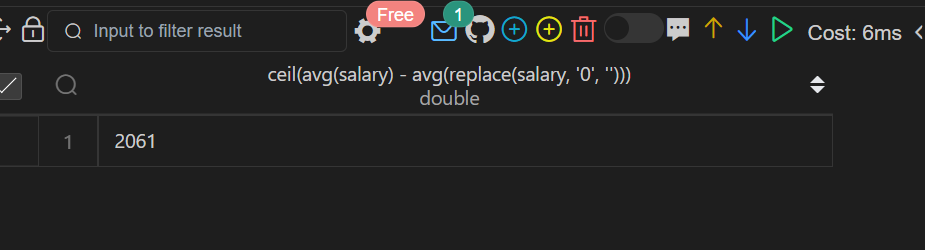
insert into EMployees Values(1,'Kristeen',1420),(2,'Ashley',2006),(3,'Julia',2210),(4,'Maria',3000);

**Write a query calculating the amount of error (i.e.: actual - miscalculated average monthly salaries), and round it up to the next integer.**

**Query:**

select ceil(avg(salary) - avg(replace(salary, '0', ''))) from EMployees;

**Output:**

****

**Q108.**

CREATE TABLE EMPLOYEE(

    employee\_id INTEGER,

    name varchar(30),

    months INTEGER,

    salary INTEGER

);

insert into EMPLOYEE VALUES(1228,'Rose',15,1968),(33645,'Angela',1,3443),(45692,'Frank',17,1608),

(56118,'Pratick',7,1345),(59725,'Lisa',11,2330),

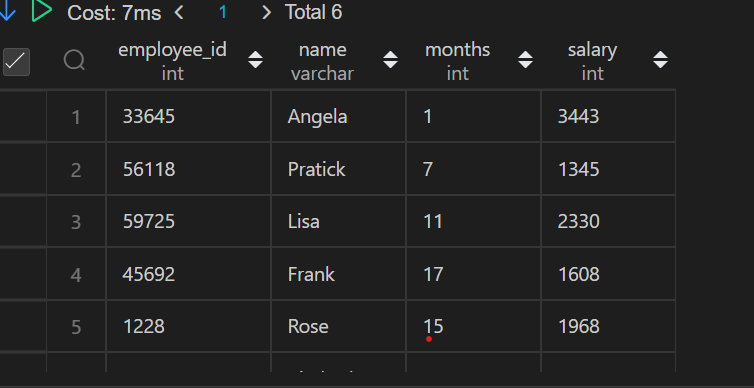
(74197,'Kimberly',16,4372);

**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.**

**Query:**

select \* from EMPLOYEE order by months\*salary;

**Output:**

****

**Q109)**

CREATE TABLE OCCUPATIONS(

    Name  VARCHAR(30),

    Occupation VARCHAR(30)

);

insert into OCCUPATIONS values('julia','Actor'),('Samantha','Doctor'),('Maria','Actor'),('Meera','Singer'),('Ashely','professor'),

('Ketty','Professor'),('Christeen','Professor'),('Jane','Actor'),

('Jenny','Doctor'),('Priya','Singer');

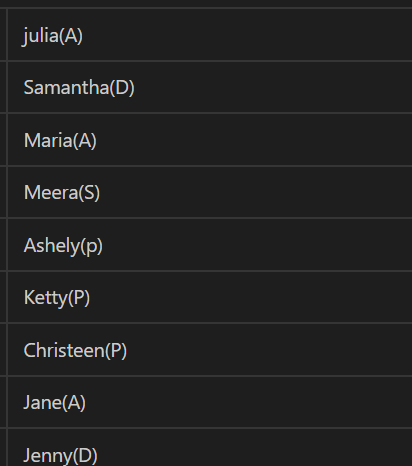
**Query the number of occurrences of each occupation in OCCUPATIONS. Sort the occurrences in ascending order, and output**

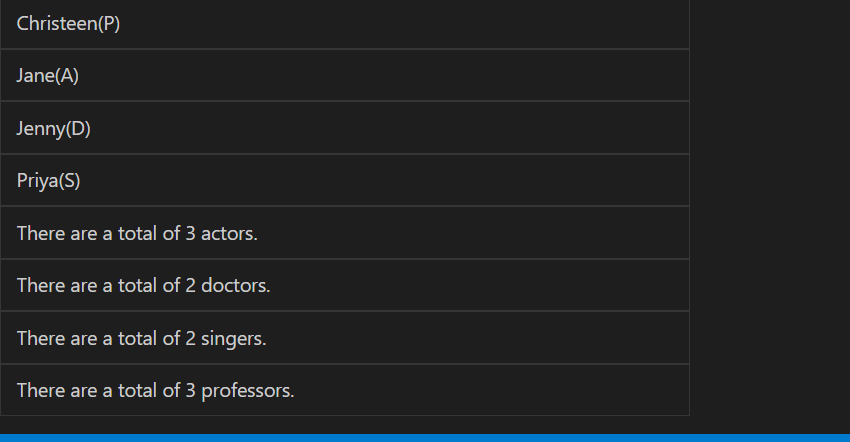
**Query:**

(SELECT CONCAT(Name,'(',SUBSTRING(OCCUPATION,1,1),')') from OCCUPATIONS order by Name asc )  UNION

(SELECT CONCAT('There are a total of ',count(OCCUPATION),' ',lower(OCCUPATION),'s','.') from OCCUPATIONS group by OCCUPATION ORDER BY COUNT(occupation),occupation asc);

**OUTPUT:**

****

****

**Q110 .**

CREATE TABLE OCCUPATIONS(

    Name  VARCHAR(30),

    Occupation VARCHAR(30)

);

insert into OCCUPATIONS values('julia','Actor'),('Samantha','Doctor'),('Maria','Actor'),('Meera','Singer'),('Ashely','professor'),

('Ketty','Professor'),('Christeen','Professor'),('Jane','Actor'),

('Jenny','Doctor'),('Priya','Singer');

**Query the number of occurrences of each occupation in OCCUPATIONS. Sort the occurrences in ascending order, and output**

**Query**

set @d=0,@p=0,@s=0,@a=0;

 select max(dname),max(pname),max(sname),max(aname) from(select

case when Occupation='Doctor' then Name end as dname,

case when Occupation='Professor' then Name end as pname,

case when Occupation='Singer' then Name end as sname,

case when Occupation='Actor' then Name end as aname,

case

    when Occupation='Doctor' then (@d:=@d+1)

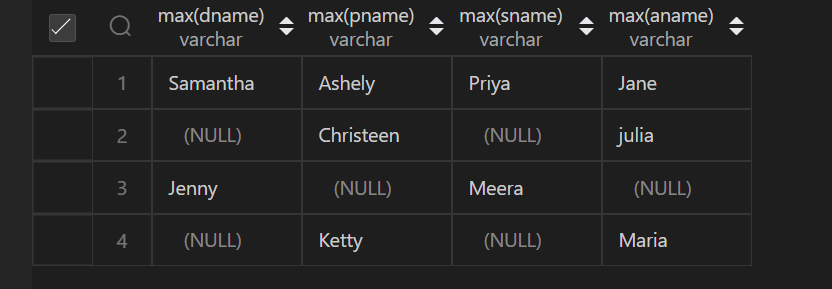
    when Occupation='Professor' then (@p:=@p+1)

    when Occupation='Singer' then (@s:=@s+1)

    when Occupation='Actor' then(@a:=@a+1)

end as count from  OCCUPATIONS order by Name ) as t group by count;

**OUTPUT**

****

**Q111.**

CREATE TABLE BST(

    N INT,

    P INT

);

INSERT INTO BST VALUES(1,2),(3,2),

(6,8),(9,8),(2,5),(8,5),(5,NULL);

**Write a query to find the node type of Binary Tree ordered by the value of the node.**

**Query**

select N,

CASE

 WHEN P is NULL then 'Root'

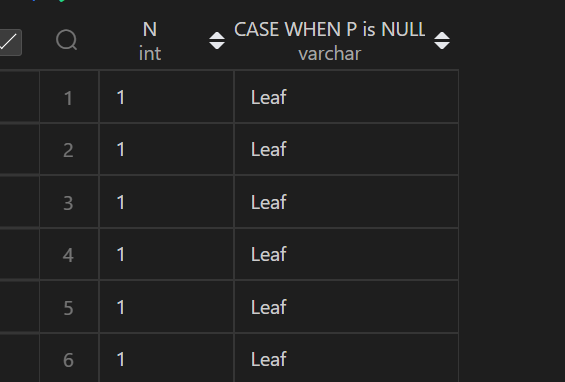
 WHEN N in (select P from BST) then 'Inner'

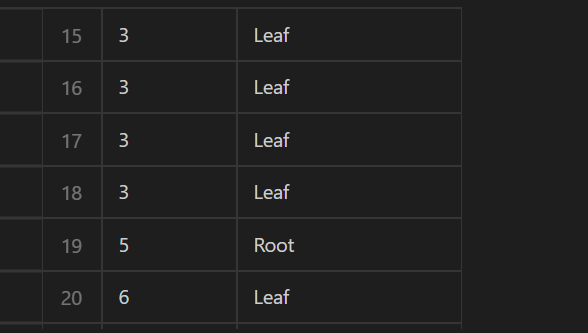
 else 'Leaf'

 end

 from BST order by N;

**OUTPUT**

****

****

**Q113.**

**Write a query to print all prime numbers less than or equal to 1000.**

**Query**

create table prime\_number(numbers int);

DECLARE @nr INT;

DECLARE @divider INT;

DECLARE @prime INT;

select @nr=1;

while @nr < 1000

    BEGIN

    SELECT @divider = @nr-1

    SELECT @prime = 1

    WHILE @divider > 1

        BEGIN

        IF @nr % @divider = 0

            SELECT @prime = 0;

        SELECT @divider = @divider-1

        END

    IF @prime = 1 AND @nr <> 1

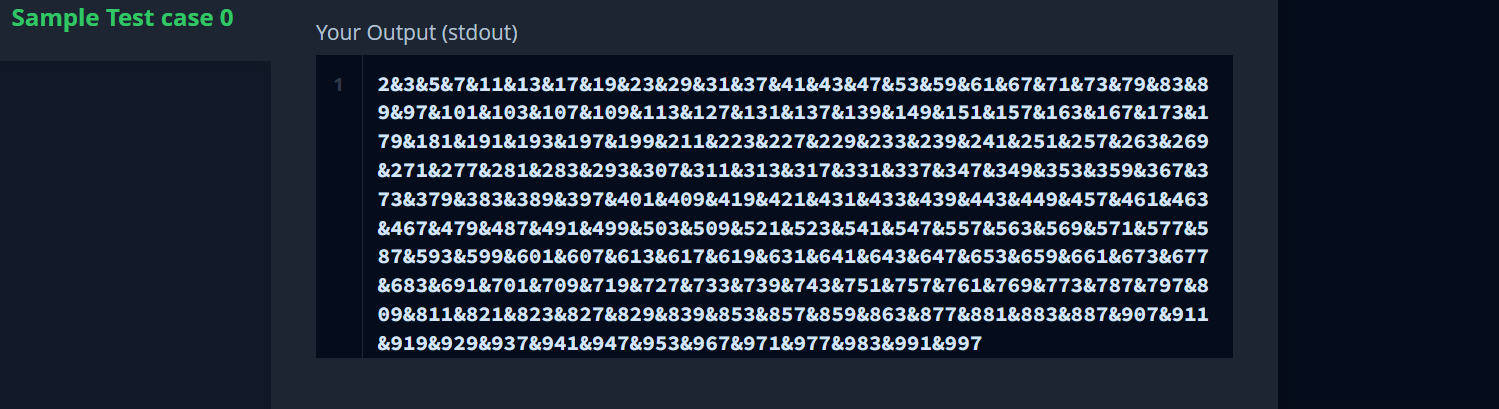
        INSERT INTO prime\_number(numbers) values(@nr);

    SELECT @nr= @nr+1;

    END

SELECT STRING\_AGG(numbers,'&') FROM prime\_number;

**OUTPUT**



**Q114.**

**P(R) represents a pattern drawn by Julia in R rows. The following pattern represents P(5)**

**Query**

DECLARE @I INT;

SELECT @I=1;

WHILE  @I < 5

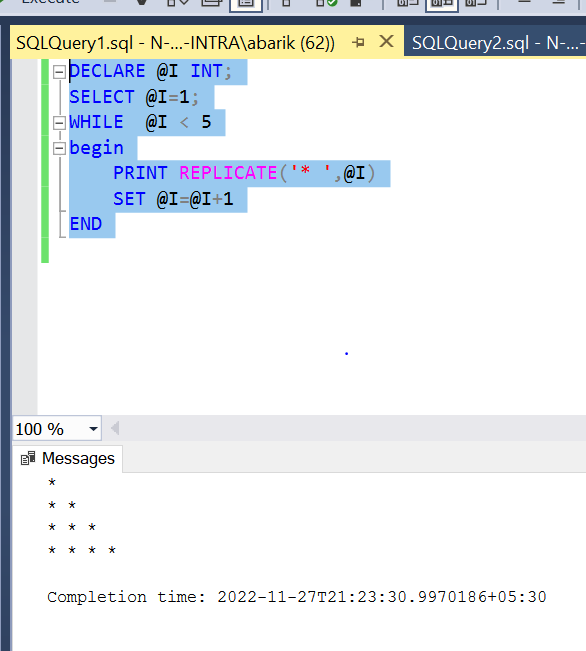
begin

    PRINT REPLICATE('\* ',@I)

    SET @I=@I+1

END

**OUTPUT**



**Q115.**

**P(R) represents a pattern drawn by Julia in R rows. The following pattern represents P(5)**

**Query**

DECLARE @I INT;

SET @I=5;

WHILE  @I >=1

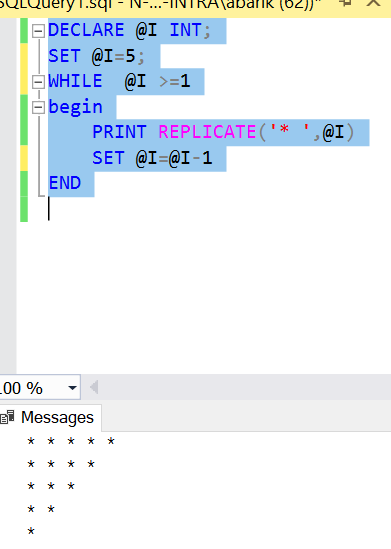
begin

    PRINT REPLICATE('\* ',@I)

    SET @I=@I-1

END

**OUTPUT**



**Q116.**

CREATE TABLE FUN(

   X INT,

   Y INT

   );

   INSERT INTO FUN VALUES(20,20),(20,20),

   (20,21),(23,22),(22,23),(21,20);

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

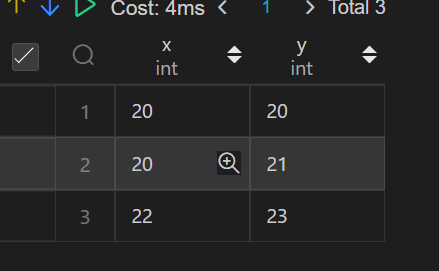
**Query**

 select  distinct A.x,A.y from

(select x,y, row\_number() over(order by x asc) as r from FUN) A join

(select x,y, row\_number() over(order by x asc) as r from FUN) B on A.x=B.y and B.x =A.y  and A.r!=B.r and A.x<=A.y  order by A.x;

**OUTPUT**

****

**Q117.**

CREATE TABLE STUDENTS (

    ID INTEGER,

    NAME VARCHAR(30),

    MARKS VARCHAR(30)

);

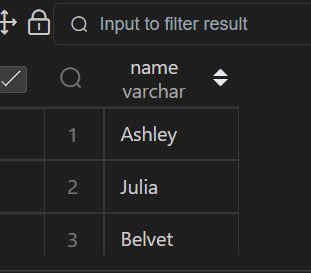
insert into STUDENTS  VALUES(1,'Ashley',81),(2,'Samantha',75),(4,'Julia',76),(3,'Belvet',84);

**Query the Name of any student in STUDENTS who scored higher than 75 Marks.**

**Query:**

SELECT name from STUDENTS  where MARKS >75 order by right(name,3);

**OUTPUT:**

****

**Q118.**

CREATE TABLE EMPLOYEE(

    employee\_id INTEGER,

    name varchar(30),

    months INTEGER,

    salary INTEGER

);

insert into EMPLOYEE VALUES(1228,'Rose',15,1968),(33645,'Angela',1,3443),(45692,'Frank',17,1608),

(56118,'Pratick',7,1345),(59725,'Lisa',11,2330),

(74197,'Kimberly',16,4372);

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

**Query:**

select \* from EMPLOYEE order by name;

**OUTPUT:**

****

**Q119.**

CREATE TABLE EMPLOYEE(

    employee\_id INTEGER,

    name varchar(30),

    months INTEGER,

    salary INTEGER

);

insert into EMPLOYEE VALUES(1228,'Rose',15,1968),(33645,'Angela',1,3443),(45692,'Frank',17,1608),

(56118,'Pratick',7,1345),(59725,'Lisa',11,2330),

(74197,'Kimberly',16,4372),(78454,'Bonnie',8,1771),

(83565,'Michele',6,2017),(98607,'Todd',5,3396),

(99989,'Joe',9,3573);

**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**.

**Query:**

select \* from EMPLOYEE where salary > 2000 and months < 10 order by  employee\_id  ;

**OUTPUT:**

****

**Q120**

create table TRIANGLES(

    A integer,

    B integer,

    C integer

);

insert into TRIANGLES VALUES(20,20,23),(20,20,20),(20,21,22),(13,14,30);

Write a query identifying the type of each record in the TRIANGLES table using its three side lengths.

**Query:**

SELECT CASE

WHEN A + B <= C OR A + C <= B OR B + C <= A THEN 'Not A Triangle'

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

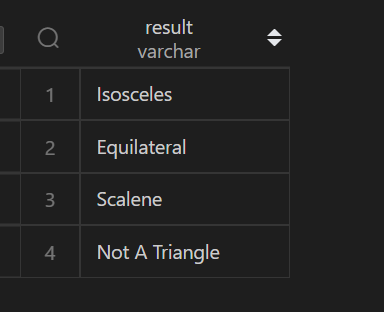
WHEN A = B OR B = C OR A = C THEN 'Isosceles'

ELSE 'Scalene'

END as result

FROM TRIANGLES;

**OUTPUT:**

****

**Q121)**

create table user\_transactions(

transaction\_id integer,

product\_id integer,

spend decimal,

transaction\_date datetime

);

;

insert into user\_transactions VALUES(1341,123424,1500.60,STR\_TO\_DATE("12/31/2019 12:00:00",'%m/%d/%Y %T')),

(1423,123424,1000.20,STR\_TO\_DATE('12/31/2020 12:00:00','%m/%d/%Y %T')),

(1623,123424,1246.44,STR\_TO\_DATE('12/31/2021 12:00:00','%m/%d/%Y %T')),

(1322,123424,2145.32,STR\_TO\_DATE('12/31/2022 12:00:00','%m/%d/%Y %T'));

**Write a query to obtain the year-on-year growth rate for the total spend of each product for each year.**

**Query:**

WITH yearsum AS

(

SELECT EXTRACT(YEAR FROM transaction\_date) as year, product\_id, SUM(spend) as spend

FROM user\_transactions

GROUP BY 1,2

)

SELECT a.year, a.product\_id,

a.spend as curr\_year\_spend,

b.spend as prev\_year\_spend,

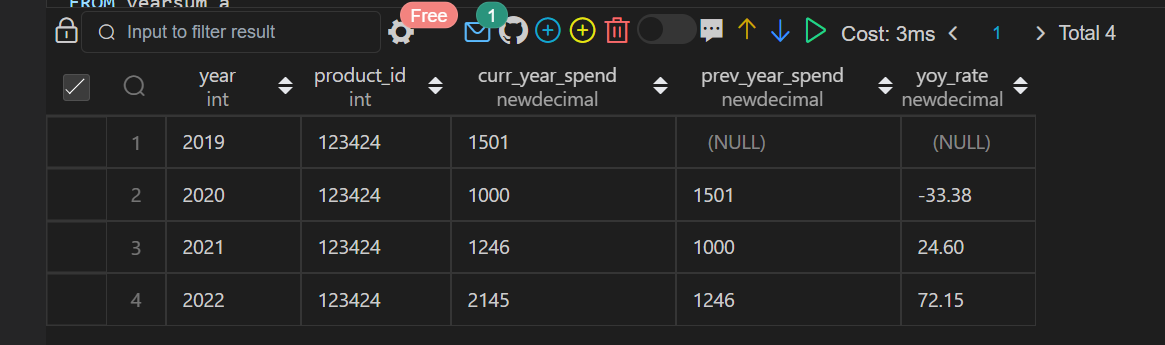
ROUND(100.00\*(a.spend - b.spend)/b.spend,2) as yoy\_rate

FROM yearsum a

LEFT JOIN yearsum b ON a.year-1=b.year AND a.product\_id = b.product\_id

ORDER BY 2,1;

**Output:**

****

**Q122)**

 create table inventory(

item\_id integer,

item\_type varchar(30),

item\_category varchar(30),

square\_footage decimal

 );

insert into inventory VALUES (1374,'prime\_eligible' ,'mini refrigerator',68.00),

(4245,'not\_prime standing','lamp',26.40),

(2452,'prime\_eligible','television',85.00),

(3255,'not\_prime','side table',22.60),

(1672,'prime\_eligible','laptop',8.50);

**Write a SQL query to find the number of prime and non-prime items that can be stored in the 500,000 square feet warehouse. Output the item type and number of items to be stocked.**

**Query:**

SELECT item\_type,

case

when item\_type = 'prime\_eligible'

then Floor(500000/sum(square\_footage))\*count(item\_type)

else floor((500000 -(select(floor(500000/sum(square\_footage)))\*sum(square\_footage) from inventory where item\_type = 'prime\_eligible'))/sum(square\_footage))\*Count(item\_type)

end

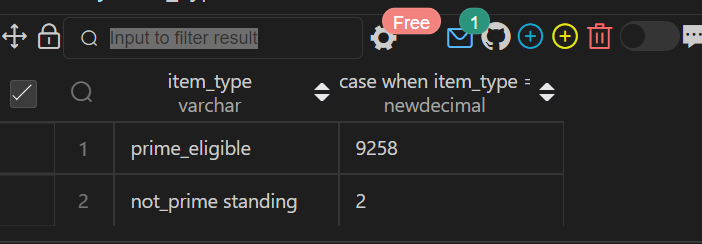
from

inventory

group by item\_type

order by item\_type desc;

**Output:**

****

**Q123)**

Create Table user\_actions (

users\_id int,

event\_id int,

event\_type varchar(50),

event\_date datetime );

nsert into user\_actions Values(445, 7765 ,'sign-in',STR\_TO\_DATE('05/31/2022 12:00:00','%m/%d/%Y %T'));

Insert into user\_actions Values(445, 3634 ,'like',STR\_TO\_DATE('06/05/2022 12:00:00','%m/%d/%Y %T'));

Insert into user\_actions Values(742, 6458 ,'sign-in', STR\_TO\_DATE('07/03/2022 12:00:00','%m/%d/%Y %T'));

Insert into user\_actions Values(742, 1374 ,'comment', STR\_TO\_DATE('07/19/2022 12:00:00','%m/%d/%Y %T'));

**Assume you have the table below containing information on Facebook user actions. Write a query to obtain the active user retention in July 2022. Output the month (in numerical format 1, 2, 3) and the number of monthly active users (MAUs).**

**Query:**

SELECT EXTRACT(MONTH FROM a1.event\_date) as month, COUNT(DISTINCT a1.users\_id) as monthly\_active\_users

from user\_actions as a1,user\_actions as a2

where a1.users\_id = a2.users\_id AND

EXTRACT(MONTH FROM a1.event\_date) = 7

AND EXTRACT(MONTH FROM a2.event\_date) =6

AND EXTRACT(YEAR FROM a1.event\_date) = 2022

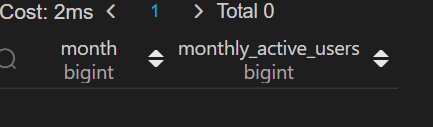
AND EXTRACT(YEAR FROM a2.event\_date) =2022

AND a1.event\_type in ( 'sign-in', 'like', 'comment') AND

a2.event\_type in ('sign-in', 'like', 'comment')

GROUP BY month;

**Output:**

****

**Q124)**

Google's marketing team is making a Superbowl commercial and needs a simple statistic to put on their TV ad: the median number of searches a person made last year. However, at Google scale, querying the 2 trillion searches is too costly. Luckily, you have access to the summary table which tells you the number of searches made last year and how many Google users fall into that bucket.

Query

WITH expanded AS(

  SELECT searches

  FROM search\_frequency

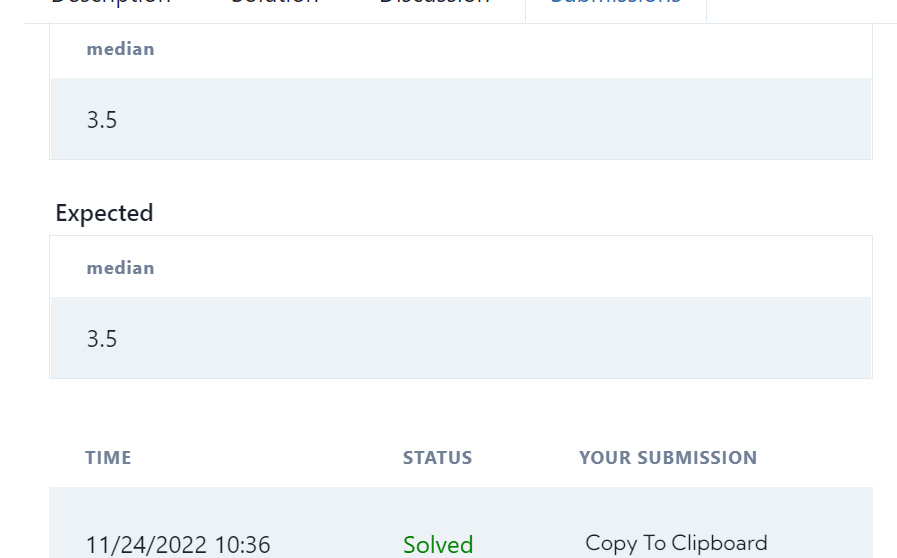
  GROUP BY searches, GENERATE\_SERIES(1,num\_users)

)

SELECT PERCENTILE\_CONT(0.5) WITHIN GROUP (ORDER BY searches) AS median

FROM expanded

**Output**

****

**Q127)**

**Sometimes, payment transactions are repeated by accident; it could be due to user error, API failure or a retry error that causes a credit card to be charged twice. Using the transactions table, identify any payments made at the same merchant with the same credit card for the same amount within 10 minutes of each other. Count such repeated payments. Level - Hard Hint- Use Partition and order by**

Query

select count(t1.merchant\_id) as payment\_count

from transactions t1

join transactions t2

on t1.merchant\_id=t2.merchant\_id

and t1.credit\_card\_id=t2.credit\_card\_id

and t1.amount=t2.amount

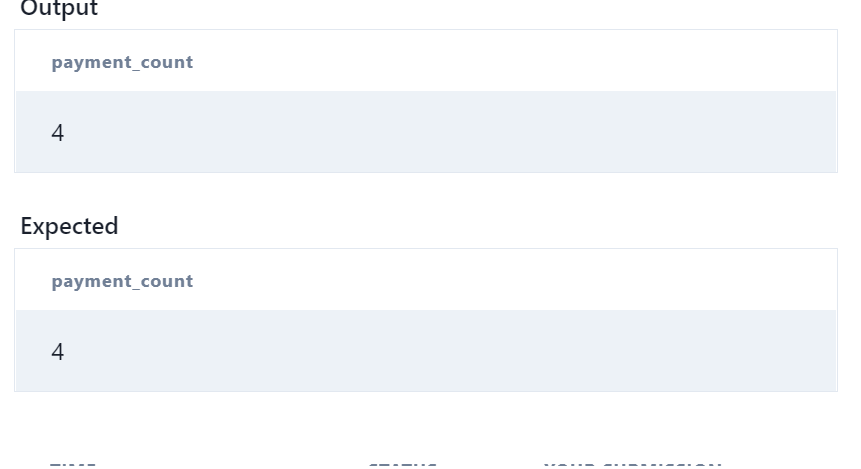
and t1.transaction\_id<t2.transaction\_id

where (date\_part('minute',t2.transaction\_timestamp)-date\_part('minute',t1.transaction\_timestamp))<=10

and  date\_part('hour',t1.transaction\_timestamp)=date\_part('hour',t2.transaction\_timestamp)

;;

**OUTPUT**

****

**Q129)**

CREATE TABLE Scores(

player\_name varchar(30),

gender varchar(30),

day date,

score\_points int,

constraint pk PRIMARY KEY((gender, day)

);

insert into Scores values('Aron','F','2020-01-01', 17),

('Alice','F','2020-01-07',23),

('Bajrang','M','2020-01-07',7),

('Khali','M','2019-12-25',11),

('Slaman','M','2019-12-30', 13),

('Joe','M','2019-12-31', 3),

('Jose','M','2019-12-18',2),

('Priya','F','2019-12-31',23),

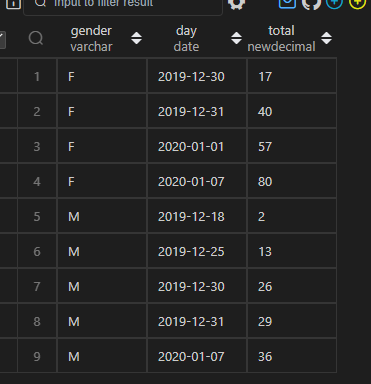
('Priyanka','F','2019-12-30',17);

**Write an SQL query to find the total score for each gender on each day.**

**Query:**

select gender,day, sum(score\_points) over(partition by gender order by gender,day  rows BETWEEN unbounded preceding and current row) as total from  Scores;

**OUTPUT:**

****

**130)**

CREATE TABLE person(

id int PRIMARY KEY,

name varchar(30),

phone\_number varchar(30)

);

 CREATE TABLE country(

name varchar(30),

country\_code varchar(30) PRIMARY KEY

 );

  CREATE TABLE calls(

 caller\_id int,

callee\_id int,

duration int

  );

insert into person values(3 , "Jonathan", "051-1234567"),(12, "Elvis", "051-7654321"),(1 , "Moncef", "212-1234567"),

(2 , "Maroua", "212-6523651"),(7 , "Meir", "972-1234567"),(9 , "Rachel", "972-0011100");

insert into country  values("Peru", '051'),("Israel", '972'),("Morocco", '212'),("Germany", '049'),("Ethiopia", '251');

insert into calls values (1, 9, 33),(2, 9, 4),(1, 2, 59),(3, 12, 102),(3, 12, 330),(12, 3, 5),(7, 9, 13),(7, 1, 3),(9, 7, 1),(1, 7, 7);

**Write an SQL query to find the countries where this company can invest. Return the result table in any order**

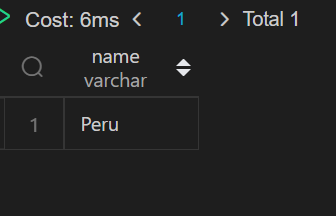
**Query:**

SELECT cc.name from person p inner join  calls c on p.id=c.caller\_id or p.id=c.callee\_id

 inner join country cc on cc.country\_code=left(p.phone\_number,3) group by cc.name having avg(c.duration) >

(select avg(duration) from calls);

**OUTPUT:**

****

**Q131.**

Create table If Not Exists Numbers (

       Number int,

       Frequency int);

insert into Numbers (Number, Frequency) values ('0', '7');

insert into Numbers (Number, Frequency) values ('1', '1');

insert into Numbers (Number, Frequency) values ('2', '3');

insert into Numbers (Number, Frequency) values ('3', '1');

**Write an SQL query to report the median of all the numbers in the database after decompressing the Numbers table. Round the median to one decimal point**.

**Query:**

select

    avg(number) median

from

   Numbers n

where

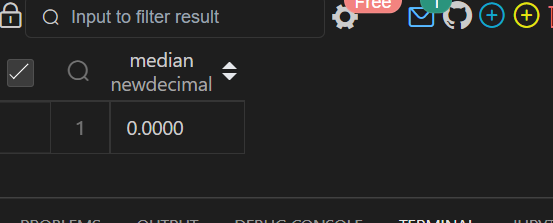
    n.frequency >= abs(

                        (select sum(Frequency) from Numbers where Number<=n.number)

                        -

                        (select sum(Frequency) from Numbers where Number>=n.number));

**OUTPUT:**

****

**Q132.**

Create table If Not Exists salary (

    id int,

    employee\_id int,

    amount int,

    pay\_date date);

Create table If Not Exists employee (

    employee\_id int,

    department\_id int);

insert into salary

    (id, employee\_id, amount, pay\_date)

values

    ('1', '1', '9000', '2017/03/31');

insert into salary

    (id, employee\_id, amount, pay\_date)

values

    ('2', '2', '6000', '2017/03/31');

insert into salary

    (id, employee\_id, amount, pay\_date)

values

    ('3', '3', '10000', '2017/03/31');

insert into salary

    (id, employee\_id, amount, pay\_date)

values

    ('4', '1', '7000', '2017/02/28');

insert into salary

    (id, employee\_id, amount, pay\_date)

values

    ('5', '2', '6000', '2017/02/28');

insert into salary

    (id, employee\_id, amount, pay\_date)

values

    ('6', '3', '8000', '2017/02/28');

insert into employee

    (employee\_id, department\_id)

values

    ('1', '1');

insert into employee

    (employee\_id, department\_id)

values

    ('2', '2');

insert into employee

    (employee\_id, department\_id)

values

    ('3', '2');

**Write an SQL query to report the comparison result (higher/lower/same) of the average salary of employees in a department to the company's average salary.**

**Query:**

select

    pay\_month,

    department\_id,

    case when dept\_avg > comp\_avg then 'higher' when dept\_avg < comp\_avg then 'lower' else 'same' end comparison

from (

        select  date\_format(b.pay\_date, '%Y-%m') pay\_month, a.department\_id, avg(b.amount) dept\_avg,  d.comp\_avg

        from employee a

        inner join salary b

            on (a.employee\_id = b.employee\_id)

        inner join (select date\_format(c.pay\_date, '%Y-%m') pay\_month, avg(c.amount) comp\_avg

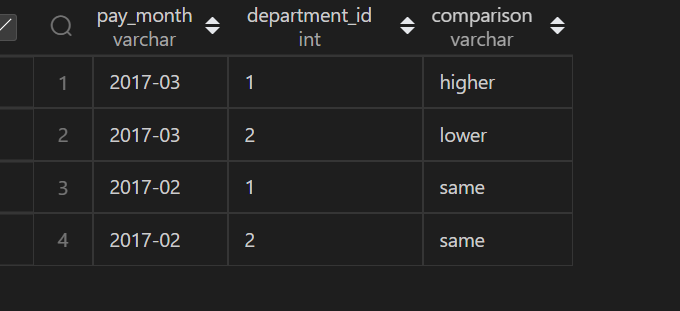
                    from salary c

                    group by date\_format(c.pay\_date, '%Y-%m')) d

            on ( date\_format(b.pay\_date, '%Y-%m') = d.pay\_month)

group by date\_format(b.pay\_date, '%Y-%m'), department\_id, d.comp\_avg) final

**OUTPUT:**



**Q133.**

CREATE TABLE Players(

player\_id int PRIMARY KEY,

group\_id varchar(30)

);

 CREATE TABLE Matches(

match\_id int primary KEY,

first\_player int,

second\_player int,

first\_score int,

second\_score int

 );

 insert into Players VALUES(15,1),

(25,1),

(30,1),

(45,1),

(10,2),

(35,2),

(50,2),

(20,3),

(40,3);

insert into Matches VALUES(1,15,45,3,0),

(2,30,25,1,2),(3,30,25,1,2),

(4,40,20,5,2),(5,35,50,1,1);

**The winner in each group is the player who scored the maximum total points within the group. In the case of a tie, the lowest player\_id wins. Write an SQL query to find the winner in each group.**

**Query:**

select group\_id,player\_id

from (

    select sc.group\_id group\_id, sc.player\_id player\_id,

       rank() over (partition by sc.group\_id order by sum(sc.score) desc, sc.player\_id asc) as rnk

    from(

        select p.group\_id group\_id,

         p.player\_id player\_id ,

         sum(m.first\_score) as score

        from Players p

        inner join Matches m

        on p.player\_id = m.first\_player

        group by p.group\_id,p.player\_id

        union all

        select p.group\_id group\_id,

         p.player\_id player\_id ,

        sum(second\_score) as score

        from Players p

        inner join Matches m

        on p.player\_id = m.second\_player

        group by p.group\_id,p.player\_id

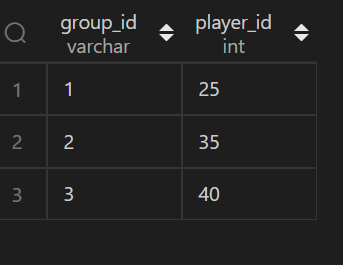
    ) sc

    group by sc.group\_id,sc.player\_id

) A

where rnk = 1;

**OUTPUT:**

****

**Q136.**

CREATE TABLE Student

(student\_id INT,

 student\_name VARCHAR(32));

INSERT INTO Student

VALUES

(1, 'Daniel'),

(2, 'Jade'),

(3, 'Stella'),

(4, 'Jonathan'),

(5, 'Will');

CREATE TABLE Exam

(exam\_id INT,

 student\_id INT,

 score INT);

 INSERT INTO Exam

 VALUES

 (10, 1, 70),

 (10, 2, 80),

 (10, 3, 90),

 (20, 1, 80),

 (30, 1, 70),

 (30, 3, 80),

 (30, 4, 90),

 (40, 1, 60),

 (40, 2, 70),

 (40, 4, 80);

**A quiet student is the one who took at least one exam and did not score the high or the low score. Write an SQL query to report the students (student\_id, student\_name) being quiet in all exams. Do not return the student who has never taken any exam**

**Query:**

WITH TMP AS

 (SELECT DISTINCT(student\_id) AS student\_id

 FROM (SELECT student\_id,

     RANK() OVER(PARTITION BY exam\_id

           ORDER BY Score) AS r1,

     RANK() OVER(PARTITION BY exam\_id

           ORDER BY Score DESC) AS r2

 FROM Exam) AS T

 WHERE r1 = 1 OR r2 = 1),

 TMP1 AS

 (SELECT DISTINCT(student\_id) AS student\_id

  FROM Exam

  WHERE student\_id NOT IN (SELECT student\_id FROM TMP))

SELECT A.student\_id, B.student\_name

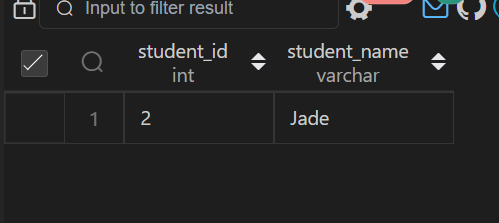
FROM TMP1 AS A

LEFT OUTER JOIN Student AS B

ON A.student\_id = B.student\_id

ORDER BY student\_id;

**OUPUT:**

****

**Q137.**

create table UserActivity(

username varchar(30),

activity varchar(30),

startDate Date,

endDate Date

);

insert into UserActivity VALUES('Alice','Travel', '2020-02-12','2020-02-20'),

('Alice','Dancing','2020-02-21','2020-02-23'),

('Alice','Travel','2020-02-24','2020-02-28'),

('Bob','Travel','2020-02-11','2020-02-18');

**Write an SQL query to show the second most recent activity of each user. If the user only has one activity, return that one. A user cannot perform more than one activity at the same time.**

**Query:**

select username,activity,startDate,endDate from  (

select \* , rank()over(partition by username order by startDate desc) as rnk,

count(username) over( partition by username order by startDate desc) as cnt

from UserActivity)tmp  where rnk=2 or cnt=1;

**Q139.**

CREATE TABLE STUDENTS (

    ID INTEGER,

    NAME VARCHAR(30),

    MARKS VARCHAR(30)

);

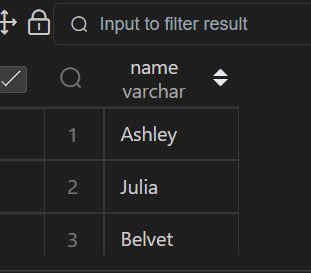
insert into STUDENTS  VALUES(1,'Ashley',81),(2,'Samantha',75),(4,'Julia',76),(3,'Belvet',84);

**Query the Name of any student in STUDENTS who scored higher than 75 Marks.**

**Query:**

SELECT name from STUDENTS  where MARKS >75 order by right(name,3);

**OUTPUT:**

****

**Q145**

CREATE TABLE OCCUPATIONS(

    Name  VARCHAR(30),

    Occupation VARCHAR(30)

);

insert into OCCUPATIONS values('julia','Actor'),('Samantha','Doctor'),('Maria','Actor'),('Meera','Singer'),('Ashely','professor'),

('Ketty','Professor'),('Christeen','Professor'),('Jane','Actor'),

('Jenny','Doctor'),('Priya','Singer');

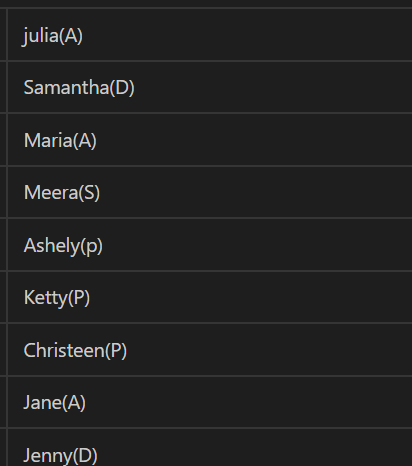
**Query the number of occurrences of each occupation in OCCUPATIONS. Sort the occurrences in ascending order, and output**

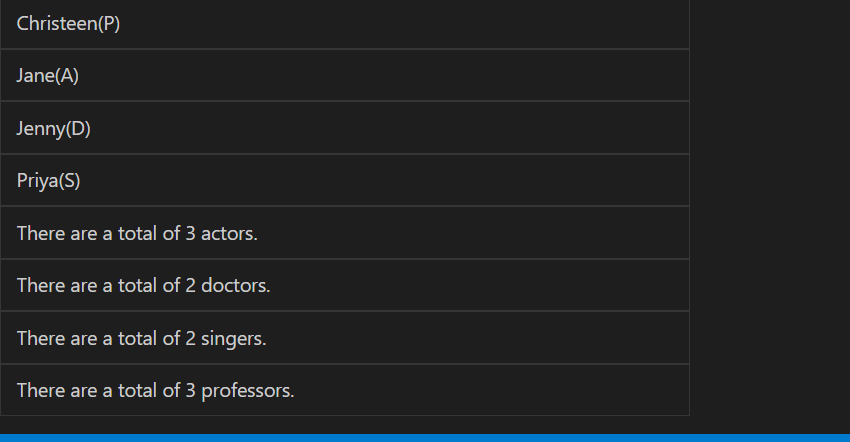
**Query:**

(SELECT CONCAT(Name,'(',SUBSTRING(OCCUPATION,1,1),')') from OCCUPATIONS order by Name asc )  UNION

(SELECT CONCAT('There are a total of ',count(OCCUPATION),' ',lower(OCCUPATION),'s','.') from OCCUPATIONS group by OCCUPATION ORDER BY COUNT(occupation),occupation asc);

**OUTPUT:**

****

****

**Q146.**

CREATE TABLE OCCUPATIONS(

    Name  VARCHAR(30),

    Occupation VARCHAR(30)

);

insert into OCCUPATIONS values('julia','Actor'),('Samantha','Doctor'),('Maria','Actor'),('Meera','Singer'),('Ashely','professor'),

('Ketty','Professor'),('Christeen','Professor'),('Jane','Actor'),

('Jenny','Doctor'),('Priya','Singer');

**Query the number of occurrences of each occupation in OCCUPATIONS. Sort the occurrences in ascending order, and output**

**Query**

set @d=0,@p=0,@s=0,@a=0;

 select max(dname),max(pname),max(sname),max(aname) from(select

case when Occupation='Doctor' then Name end as dname,

case when Occupation='Professor' then Name end as pname,

case when Occupation='Singer' then Name end as sname,

case when Occupation='Actor' then Name end as aname,

case

    when Occupation='Doctor' then (@d:=@d+1)

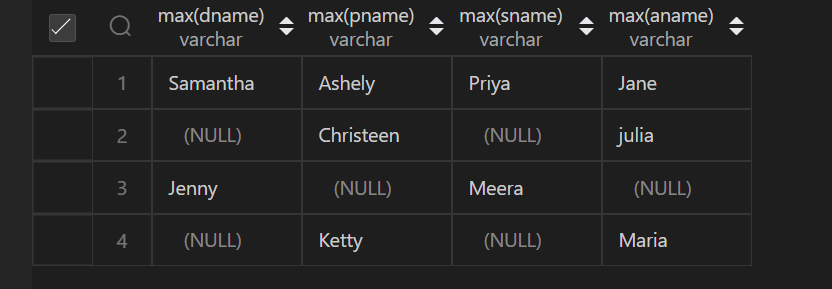
    when Occupation='Professor' then (@p:=@p+1)

    when Occupation='Singer' then (@s:=@s+1)

    when Occupation='Actor' then(@a:=@a+1)

end as count from  OCCUPATIONS order by Name ) as t group by count;

**OUTPUT**

****

**Q147.**

CREATE TABLE BST(

    N INT,

    P INT

);

INSERT INTO BST VALUES(1,2),(3,2),

(6,8),(9,8),(2,5),(8,5),(5,NULL);

**Write a query to find the node type of Binary Tree ordered by the value of the node.**

**Query**

select N,

CASE

 WHEN P is NULL then 'Root'

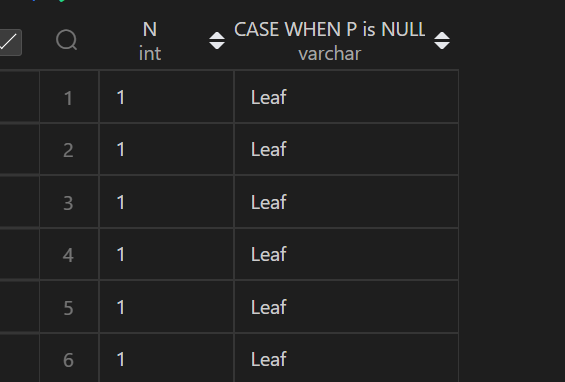
 WHEN N in (select P from BST) then 'Inner'

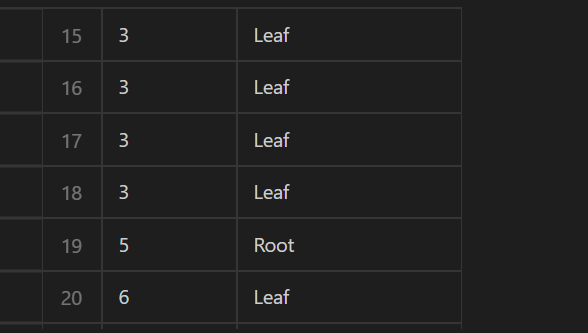
 else 'Leaf'

 end

 from BST order by N;

**OUTPUT**

****

****

**Q149.**

CREATE TABLE FUN(

   X INT,

   Y INT

   );

   INSERT INTO FUN VALUES(20,20),(20,20),

   (20,21),(23,22),(22,23),(21,20);

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

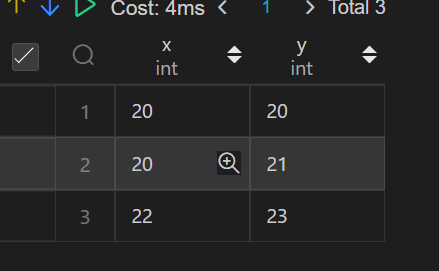
**Query**

 select  distinct A.x,A.y from

(select x,y, row\_number() over(order by x asc) as r from FUN) A join

(select x,y, row\_number() over(order by x asc) as r from FUN) B on A.x=B.y and B.x =A.y  and A.r!=B.r and A.x<=A.y  order by A.x;

**OUTPUT**

****