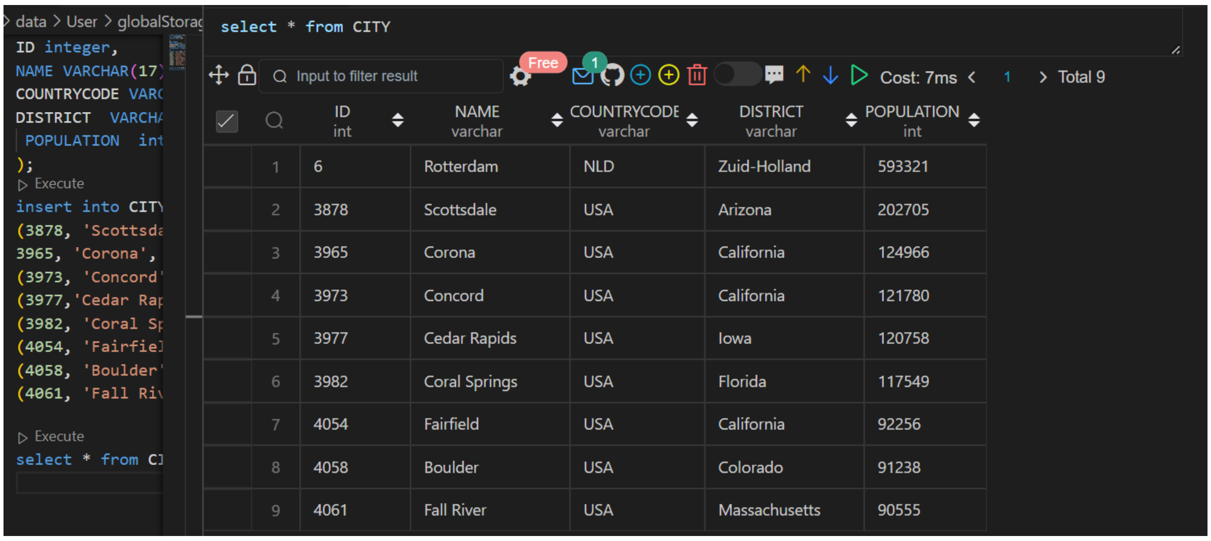
NAME: ABHIJIT BARIK
abro45172018@gmail.com

Table creation and value insertion:

```sql
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:



```
data > User > globalStorag
ID integer,
NAME VARCHAR(17
COUNTRYCODE VARC
DISTRICT  VARCH
 POPULATION  int
);
▷ Execute
insert into CITY
(3878, 'Scottsda
3965, 'Corona',
(3973, 'Concord'
(3977,'Cedar Rap
(3982, 'Coral Sp
(4054, 'Fairfiel
(4058, 'Boulder'
(4061, 'Fall Riv

▷ Execute
select * from CI
```

```sql
select * from CITY
```

Input to filter result — Cost: 7ms — 1 — Total 9

	ID int	NAME varchar	COUNTRYCODE varchar	DISTRICT varchar	POPULATION int
1	6	Rotterdam	NLD	Zuid-Holland	593321
2	3878	Scottsdale	USA	Arizona	202705
3	3965	Corona	USA	California	124966
4	3973	Concord	USA	California	121780
5	3977	Cedar Rapids	USA	Iowa	120758
6	3982	Coral Springs	USA	Florida	117549
7	4054	Fairfield	USA	California	92256
8	4058	Boulder	USA	Colorado	91238
9	4061	Fall River	USA	Massachusetts	90555

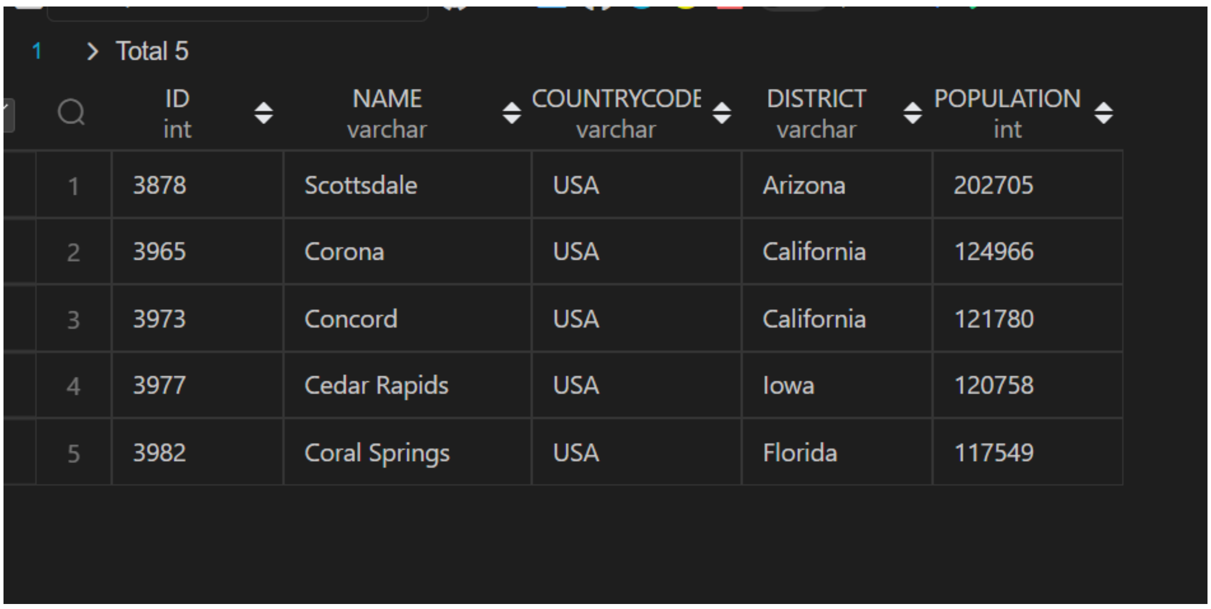
Q1.

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:

```sql
SELECT *
FROM CITY
WHERE POPULATION > 100000 AND COUNTRYCODE = "USA";
```

OUTPUT:



Total 5

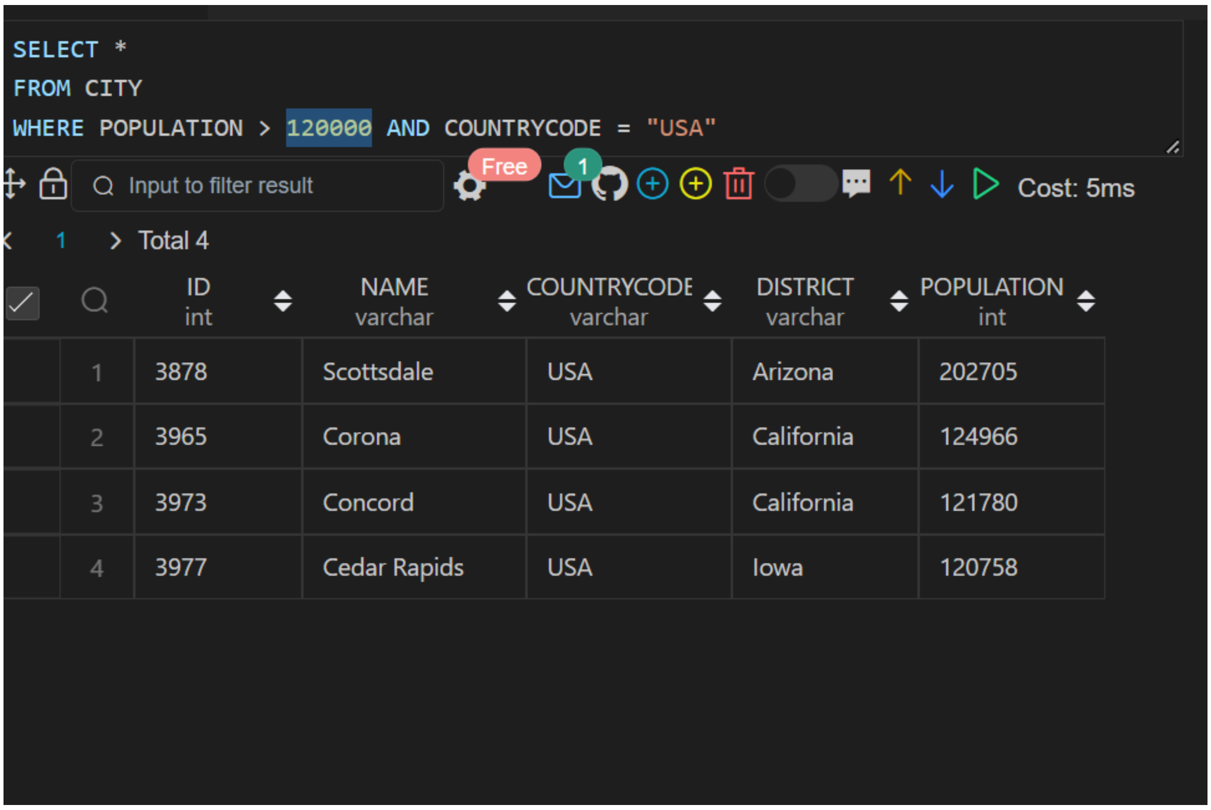
	ID int	NAME varchar	COUNTRYCODE varchar	DISTRICT varchar	POPULATION int
1	3878	Scottsdale	USA	Arizona	202705
2	3965	Corona	USA	California	124966
3	3973	Concord	USA	California	121780
4	3977	Cedar Rapids	USA	Iowa	120758
5	3982	Coral Springs	USA	Florida	117549

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:

```sql
SELECT *
FROM CITY
WHERE POPULATION > 120000 AND COUNTRYCODE = "USA";
```

OUTPUT:



```
SELECT *
FROM CITY
WHERE POPULATION > 120000 AND COUNTRYCODE = "USA"
```

Total 4

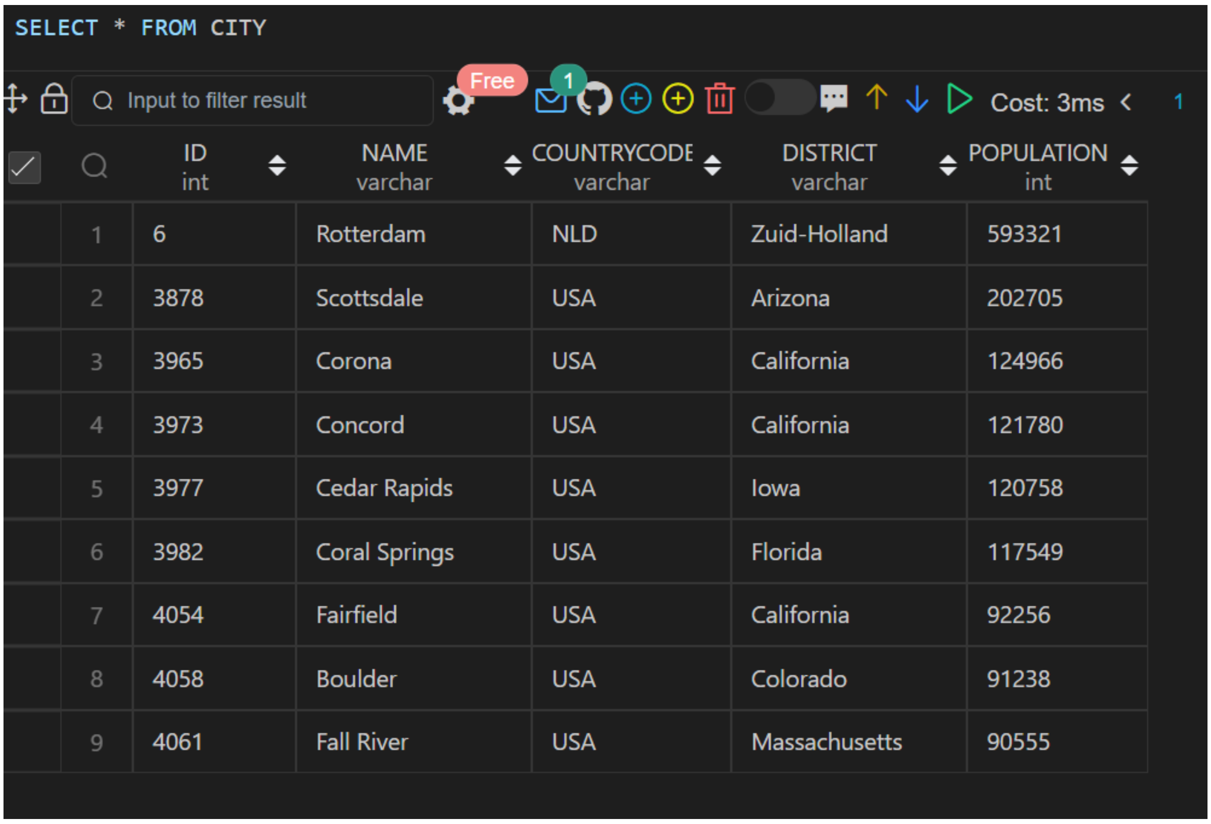
	ID int	NAME varchar	COUNTRYCODE varchar	DISTRICT varchar	POPULATION int
1	3878	Scottsdale	USA	Arizona	202705
2	3965	Corona	USA	California	124966
3	3973	Concord	USA	California	121780
4	3977	Cedar Rapids	USA	Iowa	120758

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

SQL CODE:

```
SELECT * FROM City;
```

OUTPUT:


```
SELECT * FROM CITY
```

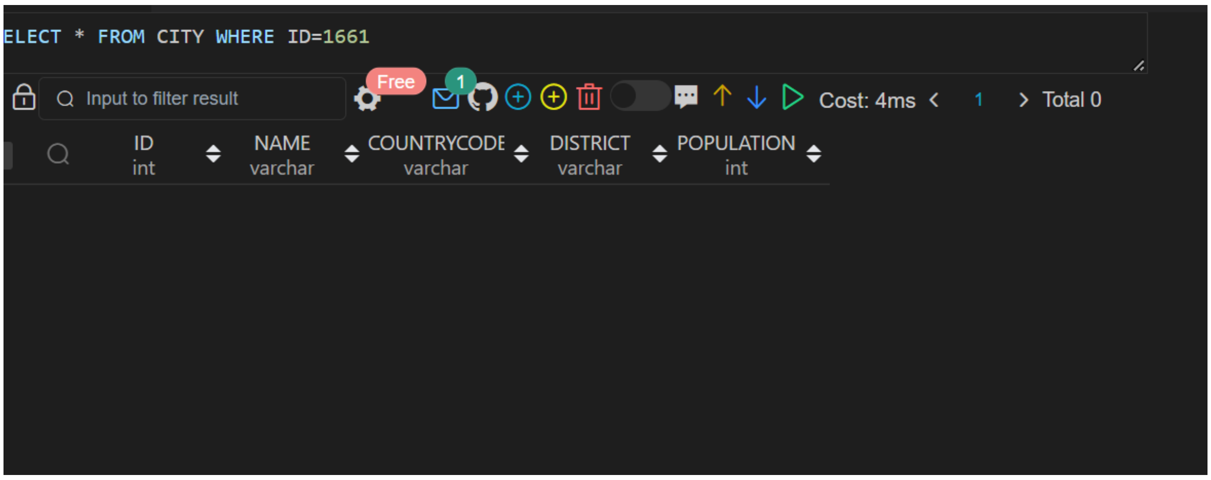
ID int	NAME varchar	COUNTRYCODE varchar	DISTRICT varchar	POPULATION int
6	Rotterdam	NLD	Zuid-Holland	593321
3878	Scottsdale	USA	Arizona	202705
3965	Corona	USA	California	124966
3973	Concord	USA	California	121780
3977	Cedar Rapids	USA	Iowa	120758
3982	Coral Springs	USA	Florida	117549
4054	Fairfield	USA	California	92256
4058	Boulder	USA	Colorado	91238
4061	Fall River	USA	Massachusetts	90555

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

SQL CODE:

```
SELECT * FROM CITY WHERE ID=1661;
```

OUTPUT:


```
ELECT * FROM CITY WHERE ID=1661
```

ID int	NAME varchar	COUNTRYCODE varchar	DISTRICT varchar	POPULATION int

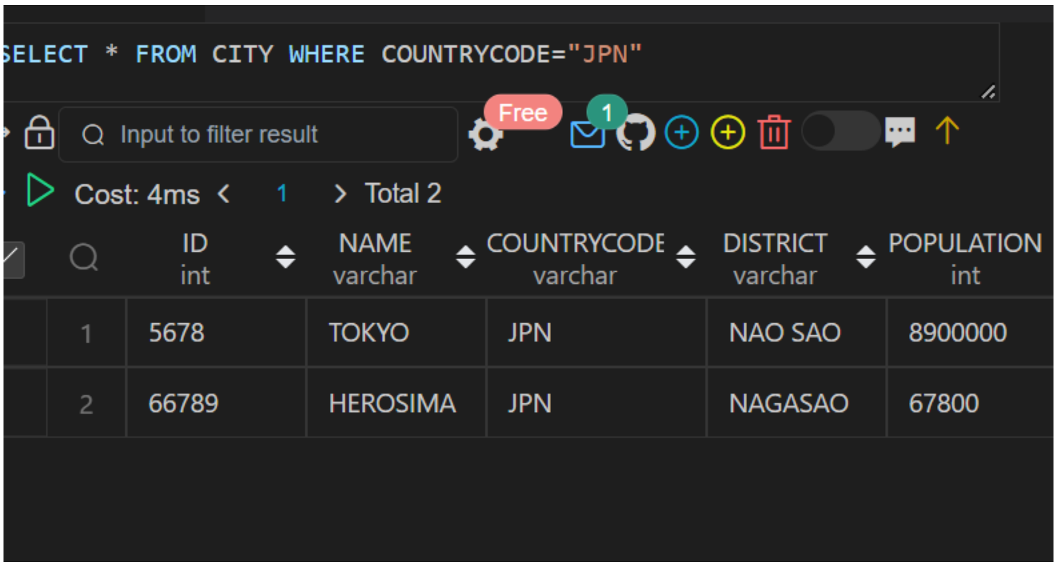
Q5. Query all attributes of every Japanese city in the CITY table. The COUNTRYCODE for Japan is JPN.

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

SQL CODE:

```sql
SELECT * FROM CITY WHERE COUNTRYCODE="JPN";
```

OUTPUT:



```
SELECT * FROM CITY WHERE COUNTRYCODE="JPN"
```

Input to filter result

Free 1

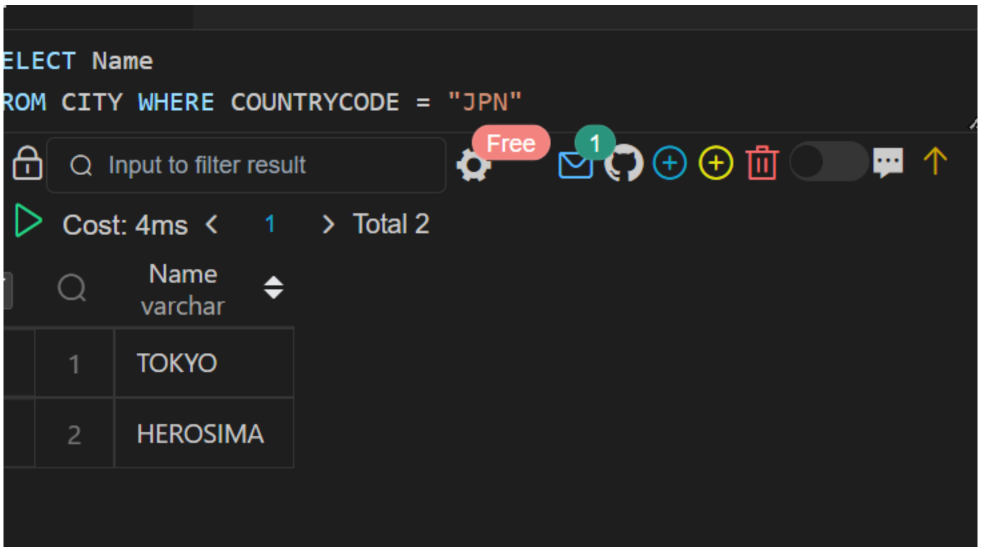
Cost: 4ms < 1 > Total 2

	ID int	NAME varchar	COUNTRYCODE varchar	DISTRICT varchar	POPULATION int
1	5678	TOKYO	JPN	NAO SAO	8900000
2	66789	HEROSIMA	JPN	NAGASAO	67800

SQL CODE:

```sql
SELECT Name
FROM CITY WHERE COUNTRYCODE = "JPN";
```

OUTPUT:



```
ELECT Name
ROM CITY WHERE COUNTRYCODE = "JPN"
```

Input to filter result

Free 1

Cost: 4ms < 1 > Total 2

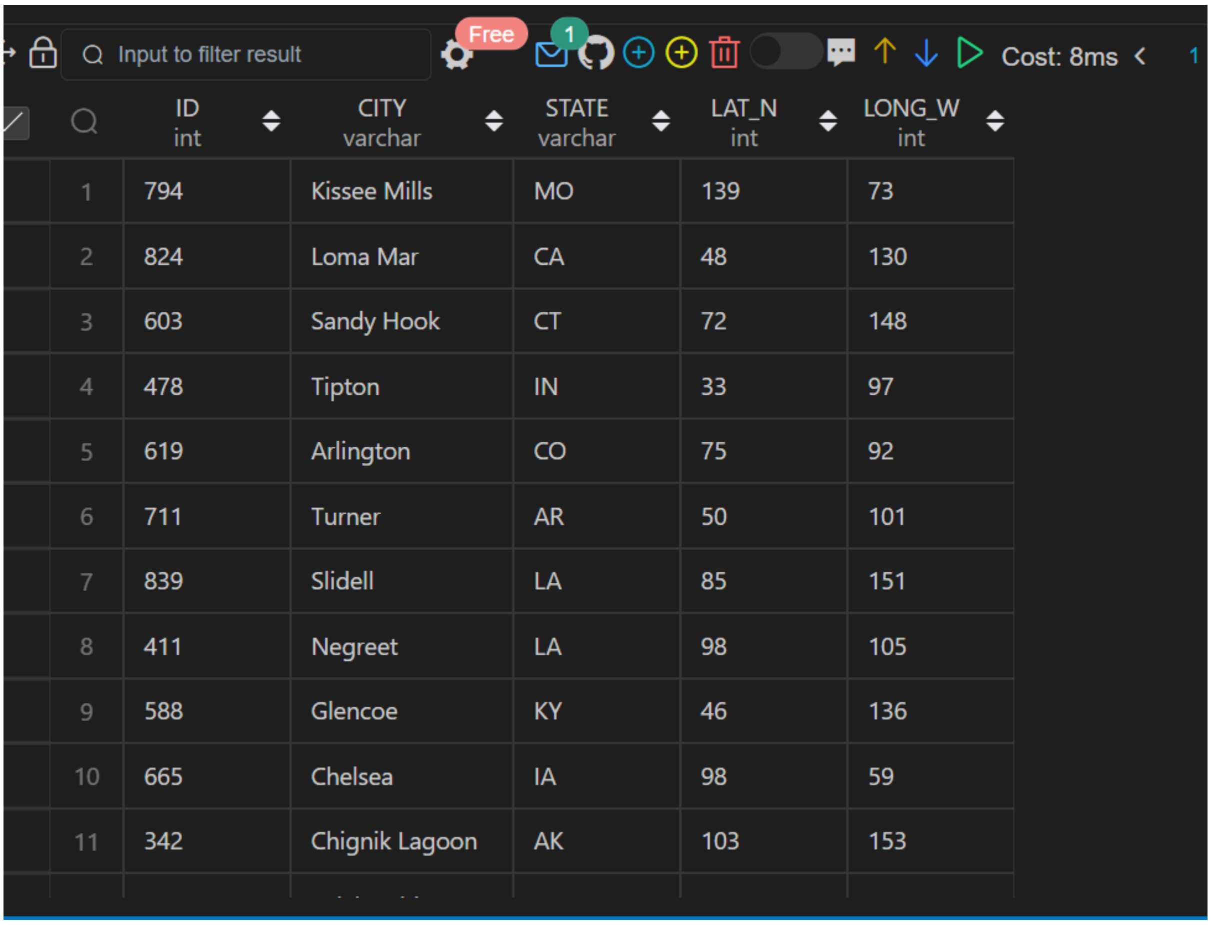
	Name varchar
1	TOKYO
2	HEROSIMA

Sample dataset-2

Table creation and value insertion:

```sql
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:



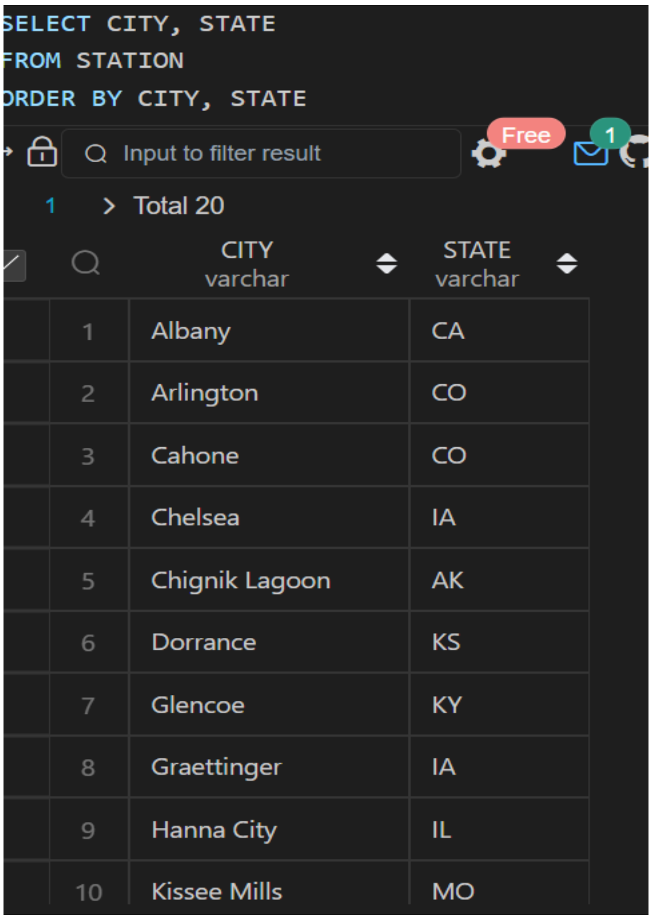
	ID int	CITY varchar	STATE varchar	LAT_N int	LONG_W int
1	794	Kissee Mills	MO	139	73
2	824	Loma Mar	CA	48	130
3	603	Sandy Hook	CT	72	148
4	478	Tipton	IN	33	97
5	619	Arlington	CO	75	92
6	711	Turner	AR	50	101
7	839	Slidell	LA	85	151
8	411	Negreet	LA	98	105
9	588	Glencoe	KY	46	136
10	665	Chelsea	IA	98	59
11	342	Chignik Lagoon	AK	103	153

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

SQL QUERY:

```
SELECT CITY, STATE
FROM STATION
ORDER BY CITY, STATE;
```

OUTPUT:



```
SELECT CITY, STATE
FROM STATION
ORDER BY CITY, STATE
```

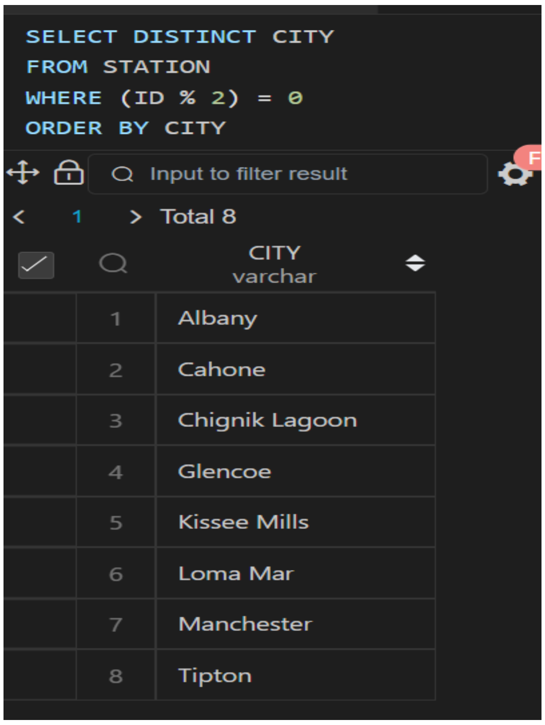
	CITY varchar	STATE varchar
1	Albany	CA
2	Arlington	CO
3	Cahone	CO
4	Chelsea	IA
5	Chignik Lagoon	AK
6	Dorrance	KS
7	Glencoe	KY
8	Graettinger	IA
9	Hanna City	IL
10	Kissee Mills	MO

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:



```
SELECT DISTINCT CITY
FROM STATION
WHERE (ID % 2) = 0
ORDER BY CITY
```

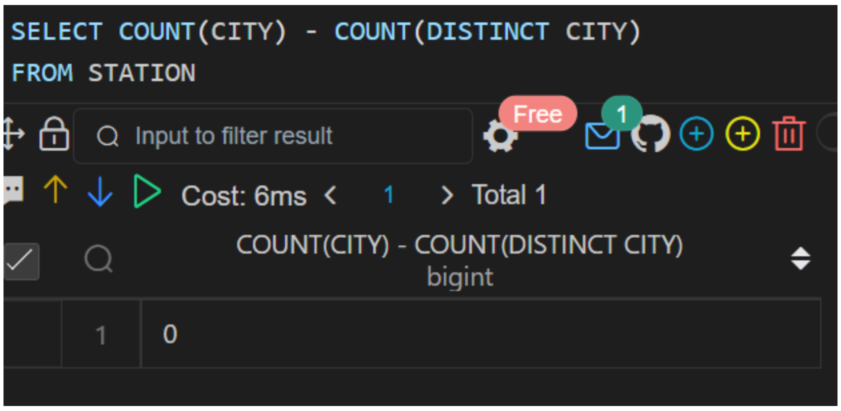
```
Total 8
   CITY
   varchar
1  Albany
2  Cahone
3  Chignik Lagoon
4  Glencoe
5  Kissee Mills
6  Loma Mar
7  Manchester
8  Tipton
```

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:



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

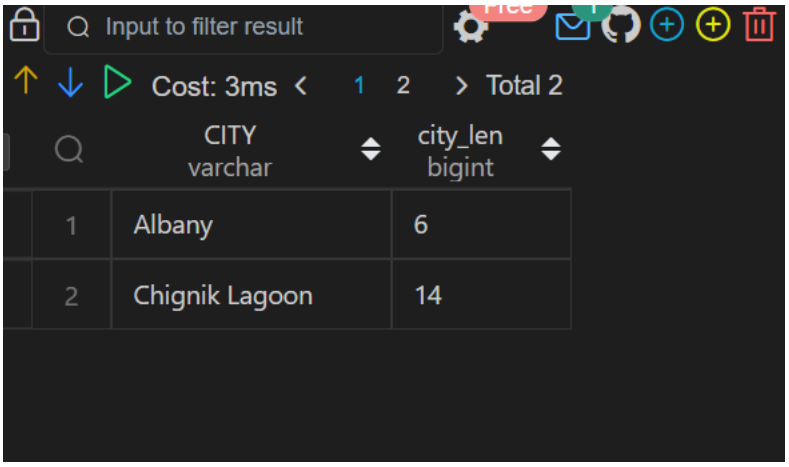
```
Cost: 6ms   Total 1
   COUNT(CITY) - COUNT(DISTINCT CITY)
   bigint
1  0
```

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:


Input to filter result

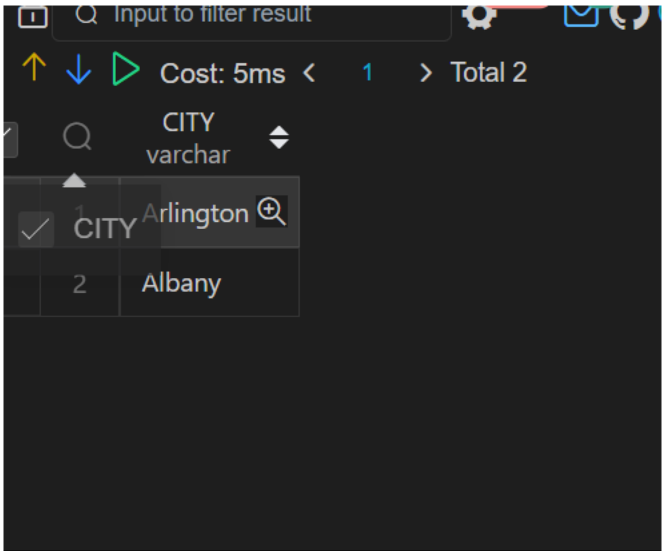
Cost: 3ms < 1 2 > Total 2

	CITY varchar	city_len bigint
1	Albany	6
2	Chignik Lagoon	14

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:


Input to filter result

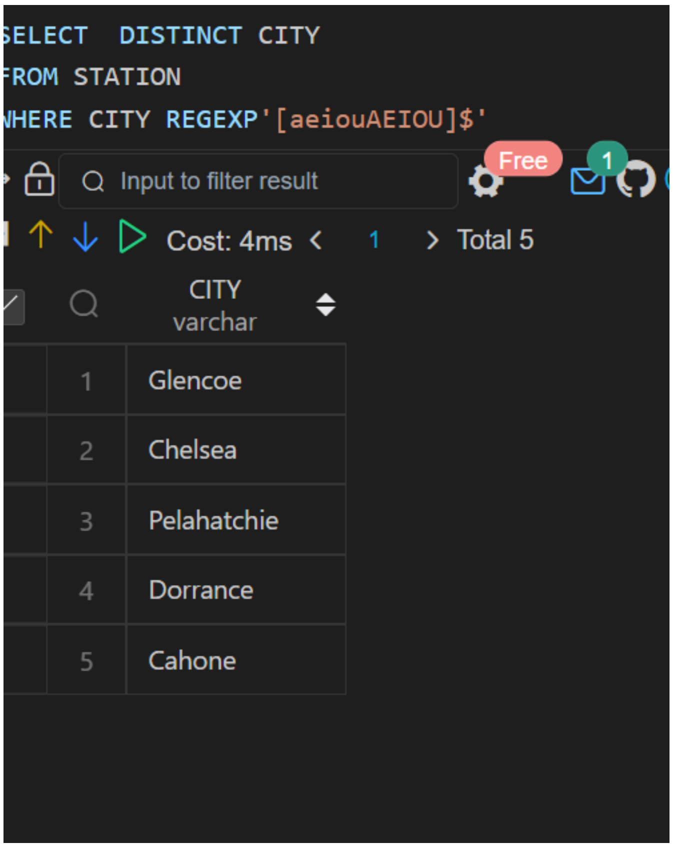
Cost: 5ms < 1 > Total 2

	CITY varchar
1	Arlington
2	Albany

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:



```
SELECT  DISTINCT CITY
FROM STATION
WHERE CITY REGEXP'[aeiouAEIOU]$'
```

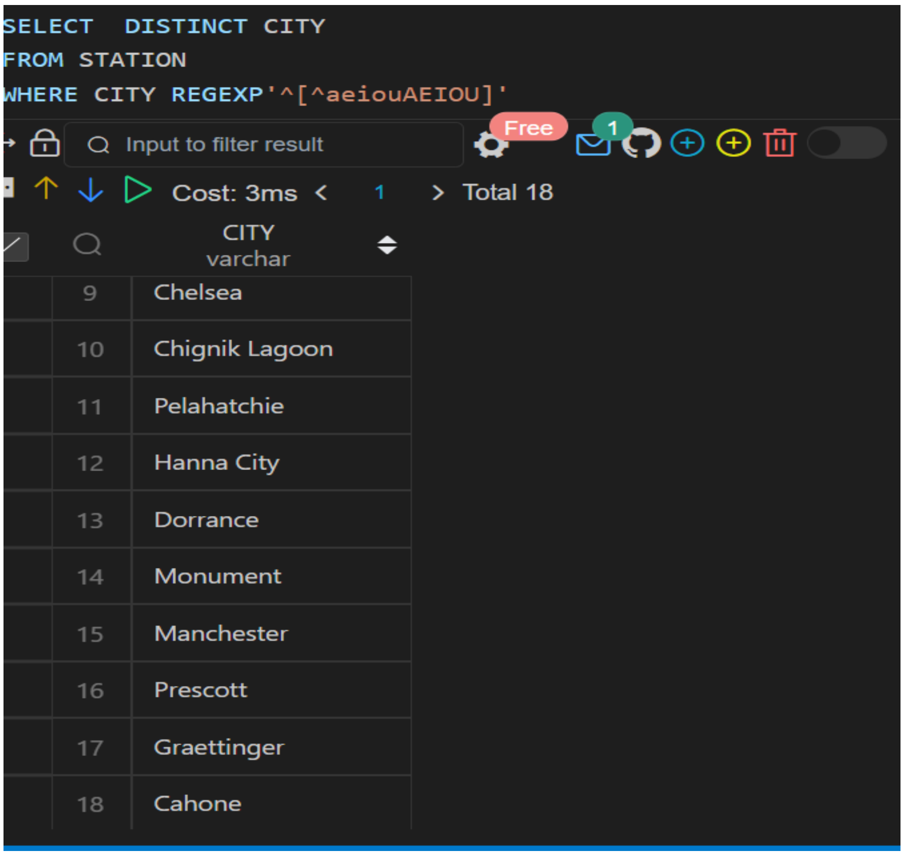
	CITY varchar
1	Glencoe
2	Chelsea
3	Pelahatchie
4	Dorrance
5	Cahone

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:



```
SELECT  DISTINCT CITY
FROM STATION
WHERE CITY REGEXP'^[^aeiouAEIOU]'
```

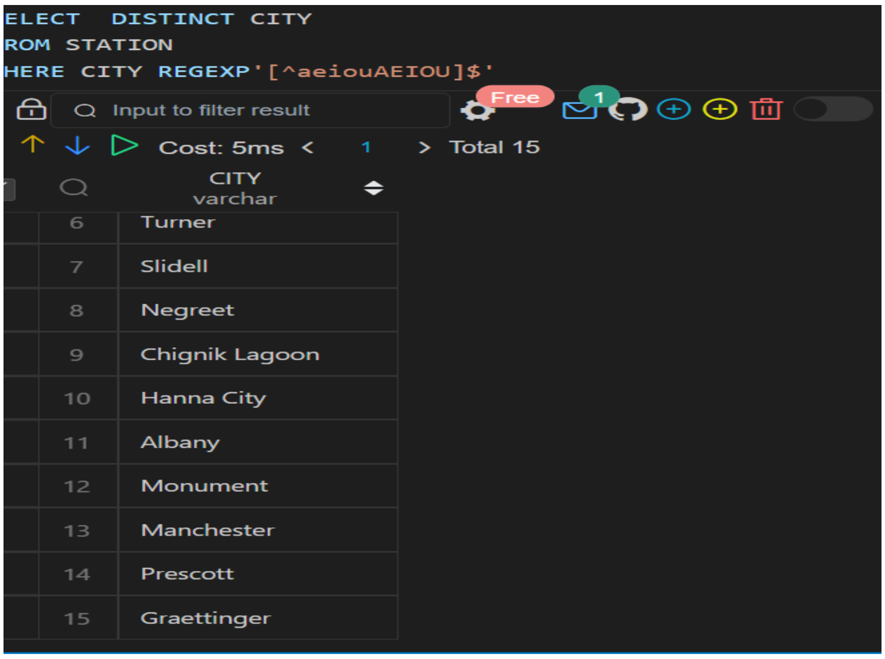
```
Cost: 3ms < 1 > Total 18
   CITY
   varchar
9  Chelsea
10 Chignik Lagoon
11 Pelahatchie
12 Hanna City
13 Dorrance
14 Monument
15 Manchester
16 Prescott
17 Graettinger
18 Cahone
```

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:



```
ELECT  DISTINCT CITY
ROM STATION
HERE CITY REGEXP'[^aeiouAEIOU]$'
```

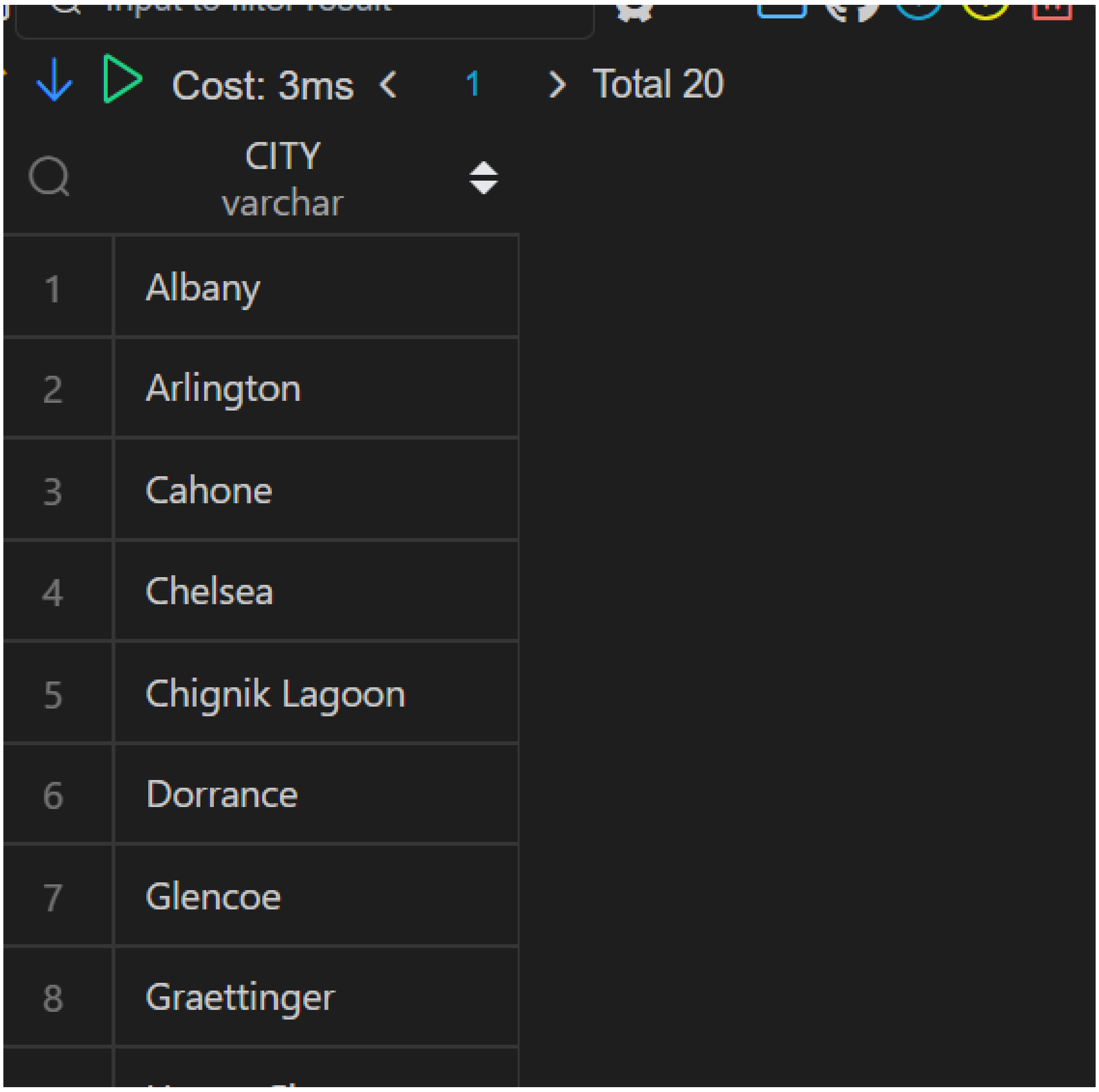
```
Cost: 5ms < 1 > Total 15
   CITY
   varchar
6  Turner
7  Slidell
8  Negreet
9  Chignik Lagoon
10 Hanna City
11 Albany
12 Monument
13 Manchester
14 Prescott
15 Graettinger
```

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:



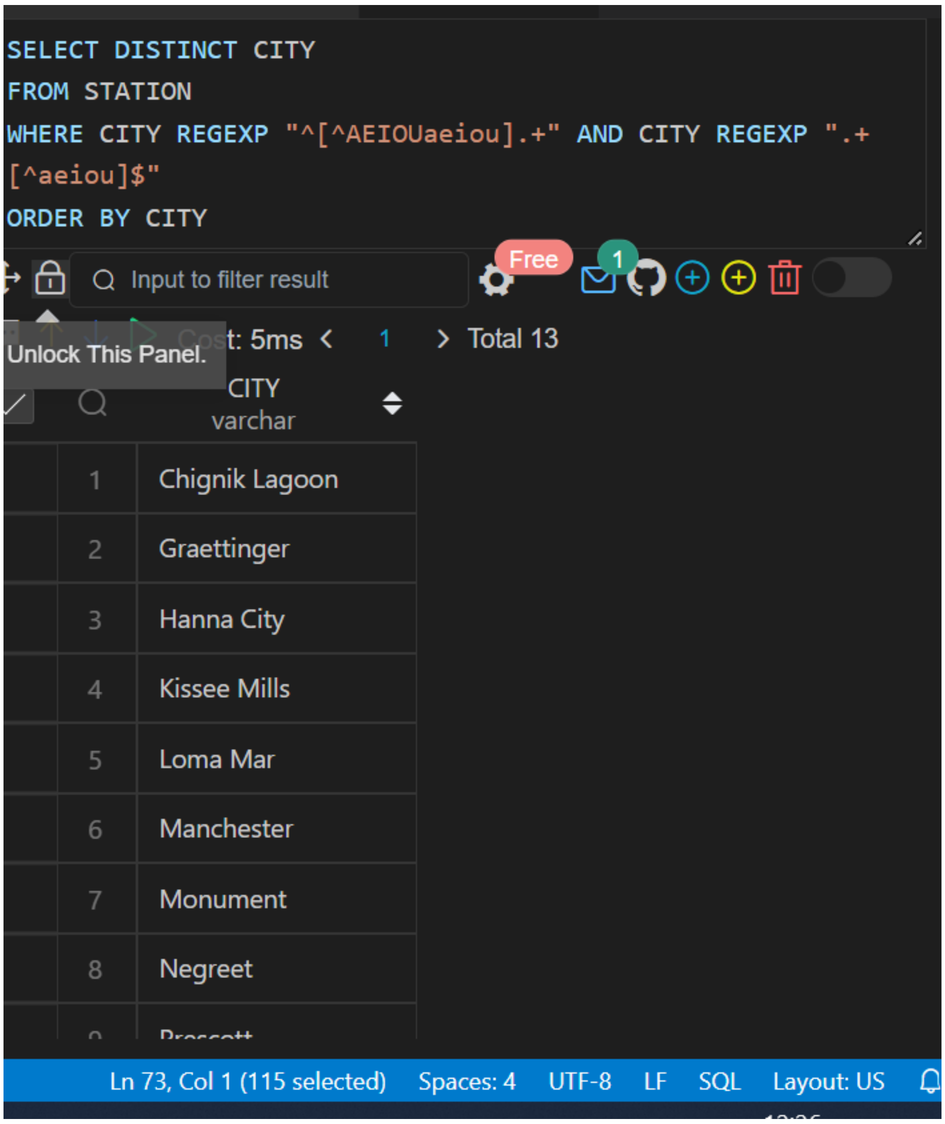
```
Cost: 3ms  <  1  >  Total 20
     CITY
     varchar
1    Albany
2    Arlington
3    Cahone
4    Chelsea
5    Chignik Lagoon
6    Dorrance
7    Glencoe
8    Graettinger
```

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:



```
SELECT DISTINCT CITY
FROM STATION
WHERE CITY REGEXP "^[^AEIOUaeiou].+" AND CITY REGEXP ".+
[^aeiou]$"
ORDER BY CITY
```

```
Cost: 5ms  <  1  >  Total 13

   CITY
   varchar
1  Chignik Lagoon
2  Graettinger
3  Hanna City
4  Kissee Mills
5  Loma Mar
6  Manchester
7  Monument
8  Negreet
```

Q17.

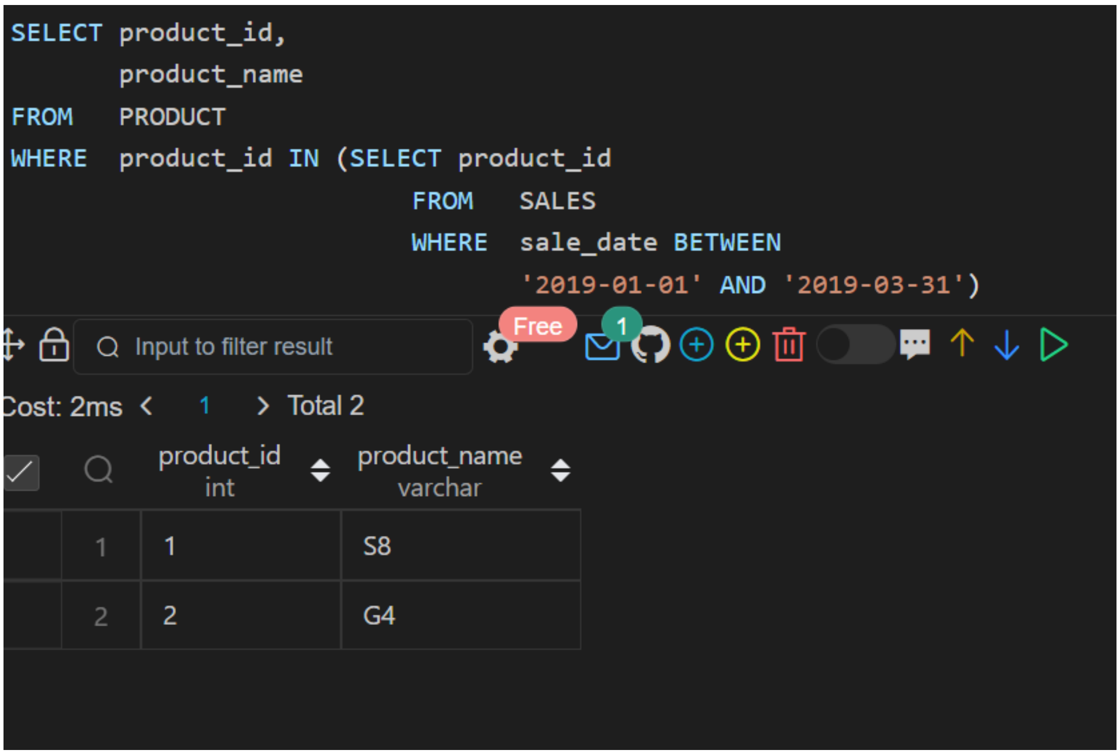
```
CREATE TABLE PRODUCT(
product_id int PRIMARY KEY,
product_name varchar(32),
unit_price int
);


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:



```
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')
```

Input to filter result

Cost: 2ms < 1 > Total 2

	product_id int	product_name varchar
1	1	S8
2	2	G4

Q18.

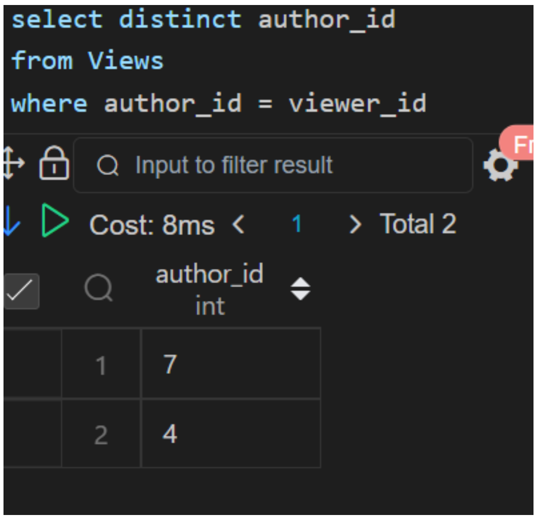
```
create TABLE Views
(article_id int,
author_id int,
viewer_id int,
view_date date
);
INSERT into Views VALUES(1,3,5,'2019-08-01'),
(1,3,6,'2019-08-02'),
(2,7,7,'2019-08-01'),
(2,7,6,'2019-08-02'),
(4,7,1,'2019-07-22'),
(3,4,4,'2019-07-21'),
(3,4,4,'2019-07-21');
```

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:



```
select distinct author_id
from Views
where author_id = viewer_id
```

```
Cost: 8ms < 1 > Total 2

   author_id
   int
1  7
2  4
```

Q19.

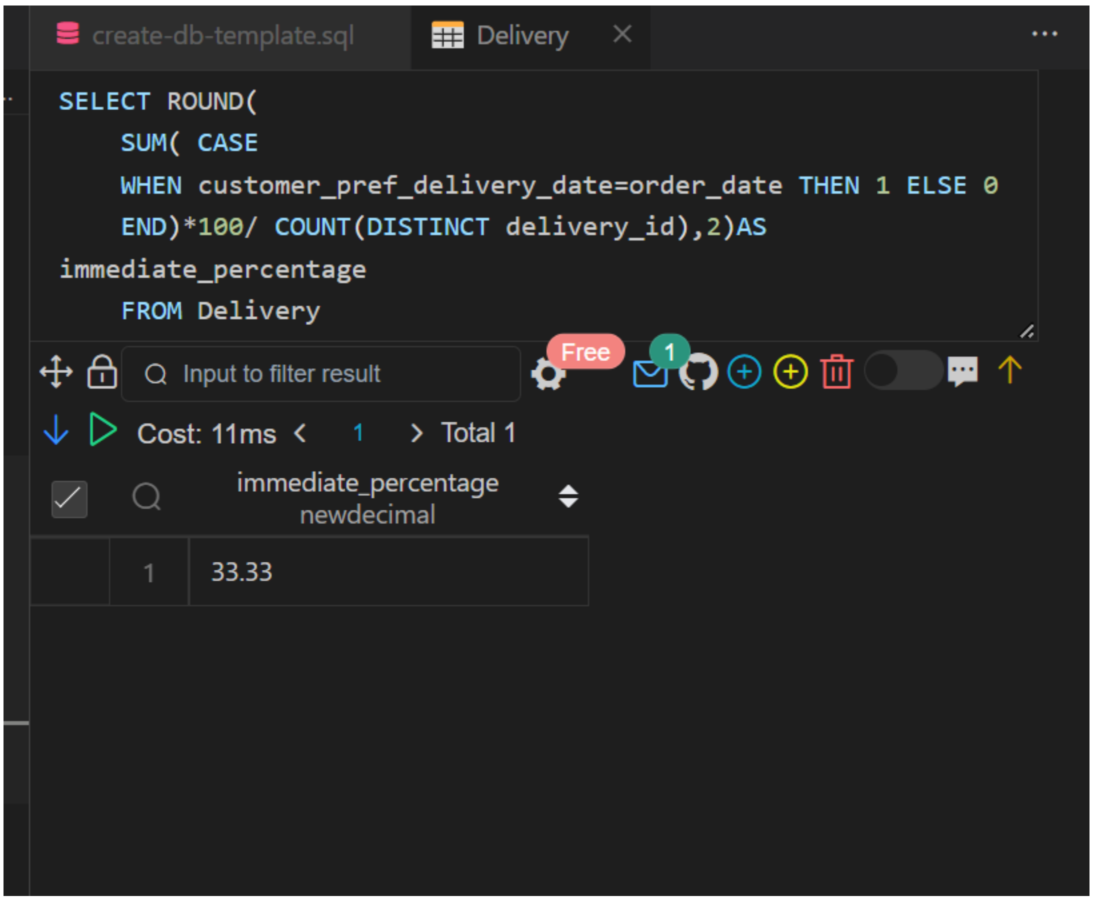
```
CREATE  TABLE Delivery(
delivery_id int PRIMARY key,
customer_id int,
order_date date,
customer_pref_delivery_date date
);
INSERT into Delivery VALUES(1,1,'2019-08-01','2019-08-02'),
(2,5,'2019-08-02','2019-08-02'),
(3,1,'2019-08-11','2019-08-11'),
(4,3,'2019-08-24', '2019-08-26'),
(5,4,'2019-08-21', '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:



```
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
```

```
Cost: 11ms < 1 > Total 1

   immediate_percentage
   newdecimal
1  33.33
```

Q20.

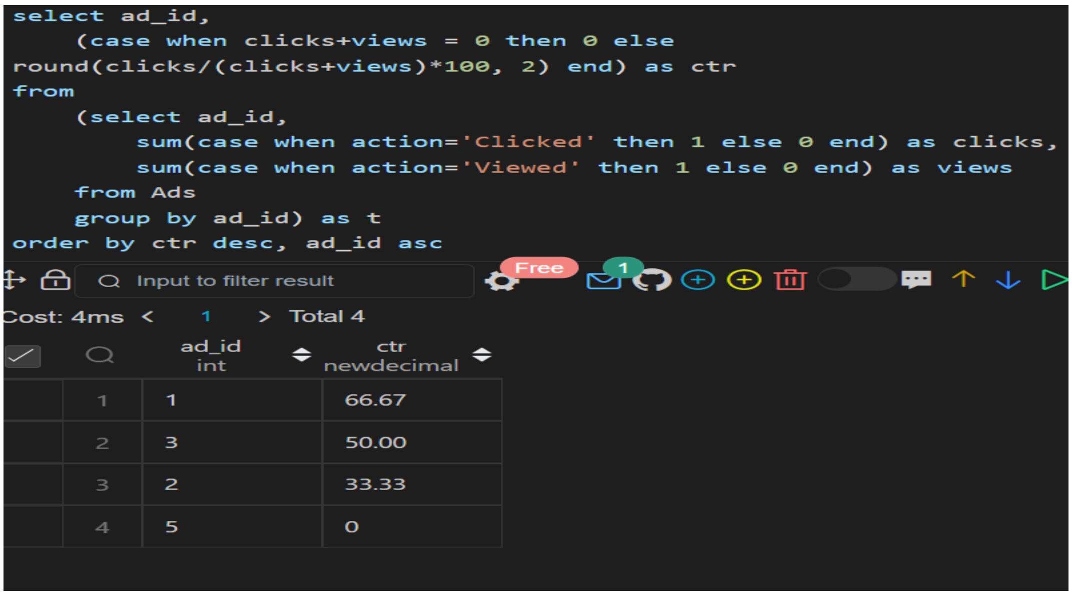
```
CREATE TABLE Ads(
ad_id int,
user_id int,
action enum('Clicked','Viewed','Ignored'),
 constraint pk Primary Key(ad_id,user_id)
);
INSERT INTO Ads VALUES(1,1,'Clicked'),
(2,2,'Clicked'),
(3,3,'Viewed'),
(5,5,'Ignored'),
(1,7,'Ignored'),
(2,7,'Viewed'),
(3,5,'Clicked'),
(1,4,'Viewed'),
(2,11,'Viewed'),
(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:



```
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
```

	ad_id int	ctr newdecimal
1	1	66.67
2	3	50.00
3	2	33.33
4	5	0

Q21.

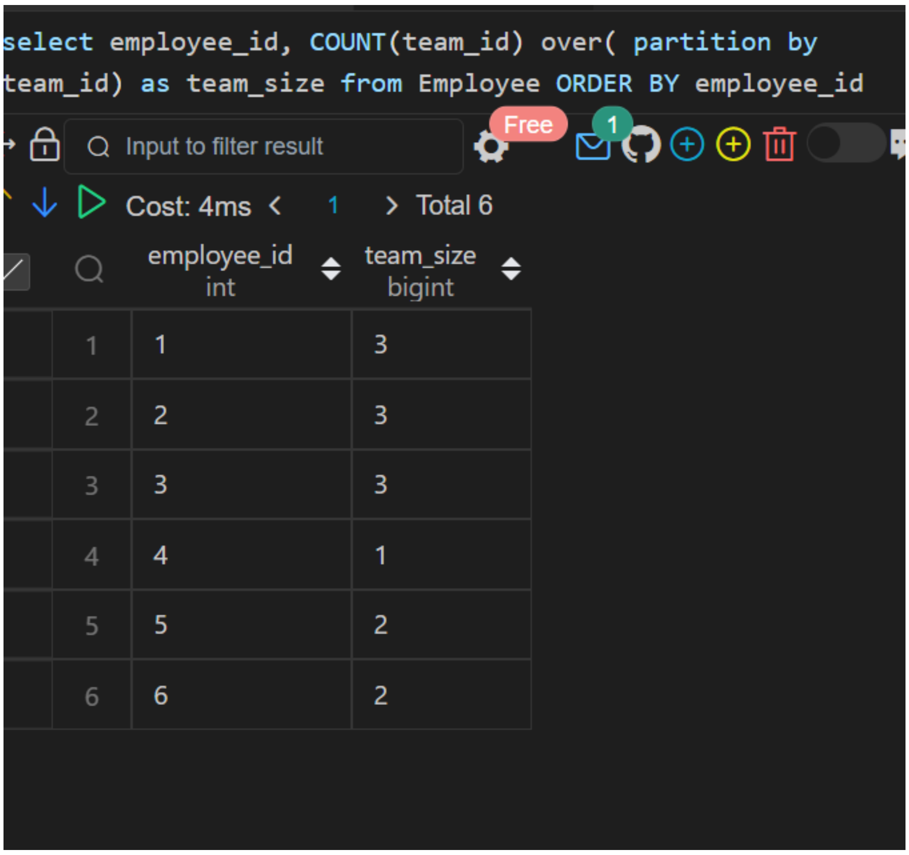
```
CREATE TABLE Employee(
employee_id int,
 team_id INT
);
insert into Employee values(1,8),(2,8),(3,8),(4,7),(5,9),(6,9);
```

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:



```
select employee_id, COUNT(team_id) over( partition by
team_id) as team_size from Employee ORDER BY employee_id
```

Cost: 4ms < 1 > Total 6

	employee_id int	team_size bigint
1	1	3
2	2	3
3	3	3
4	4	1
5	5	2
6	6	2

Q22.

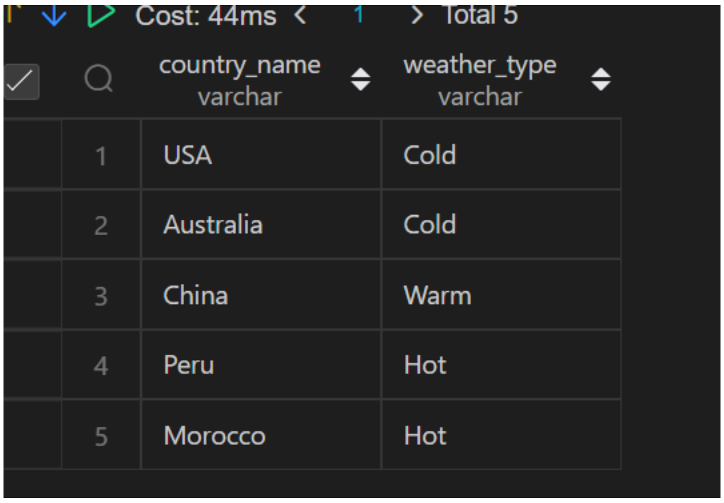
```
CREATE Table Countries(
country_id int PRIMARY key,
country_name varchar(20)
 );

 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:


Cost: 44ms 1 Total 5

	country_name varchar	weather_type varchar
1	USA	Cold
2	Australia	Cold
3	China	Warm
4	Peru	Hot
5	Morocco	Hot

Q23.

Q24.

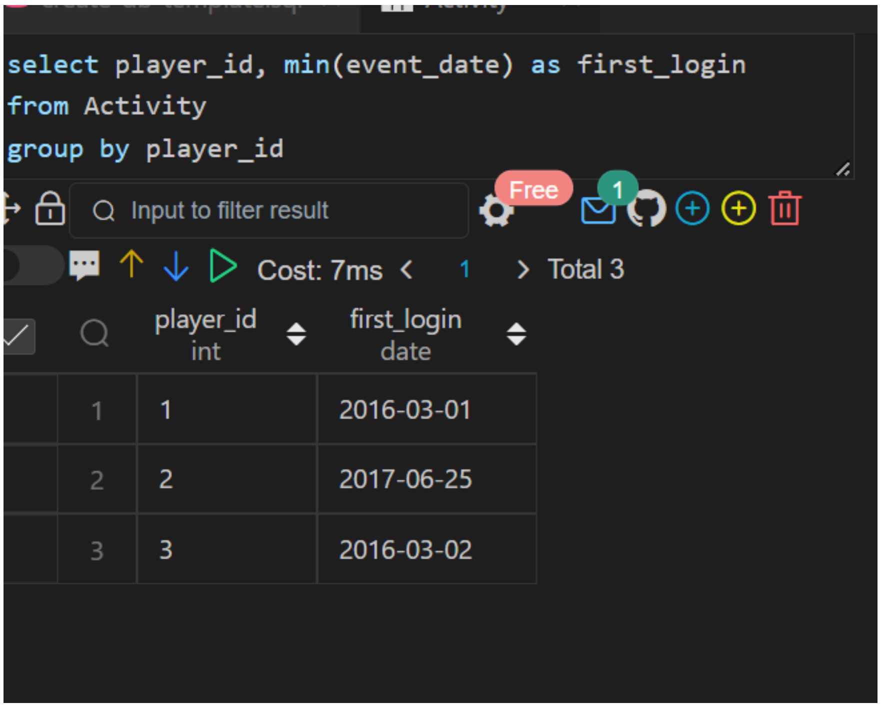
```
CREATE TABLE Activity(
player_id int,
device_id int,
event_date date,
games_played int
);

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:



```
select player_id, min(event_date) as first_login
from Activity
group by player_id
```

```
Input to filter result
Cost: 7ms  <  1  >  Total 3
```

	player_id int	first_login date
1	1	2016-03-01
2	2	2017-06-25
3	3	2016-03-02

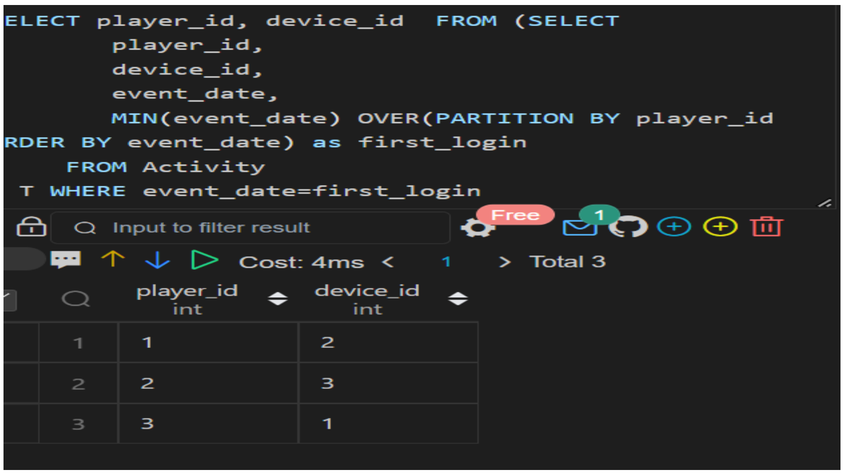
Q25.

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:



```
ELECT player_id, device_id  FROM (SELECT
       player_id,
       device_id,
       event_date,
       MIN(event_date) OVER(PARTITION BY player_id
RDER BY event_date) as first_login
    FROM Activity
 T WHERE event_date=first_login
```

```
Input to filter result
Cost: 4ms  <  1  >  Total 3
```

	player_id int	device_id int
1	1	2
2	2	3
3	3	1

Q26

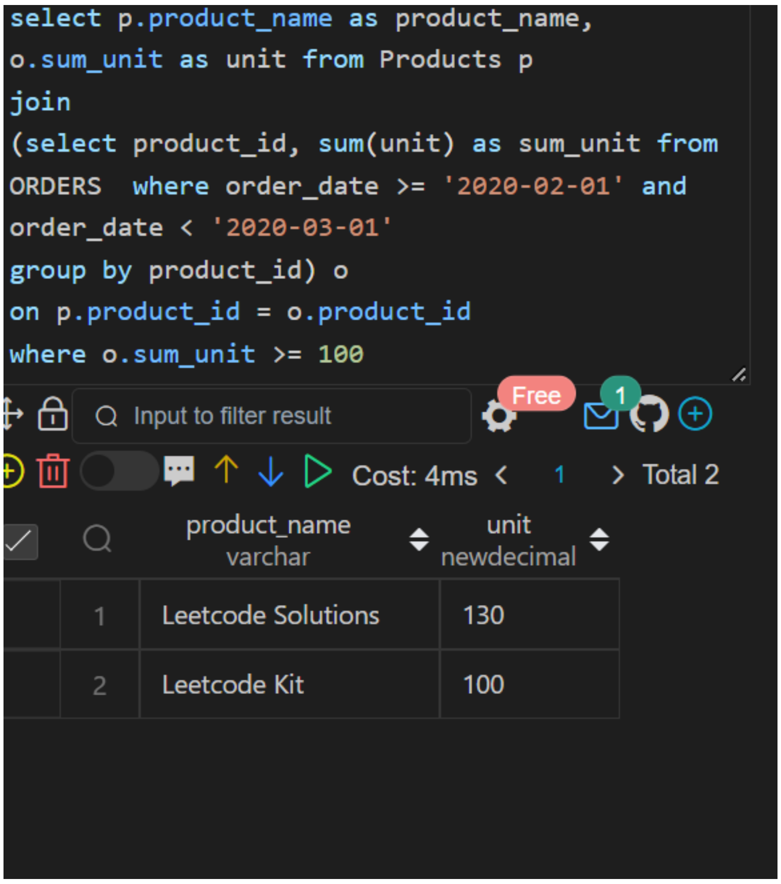
```
CREATE TABLE  Products(
product_id int,
product_name varchar(29),
product_category varchar(19)

);

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:



```
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
```

	product_name varchar	unit newdecimal
1	Leetcode Solutions	130
2	Leetcode Kit	100

Q27.

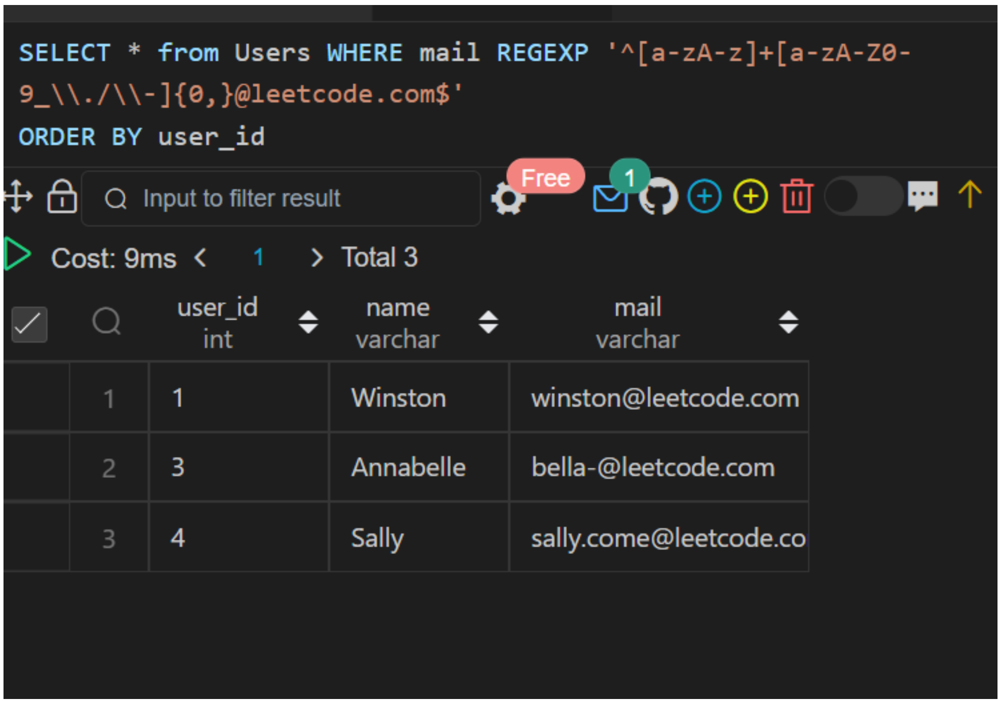
```
CREATE TABLE Users(
user_id int,
name varchar(33),
mail varchar(42)
);


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:



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

	user_id int	name varchar	mail varchar
1	1	Winston	winston@leetcode.com
2	3	Annabelle	bella-@leetcode.com
3	4	Sally	sally.come@leetcode.co

Q28)

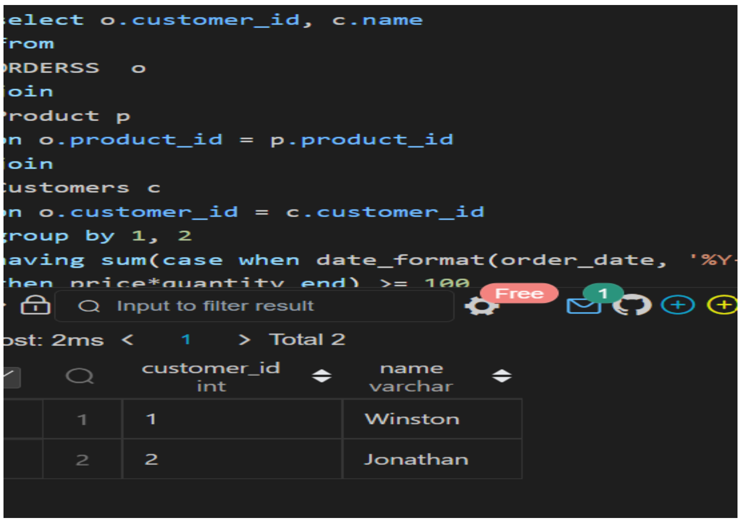
```
CREATE TABLE Customers(
customer_id int,
name varchar(33),
country varchar(33)
);
DROP TABLE Product;
CREATE TABLE Product(
product_id int PRIMARY KEY,
name varchar(33),
price int
);
CREATE TABLE ORDERSS(
order_id int PRIMARY KEY,
customer_id int,
product_id int,
order_date date,
quantity int
);
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:



```
elect o.customer_id, c.name
rom
RDERSS  o
oin
roduct p
n o.product_id = p.product_id
oin
ustomers c
n o.customer_id = c.customer_id
roup by 1, 2
aving sum(case when date_format(order_date, '%Y-
hen price*quantity end) >= 100
```

```
ost: 2ms  <   1   >  Total 2

      customer_id    name
      int            varchar
  1   1              Winston
  2   2              Jonathan
```

Q29)

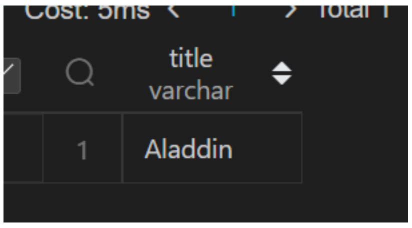
```
CREATE TABLE TV(

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:



```
Cost: 5ms <   1   > Total 1

     title
     varchar
1    Aladdin
```

Q30.

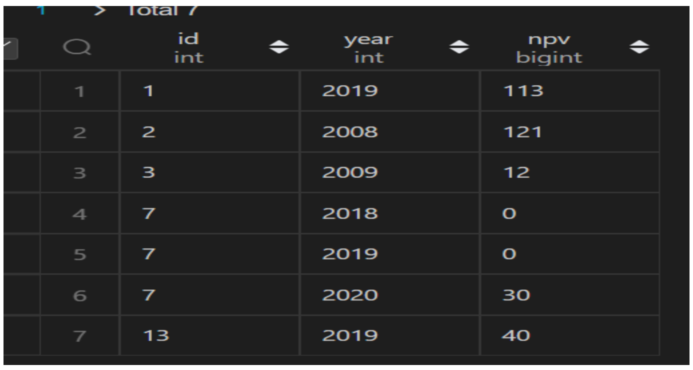
```
CREATE TABLE NPV(
id int,
year int,
npv int,
 constraint primary key(id, year)
);

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:



Total 7

	id int	year int	npv bigint
1	1	2019	113
2	2	2008	121
3	3	2009	12
4	7	2018	0
5	7	2019	0
6	7	2020	30
7	13	2019	40

Q31.

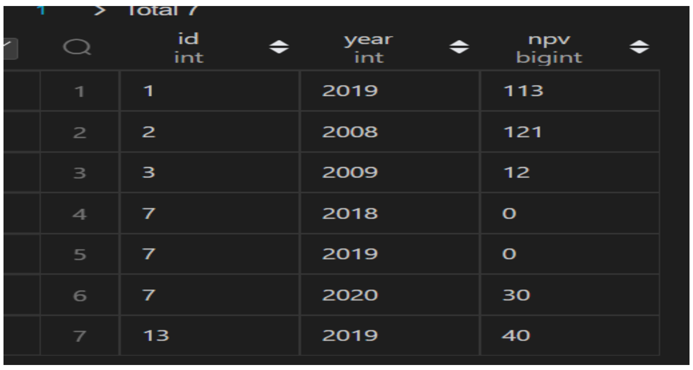
```
CREATE TABLE NPV(
id int,
year int,
npv int,
 constraint primary key(id, year)
);

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:



	id int	year int	npv bigint
1	1	2019	113
2	2	2008	121
3	3	2009	12
4	7	2018	0
5	7	2019	0
6	7	2020	30
7	13	2019	40

Q32.

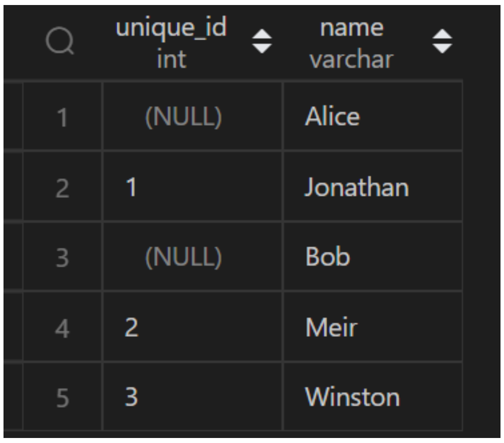
```
create TABLE Employees(
id int PRIMARY  KEY,
name varchar(40)

 );
  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:



	unique_id int	name varchar
1	(NULL)	Alice
2	1	Jonathan
3	(NULL)	Bob
4	2	Meir
5	3	Winston

Q33)

```
CREATE TABLE Users(
id int PRIMARY KEY,
name varchar(30)
);
```

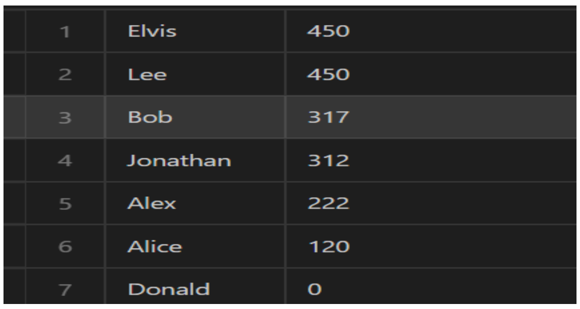
```
CREATE TABLE Rides(
id int PRIMARY KEY,
user_id int,
distance int
);

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:



1	Elvis	450
2	Lee	450
3	Bob	317
4	Jonathan	312
5	Alex	222
6	Alice	120
7	Donald	0

Q34)

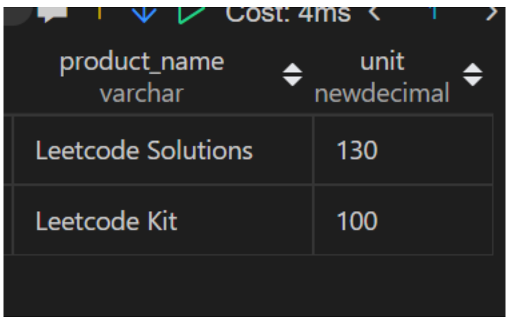
```sql
create TABLE Products(
product_id int PRIMARY KEY,
product_name varchar(30),
product_category varchar(30)
 );

  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:

```sql
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:



product_name varchar	unit newdecimal
Leetcode Solutions	130
Leetcode Kit	100

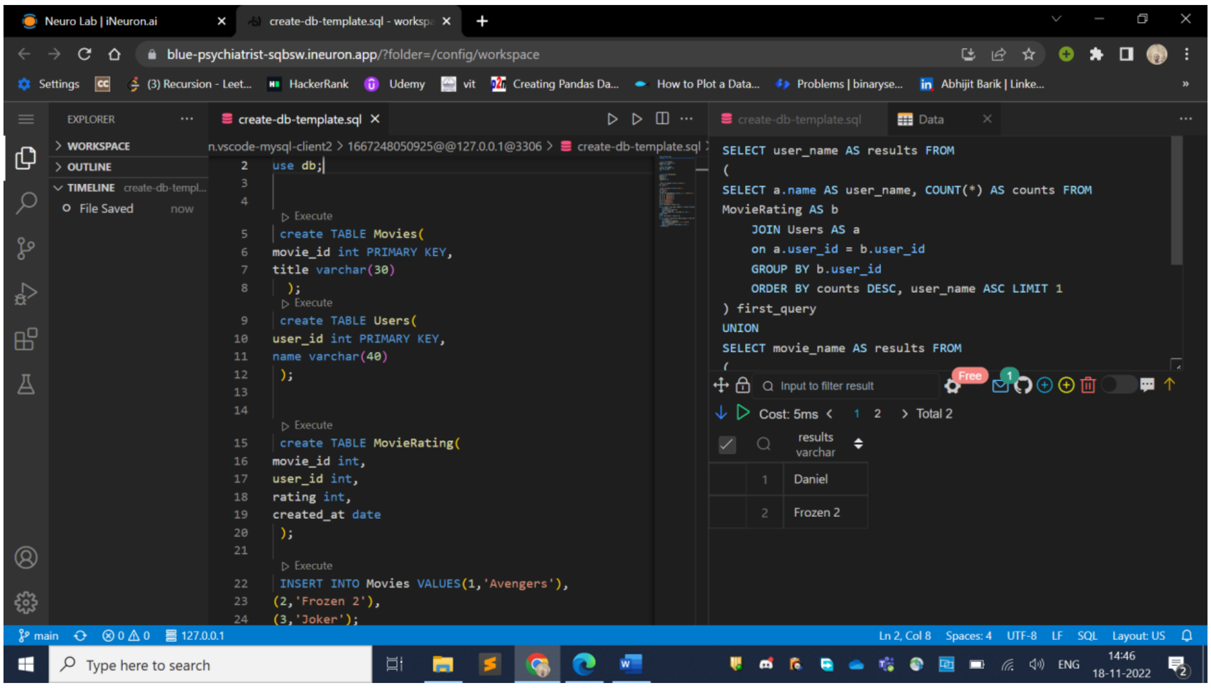
Q35

SQL QUERY:

```sql
 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:



```
use db;

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');
```

```
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
(
```

```
Cost: 5ms   Total 2

   results
   varchar
1  Daniel
2  Frozen 2
```

Q36)

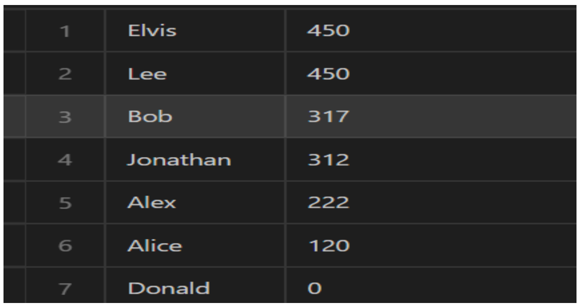
```
CREATE TABLE Users(
id int PRIMARY KEY,
name varchar(30)
);

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:



1	Elvis	450
2	Lee	450
3	Bob	317
4	Jonathan	312
5	Alex	222
6	Alice	120
7	Donald	0

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)
);
```

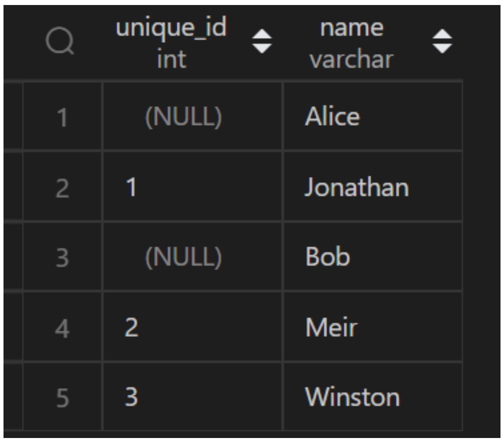
```
INSERT INTO Employees VALUES(1,'Alice'),
(7, 'Bob'),
(11, 'Meir'),
(90,'Winston'),
(3,'Jonathan');
INSERT INTO EmployeeUNI VALUES(3,1),
(11, 2),
(90,3);
```

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:



	unique_id int	name varchar
1	(NULL)	Alice
2	1	Jonathan
3	(NULL)	Bob
4	2	Meir
5	3	Winston

Q38.

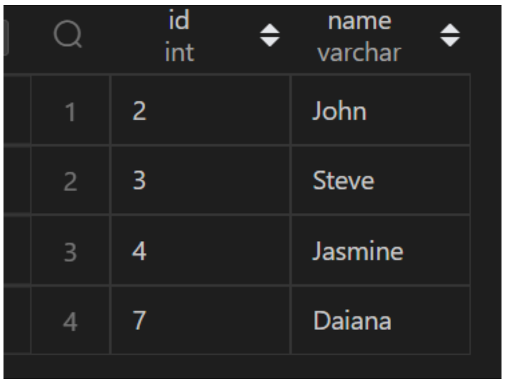
```
CREATE TABLE  Departments(
id int PRIMARY KEY,
name varchar(30)
);
CREATE TABLE Students(
id int PRIMARY KEY,
name varchar(30),
department_id int
);
INSERT INTO  Departments VALUES(1,'Electrical Engineering'),
(7,'Computer Engineering'),
(13,'Business Administration');


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:



	id int	name varchar
1	2	John
2	3	Steve
3	4	Jasmine
4	7	Daiana

Q39.

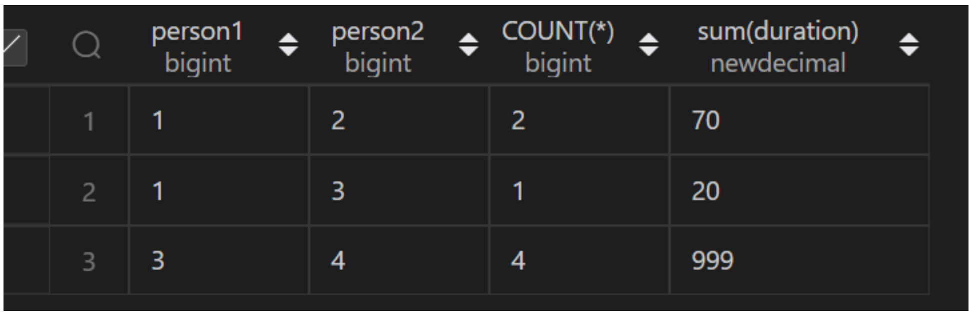
```
 CREATE TABLE Calls(
from_id int,
to_id int,
duration int
 );
 INSERT INTO Calls VALUES(1,2,59),
(2,1,11),
(1,3,20),
(3,4,100),
(3,4,200),
(3,4,200),
(4,3,499);
```

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:



	person1 bigint	person2 bigint	COUNT(*) bigint	sum(duration) newdecimal
1	1	2	2	70
2	1	3	1	20
3	3	4	4	999

Q40.

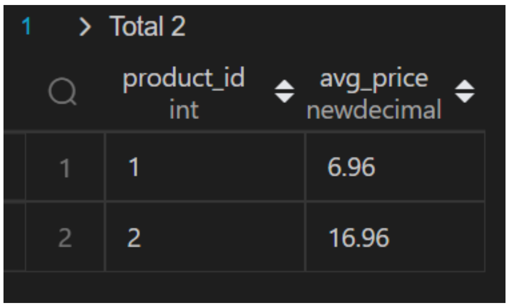
```
CREATE TABLE Pricess(
product_id int,
start_date date,
end_date date,
price int,
constraint pk PRIMARY key (product_id,start_date,end_date)
);
CREATE TABLE  UnitsSold(
product_id int,
purchase_date date,
units int
);
insert into Pricess VALUES(1,'2019-02-17','2019-02-28',5),
(1,'2019-03-01','2019-03-22',20),
(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:



```
1  > Total 2
     product_id   avg_price
     int          newdecimal
1    1            6.96
2    2            16.96
```

Q41.

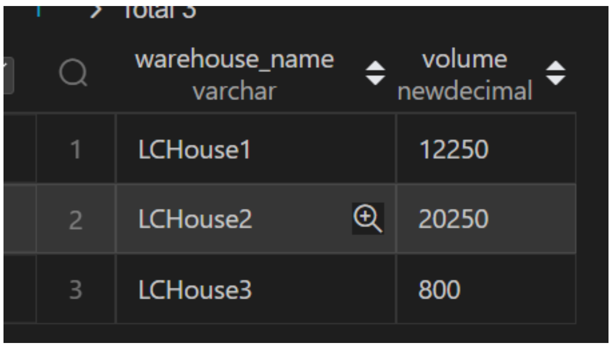
```
CREATE TABLE Warehouse(
name varchar(128),
product_id int,
units int,
CONSTRAINT PK PRIMARY KEY(name,product_id)
);

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:



	warehouse_name varchar	volume newdecimal
1	LCHouse1	12250
2	LCHouse2	20250
3	LCHouse3	800

Q42.

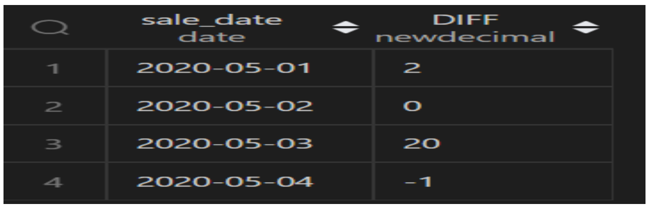
```
CREATE Table Sales(
sale_date date,
fruit enum("apples","oranges"),
sold_num int,
CONSTRAINT PK PRIMARY KEY(sale_date, fruit)
);
INSERT INTO Sales VALUES ('2020-05-01','apples', 10),
('2020-05-01', 'oranges',8),
('2020-05-02','apples', 15),
('2020-05-02','oranges',15),
('2020-05-03','apples',20),
('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:



	sale_date date	DIFF newdecimal
1	2020-05-01	2
2	2020-05-02	0
3	2020-05-03	20
4	2020-05-04	-1

Q43

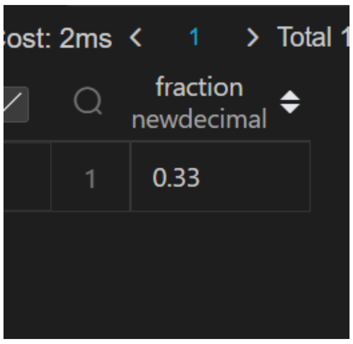
```
CREATE TABLE Activity(
player_id int,
device_id int,
event_date date,
games_played int,
CONSTRAINT PK PRIMARY KEY(player_id, event_date)
 );

 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:


```
ost: 2ms  <  1  >  Total 1
      fraction
      newdecimal
   1  0.33
```

Q44.

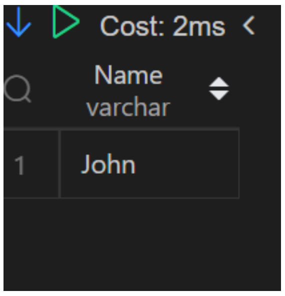
```
create TABLE Employee(
id int PRIMARY key,
name varchar(30),
department varchar(30),
managerId int
 );

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:



```
Cost: 2ms
   Name
   varchar
1  John
```

Q45.

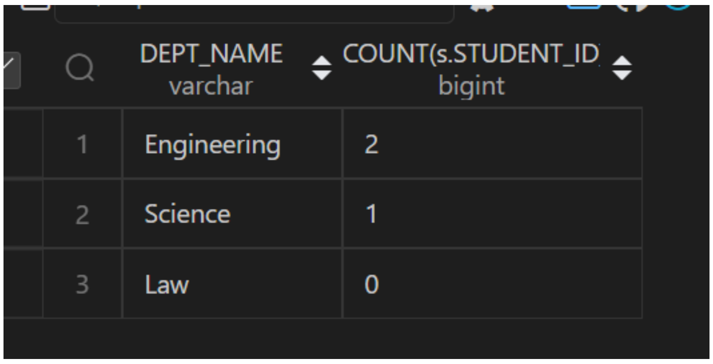
```
create TABLE Student(
student_id int,
student_name varchar(30),
gender varchar(30),
dept_id int,
constraint  PRIMARY KEY(student_id),
constraint  Foreign Key(dept_id) REFERENCES Department(dept_id)
);

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:

```sql
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:



	DEPT_NAME varchar	COUNT(s.STUDENT_ID bigint
1	Engineering	2
2	Science	1
3	Law	0

Q46.

```sql
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);
```

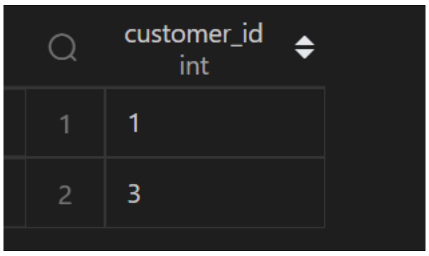
Write an SQL query to report the customer ids from the Customer table that bought all the products in the Product table

SQL QUERY:

```sql
SELECT
```

```
    customer_id
FROM Customer
GROUP BY customer_id
HAVING COUNT( DISTINCT product_key) = (SELECT COUNT(*) FROM Product)
```

OUTPUT:



	customer_id int
1	1
2	3

Q47

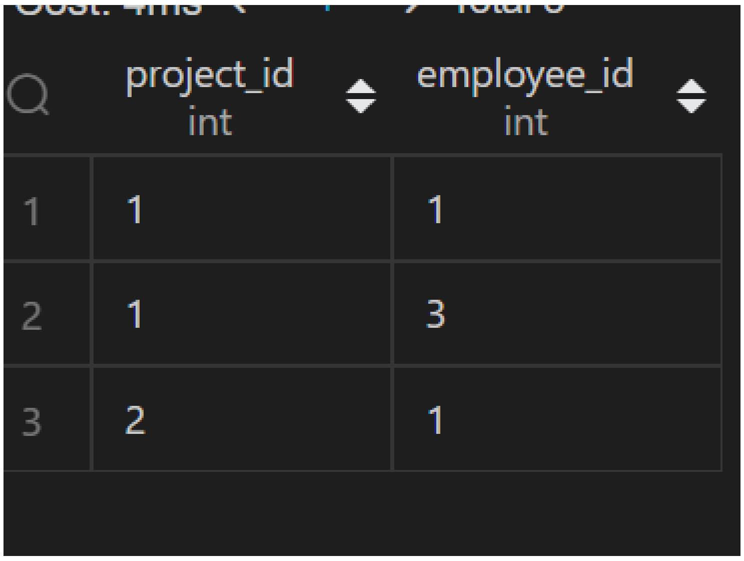
```
CREATE TABLE Project(
project_id int,
employee_id int,
constraint pk PRIMARY KEY(project_id, employee_id),
constraint fk FOREIGN KEY(employee_id) REFERENCES (employee_id)
);

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:



	project_id int	employee_id int
1	1	1
2	1	3
3	2	1

Q48.

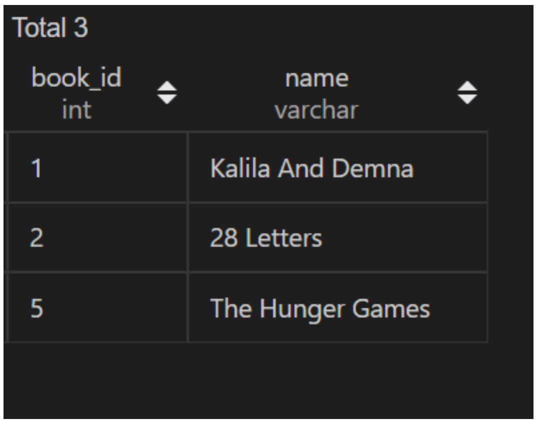
```
create table Books(
book_id int PRIMARY KEY,
name varchar(30),
available_from date
);
CREATE TABLE Orders(

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

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:



Total 3

book_id int	name varchar
1	Kalila And Demna
2	28 Letters
5	The Hunger Games

Q49.

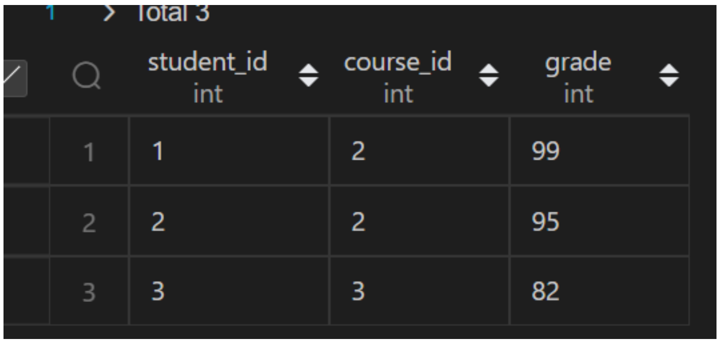
```
CREATE TABLE Enrollments(
student_id int,
course_id int,
grade int,
CONSTRAINT PK PRIMARY KEY(student_id, course_id)
);

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:



```
1  > Total 3
     student_id  course_id  grade
     int         int        int
1    1           2          99
2    2           2          95
3    3           3          82
```

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);
```

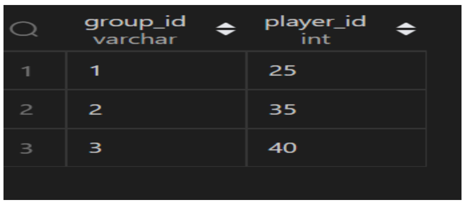
```
insert into Matches VALUES(1,15,45,3,0),
(2,30,25,1,2),(3,30,25,1,2),
(4,40,20,5,2),(5,35,50,1,1);
```

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:



	group_id varchar	player_id int
1	1	25
2	2	35
3	3	40

Q51.

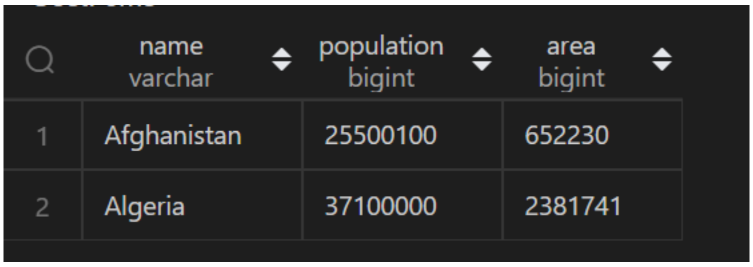
```
create TABLE world(
name varchar(30) PRIMARY KEY,
continent varchar(30),
area bigint,
population bigint,
gdp bigint
);
```

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:



	name varchar	population bigint	area bigint
1	Afghanistan	25500100	652230
2	Algeria	37100000	2381741

Q52.

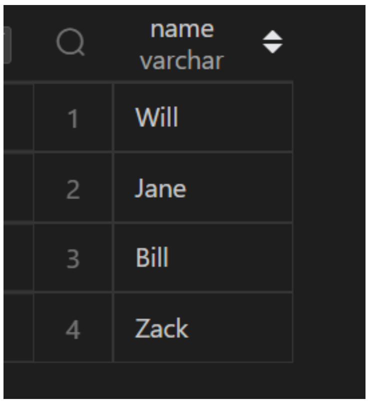
```
CREATE TABLE customer(
id int PRIMARY KEY,
name varchar(30),
referee_id int
);


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:



	name varchar
1	Will
2	Jane
3	Bill
4	Zack

Q53.

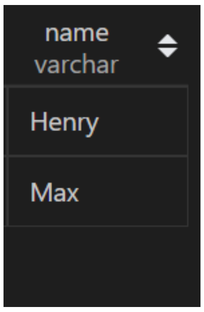
```
create TABLE Customers(
id int PRIMARY KEY,
name varchar(30)
  );
insert into Customers values(1,"Joe"),(2, "Henry"),(3, "Sam"),(4,"Max");
 create TABLE Orders(
id int PRIMARY KEY,
customerId INT
 );
 insert  into Orders VALUES(1, 3),(2, 1);
```

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:



name varchar
Henry
Max

Q54.

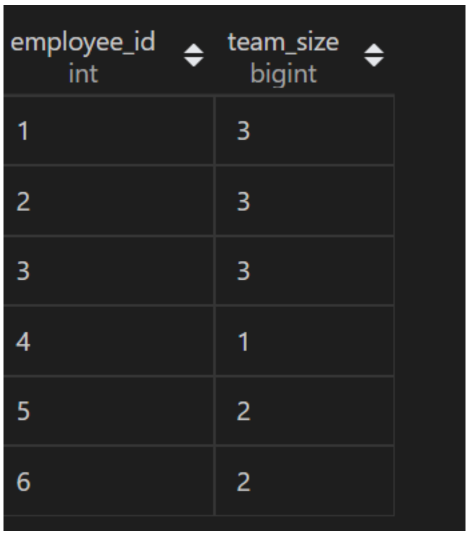
```
create table Employee(
employee_id int,
team_id int
 );
 INSERT INTO Employee values(1,8),
(2,8),(3,8),
(4,7),
(5,9),
(6,9);
```

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:



employee_id int	team_size bigint
1	3
2	3
3	3
4	1
5	2
6	2

Q55

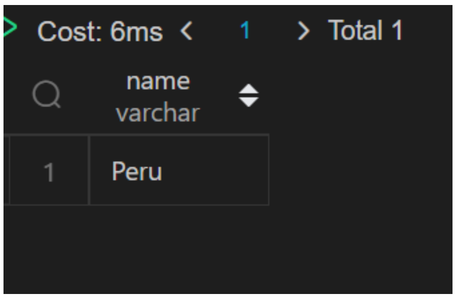
```
CREATE TABLE person(
id int PRIMARY KEY,
name varchar(30),
phone_number varchar(30)
);
 CREATE TABLE country(
name varchar(30),
country_code varchar(30) PRIMARY KEY
 );



  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:



```
Cost: 6ms <   1   > Total 1
     name
     varchar
1    Peru
```

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);
```

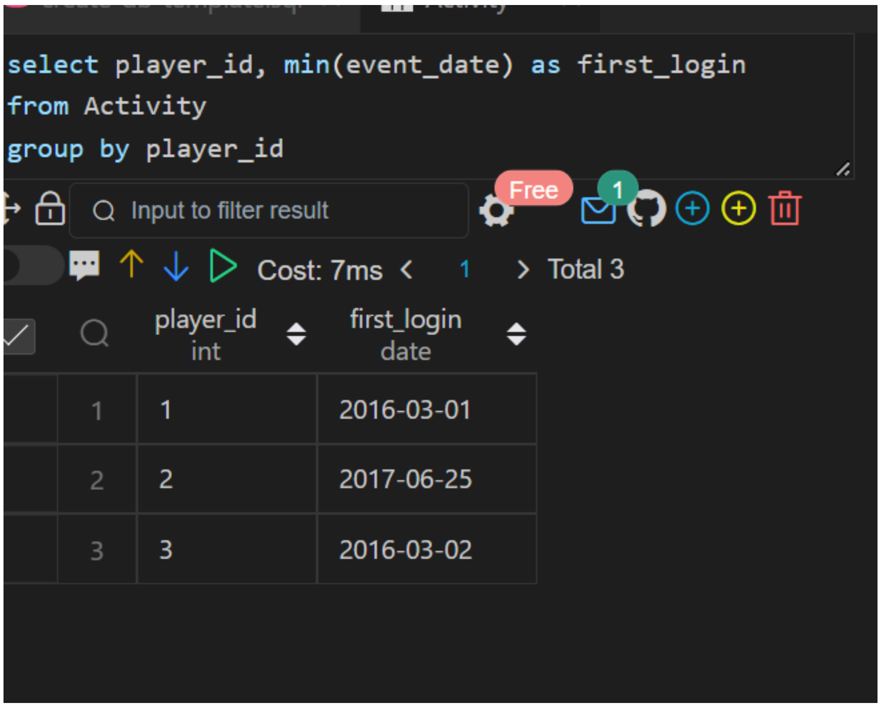
Write an SQL query to report the first login date for each player.

SQL QUERY:

```
select player_id, min(event_date) as first_login
```

```
from Activity
group by player_id;
```

OUTPUT:



```
select player_id, min(event_date) as first_login
from Activity
group by player_id
```

Input to filter result

Cost: 7ms < 1 > Total 3

	player_id int	first_login date
1	1	2016-03-01
2	2	2017-06-25
3	3	2016-03-02

Q57.

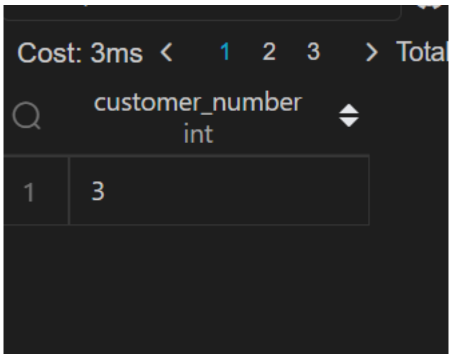
```
reate table Orders(
order_number int PRIMARY key,
customer_number int
);
INSERT into Orders VALUES(1,1),(2,2),(3,3),(4,3);
```

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:


Cost: 3ms < 1 2 3 > Total

	customer_number int
1	3

Q58.

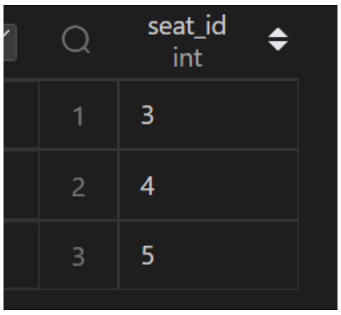
```
CREATE TABLE  Cinema(
seat_id int  PRIMARY KEY AUTO_INCREMENT,
free bool
);
INSERT INTO Cinema VALUES(1,1),
(2,0),
(3,1),
(4,1),
(5,1);
```

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:



	seat_id int
1	3
2	4
3	5

Q59.

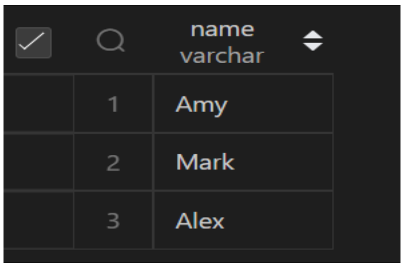
```
create TABLE SalesPerson(
sales_id int PRIMARY KEY,
name varchar(30),
salary int,
commission_rate int,
hire_date date
);

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:



	name varchar
1	Amy
2	Mark
3	Alex

Q60.

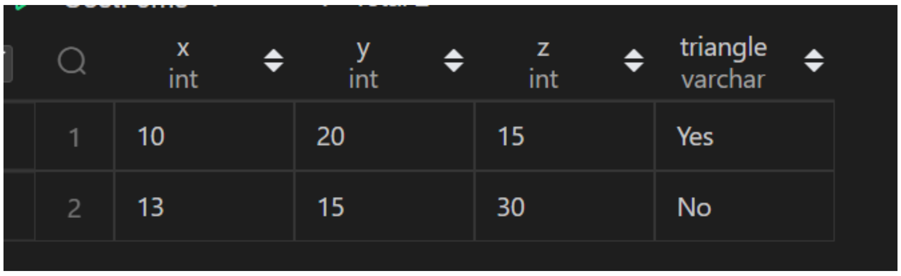
```
create DATABASE db;
use db;
create TABLE Triangle (
x int,
y int,
z int,
constraint pk PRIMARY KEY(x,y,z)

);
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:



	x int	y int	z int	triangle varchar
1	10	20	15	Yes
2	13	15	30	No

Q61.

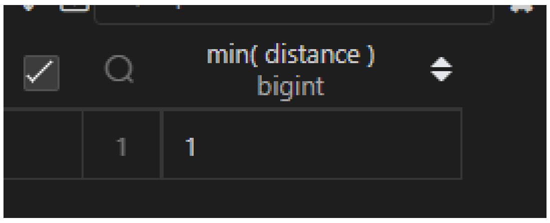
```
create table  Point(
x int
);
insert into Point  values (-1),(0),(2);
```

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:



```
min( distance )
bigint
1  1
```

Q62.

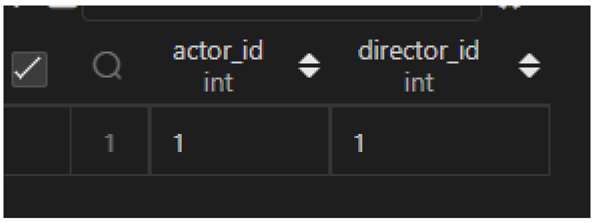
```
CREATE TABLE  ActorDirector(
actor_id int,
director_id int,
timestamp int primary key
);
insert into
ActorDirector  values(1,1,0),(1,1,1),(1,1,2),(1,2,3),(1,2,4),(2,1,5),(2,1,6);
```

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:



```
actor_id   director_id
int        int
1  1       1
```

Q63.

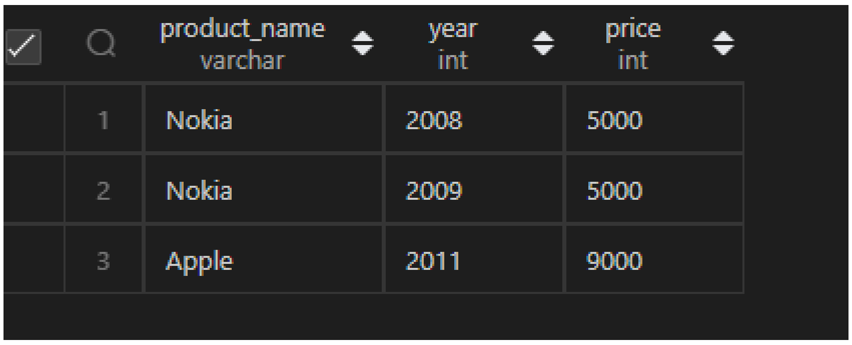
```
create Table Sales(
sale_id int,
product_id int,
year int,
quantity int,
price int,
constraint pk primary key(sale_id,year),
constraint fk foreign key(product_id) references Product(product_id)
);

 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:



	product_name varchar	year int	price int
1	Nokia	2008	5000
2	Nokia	2009	5000
3	Apple	2011	9000

Q64.

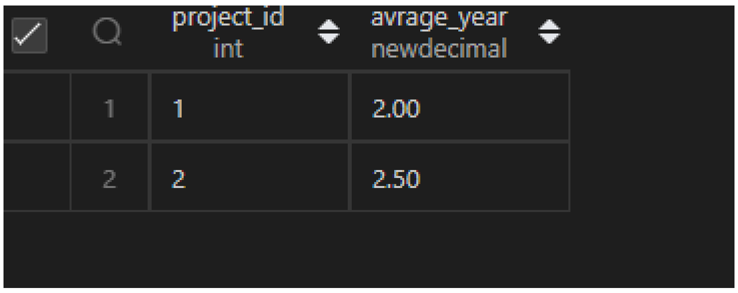
```
create Table Project(
project_id int,
employee_id int,
constraint pk primary key(project_id, employee_id)
);
INSERT INTO Project VALUES(1,1),(1,2),(1,3),(2,1),(2,4);


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:



```
   project_id   avrage_year
   int          newdecimal
1  1            2.00
2  2            2.50
```

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.

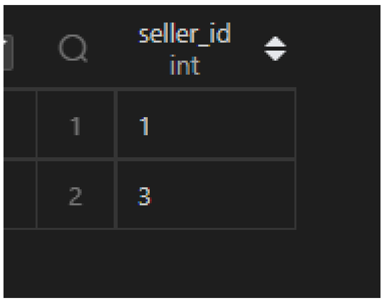
Query:

```
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:



	seller_id int
1	1
2	3

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);
```

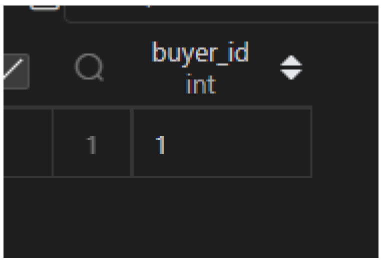
Write an SQL query that reports the buyers who have bought S8 but not iPhone. Note that S8 and iPhone are products present in the Product table.

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:



```
buyer_id
int
1   1
```

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);
```

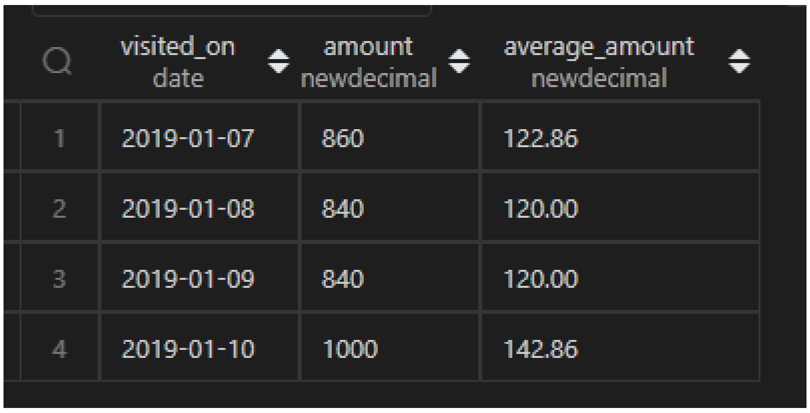
You are the restaurant owner and you want to analyse a possible expansion (there will be at least one customer every day)

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:



	visited_on date	amount newdecimal	average_amount newdecimal
1	2019-01-07	860	122.86
2	2019-01-08	840	120.00
3	2019-01-09	840	120.00
4	2019-01-10	1000	142.86

Q68.

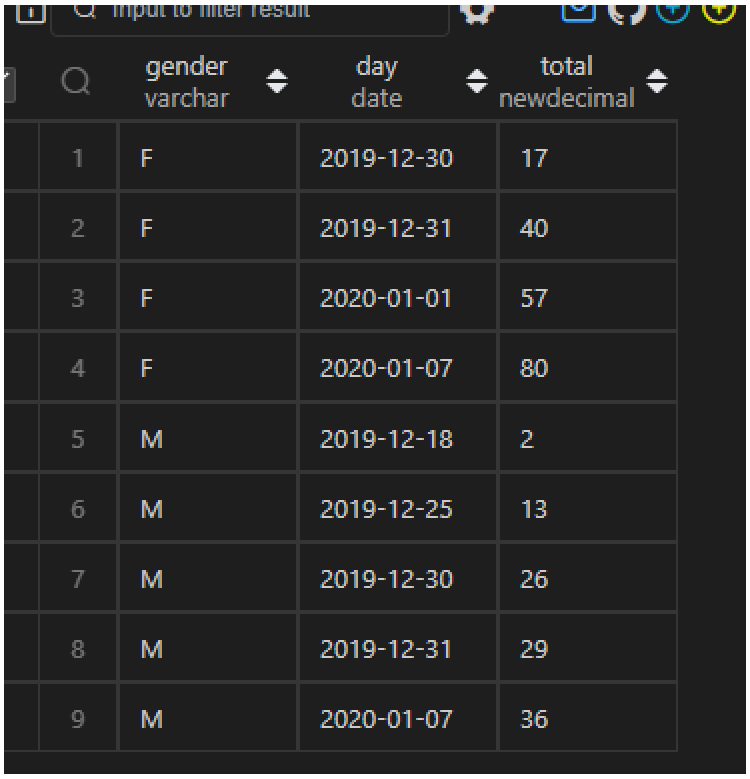
```
CREATE TABLE Scores(
player_name varchar(30),
gender varchar(30),
day date,
score_points int,
constraint pk PRIMARY KEY((gender, day)
);
insert into Scores values('Aron','F','2020-01-01', 17),
('Alice','F','2020-01-07',23),
('Bajrang','M','2020-01-07',7),
('Khali','M','2019-12-25',11),
('Slaman','M','2019-12-30', 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:



	gender varchar	day date	total newdecimal
1	F	2019-12-30	17
2	F	2019-12-31	40
3	F	2020-01-01	57
4	F	2020-01-07	80
5	M	2019-12-18	2
6	M	2019-12-25	13
7	M	2019-12-30	26
8	M	2019-12-31	29
9	M	2020-01-07	36

Q69.

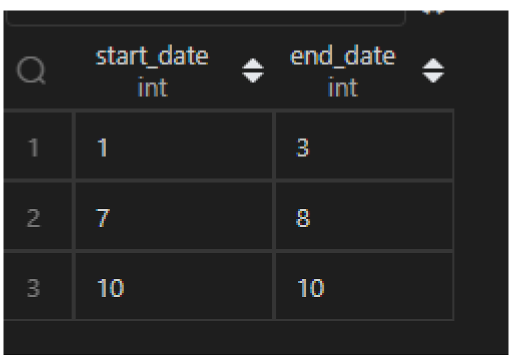
```
select gender,day, sum(score_points) over(partition by gender order by
gender,day  rows BETWEEN unbounded preceding and current row) as total
from  Scores;
create Table Logs(
log_id int
);
insert into Logs VALUES(1),(2),(3),(7),(8),(10);
```

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:



	start_date int	end_date int
1	1	3
2	7	8
3	10	10

70)

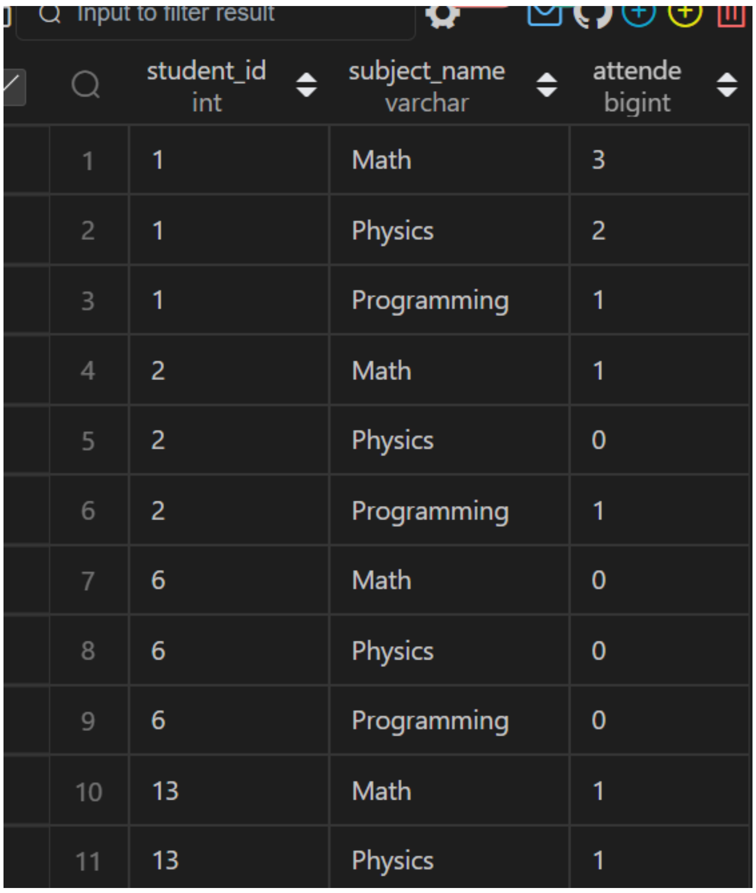
```
CREATE Table Students(
student_id int PRIMARY KEY,
student_name varchar(30)
);
CREATE Table Subjects(
subject_name varchar(30) PRIMARY KEY
);
 CREATE Table Examinations(
student_id int,
subject_name varchar(30)
 );
 INSERT INTO Students VALUES(1,'Alice'),(2,'Bob'),(13,'John'),(6,'Alex');

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:



	student_id int	subject_name varchar	attende bigint
1	1	Math	3
2	1	Physics	2
3	1	Programming	1
4	2	Math	1
5	2	Physics	0
6	2	Programming	1
7	6	Math	0
8	6	Physics	0
9	6	Programming	0
10	13	Math	1
11	13	Physics	1

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);
```

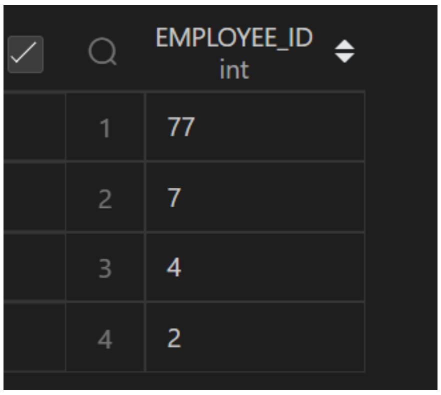
Write an SQL query to find employee_id of all employees that directly or indirectly report their work to the head of the company

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:



```
  EMPLOYEE_ID
  int
1 77
2 7
3 4
4 2
```

Q72.

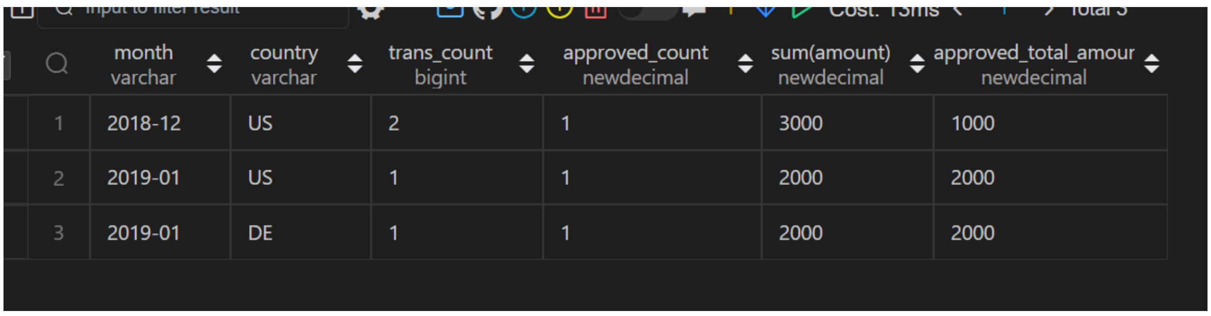
```
create Table Transactions(
id int PRIMARY KEY,
country varchar(30),
state enum("approved", "declined"),
amount int,
trans_date date
);
INSERT INTO  Transactions VALUES (121,'US', 'approved',1000,'2018-12-18'),
(122,'US','declined',2000,'2018-12-19'),
(123,'US','approved',2000,'2019-01-01'),
(124,'DE','approved',2000,'2019-01-07');
```

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:



	month varchar	country varchar	trans_count bigint	approved_count newdecimal	sum(amount) newdecimal	approved_total_amour newdecimal
1	2018-12	US	2	1	3000	1000
2	2019-01	US	1	1	2000	2000
3	2019-01	DE	1	1	2000	2000

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');
```

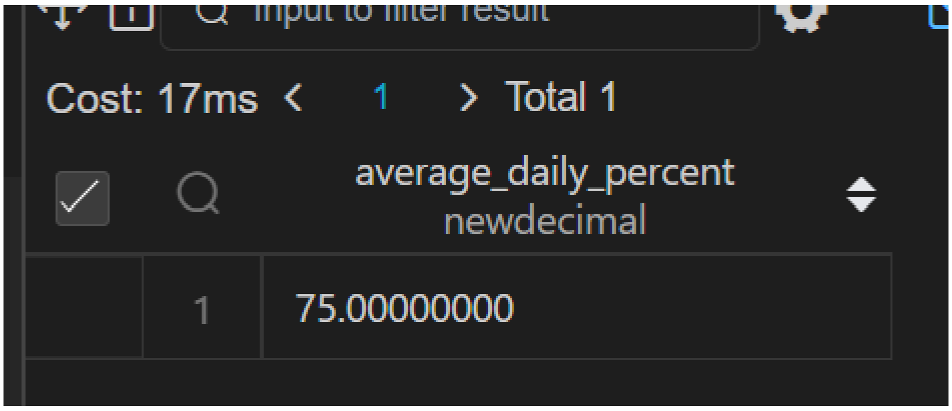
Write an SQL query to find the average daily percentage of posts that got removed after being reported as spam, rounded to 2 decimal places.

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:



```
Cost: 17ms  <  1  >  Total 1
average_daily_percent
newdecimal
1    75.00000000
```

Q74.

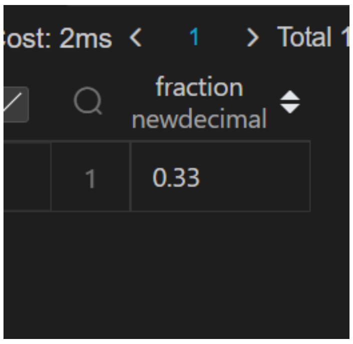
```
CREATE TABLE Activity(
player_id int,
device_id int,
event_date date,
games_played int
);

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:


```
ost: 2ms < 1 > Total 1
fraction
newdecimal
1   0.33
```

Q75.

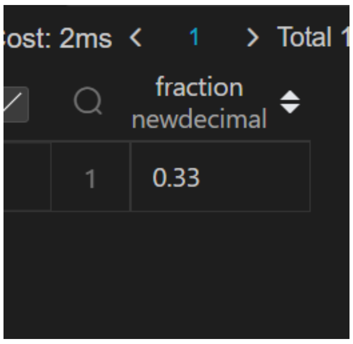
```
CREATE TABLE Activity(
player_id int,
device_id int,
event_date date,
games_played int
);

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:


```
ost: 2ms < 1 > Total 1
fraction
newdecimal
1   0.33
```

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.

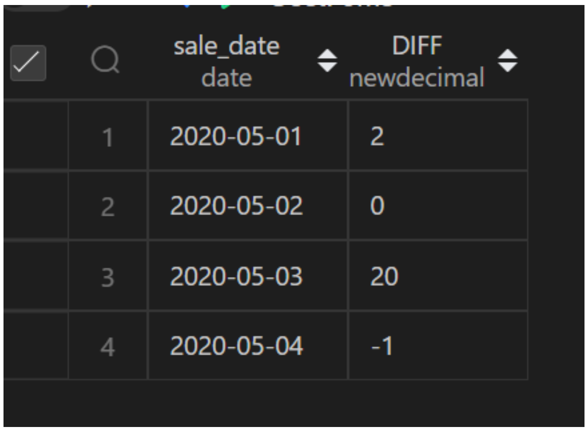
```
CREATE Table Sales(
sale_date date,
fruit enum("apples","oranges"),
sold_num int,
CONSTRAINT PK PRIMARY KEY(sale_date, fruit)
);
INSERT INTO Sales VALUES ('2020-05-01','apples', 10),
('2020-05-01', 'oranges',8),
('2020-05-02','apples', 15),
('2020-05-02','oranges',15),
('2020-05-03','apples',20),
('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:



	sale_date date	DIFF newdecimal
1	2020-05-01	2
2	2020-05-02	0
3	2020-05-03	20
4	2020-05-04	-1

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);
```

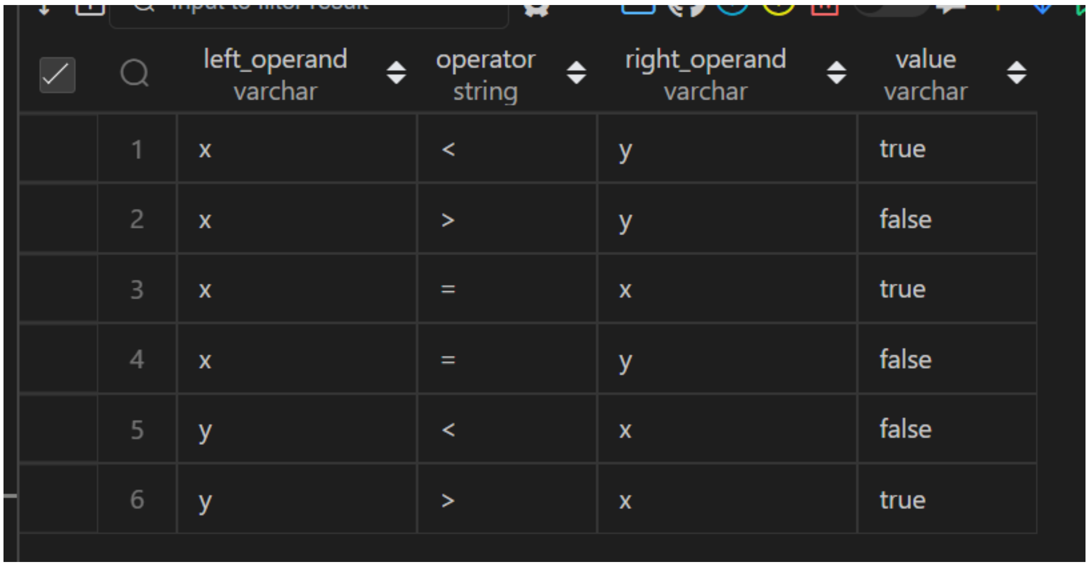
Write an SQL query to evaluate the boolean expressions in Expressions table. Return the result table in any order

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:



	left_operand varchar	operator string	right_operand varchar	value varchar
1	x	<	y	true
2	x	>	y	false
3	x	=	x	true
4	x	=	y	false
5	y	<	x	false
6	y	>	x	true

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:

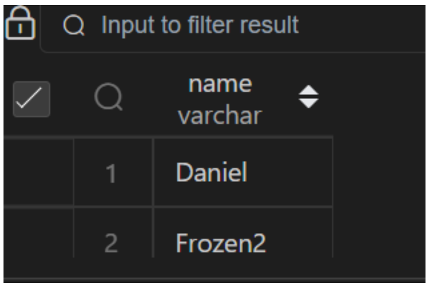
● Find the name of the user who has rated the greatest number of movies. In case of a tie, return the lexicographically smaller user name.

● 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



```
name
varchar
1  Daniel
2  Frozen2
```

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
```

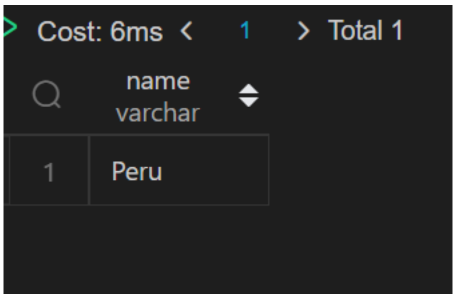
```
 );



  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:



```
Cost: 6ms <  1  > Total 1
   name
   varchar
1  Peru
```

Q81.

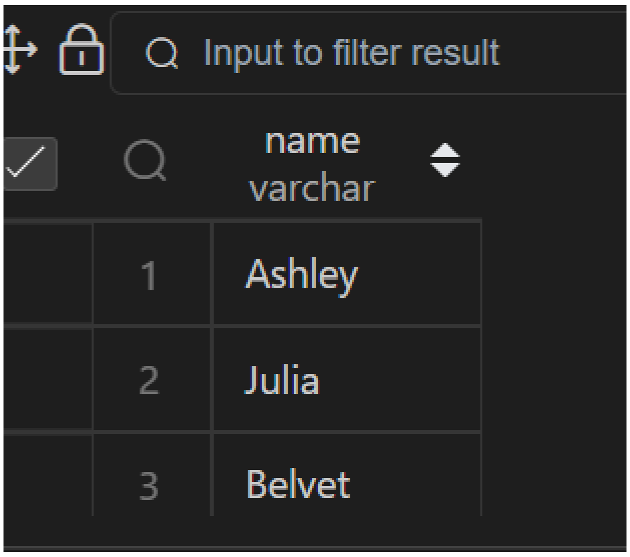
```
CREATE TABLE STUDENTS (
    ID INTEGER,
    NAME VARCHAR(30),
    MARKS VARCHAR(30)
);
insert into
STUDENTS  VALUES(1,'Ashley',81),(2,'Samantha',75),(4,'Julia',76),(3,'Belvet',8
4);
```

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:



```
Input to filter result
   name
   varchar
1  Ashley
2  Julia
3  Belvet
```

Q82.

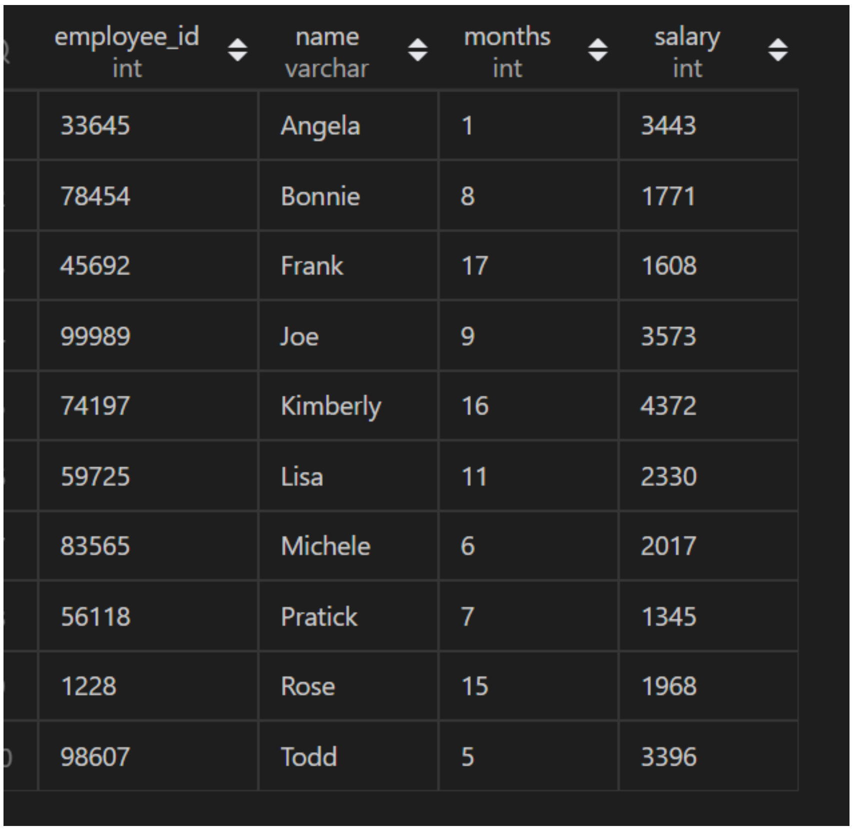
```
CREATE TABLE EMPLOYEE(
    employee_id INTEGER,
    name varchar(30),
    months INTEGER,
    salary INTEGER
);
insert into EMPLOYEE
VALUES(1228,'Rose',15,1968),(33645,'Angela',1,3443),(45692,'Frank',17,1608),
(56118,'Pratick',7,1345),(59725,'Lisa',11,2330),
(74197,'Kimberly',16,4372);
```

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:



employee_id int	name varchar	months int	salary int
33645	Angela	1	3443
78454	Bonnie	8	1771
45692	Frank	17	1608
99989	Joe	9	3573
74197	Kimberly	16	4372
59725	Lisa	11	2330
83565	Michele	6	2017
56118	Pratick	7	1345
1228	Rose	15	1968
98607	Todd	5	3396

Q83.

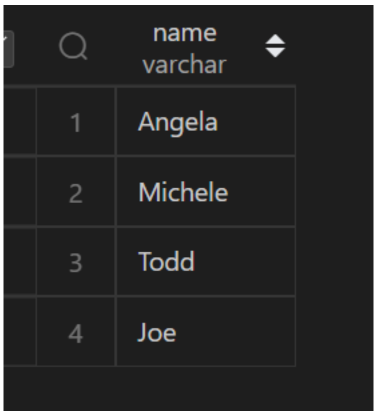
```
CREATE TABLE EMPLOYEE(
    employee_id INTEGER,
    name varchar(30),
    months INTEGER,
    salary INTEGER
);
insert into EMPLOYEE
VALUES(1228,'Rose',15,1968),(33645,'Angela',1,3443),(45692,'Frank',17,1608),
(56118,'Pratick',7,1345),(59725,'Lisa',11,2330),
(74197,'Kimberly',16,4372),(78454,'Bonnie',8,1771),
(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:



	name varchar
1	Angela
2	Michele
3	Todd
4	Joe

Q84.

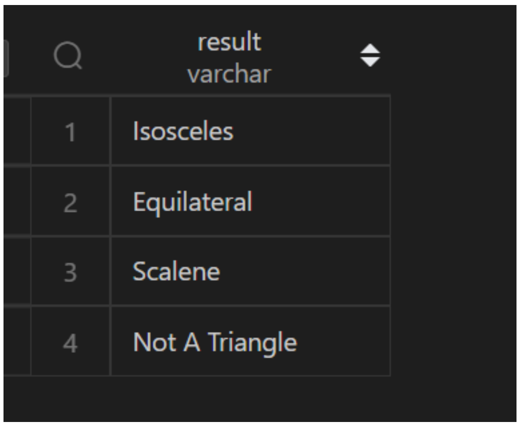
```
create table TRIANGLES(
    A integer,
    B integer,
    C integer
);
insert into TRIANGLES VALUES(20,20,23),(20,20,20),(20,21,22),(13,14,30);
```

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:



	result varchar
1	Isosceles
2	Equilateral
3	Scalene
4	Not A Triangle

85)

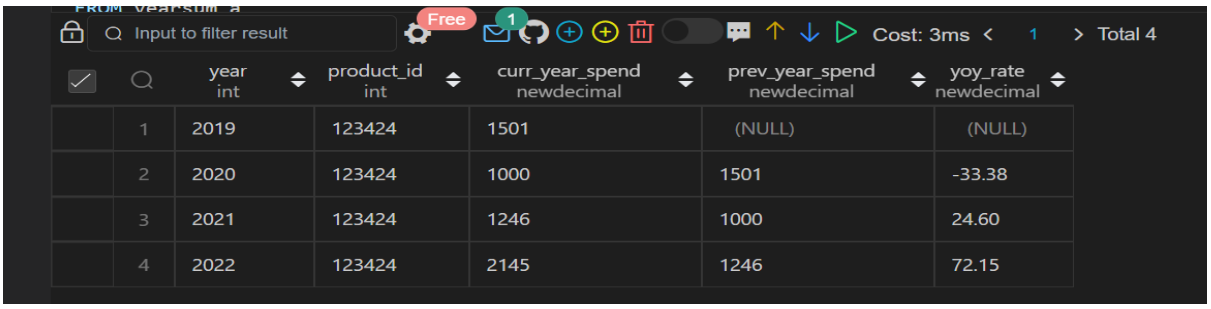
```
create table user_transactions(
transaction_id integer,
product_id integer,
spend decimal,
transaction_date datetime
);


;
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:



	year int	product_id int	curr_year_spend newdecimal	prev_year_spend newdecimal	yoy_rate newdecimal
1	2019	123424	1501	(NULL)	(NULL)
2	2020	123424	1000	1501	-33.38
3	2021	123424	1246	1000	24.60
4	2022	123424	2145	1246	72.15

86)

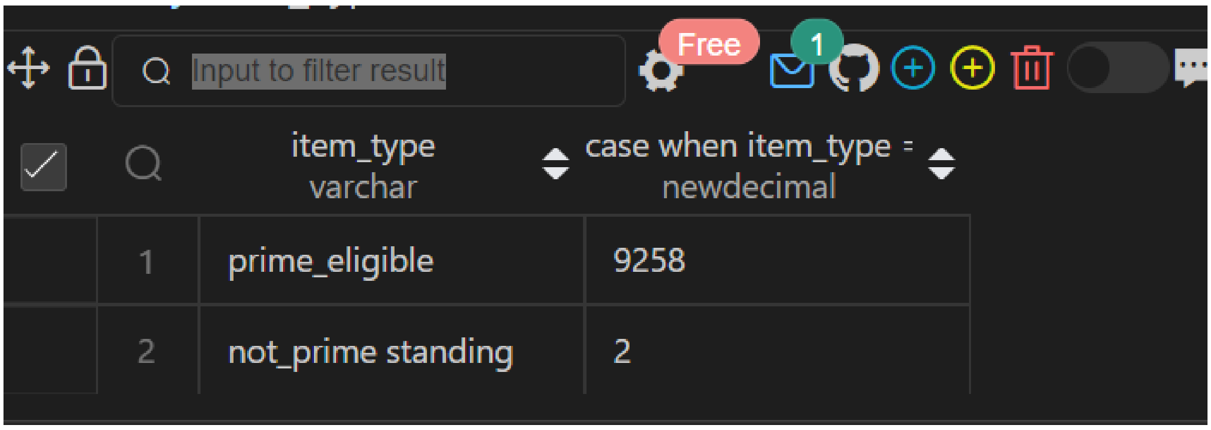
```
 create table inventory(
item_id integer,
item_type varchar(30),
item_category varchar(30),
square_footage decimal
 );

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:



	item_type varchar	case when item_type = newdecimal
1	prime_eligible	9258
2	not_prime standing	2

Q87)

```
Create Table user_actions (
users_id int,
event_id int,
event_type varchar(50),
event_date datetime );
```

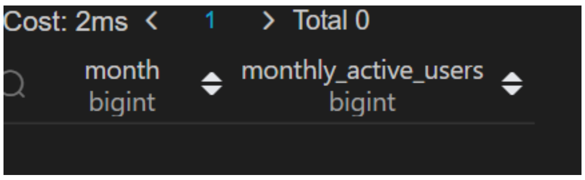
```
nsert into user_actions Values(445, 7765 ,'sign-in',STR_TO_DATE('05/31/2022
12:00:00','%m/%d/%Y %T'));
Insert into user_actions Values(445, 3634 ,'like',STR_TO_DATE('06/05/2022
12:00:00','%m/%d/%Y %T'));
Insert into user_actions Values(742, 6458 ,'sign-in', STR_TO_DATE('07/03/2022
12:00:00','%m/%d/%Y %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:



```
Cost: 2ms <   1   > Total 0
   month     monthly_active_users
   bigint    bigint
```

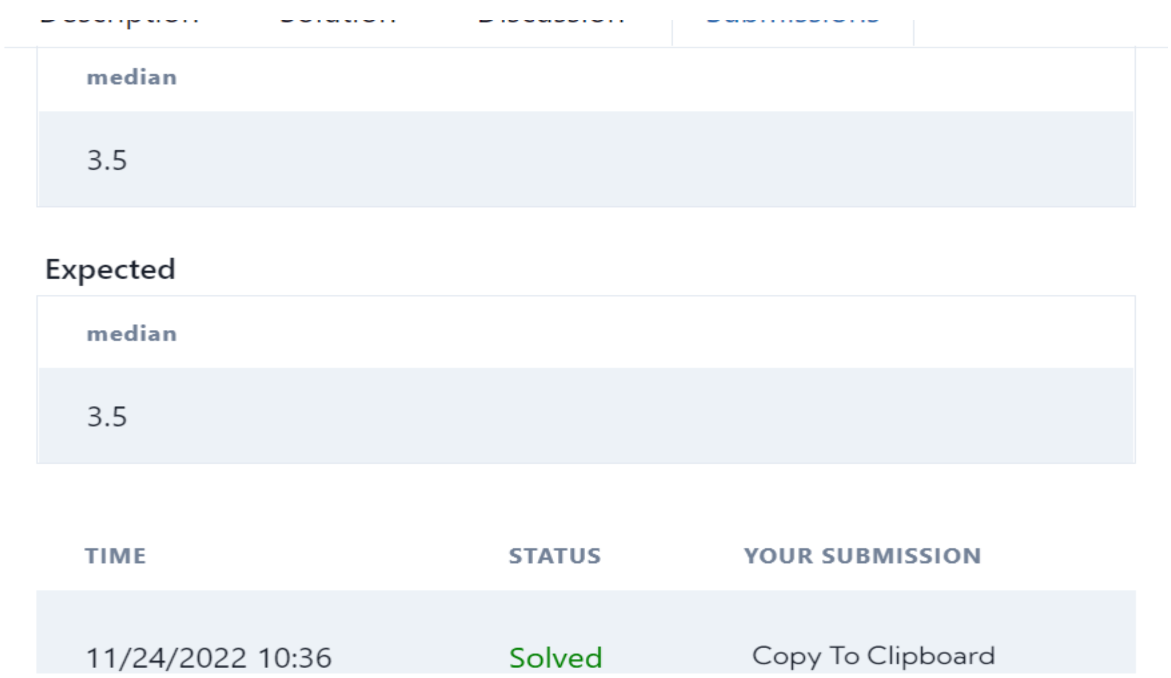
Q88)

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



median
3.5

Expected

median
3.5

TIME	STATUS	YOUR SUBMISSION
11/24/2022 10:36	Solved	Copy To Clipboard

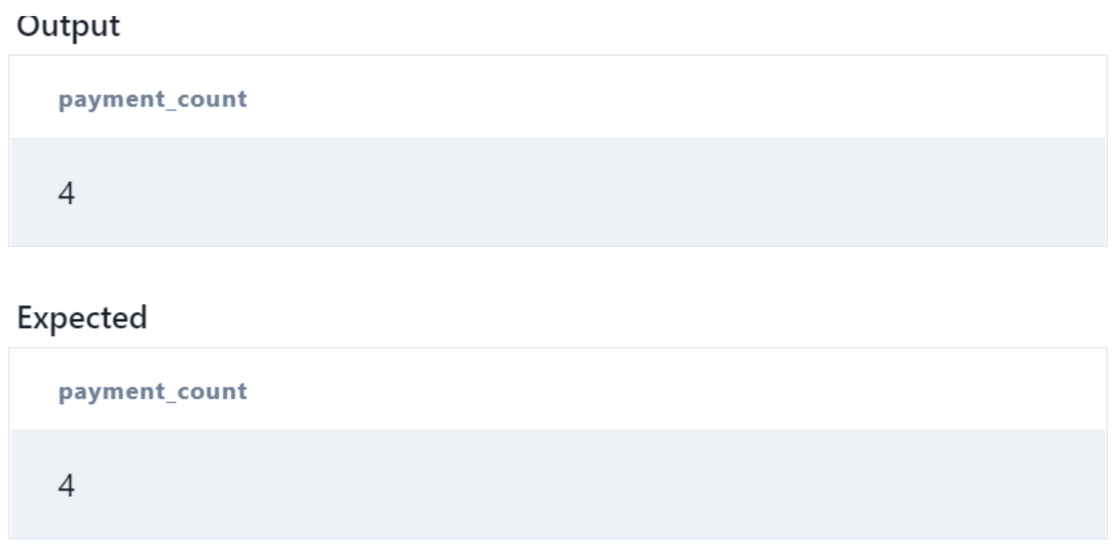
Q91.

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.transactio
n_timestamp)
;;
```

OUTPUT



Output

payment_count
4

Expected

payment_count
4

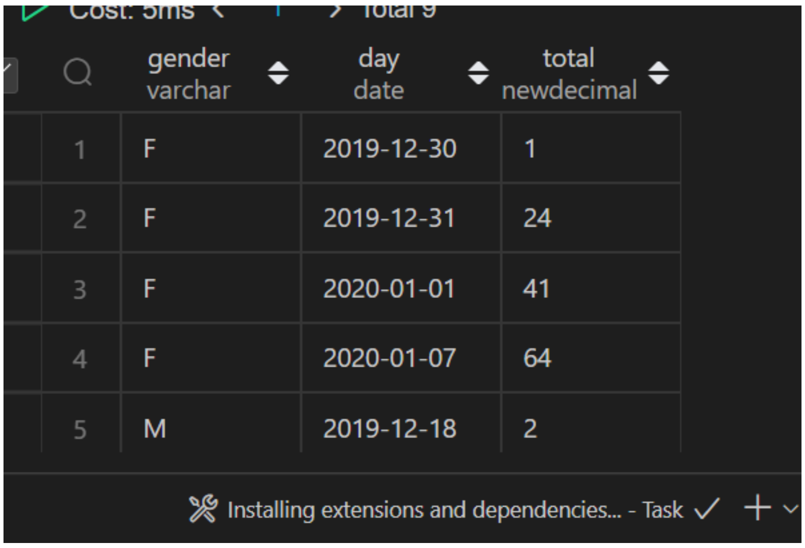
```
create Table Scores(
player_name varchar(30),
gender varchar(30),
day date,
score_points int,
 constraint pk PRIMARY key(gender, day)
);


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:



```
Cost: 5ms <  1  > Total 9
```

	gender varchar	day date	total newdecimal
1	F	2019-12-30	1
2	F	2019-12-31	24
3	F	2020-01-01	41
4	F	2020-01-07	64
5	M	2019-12-18	2

```
Installing extensions and dependencies... - Task
```

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');
```

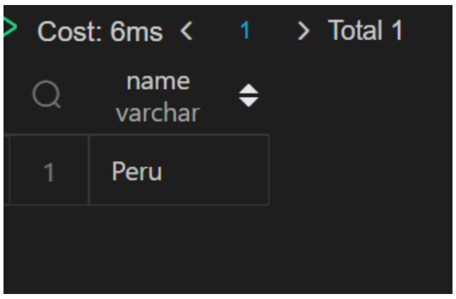
```
insert into calls values (1, 9, 33),(2, 9, 4),(1, 2, 59),(3, 12, 102),(3, 12,
330),(12, 3, 5),(7, 9, 13),(7, 1, 3),(9, 7, 1),(1, 7, 7);
```

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:



```
Cost: 6ms <  1  > Total 1
   name
   varchar
1  Peru
```

Q95.

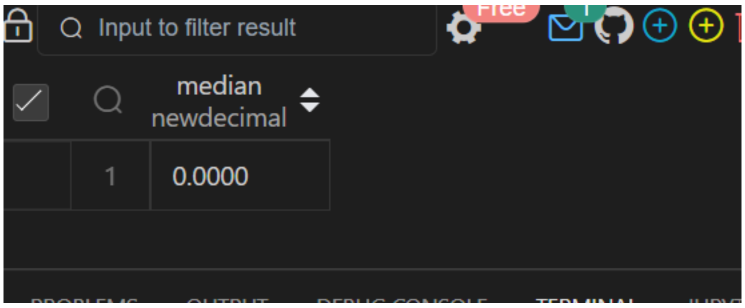
```
Create table If Not Exists Numbers (
       Number int,
       Frequency int);


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:



```
     median
     newdecimal
1    0.0000
```

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)
```

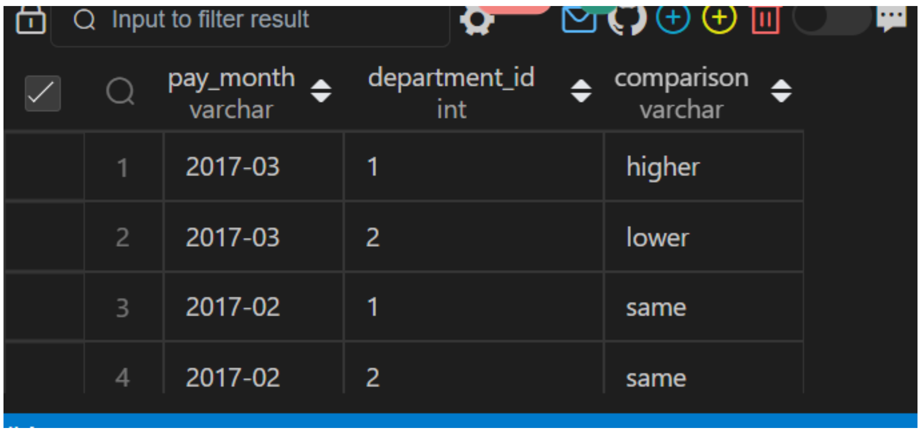
```
values
    ('6', '3', '8000', '2017/02/28');

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:



pay_month varchar	department_id int	comparison varchar
2017-03	1	higher
2017-03	2	lower
2017-02	1	same
2017-02	2	same

Q97.

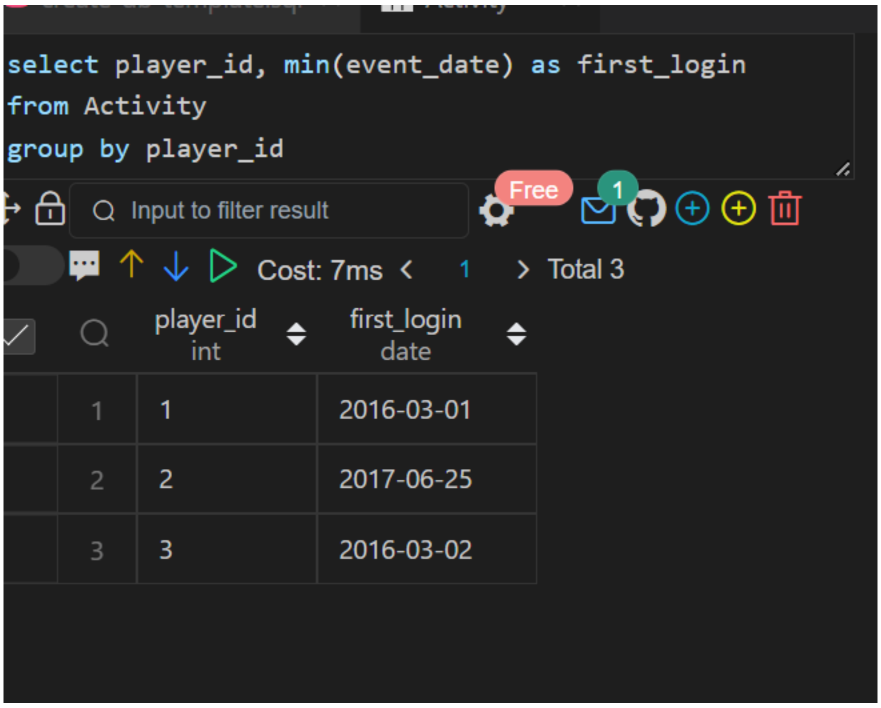
```
CREATE TABLE Activity(
player_id int,
device_id int,
event_date date,
games_played int
);

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:


```
select player_id, min(event_date) as first_login
from Activity
group by player_id
```

Input to filter result

Cost: 7ms < 1 > Total 3

	player_id int	first_login date
1	1	2016-03-01
2	2	2017-06-25
3	3	2016-03-02

Q98.

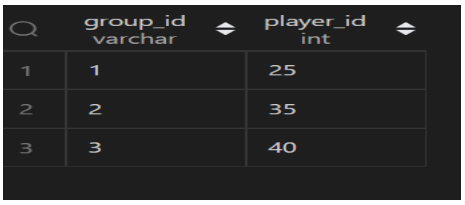
```
CREATE TABLE Players(
player_id int PRIMARY KEY,
group_id varchar(30)
);

 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:



	group_id varchar	player_id int
1	1	25
2	2	35
3	3	40

Q99)

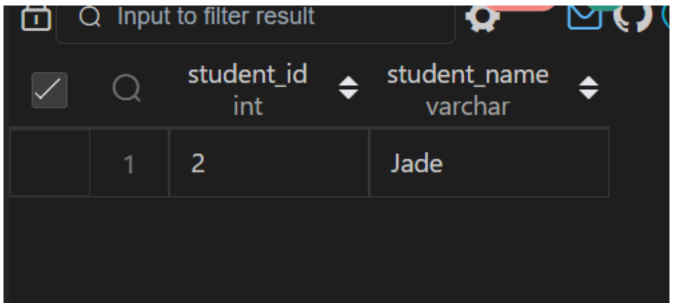
```sql
CREATE TABLE Student
(student_id INT,
 student_name VARCHAR(32));
INSERT INTO Student
VALUES
(1, 'Daniel'),
(2, 'Jade'),
(3, 'Stella'),
(4, 'Jonathan'),
(5, 'Will');

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:

```sql
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:



```
   student_id   student_name
   int          varchar
1  2            Jade
```

Q100)

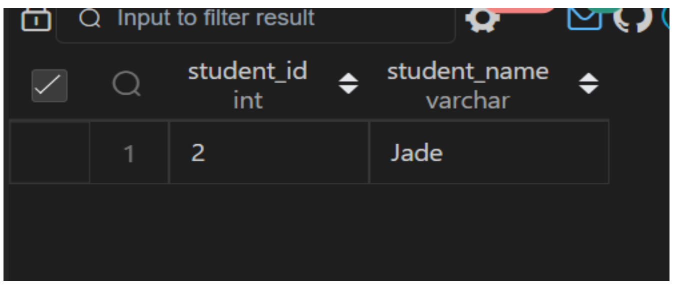
```
CREATE TABLE Student
(student_id INT,
 student_name VARCHAR(32));
INSERT INTO Student
VALUES
(1, 'Daniel'),
(2, 'Jade'),
(3, 'Stella'),
(4, 'Jonathan'),
(5, 'Will');

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:



	student_id int	student_name varchar
1	2	Jade

Q101)

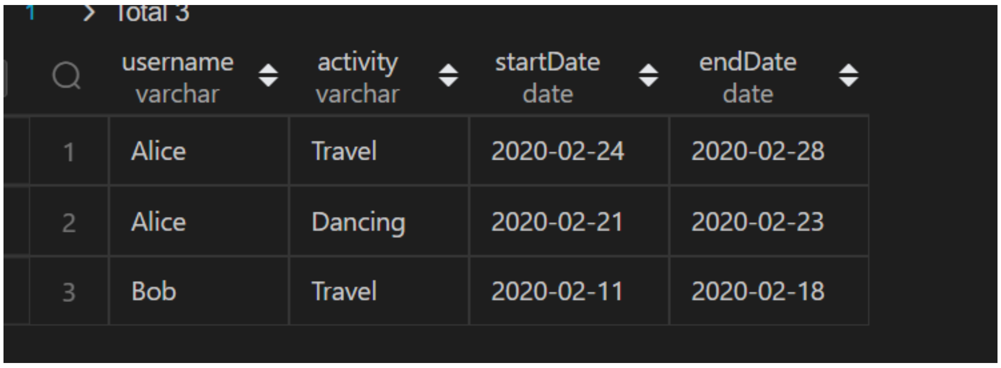
```
create table UserActivity(
username varchar(30),
activity varchar(30),
startDate Date,
endDate Date
);
insert into UserActivity VALUES('Alice','Travel', '2020-02-12','2020-02-20'),
('Alice','Dancing','2020-02-21','2020-02-23'),
('Alice','Travel','2020-02-24','2020-02-28'),
('Bob','Travel','2020-02-11','2020-02-18');
```

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:



```
Total 3
```

	username varchar	activity varchar	startDate date	endDate date
1	Alice	Travel	2020-02-24	2020-02-28
2	Alice	Dancing	2020-02-21	2020-02-23
3	Bob	Travel	2020-02-11	2020-02-18

Q102)

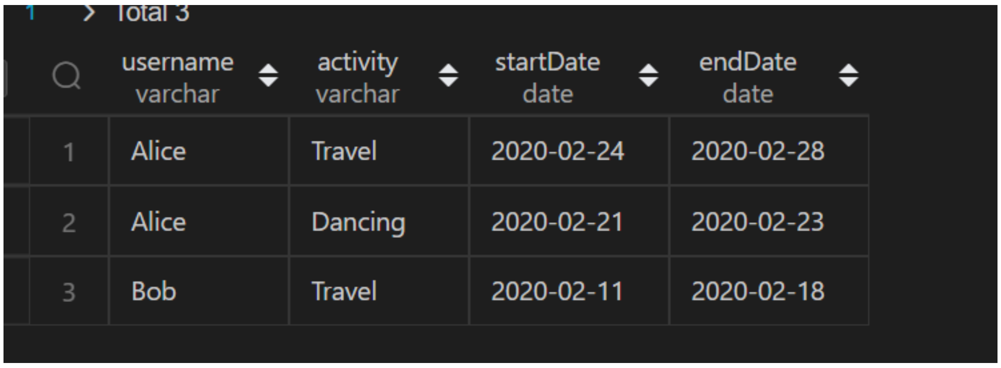
```
create table UserActivity(
username varchar(30),
activity varchar(30),
startDate Date,
endDate Date
);
insert into UserActivity VALUES('Alice','Travel', '2020-02-12','2020-02-20'),
('Alice','Dancing','2020-02-21','2020-02-23'),
('Alice','Travel','2020-02-24','2020-02-28'),
('Bob','Travel','2020-02-11','2020-02-18');
```

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:



```
1  > Total 3
   username   activity   startDate    endDate
   varchar    varchar    date         date
1  Alice      Travel     2020-02-24   2020-02-28
2  Alice      Dancing    2020-02-21   2020-02-23
3  Bob        Travel     2020-02-11   2020-02-18
```

Q103)

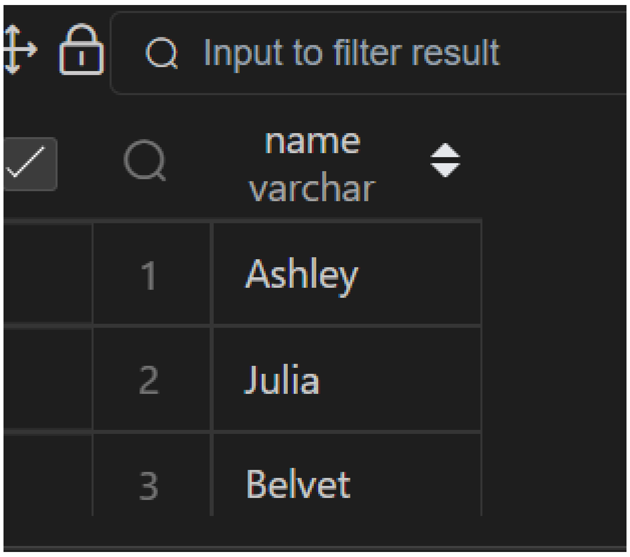
```
CREATE TABLE STUDENTS (
    ID INTEGER,
    NAME VARCHAR(30),
    MARKS VARCHAR(30)
);
insert into
STUDENTS  VALUES(1,'Ashley',81),(2,'Samantha',75),(4,'Julia',76),(3,'Belvet',8
4);
```

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:



```
name
varchar
1  Ashley
2  Julia
3  Belvet
```

104)

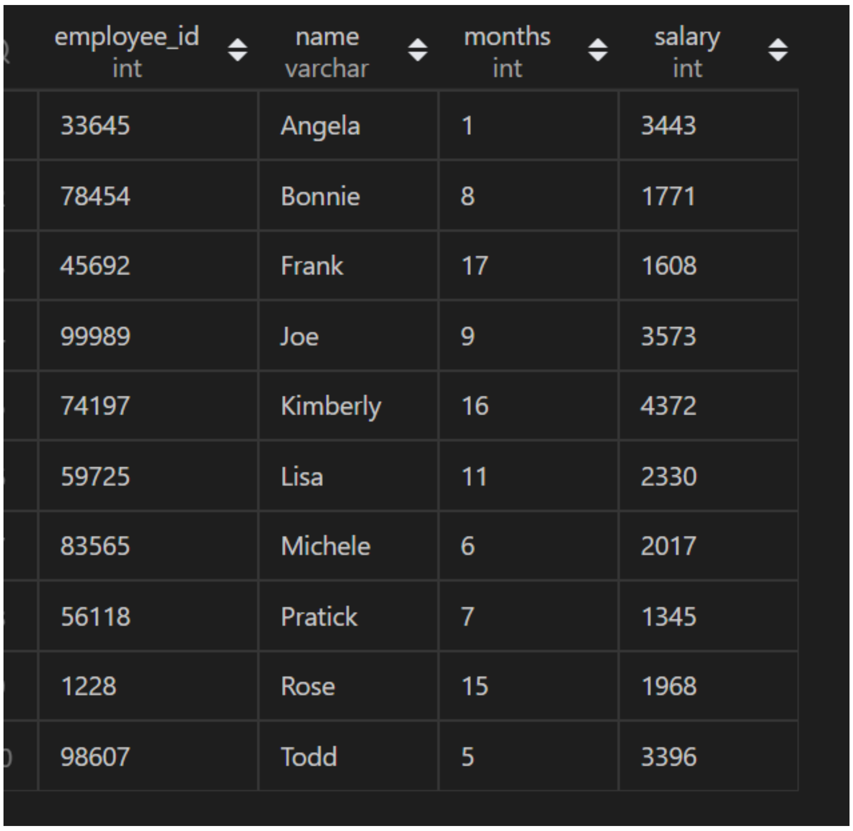
```
CREATE TABLE EMPLOYEE(
    employee_id INTEGER,
    name varchar(30),
    months INTEGER,
    salary INTEGER
);
insert into EMPLOYEE
VALUES(1228,'Rose',15,1968),(33645,'Angela',1,3443),(45692,'Frank',17,1608),
(56118,'Pratick',7,1345),(59725,'Lisa',11,2330),
(74197,'Kimberly',16,4372);
```

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:



employee_id int	name varchar	months int	salary int
33645	Angela	1	3443
78454	Bonnie	8	1771
45692	Frank	17	1608
99989	Joe	9	3573
74197	Kimberly	16	4372
59725	Lisa	11	2330
83565	Michele	6	2017
56118	Pratick	7	1345
1228	Rose	15	1968
98607	Todd	5	3396

Q105.

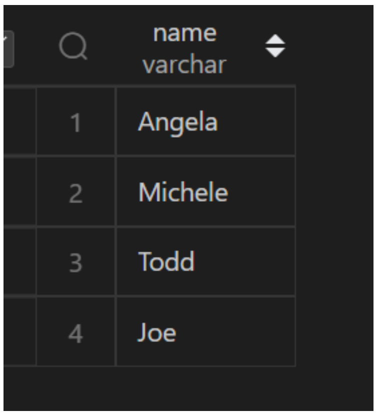
```
CREATE TABLE EMPLOYEE(
    employee_id INTEGER,
    name varchar(30),
    months INTEGER,
    salary INTEGER
);
insert into EMPLOYEE
VALUES(1228,'Rose',15,1968),(33645,'Angela',1,3443),(45692,'Frank',17,1608),
(56118,'Pratick',7,1345),(59725,'Lisa',11,2330),
(74197,'Kimberly',16,4372),(78454,'Bonnie',8,1771),
(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:



	name varchar
1	Angela
2	Michele
3	Todd
4	Joe

Q106)

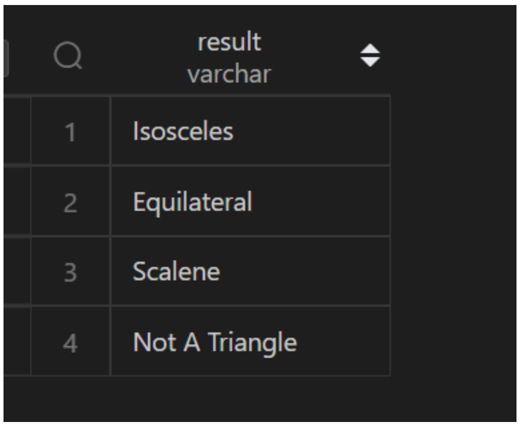
```
create table TRIANGLES(
    A integer,
    B integer,
    C integer
);
insert into TRIANGLES VALUES(20,20,23),(20,20,20),(20,21,22),(13,14,30);
```

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:



	result varchar
1	Isosceles
2	Equilateral
3	Scalene
4	Not A Triangle

Q107.

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

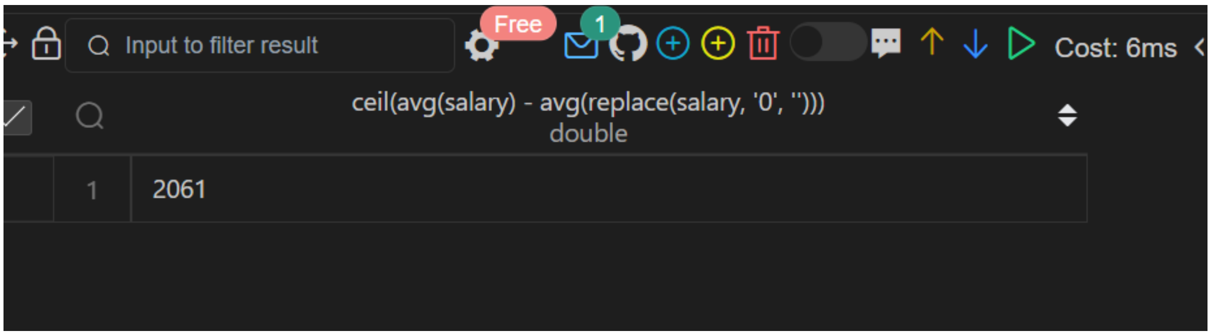
```
insert into EMployees
Values(1,'Kristeen',1420),(2,'Ashley',2006),(3,'Julia',2210),(4,'Maria',3000);
```

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:



```
ceil(avg(salary) - avg(replace(salary, '0', '')))
double
1    2061
```

Q108.

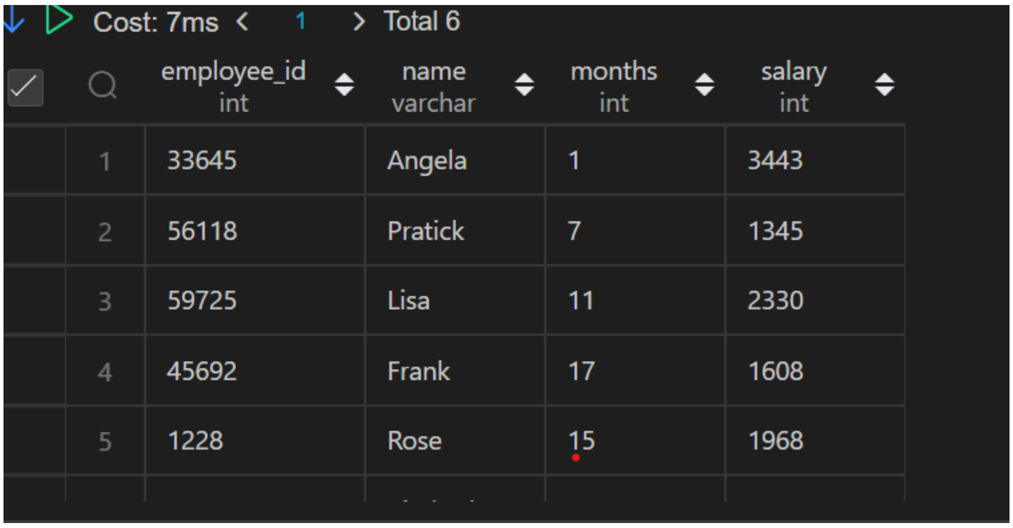
```
CREATE TABLE EMPLOYEE(
    employee_id INTEGER,
    name varchar(30),
    months INTEGER,
    salary INTEGER
);
insert into EMPLOYEE
VALUES(1228,'Rose',15,1968),(33645,'Angela',1,3443),(45692,'Frank',17,1608),
(56118,'Pratick',7,1345),(59725,'Lisa',11,2330),
(74197,'Kimberly',16,4372);
```

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:


Cost: 7ms < 1 > Total 6

	employee_id int	name varchar	months int	salary int
1	33645	Angela	1	3443
2	56118	Pratick	7	1345
3	59725	Lisa	11	2330
4	45692	Frank	17	1608
5	1228	Rose	15	1968

Q109)

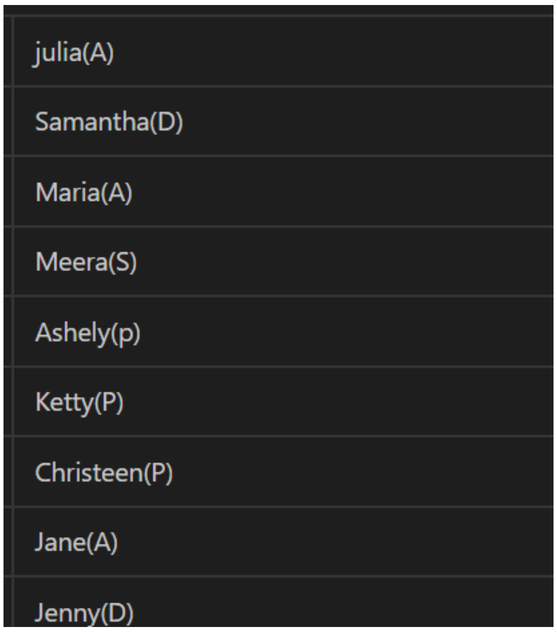
```
CREATE TABLE OCCUPATIONS(
    Name  VARCHAR(30),
    Occupation VARCHAR(30)
);
insert into OCCUPATIONS
values('julia','Actor'),('Samantha','Doctor'),('Maria','Actor'),('Meera','Sing
er'),('Ashely','professor'),
('Ketty','Professor'),('Christeen','Professor'),('Jane','Actor'),
('Jenny','Doctor'),('Priya','Singer');
```

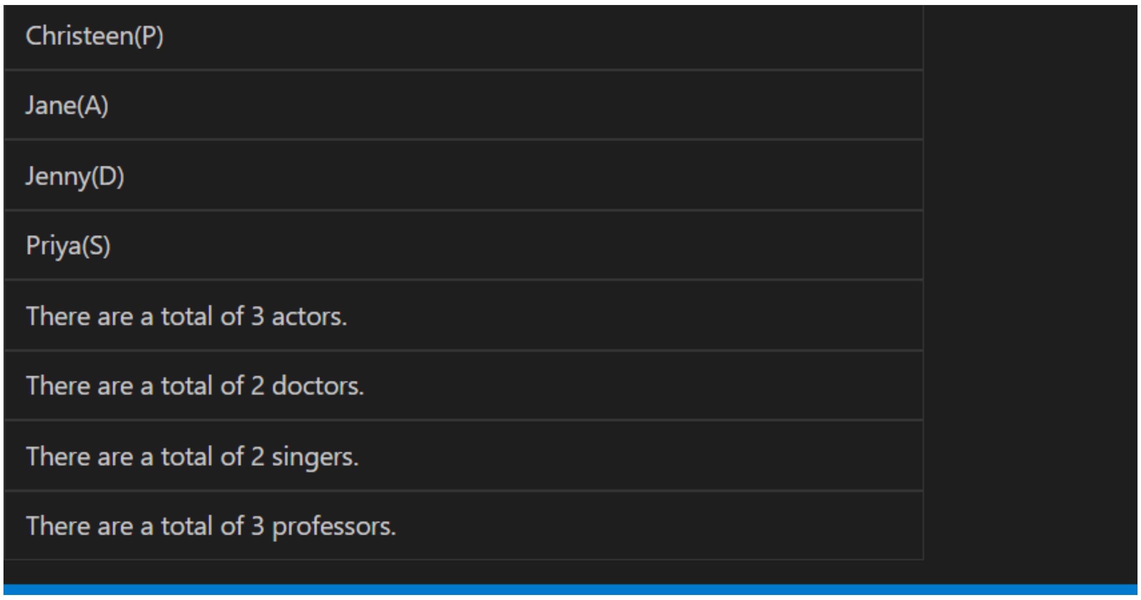
Query the number of occurrences of each occupation in OCCUPATIONS. Sort the occurrences in ascending order, and output

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:


```
julia(A)
Samantha(D)
Maria(A)
Meera(S)
Ashely(p)
Ketty(P)
Christeen(P)
Jane(A)
Jenny(D)
```


```
Christeen(P)
Jane(A)
Jenny(D)
Priya(S)
There are a total of 3 actors.
There are a total of 2 doctors.
There are a total of 2 singers.
There are a total of 3 professors.
```

Q110 .

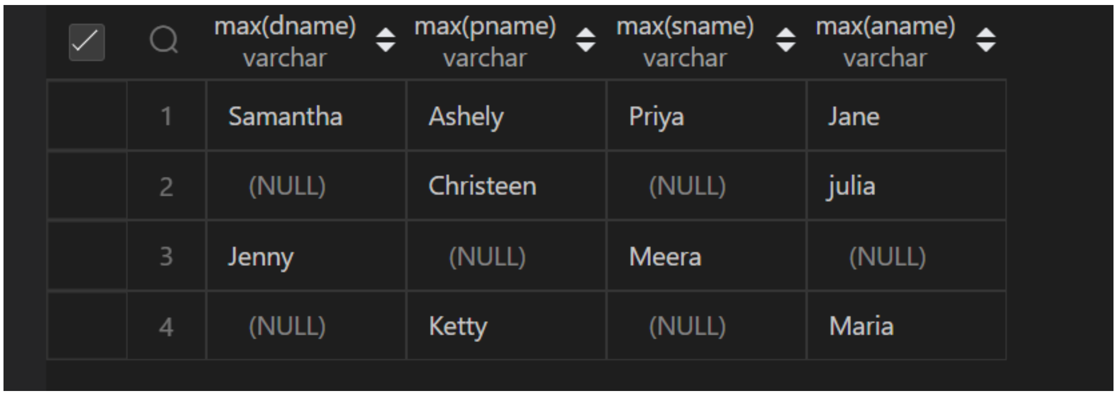
```
CREATE TABLE OCCUPATIONS(
    Name  VARCHAR(30),
    Occupation VARCHAR(30)
);
insert into OCCUPATIONS
values('julia','Actor'),('Samantha','Doctor'),('Maria','Actor'),('Meera','Sing
er'),('Ashely','professor'),
('Ketty','Professor'),('Christeen','Professor'),('Jane','Actor'),
('Jenny','Doctor'),('Priya','Singer');
```

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



	max(dname) varchar	max(pname) varchar	max(sname) varchar	max(aname) varchar
1	Samantha	Ashely	Priya	Jane
2	(NULL)	Christeen	(NULL)	julia
3	Jenny	(NULL)	Meera	(NULL)
4	(NULL)	Ketty	(NULL)	Maria

Q111.

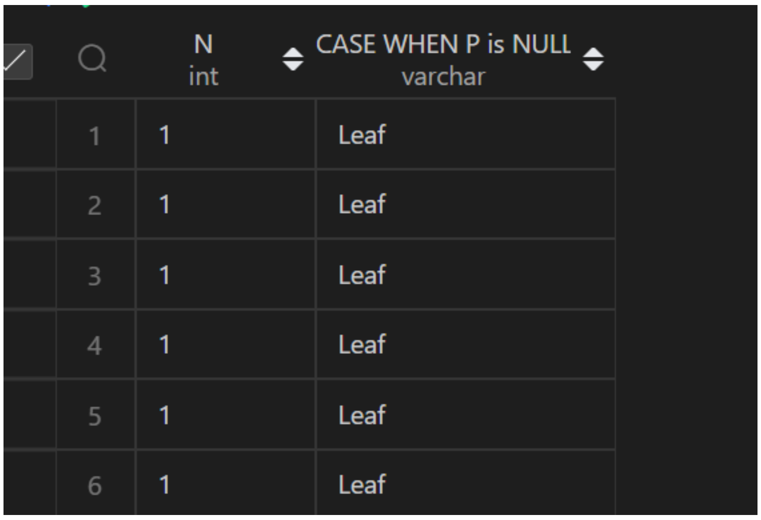
```
CREATE TABLE BST(
    N INT,
    P INT
);
INSERT INTO BST VALUES(1,2),(3,2),
(6,8),(9,8),(2,5),(8,5),(5,NULL);
```

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

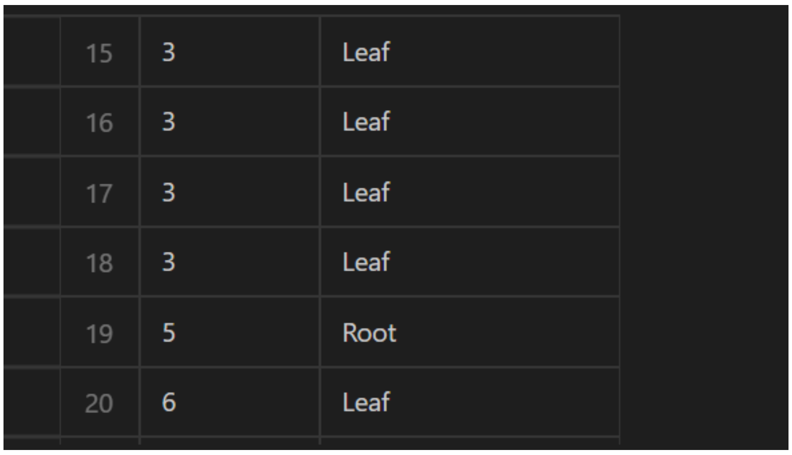
Query

```
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



	N int	CASE WHEN P is NULL varchar
1	1	Leaf
2	1	Leaf
3	1	Leaf
4	1	Leaf
5	1	Leaf
6	1	Leaf



15	3	Leaf
16	3	Leaf
17	3	Leaf
18	3	Leaf
19	5	Root
20	6	Leaf

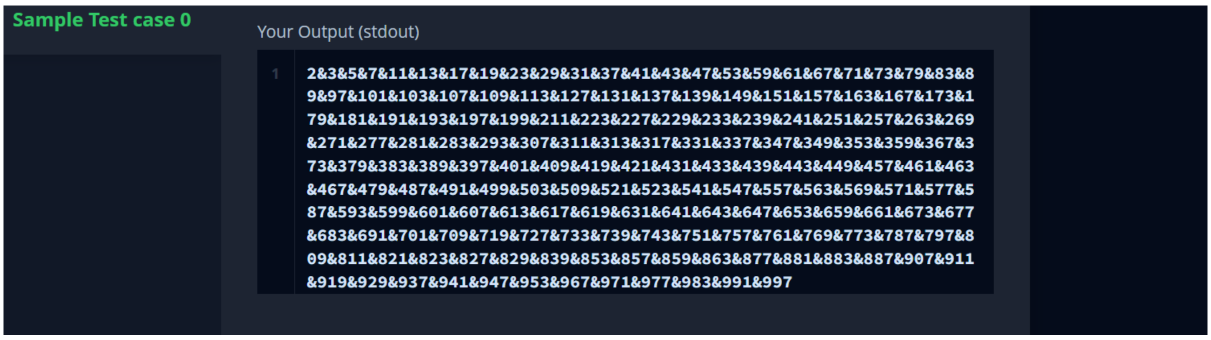
Q113.

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



```
2&3&5&7&11&13&17&19&23&29&31&37&41&43&47&53&59&61&67&71&73&79&83&8
9&97&101&103&107&109&113&127&131&137&139&149&151&157&163&167&173&1
79&181&191&193&197&199&211&223&227&229&233&239&241&251&257&263&269
&271&277&281&283&293&307&311&313&317&331&337&347&349&353&359&367&3
73&379&383&389&397&401&409&419&421&431&433&439&443&449&457&461&463
&467&479&487&491&499&503&509&521&523&541&547&557&563&569&571&577&5
87&593&599&601&607&613&617&619&631&641&643&647&653&659&661&673&677
&683&691&701&709&719&727&733&739&743&751&757&761&769&773&787&797&8
09&811&821&823&827&829&839&853&857&859&863&877&881&883&887&907&911
&919&929&937&941&947&953&967&971&977&983&991&997
```

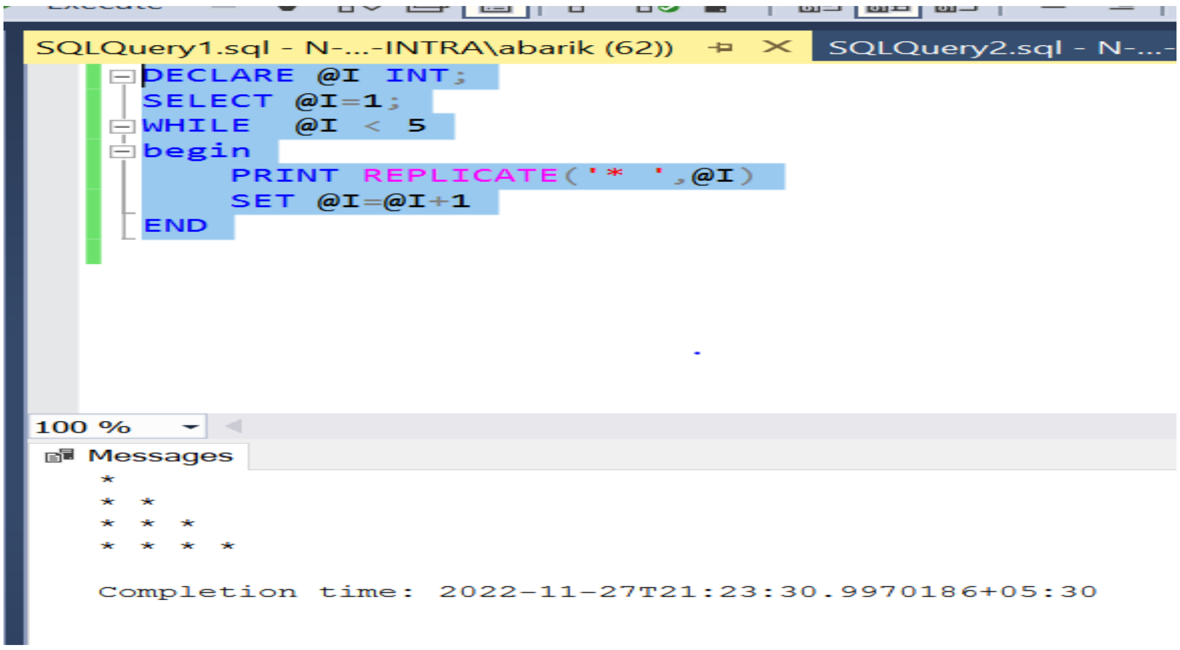
Q114.

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


SQLQuery1.sql - N-...-INTRA\abarik (62)) SQLQuery2.sql - N-...-

```
DECLARE @I INT;
SELECT @I=1;
WHILE  @I < 5
begin
      PRINT REPLICATE('* ',@I)
      SET @I=@I+1
END
```

```
*
*  *
*  *  *
*  *  *  *

Completion time: 2022-11-27T21:23:30.9970186+05:30
```

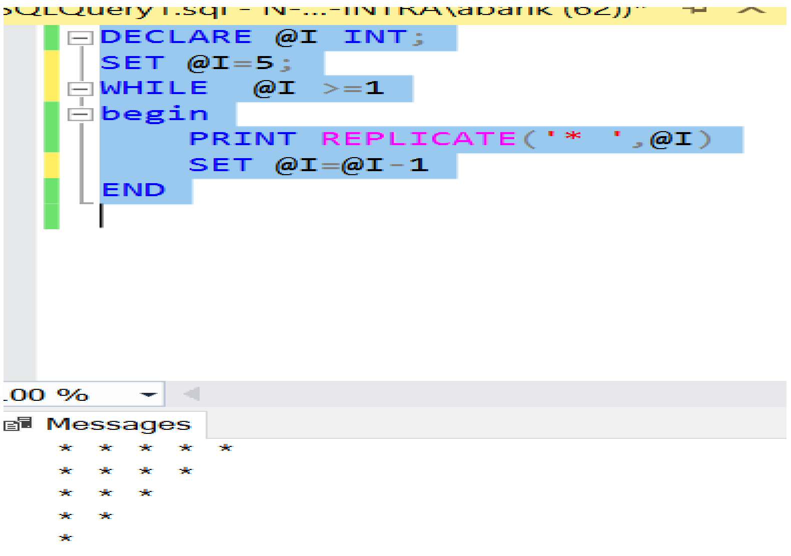
Q115.

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


SQLQuery1.sql - N-...-INTRA\abarik (62))

```
DECLARE @I INT;
SET @I=5;
WHILE  @I >=1
begin
      PRINT REPLICATE('* ',@I)
      SET @I=@I-1
END
```

```
*  *  *  *  *
*  *  *  *
*  *  *
*  *
*
```

Q116.

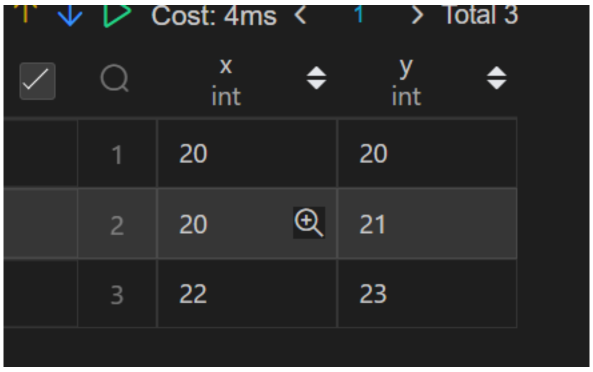
```
CREATE TABLE FUN(
   X INT,
   Y INT
   );
   INSERT INTO FUN VALUES(20,20),(20,20),
   (20,21),(23,22),(22,23),(21,20);
```

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



```
Cost: 4ms  <  1  >  Total 3
      x      y
      int    int
  1   20     20
  2   20     21
  3   22     23
```

Q117.

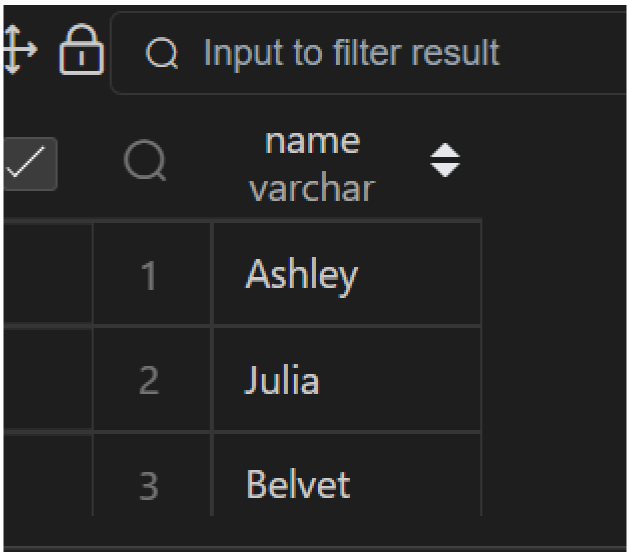
```
CREATE TABLE STUDENTS (
    ID INTEGER,
    NAME VARCHAR(30),
    MARKS VARCHAR(30)
);
insert into
STUDENTS  VALUES(1,'Ashley',81),(2,'Samantha',75),(4,'Julia',76),(3,'Belvet',8
4);
```

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:



```
name
varchar
1  Ashley
2  Julia
3  Belvet
```

Q118.

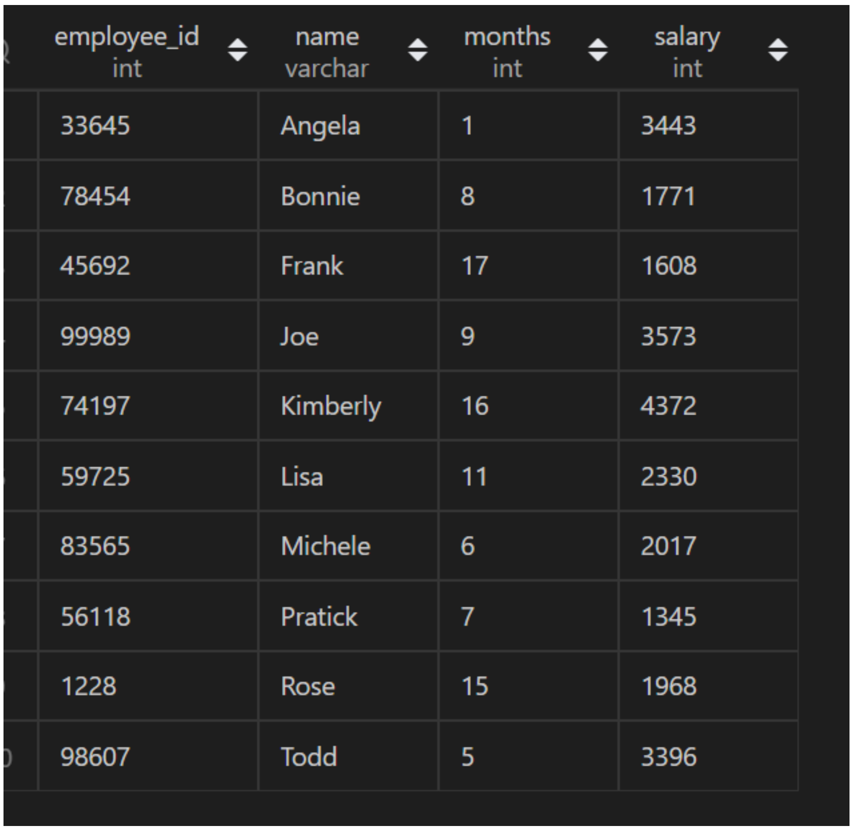
```
CREATE TABLE EMPLOYEE(
    employee_id INTEGER,
    name varchar(30),
    months INTEGER,
    salary INTEGER
);
insert into EMPLOYEE
VALUES(1228,'Rose',15,1968),(33645,'Angela',1,3443),(45692,'Frank',17,1608),
(56118,'Pratick',7,1345),(59725,'Lisa',11,2330),
(74197,'Kimberly',16,4372);
```

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:



employee_id int	name varchar	months int	salary int
33645	Angela	1	3443
78454	Bonnie	8	1771
45692	Frank	17	1608
99989	Joe	9	3573
74197	Kimberly	16	4372
59725	Lisa	11	2330
83565	Michele	6	2017
56118	Pratick	7	1345
1228	Rose	15	1968
98607	Todd	5	3396

Q119.

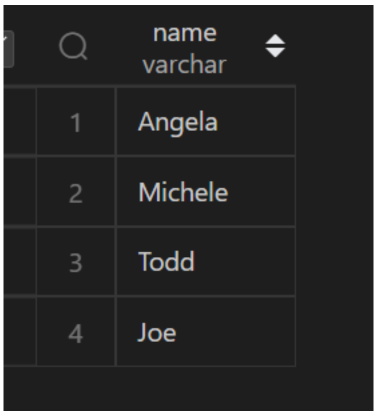
```
CREATE TABLE EMPLOYEE(
    employee_id INTEGER,
    name varchar(30),
    months INTEGER,
    salary INTEGER
);
insert into EMPLOYEE
VALUES(1228,'Rose',15,1968),(33645,'Angela',1,3443),(45692,'Frank',17,1608),
(56118,'Pratick',7,1345),(59725,'Lisa',11,2330),
(74197,'Kimberly',16,4372),(78454,'Bonnie',8,1771),
(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:



```
   name varchar
1  Angela
2  Michele
3  Todd
4  Joe
```

Q120

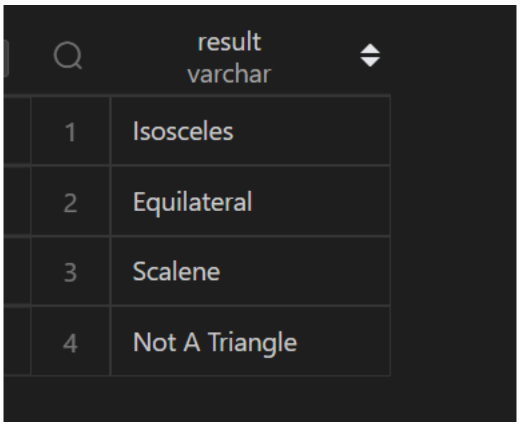
```
create table TRIANGLES(
    A integer,
    B integer,
    C integer
);
insert into TRIANGLES VALUES(20,20,23),(20,20,20),(20,21,22),(13,14,30);
```

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:



	result varchar
1	Isosceles
2	Equilateral
3	Scalene
4	Not A Triangle

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'));
```

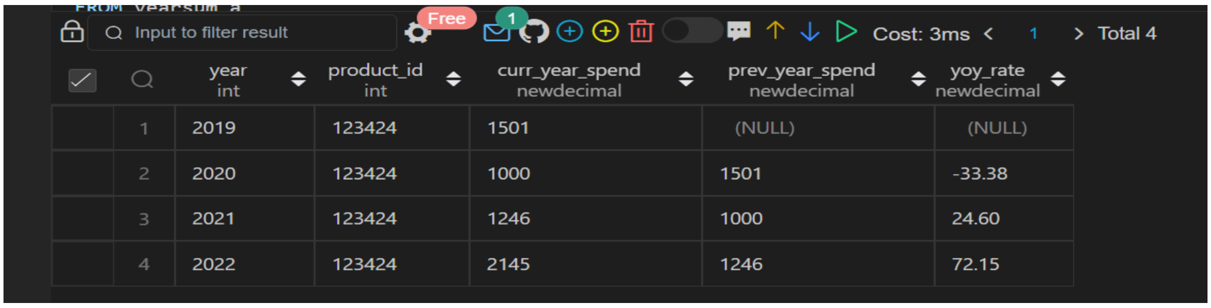
Write a query to obtain the year-on-year growth rate for the total spend of each product for each year.

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:



```
   year   product_id  curr_year_spend  prev_year_spend  yoy_rate
   int    int         newdecimal       newdecimal       newdecimal
1  2019   123424      1501             (NULL)           (NULL)
2  2020   123424      1000             1501             -33.38
3  2021   123424      1246             1000             24.60
4  2022   123424      2145             1246             72.15
```

Q122)

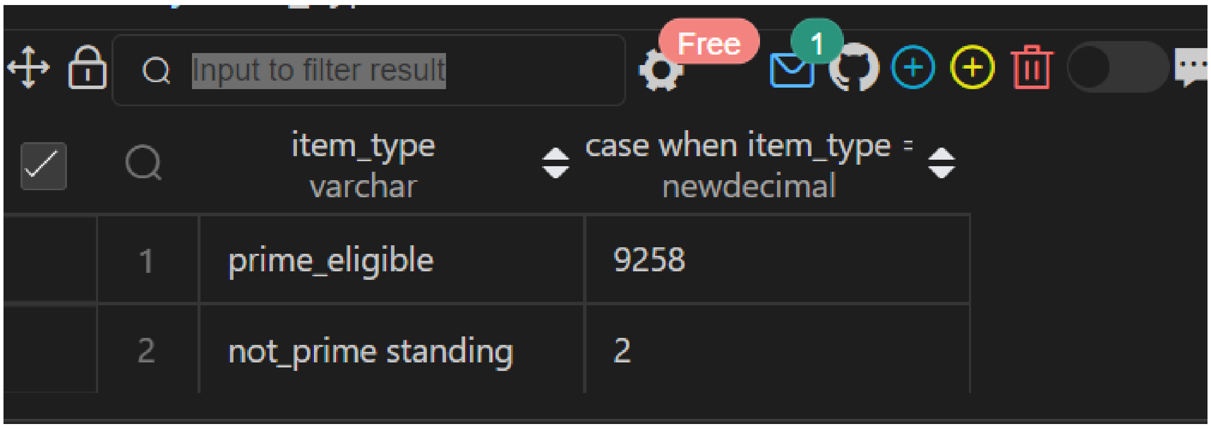
```
 create table inventory(
item_id integer,
item_type varchar(30),
item_category varchar(30),
square_footage decimal
 );

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:



	item_type varchar	case when item_type = newdecimal
1	prime_eligible	9258
2	not_prime standing	2

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'));
```

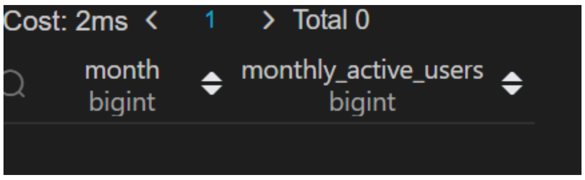
Assume you have the table below containing information on Facebook user actions. Write a query to obtain the active user retention in July 2022. Output the month (in numerical format 1, 2, 3) and the number of monthly active users (MAUs).

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
```

```sql
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:



```
Cost: 2ms  <  1  >  Total 0
month        monthly_active_users
bigint       bigint
```

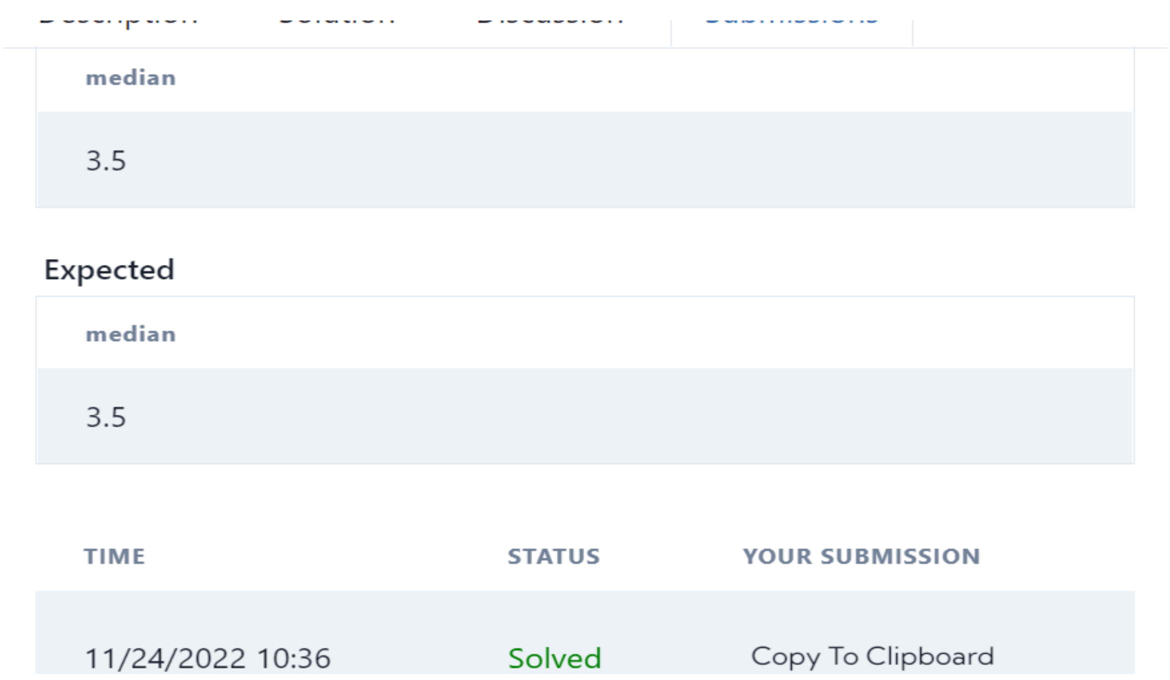
Q124)

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

```sql
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



median
3.5

Expected

median
3.5

TIME	STATUS	YOUR SUBMISSION
11/24/2022 10:36	Solved	Copy To Clipboard

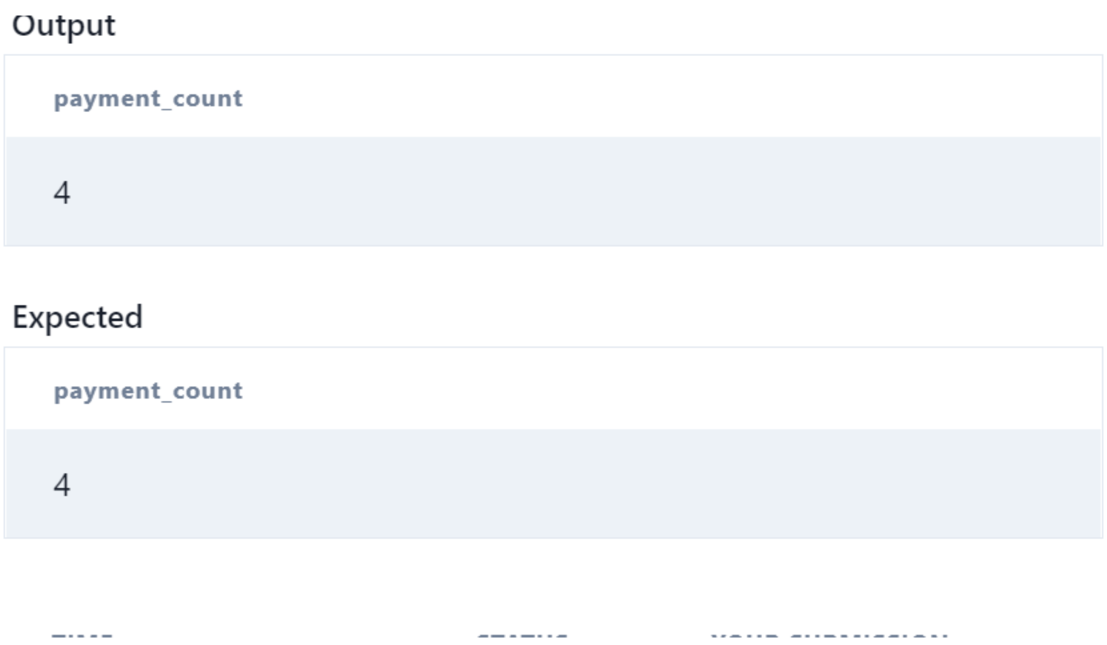
Q127)

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.transactio
n_timestamp)
;;
```

OUTPUT


Output

payment_count
4

Expected

payment_count
4

Q129)

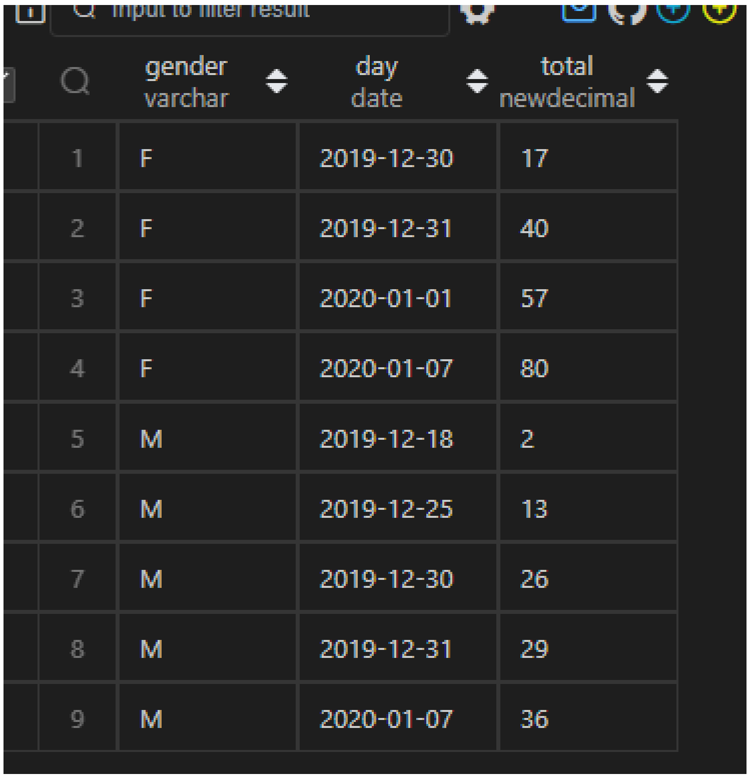
```sql
CREATE TABLE Scores(
player_name varchar(30),
gender varchar(30),
day date,
score_points int,
constraint pk PRIMARY KEY((gender, day)
);
insert into Scores values('Aron','F','2020-01-01', 17),
('Alice','F','2020-01-07',23),
('Bajrang','M','2020-01-07',7),
('Khali','M','2019-12-25',11),
('Slaman','M','2019-12-30', 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:

```sql
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:



	gender varchar	day date	total newdecimal
1	F	2019-12-30	17
2	F	2019-12-31	40
3	F	2020-01-01	57
4	F	2020-01-07	80
5	M	2019-12-18	2
6	M	2019-12-25	13
7	M	2019-12-30	26
8	M	2019-12-31	29
9	M	2020-01-07	36

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');
```

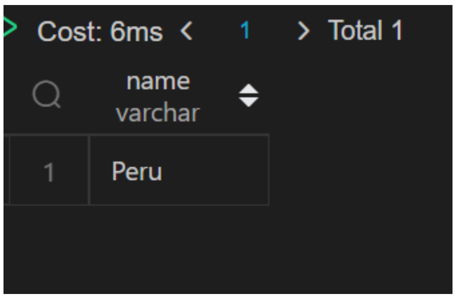
```
insert into calls values (1, 9, 33),(2, 9, 4),(1, 2, 59),(3, 12, 102),(3, 12,
330),(12, 3, 5),(7, 9, 13),(7, 1, 3),(9, 7, 1),(1, 7, 7);
```

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:



```
Cost: 6ms < 1 > Total 1
   name
   varchar
1  Peru
```

Q131.

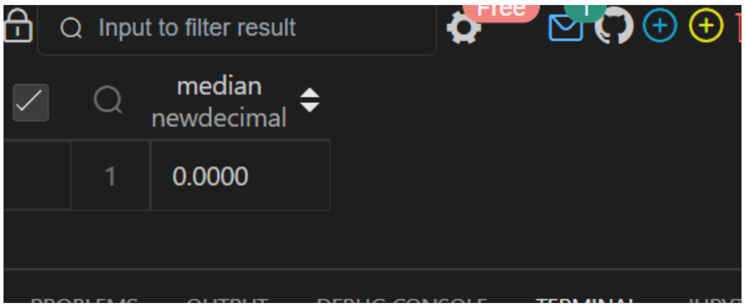
```
Create table If Not Exists Numbers (
       Number int,
       Frequency int);


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:



```
   median
   newdecimal
1  0.0000
```

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
```

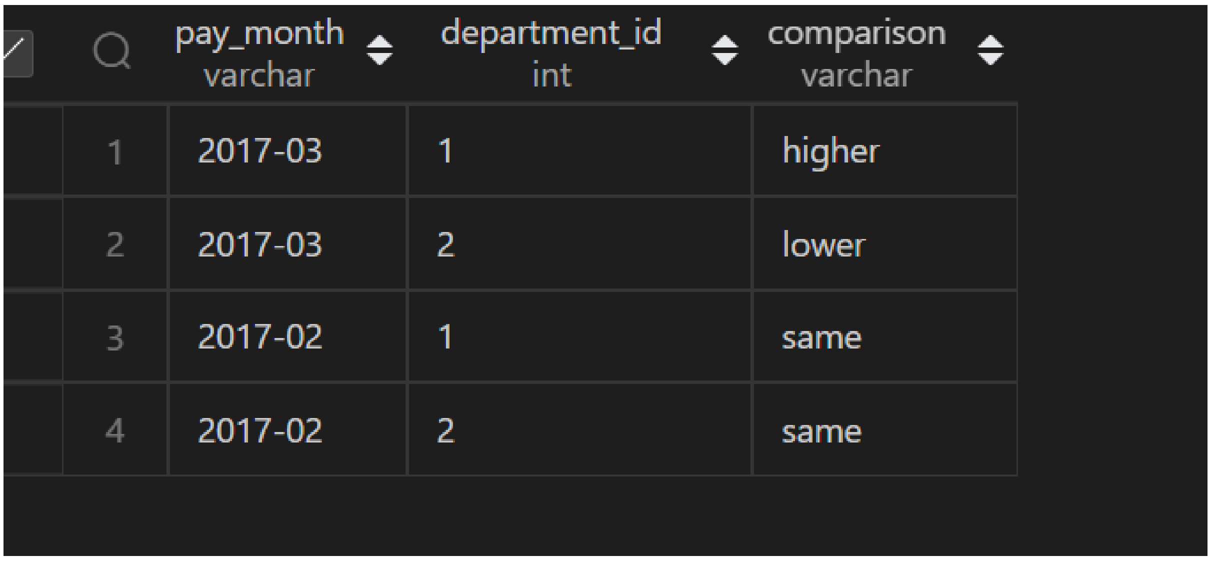
```
    ('5', '2', '6000', '2017/02/28');
insert into salary
    (id, employee_id, amount, pay_date)
values
    ('6', '3', '8000', '2017/02/28');


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:



	pay_month varchar	department_id int	comparison varchar
1	2017-03	1	higher
2	2017-03	2	lower
3	2017-02	1	same
4	2017-02	2	same

Q133.

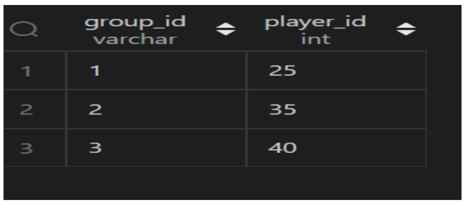
```
CREATE TABLE Players(
player_id int PRIMARY KEY,
group_id varchar(30)
);

 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:



	group_id varchar	player_id int
1	1	25
2	2	35
3	3	40

Q136.

```
CREATE TABLE Student
(student_id INT,
 student_name VARCHAR(32));
INSERT INTO Student
```

```sql
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);
```

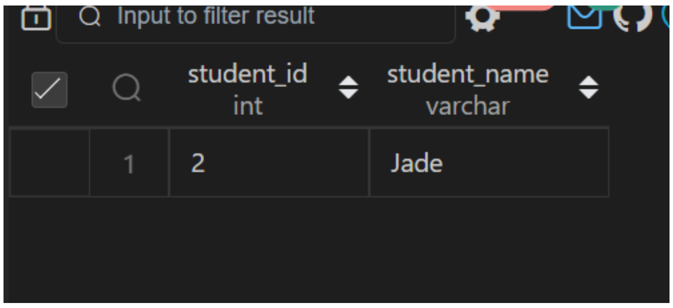
A quiet student is the one who took at least one exam and did not score the high or the low score. Write an SQL query to report the students (student_id, student_name) being quiet in all exams. Do not return the student who has never taken any exam

Query:

```sql
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:



```
     student_id   student_name
     int          varchar
1    2            Jade
```

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.

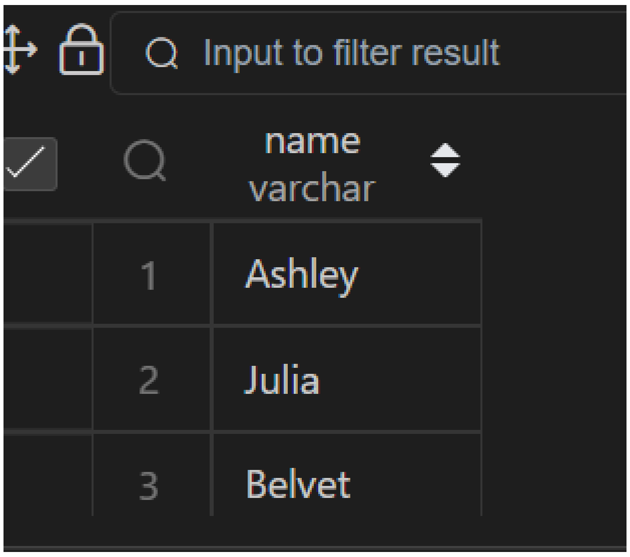
```
CREATE TABLE STUDENTS (
    ID INTEGER,
    NAME VARCHAR(30),
    MARKS VARCHAR(30)
);
insert into
STUDENTS  VALUES(1,'Ashley',81),(2,'Samantha',75),(4,'Julia',76),(3,'Belvet',8
4);
```

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:



```
1  Ashley
2  Julia
3  Belvet
```

Q145

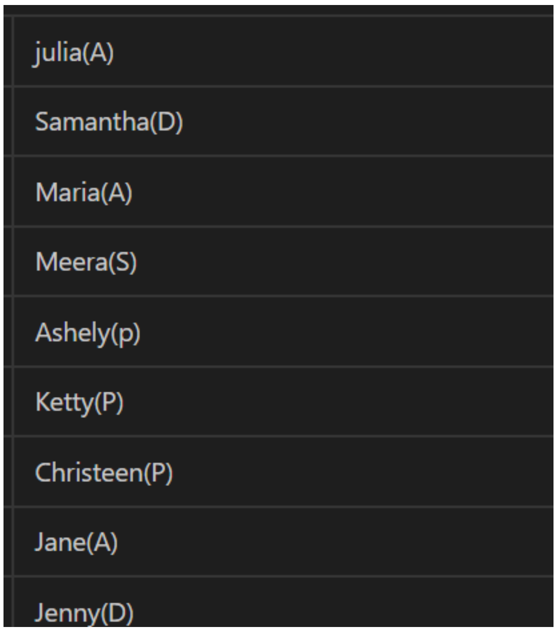
```
CREATE TABLE OCCUPATIONS(
    Name  VARCHAR(30),
    Occupation VARCHAR(30)
);
insert into OCCUPATIONS
values('julia','Actor'),('Samantha','Doctor'),('Maria','Actor'),('Meera','Sing
er'),('Ashely','professor'),
('Ketty','Professor'),('Christeen','Professor'),('Jane','Actor'),
('Jenny','Doctor'),('Priya','Singer');
```

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

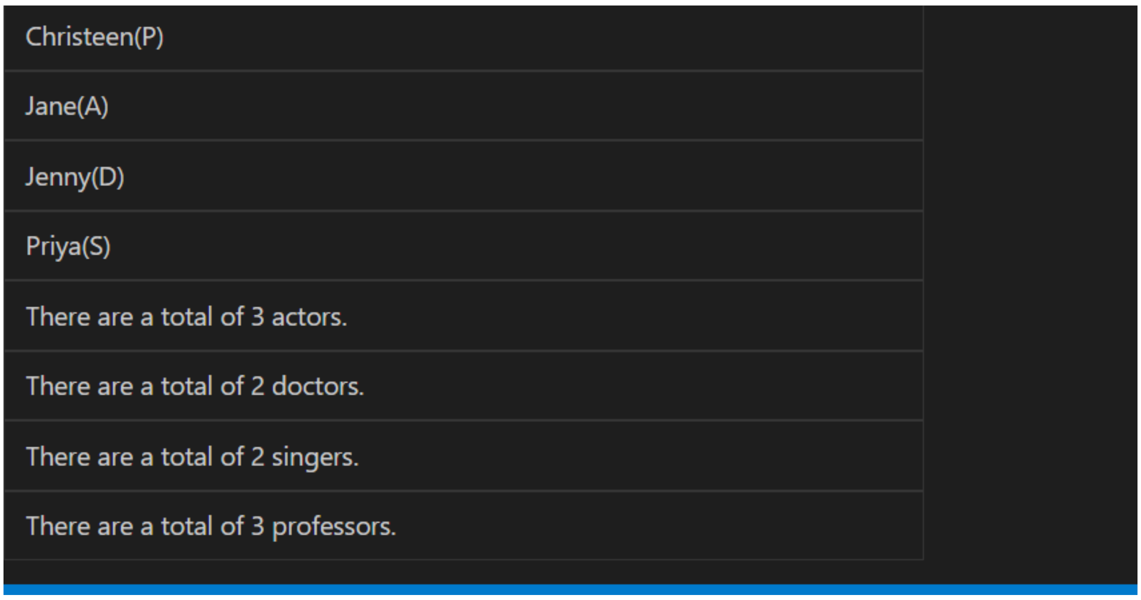
Query:

```
(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:



```
julia(A)
Samantha(D)
Maria(A)
Meera(S)
Ashely(p)
Ketty(P)
Christeen(P)
Jane(A)
Jenny(D)
```



```
Christeen(P)
Jane(A)
Jenny(D)
Priya(S)
There are a total of 3 actors.
There are a total of 2 doctors.
There are a total of 2 singers.
There are a total of 3 professors.
```

Q146.

```
CREATE TABLE OCCUPATIONS(
    Name  VARCHAR(30),
    Occupation VARCHAR(30)
);
```

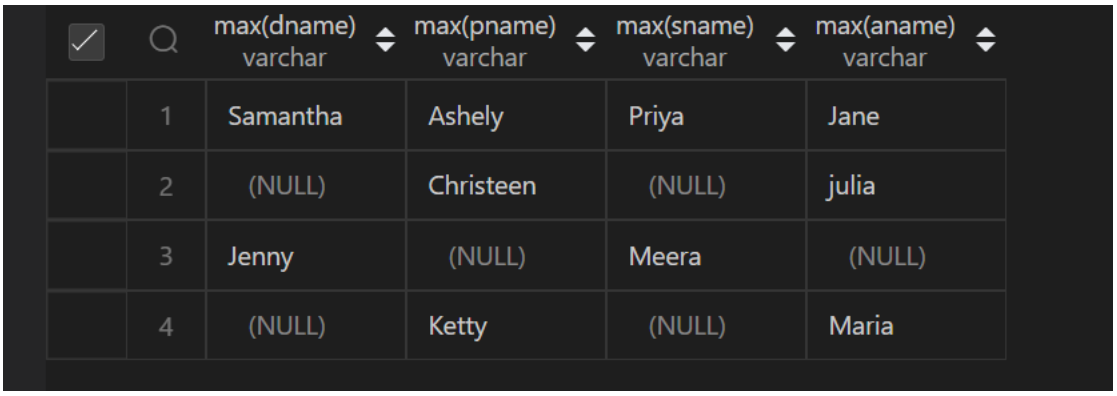
```
insert into OCCUPATIONS
values('julia','Actor'),('Samantha','Doctor'),('Maria','Actor'),('Meera','Sing
er'),('Ashely','professor'),
('Ketty','Professor'),('Christeen','Professor'),('Jane','Actor'),
('Jenny','Doctor'),('Priya','Singer');
```

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



	max(dname) varchar	max(pname) varchar	max(sname) varchar	max(aname) varchar
1	Samantha	Ashely	Priya	Jane
2	(NULL)	Christeen	(NULL)	julia
3	Jenny	(NULL)	Meera	(NULL)
4	(NULL)	Ketty	(NULL)	Maria

Q147.

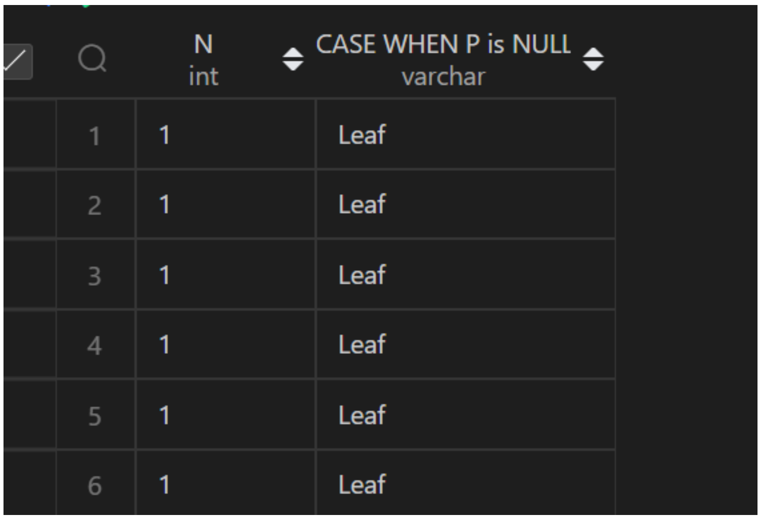
```
CREATE TABLE BST(
    N INT,
    P INT
);
INSERT INTO BST VALUES(1,2),(3,2),
(6,8),(9,8),(2,5),(8,5),(5,NULL);
```

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

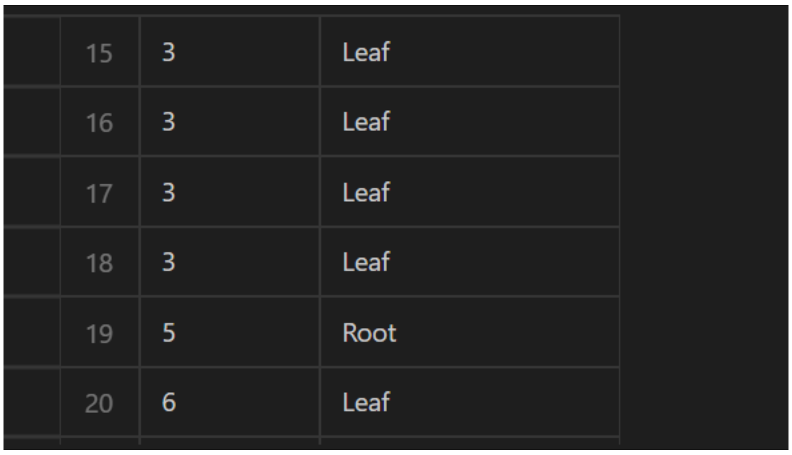
Query

```
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



```
     N      CASE WHEN P is NULL
     int    varchar
1    1      Leaf
2    1      Leaf
3    1      Leaf
4    1      Leaf
5    1      Leaf
6    1      Leaf
```



```
15   3      Leaf
16   3      Leaf
17   3      Leaf
18   3      Leaf
19   5      Root
20   6      Leaf
```

Q149.

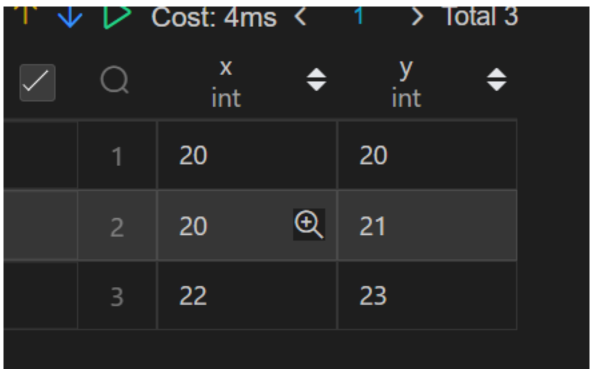
```
CREATE TABLE FUN(
   X INT,
   Y INT
   );
   INSERT INTO FUN VALUES(20,20),(20,20),
   (20,21),(23,22),(22,23),(21,20);
```

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



```
Cost: 4ms  <  1  >  Total 3

     x     y
     int   int
1    20    20
2    20    21
3    22    23
```