Q51.

Column Name	Type
name	varchar
continent	varchar
area	int
population	int
gdp	int

name is the primary key column for this table.
Each row of this table gives information about the name of a country, the continent to which it belongs, its area, the population, and its GDP value.
A country is big if:

- it has an area of at least three million (i.e., 3000000 km2), or
- it has a population of at least twenty-five million (i.e., 25000000).

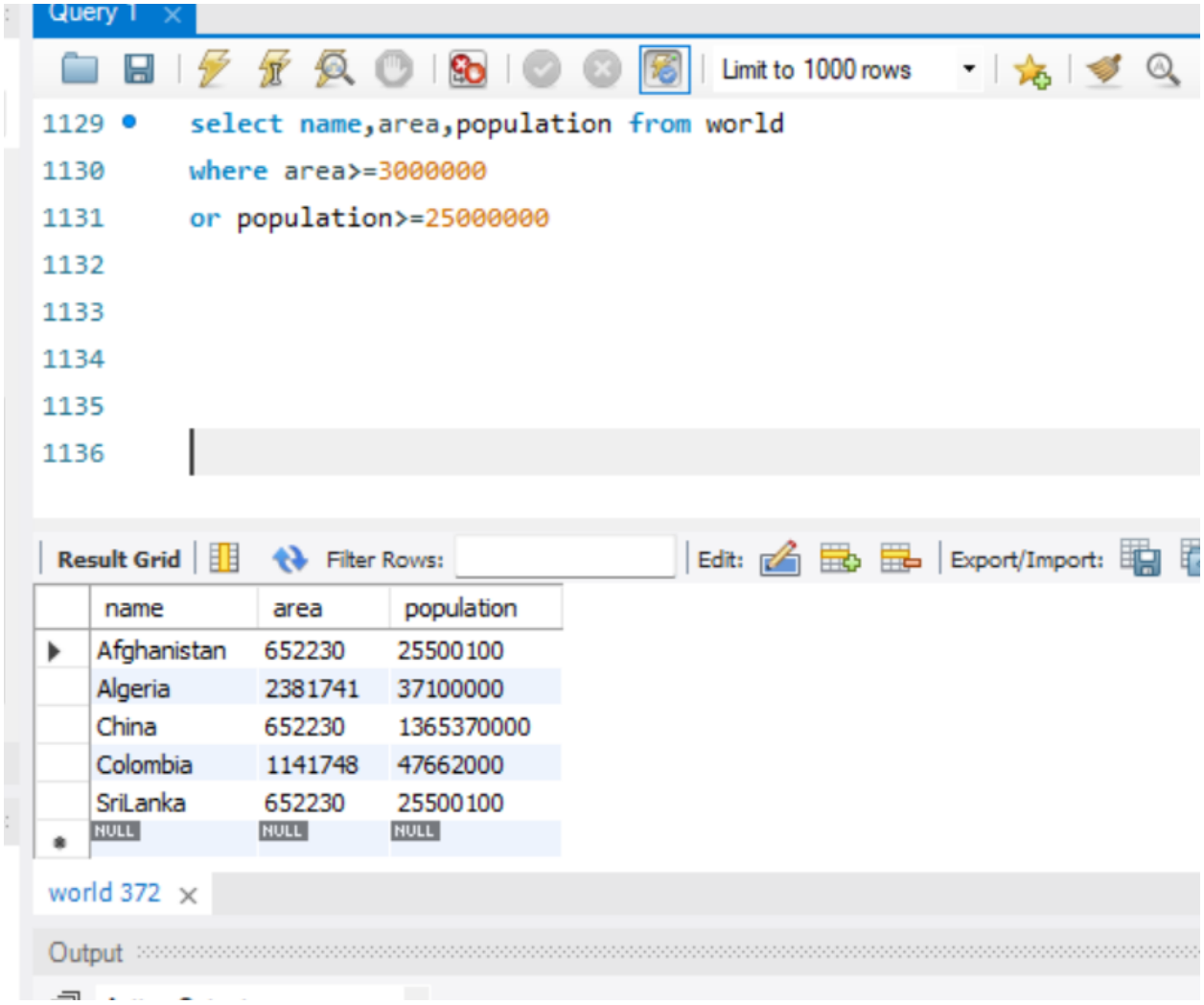
Write an SQL query to report the name, population, and area of the big countries.
Return the result table in any order.
The query result format is in the following example.

Input:
World table:

name	continent	area	population	gdp
Afghanistan	Asia	652230	25500100	20343000000
Albania	Europe	28748	2831741	12960000000
Algeria	Africa	2381741	37100000	188681000000
Andorra	Europe	468	78115	3712000000
Angola	Africa	1246700	20609294	100990000000

Output:

name	population	area
Afghanistan	25500100	652230
Algeria	37100000	2381741


Query 1
Limit to 1000 rows
1129 select name,area,population from world
1130 where area>=3000000
1131 or population>=25000000
1132
1133
1134
1135
1136
Result Grid
Filter Rows:
Edit:
Export/Import:
name area population
Afghanistan 652230 25500100
Algeria 2381741 37100000
China 652230 1365370000
Colombia 1141748 47662000
SriLanka 652230 25500100
NULL NULL NULL
world 372
Output

Q52.

Table: Customer

Column Name	Type
id	int
name	varchar
referee_id	int

id is the primary key column for this table.
Each row of this table indicates the id of a customer, their name, and the id of the customer who referred them.

Write an SQL query to report the names of the customer that are not referred by the customer with id = 2.
Return the result table in any order.
The query result format is in the following example.

Input:
Customer table:

id	name	referee_id
1	Will	null
2	Jane	null
3	Alex	2
4	Bill	null
5	Zack	1
6	Mark	2

Output:

name
Will
Jane
Bill
Zack

```
1153    select name from customer
1154    where coalesce(refree_id,0)!=2
1155
1156
1157
1158
```

Result Grid | Filter Rows: | Export:

name
Will
Jane
Bill
Zack

Q53.

Table: Customers

Column Name	Type
id	int
name	varchar

id is the primary key column for this table.
Each row of this table indicates the ID and name of a customer.

Table: Orders

Column Name	Type
id	int
customerId	int

id is the primary key column for this table.
customerId is a foreign key of the ID from the Customers table.
Each row of this table indicates the ID of an order and the ID of the customer who ordered it.

Write an SQL query to report all customers who never order anything.
Return the result table in any order.
The query result format is in the following example.

Input:
Customers table:

id	name
1	Joe
2	Henry
3	Sam
4	Max

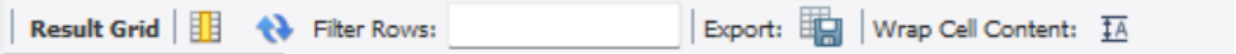
Orders table:

id	customerId
1	3
2	1

Output:

Customers
Henry
Max

```
1181 •  select distinct name as customer_name from customers
1182    left join orders on orders.customer_id=customers.id
1183    where orders.id is null
1184
1185
1186
1187
```



customer_name
HENRY
MAX

Q54.

Table: Employee

Column Name	Type
employee_id	int
team_id	int

employee_id is the primary key for this table.
Each row of this table contains the ID of each employee and their respective team.

Write an SQL query to find the team size of each of the employees.
Return result table in any order.
The query result format is in the following example.

Input:
Employee Table:

employee_id	team_id
1	8
2	8
3	8
4	7
5	9
6	9

Output:

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

Explanation:
Employees with Id 1,2,3 are part of a team with team_id = 8.
Employee with Id 4 is part of a team with team_id = 7.
Employees with Id 5,6 are part of a team with team_id = 9.

```
1202
1203 •  select employee_id,count(team_id) over(partition by team_id) as team_size
1204    from employee
1205    group by 1
1206    order by 1
1207
1208
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

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

Q55

Table Person:

Column Name	Type
id	int
name	varchar
phone_number	varchar

id is the primary key for this table.
Each row of this table contains the name of a person and their phone number.
Phone number will be in the form 'xxx-yyyyyyy' where xxx is the country code (3 characters) and yyyyyyy is the phone number (7 characters) where x and y are digits. Both can contain leading zeros.

Table Country:

Column Name	Type
name	varchar
country_code	varchar

country_code is the primary key for this table.
Each row of this table contains the country name and its code. country_code will be in the form 'xxx' where x is digits.

Table Calls:

Column Name	Type
caller_id	int
callee_id	int
duration	int

There is no primary key for this table, it may contain duplicates.
Each row of this table contains the caller id, caller id and the duration of the call in minutes. caller_id != callee_id

A telecommunications company wants to invest in new countries. The company intends to invest in the countries where the average call duration of the calls in this country is strictly greater than the global average call duration.
Write an SQL query to find the countries where this company can invest.
Return the result table in any order.
The query result format is in the following example.

Input:
Person table:

id	name	phone_number
3	Jonathan	051-1234567
12	Elvis	051-7654321
1	Moncef	212-1234567
2	Maroua	212-6523651
7	Meir	972-1234567
9	Rachel	972-0011100

Country table:

name	country_code
Peru	51
Israel	972
Morocco	212
Germany	49
Ethiopia	251

Calls table:

caller_id	callee_id	duration
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

Output:

country
Peru

```sql
with country_cust as (
select id,person.name as person,country.name as country_name
from person
left join country on
lpad(country.country_code,'3',0)=cast(substring_index(phone_number,'-',1)
as char)
)

select * from (
select country_name,avg(duration) as avg_call_duration
from calls
join country_cust on country_cust.id=callee_id or country_cust.id=caller_id
group by 1
)b
where avg_call_duration>(select avg(duration) from calls)
```

Q56.

Table: Activity

Column Name	Type
player_id	int
device_id	int
event_date	date
games_played	int

(player_id, event_date) is the primary key of this table.
This table shows the activity of players of some games.
Each row is a record of a player who logged in and played a number of games (possibly 0) before logging out on someday using some device.

Write an SQL query to report the device that is first logged in for each player.
Return the result table in any order.
The query result format is in the following example.

Input:
Activity table:

player_id	device_id	event_date	games_played
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

Output:

player_id	device_id
1	2
2	3
3	1

```
1315
1316  select player_id,device_id from (
1317  select *,row_number() over(partition by player_id order by event_date asc) as ranking
1318  from activity
1319  )b
1320  where ranking=1
1321
1322
1323
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

player_id	device_id
1	2
2	3
3	1

Q57.

Table: Orders

Column Name	Type
order_number	int
customer_number	int

order_number is the primary key for this table.
This table contains information about the order ID and the customer ID.

Write an SQL query to find the customer_number for the customer who has placed the largest number of orders.
The test cases are generated so that exactly one customer will have placed more orders than any other customer.
The query result format is in the following example.

Input:
Orders table:

order_number	customer_numbe
1	1
2	2
3	3
4	3

Output:

customer_number
3

Explanation:
The customer with number 3 has two orders, which is greater than either customer 1 or 2 because each of them only has one order.
So the result is customer_number 3.

Follow up: What if more than one customer has the largest number of orders, can you find all the customer_number in this case?

```
1340  select customer_number,count(order_number) as total_order_placed from orders
1341  group by 1
1342  order by 2 desc
1343  limit 1
1344
1345
1346
1347
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

customer_number	total_order_placed
3	2

Q58.

Table: Cinema

Column Name	Type
seat_id	int
free	bool

seat_id is an auto-increment primary key column for this table.
Each row of this table indicates whether the ith seat is free or not. 1 means free while 0 means occupied.

Write an SQL query to report all the consecutive available seats in the cinema.
Return the result table ordered by seat_id in ascending order.
The test cases are generated so that more than two seats are consecutively available.
The query result format is in the following example.

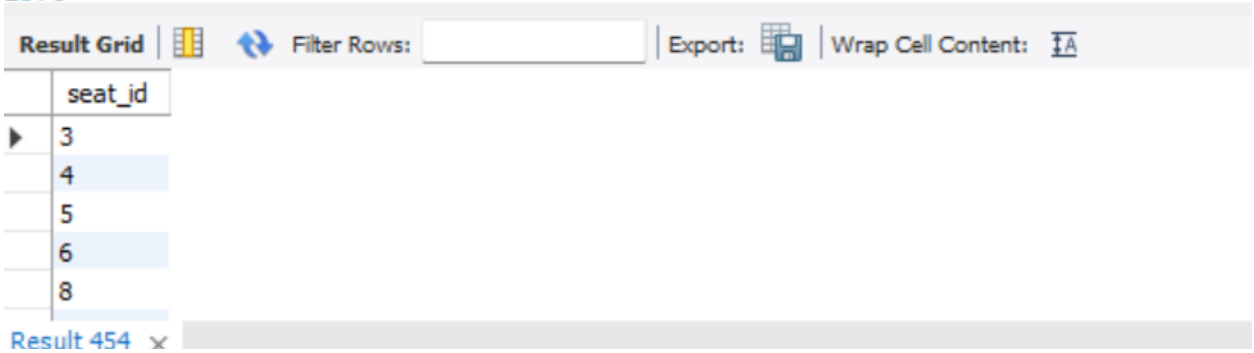
Input:
Cinema table:

seat_id	free
1	1
2	0
3	1
4	1
5	1

Output:

seat_id
3
4
5

```
1368
1369    SELECT DISTINCT c1.seat_id FROM cinema c1
1370    JOIN cinema c2 ON ABS(c1.seat_id - c2.seat_id) = 1 AND
1371            (c1.free = 1 AND c2.free = 1)
1372    ORDER BY
1373    c1.seat_id;
1374
1375
1376
```



Q59.

Table: SalesPerson

Column Name	Type
sales_id	int
name	varchar
salary	int
commission_rate	int
hire_date	date

sales_id is the primary key column for this table.
Each row of this table indicates the name and the ID of a salesperson alongside their salary, commission rate, and hire date.

Table: Company

Column Name	Type
com_id	int
name	varchar
city	varchar

com_id is the primary key column for this table.
Each row of this table indicates the name and the ID of a company and the city in which the company is located.

Table: Orders

Column Name	Type
order_id	int
order_date	date
com_id	int
sales_id	int
amount	int

order_id is the primary key column for this table.
com_id is a foreign key to com_id from the Company table.
sales_id is a foreign key to sales_id from the SalesPerson table.
Each row of this table contains information about one order. This includes the ID of the company, the ID of the salesperson, the date of the order, and the amount paid.

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".
Return the result table in any order.

The query result format is in the following example.

Input:
SalesPerson table:

sales_id	name	salary	commission_rate	hire_date
1	John	100000	6	4/1/2006
2	Amy	12000	5	5/1/2010
3	Mark	65000	12	12/25/2008
4	Pam	25000	25	1/1/2005
5	Alex	5000	10	2/3/2007

Company table:

com_id	name	city
1	RED	Boston
2	ORANGE	New York
3	YELLOW	Boston
4	GREEN	Austin

Orders table:

order_id	order_date	com_id	sales_id	amount
1	1/1/2014	3	4	10000
2	2/1/2014	4	5	5000
3	3/1/2014	1	1	50000
4	4/1/2014	1	4	25000

Output:

name
Amy
Mark
Alex

Explanation:
According to orders 3 and 4 in the Orders table, it is easy to tell that only salesperson John and Pam have sales to company RED, so we report all the other names in the table salesperson.

```
1431    select sales_person.name from sales_person
1432    except
1433    select sales_person.name from orders
1434    left join company on company.company_id=orders.company_id
1435    join sales_person on sales_person.sales_id=orders.sales_id
1436    where company.name='RED'
1437
1438
1439
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

name
AMY
MARK
ALEX

Q60.

Table: Triangle

Column Name	Type
x	int
y	int
z	int

(x, y, z) is the primary key column for this table.
Each row of this table contains the lengths of three line segments.

Write an SQL query to report for every three line segments whether they can form a triangle.
Return the result table in any order.
The query result format is in the following example.

Input:
Triangle table:

x	y	z
13	15	30
10	20	15

Output:

x	y	z	triangle
13	15	30	No
10	20	15	Yes

```
1454 •  select *,if(x<y+z and y<x+z and z<x+y ,'Yes','No') as checking
1455    from triangle
1456
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

	x	y	z	checking
▶	10	20	15	Yes
	13	15	30	No

Q61.

Table: Point

Column Name	Type
x	int

x is the primary key column for this table.
Each row of this table indicates the position of a point on the X-axis.

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.

Input:
Point table:

x
-1
0
2

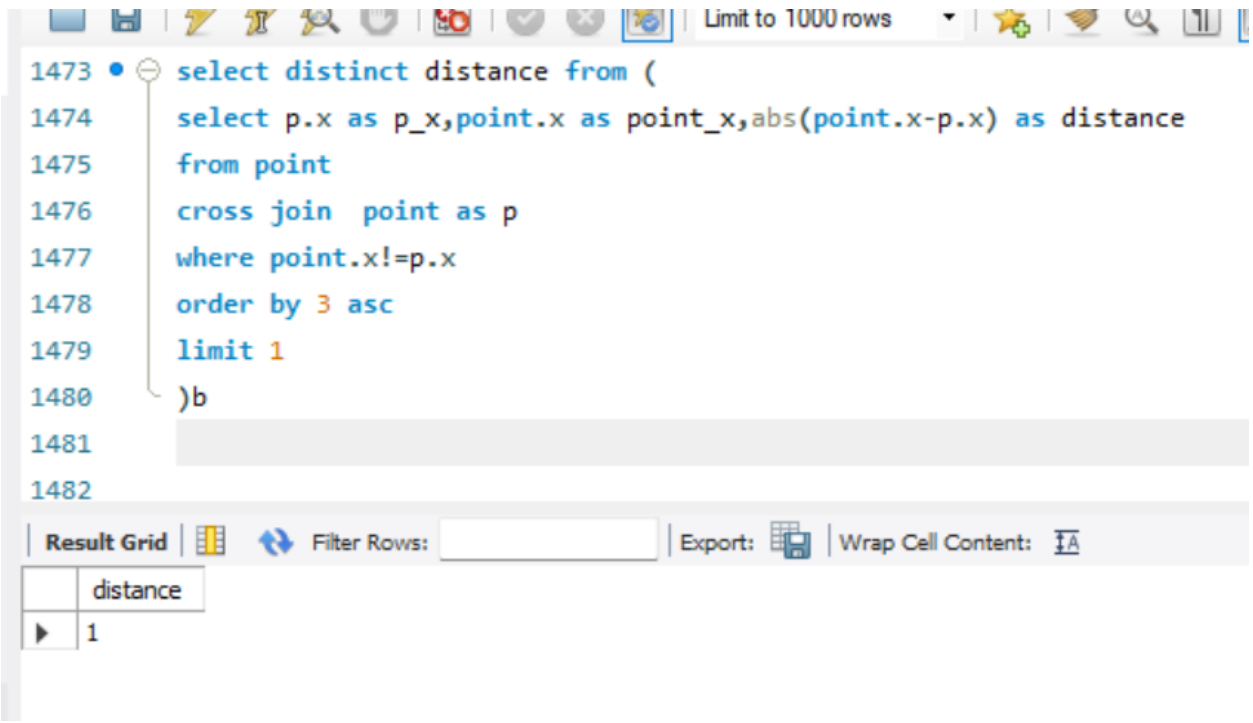
Output:

shortest
1

Explanation:

The shortest distance is between points -1 and 0 which is |(-1) - 0| = 1.

Follow up: How could you optimise your query if the Point table is ordered in ascending order?


Limit to 1000 rows
1473 select distinct distance from (
1474 select p.x as p_x,point.x as point_x,abs(point.x-p.x) as distance
1475 from point
1476 cross join point as p
1477 where point.x!=p.x
1478 order by 3 asc
1479 limit 1
1480)b
1481
1482
Result Grid
Filter Rows:
Export:
Wrap Cell Content:
distance
1

Q62.

Table: ActorDirector

Column Name	Type
actor_id	int
director_id	int
timestamp	int

timestamp is the primary key column for this table.

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.
Return the result table in any order.
The query result format is in the following example.

Input:
ActorDirector table:

actor_id	director_id	timestamp
1	1	0
1	1	1
1	1	2
1	2	3
1	2	4
2	1	5
2	1	6

Output:

actor_id	director_id
1	1

Explanation:
The only pair is (1, 1) where they cooperated exactly 3 times.

```
1503  select actor_id,director_id from (
1504  select *,count(*) over (partition by actor_id, director_id) as instance
1505  from actor_director
1506  order by 4 desc
1507  limit 1
1508  )b
1509
1510
1511
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

actor_id	director_id
1	1

Q63.

Table: Sales

Column Name	Type
sale_id	int
product_id	int
year	int
quantity	int
price	int

(sale_id, year) is the primary key of this table.
product_id is a foreign key to the Product table.
Each row of this table shows a sale on the product product_id in a certain year.
Note that the price is per unit.

Table: Product

Column Name	Type
product_id	int
product_name	varchar

product_id is the primary key of this table.
Each row of this table indicates the product name of each product.

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.
The query result format is in the following example.

Input:
Sales table:

sale_id	product_id	year	quantity	price
1	100	2008	10	5000
2	100	2009	12	5000
7	200	2011	15	9000

Product table:

product_id	product_name
100	Nokia
200	Apple
300	Samsung

Output:

product_name	year	price
Nokia	2008	5000
Nokia	2009	5000
Apple	2011	9000

Explanation:
From sale_id = 1, we can conclude that Nokia was sold for 5000 in the year 2008.
From sale_id = 2, we can conclude that Nokia was sold for 5000 in the year 2009.
From sale_id = 7, we can conclude that Apple was sold for 9000 in the year 2011.

```
1542 •   select product_name,year,price from sales
1543     join product on product.product_id=sales.product_id
1544
1545
1546
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

product_name	year	price
NOKIA	2008	5000
NOKIA	2009	5000
APPLE	2011	9000

Q64.

Table: Project

Column Name	Type
project_id	int
employee_id	int

(project_id, employee_id) is the primary key of this table.
employee_id is a foreign key to the Employee table.
Each row of this table indicates that the employee with employee_id is working on the project with project_id.

Table: Employee

Column Name	Type
employee_id	int
name	varchar
experience_years	int

employee_id is the primary key of this table.
Each row of this table contains information about one employee.

Write an SQL query that reports the average experience years of all the employees for each project, rounded to 2 digits.
Return the result table in any order.
The query result format is in the following example.

Input:
Project table:

project_id	employee_id
1	1
1	2
1	3
2	1
2	4

Employee table:

employee_id	name	experience_years
1	Khaled	3
2	Ali	2
3	John	1
4	Doe	2

Output:

project_id	average_years
1	2
2	2.5

Explanation:
The average experience years for the first project is (3 + 2 + 1) / 3 = 2.00 and for the second project is (3 + 2) / 2 = 2.50

```
1583
1584 •  select project_id,round(avg(experience_years),2) as avg_experience_yrs
1585     from project
1586    join employee on employee.employee_id=project.employee_id
1587    group by 1
1588
1589
1590
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

project_id	avg_experience_yrs
1	2.00
2	2.50

Q65.

Table: Product

Column Name	Type
product_id	int
product_name	varchar
unit_price	int

product_id is the primary key of this table.
Each row of this table indicates the name and the price of each product.

Table: Sales

Column Name	Type
seller_id	int
product_id	int
buyer_id	int
sale_date	date
quantity	int
price	int

This table has no primary key, it can have repeated rows.
product_id is a foreign key to the Product table.
Each row of this table contains some information about one sale.

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.
The query result format is in the following example.

Input:
Product table:

product_id	product_name	unit_price
1	S8	1000
2	G4	800
3	iPhone	1400

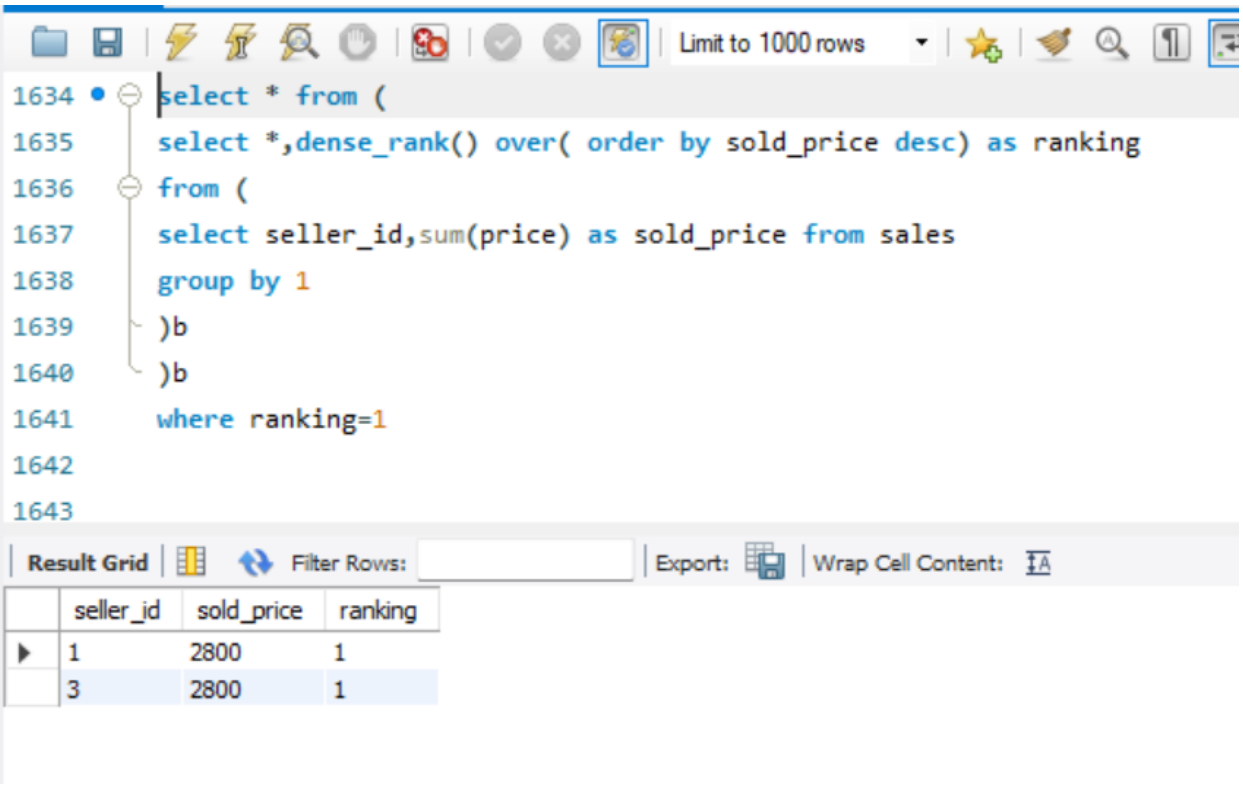
Sales table:

seller_id	product_id	buyer_id	sale_date	quantity	price
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

Output:

seller_id
1
3

Explanation: Both sellers with id 1 and 3 sold products with the most total price of 2800.



Q66.

Table: Product

Column Name	Type
product_id	int
product_name	varchar
unit_price	int

product_id is the primary key of this table.
Each row of this table indicates the name and the price of each product.

Table: Sales

Column Name	Type
seller_id	int
product_id	int
buyer_id	int
sale_date	date
quantity	int
price	int

This table has no primary key, it can have repeated rows.
product_id is a foreign key to the Product table.
Each row of this table contains some information about one sale.

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.
Return the result table in any order.
The query result format is in the following example.

Input:
Product table:

product_id	product_name	unit_price
1	S8	1000
2	G4	800
3	iPhone	1400

Sales table:

seller_id	product_id	buyer_id	sale_date	quantity	price
1	1	1	2019-01-21	2	2000
1	2	2	2019-02-17	1	800
2	1	3	2019-06-02	1	800
3	3	3	2019-05-13	2	2800

Output:

buyer_id
1

Explanation:
The buyer with id 1 bought an S8 but did not buy an iPhone. The buyer with id 3 bought both.

```
1645
1646 •  select buyer_id,count(case when product_name in ('S8','Iphone') then 1 end) as count_instance
1647    from sales
1648    join product on product.product_id=sales.product_id
1649    group by 1
1650    having count_instance=1
1651
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

buyer_id	count_instance
1	1

Q67.

Table: Customer

Column Name	Type
customer_id	int
name	varchar
visited_on	date
amount	int

(customer_id, visited_on) is the primary key for this table.
This table contains data about customer transactions in a restaurant.
visited_on is the date on which the customer with ID (customer_id) has visited the restaurant.
amount is the total paid by a customer.

You are the restaurant owner and you want to analyse a possible expansion (there will be at least one customer every day).
Write an SQL query to compute the moving average of how much the customer paid in a seven days window (i.e., current day + 6 days before). average_amount should be rounded to two decimal places.
Return result table ordered by visited_on in ascending order.
The query result format is in the following example.

Input:
Customer table:

customer_id	name	visited_on	amount
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

Output:

visited_on	amount	average_amount
2019-01-07	860	122.86

```sql
with customers as (
select visited_on,sum(amount) as amount from customer
group by 1
)
select *,avg(amount)
over( order by visited_on asc rows between  6 preceding and current row) as
moving_avg,
sum(amount)
over( order by visited_on asc rows between  6 preceding and current row) as
moving_sum

from customers
```

Q68.

Table: Scores

Column Name	Type
player_name	varchar
gender	varchar
day	date
score_points	int

(gender, day) is the primary key for this table.
A competition is held between the female team and the male team.
Each row of this table indicates that a player_name and with gender has scored score_point in someday.
Gender is 'F' if the player is in the female team and 'M' if the player is in the male team.

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.
The query result format is in the following example.

Input:
Scores table:

player_name	gender	day	score_points
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

Output:

gender	day	total
F	2019-12-30	17
F	2019-12-31	40
F	2020-01-01	57
F	2020-01-07	80
M	2019-12-18	2
M	2019-12-25	13

```
1718
1719 •  select gender,day,sum(score_points) over (partition by gender order by day ) as total_score
1720     from scores
1721
1722
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

gender	day	total_score
F	2019-12-30	17
F	2019-12-31	40
F	2020-01-01	57
F	2020-01-07	80
M	2019-12-18	2
M	2019-12-25	13

Q69.

Table: Logs

Column Name	Type
log_id	int

log_id is the primary key for this table.
Each row of this table contains the ID in a log Table.

Write an SQL query to find the start and end number of continuous ranges in the table Logs.
Return the result table ordered by start_id.
The query result format is in the following example.

Input:
Logs table:

log_id
1
2
3
7
8
10

Output:

start_id	end_id
1	3
7	8
10	10

```
1772  select min(log_id) as start_log,max(log_id) as end_log,ranking  from (
1773  SELECT log_id,sort, DENSE_RANK() OVER(ORDER BY sort) AS ranking
1774  from (
1775  SELECT log_id,ROW_number() OVER(ORDER BY log_id) AS RN,log_id - ROW_number() OVER(ORDER BY log_id) as sort
1776  FROM logs
1777  )b
1778  )b
1779  group by 3
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

start_log	end_log	ranking
1	3	1
7	8	2
10	10	3

Q70.

Table: Students

Column Name	Type
student_id	int
student_name	varchar

student_id is the primary key for this table.
Each row of this table contains the ID and the name of one student in the school.

Table: Subjects

Column Name	Type
subject_name	varchar

subject_name is the primary key for this table.
Each row of this table contains the name of one subject in the school.

Table: Examinations

Column Name	Type
student_id	int
subject_name	varchar

There is no primary key for this table. It may contain duplicates.
Each student from the Students table takes every course from the Subjects table.
Each row of this table indicates that a student with ID student_id attended the exam of subject_name.

Write an SQL query to find the number of times each student attended each exam.
Return the result table ordered by student_id and subject_name.
The query result format is in the following example.

Input:
Students table:

student_id	student_name
1	Alice
2	Bob
13	John
6	Alex

Subjects table:

subject_name
Math
Physics
Programming

Examinations table:

student_id	subject_name
1	Math
1	Physics
1	Programming
2	Programming
1	Physics
1	Math
13	Math
13	Programming
13	Physics
2	Math
1	Math

Output:

student_id	student_name	subject_name	attended_exams
1	Alice	Math	3
1	Alice	Physics	2
1	Alice	Programming	1
2	Bob	Math	1
2	Bob	Physics	0

2	Bob	Programming	1
6	Alex	Math	0
6	Alex	Physics	0
6	Alex	Programming	0
13	John	Math	1
13	John	Physics	1
13	John	Programming	1

Explanation:
The result table should contain all students and all subjects.
Alice attended the Math exam 3 times, the Physics exam 2 times, and the Programming exam 1 time.
Bob attended the Math exam 1 time, the Programming exam 1 time, and did not attend the Physics exam.
Alex did not attend any exams.
John attended the Math exam 1 time, the Physics exam 1 time, and the Programming exam 1 time.

```
1837
1838 •  select students.student_id,student_name,subject_name,count(subject_name) as exams_taken
1839    from students
1840    left join exams on exams.student_id=students.student_id
1841    group by 1,2,3
1842
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

student_id	student_name	subject_name	exams_taken
1	ALICE	MATHS	3
1	ALICE	PHYSICS	2
1	ALICE	PROGRAMMING	1
2	BOB	MATHS	1
2	BOB	PROGRAMMING	1
6	ALEX	NULL	0

Q71.

Table: Employees

Column Name	Type
employee_id	int
employee_name	varchar
manager_id	int

employee_id is the primary key for this table.
Each row of this table indicates that the employee with ID employee_id and name employee_name reports his work to his/her direct manager with manager_id
The head of the company is the employee with employee_id = 1.

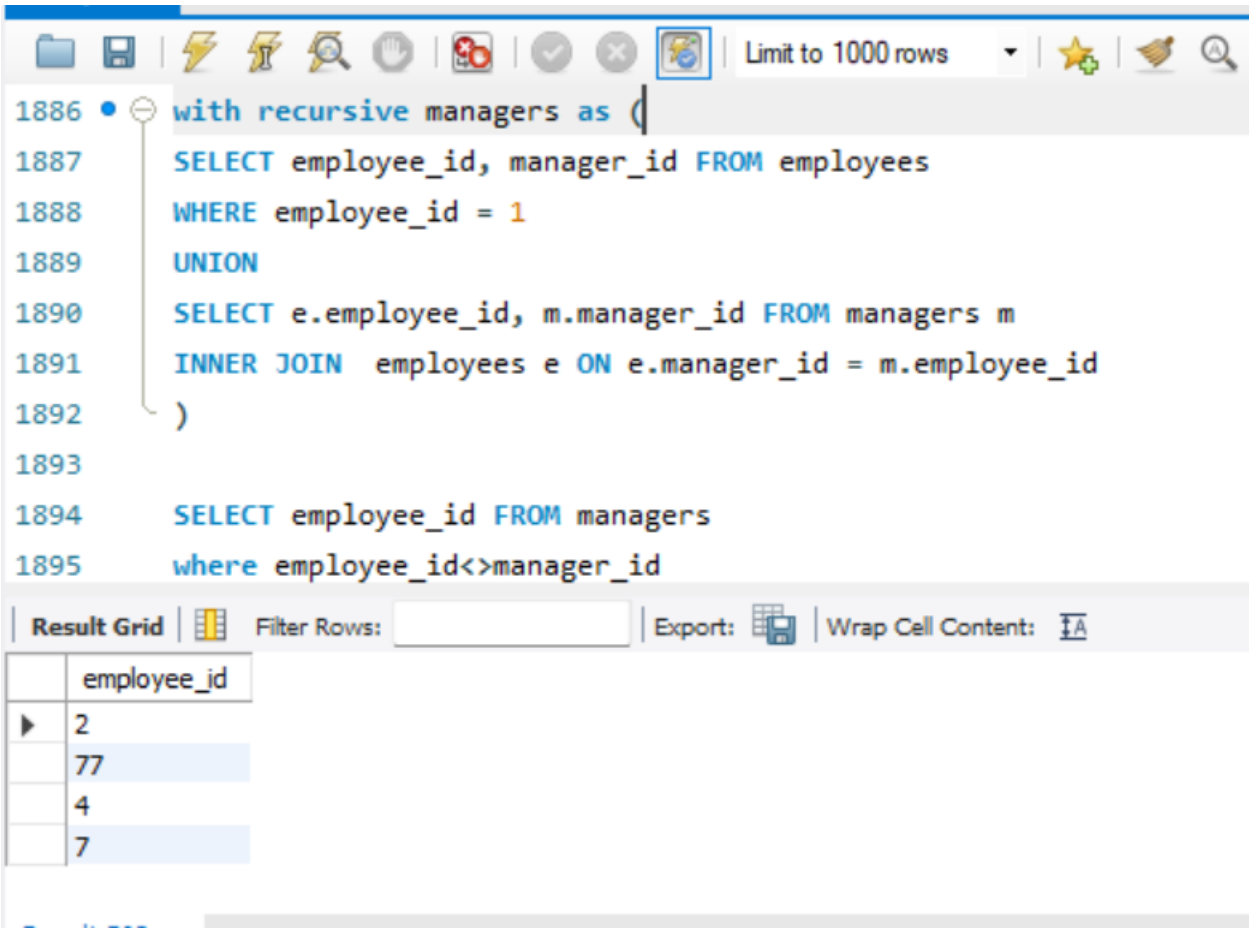
Write an SQL query to find employee_id of all employees that directly or indirectly report their work to the head of the company.
The indirect relation between managers will not exceed three managers as the company is small.
Return the result table in any order.
The query result format is in the following example.

Input:
Employees table:

employee_id	employee_name	manager_id
1	Boss	1
3	Alice	3
2	Bob	1
4	Daniel	2
7	Luis	4
8	Jhon	3
9	Angela	8
77	Robert	1

Output:

employee_id
2
77
4
7



```
1886  with recursive managers as (
1887  SELECT employee_id, manager_id FROM employees
1888  WHERE employee_id = 1
1889  UNION
1890  SELECT e.employee_id, m.manager_id FROM managers m
1891  INNER JOIN  employees e ON e.manager_id = m.employee_id
1892  )
1893
1894  SELECT employee_id FROM managers
1895  where employee_id<>manager_id
```

Limit to 1000 rows

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

employee_id
2
77
4
7

Q72.

Table: Transactions

Column Name	Type
id	int
country	varchar
state	enum
amount	int
trans_date	date

id is the primary key of this table.
The table has information about incoming transactions.
The state column is an enum of type ["approved", "declined"].

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.
Return the result table in any order.
The query result format is in the following example.

Input:
Transactions table:

id	country	state	amount	trans_date
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

Output:

month	country	trans_count	approved_count	trans_total_amount	roved_total_amo
2018-12	US	2	1	3000	1000
2019-01	US	1	1	2000	2000
2019-01	DE	1	1	2000	2000

```
1919    select country,extract(month from trans_date) as month,sum(amount) as total_amount,
1920    sum(case when state='APPROVED' then amount end) as approved_amount
1921    from transactions
1922    group by 1,2
1923
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

country	month	total_amount	approved_amount
US	12	3000	1000
US	1	2000	2000
DE	1	2000	2000

Q73.

Table: Actions

Column Name	Type
user_id	int
post_id	int
action_date	date
action	enum
extra	varchar

There is no primary key for this table, it may have duplicate rows.
The action column is an ENUM type of ('view', 'like', 'reaction', 'comment', 'report', 'share').
The extra column has optional information about the action, such as a reason for the report or a type of reaction.

Table: Removals

Column Name	Type
post_id	int
remove_date	date

post_id is the primary key of this table.
Each row in this table indicates that some post was removed due to being reported or as a result of an admin review.

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.
The query result format is in the following example.

Input:
Actions table:

user_id	post_id	action_date	action	extra
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

Removals table:

post_id	remove_date
2	2019-07-20
3	2019-07-18

Output:

average_daily_percent
75

Explanation:
The percentage for 2019-07-04 is 50% because only one post of two spam reported posts were removed.
The percentage for 2019-07-02 is 100% because one post was reported as spam and it was removed.
The other days had no spam reports so the average is (50 + 100) / 2 = 75%
Note that the output is only one number and that we do not care about the remove dates.

```
1969  select round(avg(removed/reported),2) as removed_per from (
1970  select action_date,count(case when extra ='SPAM' AND remove_date is not null then 1 end) as removed,
1971  count(case when extra ='SPAM' then 1 end) as reported from actions
1972  left join removals on removals.post_id=actions.post_id
1973  group by 1
1974  having removed!=0 and reported!=0
1975  order by 1
1976  )b
1977
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

removed_per
0.75

Q74.

Table: Activity

Column Name	Type
player_id	int
device_id	int
event_date	date
games_played	int

(player_id, event_date) is the primary key of this table.
This table shows the activity of players of some games.
Each row is a record of a player who logged in and played a number of games (possibly 0) before logging out on someday using some device.

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. In other words, you need to count the number of players that logged in for at least two consecutive days starting from their first login date, then divide that number by the total number of players.
The query result format is in the following example.

Input:
Activity table:

player_id	device_id	event_date	games_played
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

Output:

fraction
0.33

Explanation:
Only the player with id 1 logged back in after the first day he had logged in so the answer is 1/3 = 0.33

Answer:Same as Question 43 (repeat Question)

Q75.

Table: Activity

Column Name	Type
player_id	int
device_id	int
event_date	date
games_played	int

(player_id, event_date) is the primary key of this table.
This table shows the activity of players of some games.
Each row is a record of a player who logged in and played a number of games (possibly 0) before logging out on someday using some device.

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. In other words, you need to count the number of players that logged in for at least two consecutive days starting from their first login date, then divide that number by the total number of players.
The query result format is in the following example.

Input:
Activity table:

player_id	device_id	event_date	games_played
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

Output:

fraction
0.33

Explanation:
Only the player with id 1 logged back in after the first day he had logged in so the answer is 1/3 = 0.33

Answer:Same as Question 43 (repeat Question)

Q76.

Table Salaries:

Column Name	Type
company_id	int
employee_id	int
employee_name	varchar
salary	int

(company_id, employee_id) is the primary key for this table.
This table contains the company id, the id, the name, and the salary for an employee.

Write an SQL query to find the salaries of the employees after applying taxes. Round the salary to the nearest integer.
The tax rate is calculated for each company based on the following criteria:

- 0% If the max salary of any employee in the company is less than $1000.
- 24% If the max salary of any employee in the company is in the range [1000, 10000] inclusive.
- 49% If the max salary of any employee in the company is greater than $10000.

Return the result table in any order.
The query result format is in the following example.

Input:
Salaries table:

company_id	employee_id	employee_name	salary
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

Output:

company_id	employee_id	employee_name	salary
1	1	Tony	1020
1	2	Pronub	10863
1	3	Tyrrox	5508
2	1	Pam	300
2	7	Bassem	450
2	9	Hermione	700
3	7	Bocaben	76
3	2	Ognjen	1672
3	13	Nyan Cat	2508
3	15	Morning Cat	5911

Explanation:
For company 1, Max salary is 21300. Employees in company 1 have taxes = 49%
For company 2, Max salary is 700. Employees in company 2 have taxes = 0%
For company 3, Max salary is 7777. Employees in company 3 have taxes = 24%
The salary after taxes = salary - (taxes percentage / 100) * salary
For example, Salary for Morning Cat (3, 15) after taxes = 7777 - 7777 * (24 / 100) = 7777 - 1866.48 = 5910.52, which is rounded to 5911.

```
select employee_name,salary as gross_salary,round(salary-salary*per_tax) as
net_salary
from (
select *,
case
when max_salary >10000 then 0.49
when max_salary >=1000 and max_salary<=10000 then 0.24
when max_salary <1000 then 0 end as per_tax
from
(select *,max(salary) over (partition by company_id) as max_salary from
salaries)
b
)b
```

Q77.

Table Variables:

Column Name	Type
name	varchar
value	int

name is the primary key for this table.
This table contains the stored variables and their values.

Table Expressions:

Column Name	Type
left_operand	varchar
operator	enum
right_operand	varchar

(left_operand, operator, right_operand) is the primary key for this table.
This table contains a boolean expression that should be evaluated.
operator is an enum that takes one of the values ('<', '>', '=')
The values of left_operand and right_operand are guaranteed to be in the Variables table.

Write an SQL query to evaluate the boolean expressions in Expressions table.
Return the result table in any order.
The query result format is in the following example.

Input:
Variables table:

name	value
x	66
y	77

Expressions table:

left_operand	operator	right_operand
x	>	y
x	<	y
x	=	y
y	>	x
y	<	x
x	=	x

```
select *,
case
when operator='<' then  if (value1<value2,'true','false')
when operator='>' then  if (value1>value2,'true','false')
when operator='=' then  if (value1=value2,'true','false')
end as result

from (
select variables.value as value1,operator,v.value as value2 from
expressions
left join variables on variables.name=left_operand
left join variables as v on v.name=right_operand
)b
```

Q78.

Table Person:

Column Name	Type
id	int
name	varchar
phone_number	varchar

id is the primary key for this table.
Each row of this table contains the name of a person and their phone number.
Phone number will be in the form 'xxx-yyyyyyy' where xxx is the country code (3 characters) and yyyyyyy is the phone number (7 characters) where x and y are digits. Both can contain leading zeros.

Table Country:

Column Name	Type
name	varchar
country_code	varchar

country_code is the primary key for this table.
Each row of this table contains the country name and its code. country_code will be in the form 'xxx' where x is digits.

Table Calls:

Column Name	Type
caller_id	int
callee_id	int
duration	int

There is no primary key for this table, it may contain duplicates.
Each row of this table contains the caller id, callee id and the duration of the call in minutes. caller_id != callee_id

A telecommunications company wants to invest in new countries. The company intends to invest in the countries where the average call duration of the calls in this country is strictly greater than the global average call duration.
Write an SQL query to find the countries where this company can invest.
Return the result table in any order.
The query result format is in the following example.

Input:
Person table:

id	name	phone_number
3	Jonathan	051-1234567
12	Elvis	051-7654321
1	Moncef	212-1234567
2	Maroua	212-6523651
7	Meir	972-1234567
9	Rachel	972-0011100

Country table:

name	country_code
Peru	51
Israel	972
Morocco	212
Germany	49
Ethiopia	251

Calls table:

caller_id	callee_id	duration
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

Output:

country
Peru

Explanation:
The average call duration for Peru is (102 + 102 + 330 + 330 + 5 + 5) / 6 = 145.666667
The average call duration for Israel is (33 + 4 + 13 + 13 + 3 + 1 + 1 + 7) / 8 = 9.37500
The average call duration for Morocco is (33 + 4 + 59 + 59 + 3 + 7) / 6 = 27.5000
Global call duration average = (2 * (33 + 4 + 59 + 102 + 330 + 5 + 13 + 3 + 1 + 7)) / 20 = 55.70000
Since Peru is the only country where the average call duration is greater than the global average, it is the only recommended country.

Answer 78: SAME AS Question .55

Q79.

Write a query that prints a list of employee names (i.e.: the name attribute) from the Employee table in alphabetical order.
Level - Easy
Hint - Use ORDER BY
Input Format
The Employee table containing employee data for a company is described as follows:

Column	Type
employee_id	Integer
name	String
months	Integer
salary	Integer

where employee_id is an employee's ID number, name is their name, months is the total number of months they've been working for the company, and salary is their monthly salary.

Sample Input

employee_id	name	months	salary
12228	Rose	15	1968
33645	Angela	1	3443
45692	Frank	17	1608
56118	Patrick	7	1345
59725	Lisa	11	2330
74197	Kimberly	16	4372
78454	Bonnie	8	1771
83565	Michael	6	2017
98607	Todd	5	3396
99989	Joe	9	3573

Sample Output
Angela
Bonnie
Frank
Joe
Kimberly
Lisa
Michael
Patrick
Rose
Todd

```
2090
2091 •    select name from employee
2092      order by 1
2093
```

Result Grid | Filter Rows: | Export: | Wrap

name
ANGELA
BONNIE
FRANK
JOE
KINBERLY

employee 615 ×

Output

Q80.

Assume you are given the table below containing information on user transactions for particular products. Write a query to obtain the **year-on-year growth rate** for the total spend of each product for each year.

Output the year (in ascending order) partitioned by product id, current year's spend, previous year's spend and year-on-year growth rate (percentage rounded to 2 decimal places).

Level - Hard
Hint - Use extract function

user_transactions Table:

Column Name	Type
transaction_id	integer
product_id	integer
spend	decimal
transaction_date	datetime

user_transactions Example Input:

transaction_id	product_id	spend	transaction_date
1341	123424	1500.60	12/31/2019 12:00:00
1423	123424	1000.20	12/31/2020 12:00:00
1623	123424	1246.44	12/31/2021 12:00:00
1322	123424	2145.32	12/31/2022 12:00:00

Example Output:

y	product_id	curr_year_spend	prev_year_spend	yoy_rate
2	123424	1500.60		
2	123424	1000.20	1500.60	-33.35

```
2107    --          (1322,123424,2145.32, '2022/12/31 12:00:00');
2108    select *,round((spend-previous_year_transaction)/previous_year_transaction,2) as rate
2109    from (
2110    select *,extract(year from date(transaction_date)) as years,
2111    coalesce(lag(spend) over(order by extract(year from date(transaction_date)) asc),0) as previous_year_transaction
2112    from user_transactions
2113    )b
2114
2115    -- drop table user_transactions
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

transaction_id	product_id	spend	transaction_date	years	previous_year_transaction	rate
1341	123424	1500.6	2019-12-31 12:00:00	2019	0	NULL
1423	123424	1000.2	2020-12-31 12:00:00	2020	1500.6	-0.33
1623	123424	1246.44	2021-12-31 12:00:00	2021	1000.2	0.25
1322	123424	2145.32	2022-12-31 12:00:00	2022	1246.44	0.72

Q81.

Amazon wants to maximise the number of items it can stock in a 500,000 square feet warehouse. It wants to stock as many prime items as possible, and afterwards use the remaining square footage to stock the most number of non-prime items.
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.

Hint - create a table containing a summary of the necessary fields such as item type ('prime_eligible', 'not_prime'), SUM of square footage, and COUNT of items grouped by the item type.

inventory table:

Column Name	Type
item_id	integer
item_type	string
item_category	string
square_footage	decimal

inventory Example Input:

item_id	item_type	item_category	square_footage
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

Example Output:

item_type	item_count
prime_eligible	9285
not_prime	6

```
WITH temp_inventory AS (
                  SELECT
                        item_type,
                        SUM(square_foot) AS square_foot_per_category,
                        COUNT(*) AS count_of_items,
                        CASE
                              WHEN item_type = 'PRIME_ELIGIBLE' THEN
FLOOR(500000/SUM(square_foot)) * COUNT(*)
                              END AS prime_items_count
                  FROM
                        inventory
                  GROUP BY
                        item_type
            ),

    temp_inventory2 AS (
                  SELECT
                        (500000 -
SUM(square_foot_per_category)*FLOOR(500000/SUM(square_foot_per_category)))
AS area_left
                        FROM
                        temp_inventory
                  WHERE
                        item_type = 'PRIME_ELIGIBLE'
            ),

    temp_inventory3 AS (
                  SELECT
                        item_type,
                        CASE
                                WHEN item_type = 'PRIME_ELIGIBLE'
                                            THEN prime_items_count
                                WHEN item_type = 'NOT_PRIME'
                                            THEN FLOOR((SELECT
area_left FROM temp_inventory2) / square_foot_per_category) *
count_of_items
                        END AS item_count
                    FROM
                        temp_inventory
            )
```

```
SELECT
        item_type,
        item_count
FROM
        temp_inventory3;
```

Q82.

Assume you have the table below containing information on Facebook user actions. W
obtain the active user retention in July 2022. Output the month (in numerical format 1
number of monthly active users (MAUs).
Hint: An active user is a user who has user action ("sign-in", "like", or "comment") in the
and last month.

Hint- Use generic correlated subquery

user_actions Table:

Column Name	Type
user_id	integer
event_id	integer
event_type	string ("sign-in, "like", "comment")
event_date	datetime

user_actionsExample Input:

user_id	event_id	event_type	event_date
445	7765	sign-in	05/31/2022 12:00:00
742	6458	sign-in	06/03/2022 12:00:00
445	3634	like	06/05/2022 12:00:00
742	1374	comment	06/05/2022 12:00:00
648	3124	like	06/18/2022 12:00:00

month	monthly_active_users
6	1

```
WITH temp_actions AS (
                  SELECT
                        user_id,
                        event_date,
                        event_type,
                        SUBSTR(event_date, 6, 2) - lag(SUBSTR(event_date,
6, 2)) OVER w AS difference
                  FROM
                        user_actions
                  WINDOW
                        w as (PARTITION BY user_id ORDER BY event_date)
            ),

        temp_actions2 AS (
                  SELECT
                        SUBSTR(event_date, 6, 2) AS months,
                        COUNT(user_id) AS monthly_active_users
                  FROM
                        temp_actions
                  WHERE
                        difference = 1 AND event_type IN ('LIKE',
'COMMENT', 'SIGN-IN')
                  GROUP BY
                        months
                        )

SELECT
      months,
        monthly_active_users
FROM
        temp_actions2;
```

Q83.

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.

Write a query to report the median of searches made by a user. Round the median to one decimal point.

Hint- Write a subquery or common table expression (CTE) to generate a series of data (that's keyword for column) starting at the first search and ending at some point with an optional incremental value.

search_frequency Table:

Column Name	Type
searches	integer
num_users	integer

search_frequency Example Input:

searches	num_users
1	2
2	2
3	3
4	1

Example Output:

median
2.5

```
2198  select *,if(max_cumulative>=from_cumulative and max_cumulative<cumulative_num_users,(searches_f+searches)/2,'') as median
2199  from (
2200  select searches_f,searches,coalesce(lag(cumulative_num_users) over(order by searches),0) as from_cumulative,cumulative_num_users,
      max_cumulative from (
2201  select *,max(cumulative_num_users) over ()/2 as max_cumulative
2202  from (
2203  select coalesce(lag(searches)over (order by searches),0) as searches_f,searches,num_users ,sum(num_users) over (order by searches) as
      cumulative_num_users
2204  from search_frequency
2205  )b
2206  )b
2207  )b
2208
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

searches_f	searches	from_cumulative	cumulative_num_users	max_cumulative	median
0	1	0	2	4.0000	
1	2	2	4	4.0000	
2	3	4	7	4.0000	2.5000
3	4	7	8	4.0000	

Q84.

Write a query to update the Facebook advertiser's status using the daily_pay table. Advertiser is a two-column table containing the user id and their payment status based on the last payment and daily_pay table has current information about their payment. Only advertisers who paid will show up in this table.
Output the user id and current payment status sorted by the user id.

Hint- Query the `daily_pay` table and check through the advertisers in this table. .

advertiser Table:

Column Name	Type
user_id	string
status	string

advertiser Example Input:

user_id	status
bing	NEW
yahoo	NEW
alibaba	EXISTING

daily_pay Table:

Column Name	Type
user_id	string
paid	decimal

daily_pay Example Input:

user_id	paid
yahoo	45.00

alibaba	100.00
target	13.00

Definition of advertiser status:

- New: users registered and made their first payment.
- Existing: users who paid previously and recently made a current payment.
- Churn: users who paid previously, but have yet to make any recent payment.
- Resurrect: users who did not pay recently but may have made a previous payment and have made payment again recently.

Example Output:

user_id	new_status
bing	CHURN
yahoo	EXISTING
alibaba	EXISTING

Bing's updated status is CHURN because no payment was made in the daily_pay table whereas Yahoo which made a payment is updated as EXISTING.

The dataset you are querying against may have different input & output - this is just an example!

Read this before proceeding to solve the question

For better understanding of the advertiser's status, we're sharing with you a table of possible transitions based on the payment status.

#	Start	End	Condition
1	NEW	EXISTING	Paid on day T
2	NEW	CHURN	No pay on day T
3	EXISTING	EXISTING	Paid on day T
4	EXISTING	CHURN	No pay on day T
5	CHURN	RESURRECT	Paid on day T
6	CHURN	CHURN	No pay on day T
7	RESURRECT	EXISTING	Paid on day T

8	RESURRECT	CHURN	No pay on day T

1. Row 2, 4, 6, 8: As long as the user has not paid on day T, the end status is updated to CHURN regardless of the previous status.
2. Row 1, 3, 5, 7: When the user paid on day T, the end status is updated to either EXISTING or RESURRECT, depending on their previous state. RESURRECT is only possible when the previous state is CHURN. When the previous state is anything else, the status is updated to EXISTING.

```
2233  select *,
2234  case
2235  when recent_paid_status is null and old_status='NEW' then 'Churn'
2236  when recent_paid_status is not null and old_status in ('NEW','EXISTING') then 'EXISTING'
2237  when recent_paid_status is not null and old_status IS NULL then 'NEW'
2238  END as new_status from (
2239  select advertiser.user_id,advertiser.status as old_status,paid as recent_paid_status from advertiser
2240  left join daily_pay on daily_pay.user_id=advertiser.user_id
2241  )b
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

user_id	old_status	recent_paid_status	new_status
BING	NEW	NULL	Churn
YAHOO	NEW	45	EXISTING
ALIBABA	EXISTING	100	EXISTING

Q85.

Amazon Web Services (AWS) is powered by fleets of servers. Senior management has requested data-driven solutions to optimise server usage.

Write a query that calculates the total time that the fleet of servers was running. The output should be in units of full days.

Level - Hard
Hint-

1. Calculate individual uptimes

2. Sum those up to obtain the uptime of the whole fleet, keeping in mind that the result must be output in units of full days

Assumptions:

- Each server might start and stop several times.
- The total time in which the server fleet is running can be calculated as the sum of each server's uptime.

server_utilization Table:

Column Name	Type
server_id	integer
status_time	timestamp
session_status	string

server_utilization Example Input:

server_id	status_time	session_status
1	08/02/2022 10:00:00	start
1	08/04/2022 10:00:00	stop
2	08/17/2022 10:00:00	start
2	08/24/2022 10:00:00	stop

Example Output:

total_uptime_days
21

```
2258  select sum(total_uptime) as total_uptime from (
2259  select *,EXTRACT(DAY from STR_TO_DATE(status_time2, '%m/%d/%y'))-EXTRACT(DAY from STR_TO_DATE(status_time, '%m/%d/%y')) as total_uptime
2260  from (
2261  select *,lead(status_time) over (partition by server_id order by status_time) as status_time2
2262   from server_utilization
2263  )b
2264  where session_status='start'
2265  )b
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

total_uptime
9

Q86.

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

Assumptions:

- The first transaction of such payments should not be counted as a repeated payment. This means, if there are two transactions performed by a merchant with the same credit card and for the same amount within 10 minutes, there will only be 1 repeated payment.

transactions Table:

Column Name	Type
transaction_id	integer
merchant_id	integer
credit_card_id	integer
amount	integer

transaction_timestamp	datetime

transactions Example Input:

transaction_id	merchant_id	credit_card_id	amount	transaction_timestamp
1	101	1	100	09/25/2022 12:00:00
2	101	1	100	09/25/2022 12:08:00
3	101	1	100	09/25/2022 12:28:00
4	102	2	300	09/25/2022 12:00:00
6	102	2	400	09/25/2022 14:00:00

Example Output:

payment_count
1

```
2292 •  select count(*) as  error_instances
2293    from (
2294    select *,timestampdiff(minute,transaction_timestamp,next_transaction) as time_diff from (
2295    select *,lead(transaction_timestamp) over (partition by merchant_id,credit_card_id,amount order by transaction_timestamp asc) as
        next_transaction
2296    from transactions
2297    )b
2298    )b
2299    where time_diff<=10
2300
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

error_instances
1

Q87.
DoorDash's Growth Team is trying to make sure new users (those who are making orders in their first 14 days) have a great experience on all their orders in their 2 weeks on the platform.
Unfortunately, many deliveries are being messed up because:

- the orders are being completed incorrectly (missing items, wrong order, etc.)
- the orders aren't being received (wrong address, wrong drop off spot)
- the orders are being delivered late (the actual delivery time is 30 minutes later than when the order was placed). Note that the estimated_delivery_timestamp is automatically set to 30 minutes after the order_timestamp.

Hint- Use Where Clause and joins

Write a query to find the bad experience rate in the first 14 days for new users who signed up in June 2022. Output the percentage of bad experience rounded to 2 decimal places.

orders Table:

Column Name	Type
order_id	integer

customer_id	integer
trip_id	integer
status	string ('completed successfully', 'completed incorrectly', 'never received')
order_timestamp	timestamp

orders Example Input:

order_id	customer_id	trip_id	status	order_timestamp
727424	8472	100463	completed successfully	06/05/2022 09:12:00
242513	2341	100482	completed incorrectly	06/05/2022 14:40:00
141367	1314	100362	completed incorrectly	06/07/2022 15:03:00
582193	5421	100657	never_received	07/07/2022 15:22:00
253613	1314	100213	completed successfully	06/12/2022 13:43:00

trips Table:

Column Name	Type
dasher_id	integer
trip_id	integer
estimated_delivery_timestamp	timestamp
actual_delivery_timestamp	timestamp

trips Example Input:

dasher_id	trip_id	estimated_delivery_timestamp	actual_delivery_timestamp
101	100463	06/05/2022 09:42:00	06/05/2022 09:38:00
102	100482	06/05/2022 15:10:00	06/05/2022 15:46:00

101	100362	06/07/2022 15:33:00	06/07/2022 16:45:00
102	100657	07/07/2022 15:52:00	-
103	100213	06/12/2022 14:13:00	06/12/2022 14:10:00

customers Table:

Column Name	Type
customer_id	integer
signup_timestamp	timestamp

customers Example Input:

customer_id	signup_timestamp
8472	05/30/2022 00:00:00
2341	06/01/2022 00:00:00
1314	06/03/2022 00:00:00
1435	06/05/2022 00:00:00
5421	06/07/2022 00:00:00

Example Output:

bad_experience_pct
75.00

```
2356
2357 •  select round(count(case when status not like '%successfully%' then 1 end)/count(*),2)*100 as per_experience
2358     from (
2359    select order_id,orders.customer_id,orders.trip_id,status,signup_timestamp,order_timestamp,estimated_delivery_timestamp,
2360    convert(SUBSTRING(signup_timestamp, 1, 2), unsigned integer) as monthly
2361    from orders
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

per_experience
75.00

Q88

Table: Scores

Column Name	Type
player_name	varchar
gender	varchar
day	date
score_points	int

(gender, day) is the primary key for this table.
A competition is held between the female team and the male team.
Each row of this table indicates that a player_name and with gender has scored score_point in someday.
Gender is 'F' if the player is in the female team and 'M' if the player is in the male team.

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.
The query result format is in the following example.

Input:
Scores table:

player_name	gender	day	score_points
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

Output:

gender	day	total
F	2019-12-30	17
F	2019-12-31	40
F	2020-01-01	57
F	2020-01-07	80
M	2019-12-18	2
M	2019-12-25	13

M	2019-12-30	26
M	2019-12-31	29
M	2020-01-07	36

Explanation:

For the female team:
The first day is 2019-12-30, Priyanka scored 17 points and the total score for the team is 17.
The second day is 2019-12-31, Priya scored 23 points and the total score for the team is 40.
The third day is 2020-01-01, Aron scored 17 points and the total score for the team is 57.
The fourth day is 2020-01-07, Alice scored 23 points and the total score for the team is 80.

For the male team:
The first day is 2019-12-18, Jose scored 2 points and the total score for the team is 2.
The second day is 2019-12-25, Khali scored 11 points and the total score for the team is 13.
The third day is 2019-12-30, Slaman scored 13 points and the total score for the team is 26.
The fourth day is 2019-12-31, Joe scored 3 points and the total score for the team is 29.
The fifth day is 2020-01-07, Bajrang scored 7 points and the total score for the team is 36.

Q89

Answer Same as Question 68

Q89.

Table Person:

Column Name	Type
id	int
name	varchar
phone_number	varchar

id is the primary key for this table.
Each row of this table contains the name of a person and their phone number.
Phone number will be in the form 'xxx-yyyyyyy' where xxx is the country code (3 characters) and yyyyyyy is the phone number (7 characters) where x and y are digits. Both can contain leading zeros.

Table Country:

Column Name	Type
name	varchar
country_code	varchar

country_code is the primary key for this table.
Each row of this table contains the country name and its code. country_code will be in the form 'xxx' where x is digits.

Table Calls:

Column Name	Type
caller_id	int
callee_id	int
duration	int

There is no primary key for this table, it may contain duplicates.
Each row of this table contains the caller id, callee id and the duration of the call in minutes. caller_id != callee_id

A telecommunications company wants to invest in new countries. The company intends to invest in the countries where the average call duration of the calls in this country is strictly greater than the global average call duration.
Write an SQL query to find the countries where this company can invest.
Return the result table in any order.
The query result format is in the following example.

Input:
Person table:

id	name	phone_number
3	Jonathan	051-1234567
12	Elvis	051-7654321
1	Moncef	212-1234567
2	Maroua	212-6523651
7	Meir	972-1234567
9	Rachel	972-0011100

Country table:

name	country_code
Peru	51
Israel	972
Morocco	212
Germany	49
Ethiopia	251
Ethiopia	251

Calls table:

caller_id	callee_id	duration
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

Output:

country
Peru

Explanation:
The average call duration for Peru is (102 + 102 + 330 + 330 + 5 + 5) / 6 = 145.666667
The average call duration for Israel is (33 + 4 + 13 + 13 + 3 + 1 + 1 + 7) / 8 = 9.37500
The average call duration for Morocco is (33 + 4 + 59 + 59 + 3 + 7) / 6 = 27.5000
Global call duration average = (2 * (33 + 4 + 59 + 102 + 330 + 5 + 13 + 3 + 1 + 7)) / 20 = 55.70000
Since Peru is the only country where the average call duration is greater than the global average, it is the only recommended country.

Answer Same as Question 55

‘

Q90.

Table: Numbers

Column Name	Type
num	int
frequency	int

num is the primary key for this table.
Each row of this table shows the frequency of a number in the database.

The median is the value separating the higher half from the lower half of a data sample.
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.
The query result format is in the following example.

Input:
Numbers table:

num	frequency
0	7
1	1
2	3
3	1

Output:

median
0

Explanation:
If we decompose the Numbers table, we will get [0, 0, 0, 0, 0, 0, 0, 1, 2, 2, 2, 3], so the median is (0 + 0) / 2 = 0.

```
WITH RECURSIVE num_frequency (num,frequency, i) AS
(
SELECT  num,frequency,1
FROM   numbers
UNION ALL
SELECT num,frequency,i+1
FROM   num_frequency
WHERE  num_frequency.i < num_frequency.frequency
)

select (max(case when numbers=lower_limit then num else null end)
+max(case when numbers=upper_limit then num else null end))/2 as median
from (
select *,
case
when  total_number%2=0 then total_number/2
else  (total_number+1)/2 end as lower_limit,
case
when  total_number%2=0 then total_number/2+1
else (total_number+1)/2
end as upper_limit

from (
select *,max(numbers) over() as total_number from (
select num,row_number() over (order by num)
as numbers from num_frequency
)b
)b
)b
```

2 = 0.

Q91.

Table: Salary

Column Name	Type
id	int
employee_id	int
amount	int
pay_date	date

id is the primary key column for this table.
Each row of this table indicates the salary of an employee in one month.
employee_id is a foreign key from the Employee table.

Table: Employee

Column Name	Type
employee_id	int
department_id	int

employee_id is the primary key column for this table.
Each row of this table indicates the department of an employee.

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.
Return the result table in any order.
The query result format is in the following example.

Input:
Salary table:

id	employee_id	amount	pay_date
1	1	9000	2017/03/31
2	2	6000	2017/03/31
3	3	10000	2017/03/31
4	1	7000	2017/02/28
5	2	6000	2017/02/28
6	3	8000	2017/02/28

Employee table:

employee_id	department_id
1	1
2	2
3	2

Output:

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

Explanation:
In March, the company's average salary is (9000+6000+10000)/3 = 8333.33...
The average salary for department '1' is 9000, which is the salary of employee_id '1' since there is only one employee in this department. So the comparison result is 'higher' since 9000 > 8333.33 obviously.
The average salary of department '2' is (6000 + 10000)/2 = 8000, which is the average of employee_id '2' and '3'. So the comparison result is 'lower' since 8000 < 8333.33.

With the same formula for the average salary comparison in February, the result is 'same' since both the departments '1' and '2' have the same average salary with the company, which is 7000.

```
2463
2464 • select distinct extract(month from paydate) as pay_month,department_id,avg_dep_salary,avg_salary ,
2465   case when avg_dep_salary>avg_salary then 'Higher'
2466   when avg_dep_salary<avg_salary then 'Lower'
2467   else 'Equal'
2468   end as remark
2469   from (
2470   select paydate,department_id,avg(amount) over (partition by extract(month from paydate),department_id) as avg_dep_salary,avg(amount) over
       (partition by extract(month from paydate)) as avg_salary
2471   from salary
2472   join employee on employee.employee_id=salary.employee_id
2473   order by extract(month from paydate)
2474   )b
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

pay_month	department_id	avg_dep_salary	avg_salary	remark
2	1	7000.0000	7000.0000	Equal
2	2	7000.0000	7000.0000	Equal
3	1	9000.0000	8333.3333	Higher
3	2	8000.0000	8333.3333	Lower

Result Grid

Q92.

Table: Activity

Column Name	Type
player_id	int
device_id	int
event_date	date
games_played	int

(player_id, event_date) is the primary key of this table.
This table shows the activity of players of some games.
Each row is a record of a player who logged in and played a number of games (possibly 0) before logging out on someday using some device.

The install date of a player is the first login day of that player.
We define day one retention of some date x to be the number of players whose install date is x and they logged back in on the day right after x, divided by the number of players whose install date is x, rounded to 2 decimal places.
Write an SQL query to report for each install date, the number of players that installed the game on that day, and the day one retention.
Return the result table in any order.
The query result format is in the following example.

Input:
Activity table:

player_id	device_id	event_date	games_played
1	2	2016-03-01	5
1	2	2016-03-02	6
2	3	2017-06-25	1
3	1	2016-03-01	0
3	4	2016-07-03	5

Output:

install_dt	installs	Day1_retention
2016-03-01	2	0.5
2017-06-25	1	0

Explanation:
Player 1 and 3 installed the game on 2016-03-01 but only player 1 logged back in on 2016-03-02 so the day 1 retention of 2016-03-01 is 1 / 2 = 0.50
Player 2 installed the game on 2017-06-25 but didn't log back in on 2017-06-26 so the day 1 retention of 2017-06-25 is 0 / 1 = 0.00

```
2518 •  select
2519    first_login,count(distinct player_id) as install,round(count(case when event_date=first_login then nex_day_player_id end )/count(distinct
        player_id),2) as retention
2520    from (
2521    select activity.player_id as player_id,activity.event_date,a.player_id as nex_day_player_id,min(activity.event_date) over (partition by
        activity.player_id) as first_login
2522    from activity
2523    left join activity as a on a.player_id=activity.player_id and a.event_date-1=activity.event_date
2524    )b
2525    group by 1
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

first_login	install	retention
2016-03-01	2	0.50
2017-06-25	1	0.00

Result Grid

Q93.

Table: Players

Column Name	Type
player_id	int
group_id	int

player_id is the primary key of this table.
Each row of this table indicates the group of each player.

Table: Matches

Column Name	Type
match_id	int
first_player	int
second_player	int
first_score	int
second_score	int

match_id is the primary key of this table.
Each row is a record of a match, first_player and second_player contain the player_id of each match.
first_score and second_score contain the number of points of the first_player and second_player respectively.
You may assume that, in each match, players belong to the same group.

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.
Return the result table in any order.
The query result format is in the following example.

Input:
Players table:

player_id	group_id
15	1
25	1
30	1
45	1
10	2
35	2
50	2

20	3
40	3

Matches table:

match_id	first_player	second_player	first_score	second_score
1	15	45	3	0
2	30	25	1	2
3	30	15	2	0
4	40	20	5	2
5	35	50	1	1

Output:

group_id	player_id
1	15
2	35
3	40

Answer Same as Question 50

Q94.

Table: Student

Column Name	Type
student_id	int
student_name	varchar

student_id is the primary key for this table.
student_name is the name of the student.

Table: Exam

Column Name	Type
exam_id	int
student_id	int
score	int

(exam_id, student_id) is the primary key for this table.
Each row of this table indicates that the student with student_id had a score points in the exam with id exam_id.

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.

Return the result table ordered by student_id.
The query result format is in the following example.

Input:
Student table:

student_id	student_name
1	Daniel
2	Jade
3	Stella
4	Jonathan
5	Will

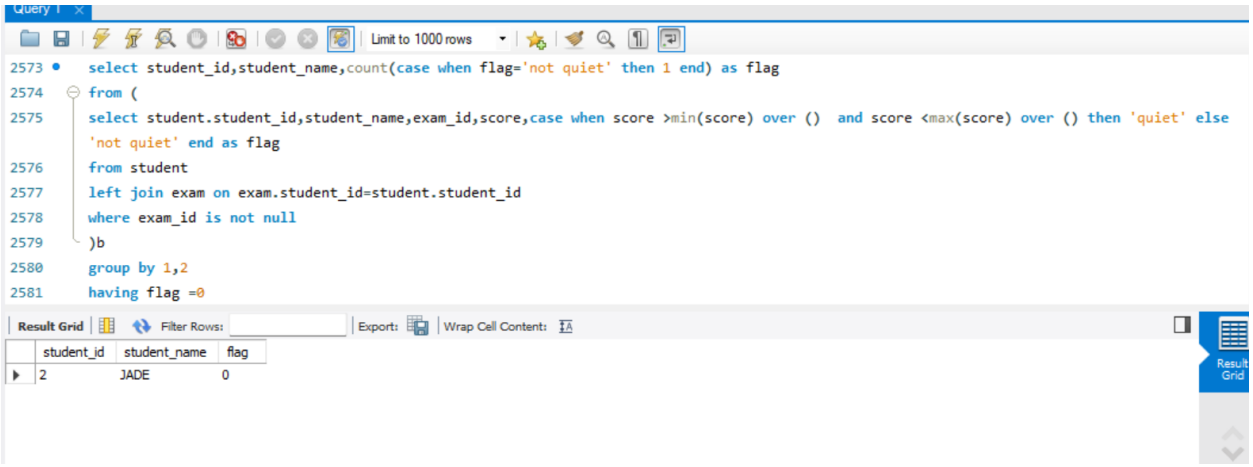
Exam table:

exam_id	student_id	score
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

Output:

student_id	student_name
2	Jade

Explanation:
For exam 1: Student 1 and 3 hold the lowest and high scores respectively.
For exam 2: Student 1 holds both the highest and lowest score.
For exam 3 and 4: Student 1 and 4 hold the lowest and high scores respectively.
Students 2 and 5 have never got the highest or lowest in any of the exams.
Since student 5 is not taking any exam, he is excluded from the result.
So, we only return the information of Student 2.


Query 1
Limit to 1000 rows
2573 select student_id,student_name,count(case when flag='not quiet' then 1 end) as flag
2574 from (
2575 select student.student_id,student_name,exam_id,score,case when score >min(score) over () and score <max(score) over () then 'quiet' else
'not quiet' end as flag
2576 from student
2577 left join exam on exam.student_id=student.student_id
2578 where exam_id is not null
2579)b
2580 group by 1,2
2581 having flag =0
Result Grid
Filter Rows:
Export:
Wrap Cell Content:
student_id student_name flag
2 JADE 0

Q95.

Table: Student

Column Name	Type
student_id	int
student_name	varchar

student_id is the primary key for this table.
student_name is the name of the student.

Table: Exam

Column Name	Type
exam_id	int
student_id	int
score	int

(exam_id, student_id) is the primary key for this table.
Each row of this table indicates that the student with student_id had a score points in the exam with id exam_id.

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.
Return the result table ordered by student_id.
The query result format is in the following example.

Input:
Student table:

student_id	student_name
1	Daniel
2	Jade
3	Stella
4	Jonathan
5	Will

Exam table:

exam_id	student_id	score
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

Output:

student_id	student_name
2	Jade

Explanation:
For exam 1: Student 1 and 3 hold the lowest and high scores respectively.
For exam 2: Student 1 holds both the highest and lowest score.
For exam 3 and 4: Student 1 and 4 hold the lowest and high scores respectively.
Students 2 and 5 have never got the highest or lowest in any of the exams.
Since student 5 is not taking any exam, he is excluded from the result.
So, we only return the information of Student 2.

Answer Same as Question 94

Q96.

You're given two tables on Spotify users' streaming data. songs_history table contains the historical streaming data and songs_weekly table contains the current week's streaming data.
Write a query to output the user id, song id, and cumulative count of song plays as of 4 August 2022 sorted in descending order.

Hint- Use group by

Definitions:

- song_weekly table currently holds data from 1 August 2022 to 7 August 2022.

- songs_history table currently holds data up to to 31 July 2022. The output should include the historical data in this table.

Assumption:

- There may be a new user or song in the songs_weekly table not present in the songs_history table.

songs_history Table:

Column Name	Type
history_id	integer
user_id	integer
song_id	integer
song_plays	integer

songs_history Example Input:

history_id	user_id	song_id	song_plays
10011	777	1238	11
12452	695	4520	1

song_plays: Refers to the historical count of streaming or song plays by the user.

songs_weekly Table:

Column Name	Type
user_id	integer
song_id	integer
listen_time	datetime

songs_weekly Example Input:

user_id	song_id	listen_time
777	1238	08/01/2022 12:00:00
695	4520	08/04/2022 08:00:00

125	9630	08/04/2022 16:00:00
695	9852	08/07/2022 12:00:00

Example Output:

user_id	song_id	song_plays
777	1238	12
695	4520	2
125	9630	1

```
2620 •  select songs_weekly.user_id,count(*)+coalesce(sum(distinct song_plays),0) as songs_played
2621    from songs_weekly
2622    left join songs_history on songs_history.user_id=songs_weekly.user_id
2623    where date_format(STR_TO_DATE(listen_time,'%m/%d/%Y'),'%m/%d/%Y')<='08/04/2022'
2624    group by 1
2625
2626
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

user_id	songs_played
125	1
695	2
777	12

Result 182 ×

Read Only

Q97.

New TikTok users sign up with their emails, so each signup requires a text confirmation to activate the new user's account.
Write a query to find the confirmation rate of users who confirmed their signups with text messages. Round the result to 2 decimal places.

Hint- Use Joins

Assumptions:

- A user may fail to confirm several times with text. Once the signup is confirmed for a user, they will not be able to initiate the signup again.
- A user may not initiate the signup confirmation process at all.

emails Table:

Column Name	Type
email_id	integer
user_id	integer
signup_date	datetime

emails Example Input:

email_id	user_id	signup_date
125	7771	06/14/2022 00:00:00

236	6950	07/01/2022 00:00:00
433	1052	07/09/2022 00:00:00

texts Table:

Column Name	Type
text_id	integer
email_id	integer
signup_action	varchar

texts Example Input:

text_id	email_id	signup_action
6878	125	Confirmed
6920	236	Not Confirmed
6994	236	Confirmed

Example Output:

confirm_rate
0.67

```
2652
2653 •   select
2654     round(count(case when signup_action='Confirmed' then 1 end)/count(user_id),2) as confirmation
2655      from emails
2656     left join texts on texts.email_id=emails.email_id
2657     where signup_action is not null
2658
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

confirmation
0.67

Result 190

Q98.

The table below contains information about tweets over a given period of time. Calculate the 3-day rolling average of tweets published by each user for each date that a tweet was posted. Output the user id, tweet date, and rolling averages rounded to 2 decimal places.
Hint- Use Count and group by

Important Assumptions:

- Rows in this table are *consecutive* and ordered by date.
- Each row represents a different day
- A day that does not correspond to a row in this table is not counted. The most recent day is the next row above the current row.

Note: Rolling average is a metric that helps us analyze data points by creating a series of averages based on different subsets of a dataset. It is also known as a moving average, running average, moving mean, or rolling mean.

tweets Table:

Column Name	Type
tweet_id	integer
user_id	integer
tweet_date	timestamp

tweets Example Input:

tweet_id	user_id	tweet_date
214252	111	06/01/2022 12:00:00
739252	111	06/01/2022 12:00:00
846402	111	06/02/2022 12:00:00
241425	254	06/02/2022 12:00:00
137374	111	06/04/2022 12:00:00

Example Output:

user_id	tweet_date	rolling_avg_3days
111	06/01/2022 12:00:00	2.00
111	06/02/2022 12:00:00	1.50
111	06/04/2022 12:00:00	1.33
254	06/02/2022 12:00:00	1.00

```
2677  select user_id,tweet_date,avg(total_tweets) over (partition by user_id order by tweet_date rows  between 3 preceding and current row ) as
      3_day_rolling_avg from (
2678  select user_id,tweet_date,count(tweet_date) as total_tweets
2679  from tweets
2680  group by 1,2
2681  order by 1
2682  )b
2683
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

user_id	tweet_date	3_day_rolling_avg
111	2022-06-01 12:00:00	2.0000
111	2022-06-02 12:00:00	1.5000
111	2022-06-04 12:00:00	1.3333
254	2022-06-02 12:00:00	1.0000

Result Grid

Result 196 | Read Only

Q99.

Assume you are given the tables below containing information on Snapchat users, their ages, and their time spent sending and opening snaps. Write a query to obtain a breakdown of the time spent sending vs. opening snaps (as a percentage of total time spent on these activities) for each age group.

Hint- Use join and case

Output the age bucket and percentage of sending and opening snaps. Round the percentage to 2 decimal places.

Notes:

- You should calculate these percentages:
  - time sending / (time sending + time opening)
  - time opening / (time sending + time opening)
- To avoid integer division in percentages, multiply by 100.0 and not 100.

activities Table:

Column Name	Type
activity_id	integer
user_id	integer
activity_type	string ('send', 'open', 'chat')
time_spent	float
activity_date	datetime

activities Example Input:

activity_id	user_id	activity_type	time_spent	activity_date
7274	123	open	4.50	06/22/2022 12:00:00
2425	123	send	3.50	06/22/2022 12:00:00
1413	456	send	5.67	06/23/2022 12:00:00
1414	789	chat	11.00	06/25/2022 12:00:00
2536	456	open	3.00	06/25/2022 12:00:00

age_breakdown Table:

Column Name	Type
user_id	integer
age_bucket	string ('21-25', '26-30', '31-25')

age_breakdown Example Input:

user_id	age_bucket
123	31-35
456	26-30
789	21-25

Example Output:

age_bucket	send_perc	open_perc
26-30	65.40	34.60
31-35	43.75	56.25

```
2715 •  select age_bucket,round(sum(case when activity_type='SEND' then time_spent end)/sum(time_spent)*100,2) as sent_per,round(sum(case when
        activity_type='OPEN' then time_spent end)/sum(time_spent)*100,2) as open_per from activities
2716    left join age_breakdown on age_breakdown.user_id=activities.user_id
2717    where activity_type<>'CHAT'
2718    group by 1
2719
2720
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

age_bucket	sent_per	open_per
31-35	43.75	56.25
26-30	65.4	34.6

Q100 .

The LinkedIn Creator team is looking for power creators who use their personal profile as a company or influencer page. This means that if someone's Linkedin page has more followers than all the companies they work for, we can safely assume that person is a Power Creator. Keep in mind that if a person works at multiple companies, we should take into account the company with the most followers.

Level - Medium
Hint- Use join and group by

Write a query to return the IDs of these LinkedIn power creators in ascending order.

Assumptions:

- A person can work at multiple companies.
- In the case of multiple companies, use the one with largest follower base.

personal_profiles Table:

Column Name	Type
profile_id	integer
name	string
followers	integer

personal_profiles Example Input:

profile_id	name	followers
1	Nick Singh	92,000
2	Zach Wilson	199,000
3	Daliana Liu	171,000
4	Ravit Jain	107,000
5	Vin Vashishta	139,000
6	Susan Wojcicki	39,000

employee_company Table:

Column Name	Type
personal_profile_id	integer
company_id	integer

employee_company Example Input:

personal_profile_id	company_id
1	4
1	9
2	2
3	1
4	3
5	6
6	5

company_pages Table:

Column Name	Type
company_id	integer

name	string
followers	integer

company_pages Example Input:

company_id	name	followers
1	The Data Science Podcast	8,000
2	Airbnb	700,000
3	The Ravit Show	6,000
4	DataLemur	200
5	YouTube	1,6000,000
6	DataScience.Vin	4,500
9	Ace The Data Science Interview	4479

Example Output:

profile_id
1
3
4
5

```
2775 • select * from (
2776   select personal_profiles.profile_id,personal_profiles.name,sum(distinct personal_profiles.followers) as personal_followers,max(
       company_pages.followers) as company_followers
2777    from employee_company
2778   left join personal_profiles on personal_profiles.profile_id=employee_company.personal_profile_id
2779   left join company_pages on company_pages.company_id=employee_company.company_id
2780   group by 1,2
2781   )b
2782   where personal_followers>company_followers
2783
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

profile_id	name	personal_followers	company_followers
1	NICK SINGH	92000	4479
3	DALIANA LIU	171000	8000
4	RAVIT JAIN	107000	6000
5	VIN VASHISHTA	139000	4500