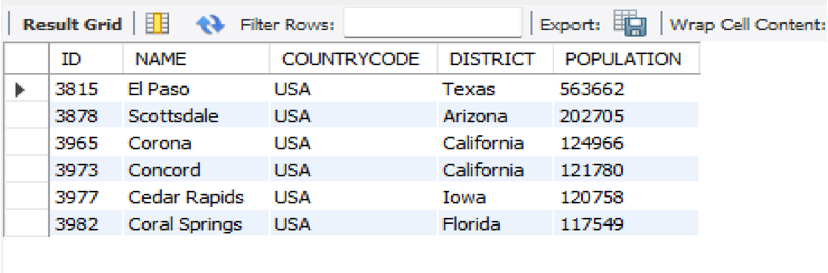
DATASET

https://docs.google.com/spreadsheets/d/1dk9kRwcMxj5USuJqxtITD05S-aOUD6fzNzVW41dcpgc/edit?usp=sharing

Q1. Query all columns for all American cities in the CITY table with populations larger than 100000. The CountryCode for America is USA.

Solution:

```
SELECT  *
        FROM city
WHERE COUNTRYCODE ='USA' AND POPULATION > 100000;
```

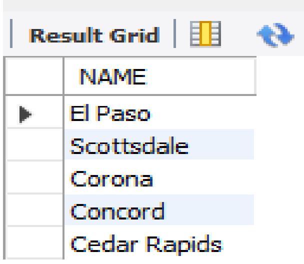


ID	NAME	COUNTRYCODE	DISTRICT	POPULATION
3815	El Paso	USA	Texas	563662
3878	Scottsdale	USA	Arizona	202705
3965	Corona	USA	California	124966
3973	Concord	USA	California	121780
3977	Cedar Rapids	USA	Iowa	120758
3982	Coral Springs	USA	Florida	117549

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

Solution:

```
SELECT
     NAME
FROM City
WHERE COUNTRYCODE='USA' AND POPULATION > 120000;
```



NAME
El Paso
Scottsdale
Corona
Concord
Cedar Rapids

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

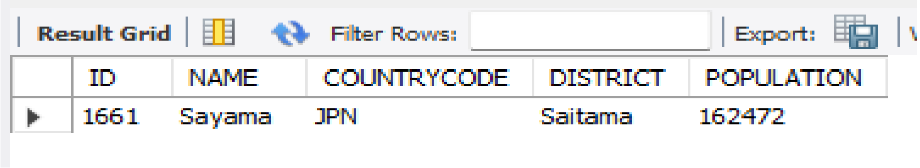
Solution:

```
SELECT * FROM City;
```

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

Solution:

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

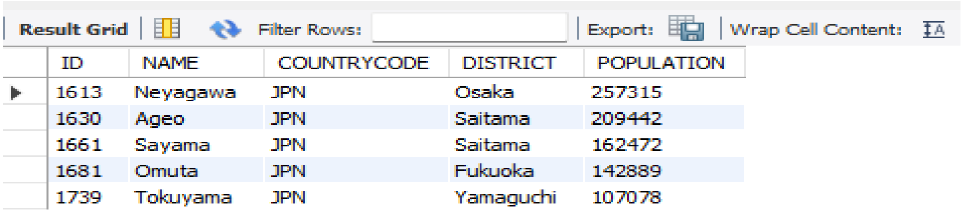


ID	NAME	COUNTRYCODE	DISTRICT	POPULATION
1661	Sayama	JPN	Saitama	162472

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

Solution:

```
SELECT *
FROM CITY
WHERE COUNTRYCODE = 'JPN';
```

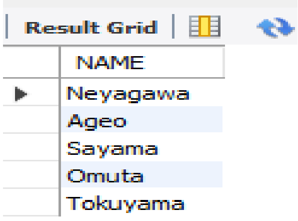


ID	NAME	COUNTRYCODE	DISTRICT	POPULATION
1613	Neyagawa	JPN	Osaka	257315
1630	Ageo	JPN	Saitama	209442
1661	Sayama	JPN	Saitama	162472
1681	Omuta	JPN	Fukuoka	142889
1739	Tokuyama	JPN	Yamaguchi	107078

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

Solution:

```
SELECT
      NAME
FROM city
WHERE COUNTRYCODE = 'JPN';
```



NAME
Neyagawa
Ageo
Sayama
Omuta
Tokuyama

DATASET

https://docs.google.com/spreadsheets/d/1sHPhE7walQD5mL7ppFNqybyoOJY3E51N0cWYzhp2UH4/edit?usp=sharing

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

Solution:

```
SELECT
        CITY,
        STATE
from Station;
```

Q8. Query a list of CITY names from STATION for cities that have an even ID number. Print the results in any order but exclude duplicates from the answer.

Solution:

Method 1:

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

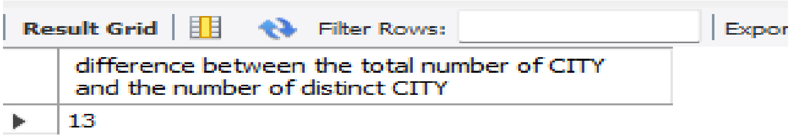
Method 2:

```
SELECT
   DISTINCT (CITY)
FROM Station
WHERE MOD(ID,2)=0
ORDER BY CITY;
```

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

Solution:

```
SELECT
        COUNT(CITY) - COUNT(DISTINCT(CITY))  as 'difference between
        the total number of CITY and the number of distinct CITY'
FROM Station;
```

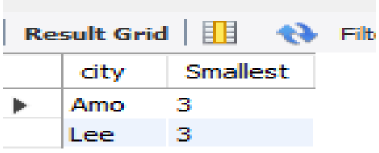


difference between the total number of CITY and the number of distinct CITY
13

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.

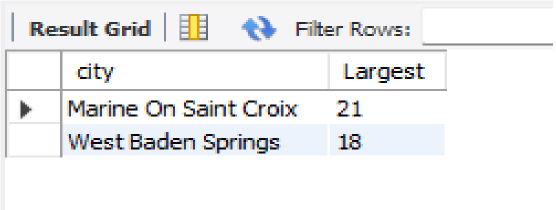
Solution:

```
SELECT
        CITY,
        LENGTH(CITY) as Smallest
FROM Station
ORDER BY LENGTH(CITY) asc, CITY asc LIMIT 2;
```



city	Smallest
Amo	3
Lee	3

```
SELECT
        CITY,
        LENGTHCITY) as Largest
FROM Station
ORDER BY LENGTH(CITY) desc, CITY asc LIMIT 2;
```



city	Largest
Marine On Saint Croix	21
West Baden Springs	18

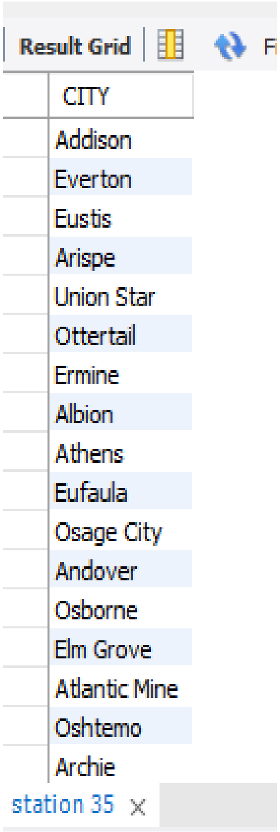
Q11. Query the list of CITY names starting with vowels (i.e., a, e, i, o, or u) from STATION. Your result cannot contain duplicates.

Solution 1:

```
SELECT
    DISTINCT City
FROM Station
where CITY like 'A%' or CITY like 'a%' or CITY like 'E%' or CITY LIKE 'e%'
or CITY like 'I%' or CITY like 'i%' or CITY like 'O%' or CITY like 'o%' or
CITY like 'U%' or CITY like 'u%';
```

Solution 2:

```
SELECT
        CITY
FROM station
WHERE CITY  RLIKE '^[AEIOUaeiou].*$';
```



```
CITY
Addison
Everton
Eustis
Arispe
Union Star
Ottertail
Ermine
Albion
Athens
Eufaula
Osage City
Andover
Osborne
Elm Grove
Atlantic Mine
Oshtemo
Archie
```

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

Solution:

```
SELECT
        DISTINCT city
FROM Station
WHERE CITY like '%A' or CITY like '%a' or CITY like '%E' or CITY LIKE
'%e' or CITY like '%I' or CITY like '%i' or CITY like '%o' or CITY like '%O' or
CITY like '%U' or CITY like '%u';
```

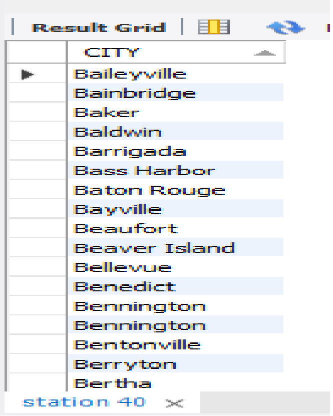
Q13. Query the list of CITY names from STATION that do not start with vowels. Your result cannot contain duplicates.

Solution 1:

```
SELECT
        DISTINCT CITY
FROM STATION
where CITY not like 'A%' and CITY not like 'a%' and CITY not like 'E%' and
CITY not like 'e%' and CITY not like 'I%' and CITY not like 'i%' and CITY not
like 'o%' and CITY not like 'O%' and CITY not like 'U%' and CITY not like
'u%';
```

Solution 2:

```
SELECT
     CITY
FROM station
WHERE CITY  NOT  RLIKE '^[AEIOUaeiou].*$';
```

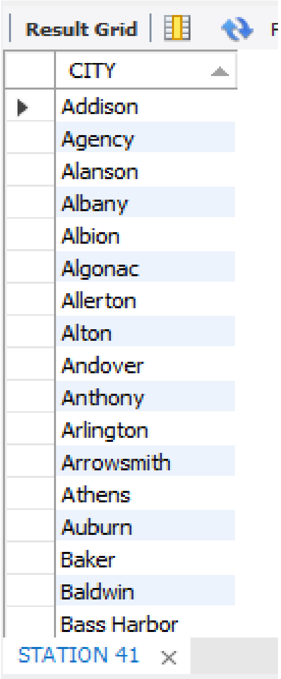


CITY
Baileyville
Bainbridge
Baker
Baldwin
Barrigada
Bass Harbor
Baton Rouge
Bayville
Beaufort
Beaver Island
Bellevue
Benedict
Bennington
Bennington
Bentonville
Berryton
Bertha

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

Solution:

```
SELECT
        DISTINCT CITY
FROM STATION
where CITY not like '%A' and CITY not like '%a' and CITY not like '%E' and
CITY not LIKE '%e' and CITY not like '%I' and CITY not like '%i' and CITY
not like '%o' and CITY not like '%O' and CITY not like '%U' and CITY not like
'%u';
```

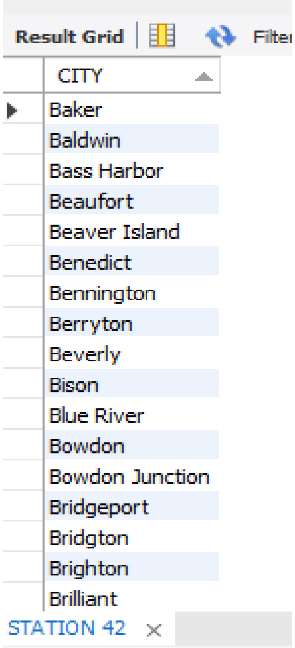


```
CITY
Addison
Agency
Alanson
Albany
Albion
Algonac
Allerton
Alton
Andover
Anthony
Arlington
Arrowsmith
Athens
Auburn
Baker
Baldwin
Bass Harbor
```

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.

Solution:

```
SELECT
        DISTINCT CITY
FROM STATION
where( CITY not like 'A%' and CITY not like 'a%' and CITY not like 'E%' and
CITY not like 'e%' and CITY not like 'I%' and CITY not like 'i%' and CITY not
like 'o%' and CITY not like 'O%' and CITY not like 'U%' and CITY not like
'u%'
AND
CITY not like '%A' and CITY not like '%a' and CITY not like '%E' and CITY
not LIKE '%e' and CITY not like '%I' and CITY not like '%i' and CITY not like
'%o' and CITY not like '%O' and CITY not like '%U' and CITY not like '%u');
```

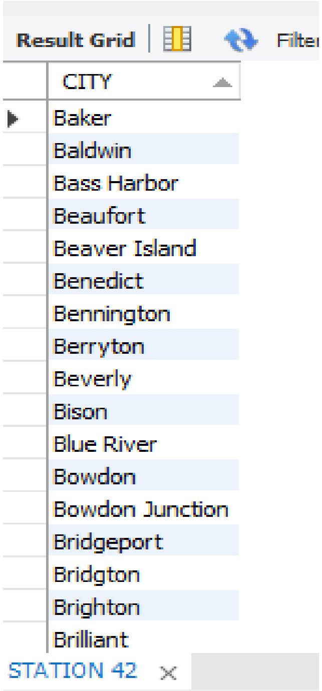


CITY
Baker
Baldwin
Bass Harbor
Beaufort
Beaver Island
Benedict
Bennington
Berryton
Beverly
Bison
Blue River
Bowdon
Bowdon Junction
Bridgeport
Bridgton
Brighton
Brilliant

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.

Solution:

```
SELECT DISTINCT CITY
FROM STATION
where( CITY not like 'A%' and CITY not like 'a%' and CITY not like 'E%' and
CITY not like 'e%' and CITY not like 'I%' and CITY not like 'i%' and CITY not
like 'o%' and CITY not like 'O%' and CITY not like 'U%' and CITY not like
'u%'
and
CITY not like '%A' and CITY not like '%a' and CITY not like '%E' and CITY
not LIKE '%e' and CITY not like '%I' and CITY not like '%i' and CITY not like
'%o' and CITY not like '%O' and CITY not like '%U' and CITY not like '%u');
```



CITY
Baker
Baldwin
Bass Harbor
Beaufort
Beaver Island
Benedict
Bennington
Berryton
Beverly
Bison
Blue River
Bowdon
Bowdon Junction
Bridgeport
Bridgton
Brighton
Brilliant

Q17. DATASET

TABLE 1: Product

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

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

TABLE 2: Sales

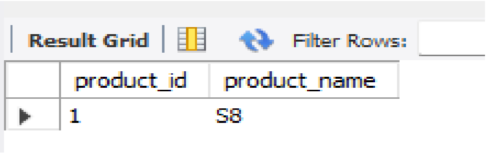
seller_id	product_id	buyer_id	sale_date	quantity	price
1	1	1	2019-01-21	2	2000
1	2	2	2019-02-17	1	800
2	2	3	2019-06-02	1	800
3	3	4	2019-05-13	2	2800

This table has no primary key, it can have repeated rows.

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. Return the result table in any order.

Solution:

```
SELECT
        p.product_id, p.product_name
FROM Product p
JOIN Sales s
ON p.product_id = s.product_id
GROUP BY s.product_id
HAVING MIN(s.sale_date) >= "2019-01-01" AND MAX(s.sale_date) <=
"2019-03-31";
```



product_id	product_name
1	S8

Q18. DATASET

TABLE: Views

article_id	author_id	viewer_id	view_date
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

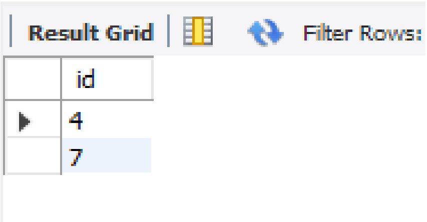
There is no primary key for this table, it may have duplicate rows. Each row of this table indicates that some viewer viewed an article (written by some author) on some date.

Note that equal author_id and viewer_id indicate the same person.

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.
Each row of this table indicates that some viewer viewed an article (written by some author) on some date. Note that equal author_id and viewer_id indicate the same person.

Solution:

```
SELECT
	DISTINCT author_id as id
FROM Views
WHERE author_id = viewer_id
ORDER BY id ASC ;
```



id
4
7

Q19. DATASET

TABLE: Delivery

delivery_id	customer_id	order_date	customer_pref_delivery_date
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

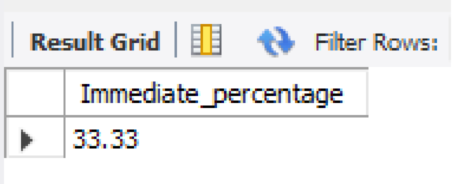
delivery_id is the primary key of this table.

The table holds information about food delivery to customers that make orders at some date and specify a preferred delivery date (on the same order date or after it). If the customer's preferred delivery date is the same as the order date, then the order is called immediately, otherwise, it is called scheduled.

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

Solution:

```
SELECT
ROUND
(
        (SELECT COUNT(delivery_id)
        FROM Delivery
        WHERE order_date = customer_pref_delivery_date)/(count(delivery_id))
        *100,  2 ) as Immediate_percentage
from Delivery;
```



	Immediate_percentage
▶	33.33

Q20. DATASET

TABLE: Ads

ad_id	user_id	action
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

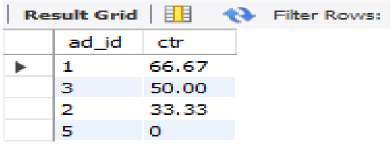
(ad_id, user_id) is the primary key for this table.
Each row of this table contains the ID of an Ad, the ID of a user, and the action taken by this user regarding this Ad. The action column is an ENUM type of ('Clicked', 'Viewed', 'Ignored').
A company is running Ads and wants to calculate the performance of each Ad.
Performance of the Ad is measured using Click-Through Rate (CTR) where:

$$CTR = \begin{cases} 0, & \text{if Ad total clicks} + \text{Ad total views} = 0 \\ \frac{\text{Ad total clicks}}{\text{Ad total clicks} + \text{Ad total views}} \times 100, & \text{otherwise} \end{cases}$$

Write an SQL query to find the ctr of each Ad. Round ctr to two decimal points. Return the result table ordered by ctr in descending order and by ad_id in ascending order in case of a tie.

Solution:

```
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) tmp
ORDER BY ctr desc, ad_id;
```



```
ad_id  ctr
1      66.67
3      50.00
2      33.33
5      0
```

Q21. DATASET

TABLE: Employee

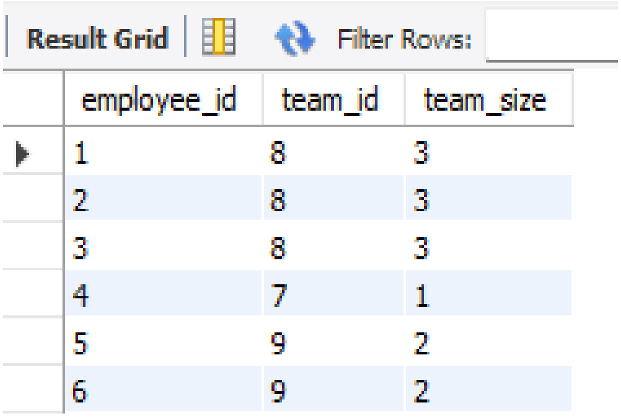
employee_id	team_id
1	8
2	8
3	8
4	7
5	9
6	9

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.

Solution:

```
SELECT
   employee_id,
   team_id,
   COUNT(1) OVER (PARTITION BY team_id) AS team_size
FROM Employee
ORDER BY employee_id;
```



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

Q22. DATASET

TABLE 1: Countries

country_id	country_name
2	USA
3	Australia
7	Peru
5	China
8	Morocco
9	Spain

country_id is the primary key for this table.
Each row of this table contains the ID and the name of one country.

TABLE 2: Weather

country_id	weather_state	day
2	15	01-11-2019
2	12	28-10-2019
2	12	27-10-2019
3	-2	10-11-2019
3	0	11-11-2019
3	3	12-11-2019
5	16	07-11-2019
5	18	09-11-2019
5	21	23-11-2019
7	25	28-11-2019
7	22	01-12-2019
7	20	02-12-2019
8	25	05-11-2019
8	27	15-11-2019
8	31	25-11-2019
9	7	23-10-2019
9	3	23-12-2019

(country_id, day) is the primary key for this table.
Each row of this table indicates the weather state in a country for one day.

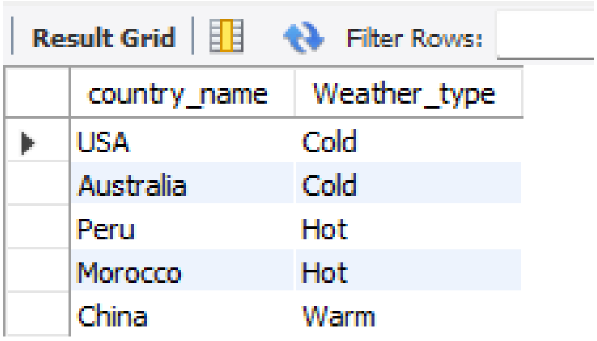
Write an SQL query to find the type of weather in each country for November 2019.
The type of weather is:

- **Cold if the average weather_state is less than or equal 15,**
- **Hot if the average weather_state is greater than or equal to 25, and**
- **Warm otherwise.**

Return result table in any order

Solution:

```
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 c
JOIN weather w
on c.country_id = w.country_id
WHERE month(day) =11
GROUP BY country_name;
```



country_name	Weather_type
USA	Cold
Australia	Cold
Peru	Hot
Morocco	Hot
China	Warm

Q23. DATASET

TABLE 1: Prices

product_id	start_date	end_date	price
1	43513	43524	5
1	43525	43546	20
2	43497	43516	15
2	43517	43555	30

(product_id, start_date, end_date) is the primary key for this table.
Each row of this table indicates the price of the product_id in the period from start_date to end_date. For each product_id there will be no two overlapping periods. That means there will be no two intersecting periods for the same product_id.

TABLE 1: UnitsSold

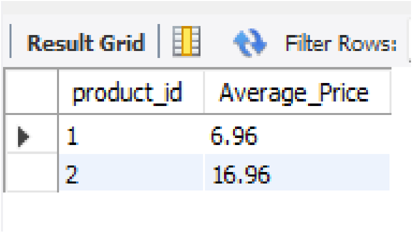
product_id	purchase_date	units
1	25-02-2019	100
1	01-03-2019	15
2	10-02-2019	200
2	22-03-2019	30

There is **no primary key for this table**, it may contain duplicates.
Each row of this table indicates the date, units, and product_id of each product sold.

Write an SQL query to find the average selling price for each product. average_price should be rounded to 2 decimal places. Return the result table in any order.

Solution:

```
SELECT
        product_id,
        ROUND(SUM(Total_Price) / SUM(Units),2) as Average_Price
FROM
          (Select
          p.price * u.units as Total_Price,
          p.product_id,
          u.units
          FROM
          Prices p join UnitsSold u
          on p.product_id = u.product_id
          and
          u.purchase_date between p.start_date and p.end_date) tmp
GROUP BY product_id;
```



product_id	Average_Price
1	6.96
2	16.96

Q24. DATASET

TABLE: Activity

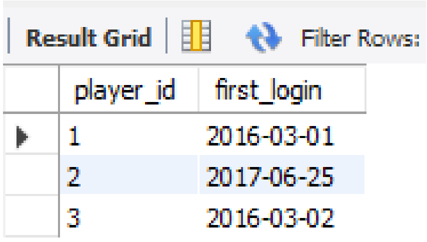
player_id	device_id	event_date	games_played
1	2	2016-03-01	5
1	2	2016-05-02	6
2	3	2017-06-25	1
3	1	2016-03-02	0
3	4	2018-07-03	5

(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 first login date for each player.
Return the result table in any order.

Solution:

```
SELECT
	player_id,
	event_date as first_login
FROM Activity
GROUP BY player_id
HAVING MIN(event_date);
```



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

Q25. DATASET

TABLE: Activity

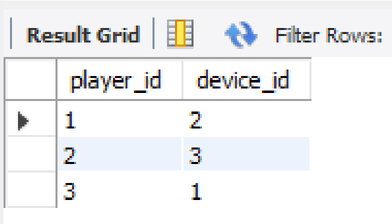
player_id	device_id	event_date	games_played
1	2	2016-03-01	5
1	2	2016-05-02	6
2	3	2017-06-25	1
3	1	2016-03-02	0
3	4	2018-07-03	5

(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.

Solution:

```
SELECT
        player_id,
        device_id
FROM Activity
GROUP BY player_id
HAVING MIN(event_date);
```



player_id	device_id
1	2
2	3
3	1

Q26. DATASET

TABLE 1: Products

product_id	product_name	product_category
1	Leetcode Solutions	Book
2	Jewels of Stringology	Book
3	HP	Laptop
4	Lenovo	Laptop
5	Leetcode Kit	T-shirt

product_id is the primary key for this table.
This table contains data about the company's products.

TABLE 2: Orders

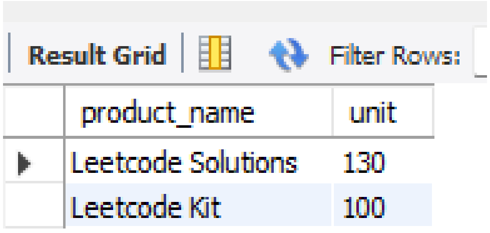
product_id	order_date	unit
1	2020-02-05	60
1	2020-02-10	70
2	2020-01-18	30
2	2020-02-11	80
3	2020-02-17	2
3	2020-02-24	3
4	2020-03-01	20
4	2020-03-04	30
4	2020-03-04	60
5	2020-02-25	50
5	2020-02-27	50
5	2020-03-01	50

There is **no primary key** for this table. It may have duplicate rows.
product_id is a foreign key to the Products table.
unit is the number of products ordered in order_date.

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

Solution:

```
SELECT
        product_name,
       SUM(unit) as unit
FROM
Products p
JOIN Orders o
ON p.product_id = o.product_id
WHERE month(order_date)=2
GROUP BY product_name
HAVING SUM(unit) >= 100;
```



product_name	unit
Leetcode Solutions	130
Leetcode Kit	100

Q27. DATASET

TABLE: Users

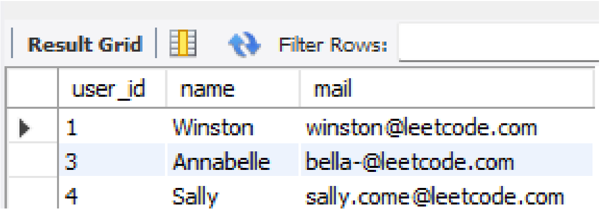
user_id	name	mail
1	Winston	winston@leetcode.com
2	Jonathan	jonathanisgreat
3	Annabelle	bella-@leetcode.com
4	Sally	sally.come@lee tcode.com
5	Marwan	quarz#2020@le etcode.com
6	David	david69@gmail.com
7	Shapiro	.shapo@leetcode.com

user_id is the primary key for this table.
This table contains information of the users signed up in a website. Some emails are invalid.

Write an SQL query to find the users who have valid emails.
A valid e-mail has a prefix name and a domain where:
● The prefix name is a string that may contain letters (upper or lower case), digits, underscore
'_', period '.', and/or dash '-'. The prefix name must start with a letter.
● The domain is '@leetcode.com'.
Return the result table in any order.

Solution:

```
SELECT *
FROM Users
WHERE mail REGEXP '^[a-zA-Z][[:alnum:]_.-]*@leetcode\.com$';
```



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

Q28. DATASET

TABLE 1: Customers

customer_id	name	country
1	Winston	USA
2	Jonathan	Peru
3	Moustafa	Egypt

customer_id is the primary key for this table.
This table contains information about the customers in the company.

TABLE 2: Product

product_id	description	price
10	LC Phone	300
20	LC T-Shirt	10
30	LC Book	45
40	LC Keychain	2

product_id is the primary key for this table. This table contains information on the products in the company. price is the product cost.

TABLE 3: Orders

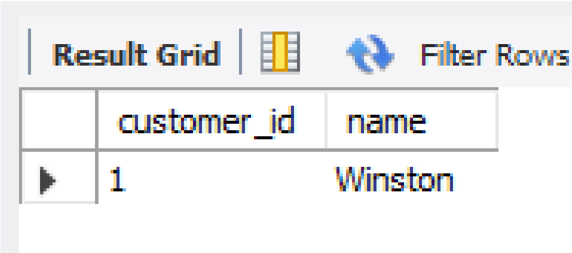
order_id	customer_id	product_id	order_date	quantity
1	1	10	2020-06-10	1
2	1	20	2020-07-01	1
3	1	30	2020-07-08	2
4	2	10	2020-06-15	2
5	2	40	2020-07-01	10
6	3	20	2020-06-24	2
7	3	30	2020-06-25	2
9	3	30	2020-05-08	3

order_id is the primary key for this table. This table contains information on customer orders. customer_id is the id of the customer who bought "quantity" products with id "product_id". Order_date is the date in format **('YYYY-MM-DD')** when the order was shipped.

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. Return the result table in any order.

Solution:

```
SELECT
        c.customer_id,
        name
FROM Customers c
JOIN Orders o
ON c.customer_id = o.customer_id
JOIN product p
ON p.product_id = o.product_id
GROUP BY customer_id
HAVING SUM(IF( month(order_date) =06, quantity, 0) * price) >= 100
AND SUM(IF( month(order_date) =07, quantity, 0) * price) >= 100;
```



customer_id	name
1	Winston

Q29. DATASET

TABLE 1: TVProgram

program_date	content_id	channel
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

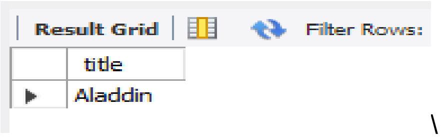
(program_date, content_id) is the primary key for this table.
This table contains information about the programs on the TV.
content_id is the id of the program in some channel on the TV.

TABLE 2: Content

content_id	title	Kids_content	content_type
1	Leetcode Movie	N	Movies
2	Alg. for Kids	Y	Series
3	Database Sols	N	Series
4	Aladdin	Y	Movies
5	Cinderella	Y	Movies

content_id is the primary key for this table. Kids_content is an enum that takes one of the values ('Y', 'N') where: ' Y' means content for kids, otherwise 'N' is not content for kids. content_type is the category of the content as movies, series, etc.
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.
Solution:

```
SELECT
        DISTINCT title
FROM content c
JOIN TVProgram t
ON c.content_id=t.content_id
WHERE kids_content = 'Y'
AND content_type like '%Movies%'
AND month(program_date) = '06';
```



title
Aladdin

\

Q30. DATASET

TABLE: NPV

id	year	npv
1	2018	100
7	2020	30
13	2019	40
1	2019	113
2	2008	121
3	2009	12
11	2020	99
7	2019	0

(id, year) is the primary key of this table. The table has information about the id and the year of each inventory and the corresponding net present value.

TABLE: Queries

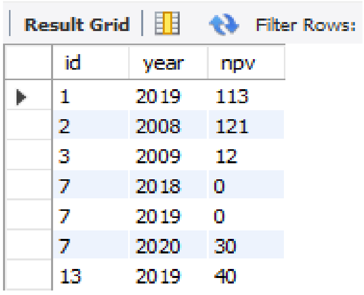
id	year
1	2019
2	2008
3	2009
7	2018
7	2019
7	2020
13	2019

(id, year) is the primary key of this table.
The table has information about the id and the year of each inventory query.

Write an SQL query to find the npv of each query of the Queries table. Return the result table in any order.

Solution:

```
SELECT
         Q.id,Q.year,
          ifnull(npv,0) as npv
FROM Queries Q
LEFT JOIN NPV N
on Q.id=N.id and Q.year=N.year
ORDER BY Q.id;
```

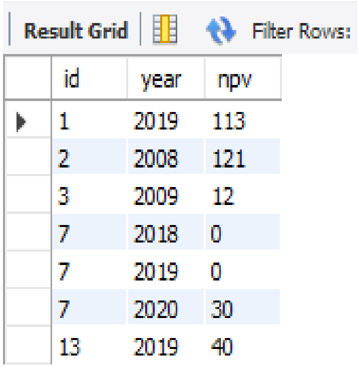


id	year	npv
1	2019	113
2	2008	121
3	2009	12
7	2018	0
7	2019	0
7	2020	30
13	2019	40

Q31. Write an SQL query to find the npv of each query of the Queries table.
Return the result table in any order.

Solution:

```
SELECT
       Q.id,Q.year, ifnull(npv,0) as npv
FROM Queries Q
LEFT JOIN NPV N
on Q.id=N.id and Q.year=N.year
ORDER BY Q.id;
```



id	year	npv
1	2019	113
2	2008	121
3	2009	12
7	2018	0
7	2019	0
7	2020	30
13	2019	40

Q32. DATASET

TABLE 1: Employees

id	name
1	Alice
7	Bob
11	Meir
90	Winston
3	Jonathan

id is the primary key for this table. Each row of this table contains the id and the name of an employee in a company.

TABLE 2: EmployeeUNI

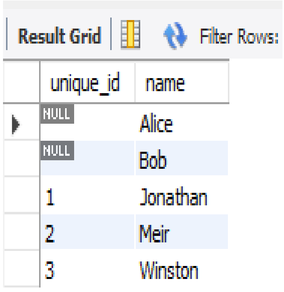
id	unique_id
3	1
11	2
90	3

(id, unique_id) is the primary key for this table. Each row of this table **contains** the id and the corresponding unique id of an employee in the company.

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. Return the result table in any order.

Solution:

```
SELECT
	EU.unique_id,
	E.name
FROM EmployeeUNI EU
RIGHT JOIN Employees E
ON EU.id = E.id
ORDER by  EU.unique_id;
```



```
unique_id  name
NULL       Alice
NULL       Bob
1          Jonathan
2          Meir
3          Winston
```

Q33. DATASET

TABLE 1: Users

id	name
1	Alice
2	Bob
3	Alex
4	Donald
7	Lee
13	Jonathan
19	Elvis

id is the primary key for this table.
name is the name of the user.

TABLE 2: Rides

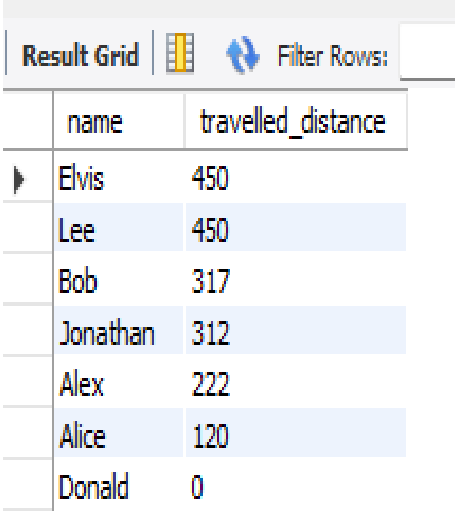
id	user_id	distance
1	1	120
2	2	317
3	3	222
4	7	100
5	13	312
6	19	50
7	7	120
8	19	400
9	7	230

id is the primary key for this table.
user_id is the id of the user who travelled the distance "distance".

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.

Solution:

```
SELECT
        U.name,
         ifNULL(sum(R.distance),0) as travelled_distance
FROM Users U
LEFT JOIN Rides R
ON U.id=R.user_id
GROUP BY U.id
ORDER BY travelled_distance DESC, name;
```



name	travelled_distance
Elvis	450
Lee	450
Bob	317
Jonathan	312
Alex	222
Alice	120
Donald	0

Q35. DATASET

TABLE 1: Movies

movie_id	title
1	Avengers
2	Frozen 2
3	Joker

movie_id is the primary key for this table.
The title is the name of the movie.

TABLE 2: Users

user_id	name
1	Daniel
2	Monica
3	Maria
4	James

user_id is the primary key for this table.

TABLE 3: MovieRating

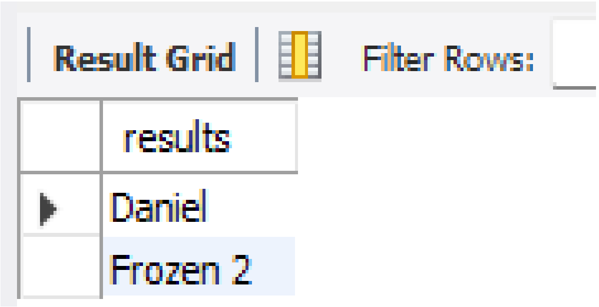
movie_id	user_id	rating	created_at
1	1	3	2020-01-12
1	2	4	2020-02-11
1	3	2	2020-02-12
1	4	1	2020-01-01
2	1	5	2020-02-17
2	2	2	2020-02-01
2	3	2	2020-03-01
3	1	3	2020-02-22
3	2	4	2020-02-25

(movie_id, user_id) is the primary key for this table.
This table contains the rating of a movie by a user in their review. created_at is the user's review date.

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

Solution:

```
(SELECT
        U.name AS results
FROM Users U
JOIN MovieRating MR
ON U.user_id = MR.user_id
GROUP BY U.user_id
ORDER BY count(MR.movie_id) desc, U.name LIMIT 1
)
UNION
(SELECT
        M.title as Movie_Name
FROM Movies M JOIN MovieRating MR
ON M.movie_id = MR.movie_id
WHERE month(created_at)=2
GROUP BY MR.movie_id
ORDER BY AVG(MR.rating) desc, M.title LIMIT 1
);
```



	results
▶	Daniel
	Frozen 2

Q38. DATASET

TABLE 1: Department

id	name
1	Electrical Engineering
7	Computer Engineering
13	Business Administration

id is the primary key of this table.
The table has information about the id of each department of a university.

TABLE 2: Student

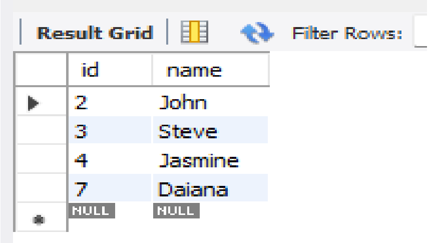
id	name	department_id
23	Alice	1
1	Bob	7
5	Jennifer	13
2	John	14
4	Jasmine	77
3	Steve	74
6	Luis	1
8	Jonathan	7
7	Daiana	33
11	Madelynn	1

id is the primary key of this table. The table has information about the id of each student at a university and the id of the department he/she studies at.

Write an SQL query to find the id and the name of all students who are enrolled in departments that no longer exist. Return the result table in any order.

Solution:

```
SELECT
	id,
	name
FROM Students
WHERE department_id NOT IN (SELECT id FROM Departments)
ORDER BY id;
```



id	name
2	John
3	Steve
4	Jasmine
7	Daiana
NULL	NULL

Q39. DATASET

TABLE: Calls

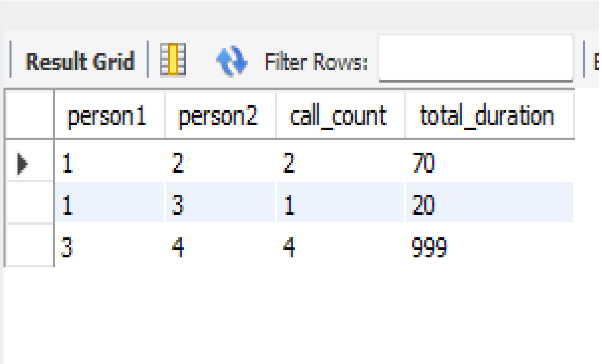
from_id	to_id	duration
1	2	59
2	1	11
1	3	20
3	4	100
3	4	200
3	4	200
4	3	499

This table does **not have a primary key**, it may contain duplicates.
This table contains the duration of a phone call between from_**id and to_id.**
from_id != to_id

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. Return the result table in any order.

Solution:

```
SELECT
	LEAST (from_id, to_id) AS person1,
	GREATEST(from_id, to_id) AS person2,
	COUNT(duration) AS call_count,
	SUM(duration) AS total_duration
FROM calls
GROUP BY person1, person2;
```



```
person1  person2  call_count  total_duration
1        2        2           70
1        3        1           20
3        4        4           999
```

Q41. DATASET

TABLE 1: Warehouse

name	product_id	units
LCHouse1	1	1
LCHouse1	2	10
LCHouse1	3	5
LCHouse2	1	2
LCHouse2	2	2
LCHouse3	4	1

(name, product_id) is the primary key for this table. Each row of this table contains the information of the products in each warehouse.

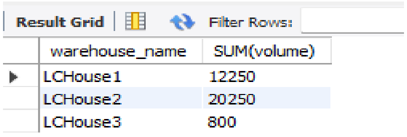
TABLE 2: Products

product_id	product_name	Width	Length	Height
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

product_id is the primary key for this table. Each row of this table contains information about the product dimensions (Width, Length, and Height)
in feet of each product.
Write an SQL query to report the number of cubic feet of volume the inventory occupies in each warehouse. Return the result table in any order.

Solution:

```
SELECT
        warehouse_name, SUM(volume)
         FROM
          (
          SELECT W.name AS warehouse_name,
           W.product_id,
           W.units * width * length * height as volume
            FROM Warehouse W LEFT JOIN Products P
            ON W.product_id = P.product_id) TMP
GROUP BY warehouse_name;
```



warehouse_name	SUM(volume)
LCHouse1	12250
LCHouse2	20250
LCHouse3	800

Q42. DATASET

TABLE: Sales

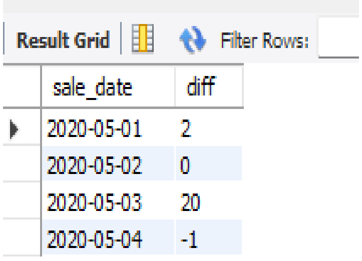
sale_date	fruit	sold_num
2020-05-01	apples	10
2020-05-01	oranges	8
2020-05-02	apples	15
2020-05-02	oranges	15
2020-05-03	apples	20
2020-05-03	oranges	0
2020-05-04	apples	15
2020-05-04	oranges	16

(sale_date, fruit) is the primary key for this table.
This table contains the sales of "apples" and "oranges" sold each day.

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.

Solution:

```
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;
```



sale_date	diff
2020-05-01	2
2020-05-02	0
2020-05-03	20
2020-05-04	-1

Q43. DATASET

TABLE: Activity

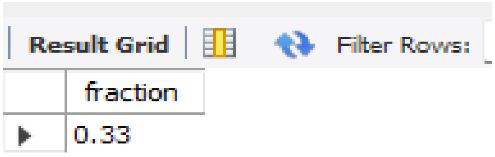
player_id	device_id	event_date	games_played
1	2	2016-03-01	5
1	2	2016-03-02	6
2	3	2017-06-25	1
3	1	2016-03-02	0
3	4	2018-07-03	5

(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.

Solution:

```
SELECT
        ROUND(SUM(CASE WHEN A1.event_date = A2.First_event +1 THEN
        1 ELSE 0 END) /count(DISTINCT A1.player_id), 2) AS fraction
        FROM Activity AS A1
        JOIN
        (
                SELECT player_id, MIN(event_date) AS First_event
                FROM Activity
                GROUP BY player_id ) AS A2
ON A1.player_id = A2.player_id;
```



```
fraction
0.33
```

Q44. DATASET

TABLE: Employee

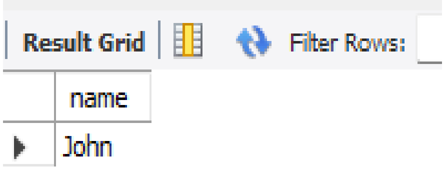
id	name	department	managerId
101	John	A	None
102	Dan	A	101
103	James	A	101
104	Amy	A	101
105	Anne	A	101
106	Ron	B	101

id is the primary key column for this table.
Each row of this table indicates the name of an employee, their department, and the id of theirmanager. If managerId is null, then the employee does not have a manager.No employee will be the manager of themself.

Write an SQL query to report the managers with at least five direct reports. Return the result table in any order.

Solution:

```
SELECT
        E.name
        FROM Employee E
        JOIN Employee E1
        ON (E.id = E1.managerid)
GROUP BY E.name
HAVING COUNT(*)>=5;
```



name
John

Q45. DATASET

TABLE 1: Student

student_id	student_name	gender	dept_id
1	Jack	M	1
2	Jane	F	1
3	Mark	M	2

student_id is the primary key column for this table.
dept_id is a foreign key to dept_id in the Department tables.
Each row of this table indicates the name of a student, their gender, and the id of their department.

TABLE 2: Department

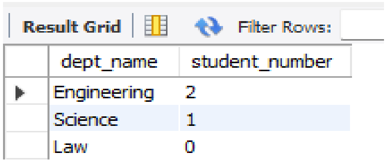
dept_id	dept_name
1	Engineering
2	Science
3	Law

dept_id is the primary key column for this table.
Each row of this table contains the id and the name of a department.

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 (even ones with no current students). Return the result table ordered by student_number in descending order. In case of a tie, order them by dept_name alphabetically.

Solution:

```
SELECT
        D.dept_name,
        COUNT(S.student_name) AS student_number
        FROM Department D LEFT JOIN Student S
        ON D. dept_id = S.dept_id
GROUP BY d.dept_name
ORDER BY student_number DESC;
```



dept_name	student_number
Engineering	2
Science	1
Law	0

Q46. DATASET

TABLE 1: Customer

customer_id	product_key
1	5
2	6
3	5
3	6
1	6

There is **no primary key** for this table. It may contain duplicates.
product_key is a foreign key to the Product table.

TABLE 2: Product

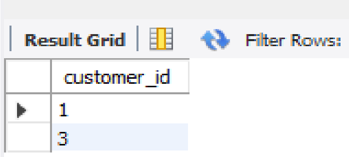
product_key
5
6

product_key is the primary key column for this table.

Write an SQL query to report the customer ids from the Customer table that bought all the products in the Product table. Return the result table in any order.

Solution:

```
SELECT
        customer_id
        FROM Customer
GROUP BY customer_id
HAVING COUNT(DISTINCT product_key) = (SELECT
COUNT(product_key) FROM product);
```



customer_id
1
3

Q47. DATASET

TABLE 1: Project

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

(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 2: Employee

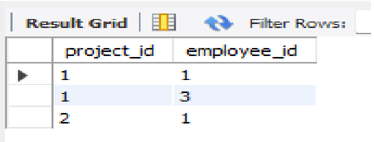
employee_id	name	experience_years
1	Khaled	3
2	Ali	2
3	John	3
4	Doe	2

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 most experienced employees in each project. In case of a tie, report all employees with the maximum number of experience years. Return the result table in any order.

Solution:

```
SELECT
        project_id,
        employee_id
        FROM
                (SELECT P.project_id, P.employee_id, E.experience_year,
                DENSE_RANK() OVER(PARTITION BY P.project_id ORDER
                BY E.experience_year DESC) AS rnk
                FROM project P JOIN Employee E
                ON P.employee_id = E.employee_id) tmp
WHERE rnk =1;
```



```
project_id  employee_id
1           1
1           3
2           1
```

Q49. DATASET

TABLE: Enrollments

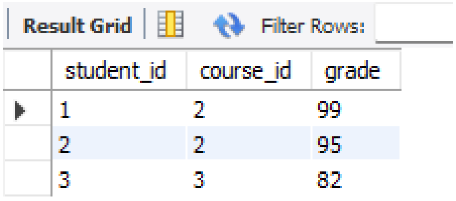
student_id	course_id	grade
2	2	95
2	3	95
1	1	90
1	2	99
3	1	80
3	2	75
3	3	82

(student_id, course_id) is the primary key of this table.

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

Solution:

```
SELECT
        student_id,
         course_id,
        grade
FROM
        (SELECT *,
                ROW_NUMBER() OVER(PARTITION BY student_id ORDER
                BY grade DESC, course_id ASC) AS 'rnk'
                FROM Enrollments) T
WHERE rnk=1;
```



student_id	course_id	grade
1	2	99
2	2	95
3	3	82

Q51. DATASET

TABLE: WORLD

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

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.

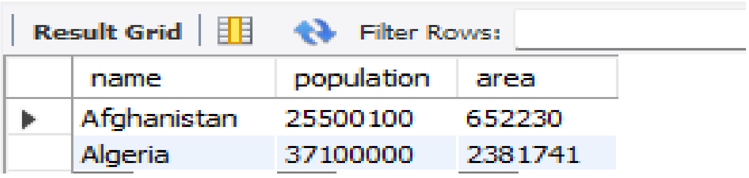
A country is big if:

- 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.

Solution:

```
SELECT
	name,
	population,
	area
FROM World
WHERE area >= 3000000 OR population >=25000000;
```



name	population	area
Afghanistan	25500100	652230
Algeria	37100000	2381741

Q52. DATASET

TABLE: Customer

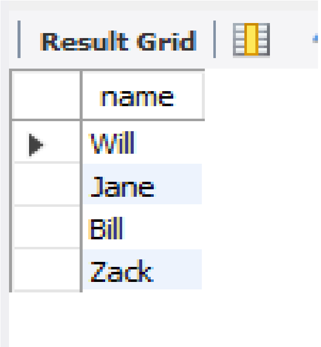
id	name	referee_id
1	Will	null
2	Jane	null
3	Alex	2
4	Bill	null
5	Zack	1
6	Mark	2

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.

Solution:

```
SELECT
        name
FROM customer
WHERE referee_id <> 2 OR referee_id is NULL;
```



```
name
Will
Jane
Bill
Zack
```

Q53. DATASET

TABLE 1: Customers

id	name
1	Joe
2	Henry
3	Sam
4	Max

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

TABLE 2: Orders

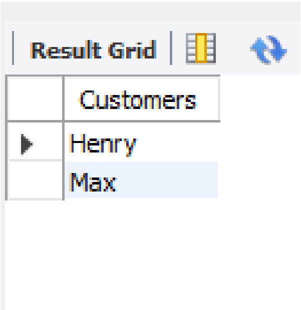
id	customerId
1	3
2	1

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.

Solution:

```
SELECT
        name AS Customers
FROM Customers
WHERE id NOT IN (SELECT Customerid FROM Orders);
```



Customers
Henry
Max

Q54. DATASET

TABLE: Employee

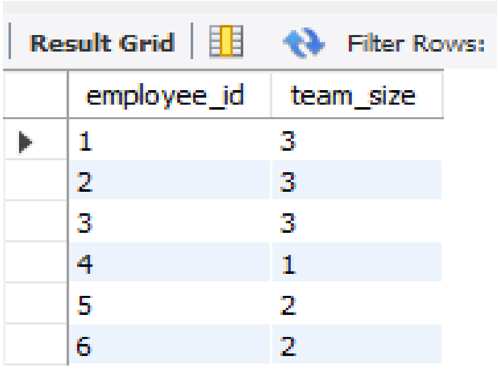
employee_id	team_id
1	8
2	8
3	8
4	7
5	9
6	9

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.

Solution:

```
SELECT
	employee_id,
	COUNT(team_id) OVER(PARTITION BY team_id) as team_size
FROM Employee
ORDER BY employee_id;
```



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

Q55. DATASET

TABLE 1: Person

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

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 2: Country

name	country_code
Peru	051
Israel	972
Morocco	212
Germany	49
Ethiopia	251

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 3: Calls

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

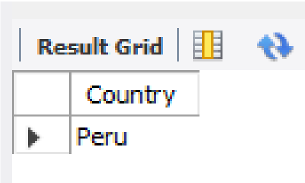
There is no primary key for this table, it may contain duplicates.
Each row of this table contains the caller id, callee id and the duration of the call in minutes. **caller_id != callee_id.**

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.

Solution:

```
SELECT
          C.name AS Country
          FROM Person P
          JOIN Calls Ca
          ON P.id = Ca.caller_id or P.id = Ca.callee_id
          JOIN Country C
          ON LEFT (P.phone_number,3) = C.country_code
GROUP BY C.name
HAVING AVG(duration) > (SELECT AVG(duration) FROM Calls);
```



```
Country
Peru
```

Q56. DATASET

TABLE: Activity

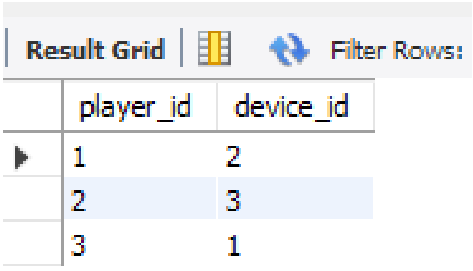
player_id	device_id	event_date	games_played
1	2	2016-03-01	5
1	2	2016-05-02	6
2	3	2017-06-25	1
3	1	2016-03-02	0
3	4	2018-07-03	5

(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.

Solution:

```
SELECT
        DISTINCT player_id,
        FIRST_VALUE(device_id) OVER(PARTITION BY player_id ORDER
        BY event_date ASC) AS device_id
FROM Activity;
```



player_id	device_id
1	2
2	3
3	1

Q57. DATASET

TABLE: Orders

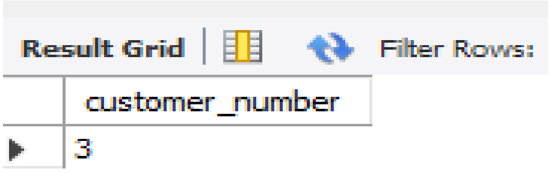
order_number	customer_number
1	1
2	2
3	3
4	3

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.

Solution:

```
SELECT
        customer_number
FROM Orders
GROUP BY customer_number
ORDER BY COUNT(customer_number) DESC LIMIT 1;
```



customer_number
3

Q58. DATASET

TABLE: Cinema

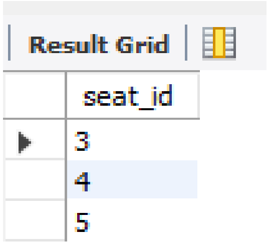
seat_id	free
1	1
2	0
3	1
4	1
5	1

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.

Solution:

```
SELECT
        seat_id
FROM
        (select seat_id, free,
        lead(free,1) over() as next,
        lag(free,1) over() as prev
        from cinema) tmp
WHERE tmp.free=1 AND (NEXT=1 OR PREV=1)
ORDER BY seat_id;
```



seat_id
3
4
5

Q59. DATASET

TABLE 1: SalesPerson

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

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 2: Company

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

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 3: Orders

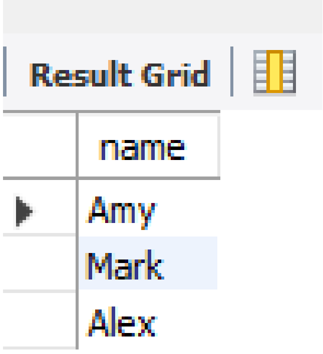
order_id	order_date	com_id	sales_id	amount
1	1/1/2014	3	4	10000
2	2/1/2014	4	5	5000
3	3/1/2014	1	1	50000
4	4/1/2014	1	4	25000

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".

Solution:

```
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'
);
```



```
name
Amy
Mark
Alex
```

Q60. DATASET

TABLE: Triangle

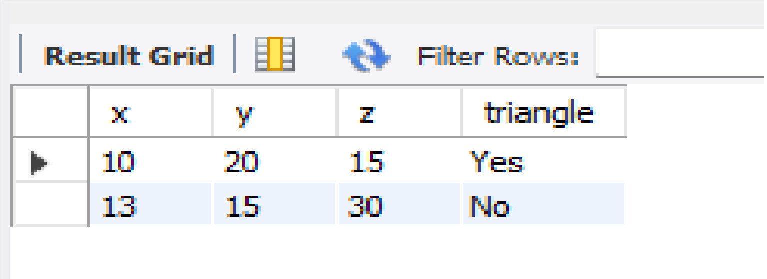
x	y	z
13	15	30
10	20	15

(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 segment whether they can form a triangle. Return the result table in any order.

Solution:

```
SELECT
        x,
        y,
        z,
CASE WHEN ((x+y>z) AND (x+z>y) AND (y+z>x)) THEN 'Yes' ELSE 'No'
END AS triangle
FROM Triangle;
```



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

Q61. DATASET

TABLE: Point

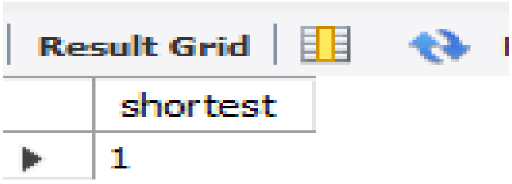
x
-1
0
2

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.

Solution:

```
SELECT
        MIN(ABS(P1.x-P2.x)) as shortest
FROM Point P1 JOIN Point P2
ON P1.x<>P2.x;
```



shortest
1

Q62. DATASET

TABLE: ActorDirector

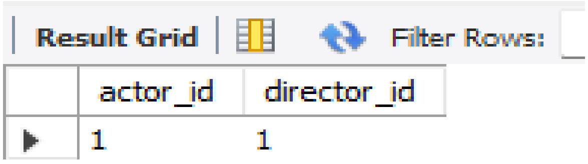
actor_id	director_id	timestamp
1	1	0
1	1	1
1	1	2
1	2	3
1	2	4
2	1	5
2	1	6

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.

Solution:

```
SELECT
        actor_id,
        director_id
FROM ActorDirector
GROUP BY actor_id, director_id
HAVING COUNT(timestamp)>=3;
```



actor_id	director_id
1	1

Q63. DATASET

TABLE 1: Sales

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

(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 2: Product

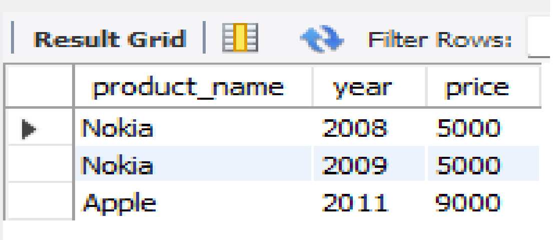
product_id	product_name
100	Nokia
200	Apple
300	Samsung

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.

Solution:

```
SELECT
          P.product_name,
           S.year,
          S.price
FROM Product P RIGHT JOIN Sales S
ON P.product_id = S.product_id;
```



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

Q64. DATASET

TABLE 1: Project

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

(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 2: Employee

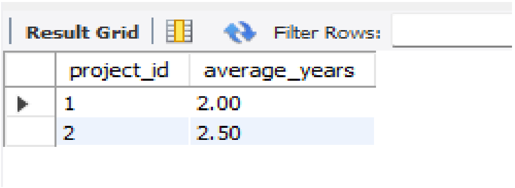
employee_id	name	experience_years
1	Khaled	3
2	Ali	2
3	John	1
4	Doe	2

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.

Solution:

```
SELECT
        P.project_id,
        ROUND(AVG(experience_years), 2) AS average_years
FROM Project P JOIN Employee E
ON P.employee_id = E.employee_id
GROUP BY P.project_id;
```



```
project_id  average_years
1           2.00
2           2.50
```

Q65. DATASET

TABLE 1: Product

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

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

TABLE 2: Sales

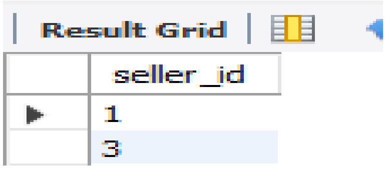
seller_id	product_id	buyer_id	sale_date	quantity	price
1	1	1	2019-01-21	2	2000
1	2	2	2019-02-17	1	800
2	2	3	2019-06-02	1	800
3	3	4	2019-05-13	2	2800

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.

Solution:

```
SELECT
        seller_id
FROM Sales
GROUP BY seller_id
HAVING SUM(price) = (
                                SELECT
                                        SUM(price)
                                FROM sales
                                GROUP BY seller_id
                                 ORDER BY sum(price) DESC LIMIT 1
                                );
```



seller_id
1
3

Q66. DATASET

TABLE 1: Product

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

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

TABLE 2: Sales

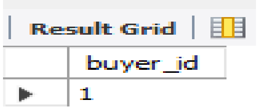
seller_id	product_id	buyer_id	sale_date	quantity	price
1	1	1	2019-01-21	2	2000
1	2	2	2019-02-17	1	800
2	1	3	2019-06-02	1	800
3	3	3	2019-05-13	2	2800

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.

Solution:

```
SELECT
        S.buyer_id
FROM Sales AS S
JOIN Product AS P
ON S.product_id = P.product_id
WHERE  product_name = 'S8'
AND S.buyer_id NOT IN
(       SELECT
                S.buyer_id
        FROM Sales AS S
        JOIN Product AS P
        ON S.product_id = P.product_id
        WHERE  product_name = 'Iphone'
);
```



```
buyer_id
1
```

Q67. DATASET

TABLE: Customer

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

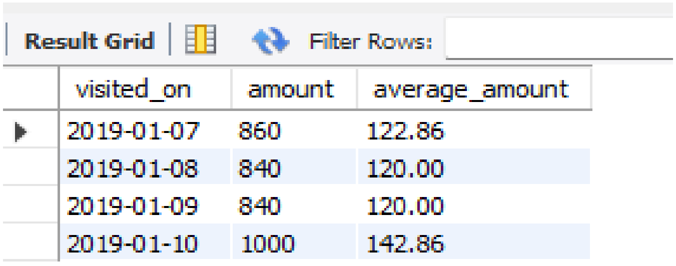
(customer_id, visited_on) is the primary key for this table. This table contains data about customer transactions in a restaurant. visited_on is the date on which the customer with ID (customer_id) has visited the restaurant.\ amount is the total paid by a customer.

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-day 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.

Solution:

```
SELECT
        a.visited_on AS visited_on,
        SUM(b.day_sum) AS amount,
        ROUND(AVG(b.day_sum), 2) AS average_amount
FROM
        (SELECT visited_on, SUM(amount) AS day_sum FROM Customer
        GROUP BY visited_on ) a,
        (SELECT visited_on, SUM(amount) AS day_sum FROM Customer
        GROUP BY visited_on ) b
WHERE DATEDIFF(a.visited_on, b.visited_on) BETWEEN 0 AND 6
GROUP BY a.visited_on
HAVING COUNT(b.visited_on) = 7
ORDER BY visited_on;
```



	visited_on	amount	average_amount
▶	2019-01-07	860	122.86
	2019-01-08	840	120.00
	2019-01-09	840	120.00
	2019-01-10	1000	142.86

Q68. DATASET

TABLE: Scores

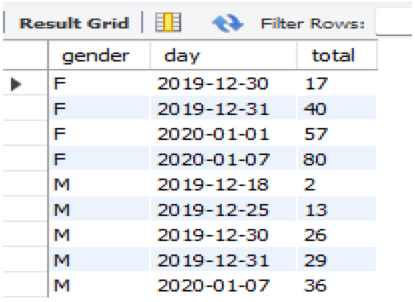
player_name	gender	day	score_points
Aron	F	2020-01-01	17
Alice	F	2020-01-07	23
Bajrang	M	2020-01-07	7
Khali	M	2019-12-25	11
Slaman	M	2019-12-30	13
Joe	M	2019-12-31	3
Jose	M	2019-12-18	2
Priya	F	2019-12-31	23
Priyanka	F	2019-12-30	17

(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 insomeday.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.

Solution:

```
SELECT
        gender,
         day,
        sum(score_points) OVER(PARTITION BY gender ORDER BY gender,
        day ) AS total
FROM Scores;
```



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

Q69. DATASET

TABLE: Logs

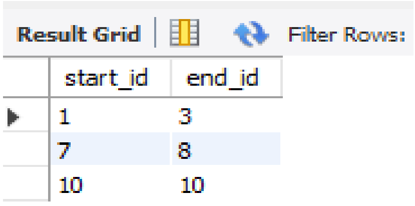
log_id
1
2
3
7
8
10

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.

Solution:

```
SELECT
     MIN(log_id) as start_id,
     MAX(log_id) as end_id
FROM
          (SELECT *,
                    ROW_NUMBER() OVER(ORDER BY log_id) as rnk
                    FROM Logs) l
GROUP BY log_id-rnk
ORDER BY start_id;
```



start_id	end_id
1	3
7	8
10	10

Q70. DATASET

TABLE 1: Students

student_id	student_name
1	Alice
2	Bob
13	John
6	Alex

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 2: Subjects

subject_name
Math
Physics
Programming

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

TABLE 3: Examination

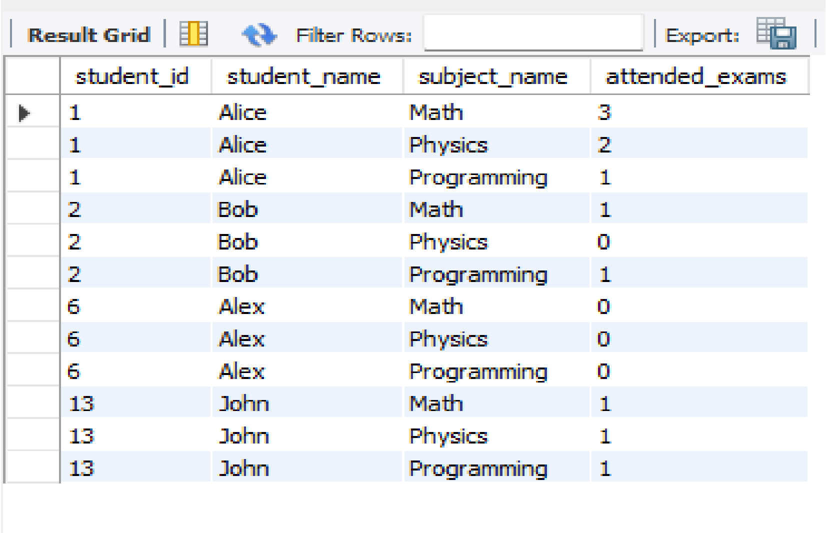
student_id	subject_name
1	Math
1	Physics
1	Programming
2	Programming
1	Physics
1	Math
13	Math
13	Programming
13	Physics
2	Math
1	Math

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.

Solution:

```
SELECT
        S.student_id,
         S.student_name,
        SU.subject_name,
         COUNT(E.subject_name) as attended_exams
FROM Students S JOIN Subjects SU
LEFT JOIN Examination E
ON E.student_id = S.student_id AND E.subject_name = SU. subject_name
GROUP BY student_name, subject_name
ORDER BY student_id, subject_name;
```



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

Q71. DATASET

TABLE: Employees

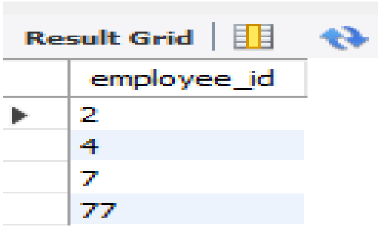
employee_id	employee_nam e	manager_id
1	Boss	1
3	Alice	3
2	Bob	1
4	Daniel	2
7	Luis	4
8	Jhon	3
9	Angela	8
77	Robert	1

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.

Solution:

```
SELECT
        E1.employee_id
FROM Employees E1 JOIN Employees E2
ON E1.manager_id = E2.employee_id
JOIN Employees E3
ON E2.manager_id = E3.employee_id
WHERE E1.employee_id!=1 AND (E2.manager_id = 1 OR E3.manager_id =1);
```



```
employee_id
2
4
7
77
```

Q72. DATASET

TABLE: Transactions

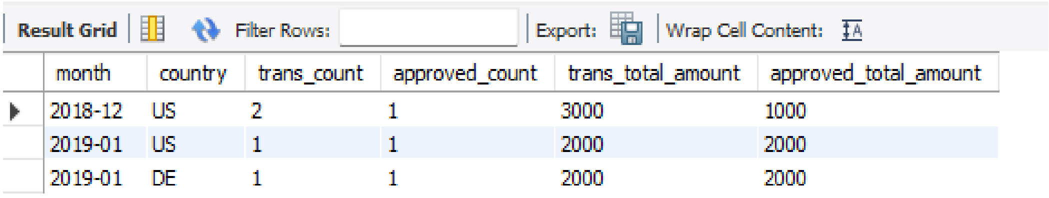
id	country	state	amount	trans_date
121	US	approved	1000	2018-12-18
122	US	declined	2000	2018-12-19
123	US	approved	2000	2019-01-01
124	DE	approved	2000	2019-01-07

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.

Solution:

```
SELECT
   DATE_FORMAT(trans_date, '%Y-%m') AS month,
   country,
   COUNT(id)AS trans_count,
   COUNT(IF(state = 'approved', 1, NULL)) AS approved_count,
   SUM(amount) AS trans_total_amount,
   SUM(IF(state = 'approved', amount, 0)) AS approved_total_amount
FROM Transactions
GROUP BY month, country;
```



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

Q73. DATASET

TABLE 1: Actions

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

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 2: Removals

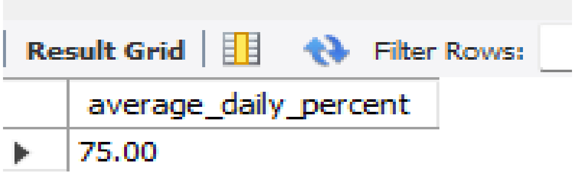
post_id	remove_date
2	2019-07-20
3	2019-07-18

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.

Solution:

```
SELECT
        ROUND(AVG(removal_post/total_post) * 100,2) AS
        average_daily_percent
FROM
        (SELECT
                action_date,
                COUNT(DISTINCT A.post_id) AS total_post,
                COUNT(DISTINCT R.post_id) AS removal_post
        FROM Actions A LEFT JOIN Removals R
        ON A.post_id = R.post_id
        WHERE extra = 'spam' AND action = 'report'
        GROUP BY action_date)T
```



```
average_daily_percent
75.00
```

Q74. DATASET

TABLE: Activity

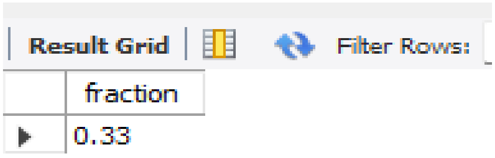
player_id	device_id	event_date	games_played
1	2	2016-03-01	5
1	2	2016-03-02	6
2	3	2017-06-25	1
3	1	2016-03-02	0
3	4	2018-07-03	5

(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.

Solution:

```
SELECT
        ROUND(SUM(CASE WHEN A1.event_date = A2.First_event +1 THEN
        1 ELSE 0 END) /count(DISTINCT A1.player_id), 2) AS fraction
        FROM Activity AS A1
        JOIN
        (
                SELECT player_id, MIN(event_date) AS First_event
                FROM Activity
                GROUP BY player_id ) AS A2
ON A1.player_id = A2.player_id;
```



fraction
0.33

Q75. DATASET

TABLE: Activity

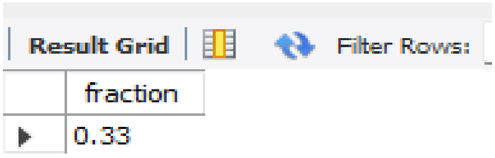
player_id	device_id	event_date	games_played
1	2	2016-03-01	5
1	2	2016-03-02	6
2	3	2017-06-25	1
3	1	2016-03-02	0
3	4	2018-07-03	5

(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.

Solution:

```
SELECT
        ROUND(SUM(CASE WHEN A1.event_date = A2.First_event +1 THEN
        1 ELSE 0 END) /count(DISTINCT A1.player_id), 2) AS fraction
        FROM Activity AS A1
        JOIN
        (
                SELECT player_id, MIN(event_date) AS First_event
                FROM Activity
                GROUP BY player_id ) AS A2
ON A1.player_id = A2.player_id;
```



	fraction
▶	0.33

Q76. DATASET

TABLE: Salaries

company_id	employee_id	employee_nam e	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

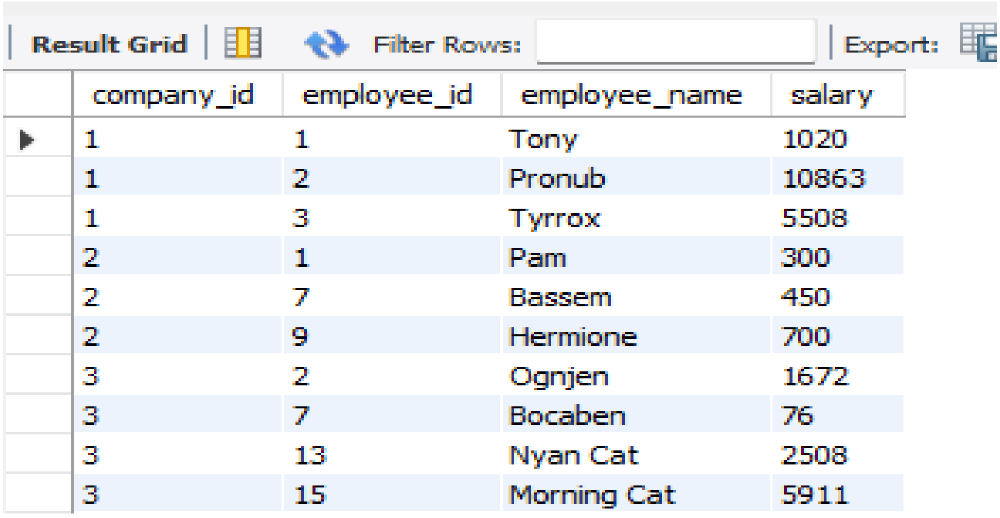
(company_id, employee_id) is the primary key for this table.
This table contains the company id, the id, the name, and the salary for an employee.

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.

Solution:

```
SELECT
        company_id,
        employee_id,
        employee_name,
ROUND(CASE
                WHEN MAX(salary) OVER(PARTITION BY company_id) <
                1000 THEN salary
                WHEN MAX(salary) OVER(PARTITION BY company_id)
                BETWEEN 1000 AND 10000 THEN salary*(1-0.24)
                ELSE salary*(1-0.49) END, 0) salary
FROM Salaries;
```



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	2	Ognjen	1672
3	7	Bocaben	76
3	13	Nyan Cat	2508
3	15	Morning Cat	5911

Q77. DATASET

TABLE: Sales

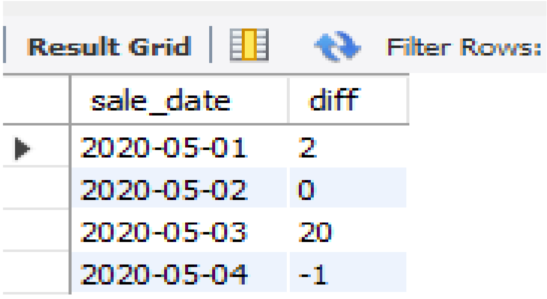
sale_date	fruit	sold_num
2020-05-01	apples	10
2020-05-01	oranges	8
2020-05-02	apples	15
2020-05-02	oranges	15
2020-05-03	apples	20
2020-05-03	oranges	0
2020-05-04	apples	15
2020-05-04	oranges	16

(sale_date, fruit) is the primary key for this table.
This table contains the sales of "apples" and "oranges" sold each day.

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.

Solution:

```
SELECT
        sale_date,
        SUM(IF(fruit = "apples", 1, -1) * sold_num) AS diff
FROM Sales
GROUP BY sale_date;
```



```
sale_date    diff
2020-05-01   2
2020-05-02   0
2020-05-03   20
2020-05-04   -1
```

Q78. DATASET

TABLE 1: Variables

name	value
x	66
y	77

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

TABLE 2: Expressions

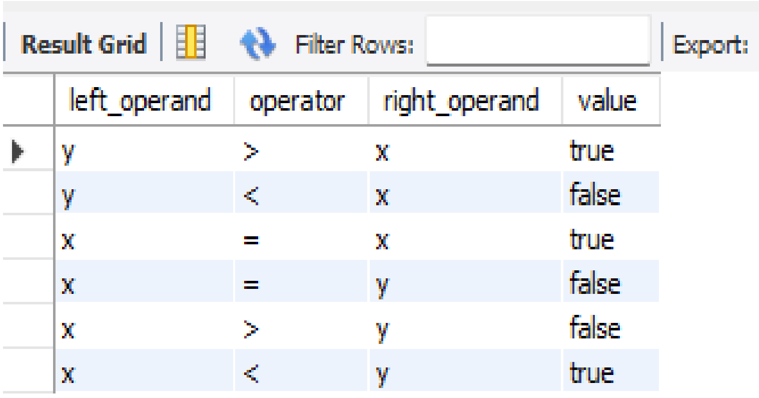
left_operand	operator	right_operand
x	>	y
x	<	y
x	=	y
y	>	x
y	<	x
x	=	x

(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.

Solution:

```
SELECT
          E.left_operand,
           E.operator,
          E.right_operand,
    CASE WHEN operator = '<' THEN IF(V1.value < V2.value, 'true','false')
                WHEN operator = '>' THEN IF(V1.value > V2.value, 'true','false')
                 ELSE IF(V1.value = V2.value, 'true', 'false')
     END AS value
FROM Expressions E
JOIN Variables V1 ON V1.name = E.left_operand
JOIN Variables V2 ON v2.name = E.right_operand
```



left_operand	operator	right_operand	value
y	>	x	true
y	<	x	false
x	=	x	true
x	=	y	false
x	>	y	false
x	<	y	true

Q79. DATASET

TABLE 1: Movies

movie_id	title
1	Avengers
2	Frozen 2
3	Joker

movie_id is the primary key for this table.
the title is the name of the movie.

TABLE 2: Users

user_id	name
1	Daniel
2	Monica
3	Maria
4	James

user_id is the primary key for this table.

TABLE 3: MovieRating

movie_id	user_id	rating	created_at
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

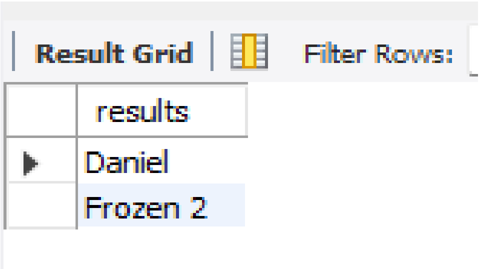
(movie_id, user_id) is the primary key for this table.
This table contains the rating of a movie by a user in their review.
created_at is the user's review date.

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.**

Solution:

```
(SELECT
        U.name AS results
FROM Users U
JOIN MovieRating MR
ON U.user_id = MR.user_id
GROUP BY U.user_id
ORDER BY count(MR.movie_id) desc, U.name LIMIT 1
)
UNION
(SELECT
        M.title as Movie_Name
FROM Movies M JOIN MovieRating MR
ON M.movie_id = MR.movie_id
WHERE month(created_at)=2
GROUP BY MR.movie_id
ORDER BY AVG(MR.rating) desc, M.title LIMIT 1
);
```



results
Daniel
Frozen 2

Q80. DATASET

TABLE 1: Person

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

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 2: Country

name	country_code
Peru	051
Israel	972
Morocco	212
Germany	49
Ethiopia	251

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 3: Calls

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

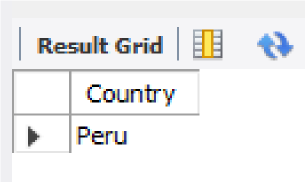
There is no primary key for this table, it may contain duplicates.
Each row of this table contains the caller id, callee id and the duration of the call in minutes. **caller_id != callee_id.**

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.

Solution:

```
SELECT
        C.name AS Country
        FROM Person P
        JOIN Calls Ca
        ON P.id = Ca.caller_id or P.id = Ca.callee_id
        JOIN Country C
        ON LEFT (P.phone_number,3) = C.country_code
GROUP BY C.name
HAVING AVG(duration) > (SELECT AVG(duration) FROM Calls);
```



	Country
▶	Peru

Q81. DATASET

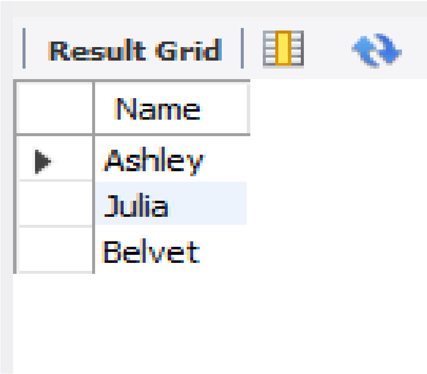
TABLE: Students

ID	Name	Marks
1	Ashley	81
2	Samantha	75
4	Julia	76
3	Belvet	84

Query the Name of any student in STUDENTS who scored higher than 75 Marks. Order your output by the last three characters of each name. If two or more students both have names ending in the same last three characters (i.e.: Bobby, Robby, etc.), secondary sort them by ascending ID.

Solution:

```
SELECT
        Name
FROM Students
WHERE Marks > 75
ORDER BY RIGHT(Name, 3), id ASC;
```



```
Name
Ashley
Julia
Belvet
```

Q82. DATASET

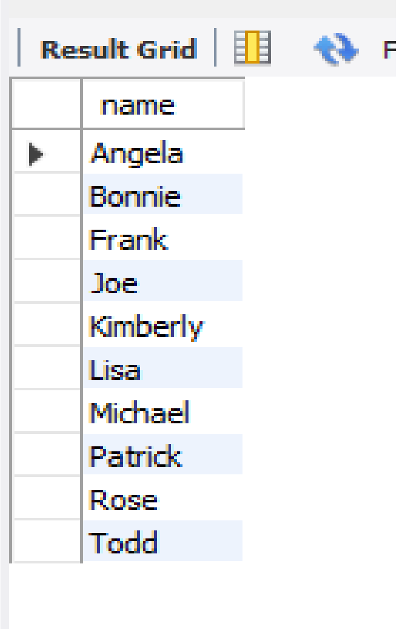
TABLE: Employee

employee_id	name	months	salary
12228	Rose	15	1968
33645	Angela	1	3443
45692	Frank	17	4608
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

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

Solution:

```
SELECT
	name
FROM Employee
ORDER BY name ASC;
```



name
Angela
Bonnie
Frank
Joe
Kimberly
Lisa
Michael
Patrick
Rose
Todd

Q83. DATASET

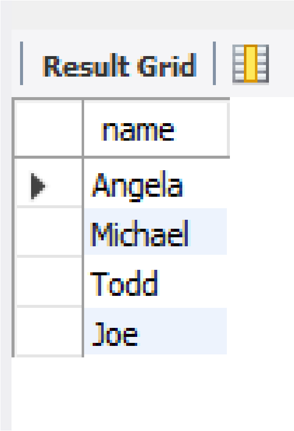
TABLE: Employee

employee_id	name	months	salary
12228	Rose	15	1968
33645	Angela	1	3443
45692	Frank	17	4608
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

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.

Solution:

```
SELECT
        name
FROM Employee
WHERE salary >2000 AND months < 10
ORDER BY employee_id;
```



name
Angela
Michael
Todd
Joe

Q84. DATASET

TABLE: Triangles

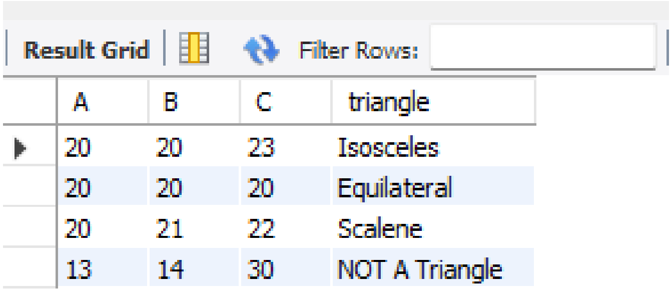
A	B	C
20	20	23
20	20	20
20	21	22
13	14	30

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

Write a query identifying the type of each record in the TRIANGLES table using its three side lengths.
Output one of the following statements for each record in the table:
● Equilateral: It's a triangle with sides of equal length.
● Isosceles: It's a triangle with sides of equal length.
● Scalene: It's a triangle with sides of differing lengths.
● Not A Triangle: The given values of A, B, and C don't form a triangle.

Solution:

```
SELECT
        A,
         B,
          C,
        CASE WHEN A+B<=C OR B+C<=A OR A+C<=B THEN  'NOT A
        Triangle'
                  WHEN A = B AND B=C AND A = C THEN 'Equilateral'
                 WHEN A = B OR B = C OR A = C THEN 'Isosceles'
                ELSE 'Scalene' END  as triangle
FROM Triangles;
```



A	B	C	triangle
20	20	23	Isosceles
20	20	20	Equilateral
20	21	22	Scalene
13	14	30	NOT A Triangle

Q85. DATASET

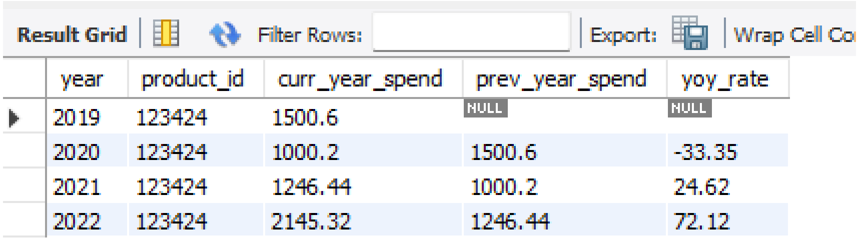
TABLE: user_transactions

transaction_id	product_id	spend	transaction_date
1341	123424	1500.6	31-12-2019 12:00
1423	123424	1000.2	31-12-2020 12:00
1623	123424	1246.44	31-12-2021 12:00
1322	123424	2145.32	31-12-2022 12:00

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).

Solution:

```
SELECT *,
        ROUND(((curr_year_spend - prev_year_spend) * 100) /
        prev_year_spend, 2) AS yoy_rate
FROM
(SELECT EXTRACT(YEAR FROM transaction_date) AS year,
       product_id,
       spend AS curr_year_spend,
       ROUND(
        LAG(spend) OVER(PARTITION BY product_id ORDER BY
        EXTRACT(YEAR FROM transaction_date)), 2) AS prev_year_spend
 FROM user_transactions
)t
```



year	product_id	curr_year_spend	prev_year_spend	yoy_rate
2019	123424	1500.6	NULL	NULL
2020	123424	1000.2	1500.6	-33.35
2021	123424	1246.44	1000.2	24.62
2022	123424	2145.32	1246.44	72.12

Q85. DATASET

TABLE: inventory

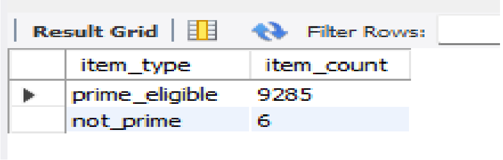
item_id	item_type	item_category	square_footage
1374	prime_eligible	mini refrigerator	68
4245	not_prime	standing lamp	26.4
2452	prime_eligible	television	85
3255	not_prime	side table	22.6
1672	prime_eligible	laptop	8.5

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.

Solution:

```
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 AS item_count
FROM
inventory
GROUP BY item_type
ORDER BY item_type desc;
```



```
item_type        item_count
prime_eligible   9285
not_prime        6
```

Q87. DATASET

TABLE: User_actions

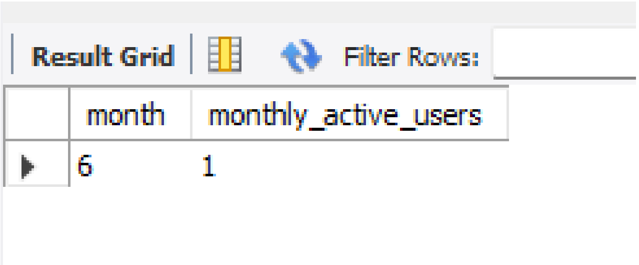
user_id	event_id	event_type	event_date
445	7765	sign-in	05/31/2022 12:00:00
742	6458	sign-in	06/03/2022 12:00:00
445	3634	like	06/05/2022 12:00:00
742	1374	comment	06/05/2022 12:00:00
648	3124	like	06/18/2022 12:00:00

Assume you have the table 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).

Hint: An active user is a user who has user action ("sign-in", "like", or "comment") in the current month and last month.

Solution:

```
SELECT
        EXTRACT(MONTH FROM U1.event_date) as month,
         COUNT(DISTINCT U1.user_id) as monthly_active_users
FROM user_actions U1 JOIN user_actions U2
ON U1.user_id = U2.user_id
AND EXTRACT(MONTH FROM U1.event_date) = EXTRACT(MONTH
FROM U2.event_date) +1
WHERE EXTRACT(MONTH FROM U1.event_date)=6
GROUP BY EXTRACT(MONTH FROM U1.event_date);
```



month	monthly_active_users
6	1

Q88. DATASET

TABLE: search_frequency

searches	num_users
1	2
2	2
3	3
4	1

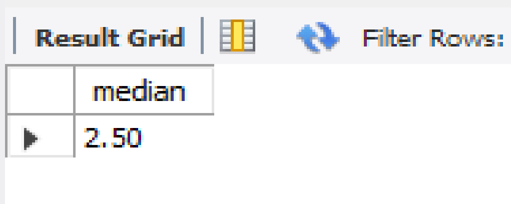
Google's marketing team is making a Superbowl commercial and needs a simple statistic to put on their TV ad: the median number of searches a person made last year. However, at Google scale, querying the 2 trillion searches is too costly. Luckily, you have access to the summary table which tells you the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

Solution:

```
SELECT
        ROUND(AVG(Searches*1.0),2) AS median
FROM (SELECT *,
                SUM(num_users) OVER (ORDER BY Searches ASC) AS
                accumulated_sum,
                SUM(num_users) OVER () / 2 as medium_num
         FROM
        search_frequency)tmp
WHERE accumulated_sum - num_users <= medium_num AND
accumulated_sum >= medium_num;
```



```
median
2.50
```

Q89. DATASET

TABLE 1: advertiser

user_id	status
bing	NEW
yahoo	NEW
alibaba	EXISTING

TABLE 2: daily_pay

user_id	paid
yahoo	45.00
alibaba	100.00
target	13.00

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.

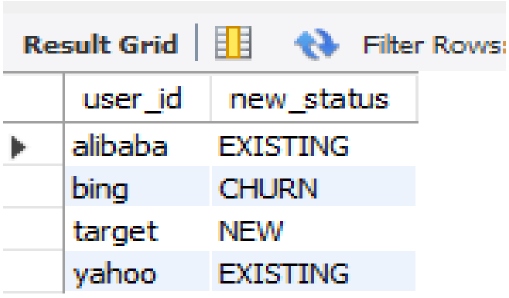
Definition of advertiser status:

- **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.**

#	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

Solution:

```
SELECT
        user_id, 'EXISTING' AS new_status FROM advertiser
        WHERE user_id in (SELECT user_id FROM daily_pay)
        and status <> 'CHURN'
UNION ALL
        SELECT user_id, 'RESURRECT' AS new_status FROM advertiser
        WHERE user_id in (SELECT user_id FROM daily_pay)
        and status = 'CHURN'
UNION ALL
        SELECT user_id, 'NEW' as new_status FROM daily_pay
        WHERE user_id not in (SELECT user_id FROM advertiser)
UNION ALL
        SELECT user_id, 'CHURN' AS new_status FROM advertiser
        WHERE user_id not in (SELECT user_id FROM daily_pay)
ORDER BY user_id;
```



user_id	new_status
alibaba	EXISTING
bing	CHURN
target	NEW
yahoo	EXISTING

Q91. DATASET

TABLE: transactions

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

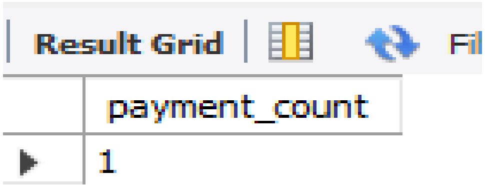
Sometimes, payment transactions are repeated by accident; it could be due to user error, API failure or a retry error that causes a credit card to be charged twice. Using the transactions table, identify any payments made at the same merchant with the same credit card for the same amount within 10 minutes 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.

Solution:

```
SELECT
        COUNT(merchant_id) as payment_count
FROM (
        SELECT *,
                transaction_timestamp - lag(transaction_timestamp)
                OVER(PARTITION BY merchant_id, credit_card_id, amount
                ORDER BY transaction_timestamp) as Diff
        FROM transactions) PT
WHERE EXTRACT(minute FROM Diff)<10;
```



	payment_count
▶	1

Q92. DATASET

TABLE 1: Orders

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

TABLE 2: Trips

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

TABLE 3: Customers

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

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:

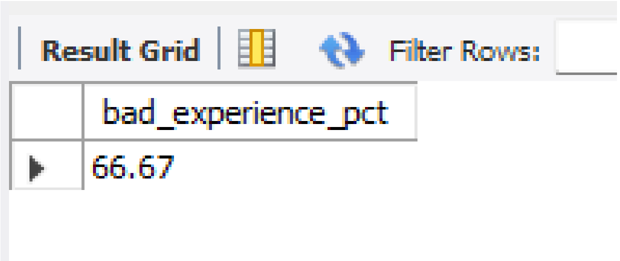
- the orders are being completed incorrectly (missing items, wrong order, etc.)
- the orders aren't being received (wrong address, wrong drop off spot)
- the orders are being delivered late (the actual delivery time is 30 minutes later than when the order was placed). Note that the estimated_delivery_timestamp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

Solution:

```
SELECT
        ROUND(100.0 * COUNT(CASE WHEN status='completed successfully'
        THEN NULL ELSE trip_id END)
        / COUNT(trip_id),2) as bad_experience_pct
FROM orders
JOIN customers ON customers.customer_id=orders.customer_id
WHERE EXTRACT(MONTH FROM signup_timestamp)='6' AND
signup_timestamp + INTERVAL 14 DAY>= order_timestamp;
```



	bad_experience_pct
▶	66.67

Q93. DATASET

TABLE: Scores

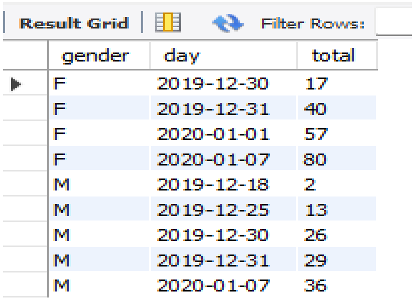
player_name	gender	day	score_points
Aron	F	2020-01-01	17
Alice	F	2020-01-07	23
Bajrang	M	2020-01-07	7
Khali	M	2019-12-25	11
Slaman	M	2019-12-30	13
Joe	M	2019-12-31	3
Jose	M	2019-12-18	2
Priya	F	2019-12-31	23
Priyanka	F	2019-12-30	17

(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 insomeday.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.

Solution:

```
SELECT
        gender,
         day,
        sum(score_points) OVER(PARTITION BY gender ORDER BY gender,
        day ) AS total
FROM Scores;
```



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

Q94. DATASET

TABLE 1: Person

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

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 2: Country

name	country_code
Peru	051
Israel	972
Morocco	212
Germany	49
Ethiopia	251

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 3: Calls

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

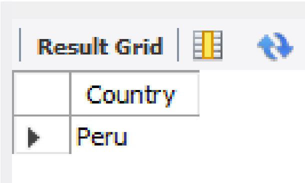
There is no primary key for this table, it may contain duplicates.
Each row of this table contains the caller id, callee id and the duration of the call in minutes. **caller_id != callee_id.**

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.

Solution:

```
SELECT
        C.name AS Country
        FROM Person P
        JOIN Calls Ca
        ON P.id = Ca.caller_id or P.id = Ca.callee_id
        JOIN Country C
        ON LEFT (P.phone_number,3) = C.country_code
GROUP BY C.name
HAVING AVG(duration) > (SELECT AVG(duration) FROM Calls);
```



Country
Peru

Q95. DATASET

TABLE: Numbers

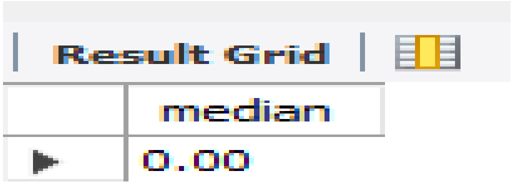
num	frequency
0	7
1	1
2	3
3	1

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.

Solution:

```
SELECT
        ROUND(AVG(num*1.0),2) AS median
FROM (SELECT *,
      SUM(frequency) OVER (ORDER BY num ASC) AS Accsum,
      SUM(frequency) OVER () / 2 as medium_num
      FROM Numbers) tmp
WHERE Accsum - frequency <= medium_num AND Accsum >= medium_num
```



```
median
0.00
```

Q96. DATASET

TABLE 1: Salary

employee_id	amount	pay_date
1	9000	2017-03-31
2	6000	2017-03-31
3	10000	2017-03-31
1	7000	2017-02-28
2	6000	2017-02-28
3	8000	2017-02-28

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 1: Salary

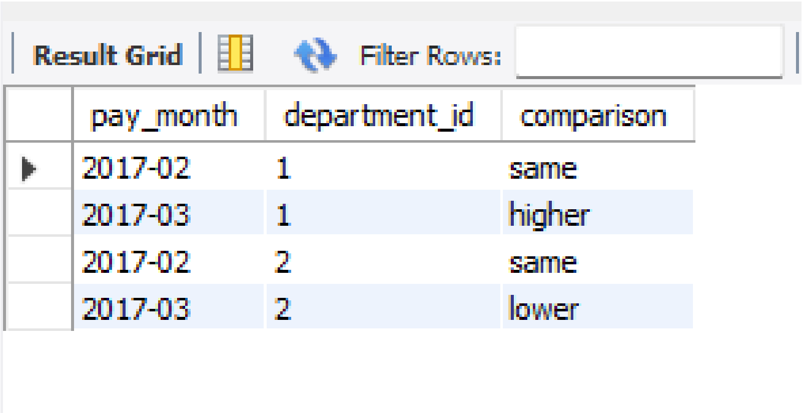
employee_id	department_id
1	1
2	2
3	2

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.

Solution:

```
SELECT
        DISTINCT pay_month,
        department_id,
        (CASE WHEN department_avg_salary > company_avg_salary THEN
        'higher'
                 WHEN department_avg_salary < company_avg_salary THEN
                'lower'
                WHEN department_avg_salary = company_avg_salary THEN
                'same'
                END) AS comparison
        FROM
                (SELECT
                        A.employee_id,
                        amount,
                        pay_date,
                        department_id,
                        LEFT(pay_date,7) as pay_month,
                        AVG(amount) OVER(PARTITION BY A.pay_date) AS
                        company_avg_salary,
                        AVG(amount) OVER(PARTITION BY A.pay_date,
                        B.department_id) AS department_avg_salary
                        FROM salary AS A
                        JOIN employee AS B
                        ON A.employee_id = B.employee_id)tmp
ORDER BY department_id;
```



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

Q97. DATASET

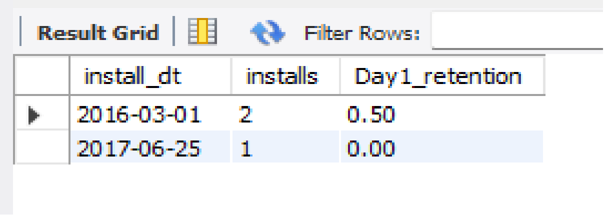
TABLE: Activity

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

(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.

Solution:

```
SELECT
        a1.first_login as "install_dt",
        count(a1.player_id) as "installs",
        round(count(a2.player_id)/count(a1.player_id),2) as "Day1_retention"
FROM
(SELECT
        player_id, MIN(event_date) AS first_login
        FROM Activity
        GROUP BY player_id) a1
LEFT JOIN Activity a2
ON a1.player_id = a2.player_id AND a1.first_login = a2.event_date-1
GROUP BY a1.first_login;
```



install_dt	installs	Day1_retention
2016-03-01	2	0.50
2017-06-25	1	0.00

Q98. DATASET

TABLE 1: Players

player_id	group_id
15	1
25	1
30	1
45	1
10	2
35	2
50	2
20	3
40	3

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

TABLE 2: Matches

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

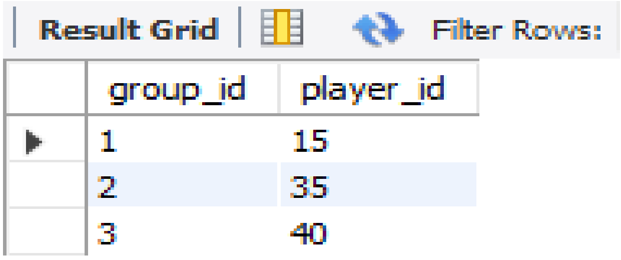
match_id is the primary key of this table.
Each row is a record of a match, first_player and second_player contain the player_id of each match. first_score and second_score contain the number of points of the first_player and second_player respectively.

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.

Solution:

```
SELECT
          group_id,
          player_id
FROM
(SELECT
          group_id,
            player_id,
            RANK() OVER(PARTITION BY group_id ORDER BY score DESC,
            player_id) as rnk
FROM
(SELECT
          group_id,
           a.player_id,
           SUM(score) as score
FROM
(SELECT
          first_player as player_id,
          first_score as score from Matches
UNION ALL
SELECT
          second_player as player_id,
          second_score as score from Matches) a
LEFT JOIN Players b
on a.player_id = b.player_id
group by group_id, a.player_id) a) a
where rnk = 1;
```



	group_id	player_id
▶	1	15
	2	35
	3	40

Q99. DATASET

TABLE 1: Student

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

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

TABLE 2: Exam

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

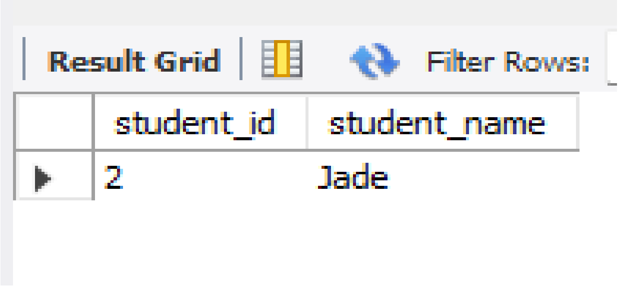
(exam_id, student_id) is the primary key for this table.
Each row of this table indicates that the student with student_id had a score points in the exam with id exam_id.

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.

Solution:

```
SELECT
   e.student_id,
   s.student_name
FROM (
   SELECT
        exam_id,
        student_id,
        RANK() OVER(PARTITION BY exam_id ORDER BY score DESC)
        AS desc_rk,
        RANK() OVER(PARTITION BY exam_id ORDER BY score) AS
        asc_rk
        FROM Exam) e
JOIN Student AS s USING(student_id)
GROUP BY e.student_id
HAVING MIN(e.desc_rk) != 1 AND MIN(e.asc_rk) != 1
ORDER BY e.student_id;
```



student_id	student_name
2	Jade

Q100. DATASET

TABLE 1: Student

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

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

TABLE 2: Exam

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

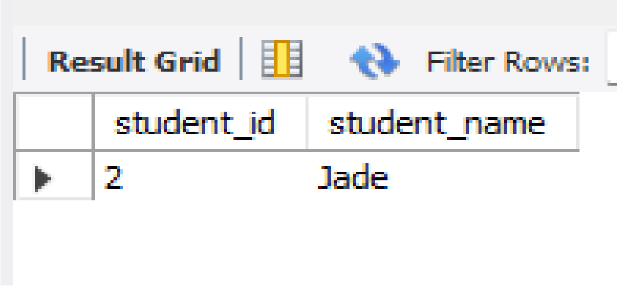
(exam_id, student_id) is the primary key for this table.
Each row of this table indicates that the student with student_id had a score points in the exam with id exam_id.

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.

Solution:

```
SELECT
   e.student_id,
   s.student_name
FROM (
   SELECT
        exam_id,
        student_id,
        RANK() OVER(PARTITION BY exam_id ORDER BY score DESC)
       AS desc_rk,
        RANK() OVER(PARTITION BY exam_id ORDER BY score) AS
       asc_rk
       FROM Exam) e
JOIN Student AS s USING(student_id)
GROUP BY e.student_id
HAVING MIN(e.desc_rk) != 1 AND MIN(e.asc_rk) != 1
ORDER BY e.student_id;
```



student_id	student_name
2	Jade

Q101. DATASET

TABLE: UserActivity

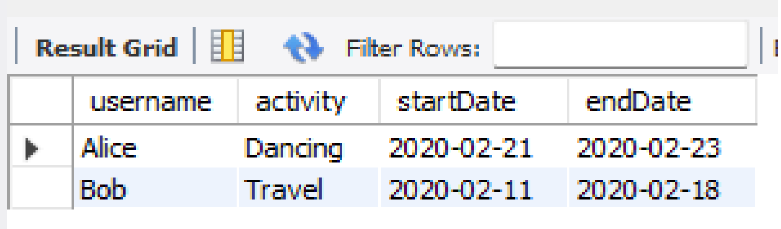
username	activity	startDate	endDate
Alice	Travel	2020-02-12	2020-02-20
Alice	Dancing	2020-02-21	2020-02-23
Alice	Travel	2020-02-24	2020-02-28
Bob	Travel	2020-02-11	2020-02-18

There is **no primary key** for this table. It may contain duplicates.
This table contains information about the activity performed by each user in a period of time. A person with a username performed an activity from startDate to endDate.

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.
Return the result table in any order.

Solution:

```
SELECT
        username,
        activity,
         startDate,
         endDate
FROM (SELECT *,
                        COUNT(activity)  OVER(PARTITION  BY  username)  AS
                        act,
                        ROW_NUMBER()   OVER(PARTITION   BY   username
                        ORDER BY startDate DESC)  As rn
                FROM UserActivity) tmp
WHERE rn = 2 OR act < 2;
```



username	activity	startDate	endDate
Alice	Dancing	2020-02-21	2020-02-23
Bob	Travel	2020-02-11	2020-02-18

Q102. DATASET

TABLE: UserActivity

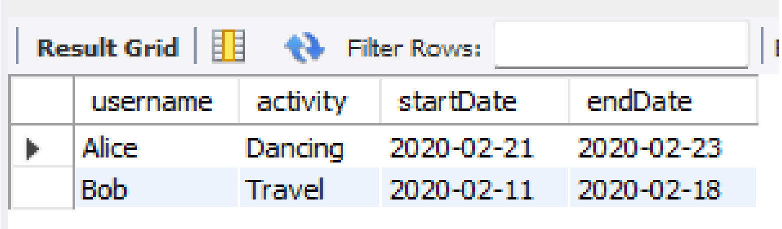
username	activity	startDate	endDate
Alice	Travel	2020-02-12	2020-02-20
Alice	Dancing	2020-02-21	2020-02-23
Alice	Travel	2020-02-24	2020-02-28
Bob	Travel	2020-02-11	2020-02-18

There is **no primary key** for this table. It may contain duplicates.
This table contains information about the activity performed by each user in a period of time. A person with a username performed an activity from startDate to endDate.

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.
Return the result table in any order.

Solution:

```
SELECT
          username,
          activity,
           startDate,
           endDate
FROM (SELECT *,
                              COUNT(activity) OVER(PARTITION BY username) AS
                              act,
                              ROW_NUMBER() OVER(PARTITION BY username
                              ORDER BY startDate DESC) As rn
                    FROM UserActivity) tmp
WHERE rn = 2 OR act < 2;
```



username	activity	startDate	endDate
Alice	Dancing	2020-02-21	2020-02-23
Bob	Travel	2020-02-11	2020-02-18

Q103. DATASET

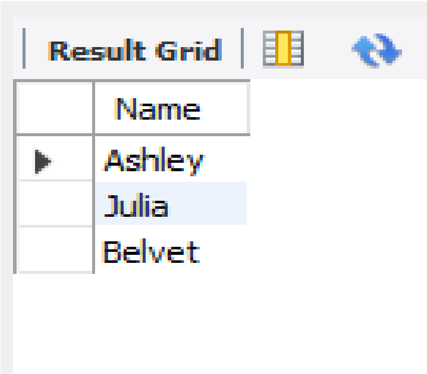
TABLE: Students

ID	Name	Marks
1	Ashley	81
2	Samantha	75
4	Julia	76
3	Belvet	84

Query the Name of any student in STUDENTS who scored higher than 75 Marks. Order your output by the last three characters of each name. If two or more students both have names ending in the same last three characters (i.e.: Bobby, Robby, etc.), secondary sort them by ascending ID.

Solution:

```
SELECT
        Name
FROM Students
WHERE Marks > 75
ORDER BY RIGHT(Name, 3), id ASC;
```



```
Name
Ashley
Julia
Belvet
```

Q104. DATASET

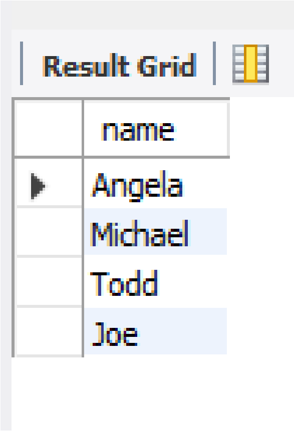
TABLE: Employee

employee_id	name	months	salary
12228	Rose	15	1968
33645	Angela	1	3443
45692	Frank	17	4608
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

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.

Solution:

```
SELECT
        name
FROM Employee
WHERE salary >2000 AND months < 10
ORDER BY employee_id;
```



```
name
Angela
Michael
Todd
Joe
```

Q105. DATASET

TABLE: Triangles

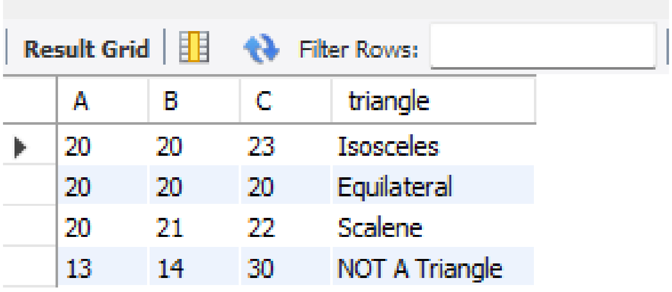
A	B	C
20	20	23
20	20	20
20	21	22
13	14	30

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

Write a query identifying the type of each record in the TRIANGLES table using its three side lengths.
Output one of the following statements for each record in the table:
● Equilateral: It's a triangle with sides of equal length.
● Isosceles: It's a triangle with sides of equal length.
● Scalene: It's a triangle with sides of differing lengths.
● Not A Triangle: The given values of A, B, and C don't form a triangle.

Solution:

```
SELECT
         A,
          B,
           C,
         CASE WHEN A+B<=C OR B+C<=A OR A+C<=B THEN  'NOT A
         Triangle'
                    WHEN A = B AND B=C AND A = C THEN 'Equilateral'
                   WHEN A = B OR B = C OR A = C THEN 'Isosceles'
                  ELSE 'Scalene' END  as triangle
FROM Triangles;
```



```
A    B    C    triangle
20   20   23   Isosceles
20   20   20   Equilateral
20   21   22   Scalene
13   14   30   NOT A Triangle
```

Q106. DATASET

TABLE: EMPLOYEES

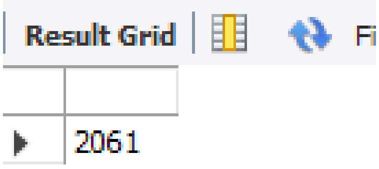
ID	Name	Salary
1	Kristeen	1420
2	Ashley	2006
3	Julia	2210
4	Maria	3000

Samantha was tasked with calculating the average monthly salaries for all employees in the EMPLOYEES table, but did not realise her keyboard's 0 key was broken until after completing the calculation. She wants your help finding the difference between her miscalculation (using salaries with any zeros removed), and the actual average salary.

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

Solution:

```
SELECT
        CEIL(AVG(SALARY)-AVG(REPLACE(SALARY, 0,"))) as ' '
FROM EMPLOYEES;
```



```
2061
```

NOTE: Samantha computes an average salary of 98.00 . The actual average salary is 2159.00. The resulting error between the two calculations is 2159.00-98.00 = 2061.00. Since it is equal to the integer 2061, it does not get rounded up.

Q107. DATASET

TABLE: EMPLOYEE

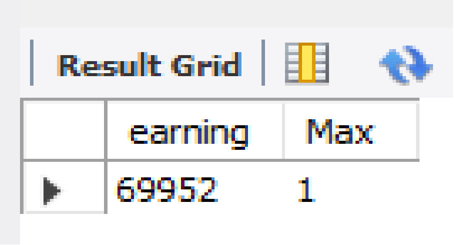
employee_id	name	month	salary
12228	Rose	15	1968
33645	Angela	1	3443
45692	Frank	17	1608
56118	Patrick	7	1345
59725	Lisa	11	2330
74197	Kimberly	16	4372
78454	Bonnie	8	1771
83565	Michael	6	2017
98607	Todd	5	3396
99989	Joe	9	3573

We define an employee's total earnings to be their monthly salary * months worked, and the maximum total earnings to be the maximum total earnings for any employee in the Employee table.

Write a query to find the maximum total earnings for all employees as well as the total number of employees who have maximum total earnings. Then print these values as 2 space-separated integers.

Solution:

```
SELECT
        earning,
        COUNT(earning) as Max
FROM
(SELECT *,
              salary*months earning FROM Employee) A
GROUP BY earning ORDER BY earning DESC limit 1;
```



earning	Max
69952	1

Q108. DATASET

TABLE: Occupations

Name	Occupation
Samantha	Doctor
Julia	Actor
Maria	Actor
Meera	Singer
Ashley	Professor
Ketty	Professor
Christeen	Professor
Jane	Actor
Jenny	Doctor
Priya	Singer

Generate the following two result sets:

1.Query an alphabetically ordered list of all names in OCCUPATIONS, immediately followed by the first letter of each profession as a parenthetical (i.e.: enclosed in parentheses). For example: AnActorName(A), ADoctorName(D), AProfessorName(P), and ASingerName(S). Query the number of occurrences of each occupation in OCCUPATIONS. Sort the occurrences in ascending order, and output them in the following format:

Level – Medium There are a total of [occupation_count] [occupation]s.

2. where [occupation_count] is the number of occurrences of an occupation in OCCUPATIONS and [occupation] is the lowercase occupation name. If more than one Occupation has the same [occupation_count], they should be ordered alphabetically.

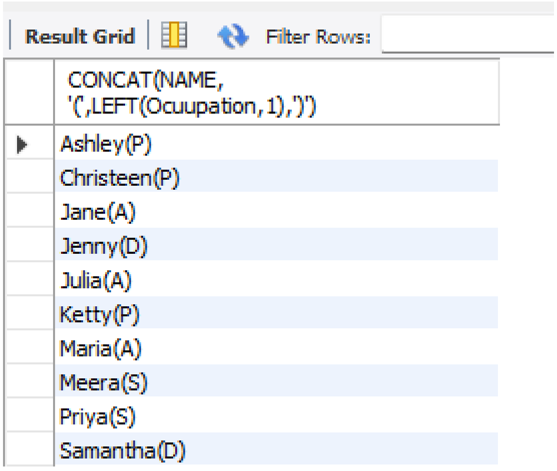
Note: There will be at least two entries in the table for each type of occupation.

Input Format

The OCCUPATIONS table is described as follows:

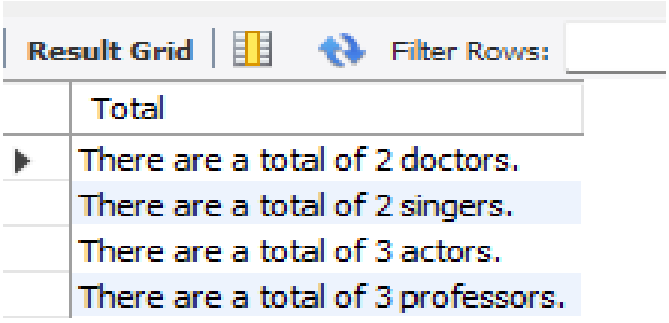
Occupation will only contain one of the following values: Doctor, Professor, Singer or Actor.

```
SELECT
        CONCAT(NAME, '(',LEFT(Ocuupation,1),')')
        FROM Occupations
ORDER BY NAME;
```



```
CONCAT(NAME,
'(',LEFT(Ocuupation,1),')')
Ashley(P)
Christeen(P)
Jane(A)
Jenny(D)
Julia(A)
Ketty(P)
Maria(A)
Meera(S)
Priya(S)
Samantha(D)
```

```
SELECT
        CONCAT('There are a total of ', COUNT(Ocuupation), '
        ',LOWER(Ocuupation), 's.') AS Total
 FROM Occupations
GROUP BY Ocuupation
ORDER BY COUNT(Ocuupation) , Ocuupation;
```



```
Total
There are a total of 2 doctors.
There are a total of 2 singers.
There are a total of 3 actors.
There are a total of 3 professors.
```

Q110. DATASET

TABLE: BST

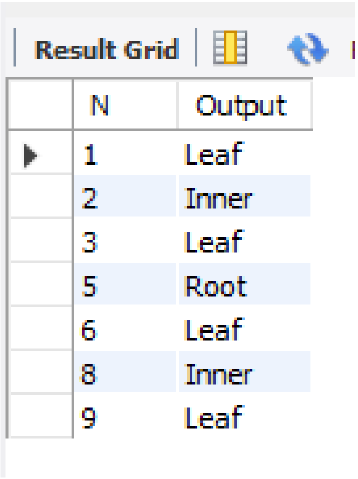
N	P
1	2
3	2
6	8
9	8
2	5
8	5
5	null

Write a query to find the node type of Binary Tree ordered by the value of the node. Output one of the following for each node:

- **Root: If node is root node.**
- **Leaf: If node is leaf node.**
- **Inner: If node is neither root nor leaf node.**

Solution:

```
SELECT
        N,
         CASE WHEN P is null then 'Root'
         WHEN N in (SELECT DISTINCT P FROM BST) THEN 'Inner'
         ELSE 'Leaf' END AS Output
FROM BST
ORDER BY N;
```



N	Output
1	Leaf
2	Inner
3	Leaf
5	Root
6	Leaf
8	Inner
9	Leaf

Q111. DATASET

TABLE 1: Company

company_code	founder
C1	Monika
C2	Samantha

TABLE 2: Lead_Manager

lead_manager_code	company_code
LM1	C1
LM2	C2

TABLE 3: Senior_Manager

seinor_manager_code	lead_manager_code	company_code
SM1	LM1	C1
SM2	LM1	C1
SM3	LM2	C2

TABLE 4: Manager

Manager_code	seinor_manager_code	lead_manager_code	company_code
M1	SM1	LM1	C1
M2	SM3	LM2	C2
M3	SM3	LM2	C2

TABLE 5: Employee

employee_code	Manager_code	seinor_manager_code	lead_manager_code	company_code
E1	M1	SM1	LM1	C1
E2	M1	SM1	LM1	C1
E3	M2	SM3	LM2	C2
E4	M3	SM3	LM2	C2

write a query to print the company_code, founder name, total numberof lead managers, total number of senior managers, total number of managers, and total number of employees. Order your output by ascending company_code.

Solution:

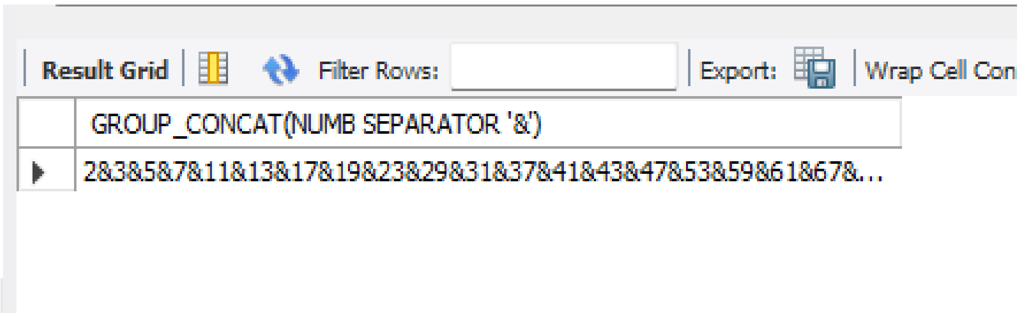
```
Select c.company_code, founder,
       count(distinct lm.lead_manager_code),
       count(distinct sm.senior_manager_code),
       count(distinct m.manager_code),
       count(distinct e.employee_code)
from Company as c
LEFT JOIN Lead_Manager lm
ON lm.company_code = c.company_code
LEFT JOIN Senior_Manager sm
on sm.lead_manager_code = lm.lead_manager_code
LEFT JOIN Manager m
on m.senior_manager_code = sm.senior_manager_code
LEFT  JOIN Employee e
on e.manager_code = m.manager_code
group by c.company_code, founder
order by company_code ASC
```

Q112.

Write a query to print all prime numbers less than or equal to 1000. Print your result on a single line and use the ampersand () character as your separator (instead of a space).
For example, the output for all prime numbers <=10 would be: 2&3&5&7'

Solution:

```
SELECT
        GROUP_CONCAT(NUMB SEPARATOR '&')
FROM (
   SELECT
                @num:=@num+1 as NUMB FROM
                information_schema.tables t1,
                information_schema.tables t2,
                (SELECT @num:=1) tmp
        ) tmp
WHERE NUMB<=1000 AND NOT EXISTS(
                SELECT *
                FROM (SELECT @nu:=@nu+1 as NUMA FROM
                           information_schema.tables t1,
                           information_schema.tables t2,
                           (SELECT @nu:=1) tmp1
                           LIMIT 1000
                        ) t
                WHERE FLOOR(NUMB/NUMA)=(NUMB/NUMA) AND
                NUMA<NUMB AND NUMA>1
        )
```



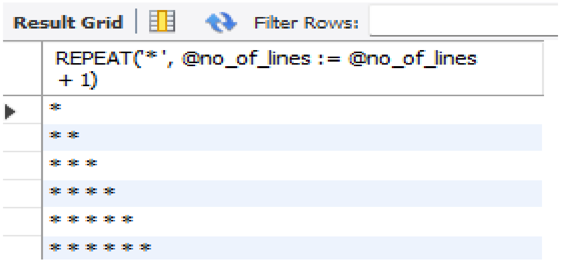
```
GROUP_CONCAT(NUMB SEPARATOR '&')
2&3&5&7&11&13&17&19&23&29&31&37&41&43&47&53&59&61&67&...
```

Q113. P(R) represents a pattern drawn by Julia in R rows. The following pattern represents P(5):

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

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

```
SET @no_of_lines = 0;
SELECT
        REPEAT('* ', @no_of_lines := @no_of_lines + 1)
FROM INFORMATION_SCHEMA.TABLES
LIMIT 20;
```



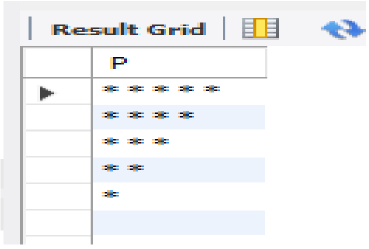
```
REPEAT('* ', @no_of_lines := @no_of_lines
 + 1)
*
* *
* * *
* * * *
* * * * *
* * * * * *
```

Q114. P(R) represents a pattern drawn by Julia in R rows. The following pattern represents P(5):

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

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

```
SET @no_of_lines = 6;
SELECT REPEAT('* ', @no_of_lines := @no_of_lines -1)
FROM INFORMATION_SCHEMA.TABLES;
```



```
P
* * * * *
* * * *
* * *
* *
*
```

Q115. DATASET

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

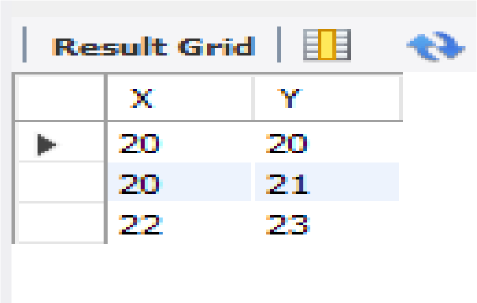
X	Y
20	20
20	20
20	21
23	22
22	23
21	20

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

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

Solution:

```
SELECT
          A.X,
           A.Y
FROM sys A JOIN sys B
ON A.X=B.Y AND B.X=A.Y
GROUP BY A.X,A.Y
HAVING COUNT(A.X)>1 OR A.X<A.Y
ORDER BY A.X;
```



X	Y
20	20
20	21
22	23

Q116. DATASET

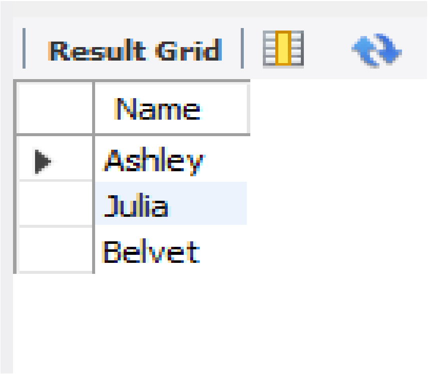
TABLE: Students

ID	Name	Marks
1	Ashley	81
2	Samantha	75
4	Julia	76
3	Belvet	84

Query the Name of any student in STUDENTS who scored higher than 75 Marks. Order your output by the last three characters of each name. If two or more students both have names ending in the same last three characters (i.e.: Bobby, Robby, etc.), secondary sort them by ascending ID.

Solution:

```
SELECT
        Name
FROM Students
WHERE Marks > 75
ORDER BY RIGHT(Name, 3), id ASC;
```



Name
Ashley
Julia
Belvet

Q117. DATASET

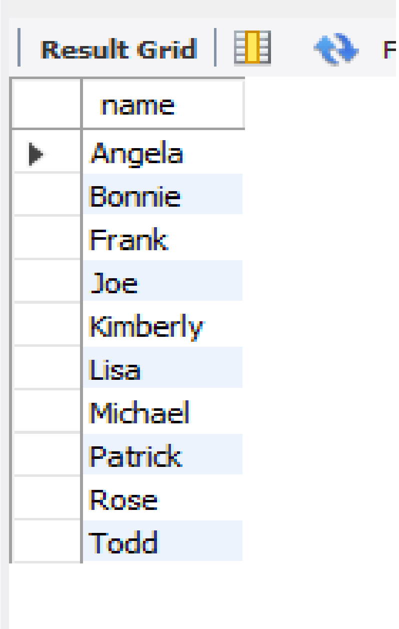
TABLE: Employee

employee_id	name	months	salary
12228	Rose	15	1968
33645	Angela	1	3443
45692	Frank	17	4608
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

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

Solution:

```
SELECT
        name
FROM Employee
ORDER BY name ASC;
```



```
name
Angela
Bonnie
Frank
Joe
Kimberly
Lisa
Michael
Patrick
Rose
Todd
```

Q118. DATASET

TABLE: Triangles

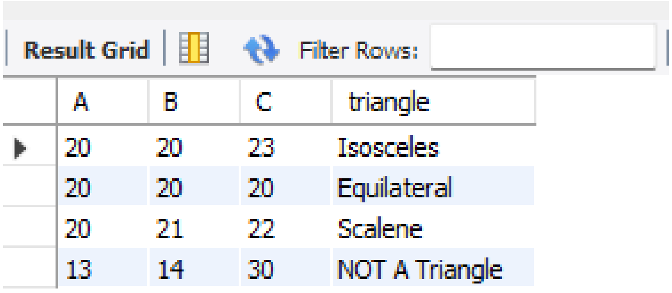
A	B	C
20	20	23
20	20	20
20	21	22
13	14	30

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

Write a query identifying the type of each record in the TRIANGLES table using its three side lengths.
Output one of the following statements for each record in the table:
● Equilateral: It's a triangle with sides of equal length.
● Isosceles: It's a triangle with sides of equal length.
● Scalene: It's a triangle with sides of differing lengths.
● Not A Triangle: The given values of A, B, and C don't form a triangle.

Solution:

```
SELECT
          A,
           B,
            C,
          CASE WHEN A+B<=C OR B+C<=A OR A+C<=B THEN  'NOT A
          Triangle'
                      WHEN A = B AND B=C AND A = C THEN 'Equilateral'
                     WHEN A = B OR B = C OR A = C THEN 'Isosceles'
                    ELSE 'Scalene' END  as triangle
FROM Triangles;
```



A	B	C	triangle
20	20	23	Isosceles
20	20	20	Equilateral
20	21	22	Scalene
13	14	30	NOT A Triangle

Q119. DATASET

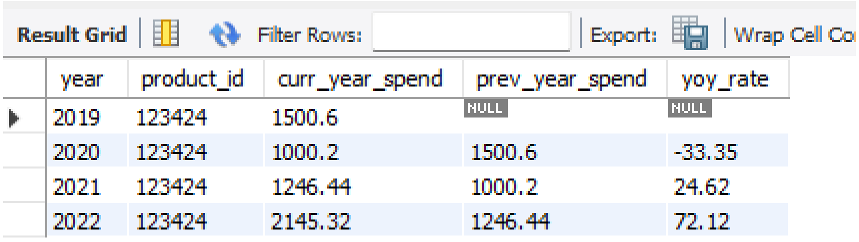
TABLE: user_transactions

transaction_id	product_id	spend	transaction_date
1341	123424	1500.6	31-12-2019 12:00
1423	123424	1000.2	31-12-2020 12:00
1623	123424	1246.44	31-12-2021 12:00
1322	123424	2145.32	31-12-2022 12:00

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).

Solution:

```
SELECT *,
        ROUND(((curr_year_spend - prev_year_spend) * 100) /
        prev_year_spend, 2) AS yoy_rate
FROM
(SELECT EXTRACT(YEAR FROM transaction_date) AS year,
       product_id,
       spend AS curr_year_spend,
       ROUND(
        LAG(spend) OVER(PARTITION BY product_id ORDER BY
        EXTRACT(YEAR FROM transaction_date)), 2) AS prev_year_spend
 FROM user_transactions
)t
```



year	product_id	curr_year_spend	prev_year_spend	yoy_rate
2019	123424	1500.6	NULL	NULL
2020	123424	1000.2	1500.6	-33.35
2021	123424	1246.44	1000.2	24.62
2022	123424	2145.32	1246.44	72.12

Q120. DATASET

TABLE: inventory

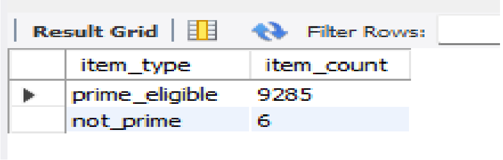
item_id	item_type	item_category	square_footage
1374	prime_eligible	mini refrigerator	68
4245	not_prime	standing lamp	26.4
2452	prime_eligible	television	85
3255	not_prime	side table	22.6
1672	prime_eligible	laptop	8.5

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.

Solution:

```
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 AS item_count
FROM
inventory
GROUP BY item_type
ORDER BY item_type desc;
```



item_type	item_count
prime_eligible	9285
not_prime	6