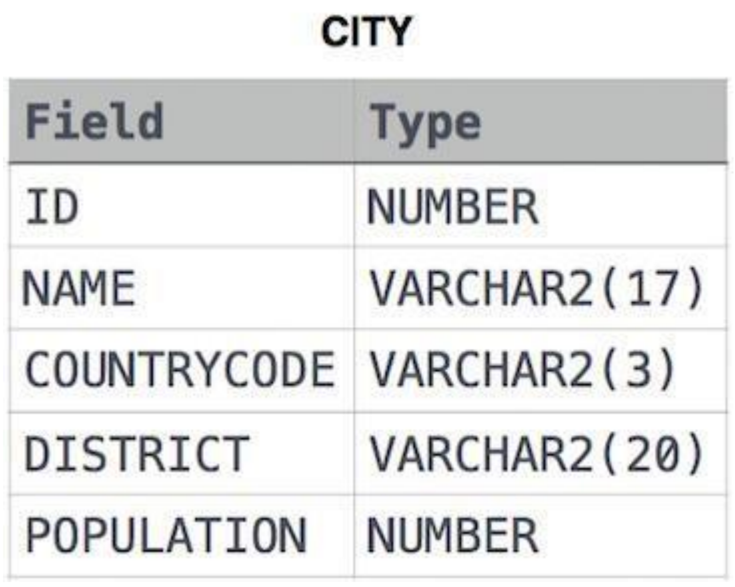
MYSQL Assignment

-JS Mohammed Hafeez

21WU0101015

B.Tech, CSE

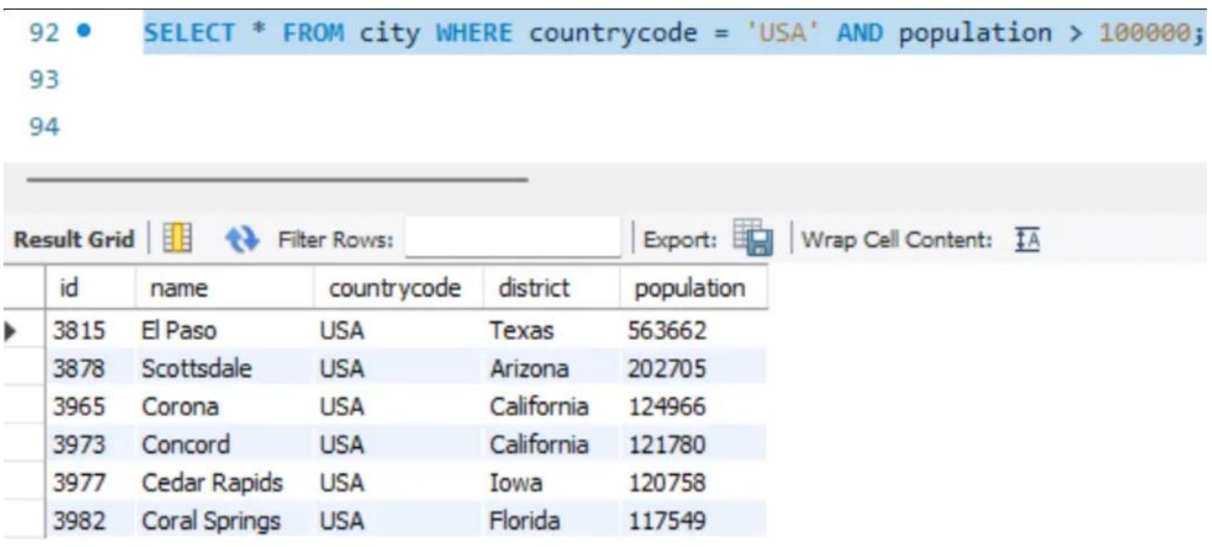
Q1. Query all columns for all American cities in the CITY table with populations larger than 100000.
The CountryCode for America is USA.
The CITY table is described as follows:



CITY

Field	Type
ID	NUMBER
NAME	VARCHAR2(17)
COUNTRYCODE	VARCHAR2(3)
DISTRICT	VARCHAR2(20)
POPULATION	NUMBER

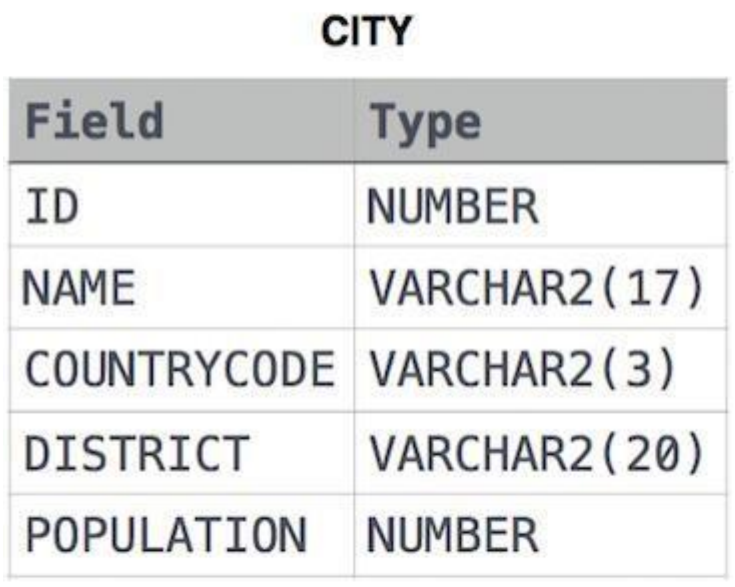
SELECT * FROM city WHERE countrycode = 'USA' AND population > 100000;



```
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
```

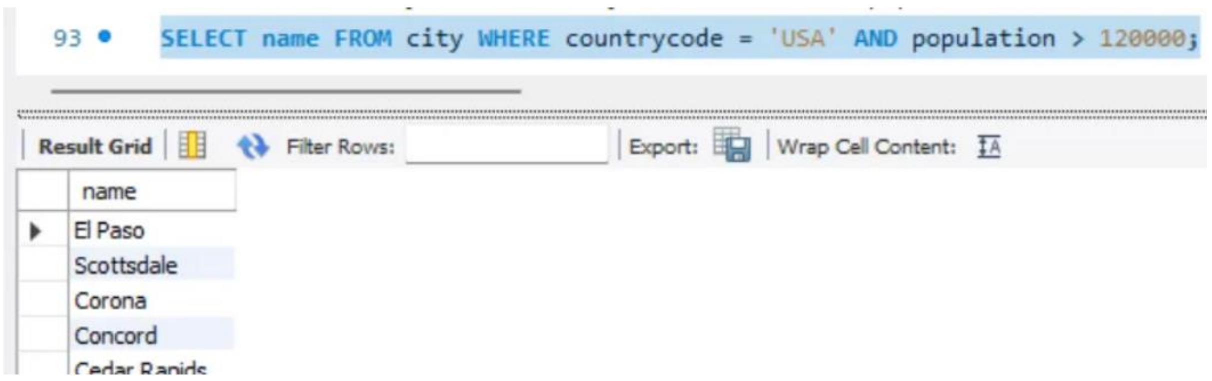
Q2. Query the NAME field for all American cities in the CITY table with populations larger than 120000. The CountryCode for America is USA.
The CITY table is described as follows:



CITY

Field	Type
ID	NUMBER
NAME	VARCHAR2(17)
COUNTRYCODE	VARCHAR2(3)
DISTRICT	VARCHAR2(20)
POPULATION	NUMBER

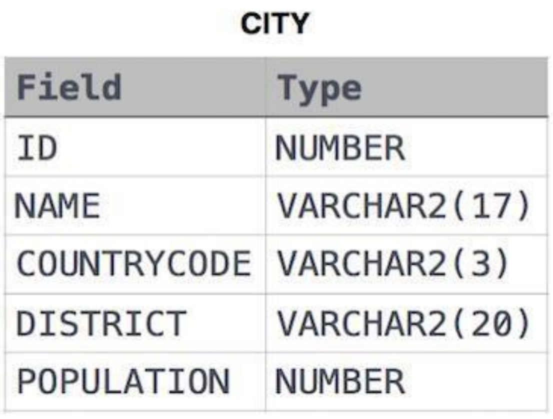
SELECT name FROM city WHERE countrycode = 'USA' AND population > 120000;



```
SELECT name FROM city WHERE countrycode = 'USA' AND population > 120000;
```

```
name
El Paso
Scottsdale
Corona
Concord
Cedar Rapids
```

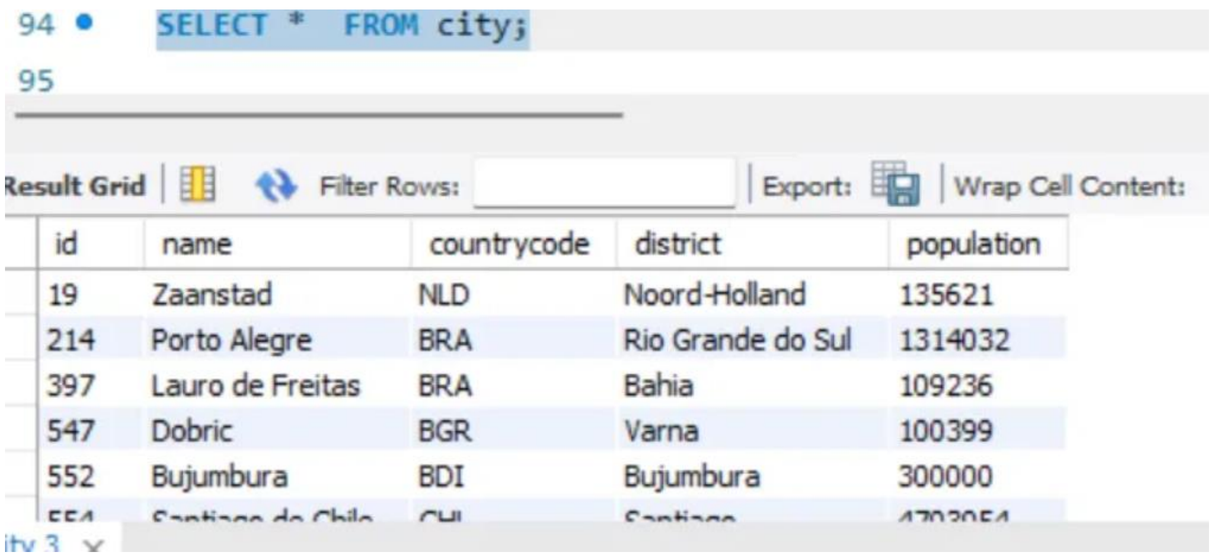
Q3. Query all columns (attributes) for every row in the CITY table. The CITY table is described as follows:



CITY

Field	Type
ID	NUMBER
NAME	VARCHAR2(17)
COUNTRYCODE	VARCHAR2(3)
DISTRICT	VARCHAR2(20)
POPULATION	NUMBER

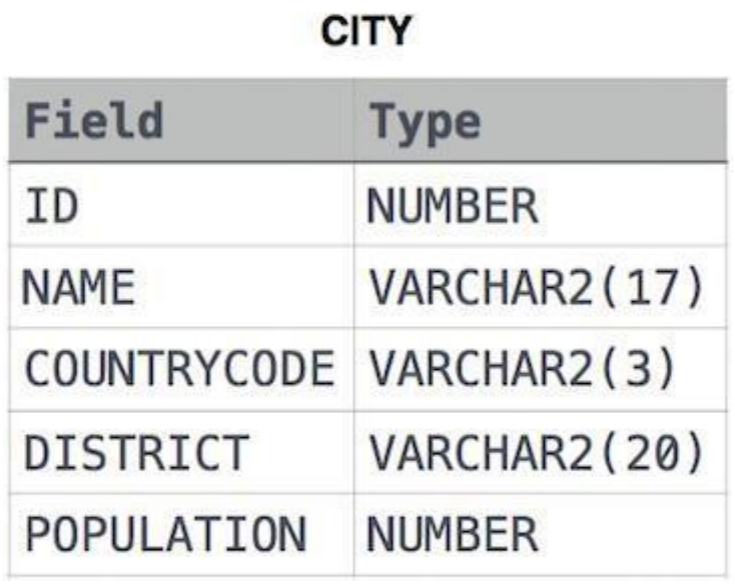
SELECT * FROM city;



```
SELECT *  FROM city;
```

id	name	countrycode	district	population
19	Zaanstad	NLD	Noord-Holland	135621
214	Porto Alegre	BRA	Rio Grande do Sul	1314032
397	Lauro de Freitas	BRA	Bahia	109236
547	Dobric	BGR	Varna	100399
552	Bujumbura	BDI	Bujumbura	300000
554	Santiago de Chile	CHL	Santiago	4703954

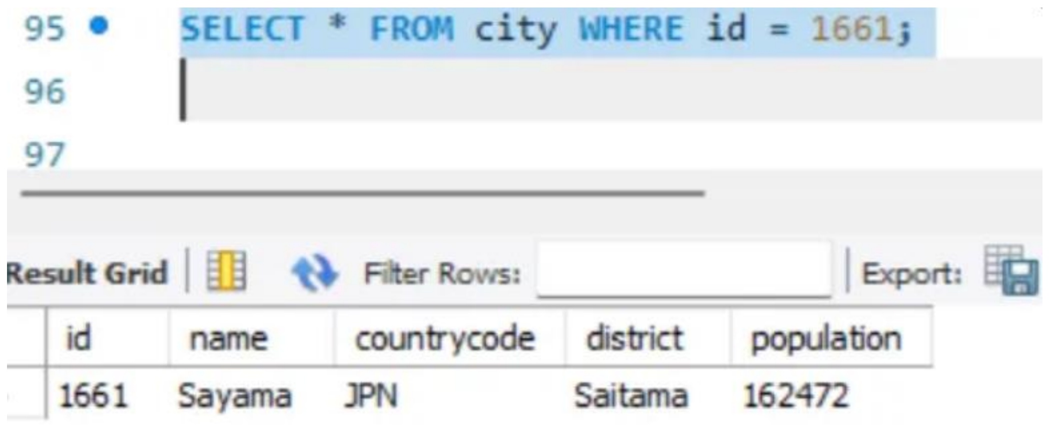
Q4. Query all columns for a city in CITY with the ID 1661. The CITY table is described as follows:



CITY

Field	Type
ID	NUMBER
NAME	VARCHAR2(17)
COUNTRYCODE	VARCHAR2(3)
DISTRICT	VARCHAR2(20)
POPULATION	NUMBER

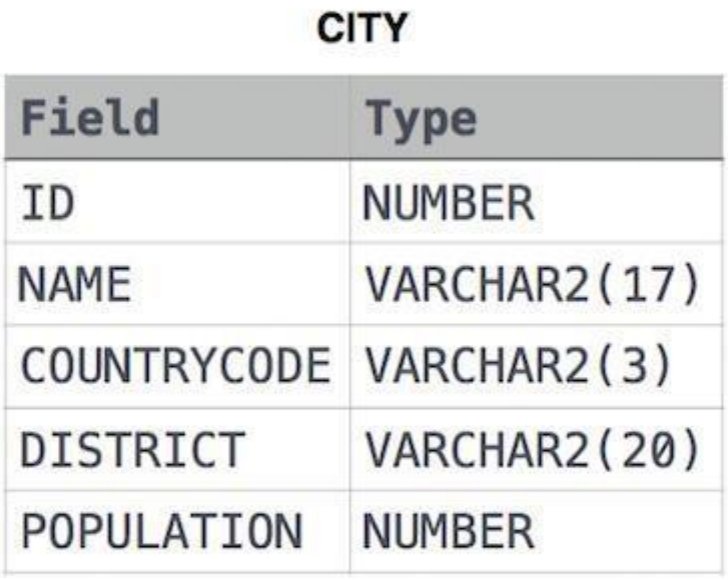
SELECT * FROM city WHERE id = 1661;



```
SELECT * FROM city WHERE id = 1661;
```

id	name	countrycode	district	population
1661	Sayama	JPN	Saitama	162472

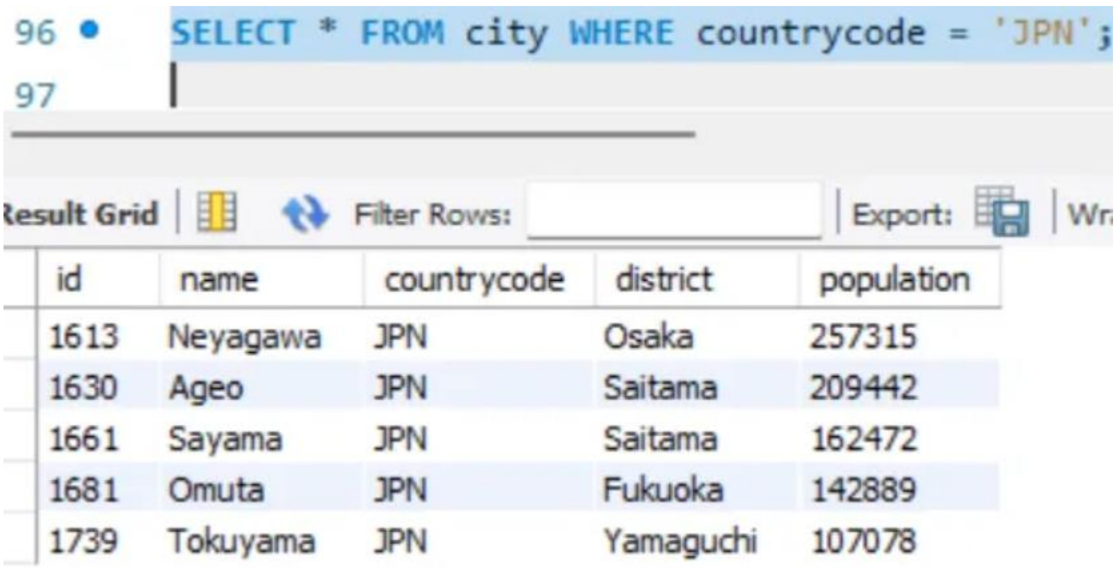
Q5. Query all attributes of every Japanese city in the CITY table. The COUNTRYCODE for Japan is JPN. The CITY table is described as follows:



CITY

Field	Type
ID	NUMBER
NAME	VARCHAR2(17)
COUNTRYCODE	VARCHAR2(3)
DISTRICT	VARCHAR2(20)
POPULATION	NUMBER

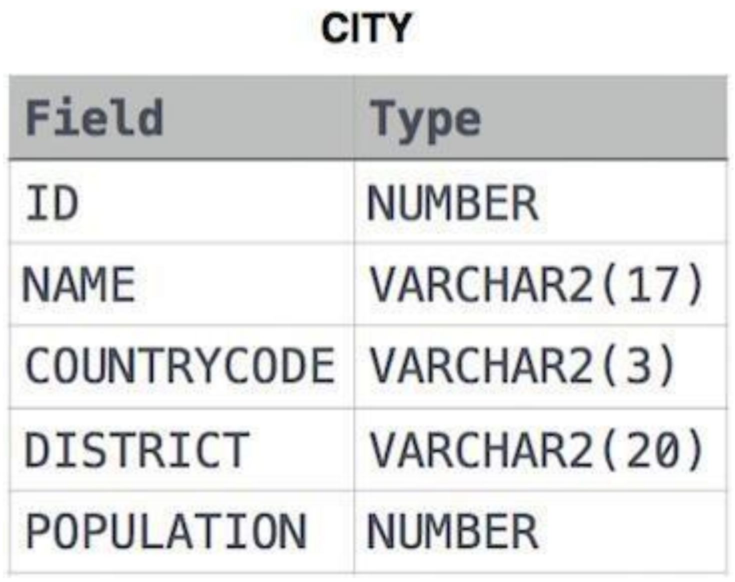
SELECT * FROM city WHERE countrycode = 'JPN';



```
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

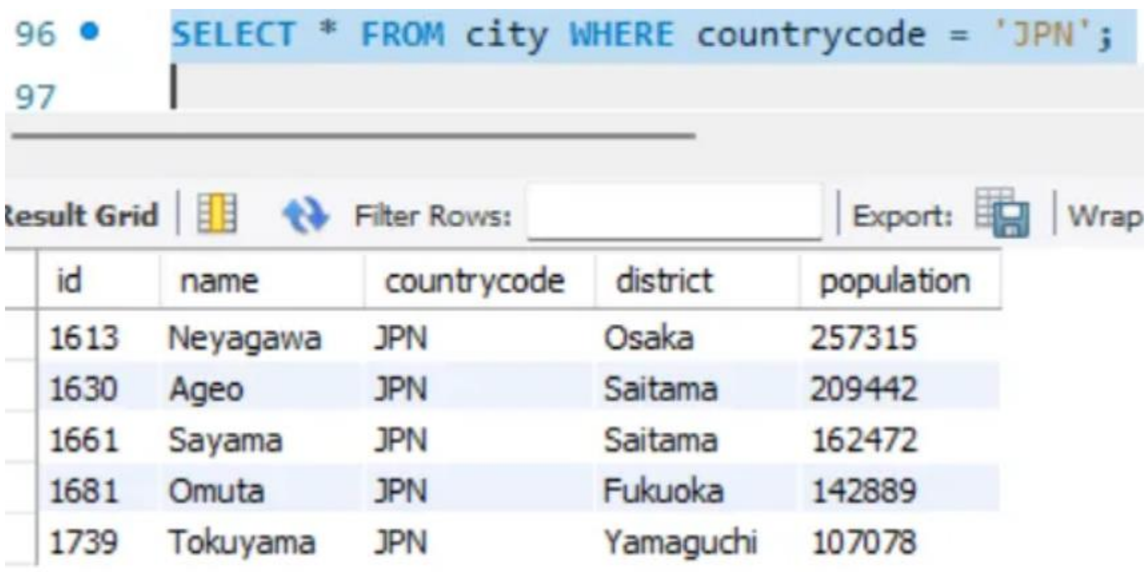
Q6. Query the names of all the Japanese cities in the CITY table. The COUNTRYCODE for Japan is JPN.
The CITY table is described as follows:



CITY

Field	Type
ID	NUMBER
NAME	VARCHAR2(17)
COUNTRYCODE	VARCHAR2(3)
DISTRICT	VARCHAR2(20)
POPULATION	NUMBER

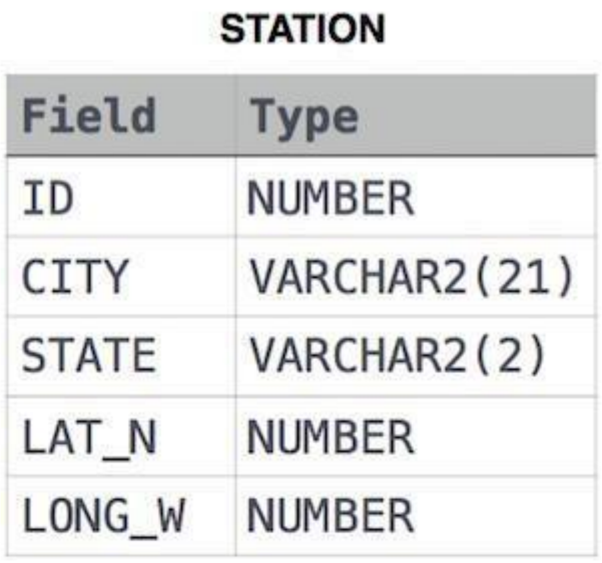
SELECT * FROM city WHERE countrycode = 'JPN';



```sql
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

Q7. Query a list of CITY and STATE from the STATION table. The STATION table is described as follows:

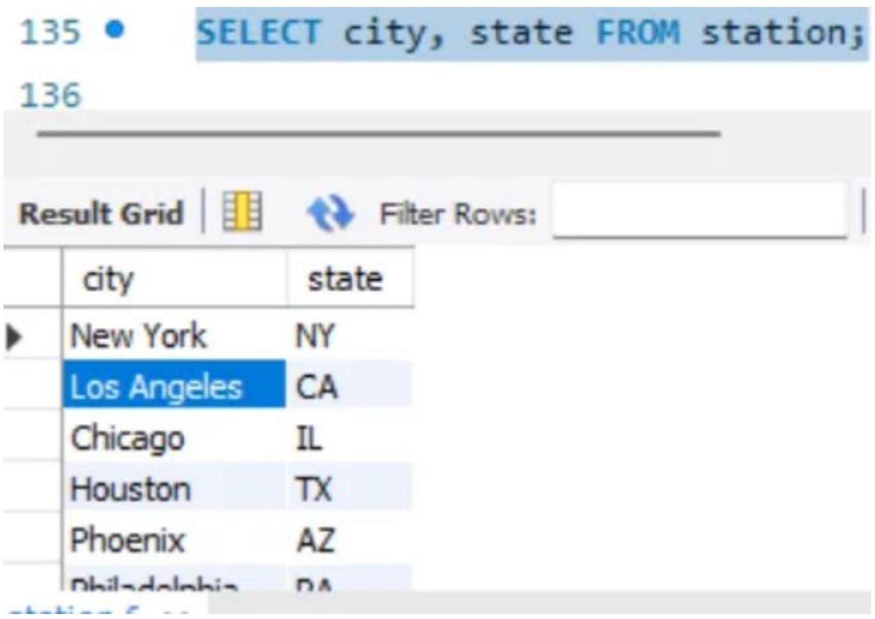


STATION

Field	Type
ID	NUMBER
CITY	VARCHAR2(21)
STATE	VARCHAR2(2)
LAT_N	NUMBER
LONG_W	NUMBER

where LAT_N is the northern latitude and LONG_W is the western longitude.

SELECT city, state FROM station;

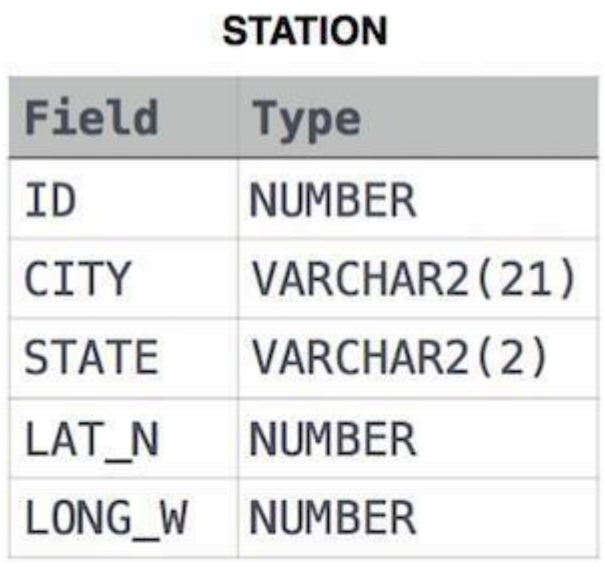


```
SELECT city, state FROM station;
```

Result Grid | Filter Rows:

```
city         state
New York     NY
Los Angeles  CA
Chicago      IL
Houston      TX
Phoenix      AZ
```

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. The STATION table is described as follows:

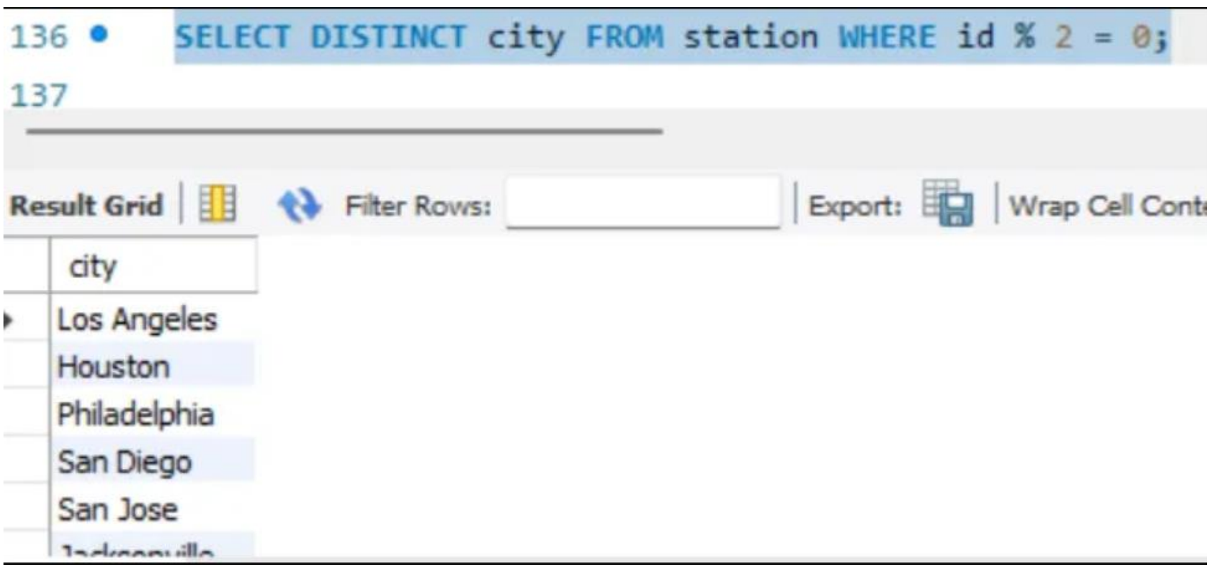


STATION

Field	Type
ID	NUMBER
CITY	VARCHAR2(21)
STATE	VARCHAR2(2)
LAT_N	NUMBER
LONG_W	NUMBER

where LAT_N is the northern latitude and LONG_W is the western longitude

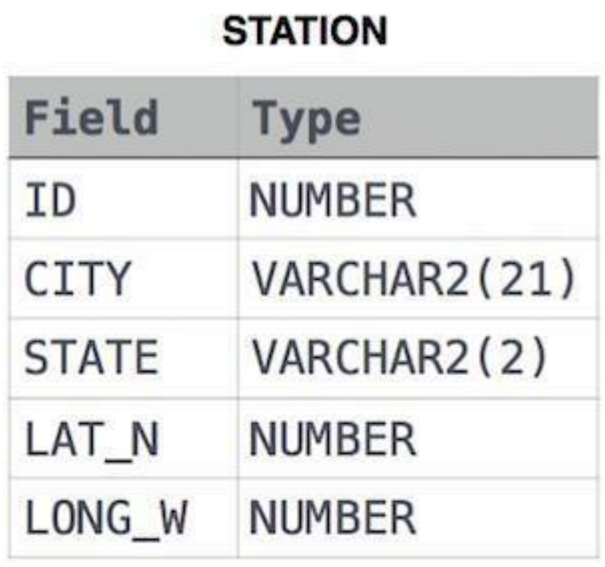
SELECT DISTINCT city FROM station WHERE id % 2 = 0;



```
SELECT DISTINCT city FROM station WHERE id % 2 = 0;
```

```
city
Los Angeles
Houston
Philadelphia
San Diego
San Jose
```

Q9. Find the difference between the total number of CITY entries in the table and the number of distinct CITY entries in the table.
The STATION table is described as follows:



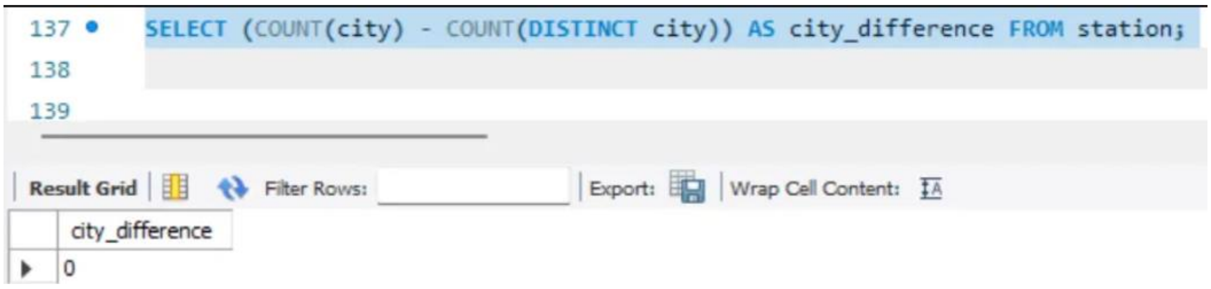
STATION

Field	Type
ID	NUMBER
CITY	VARCHAR2(21)
STATE	VARCHAR2(2)
LAT_N	NUMBER
LONG_W	NUMBER

where LAT_N is the northern latitude and LONG_W is the western longitude.

For example, if there are three records in the table with CITY values 'New York', 'New York', 'Bengalaru', there are 2 different city names: 'New York' and 'Bengalaru'. The query returns , because total number of records - number of unique city names = 3-2 =1

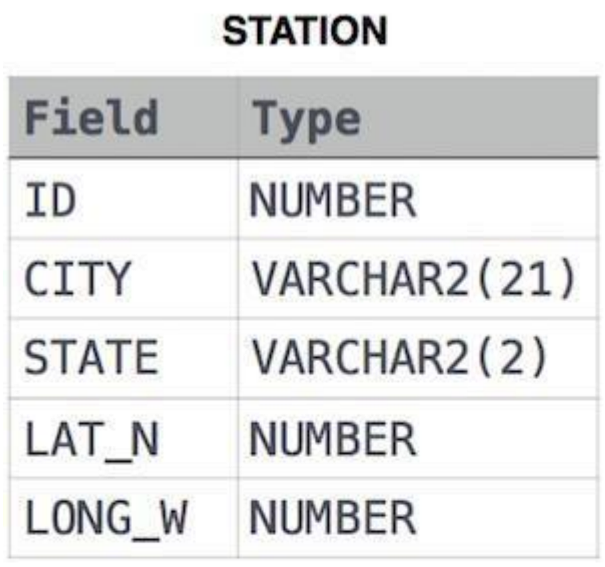
SELECT (COUNT(city) - COUNT(DISTINCT city)) AS city_difference FROM station;



```
SELECT (COUNT(city) - COUNT(DISTINCT city)) AS city_difference FROM station;
```

```
city_difference
0
```

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



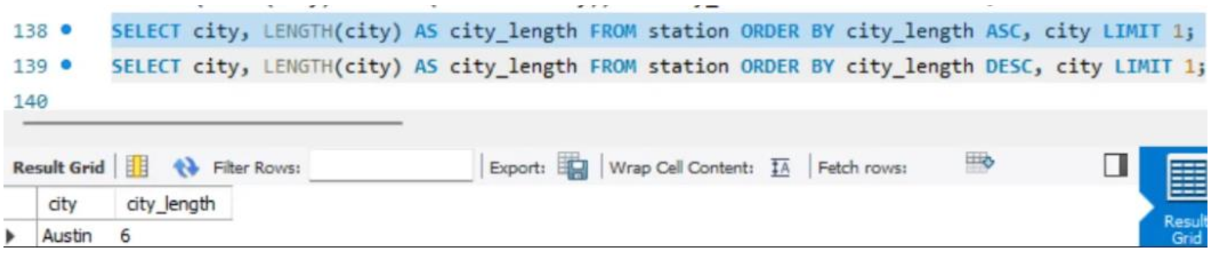
STATION

Field	Type
ID	NUMBER
CITY	VARCHAR2(21)
STATE	VARCHAR2(2)
LAT_N	NUMBER
LONG_W	NUMBER

where LAT_N is the northern latitude and LONG_W is the western longitude.
Sample Input
For example, CITY has four entries: DEF, ABC, PQRS and WXY.
Sample Output
ABC 3
PQRS 4

Hint -
When ordered alphabetically, the CITY names are listed as ABC, DEF, PQRS, and WXY, with lengths and. The longest name is PQRS, but there are options for shortest named city. Choose ABC, because it comes first alphabetically.
Note
You can write two separate queries to get the desired output. It need not be a single query.

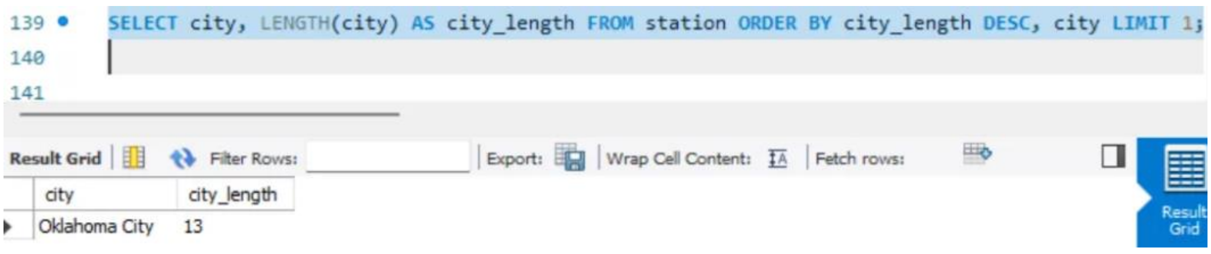
SELECT city, LENGTH(city) AS city_length FROM station ORDER BY city_length ASC, city LIMIT 1; -- shortest city



```
SELECT city, LENGTH(city) AS city_length FROM station ORDER BY city_length ASC, city LIMIT 1;
SELECT city, LENGTH(city) AS city_length FROM station ORDER BY city_length DESC, city LIMIT 1;
```

city	city_length
Austin	6

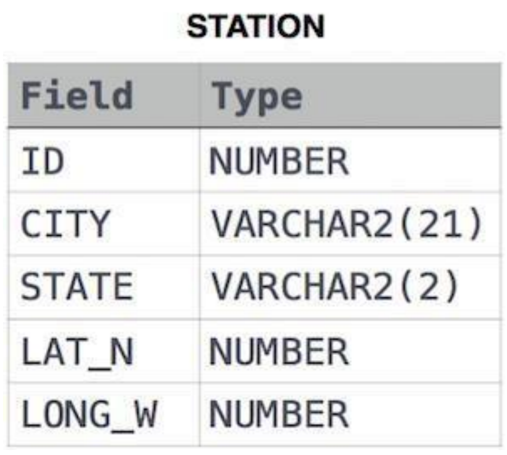
SELECT city, LENGTH(city) AS city_length FROM station ORDER BY city_length DESC, city LIMIT 1; -- longest city



```
SELECT city, LENGTH(city) AS city_length FROM station ORDER BY city_length DESC, city LIMIT 1;
```

```
city          city_length
Oklahoma City 13
```

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.
Input Format
The STATION table is described as follows:

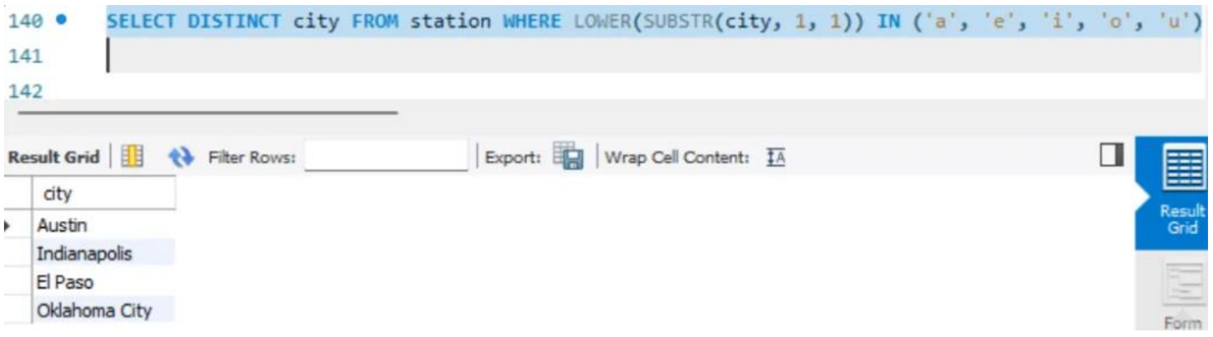


STATION

Field	Type
ID	NUMBER
CITY	VARCHAR2(21)
STATE	VARCHAR2(2)
LAT_N	NUMBER
LONG_W	NUMBER

where LAT_N is the northern latitude and LONG_W is the western longitude.

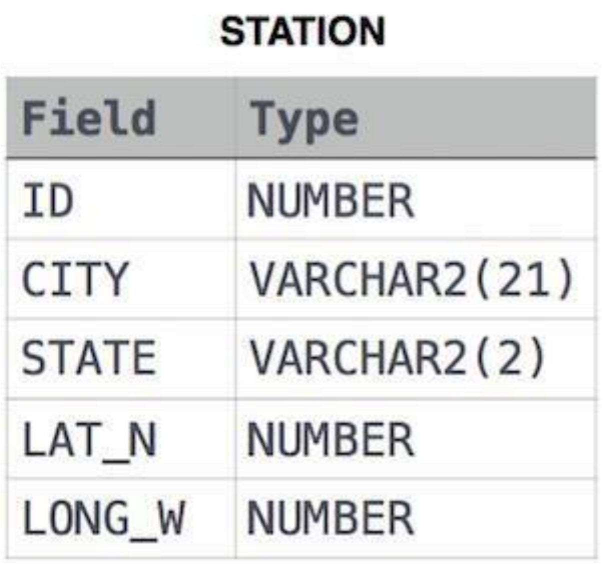
SELECT DISTINCT city FROM station WHERE LOWER(SUBSTR(city, 1, 1)) IN ('a', 'e', 'i', 'o', 'u');



```
SELECT DISTINCT city FROM station WHERE LOWER(SUBSTR(city, 1, 1)) IN ('a', 'e', 'i', 'o', 'u')
```

```
city
Austin
Indianapolis
El Paso
Oklahoma City
```

Q12. Query the list of CITY names ending with vowels (a, e, i, o, u) from STATION. Your result cannot contain duplicates.
Input Format
The STATION table is described as follows:

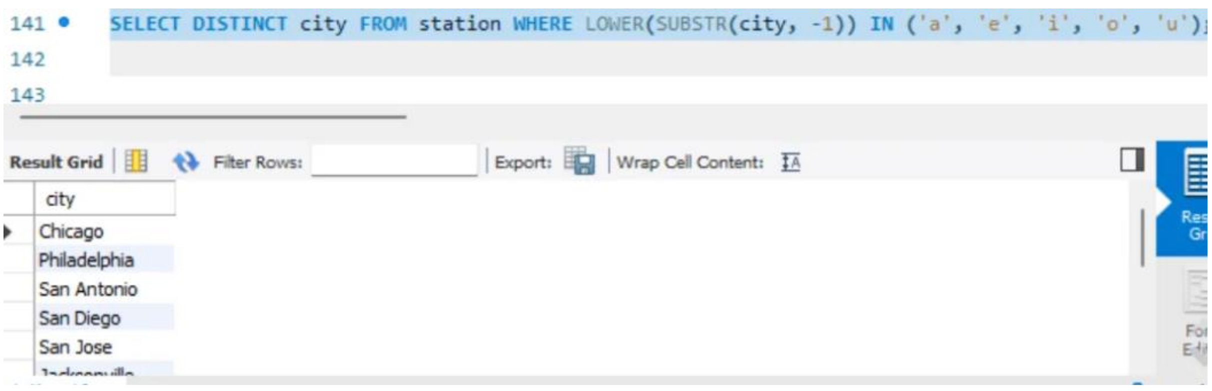


STATION

Field	Type
ID	NUMBER
CITY	VARCHAR2(21)
STATE	VARCHAR2(2)
LAT_N	NUMBER
LONG_W	NUMBER

where LAT_N is the northern latitude and LONG_W is the western longitude.

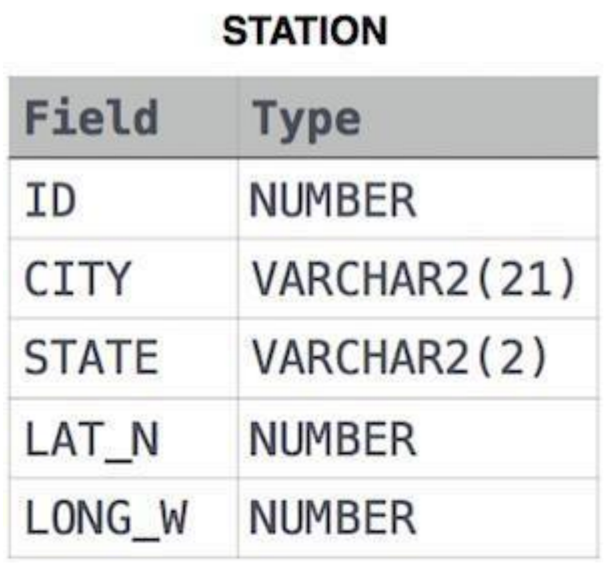
SELECT DISTINCT city FROM station WHERE LOWER(SUBSTR(city, -1)) IN ('a', 'e', 'i', 'o', 'u');



```
SELECT DISTINCT city FROM station WHERE LOWER(SUBSTR(city, -1)) IN ('a', 'e', 'i', 'o', 'u');
```

```
city
Chicago
Philadelphia
San Antonio
San Diego
San Jose
```

Q13. Query the list of CITY names from STATION that do not start with vowels. Your result cannot contain duplicates.
Input Format
The STATION table is described as follows:

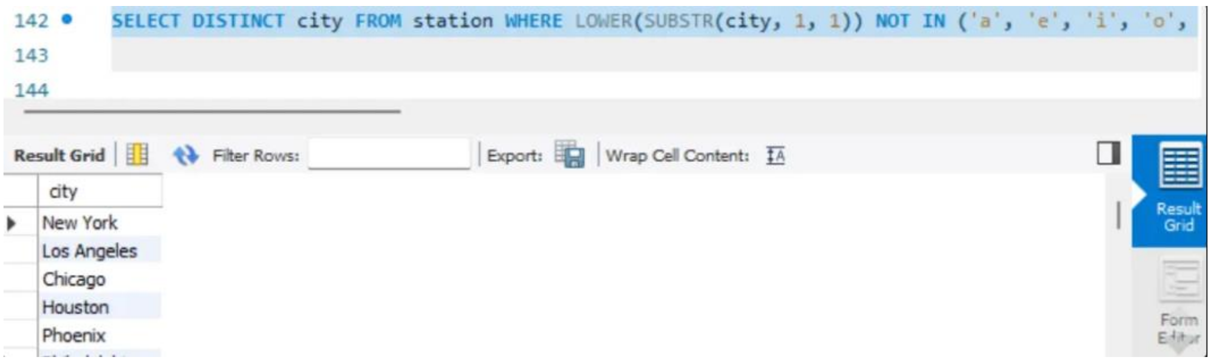


STATION

Field	Type
ID	NUMBER
CITY	VARCHAR2(21)
STATE	VARCHAR2(2)
LAT_N	NUMBER
LONG_W	NUMBER

where LAT_N is the northern latitude and LONG_W is the western longitude.

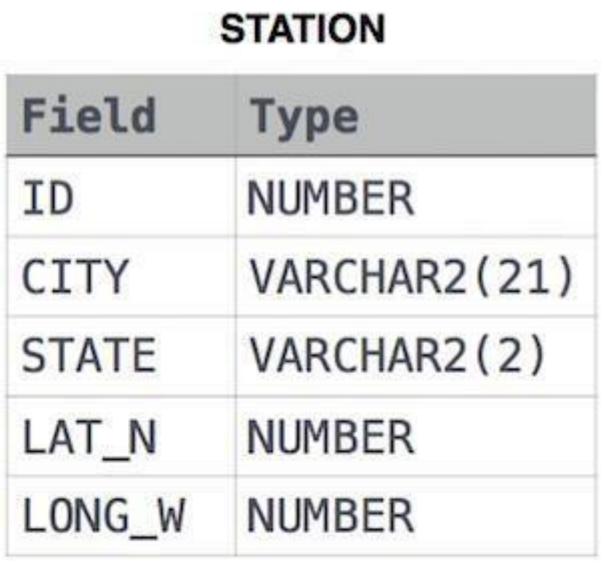
SELECT DISTINCT city FROM station WHERE LOWER(SUBSTR(city, 1, 1)) NOT IN ('a', 'e', 'i', 'o', 'u');



```
SELECT DISTINCT city FROM station WHERE LOWER(SUBSTR(city, 1, 1)) NOT IN ('a', 'e', 'i', 'o',
```

```
city
New York
Los Angeles
Chicago
Houston
Phoenix
```

Q14. Query the list of CITY names from STATION that do not end with vowels. Your result cannot contain duplicates.
Input Format
The STATION table is described as follows:

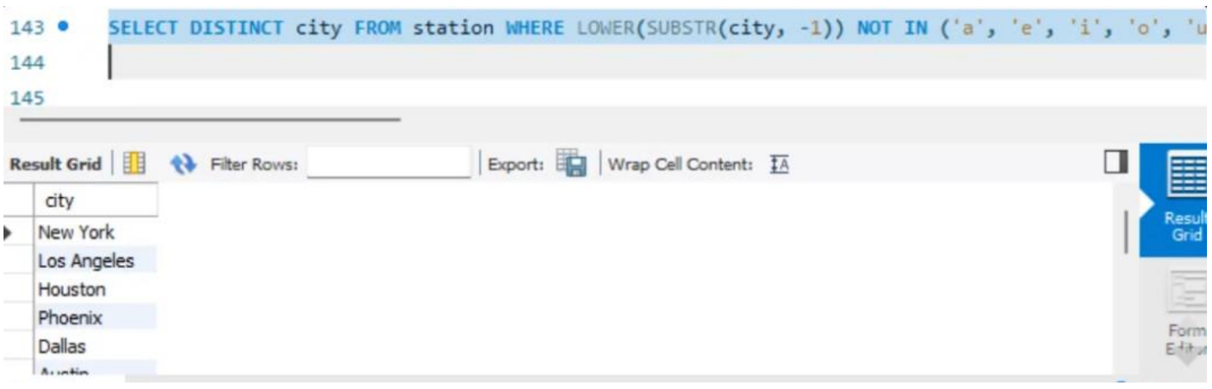


STATION

Field	Type
ID	NUMBER
CITY	VARCHAR2(21)
STATE	VARCHAR2(2)
LAT_N	NUMBER
LONG_W	NUMBER

where LAT_N is the northern latitude and LONG_W is the western longitude.

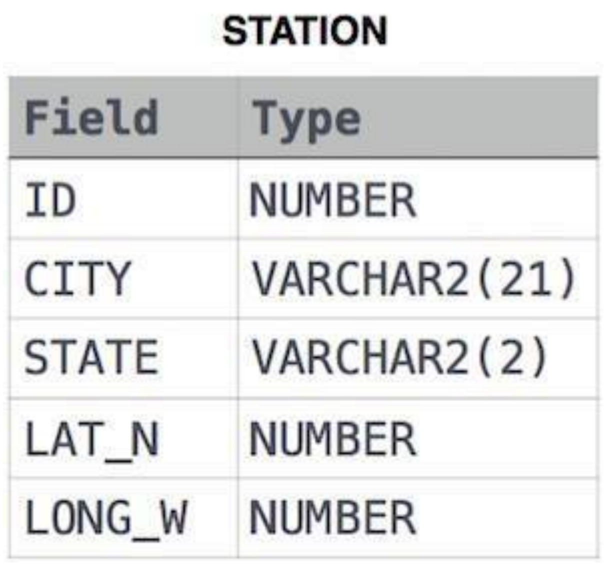
SELECT DISTINCT city FROM station WHERE LOWER(SUBSTR(city, -1)) NOT IN ('a', 'e', 'i', 'o', 'u');



```
SELECT DISTINCT city FROM station WHERE LOWER(SUBSTR(city, -1)) NOT IN ('a', 'e', 'i', 'o', 'u
```

```
city
New York
Los Angeles
Houston
Phoenix
Dallas
```

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.
Input Format
The STATION table is described as follows:

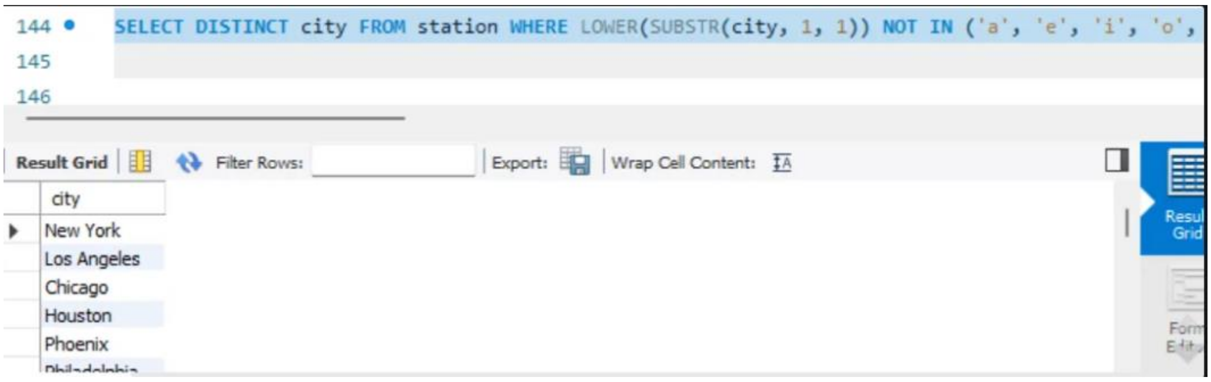


STATION

Field	Type
ID	NUMBER
CITY	VARCHAR2(21)
STATE	VARCHAR2(2)
LAT_N	NUMBER
LONG_W	NUMBER

where LAT_N is the northern latitude and LONG_W is the western longitude.

SELECT DISTINCT city FROM station WHERE LOWER(SUBSTR(city, 1, 1)) NOT IN ('a', 'e', 'i', 'o', 'u') OR LOWER(SUBSTR(city, -1)) NOT IN ('a', 'e', 'i', 'o', 'u');



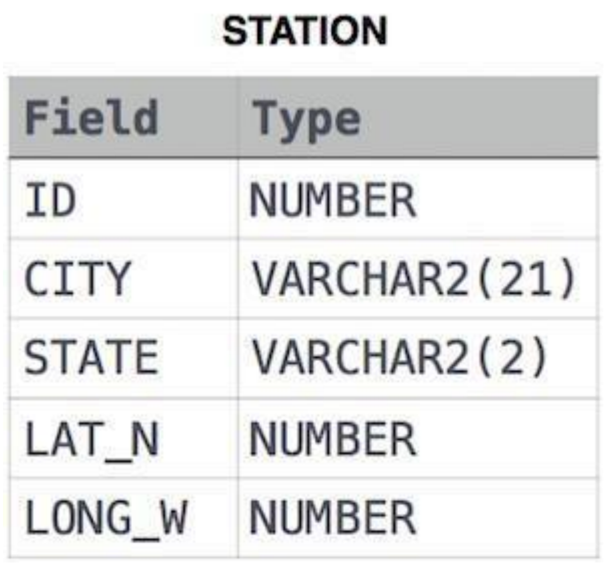
```
SELECT DISTINCT city FROM station WHERE LOWER(SUBSTR(city, 1, 1)) NOT IN ('a', 'e', 'i', 'o',
```

```
city
New York
Los Angeles
Chicago
Houston
Phoenix
```

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.

Input Format

The STATION table is described as follows:

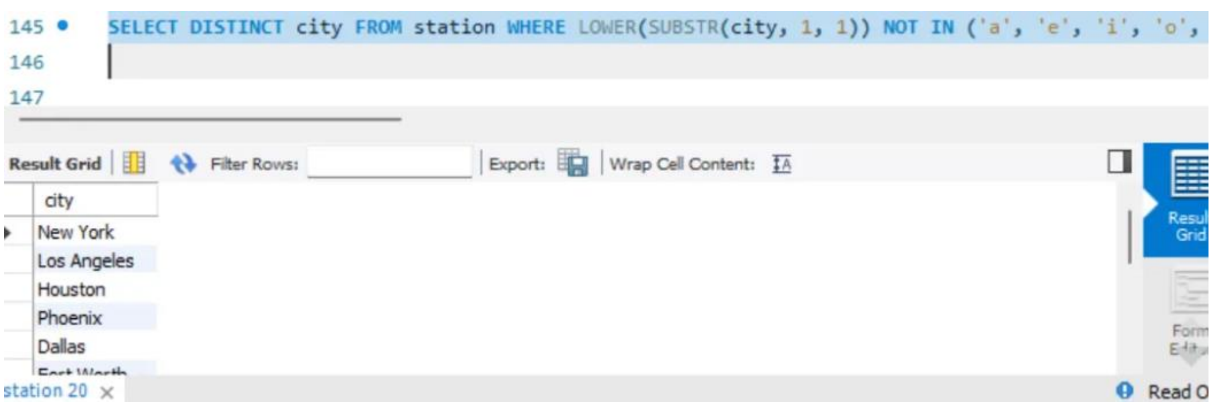


STATION

Field	Type
ID	NUMBER
CITY	VARCHAR2(21)
STATE	VARCHAR2(2)
LAT_N	NUMBER
LONG_W	NUMBER

where LAT_N is the northern latitude and LONG_W is the western longitude.

SELECT DISTINCT city FROM station WHERE LOWER(SUBSTR(city, 1, 1)) NOT IN ('a', 'e', 'i', 'o', 'u') AND LOWER(SUBSTR(city, -1)) NOT IN ('a', 'e', 'i', 'o', 'u');



```
SELECT DISTINCT city FROM station WHERE LOWER(SUBSTR(city, 1, 1)) NOT IN ('a', 'e', 'i', 'o',
```

```
city
New York
Los Angeles
Houston
Phoenix
Dallas
```

Q17.

Table: Product

Column Name	Type
product_id	int
product_name	varchar
unit_price	int

product_id is the primary key of this table.

Each row of this table indicates the name and the price of each product.

Table: Sales

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

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

Write an SQL query that reports the 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.
The query result format is in the following example.

Input:
Product table:

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

Sales table:

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

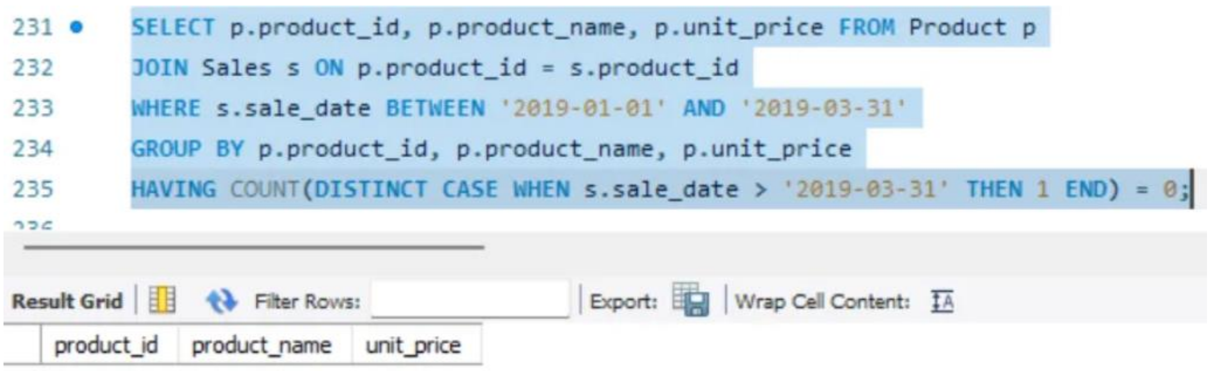
Output:

product_id	product_name
1	S8

Explanation:
The product with id 1 was only sold in the spring of 2019.
The product with id 2 was sold in the spring of 2019 but was also sold after the spring of 2019.

The product with id 3 was sold after spring 2019.
We return only product 1 as it is the product that was only sold in the spring of 2019.

SELECT p.product_id, p.product_name, p.unit_price FROM Product p
JOIN Sales s ON p.product_id = s.product_id
WHERE s.sale_date BETWEEN '2019-01-01' AND '2019-03-31'
GROUP BY p.product_id, p.product_name, p.unit_price
HAVING COUNT(DISTINCT CASE WHEN s.sale_date > '2019-03-31' THEN 1 END) = 0;



```
SELECT p.product_id, p.product_name, p.unit_price FROM Product p
JOIN Sales s ON p.product_id = s.product_id
WHERE s.sale_date BETWEEN '2019-01-01' AND '2019-03-31'
GROUP BY p.product_id, p.product_name, p.unit_price
HAVING COUNT(DISTINCT CASE WHEN s.sale_date > '2019-03-31' THEN 1 END) = 0;
```

product_id	product_name	unit_price

Q18.

Table: Views

Column Name	Type
article_id	int
author_id	int
viewer_id	int
view_date	date

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. The
query result format is in the following example.

Input:
Views table:

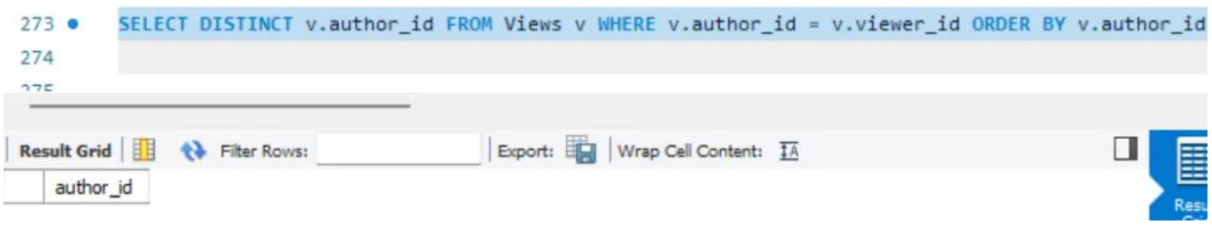
article_id	author_id	viewer_id	view_date
1	3	5	2019-08-01

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

Output:

id
4
7

SELECT DISTINCT v.author_id FROM Views v WHERE v.author_id = v.viewer_id ORDER BY v.author_id ASC;



```
SELECT DISTINCT v.author_id FROM Views v WHERE v.author_id = v.viewer_id ORDER BY v.author_id
```

```
author_id
```

Q19.

Table: Delivery

Column Name	Type
delivery_id	int
customer_id	int
order_date	date
customer_pref_delivery_date	date

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.
The query result format is in the following example.

Input:
Delivery table:

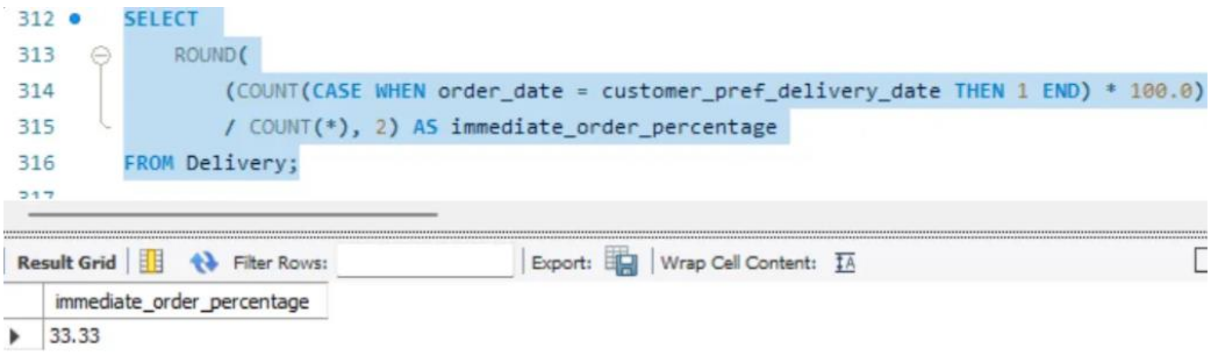
delivery_id	customer_id	order_date	customer_pref_delivery_date
1	1	2019-08-01	2019-08-02
2	5	2019-08-02	2019-08-02
3	1	2019-08-11	2019-08-11
4	3	2019-08-24	2019-08-26
5	4	2019-08-21	2019-08-22
6	2	2019-08-11	2019-08-13

Output:

immediate_percentage
33.33

Explanation: The orders with delivery id 2 and 3 are immediate while the others are scheduled.

```
SELECT
   ROUND(
      (COUNT(CASE WHEN order_date = customer_pref_delivery_date THEN 1
END) * 100.0)
      / COUNT(*), 2) AS immediate_order_percentage
FROM Delivery;
```



```
SELECT
    ROUND(
        (COUNT(CASE WHEN order_date = customer_pref_delivery_date THEN 1 END) * 100.0)
        / COUNT(*), 2) AS immediate_order_percentage
FROM Delivery;
```

immediate_order_percentage
33.33

Q20.

Table: Ads

Column Name	Type
ad_id	int
user_id	int
action	enum

(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.
The query result format is in the following example.

Input:
Ads table:

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

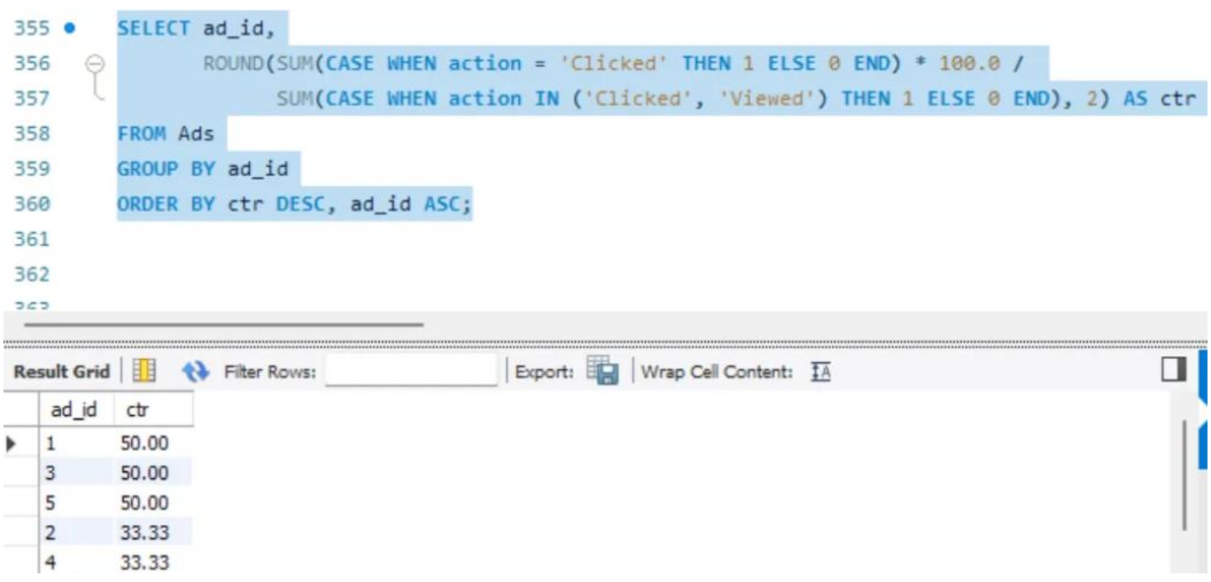
Output:

ad_id	ctr
1	66.67

3	50
2	33.33
5	0

Explanation:
for ad_id = 1, ctr = (2/(2+1)) * 100 = 66.67 for ad_id = 2, ctr = (1/(1+2)) * 100 = 33.33 for ad_id = 3, ctr = (1/(1+1)) * 100 = 50.00 for ad_id = 5, ctr = 0.00, Note that ad_id = 5 has no clicks or views. Note that we do not care about Ignored Ads.

```
SELECT ad_id,
      ROUND(SUM(CASE WHEN action = 'Clicked'
THEN 1 ELSE 0 END) * 100.0 /
          SUM(CASE WHEN action IN ('Clicked',
'Viewed') THEN 1 ELSE 0 END), 2) AS ctr
FROM Ads
GROUP BY ad_id
ORDER BY ctr DESC, ad_id ASC;
```



```
SELECT ad_id,
       ROUND(SUM(CASE WHEN action = 'Clicked' THEN 1 ELSE 0 END) * 100.0 /
             SUM(CASE WHEN action IN ('Clicked', 'Viewed') THEN 1 ELSE 0 END), 2) AS ctr
FROM Ads
GROUP BY ad_id
ORDER BY ctr DESC, ad_id ASC;
```

ad_id	ctr
1	50.00
3	50.00
5	50.00
2	33.33
4	33.33

Q21.

Table: Employee

Column Name	Type
employee_id	int
team_id	int

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

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

Input:
Employee Table:

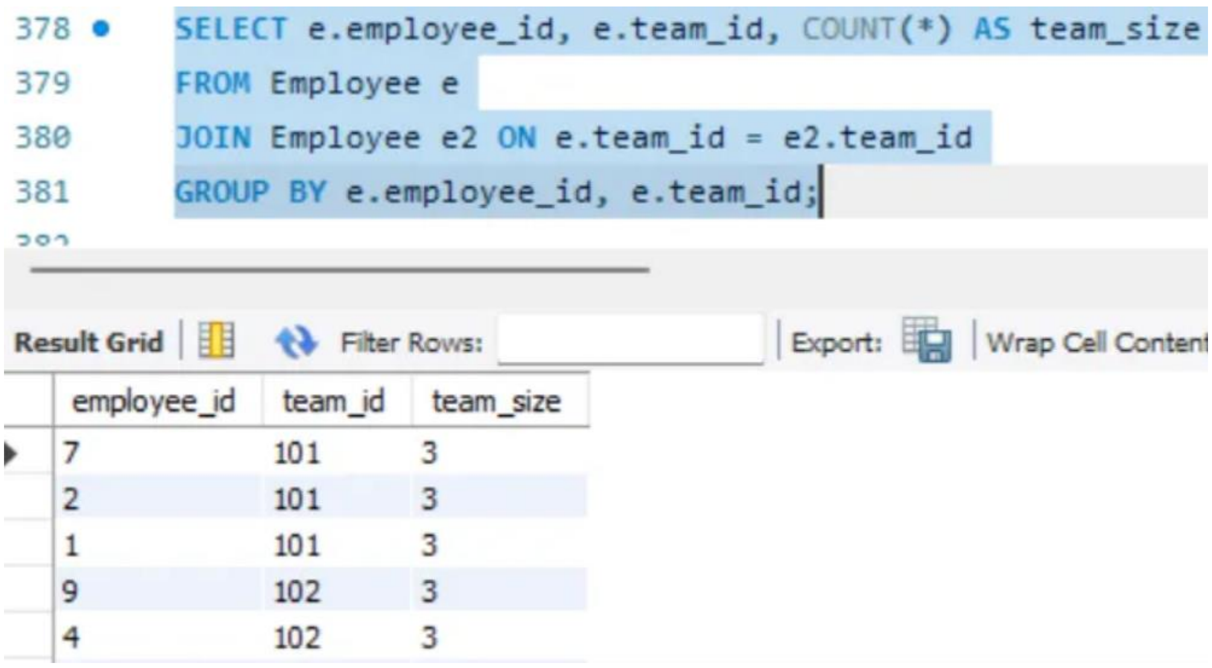
employee_id	team_id
1	8
2	8
3	8
4	7
5	9
6	9

Output:

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

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

```
SELECT e.employee_id, e.team_id, COUNT(*) AS team_size
FROM Employee e
JOIN Employee e2 ON e.team_id = e2.team_id
GROUP BY e.employee_id, e.team_id;
```



```
SELECT e.employee_id, e.team_id, COUNT(*) AS team_size
FROM Employee e
JOIN Employee e2 ON e.team_id = e2.team_id
GROUP BY e.employee_id, e.team_id;
```

employee_id	team_id	team_size
7	101	3
2	101	3
1	101	3
9	102	3
4	102	3

Q22.

Table: Countries

Column Name	Type
country_id	int
country_name	varchar

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

Table: Weather

Column Name	Type
country_id	int
weather_state	int
day	date

(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.

The query result format is in the following example.

Input:
Countries table:

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

Weather table:

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

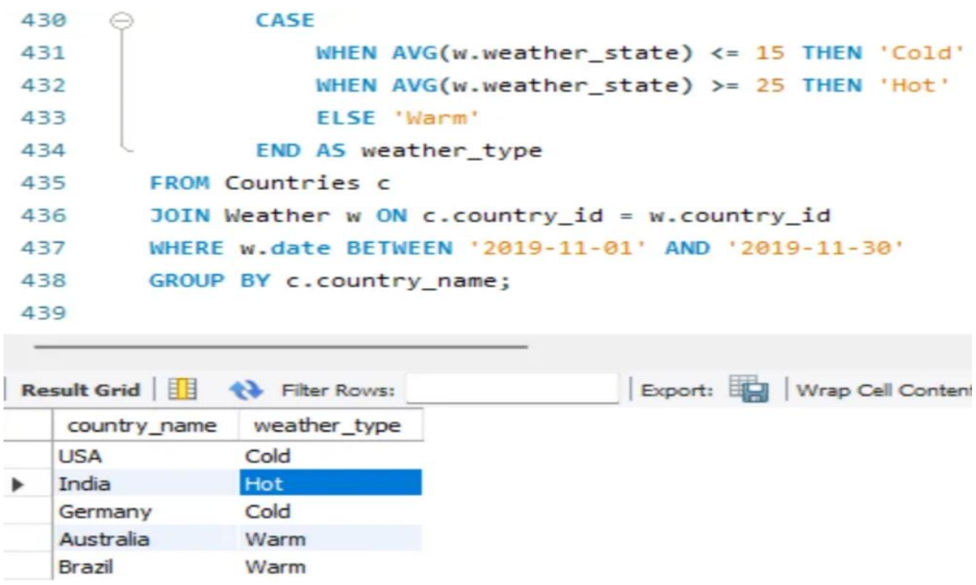
Output:

country_name	weather_type

USA	Cold
Australia	Cold
Peru	Hot
Morocco	Hot
China	Warm

Explanation:
Average weather_state in the USA in November is (15) / 1 = 15 so the weather type is Cold.
Average weather_state in Australia in November is (-2 + 0 + 3) / 3 = 0.333 so the weather type is Cold.
Average weather_state in Peru in November is (25) / 1 = 25 so the weather type is Hot.
The average weather_state in China in November is (16 + 18 + 21) / 3 = 18.333 so the weather type is warm.
Average weather_state in Morocco in November is (25 + 27 + 31) / 3 = 27.667 so the weather type is Hot.
We know nothing about the average weather_state in Spain in November so we do not include it in the result table.

```
SELECT c.country_name,
    CASE
      WHEN AVG(w.weather_state) <= 15 THEN 'Cold'
      WHEN AVG(w.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 w.date BETWEEN '2019-11-01' AND '2019-11-30'
GROUP BY c.country_name;
```



```
        CASE
            WHEN AVG(w.weather_state) <= 15 THEN 'Cold'
            WHEN AVG(w.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 w.date BETWEEN '2019-11-01' AND '2019-11-30'
    GROUP BY c.country_name;
```

country_name	weather_type
USA	Cold
India	Hot
Germany	Cold
Australia	Warm
Brazil	Warm

Q23.

Table: Prices

Column Name	Type
product_id	int
start_date	date
end_date	date
price	int

(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: UnitsSold

Column Name	Type
product_id	int
purchase_date	date
units	int

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.
The query result format is in the following example.
Input:
Prices table:

product_id	start_date	end_date	price
1	2019-02-17	2019-02-28	5
1	2019-03-01	2019-03-22	20
2	2019-02-01	2019-02-20	15
2	2019-02-21	2019-03-31	30

UnitsSold table:

product_id	purchase_date	units
1	2019-02-25	100

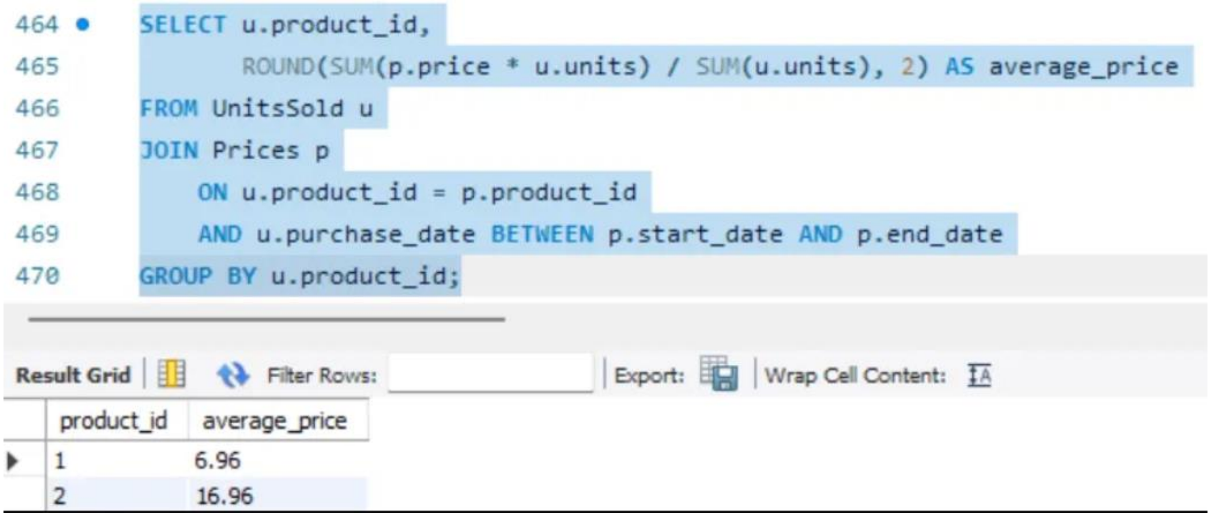
1	2019-03-01	15
2	2019-02-10	200
2	2019-03-22	30

Output:

product_id	average_price
1	6.96
2	16.96

Explanation:
Average selling price = Total Price of Product / Number of products sold.
Average selling price for product 1 = ((100 * 5) + (15 * 20)) / 115 = 6.96 Average selling price for product 2 = ((200 * 15) + (30 * 30)) / 230 = 16.96

```
SELECT u.product_id,
      ROUND(SUM(p.price * u.units) / SUM(u.units), 2) AS
average_price
FROM UnitsSold u
JOIN Prices p
   ON u.product_id = p.product_id
   AND u.purchase_date BETWEEN p.start_date AND
p.end_date
GROUP BY u.product_id;
```



```
SELECT u.product_id,
       ROUND(SUM(p.price * u.units) / SUM(u.units), 2) AS average_price
FROM UnitsSold u
JOIN Prices p
    ON u.product_id = p.product_id
    AND u.purchase_date BETWEEN p.start_date AND p.end_date
GROUP BY u.product_id;
```

product_id	average_price
1	6.96
2	16.96

Q24.

Table: Activity

Column Name	Type
player_id	int
device_id	int
event_date	date
games_played	int

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

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

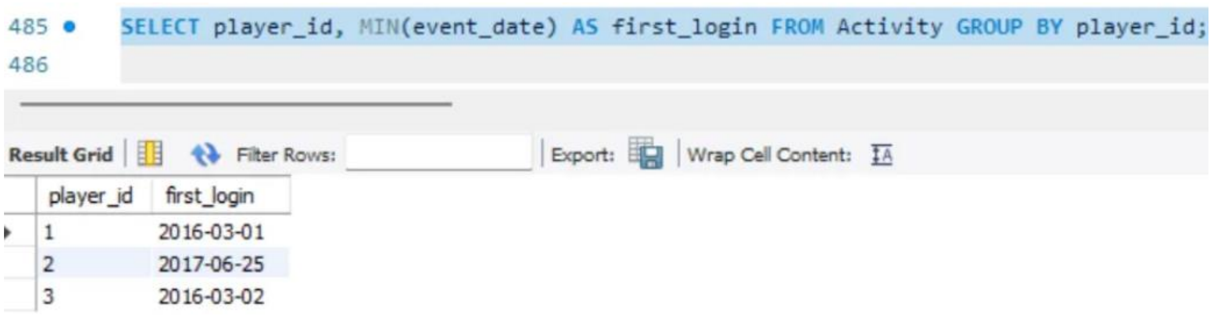
Input:
Activity table:

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

Output:

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

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



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

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

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

Q25.

Table: Activity

Column Name	Type
player_id	int
device_id	int
event_date	date
games_played	int

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

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

Input:
Activity table:

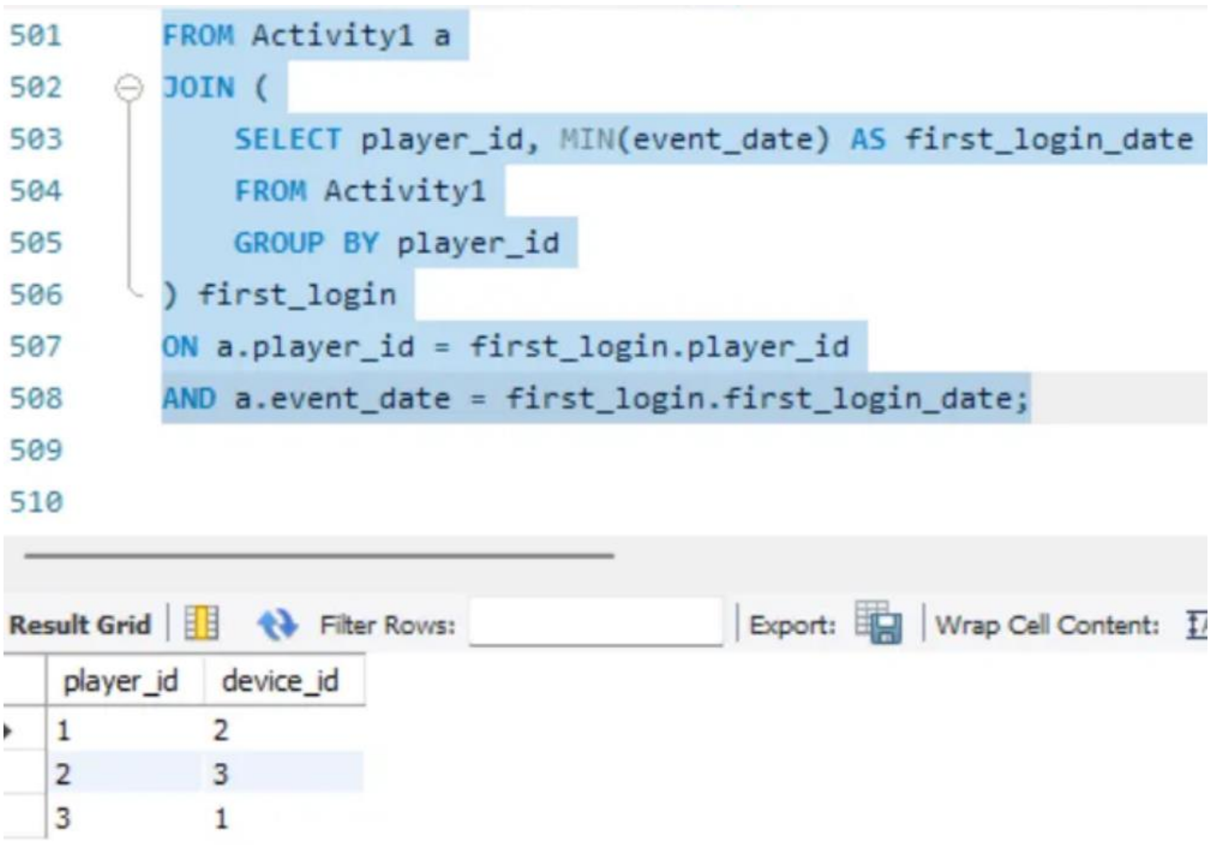
player_id	device_id	event_date	games_played
1	2	2016-03-01	5
1	2	2016-05-02	6
2	3	2017-06-25	1
3	1	2016-03-02	0
3	4	2018-07-03	5

Output:

player_id	device_id
1	2
2	3

3	1

```
SELECT a.player_id, a.device_id
FROM Activity1 a
JOIN (
   SELECT player_id, MIN(event_date) AS first_login_date
   FROM Activity1
   GROUP BY player_id
) first_login
ON a.player_id = first_login.player_id
AND a.event_date = first_login.first_login_date;
```



```
FROM Activity1 a
JOIN (
    SELECT player_id, MIN(event_date) AS first_login_date
    FROM Activity1
    GROUP BY player_id
) first_login
ON a.player_id = first_login.player_id
AND a.event_date = first_login.first_login_date;
```

```
player_id  device_id
1          2
2          3
3          1
```

Q26.

Table: Products

Column Name	Type
product_id	int
product_name	varchar

product_category	varchar

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

Table: Orders

Column Name	Type
product_id	int
order_date	date
unit	int

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.
The query result format is in the following example.

Input:
Products table:

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

Orders table:

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

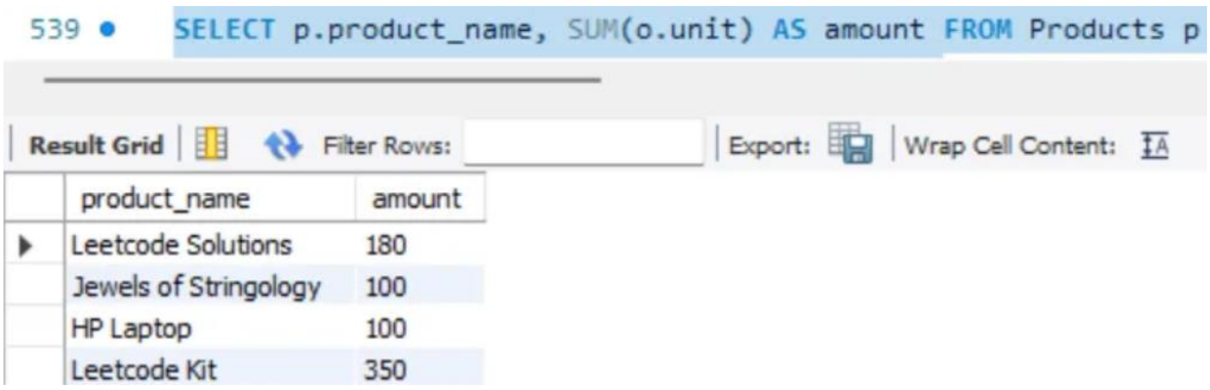
3	2020-02-24	3
4	2020-03-01	20
4	2020-03-04	30
4	2020-03-04	60
5	2020-02-25	50
5	2020-02-27	50
5	2020-03-01	50

Output:

product_name	unit
Leetcode Solutions	130
Leetcode Kit	100

Explanation:
Products with product_id = 1 is ordered in February a total of (60 + 70) = 130.
Products with product_id = 2 is ordered in February a total of 80.
Products with product_id = 3 is ordered in February a total of (2 + 3) = 5.
Products with product_id = 4 was not ordered in February 2020.
Products with product_id = 5 is ordered in February a total of (50 + 50) = 100.

```
SELECT p.product_name, SUM(o.unit) AS amount FROM Products p
JOIN Orders o ON p.product_id = o.product_id
WHERE o.order_date BETWEEN '2020-02-01' AND '2020-02-29'
GROUP BY p.product_name
HAVING SUM(o.unit) >= 100;
```


```
SELECT p.product_name, SUM(o.unit) AS amount FROM Products p
```

product_name	amount
Leetcode Solutions	180
Jewels of Stringology	100
HP Laptop	100
Leetcode Kit	350

Q27.

Table: Users

Column Name	Type
user_id	int
name	varchar
mail	varchar

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.
The query result format is in the following example.
Input:
Users table:

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

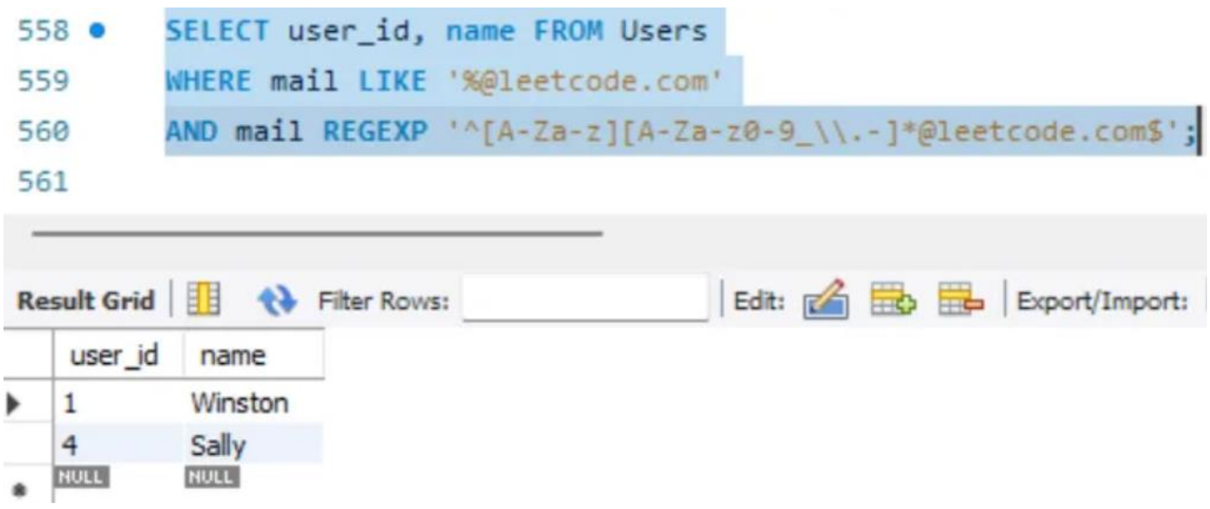
Output:

user_id	name	mail
1	Winston	winston@leetcode.com
3	Annabelle	bella-@leetcode.com

4	Sally	sally.come@leetcode.com

Explanation:
The mail of user 2 does not have a domain.
The mail of user 5 has the # sign which is not allowed. The mail of user 6 does not have the leetcode domain. The mail of user 7 starts with a period.

```
SELECT user_id, name FROM Users
WHERE mail LIKE '%@leetcode.com'
AND mail REGEXP '^[A-Za-z][A-Za-z0-9_\\.-]*@leetcode.com$';
```



```
SELECT user_id, name FROM Users
WHERE mail LIKE '%@leetcode.com'
AND mail REGEXP '^[A-Za-z][A-Za-z0-9_\\.-]*@leetcode.com$';
```

user_id	name
1	Winston
4	Sally
NULL	NULL

Q28.

Table: Customers

Column Name	Type
customer_id	int
name	varchar
country	varchar

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

Table: Product

Column Name	Type
customer_id	int

name	varchar
country	varchar

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:
Orders

Column Name	Type
order_id	int
customer_id	int
product_id	int
order_date	date
quantity	int

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.
The query result format is in the following example.

Input:
Customers table:

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

Product table:

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

Orders table:

order_id	customer_id	product_id	order_date	quantity

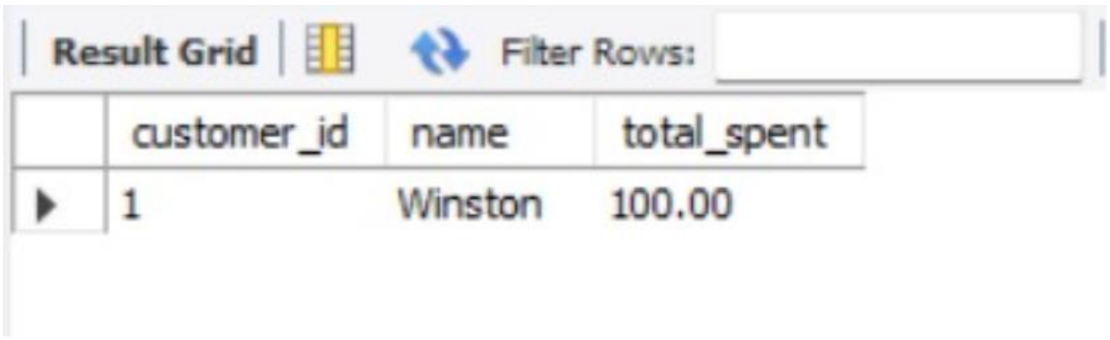
1	1	10	2020-06-10	1
2	1	20	2020-07-01	1
3	1	30	2020-07-08	2
4	2	10	2020-06-15	2
5	2	40	2020-07-01	10
6	3	20	2020-06-24	2
7	3	30	2020-06-25	2
9	3	30	2020-05-08	3

Output:

customer_id	name
1	Winston

Explanation:
Winston spent $300 (300 * 1) in June and $100 (10 * 1 + 45 * 2) in July 2020.
Jonathan spent $600 (300 * 2) in June and $20 (2 * 10) in July 2020. Moustafa spent
$110 (10 * 2 + 45 * 2) in June and $0 in July 2020.

```
SELECT
o.customer_id,
c.name,
SUM(p.price * o.quantity) AS total_spent
FROM
Orders o
JOIN
Customers c ON o.customer_id = c.customer_id
JOIN
Products p ON o.product_id = p.product_id
WHERE
YEAR(o.order_date) = 2020 AND MONTH(o.order_date) = 7
GROUP BY
o.customer_id, c.name
ORDER BY
total_spent DESC
LIMIT 1;
```



	customer_id	name	total_spent
▶	1	Winston	100.00

Q29.

Table: TVProgram

Column Name	Type
program_date	date
content_id	int
channel	varchar

(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: Content

Column Name	Type
content_id	varchar
title	varchar
Kids_content	enum
content_type	varchar

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.
The query result format is in the following example.

Input:
TVProgram table:

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

Content table:

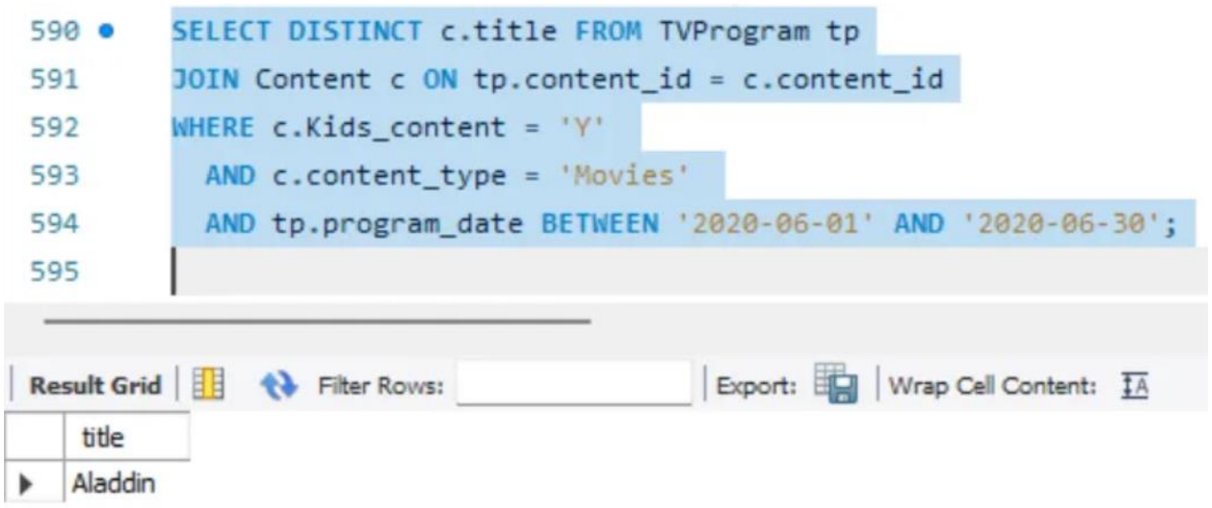
content_id	title	Kids_content	content_type
1	Leetcode Movie	N	Movies
2	Alg. for Kids	Y	Series
3	Database Sols	N	Series
4	Aladdin	Y	Movies
5	Cinderella	Y	Movies

Output:

title
Aladdin

Explanation:
"Leetcode Movie" is not a content for kids. "Alg.
for Kids" is not a movie.
"Database Sols" is not a movie
"Alladin" is a movie, content for kids and was streamed in June 2020. "Cinderella" was not streamed in June 2020.

```
SELECT DISTINCT c.title FROM TVProgram tp
JOIN Content c ON tp.content_id = c.content_id
WHERE c.Kids_content = 'Y'
  AND c.content_type = 'Movies'
  AND tp.program_date BETWEEN '2020-06-01' AND '2020-06-30';
```



```sql
SELECT DISTINCT c.title FROM TVProgram tp
JOIN Content c ON tp.content_id = c.content_id
WHERE c.Kids_content = 'Y'
  AND c.content_type = 'Movies'
  AND tp.program_date BETWEEN '2020-06-01' AND '2020-06-30';
```

```
title
Aladdin
```

Q30.
Table: NPV

Column Name	Type
id	int
year	int
npv	int

(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

Column Name	Type
id	int
year	int

(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.
The query result format is in the following example.

Input:
NPV table:

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

Queries table:

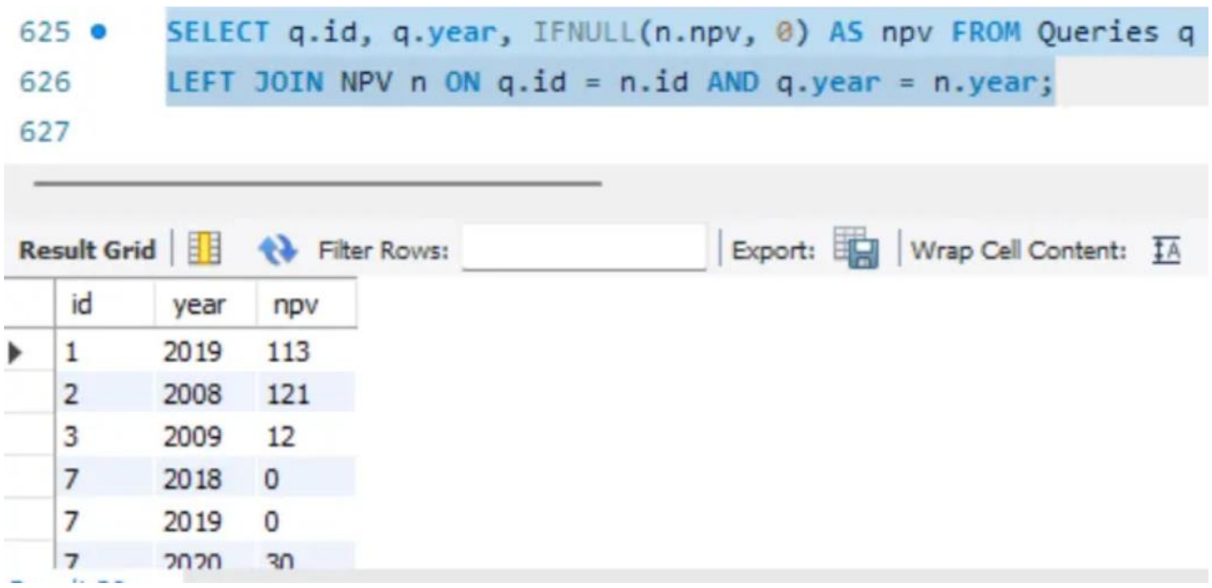
id	year
1	2019
2	2008
3	2009
7	2018
7	2019
7	2020
13	2019

Output:

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

Explanation:
The npv value of (7, 2018) is not present in the NPV table, we consider it 0. The npv values of all other queries can be found in the NPV table.

```
SELECT q.id, q.year, IFNULL(n.npv, 0) AS npv FROM Queries q
LEFT JOIN NPV n ON q.id = n.id AND q.year = n.year;
```



```
SELECT q.id, q.year, IFNULL(n.npv, 0) AS npv FROM Queries q
LEFT JOIN NPV n ON q.id = n.id AND q.year = n.year;
```

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

Q31.

Table: NPV

Column Name	Type
id	int
year	int
npv	int

(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

Column Name	Type
id	int
year	int

(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.
The query result format is in the following example.

Input:
NPV table:

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

Queries table:

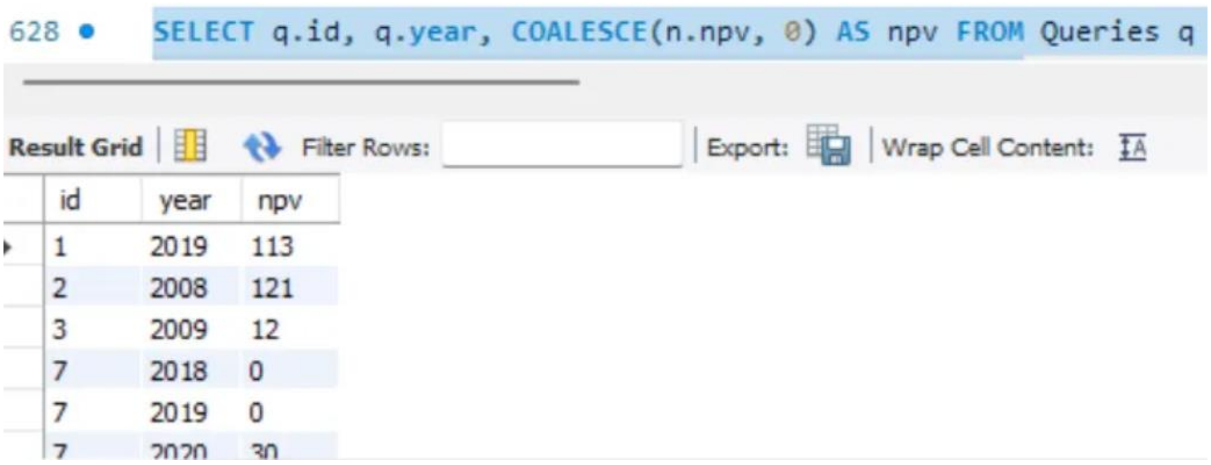
id	year
1	2019
2	2008
3	2009
7	2018
7	2019
7	2020
13	2019

Output:

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

Explanation:
The npv value of (7, 2018) is not present in the NPV table, we consider it 0. The npv values of all other queries can be found in the NPV table.

```
SELECT q.id, q.year, COALESCE(n.npv, 0) AS npv FROM
Queries q
LEFT JOIN NPV n ON q.id = n.id AND q.year = n.year;
```



```
SELECT q.id, q.year, COALESCE(n.npv, 0) AS npv FROM Queries q
```

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

Q32.

Table: Employees

Column Name	Type
id	int
name	varchar

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: EmployeeUNI

Column Name	Type
id	int
unique_id	int

(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.
The query result format is in the following example.

Input:
Employees table:

id	name
1	Alice
7	Bob
11	Meir

90	Winston
3	Jonathan

EmployeeUNI table:

id	unique_id
3	1
11	2
90	3

Output:

unique_id	name
null	Alice
null	Bob
2	Meir
3	Winston
1	Jonathan

Explanation:

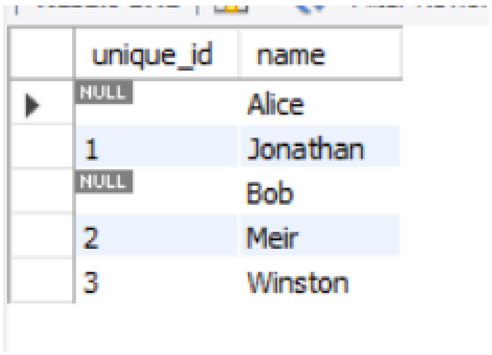
Alice and Bob do not have a unique ID, We will show null instead.

The unique ID of Meir is 2.

The unique ID of Winston is 3.

The unique ID of Jonathan is 1.

```
SELECT
 eu.unique_id,
    e.name
FROM
    Employees e
LEFT JOIN
    EmployeeUNI eu
ON
    e.id = eu.id;
```



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

Q33.

Table: Users

Column Name	Type
id	int
name	varchar

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

Column Name	Type
id	int
user_id	int
distance	int

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.
The query result format is in the following example.

Input:
Users table:

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

Rides table:

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

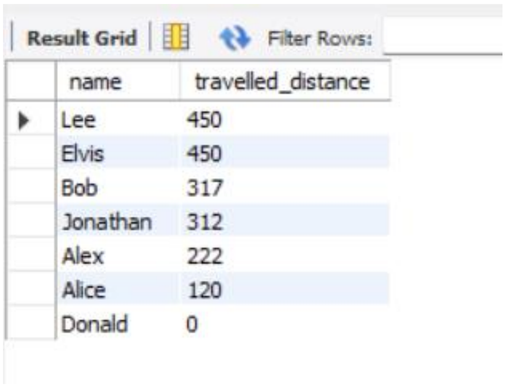
Output:

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

Explanation:
Elvis and Lee travelled 450 miles, Elvis is the top traveller as his name is alphabetically smaller than Lee.
Bob, Jonathan, Alex, and Alice have only one ride and we just order them by the total distances of the ride.
Donald did not have any rides, the distance travelled by him is 0.

```
SELECT
u.name,
COALESCE(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	travelled_distance
Lee	450
Elvis	450
Bob	317
Jonathan	312
Alex	222
Alice	120
Donald	0

Q34.

Table: Products

Column Name	Type
product_id	int
product_name	varchar
product_category	varchar

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

Table: Orders

Column Name	Type
product_id	int
order_date	date

unit	int

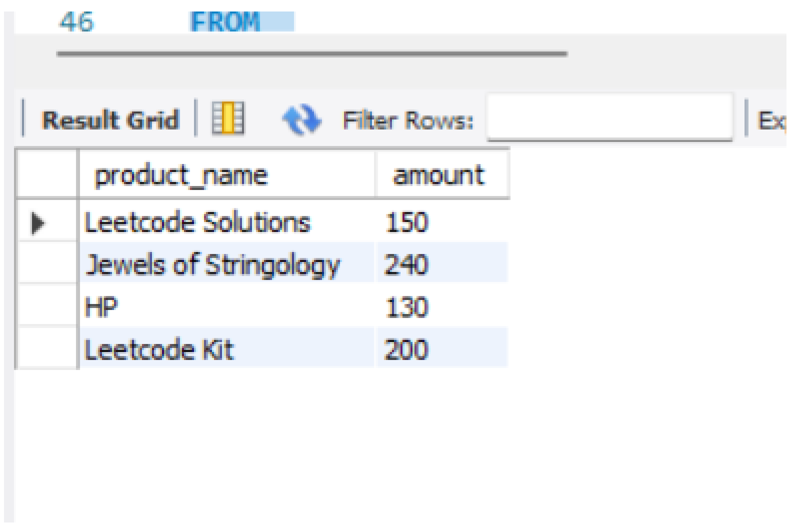
There is no primary key for this table. It may have duplicate rows.
product_id is a foreign key to the Products table. unit is
the number of products ordered in order_date.

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.
The query result format is in the following example.

Input:
Products table:

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

```
SELECT
   p.product_name,
   SUM(o.unit) AS amount
FROM
   Products p
JOIN
   Orders o ON p.product_id = o.product_id
WHERE
   o.order_date BETWEEN '2020-02-01' AND '2020-02-29'
GROUP BY
   p.product_name
HAVING
   SUM(o.unit) >= 100;
```



```
46        FROM
```

Result Grid | Filter Rows: | Ex

product_name	amount
Leetcode Solutions	150
Jewels of Stringology	240
HP	130
Leetcode Kit	200

Q35.

Table: Movies

Column Name	Type
movie_id	int
title	varchar

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

Column Name	Type
user_id	int
name	varchar

user_id is the primary key for this table.

Table: MovieRating

Column Name	Type
movie_id	int
user_id	int
rating	int
created_at	date

(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.

The query result format is in the following example.

Input:

Movies table:

movie_id	title
1	Avengers
2	Frozen 2
3	Joker

Users table:

user_id	name
1	Daniel
2	Monica
3	Maria
4	James

MovieRating table:

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

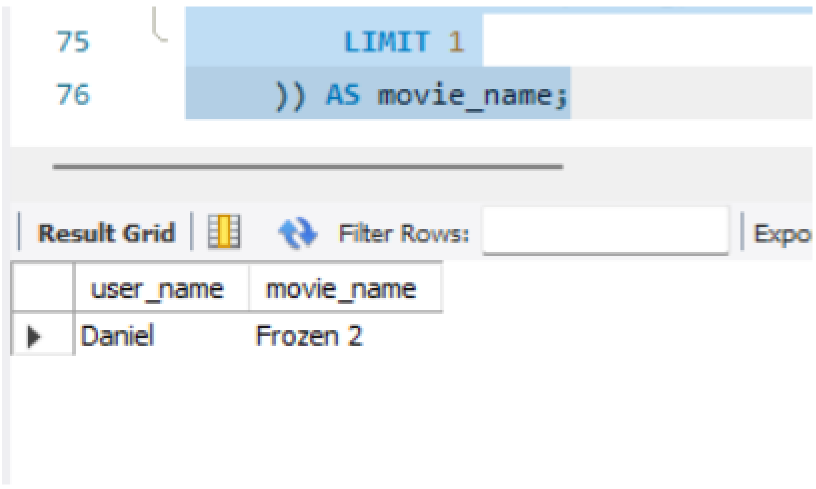
3	2	4	2020-02-25

Output:

results
Daniel
Frozen 2

Explanation:
Daniel and Monica have rated 3 movies ("Avengers", "Frozen 2" and "Joker") but Daniel is smaller lexicographically.
Frozen 2 and Joker have a rating average of 3.5 in February but Frozen 2 is smaller lexicographically.

```
SELECT
   (SELECT name
    FROM Users
    WHERE user_id = (
       SELECT user_id
       FROM MovieRating
       GROUP BY user_id
       ORDER BY COUNT(DISTINCT movie_id) DESC, name ASC
       LIMIT 1
    )) AS user_name,
   (SELECT title
    FROM Movies
    WHERE movie_id = (
       SELECT movie_id
       FROM MovieRating
       WHERE created_at BETWEEN '2020-02-01' AND '2020-02-29'
       GROUP BY movie_id
       ORDER BY AVG(rating) DESC, title ASC
       LIMIT 1
    )) AS movie_name;
```



```
            LIMIT 1
        )) AS movie_name;
```

	user_name	movie_name
▶	Daniel	Frozen 2

Q36.

Table: Users

Column Name	Type
id	int
name	varchar

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

Column Name	Type
id	int
user_id	int
distance	int

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.
The query result format is in the following example.

Input:
Users table:

id	name
1	Alice
2	Bob
3	Alex
4	Donald

7	Lee
13	Jonathan
19	Elvis

Rides table:

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

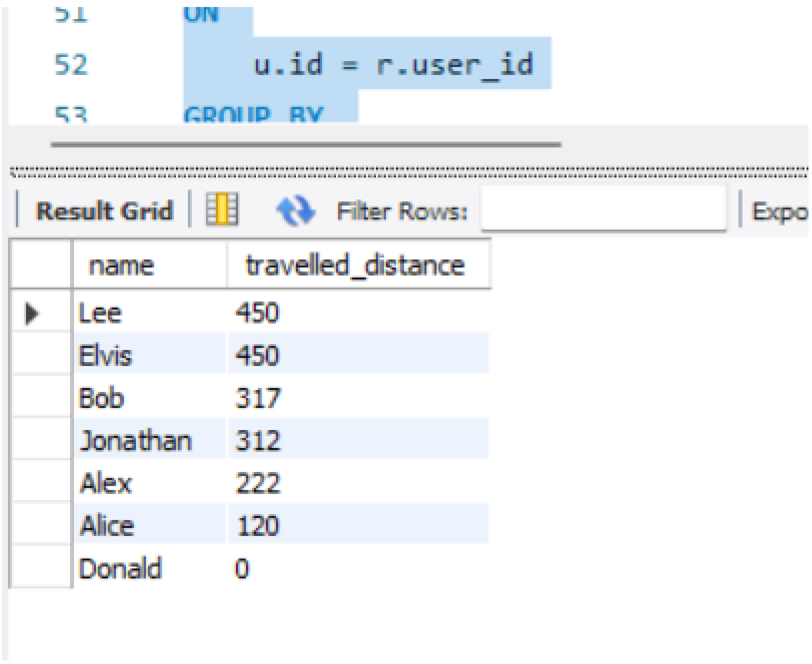
Output:

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

Explanation:
Elvis and Lee travelled 450 miles, Elvis is the top traveller as his name is alphabetically smaller than Lee.
Bob, Jonathan, Alex, and Alice have only one ride and we just order them by the total distances of the ride.
Donald did not have any rides, the distance travelled by him is 0.

```
SELECT
   u.name,
   COALESCE(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;
```



```
ON
    u.id = r.user_id
GROUP BY
```

Result Grid | Filter Rows: | Expo

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

Q37.

Table: Employees

Column Name	Type
id	int
name	varchar

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: EmployeeUNI

Column Name	Type
id	int
unique_id	int

(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.
The query result format is in the following example.

Input:
Employees table:

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

EmployeeUNI table:

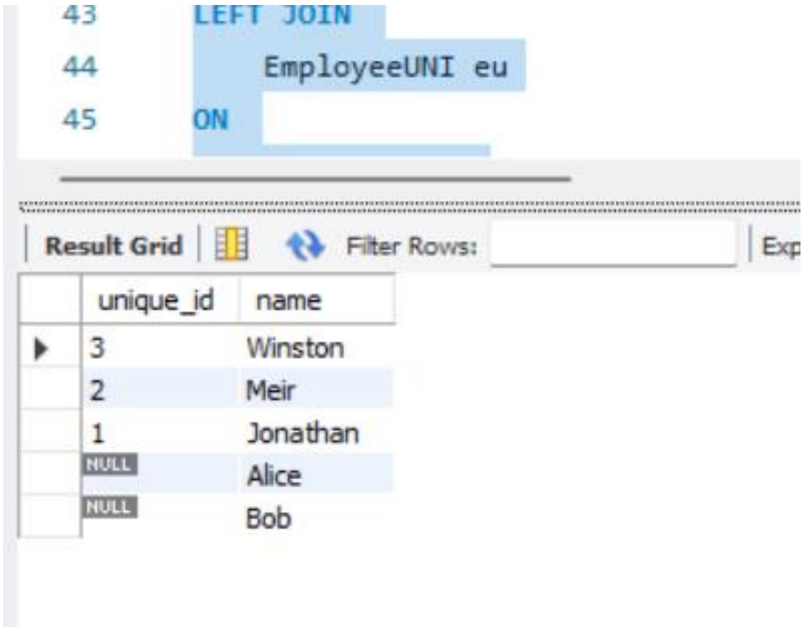
id	unique_id
3	1
11	2
90	3

Output:

unique_id	name
null	Alice
null	Bob
2	Meir
3	Winston
1	Jonathan

Explanation:
Alice and Bob do not have a unique ID, We will show null instead.
The unique ID of Meir is 2.
The unique ID of Winston is 3.
The unique ID of Jonathan is 1.

```
SELECT
    eu.unique_id,
    e.name
FROM
    Employees e
LEFT JOIN
    EmployeeUNI eu
ON
    e.id = eu.id
ORDER BY
    unique_id DESC, name;
```



```
LEFT JOIN
    EmployeeUNI eu
ON
```

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

Q38.

Table: Departments

Column Name	Type
id	int
name	varchar

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

Column Name	Type
id	int
name	varchar
department_id	int

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.
The query result format is in the following example.

Input:
Departments table:

id	name
1	Electrical Engineering
7	Computer Engineering
13	Business Administration

Students table:

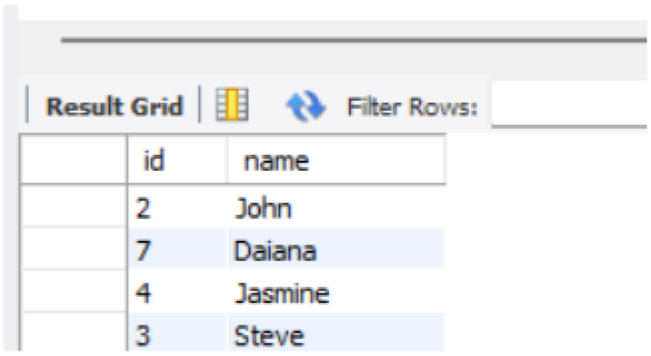
id	name	department_id
23	Alice	1
1	Bob	7
5	Jennifer	13
2	John	14
4	Jasmine	77
3	Steve	74
6	Luis	1
8	Jonathan	7
7	Daiana	33
11	Madelynn	1

Output:

id	name
2	John
7	Daiana
4	Jasmine
3	Steve

Explanation:
John, Daiana, Steve, and Jasmine are enrolled in departments 14, 33, 74, and 77 respectively. Department 14, 33, 74, and 77 do not exist in the Departments table.

```
SELECT
   s.id,
   s.name
FROM
   Students s
LEFT JOIN
   Departments d ON s.department_id = d.id
WHERE
   d.id IS NULL;
```



id	name
2	John
7	Daiana
4	Jasmine
3	Steve

Q39.

Table: Calls

Column Name	Type
from_id	int
to_id	int
duration	int

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.
The query result format is in the following example.

Input:
Calls table:

from_id	to_id	duration
1	2	59
2	1	11
1	3	20

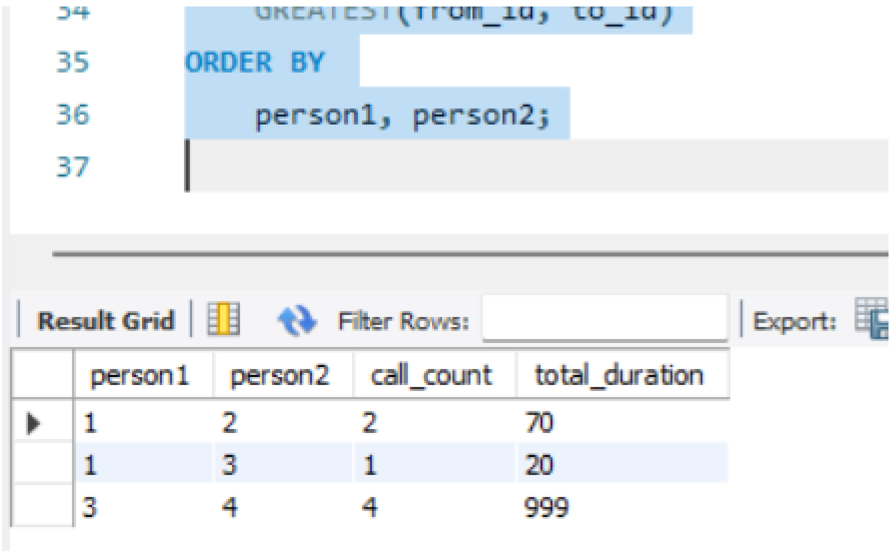
3	4	100
3	4	200
3	4	200
4	3	499

Output:

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

Explanation:
Users 1 and 2 had 2 calls and the total duration is 70 (59 + 11).
Users 1 and 3 had 1 call and the total duration is 20.
Users 3 and 4 had 4 calls and the total duration is 999 (100 + 200 + 200 + 499).

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



```
    GREATEST(from_id, to_id)
ORDER BY
    person1, person2;
```

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

Q40.

Table: Prices

Column Name	Type
product_id	int
start_date	date
end_date	date
price	int

(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: UnitsSold

Column Name	Type
product_id	int
purchase_date	date
units	int

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.
The query result format is in the following example.

Input:
Prices table:

product_id	start_date	end_date	price
1	2019-02-17	2019-02-28	5
1	2019-03-01	2019-03-22	20
2	2019-02-01	2019-02-20	15
2	2019-02-21	2019-03-31	30

UnitsSold table:

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

Output:

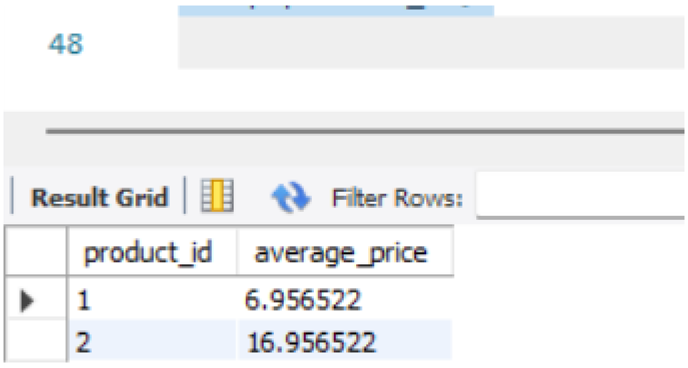
product_id	average_price
1	6.96
2	16.96

Explanation:
Average selling price = Total Price of Product / Number of products sold.
Average selling price for product 1 = ((100 * 5) + (15 * 20)) / 115 = 6.96 Average

selling price for product 2 = ((200 * 15) + (30 * 30)) / 230 = 16.96

```
SELECT
    p.product_id,
    SUM(u.units * p.price) / SUM(u.units) AS average_price
FROM
    Prices p
JOIN
    UnitsSold u ON p.product_id = u.product_id
WHERE
    u.purchase_date BETWEEN p.start_date AND p.end_date
GROUP BY
```

```
p.product_id;
```


48

```
product_id  average_price
1           6.956522
2           16.956522
```

Q41.

Table: Warehouse

Column Name	Type
name	varchar
product_id	int
units	int

(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: Products

Column Name	Type
product_id	int
product_name	varchar
Width	int
Length	int
Height	int

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 feets 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.
The query result format is in the following example.
Input:
Warehouse table:

name	product_id	units
LCHouse1	1	1

LCHouse1	2	10
LCHouse1	3	5
LCHouse2	1	2
LCHouse2	2	2
LCHouse3	4	1

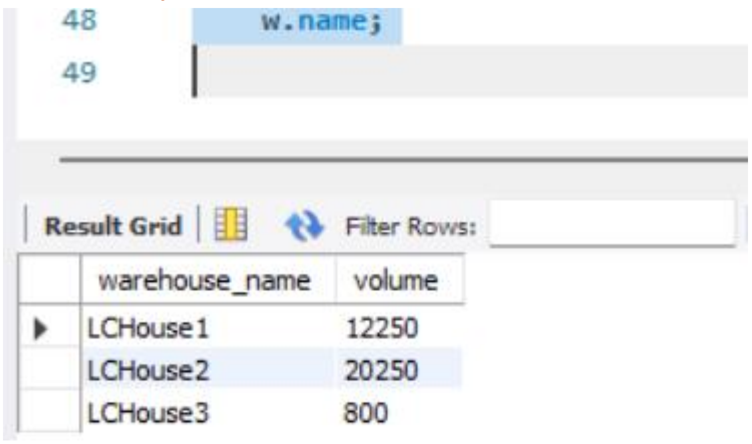
Products table:

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

Output:

warehouse_name	volume
LCHouse1	12250
LCHouse2	20250
LCHouse3	800

```
SELECT
   w.name AS warehouse_name,
   SUM(w.units * (p.Width * p.Length * p.Height)) AS volume
FROM
   Warehouse w
JOIN
   Products p ON w.product_id = p.product_id
GROUP BY
   w.name;
```



```
        w.name;
```

```
warehouse_name  volume
LCHouse1        12250
LCHouse2        20250
LCHouse3        800
```

Q42.
Table: Sales

Column Name	Type
sale_date	date
fruit	enum
sold_num	int

(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.
The query result format is in the following example.

Input:
Sales table:

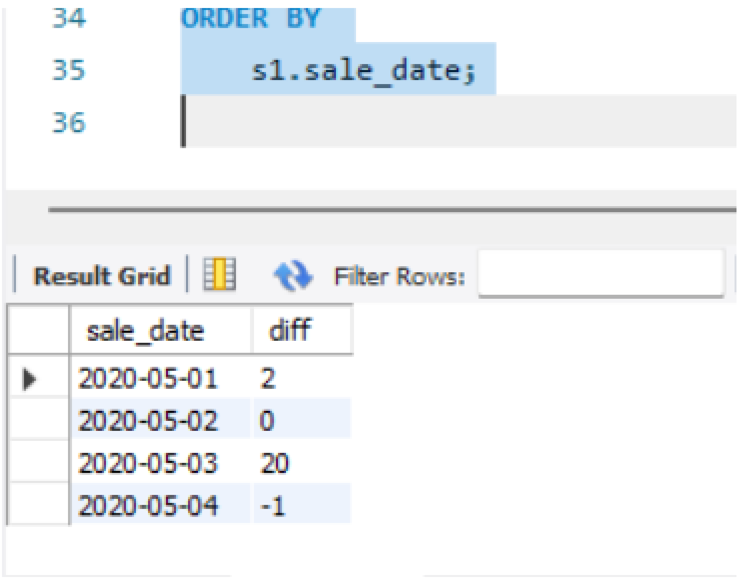
sale_date	fruit	sold_num
2020-05-01	apples	10
2020-05-01	oranges	8
2020-05-02	apples	15
2020-05-02	oranges	15
2020-05-03	apples	20
2020-05-03	oranges	0
2020-05-04	apples	15
2020-05-04	oranges	16

Output:

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

Explanation:
Day 2020-05-01, 10 apples and 8 oranges were sold (Difference 10 - 8 = 2).
Day 2020-05-02, 15 apples and 15 oranges were sold (Difference 15 - 15 = 0).
Day 2020-05-03, 20 apples and 0 oranges were sold (Difference 20 - 0 = 20).
Day 2020-05-04, 15 apples and 16 oranges were sold (Difference 15 - 16 = -1).

```
SELECT
   s1.sale_date,
   (SUM(CASE WHEN s1.fruit = 'apples' THEN s1.sold_num ELSE 0 END) -
    SUM(CASE WHEN s1.fruit = 'oranges' THEN s1.sold_num ELSE 0 END)) AS diff
FROM
   Sales s1
GROUP BY
   s1.sale_date
ORDER BY
   s1.sale_date;
```



```
ORDER BY
    s1.sale_date;
```

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

Q43.

Table: Activity

Column Name	Type
player_id	int
device_id	int
event_date	date
games_played	int

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

Write an SQL query to report the fraction of players that logged in again on the day after the day they first logged in, rounded to 2 decimal places. In other words, you need to count the number of players that logged in for at least two consecutive days starting from their first login date, then divide that number by the total number of players.

The query result format is in the following example.

Input:
Activity table:

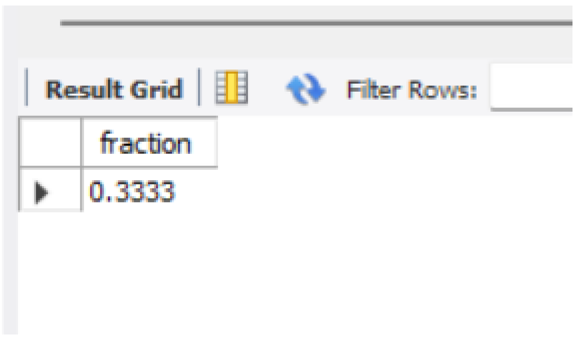
player_id	device_id	event_date	games_played
1	2	2016-03-01	5
1	2	2016-03-02	6
2	3	2017-06-25	1
3	1	2016-03-02	0
3	4	2018-07-03	5

Output:

fraction
0.33

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

```
SELECT
   SUM(CASE WHEN games_played = 0 THEN 1 ELSE 0 END) / COUNT(*) AS
fraction
FROM
   Activity;
```



```
fraction
0.3333
```

Q44.

Table: Employee

Column Name	Type
id	int
name	varchar
department	varchar
managerId	int

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 their manager.
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.
The query result format is in the following example.

Input:
Employee table:

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

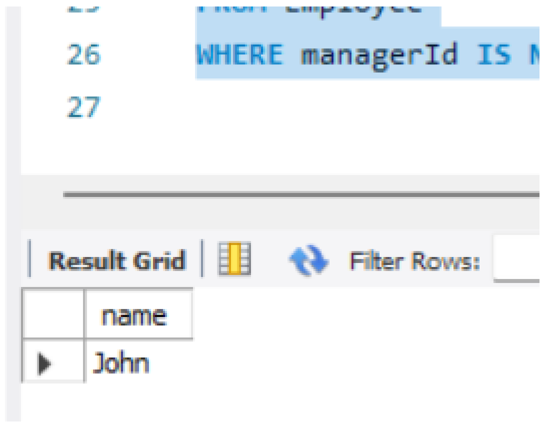
Output:

name
John

```
SELECT name

FROM Employee

WHERE managerId IS NULL;
```



```
WHERE managerId IS N
```

	name
▶	John

Q45.

Table: Student

Column Name	Type
student_id	int
student_name	varchar
gender	varchar
dept_id	int

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: Department

Column Name	Type
dept_id	int
dept_name	varchar

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.
The query result format is in the following example.

Input:
Student table:

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

3	Mark	M	2

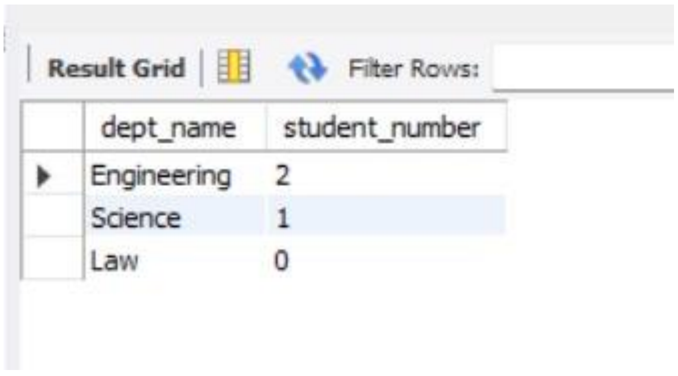
Department table:

dept_id	dept_name
1	Engineering
2	Science
3	Law

Output:

dept_name	student_numbe r
Engineering	2
Science	1
Law	0

```
SELECT
    d.dept_name,
    COUNT(s.student_id) 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,
    d.dept_name ASC;
```



```
dept_name    student_number
Engineering  2
Science      1
Law          0
```

Q46.

Table: Customer

Column Name	Type

customer_id	int
product_key	int

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

Column Name	Type
product_key	int

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.
The query result format is in the following example.

Input:
Customer table:

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

Product table:

product_key
5
6

Output:

customer_id
1
3

Explanation:
The customers who bought all the products (5 and 6) are customers with IDs 1 and 3.

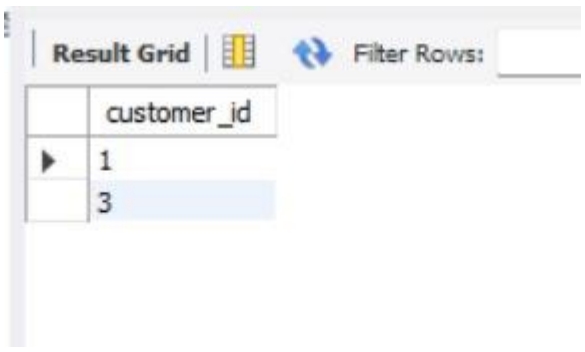
```
SELECT customer_id

FROM Customer

GROUP BY customer_id
```

```
HAVING COUNT(DISTINCT product_key) = (SELECT

COUNT(*) FROM Product);
```



	customer_id
▶	1
	3

Q47.

Table: Project

Column Name	Type
project_id	int
employee_id	int

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

Table: Employee

Column Name	Type
employee_id	int
name	varchar
experience_years	int

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

Write an SQL query that reports the 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.
The query result format is in the following example.

Input:
Project table:

project_id	employee_id

1	1
1	2
1	3
2	1
2	4

Employee table:

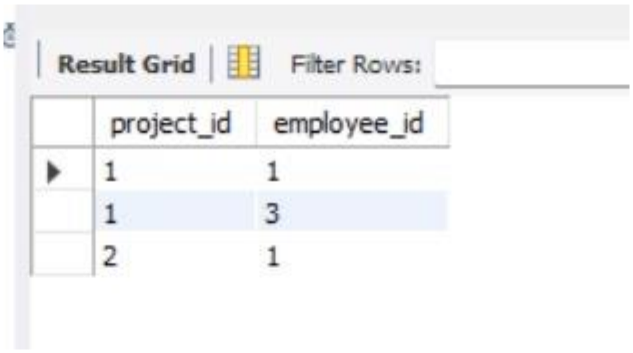
employee_id	name	experience_years
1	Khaled	3
2	Ali	2
3	John	3
4	Doe	2

Output:

project_id	employee_id
1	1
1	3
2	1

Explanation:
Both employees with id 1 and 3 have the most experience among the employees of the first project. For the second project, the employee with id 1 has the most experience.

```
SELECT
        p.project_id,
        p.employee_id,
        e.experience_years
    FROM
        Project p
    JOIN
        Employee e
    ON
        p.employee_id = e.employee_id
),
MaxExperiencePerProject AS (
    SELECT
        project_id,
        MAX(experience_years) AS max_experience
    FROM
        ProjectEmployeeExperience
    GROUP BY
        project_id
)
SELECT
    p.project_id,
    p.employee_id
FROM
    ProjectEmployeeExperience p
JOIN
    MaxExperiencePerProject m
ON
    p.project_id = m.project_id AND p.experience_years = m.max_experience
ORDER BY
    project_id, employee_id;
```



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

Q48.

Table: Books

Column Name	Type
book_id	int
name	varchar
available_from	date

book_id is the primary key of this table. Table: Orders

Column Name	Type
order_id	int
book_id	int

quantity	int
dispatch_date	date

order_id is the primary key of this table. book_id is a foreign key to the Books table.

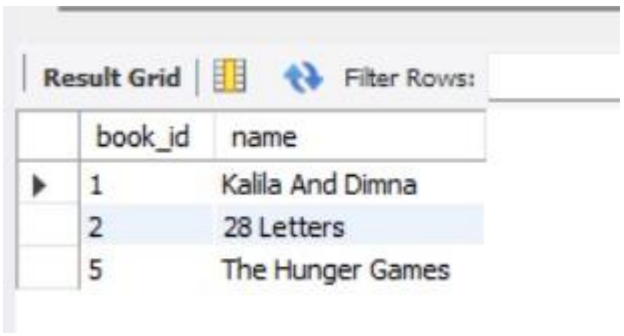
Write an SQL query that reports the books that have sold less than 10 copies in the last year, excluding books that have been available for less than one month from today. Assume today is 2019-06-23.
Return the result table in any order.
The query result format is in the following example.

Input:
Books table:

book_id	name	available_from
1	"Kalila And Demna"	2010-01-01
2	"28 Letters"	2012-05-12
3	"The Hobbit"	2019-06-10
4	"13 Reasons Why"	2019-06-01
5	"The Hunger Games"	2008-09-21

```
SELECT
   b.book_id,
   b.name
FROM
   Books b
JOIN
   Orders o ON b.book_id = o.book_id
WHERE
   o.dispatch_date BETWEEN DATE_SUB('2019-06-23', INTERVAL 1 YEAR) AND
'2019-06-23'
GROUP BY
   b.book_id, b.name, b.available_from
HAVING
   SUM(o.quantity) < 10
   AND b.available_from <= DATE_SUB('2019-06-23', INTERVAL 1 MONTH)
ORDER BY
```

```
b.book_id;
```



book_id	name
1	Kalila And Dimna
2	28 Letters
5	The Hunger Games

Q49.

Table: Enrollments

Column Name	Type
student_id	int
course_id	int
grade	int

(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.

Input:
Enrollments table:

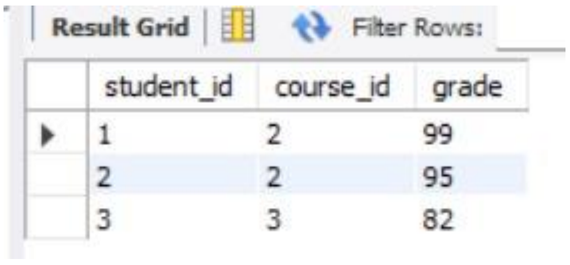
student_id	course_id	grade
2	2	95
2	3	95
1	1	90
1	2	99
3	1	80
3	2	75
3	3	82

Output:

student_id	course_id	grade
1	2	99
2	2	95

3	3	82

```
SELECT
   student_id,
   course_id,
   grade
FROM (
   SELECT
      student_id,
      course_id,
      grade,
      RANK() OVER (PARTITION BY student_id ORDER BY grade DESC, course_id
ASC) AS rank_col
   FROM Enrollments
) ranked
WHERE rank_col = 1
ORDER BY student_id;
```



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

Q50.

Table: Teams

Column Name	Type
team_id	int
team_name	varchar

team_id is the primary key of this table.
Each row of this table represents a single football team. Table:
Matches

Column Name	Type
match_id	int
host_team	int

guest_team	int
host_goals	int
guest_goals	int

match_id is the primary key of this table.
Each row is a record of a finished match between two different teams.
Teams host_team and guest_team are represented by their IDs in the Teams table (team_id), and they scored host_goals and guest_goals goals, respectively.

The winner in each group is the player who scored the maximum total points within the group. In the case of a tie, the lowest player_id wins.
Write an SQL query to find the winner in each group.
Return the result table in any order.
The query result format is in the following example.

Input:
Players table:

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

Matches table:

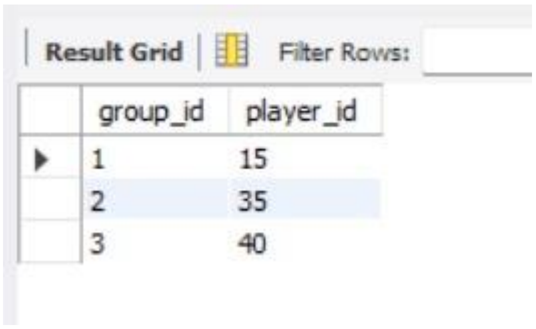
match_id	first_player	second_player	first_score	second_score
1	15	45	3	0
2	30	25	1	2
3	30	15	2	0
4	40	20	5	2
5	35	50	1	1

Output:

group_id	player_id

1	15
2	35
3	40

```
SELECT
        p.project_id,
        p.employee_id,
        e.experience_years
    FROM
        Project p
    JOIN
        Employee e
    ON
        p.employee_id = e.employee_id
),
MaxExperiencePerProject AS (
    SELECT
        project_id,
        MAX(experience_years) AS max_experience
    FROM
        ProjectEmployeeExperience
    GROUP BY
        project_id
)
SELECT
    p.project_id,
    p.employee_id
FROM
    ProjectEmployeeExperience p
JOIN
    MaxExperiencePerProject m
ON
    p.project_id = m.project_id AND p.experience_years = m.max_experience
ORDER BY
    project_id, employee_id;
```



group_id	player_id
1	15
2	35
3	40