# SQL Questions and Answers Day -2

**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 |

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

```
92 •  SELECT * FROM city WHERE countrycode = 'USA' AND population > 100000;
93
94
```

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

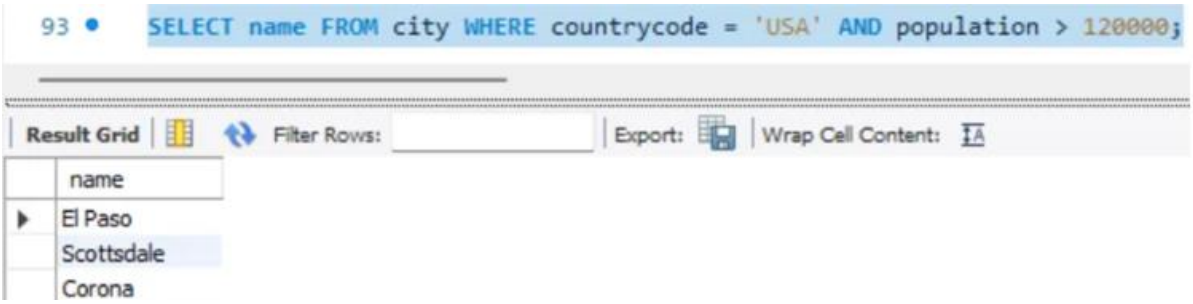
| id | name | countrycode | district | population |
|---|---|---|---|---|
| 3815 | El Paso | USA | Texas | 563662 |
| 3878 | Scottsdale | USA | Arizona | 202705 |
| 3965 | Corona | USA | California | 124966 |
| 3973 | Concord | USA | California | 121780 |
| 3977 | Cedar Rapids | USA | Iowa | 120758 |
| 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 |

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




**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 |

Ans) SELECT * FROM city;

```
94 •  SELECT *  FROM city;
95
```

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

| 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 |

**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 |

Ans) SELECT * FROM city WHERE id = 1661;

```
95 •  SELECT * FROM city WHERE id = 1661;
96
97
```

Result Grid | Filter Rows: | Export:

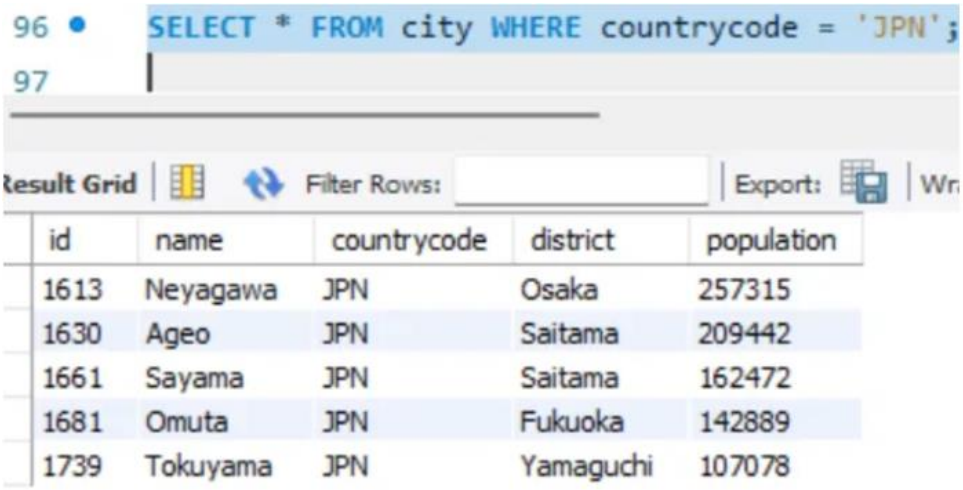
| id | name | countrycode | district | population |
|---|---|---|---|---|
| 1661 | Sayama | JPN | Saitama | 162472 |

**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 |

Ans) 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 |

Ans) SELECT * FROM city WHERE countrycode = 'JPN';

```
96 •  SELECT * FROM city WHERE countrycode = 'JPN';
97
```

Result Grid | Filter Rows: | Export: | Wrap

| 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 |

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

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

**STATION**

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

Ans) SELECT city, state FROM station;

```
135 •   SELECT city, state FROM station;
136
```

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:

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

**STATION**

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

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

```
136 •   SELECT DISTINCT city FROM station WHERE id % 2 = 0;
137
```

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

| 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.

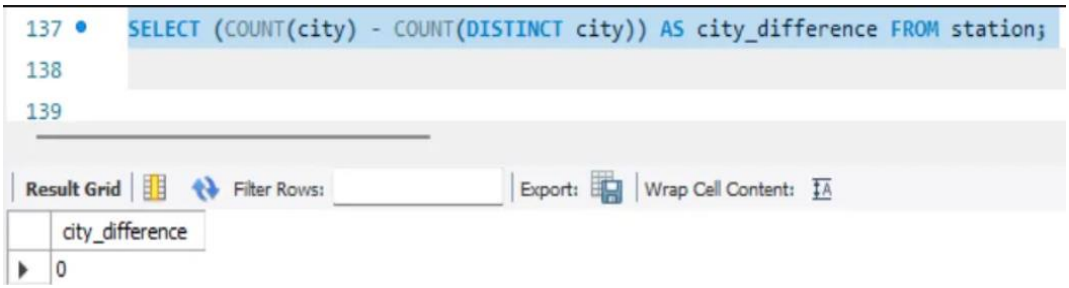
The STATION table is described as follows:
where LAT_N is the northern latitude and LONG_W is the western longitude.

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

**STATION**

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

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



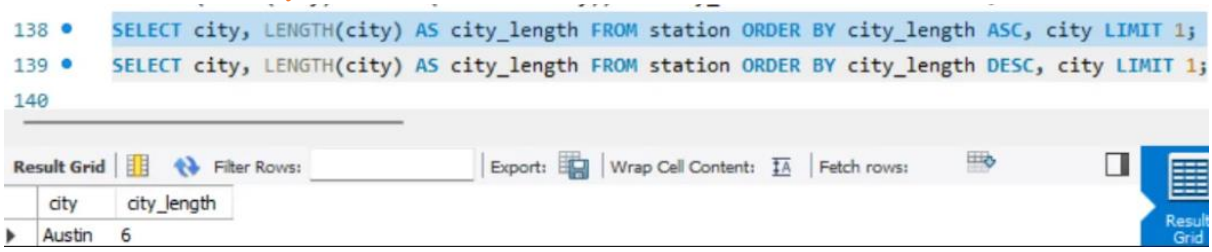

**Q10)** Query the two cities in STATION with the shortest and longest CITY names, as well as their respective lengths (i.e.: number of characters in the name). If there is more than one smallest or largest city, choose the one that comes first when ordered alphabetically. The STATION table is described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

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

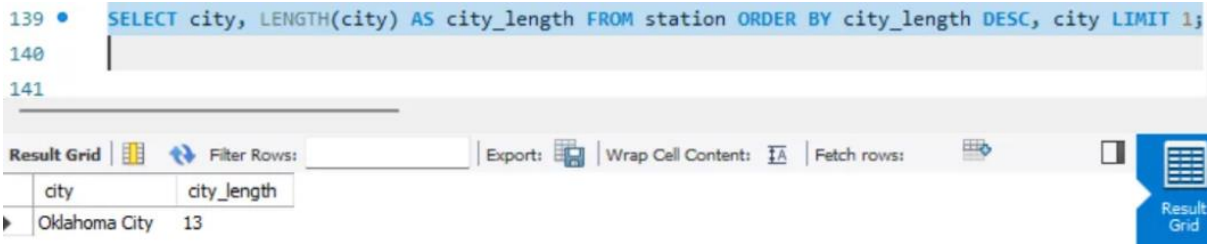
**STATION**

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

Ans) 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 DESC, city LIMIT 1; -- longest city




**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

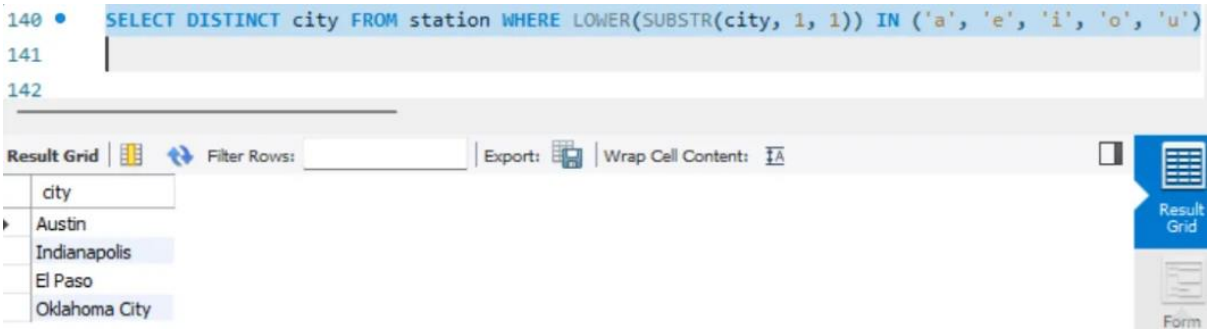
The STATION table is described as follows:

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

**STATION**

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

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




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

Input Format

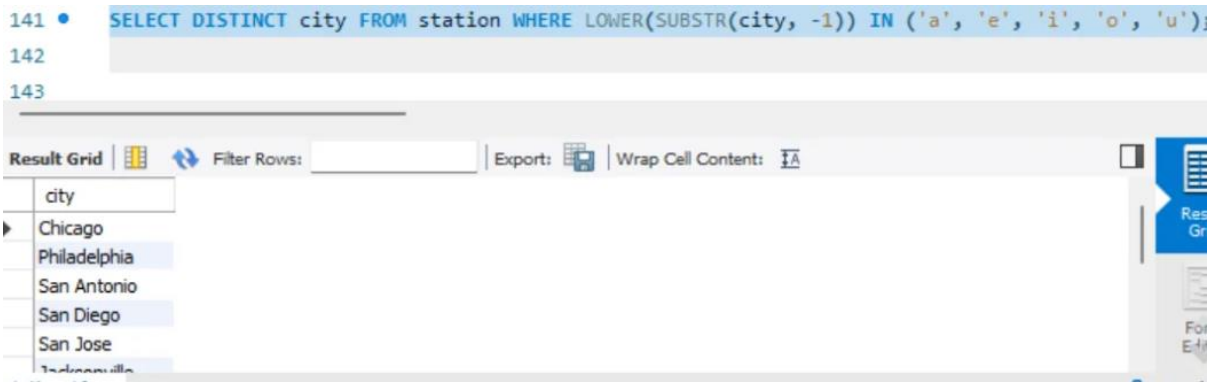
The STATION table is described as follows:

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

**STATION**

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

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

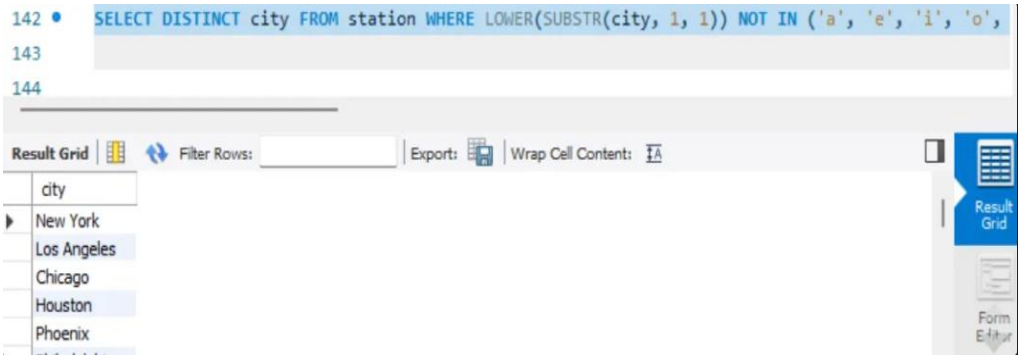



**Q13)** Query the list of CITY names from STATION that do not start with vowels. Your result cannot contain duplicates.
Input Format
The STATION table is described as follows:
where LAT_N is the northern latitude and LONG_W is the western longitude.

**STATION**

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

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

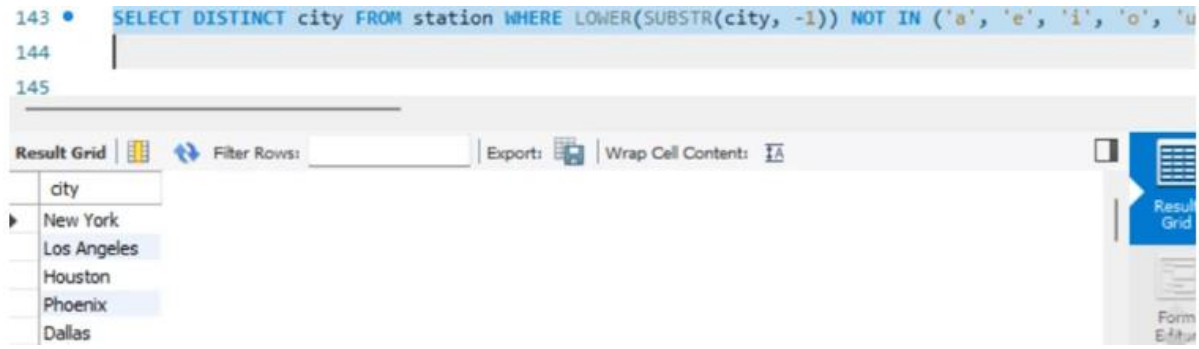



**Q14)** Query the list of CITY names from STATION that do not end with vowels. Your result cannot contain duplicates.
Input Format
The STATION table is described as follows:
where LAT_N is the northern latitude and LONG_W is the western longitude.

**STATION**

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

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



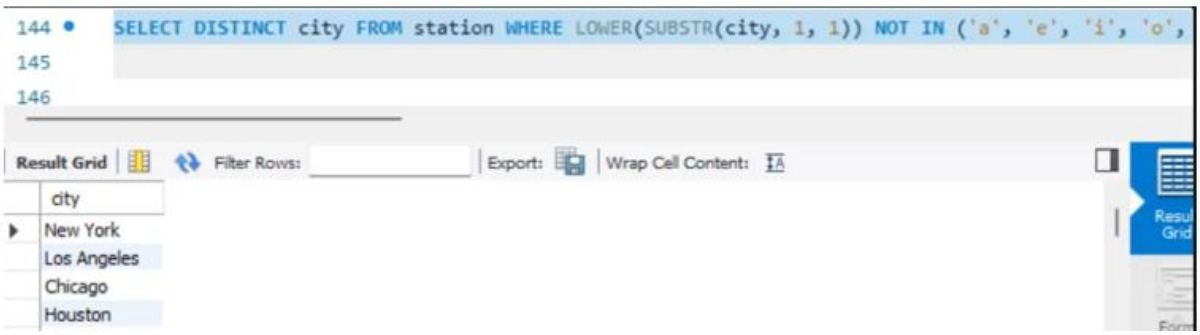
**Q15)** Query the list of CITY names from STATION that either do not start with vowels or do not end with vowels. Your result cannot contain duplicates.
Input Format
The STATION table is described as follows:

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

**STATION**

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

Ans) 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');



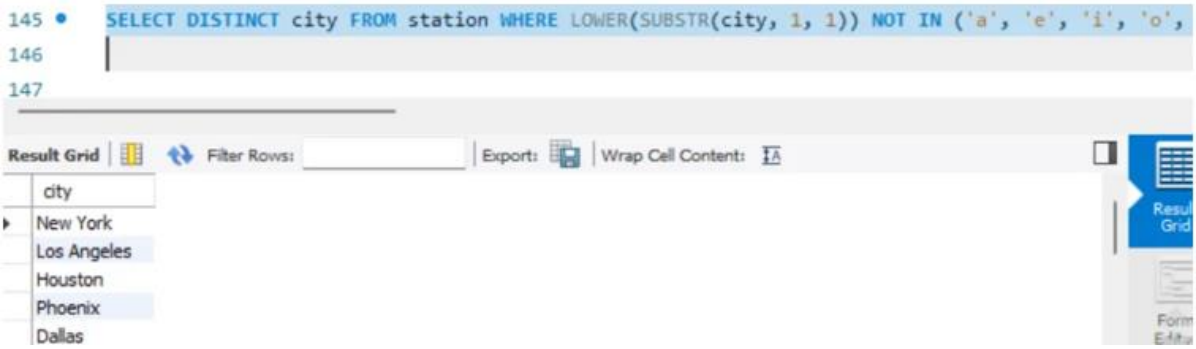

**Q16)** Query the list of CITY names from STATION that do not start with vowels and do not end with vowels. Your result cannot contain duplicates.
Input Format
The STATION table is described as follows:

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

**STATION**

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

Ans) 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');




**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 |

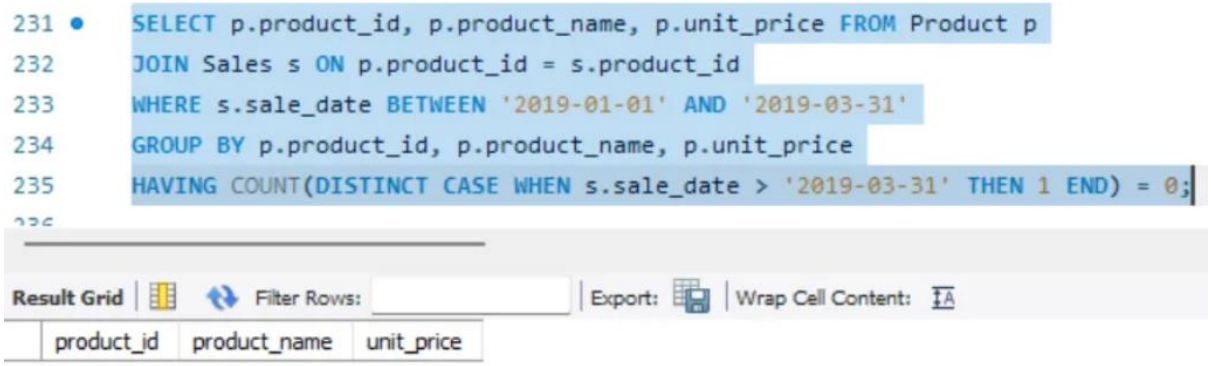
Output:

| 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.

Ans) 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;
```




**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:

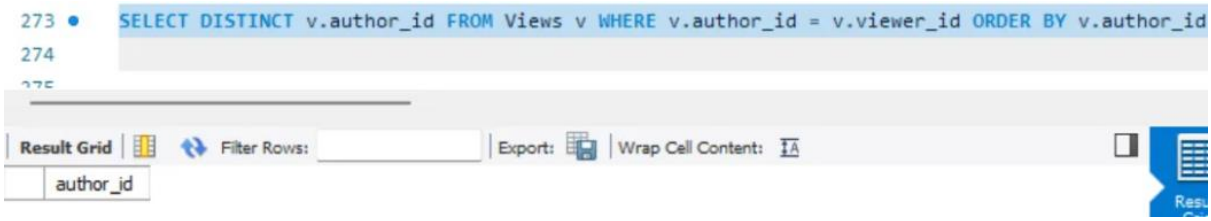
| article_id | author_id | viewer_id | view_date |
|---|---|---|---|
| 1 | 3 | 5 | 2019-08-01 |
| 1 | 3 | 6 | 2019-08-02 |
| 2 | 7 | 7 | 2019-08-01 |
| 2 | 7 | 6 | 2019-08-02 |
| 4 | 7 | 1 | 2019-07-22 |

| | | | |
|---|---|---|---|
| 3 | 4 | 4 | 2019-07-21 |
| 3 | 4 | 4 | 2019-07-21 |

Output:

| id |
|---|
| 4 |
| 7 |

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




**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 |

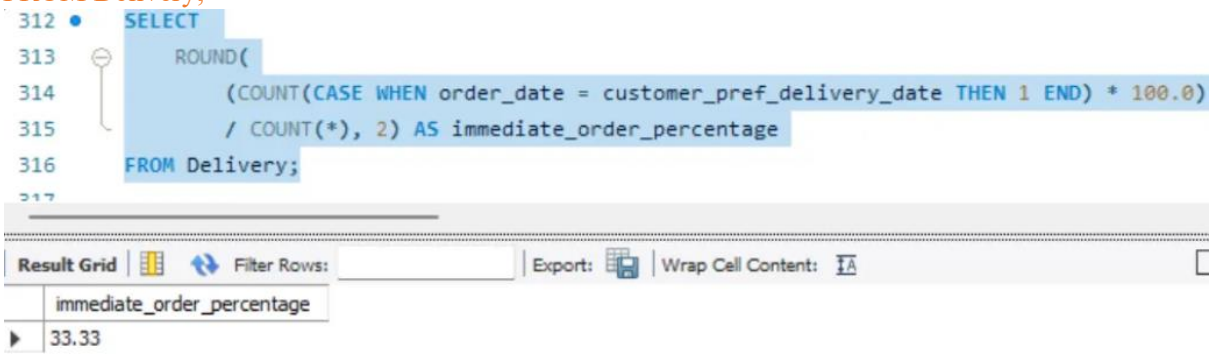
| | | | |
|---|---|---|---|
| 2 | 5 | 2019-08-02 | 2019-08-02 |
| 3 | 1 | 2019-08-11 | 2019-08-11 |
| 4 | 3 | 2019-08-24 | 2019-08-26 |
| 5 | 4 | 2019-08-21 | 2019-08-22 |
| 6 | 2 | 2019-08-11 | 2019-08-13 |

Output:

| immediate_percentage |
|---|
| 33.33 |

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

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




**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,
```

Ans) 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;

```
355 •  SELECT ad_id,
356        ROUND(SUM(CASE WHEN action = 'Clicked' THEN 1 ELSE 0 END) * 100.0 /
357              SUM(CASE WHEN action IN ('Clicked', 'Viewed') THEN 1 ELSE 0 END), 2) AS ctr
358    FROM Ads
359    GROUP BY ad_id
360    ORDER BY ctr DESC, ad_id ASC;
361
362
```

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

| 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 |

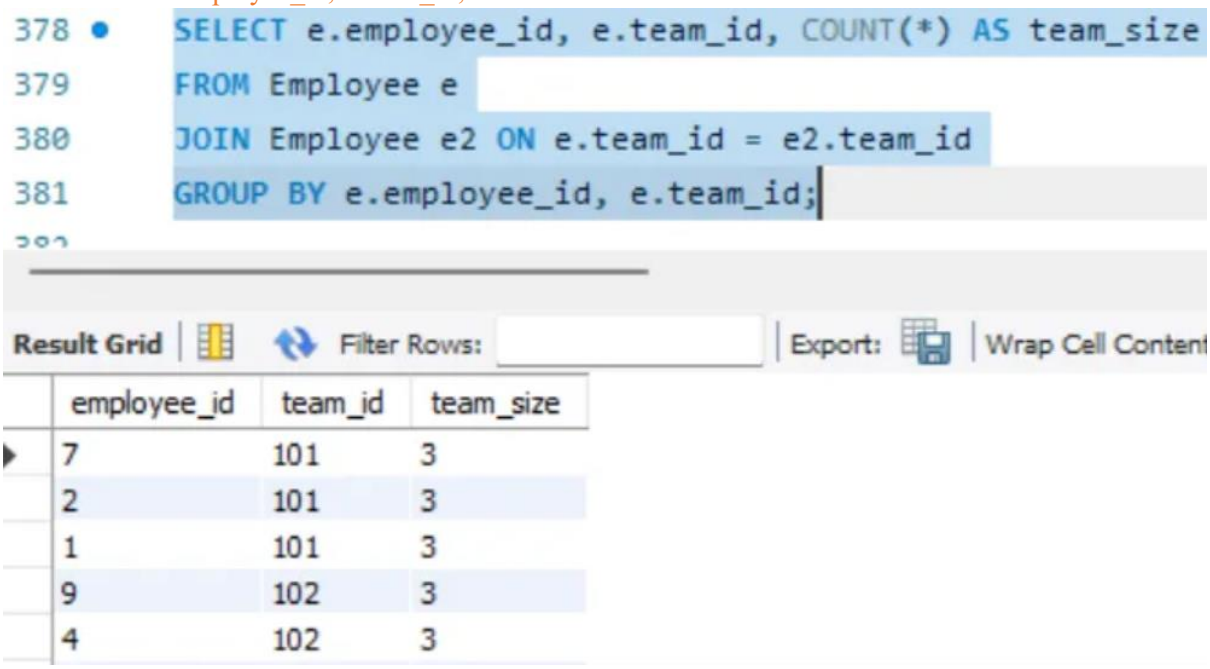
| | |
|---|---|
| 5 | 9 |
| 6 | 9 |

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.

Ans)
```
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;
```




**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.

Ans)
```
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;
```

```
430         CASE
431             WHEN AVG(w.weather_state) <= 15 THEN 'Cold'
432             WHEN AVG(w.weather_state) >= 25 THEN 'Hot'
433             ELSE 'Warm'
434         END AS weather_type
435     FROM Countries c
436     JOIN Weather w ON c.country_id = w.country_id
437     WHERE w.date BETWEEN '2019-11-01' AND '2019-11-30'
438     GROUP BY c.country_name;
439
```

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

| 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

Ans)
```
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;
```

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

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

| 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:

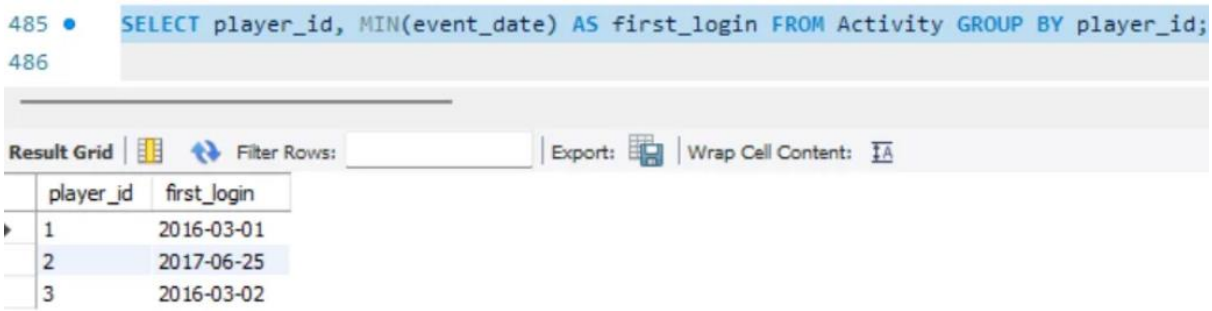
Activity table:

| 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 |

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




**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 |

Ans)
```
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;

```
501     FROM Activity1 a
502     JOIN (
503         SELECT player_id, MIN(event_date) AS first_login_date
504         FROM Activity1
505         GROUP BY player_id
506     ) first_login
507     ON a.player_id = first_login.player_id
508     AND a.event_date = first_login.first_login_date;
509
510
```

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

| player_id | device_id |
|---|---|
| 1 | 2 |
| 2 | 3 |
| 3 | 1 |