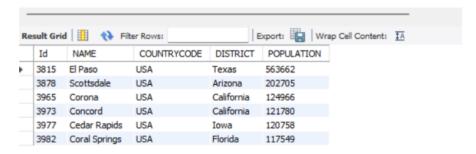
**Q1**. Query all columns for all American cities in theCITY table with populations larger than 100000. The CountryCode for America is USA.

The CITY table is described as follows:

#### Ans.:

```
1 • select * from city_table
2 where COUNTRYCODE = 'USA' and POPULATION > 100000
```

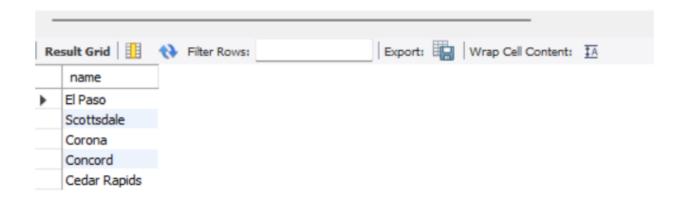


**Q2**. Query the NAME field for all American cities in the CITY table with populations larger than 120000. The CountryCode for America is USA.

The CITY table is described as follows:

### Ans.

1 • select name from city\_table
2 where COUNTRYCODE = 'USA' and POPULATION > 120000

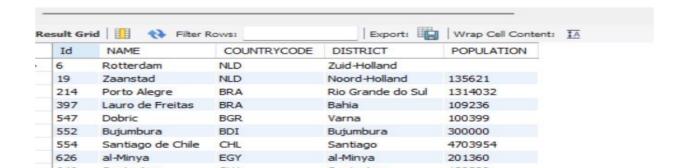


**Q3.**Query all columns (attributes) for every row in the CITY table. The CITY table is described as follows:

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

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

```
1 • select * from city_table
2
```

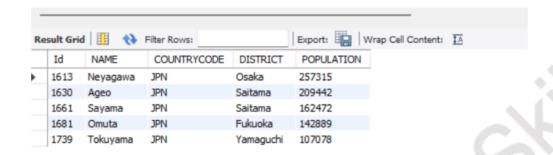


**Q5**. Query all attributes of every Japanese city in the CITY table. The COUNTRYCODE for Japan is JPN.

The CITY table is described as follows:

Ans:

1 • select \* from city\_table where countrycode = 'JPN'



**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	Туре
ID	NUMBER
NAME	VARCHAR2(17)
COUNTRYCODE	VARCHAR2(3)
DISTRICT	VARCHAR2(20)
POPULATION	NUMBER

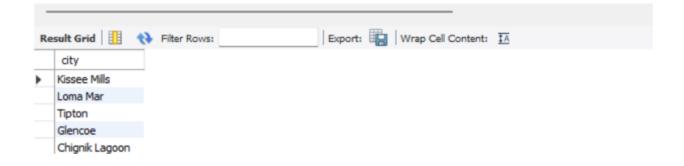
**Q7.**Query a list of CITY and STATE from the STATIONtable. The STATION table is described as follows:

```
select city,state from stationdata
```

**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:

```
1 • select distinct city from stationdata where mod(id,2) = 0
2
```



**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:

Ans:

```
select count(city) - count(distinct city) from stationdata
```

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

**Q10.**Query the two cities in STATION with the shortestand 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:

Ans:

```
select city , length(city) from stationdata
order by length(city) asc ,city asc
limit 1

select city , length(city) from stationdata
order by length(city) desc,city asc
limit 1
```

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.

**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:

Ans:

```
SELECT DISTINCT(CITY)

FROM stationdata

WHERE CITY REGEXP '^[aeiou]';
```

where LAT\_N is the northern latitude and LONG\_W is the western longitude.

**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:

```
SELECT DISTINCT(CITY)

FROM stationdata

WHERE CITY REGEXP '[aeiou]$';
```

# **STATION**

Field	Туре	
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.

**Q13.**Query the list of CITY names from STATION thatdo 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.

Ans:

```
SELECT DISTINCT(CITY)
FROM stationdata
WHERE CITY not REGEXP '^[aeiou]';
```

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

```
SELECT DISTINCT(CITY)

FROM stationdata

WHERE CITY not REGEXP '[aeiou]$';
```

#### **Input Format**

The STATION table is described as follows:

# STATION

Field	Туре
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.

**Q15.**Query the list of CITY names from STATION thateither 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:

```
SELECT DISTINCT(CITY)

FROM stationdata

WHERE CITY NOT REGEXP '^[aeiou]' OR CITY NOT REGEXP '[aeiou]$';
```

**Q16.**Query the list of CITY names from STATION thatdo not start with vowels and do not end with vowels. Your result cannot contain duplicates.

```
SELECT DISTINCT(CITY)

FROM stationdata

WHERE CITY NOT REGEXP '^[aeiou]' and CITY NOT REGEXP '[aeiou]$';
```

# **STATION**

Field	Туре
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.

1	<u> </u>	
	Column Name	Type
	Tano Medu Pt∡oid uct	int
	product_name v	archar
	unit_price	int

product\_id is the primary key of this table.

int

int

quantity

price

eshirawnathis t	able indicates the	name and the price of each product.
seller id Fable: Sales	int	
product_id	int	
buyer_id	int	
sale_date	date	

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 guery result format is in the following example.

#### Input:

#### Product table:

product_id	product_name u	nit_price
1	S8	1000
2	G4	800
3	iPhone	1400

#### Sales table:

seller_id	product_id	buyer_id	sale_date	quantity	price
1123	1223	1234	2019-01-21	2112	2000
			2019-02-17		800
			2019-06-02		800
			2019-05-13		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 DISTINCT p.product_id, p.product_name

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'

AND p.product_id not in(
    select s.product_id
    from Sales s
    where s.sale_date < '2019-01-01' or s.sale_date > '2019-03-31'
)

SELECT p.product_id, p.product_name

FROM Product p

JOIN Sales s ON p.product_id = s.product_id

GROUP BY p.product_id, p.product_name

HAVING MIN(s.sale_date) >= '2019-01-01'

AND MAX(s.sale_date) <= '2019-03-31';
```

#### Q18.

Table: Views

Column Name	Туре
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.

```
select distinct author_id as id
from Views
where author_id = viewer_id
order by author_id asc
```

## Q19.

Table: Delivery

Cyphemn Name	
idnetlivery_id	
icnutstomer_id	
<b>da</b> der_date	
customer_pref_delivery_date d	ate

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:					
			customer_pref_		
delivery_id	customer_id	order_date	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(SUM(order_date = customer_pref_delivery_date) / COUNT(1) * 100, 2) AS immediate_percentage
FROM Delivery;
```

. ......

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.

```
Q.21
select e.employee_id,(select count(team_id) from employee_team where e.team_id = team_id) as team size
from employee_team e
SELECT employee_id, COUNT(team_id) OVER (PARTITION BY team_id) team_size
FROM employee_team
022.
 SELECT c.country name,
CASE WHEN AVG(w.weather_state) <= 15 THEN 'Cold'
      WHEN AVG(w.weather state) >= 25 THEN 'Hot'
      ELSE 'Warm' E
- END AS weather type
 FROM Weather w INNER JOIN Countries c ON w.country_id = c.country_id
 WHERE w.day BETWEEN '2019-11-01' AND '2019-11-30'
 GROUP BY c.country name;
Q23.
SELECT p.product_id, IFNULL(ROUND(SUM(units*price)/SUM(units),2),0) AS
  average price
FROM Prices p LEFT JOIN UnitsSold u
1 ON p.product_id = u.product_id AND
5 u.purchase_date BETWEEN start_date AND end_date
5 group by product_id
Q24.
 select player_id,min(event_date) as first_login
 from Activity
```

group by player\_id

```
select player_id, device_id from(
 select player_id,device_id ,row_number() over(partition by player_id order by count(*) desc) as rn
 from Activity
 group by player_id, device_id
) ranked
 where rn = 1
 order by player_id
0.26
select p.product name, sum(unit) as unit
from products p left join orders o on
p.product_id = o.product_id
where o.order_date between '2020-02-01' and '2020-02-29'
group by product_name
having sum(unit) >= 100
0..27
# Write your MySQL query statement below
select * from users
WHERE mail REGEXP '^[a-zA-Z][a-zA-Z0-9_.-]*@leetcode[.]com$';
0.28
  select o.customer_id,c.name
  from orders o
  join customers c on o.customer_id = c.customer_id
  join product p on p.product_id = o.product_id
  group by customer id
having(
  sum(case when o.order_date like '2020-06%' then o.quantity*p.price else 0 end ) >= 100
  and sum(case when o.order_date like '2020-07%' then o.quantity*p.price else 0 end ) >= 100
Q29.
select distinct title
from content c join tvprogram t on c.content_id = t.content_id
where program_date like '2020-06%' and content_type = 'Movies' and Kids_content = 'Y'
```

Q30

```
SELECT Q.id,Q.year,IFNULL(npv, 0)
 FROM queries Q
 LEFT JOIN npv N ON Q.id = N.id and Q.year = N.year
032.
  # Write your MySQL query statement below
! SELECT ifnull(unique id, null) as unique id, name
from Employees e
left join EmployeeUNI eu on e.id = eu.id
Q.33
# Write your MySQL query statement below
select name,ifnull(sum(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 asc
034.
  # Write your MySQL query statement below
  select p.product name, sum(unit) as unit
  from products p left join orders o on
  p.product id = o.product id
  where o.order_date between '2020-02-01' and '2020-02-29'
  group by product_name
```

having sum(unit) >= 100

```
# write your MySQL query statement below
/select name as results from(
      select name,count(*) as counts
     from Users u join MovieRating m on u.user_id=m.user_id
     group by m.user_id
     order by counts desc, name asc
      limit 1
 ) first_query
 union all
/select title as results from(
      select title,avg(rating) as ratings
      from Movies mo join MovieRating mr on mo.movie_id=mr.movie_id
     where created_at like('2020-02%')
     group by mr.movie_id
     order by ratings desc, title asc
     limit 1
 )second_query
038.
select s.id, s.name
from students s left join departments d on d.id = s.department_id
where d.id is null
SELECT id, name
FROM Students
WHERE department id not in (SELECT id from Departments)
039.
select from id as person1, to id as person2, count(duration) as call_count, sum(duration) as total_duration
from (select * from Calls
    select to_id, from_id, duration from Calls) t1
where from_id < to_id
group by person1, person2
```

```
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 person1, person2
0.41
 select name, sum(w.units*p.Width*p.Length*p.Height)
 from warehouse w
 join products p on w.product_id = p.product_id
 group by name
Q.42
select a.sale_date,(a.sold_num-b.sold_num) as diff
from Sales a left join Sales b on a.sale date = b.sale date
where a.fruit = 'apples' and b.fruit = 'oranges'
0.43
Solve it later
Q.44
 # Write your MySQL query statement below
  select name from (select e.name,count(*)
 from Employee e join Employee m on e.id = m.managerId
 group by name
 having count(*) >=5) t
Q45.
```

```
select d.dept name, ifnull(count(student id),0) as student number
 from Department d left join student s on s.dept id = d.dept id
 group by dept name
 ORDER BY student number DESC, dept name;
Q46.
select customer id
from Customer
group by customer_id
having(count(distinct product_key) = (select count(distinct product_key) as
p_count from Product ))
047.
 select project_id,p.employee_id
 from Project p left join Employee e on p.employee_id = e.employee_id
where (project_id, experience_years) in (select project_id, max(experience_years)
                                       from Project join Employee using (employee_id)
                                       group by project_id)
048.
129 Select book id, name from Books where book id not in(select book id from orders
        where dispatch date >= '2018-06-23' and dispatch date <= '2019-06-22'
130
        group by book id
131
       having sum(quantity) >= 10 )
132
        and available from < '2019-05-23'
133
134
Result Grid
            Filter Rows:
                                       Export: Wrap Cell Content: IA
   book_id name
          Kalila And Demna
  1
  2
          28 Letters
  5
          The Hunger Games
```

```
select student_id,course_id,grade from (SELECT *, ROW_NUMBER() OVER (PARTITION BY student_id ORDER BY grade DESC,course_id asc) AS r
153
       FROM Enrollments) e
154
155
       where e.rn = 1
157
Export: Wrap Cell Content: TA
                                                                                      student_id course_id grade
      2
            95
  select student_id, min(course_id) as course_id, grade
  from Enrollments
 where (student_id, grade) in
       (select student id, max(grade)
       from Enrollments
       group by student id)
  group by student id, grade
  order by student id asc
```

#### O50.

```
WITH PlayerScores AS (
   SELECT
       p.group_id,
       CASE
           WHEN m.first_score >= m.second_score THEN m.first_player
           ELSE m.second player
       END AS player_id,
       COALESCE(m.first_score, 0) + COALESCE(m.second_score, 0) AS total_score
   JOIN Matches m ON p.player_id = m.first_player OR p.player_id = m.second_player
SELECT group_id, player_id
FROM (
   SELECT *,
          ROW_NUMBER() OVER (PARTITION BY group_id ORDER BY total_score DESC, player_id ASC) AS rn
   FROM PlayerScores
) RankedScores
WHERE rn = 1;
```

Result Grid Filter Ro		
	group_id	player_id
•	1	15
	2	35
	3	40