Sample_Dataset 1:

```
ID,NAME,COUNTRYCODE,DISTRICT,POPULATION
6,Rotterdam,NLD,Zuid-Holland,593321
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
4054,Fairfield,USA,California,92256
4058,Boulder,USA,Colorado,91238
4061,Fall River,USA,Massachusetts,90555
```

```
create database set1;

use set1;

create table city(ID int, NAME varchar(17),
COUNTRYCODE varchar(3), DISTRICT varchar(20), POPULATION int);

insert into city values
(6,'Rotterdam','NLD','Zuid-Holland',593321),
(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),
(4054,'Fairfield','USA','California',92256),
(4058,'Boulder','USA','Colorado',91238),
(4061,'Fall River','USA','Massachusetts',90555);
```

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

```
select * from city
where countrycode='USA' and population>100000;
```

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

```
select Name from city
where countrycode='USA' and population>120000;
```

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

```
select * from city;
```

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

```
select * from city
where id=1661;
```

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

```
select * from city
where countrycode='JPN';
```

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

```
SELECT name from city
where countrycode='JPN';
```

Sample_Dataset 2:

```
ID,city,state,LAT_N,LONG_W
794,Kissee Mills,MO,139,73
824,Loma Mar,CA,48,130
603,Sandy Hook,CT,72,148
478,Tipton,IN,33,97
619,Arlington,CO,75,92
711,Turner,AR,50,101
839,Slidell,LA,85,151
411,Negreet,LA,98,105
588,Glencoe,KY,46,136
665,Chelsea,IA,98,59
342,Chignik Lagoon,AK,103,153
733,Pelahatchie,MS,38,28
441,Hanna City,IL,50,136
```

```
811,Dorrance,KS,102,121
698,Albany,CA,49,80
325,Monument,KS,70,141
414,Manchester,MD,73,37
113,Prescott,IA,39,65
971,Graettinger,IA,94,150
266,Cahone,CO,116,127

create table station
(ID int, city varchar(21), state varchar(2), LAT_N int, LONG_W int);

insert into station values
(794,'Kissee Mills','MO',139,73),
(824,'Loma Mar','CA',48,130),
(603,'Sandy Hook','CT',72,148),
(478,'Tipton','IN',33,97),
(619,'Arlington','CO',75,92),
(711,'Turner','AR',50,101),
(839,'Slidell','LA',85,151),
(411,'Negreet','LA',98,105),
(588,'Glencoe','KY',46,136),
(665,'Chelsea','IA',98,59),
(342,'Chignik Lagoon','AK',103,153),
(733,'Pelahatchie','MS',38,28),
(441,'Hanna City','IL',50,136),
(811,'Dorrance','KS',102,121),
(698,'Albany','CA',49,80),
(325,'Monument','KS',70,141),
(414,'Manchester','MD',73,37),
(113,'Prescott','IA',39,65),
(971,'Graettinger','IA',94,150),
(266,'Cahone','CO',116,127);

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

select city,state from station
WHERE LAT_N between 0 and 90
and LONG_W BETWEEN 0 and 180;

8. Query a list of CITY names from STATION for cities that have an even ID number

select distinct city from station
WHERE mod(id,2) = 0 and
LAT_N between 0 and 90
and LONG_W BETWEEN 0 and 180;

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

select city,(count(city) - count(distinct city)) as No_of_duplicate_entries from station
group by city
HAVING COUNT(city)>1
LAT_N between 0 and 90
and LONG_W BETWEEN 0 and 180;

10. 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

(select city, LENGTH(city) as No_of_characters from station
where LAT_N between 0 and 90
and LONG_W BETWEEN 0 and 180
ORDER BY length(city),city LIMIT 1)
UNION
(select city, LENGTH(city) as No_of_characters from station
where LAT_N between 0 and 90
and LONG_W BETWEEN 0 and 180
ORDER BY length(city) DESC,city LIMIT 1);

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

SELECT distinct city from station
where substr(city,1,1) in ('a','e','i','o','u') and
LAT_N between 0 and 90
and LONG_W BETWEEN 0 and 180;

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

select distinct city from station
where substr(city,-1,1) in ('a','e','i','o','u') and
```

```
LAT_N between 0 and 90
and LONG_W BETWEEN 0 and 180;
```

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

```
select distinct city from station
WHERE substr(city,1,1) not in ('a','e','i','o','u') and
LAT_N between 0 and 90
and LONG_W BETWEEN 0 and 180;
```

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

```
select distinct city from station
WHERE substr(city,-1,1) not in ('a','e','i','o','u') and
LAT_N between 0 and 90
and LONG_W BETWEEN 0 and 180;
```

15. Query the list of CITY names from STATION that either do not start with vowels or do not end with vowels.

```
select distinct city from station
WHERE substr(city,1,1) not in ('a','e','i','o','u') or
substr(city,-1,1) not in ('a','e','i','o','u') and
LAT_N between 0 and 90
and LONG_W BETWEEN 0 and 180;
```

16. Query the list of CITY names from STATION that do not start with vowels and do not end with vowels.

```
select distinct city from station
WHERE substr(city,1,1) not in ('a','e','i','o','u') and
substr(city,-1,1) not in ('a','e','i','o','u') and
LAT_N between 0 and 90
and LONG_W BETWEEN 0 and 180;
```

17. 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

Product table:

```
create table product
(product_id int NOT NULL,
product_name VARCHAR(30),
unit_price int,
PRIMARY KEY (product_id));
```

Sales table:

```
create table sales
(seller_id INT,
product_id INT,
buyer_id INT,
sale_date DATE,
quantity INT,
price INT,
Foreign Key (product_id) REFERENCES product(product_id));

insert into product VALUES
(1,'S8',1000),
(2,'G4',800),
(3,'iPhone',1400);

insert into sales VALUES
(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);


--try getting total sales count of all products and also get sales count of product sales in 1st quarter, now cl
select product_id, product_name from product
where product_id in (SELECT t1.product_id from
(select count(product_id) as No_of_sales,product_id from sales
GROUP BY product_id) t1
 join
(select count(product_id) as No_of_sales,product_id from sales
where sale_date BETWEEN '2019-01-01' and '2019-03-31'
GROUP BY product_id) t2 on t1.product_id=t2.product_id
where t1.No_of_sales = 1);
```

18. Write an SQL query to find all the authors that viewed at least one of their own articles.

```
create table views(article_id int,
author_id int,
viewer_id int,
view_date date);

select distinct author_id from views
where author_id=viewer_id
ORDER BY author_id;
```

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

```
create table delivery(delivery_id int,
customer_id int,
order_date date,
customer_pref_delivery_date date,
primary key (delivery_id));

select round((t2.immediate_deliveries/t1.total_deliveries*100),2) as immediate_percentage
from (select count(*) total_deliveries from delivery) t1
join
(select count(*) immediate_deliveries from delivery
where order_date=customer_pref_delivery_date) t2
```

20. Write an SQL query to find the ctr of each Ad. Round ctr to two decimal points.

```
create table ads
(
    ad_id INT,
    user_id INT,
    action ENUM('Clicked', 'Viewed', 'Ignored'),
    PRIMARY KEY (ad_id, user_id)
);

insert into ads VALUES
(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');

select distinct ad_id,
case
    when  round(sum(action = 'Clicked') / (sum(action = 'Clicked') + sum(action = 'Viewed')) * 100, 2) then 0
end as ctr
from ads
group by ad_id
order by ctr desc, ad_id;
```

21. Write an SQL query to find the team size of each of the employees

```
create table employee
(
    employee_id int,
    team_id int,
    PRIMARY KEY(employee_id)
);

insert into employee VALUES
(1,8),
(2,8),
(3,8),
(4,7),
(5,9),
(6,9);

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

```
22. Write an SQL query to find the type of weather in each country for November 2019.

create table countries
(
    country_id int,
    country_name varchar(40),
    PRIMARY KEY (country_id)
);

create table weather
(
    country_id int,
    weather_state int,
    day date,
    PRIMARY KEY (country_id,day)
);

insert into countries VALUES
(2,'USA'),
(3,'Australia'),
(7,'Peru'),
(5,'China'),
(8,'Morocco'),
(9,'Spain');

insert into weather VALUES
(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');

select c.country_name,
case
    when avg(weather_state)<=15 then 'Cold'
    when avg(weather_state)>=25 then 'Hot'
    else 'Warm'
end as weather_type
from countries c
JOIN
weather w
WHERE c.country_id = w.country_id
and day like '2019-11%'
GROUP BY c.country_id;



23. Write an SQL query to find the average selling price for each product. average_price should be
rounded to 2 decimal places.

CREATE table prices
(
    product_id int,
    start_date date,
    end_date date,
    price int,
    PRIMARY KEY (product_id, start_date, end_date)
);

create table UnitsSold
(
    product_id int,
    purchase_date date,
    units int
);

insert into prices VALUES
(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);

INSERT into UnitsSold VALUES
(1,'2019-02-25',100),
(1,'2019-03-01',15),
(2,'2019-02-10',200),
(2,'2019-03-22',30);

select p.product_id,round(sum(p.price*u.units)/sum(u.units),2) as average_price
from prices p
JOIN
UnitsSold u
on p.product_id=u.product_id
where u.purchase_date>=p.start_date and u.purchase_date<=p.end_date
GROUP BY p.product_id;
```

24. Write an SQL query to report the first login *date* for each player.

```
create table activity
(
    player_id int,
    device_id int,
    event_date date,
    games_played int,
    PRIMARY KEY (player_id, event_date)
);

insert into activity VALUES
(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);

select player_id,min(event_date) as first_login
from activity
GROUP BY player_id;
```

25. Write an SQL query to report the device that is first logged in for each player.

```
create table activity
(
    player_id int,
    device_id int,
    event_date date,
    games_played int,
    PRIMARY KEY (player_id, event_date)
);

insert into activity VALUES
(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);

select player_id,device_id from activity a
join
(select player_id pid,min(event_date) as first_login
from activity
GROUP BY player_id) tb1
on a.player_id = tb1.pid
where a.event_date = tb1.first_login;
```

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

```
create table products
(
    product_id int,
    product_name varchar(30),
    product_category varchar(20),
    PRIMARY KEY (product_id)
```

```
);

create table orders
(
    product_id int,
    order_date date,
    unit int,
    Foreign Key (product_id) REFERENCES products(product_id)
);

insert into products VALUES
(1,'Leetcode Solutions','Book'),
(2,'Jewels of Stringology','Book'),
(3,'HP','Laptop'),
(4,'Lenovo','Laptop'),
(5,'Leetcode Kit','T-shirt');

insert into orders VALUES
(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);


select p.product_name,sum(o.unit) as unit from products p
JOIN
orders o
on p.product_id = o.product_id
where o.order_date like '%-02-%'
GROUP BY p.product_name
having sum(unit)>=100;


27.
like '[a-z]%.@leetcode.com' or 'A-Z%.@leetcode.com'



28. 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.

create table customers
(
    customer_id int,
    name varchar(30),
    country varchar(20),
    PRIMARY KEY (customer_id)
);

create table orders
(
    order_id int,
    customer_id int,
    product_id int,
    order_date date,
    quantity int,
    PRIMARY KEY (order_id)
);

create table product
(
    product_id int,
    description varchar(30),
    price INT,
    PRIMARY KEY (product_id)
);

insert into customers VALUES
(1,'Winston','USA'),
(2,'Jonathan','Peru'),
(3,'Moustafa','Egypt');

insert into orders VALUES
(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);

insert into product VALUES
(10,'LC Phone',300),
(20,'LC T-Shirt',10),
(30,'LC Book',45),
(40,'LC Keychain',2);

select c.customer_id, c.name
from
customers c
JOIN
product p
JOIN
orders o
ON c.customer_id = o.customer_id and p.product_id = o.product_id
group by c.customer_id
having
(
    sum(case when o.order_date like '%-06-%' then o.quantity*p.price else 0 end) >= 100
    and
    sum(case when o.order_date like '%-07-%' then o.quantity*p.price else 0 end) >= 100
);



29. Write an SQL query to report the distinct titles of the kid-friendly movies streamed in June 2020.

create table TVProgram
(
    program_date date,
    content_id int,
    channel varchar(30),
    PRIMARY KEY (program_date, content_id)
);

create table content
(
    content_id int,
    title varchar(30),
    Kids_content enum('Y', 'N'),
    content_type varchar(30),
    PRIMARY KEY (content_id)
);

insert into TVProgram VALUES
('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');

insert into content VALUES
(1,'Leetcode Movie','N','Movies'),
(2,'Alg. for Kids','Y','Series'),
(3,'Database Sols','N','Series'),
(4,'Aladdin','Y','Movies'),
(5,'Cinderella','Y','Movies');

select c.title as title from TVProgram t
JOIN
content c
on t.content_id = c.content_id
where t.program_date like '%-06-%' and c.Kids_content = 'Y';



30. Write an SQL query to find the npv of each query of the Queries table.

create table NPV
(
    id int,
    year int,
    npv int,
    PRIMARY KEY (id, year)
```

```
);

create table queries
(
    id int,
    year int,
    PRIMARY KEY (id, year)
);

insert into NPV VALUES
(1,2018,100),
(7,2020,30),
(13,2019,40),
(1,2019,113),
(2,2008,121),
(3,2009,12),
(11,2020,99),
(7,2019,0);

insert into queries VALUES
(1,2019),
(2,2008),
(3,2009),
(7,2018),
(7,2019),
(7,2020),
(13,2019);

select q.id,q.year,
case
    when n.npv is NULL then 0
    else n.npv
end as npv
from NPV n
right JOIN
queries q
on (q.id, q.year) = (n.id, n.year);
```

32. 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*.

```
create table employees
(
    id int,
    name varchar(40),
    PRIMARY KEY(id)
);

create table employeeUNI
(
    id int,
    unique_id int,
    PRIMARY KEY (id, unique_id)
);

insert into employees VALUES
(1,'Alice'),
(7,'Bob'),
(11,'Meir'),
(90,'Winston'),
(3,'Jonathan');

insert into employeeUNI VALUES
(3,1),
(11,2),
(90,3);

select eu.unique_id unique_id,e.name
from employees e
left JOIN
employeeUNI eu
ON e.id = eu.id
ORDER BY unique_id;
```

33. Write an SQL query to report the distance travelled by each user.

```
create table users
```

```
(
    id int,
    name varchar(30),
    PRIMARY KEY (id)
);

create table rides
(
    id int,
    user_id int,
    distance int,
    PRIMARY KEY (id)
);

insert into users VALUES
(1,'Alice'),
(2,'Bob'),
(3,'Alex'),
(4,'Donald'),
(7,'Lee'),
(13,'Jonathan'),
(19,'Elvis');

insert into rides VALUES
(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);

select u.name,
case
    when sum(r.distance) is null then 0
    else sum(r.distance)
end as travelled_distance
from users u
left JOIN
rides r
on u.id = r.user_id
GROUP BY name
ORDER BY travelled_distance DESC, name ASC;


34. Incomplete question (data missing)



35. 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.

CREATE Table movies
(
    movie_id int,
    title varchar(30),
    PRIMARY KEY (movie_id)
);

create table users
(
    user_id int,
    name varchar(40),
    PRIMARY KEY (user_id)
);

create table MovieRating
(
    movie_id int,
    user_id int,
    rating int,
    created_at date,
    PRIMARY KEY (movie_id, user_id)
);

insert into movies VALUES
(1,'Avengers'),
```

```
(2,'Frozen 2'),
(3,'Joker');

insert into users VALUES
(1,'Daniel'),
(2,'Monica'),
(3,'Maria'),
(4,'James');

insert into MovieRating VALUES
(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');

(select u.name as result from users u
join
MovieRating mr
on u.user_id = mr.user_id
GROUP BY u.user_id
ORDER BY count(mr.rating) desc,u.name
LIMIT 1)
UNION
(select m.title as result from movies m
join
MovieRating mr
on m.movie_id = mr.movie_id
where mr.created_at like '2020-02-%'
GROUP BY m.movie_id
ORDER BY avg(mr.rating) desc,m.title ASC
LIMIT 1);
```

38. Write an SQL query to find the id and the name of all students who are enrolled in departments that no longer exist.

```
create table departments
(
    id int,
    name varchar(30),
    PRIMARY KEY (id)
);

create table students
(
    id int,
    name varchar(30),
    department_id int,
    PRIMARY KEY (id)
);

insert into departments VALUES
(1,'Electrical Engineering'),
(7,'Computer Engineering'),
(13,'Business Administration');

insert into students VALUES
(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);

select id,name from students
where department_id not in (select id from departments);
```

39. 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.

```
create table calls
(
    from_id int,
    to_id int,
    duration int
);

insert into calls VALUES
(1,2,59),
(2,1,11),
(1,3,20),
(3,4,100),
(3,4,200),
(3,4,200),
(4,3,499);

select tb1.from_id person1,tb1.to_id person2,
CASE
    when tb2.call_count is NULL then tb1.call_count
    else tb1.call_count+tb2.call_count
end as count_id,
CASE
    when tb2.total_duration is NULL then tb1.total_duration
    else tb1.total_duration+tb2.total_duration
end as total_duration
from (select from_id,to_id,count(duration) call_count,sum(duration) total_duration from calls
where from_id<to_id
GROUP BY from_id,to_id) tb1
left JOIN
(select from_id,to_id,count(duration) call_count,sum(duration) total_duration from calls
where from_id>to_id
GROUP BY from_id,to_id) tb2
on (tb1.from_id,tb1.to_id) = (tb2.to_id,tb2.from_id);
```

41. Write an SQL query to report the *number* of cubic feet of volume the inventory occupies in each warehouse.

```
create table warehouse
(
    name varchar(30),
    product_id int,
    units int,
    PRIMARY KEY (name, product_id)
);

create table products
(
    product_id int,
    product_name varchar(30),
    Width int,
    Length int,
    Height int,
    PRIMARY KEY (product_id)
);

insert into warehouse VALUES
('LCHouse1',1,1),
('LCHouse1',2,10),
('LCHouse1',3,5),
('LCHouse2',1,2),
('LCHouse2',2,2),
('LCHouse3',4,1);

insert into products VALUES
(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);

select w.name,
(sum(p.width*p.length*p.height*w.units)) volume from products p
JOIN
warehouse w
on p.product_id = w.product_id
GROUP BY w.name;
```

42. Write an SQL query to report the difference between the *number* of apples and oranges sold each day.

```
CREATE table sales
(
    sale_date date,
    fruit enum('apples','oranges'),
    sold_num int,
    PRIMARY KEY (sale_date, fruit)
);

insert into sales VALUES
('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);

select sa.sale_date sale_date,(sa.sold_num-so.sold_num) diff
from sales sa
JOIN
sales so
on sa.sale_date = so.sale_date
where sa.fruit = 'apples' and so.fruit = 'oranges';
```

43. 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.

```
create table activity
(
    player_id int,
    device_id int,
    event_date date,
    games_played int,
    PRIMARY KEY (player_id, event_date)
);

insert into activity VALUES
(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);

select round(count(DISTINCT tb1.player_id)/count(DISTINCT tb2.player_id),2) as fraction
from(select a1.player_id
from activity a1
JOIN
activity a2
on day(a1.event_date)+1=day(a2.event_date) and month(a1.event_date)=month(a2.event_date)) tb1
JOIN
activity tb2;
```

44. Write an SQL query to report the managers with at least five direct reports.

```
create table employee
(
    id int,
    name varchar(30),
    department varchar(3),
    managerId int,
    PRIMARY KEY (id)
);

insert into employee VALUES
(101,'John','A',Null),
(102,'Dan','A',101),
(103,'James','A',101),
(104,'Amy','A',101),
(105,'Anne','A',101),
(106,'Ron','B',101);

select tb1.name name
from(select e2.name,count(e2.name) cnt from employee e1
JOIN
```

```
employee e2
on e1.managerId = e2.id
GROUP BY e2.name) tb1
where tb1.cnt>=5;
```

45. 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).

```
create table department
(
    dept_id int,
    dept_name varchar(30),
    PRIMARY KEY (dept_id)
);

create table student
(
    student_id int,
    student_name varchar(30),
    gender varchar(1),
    dept_id int,
    PRIMARY KEY (student_id),
    Foreign Key (dept_id) REFERENCES department(dept_id)
);

insert into department VALUES
(1,'Engineering'),
(2,'Science'),
(3,'Law');

insert into student VALUES
(1,'Jack','M',1),
(2,'Jane','F',1),
(3,'Mark','M',2);


select d.dept_name,
case
    when count(s.dept_id) is null then 0
    else count(s.dept_id)
end as student_number
from department d
left JOIN
student s
on d.dept_id = s.dept_id
GROUP BY d.dept_name;
```

46. Write an SQL query to report the customer ids from the Customer table that bought all the products in the Product table.

```
create table product
(
    product_key int,
    PRIMARY KEY(product_key)
);

create table customer
(
    customer_id int,
    product_key int,
    Foreign Key (product_key) REFERENCES product(product_key)
);

insert into product VALUES
(5),
(6);

insert into customer VALUES
(1,5),
(2,6),
(3,5),
(3,6),
(1,6);

select tb1.customer_id
from(select customer_id,count(product_key) cnt from customer
GROUP BY customer_id) tb1
JOIN
```

```
(select count(product_key) cnt from product) tb2
where tb1.cnt = tb2.cnt;
```

47. 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.

```
create table employee
(
    employee_id int,
    name varchar(30),
    experience_years int,
    PRIMARY KEY (employee_id)
);

create table project
(
    project_id int,
    employee_id int,
    PRIMARY KEY (project_id, employee_id),
    Foreign Key (employee_id) REFERENCES employee(employee_id)
);

insert into employee VALUES
(1,'Khaled',3),
(2,'Ali',2),
(3,'John',3),
(4,'Doe',2);

insert into project VALUES
(1,1),
(1,2),
(1,3),
(2,1),
(2,4);

SELECT project_id,employee_id
FROM
(SELECT p.project_id,p.employee_id,
DENSE_RANK() OVER(PARTITION BY p.project_id ORDER BY e.experience_years DESC) as rnk
FROM project as p JOIN employee as e
ON p.employee_id = e.employee_id
) tb1
WHERE rnk = 1;
```

48. INCOMPLETE QUESTION (data missing)
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.

```
create table books
(
    book_id int,
    name varchar(30),
    available_from date,
    PRIMARY KEY (book_id)
);

create table orders
(
    order_id int,
    book_id int,
    quantity int,
    dispatch_date date,
    PRIMARY KEY (order_id),
    Foreign Key (book_id) REFERENCES books(book_id)
);

insert into books VALUES
(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');
```

49. 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.

```
create table enrollments
```

```
(
    student_id int,
    course_id int,
    grade int,
    PRIMARY KEY (student_id, course_id)
);

insert into enrollments VALUES
(2,2,95),
(2,3,95),
(1,1,90),
(1,2,99),
(3,1,80),
(3,2,75),
(3,3,82);

select e1.student_id, min(e1.course_id) as course_id, e1.grade
from enrollments e1
where e1.grade in
(select max(grade) as mg
from enrollments e2 where e1.student_id = e2.student_id)
group by e1.student_id,e1.grade
order by e1.student_id;


50. 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.

create table players
(
    player_id int,
    group_id int,
    PRIMARY KEY (player_id)
);

insert into players VALUES
(15,1),
(25,1),
(30,1),
(45,1),
(10,2),
(35,2),
(50,2),
(20,3),
(40,3);

create table matches
(
    match_id INT,
    first_player int,
    second_player int,
    first_score int,
    second_score int,
    PRIMARY KEY (match_id)
);

insert into matches VALUES
(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);


with ps as
(SELECT first_player AS player_id, first_score  AS score
FROM matches
UNION ALL
SELECT second_player AS player_id,second_score  AS score
FROM matches)


select group_id,player_id
from(SELECT  group_id,player_id,
row_number() over(partition BY tb1.group_id order by tb1.score DESC) rnk
from (SELECT p.group_id, ps.player_id, Sum(ps.score) AS score
FROM players p
INNER JOIN
ps
ON  p.player_id = ps.player_id
GROUP  BY ps.player_id
```

```
ORDER  BY group_id, score DESC, player_id) tb1) tb2
where tb2.rnk = 1;
```