SET-1

```
create database set1

use set1
```

Sample Dataset-1

```
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)
```

Q1. 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;
```

Q2. 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 City_in_USA from city where countrycode = "usa" and population > 120000;
```

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

```
select * from city;
```

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

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

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

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

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

```
select name City_in_Japan from city where countrycode = "JPN";
```

Sample Dataset-2

```
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);

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

```
select City, State from station
```

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

```
select distinct(City) as City_with_even_Id from station where id % 2 = 0;
```

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

```
select difference(count(city), count(distinct(city))) from station

select (count(city) - count(distinct(city))) as Diff_btwn_count_city 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

```
select * from (select city, char_length(city) city_length from station

order by city, city_length) a group by a.city_length order by a.city_length limit 1;
```

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.

```
select distinct(city) as City_starts_with_Vowels from station

where city like "a%" or

city like "e%" or

city like "i%" or

city like "o%" or

city like "u%" order by city
```

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

```
select distinct(city) as City_ends_with_Vowels from station

where city like "%a" or

city like "%e" or
```

```
city like "%i" or
city like "%o" or
city like "%u" order by city
```

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

```
select distinct(city) as City_donot_starts_with_Vowels from station
where city not like "a%" and
city not like "e%" and
city not like "i%" and
city not like "o%" and
city not like "u%"
order by city
```

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

```
select distinct(city) as City_donot_ends_with_Vowels from station
where city not like "%a" and
city not like "%e" and
city not like "%i" and
city not like "%o" and
city not like "%u" order by city
```

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

```
select distinct(city) as City_donot_starts_with_Vowels from station
where city not like "a%" and
city not like "e%" and
city not like "i%" and
city not like "o%" and
city not like "u%"
or
```

```
city not like "%a" and
city not like "%e" and
city not like "%i" and
city not like "%o" and
city not like "%u" order by city
```

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

```
select distinct(city) as City_donot_startsandends_with_Vowels from station
where city not like "a%" and
city not like "e%" and
city not like "i%" and
city not like "o%" and
city not like "u%"
and
city not like "%a" and
city not like "%e" and
city not like "%i" and
city not like "%o" and
city not like "%u" order by city
```

Q17. 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.

```
create table product
( product_id int not null,
product_name varchar(50),
unit_price int,
constraint pk primary key(product_id)
);
create table Sales
(seller_id int,
```

```
product_id int,
buyer_id int,
sale_date date,
quantity int,
price int,
constraint fk 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)
select product_id from sales
where sale_date between "2019-01-01" and "2019-03-31";
```

Q18. 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.

```
create table views
(article_id int,
author_id int,
viewer_id int,
view_date date
)
insert into views values
(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");

select * from views

select distinct(author_id) Author_viewed_his_own_article from views where author_id = viewer_id

order by author_id

Q19. 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,

constraint pk primary key(delivery_id)

)

insert into delivery values

(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")

select * from delivery

select count(*) *100 / (select count(*) from delivery) immediate_percentage

from delivery where order_date = customer_pref_delivery_date

Q20. 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.

create table ads

(

ad_id int,

user_id int,

```
action varchar(20),
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 * from ads
```

Q21. Write an SQL query to find the team size of each of the employees. Return result table in any order.

```
create table if not exists employee
(
employee_id int not null,
team_id int,
constraint pk 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) as team_size
from employee order by employee_id;
select e.employee_id, (select count(team_id) from Employee where e.team_id = team_id) as
team_size from Employee e;
```

Q22Write 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

```
create table if not exists countries
(
country_id int,
country_name varchar(20),
constraint pk primary key (country_id));
create table if not exists weather
(country_id int,
weather_state int,
day date);
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 inner join weather w on c.country_id = w.country_id
where day between "2019-11-01" and "2019-11-30"
group by country_name
```

Q24. Write an SQL query to report the first login date for each player. Return the result table in any order.

```
create table activity
(player_id int,
device_id int,
event_date date,
games_played int,
constraint pk 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_date from activity
group by player_id
```

Q25. Write an SQL query to report the device that is first logged in for each player. Return the result table in any order

```
select player_id, min(device_id) as device_first_logged_in from activity
group by player_id
```

Q26. 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.

```
create table if not exists products
```

```
(product_id int,
product_name varchar(100),
product_category varchar(100),
constraint pk primary key (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");
create table orders
(product_id int,
order_date date,
unit int,
constraint fkey foreign key (product_id) references products(product_id));
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 o.product_id, p.product_name, sum(o.unit) unit from orders o inner join products p
on p.product_id = o.product_id
group by product_id
```

where o.unit >=100

and order_date in ("2020-02-01" and "2020-02-28")

Q27. 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.

create table users

(user_id int,

name varchar(100),

mail varchar(100),

constraint pk primary key(user_id));

insert into users values

(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");

select * from users

where mail like ("%@leetcode.com", "%-%") mail in ("%_%", ".", "-")

create database set2
use set2

Q51. Write an SQL query to report the name, population, and area of the big countries. Return the result table in any order

```
create table if not exists country
(name varchar(20),
continent varchar(20),
area int,
population bigint,
gdp bigint,
constraint pk primary key(name));
insert into country values
("Afghanistan", "Asia", 652230, 25500100, 20343000000),
("Albania", "Europe", 28748, 2831741, 12960000000),
("Algeria", "Africa", 2381741, 37100000, 188681000000),
("Andorra", "Europe", 468, 78115, 3712000000),
("Angola", "Africa", 1246700, 20609294, 100990000000);
select name, population, area from country
where area >=  3000000 or population >= 25000000
```

52. Write an SQL query to report the names of the customer that are not referred by the customer with id = 2. Return the result table in any order.

```
create table if not exists customer
(id int,
name varchar(20),
referee_id int,
constraint pk primary key(id));
insert into customer values
```

(1, "Will", null),

(2, "Jane", null),

(3, "Alex", 2),

(4, "Bill", null),

(5, "Zack", 1),

(6, "Mark", 2);

select name from customer

where referee_id != 2 or referee_id = null

Q53. Write an SQL query to report all customers who never order anything. Return the result table in any order.

create table if not exists customers

(id int not null,

name varchar(20),

primary key(id));

insert into customers values

(1, "Joe"), (2, "Henry"), (3, "Sam"), (4, "Max")

create table if not exists orders

(id int not null,

customerId int,

primary key (id),

foreign key (customerId) references customers(id));

insert into orders values

(1, 3), (2, 1)

select c.name as customers from customers c left join orders o

on c.id = o.customerId

where c.id not in (select customerid from orders)

Q54. Write an SQL query to find the team size of each of the employees. Return result table in any order.

create table if not exists employee

(employee_id int not null,

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) as team_size

from employee order by employee_id
```

Q56 already done refer Q24 in set1

Q57. Write an SQL query to find the customer_number for the customer who has placed the largest number of orders. The test cases are generated so that exactly one customer will have placed more orders than any other customer

```
create table if not exists orders2

(order_number int not null,

customer_number int,

primary key (order_number));

insert into orders2 values

(1, 1), (2, 2), (3, 3), (4, 3);

select customer_number from orders2

group by customer_number order by count(customer_number) desc limit 1
```

Q62. Write a SQL query for a report that provides the pairs (actor_id, director_id) where the actor has cooperated with the director at least three times.

```
create table if not exists actordirector

(actor_id int,

director_id int,

timestamp int not null,

primary key (timestamp));

insert into actordirector values

(1, 1, 0), (1, 1, 1), (1, 1, 2), (1, 2, 3), (1, 2, 4), (2, 1, 5), (2, 1, 6);

select actor_id, director_id from actordirector

where count(actor_id) >= (select count(director_id) from actordirector) group by

director_id)

group by actor_id
```

Q63. Write an SQL query that reports the product_name, year, and price for each sale_id in the Sales table. Return the resulting table in any order

```
create table if not exists Product
(product_id int,
product_name varchar(20),
primary key (product_id));
insert into product values
(100, "Nokia"), (200, "Apple"), (300, "Samsung");
create table if not exists sales
(sale_id int,
product_id int,
year int,
quantity int,
price int,
primary key (sale_id, year),
foreign key (product_id) references product (product_id));
insert into sales values
(1, 100, 2008, 10, 5000), (2, 100, 2009, 12, 5000), (7, 200, 2011, 15, 9000);
select p.product_name, s.year, s.price from sales s inner join product p
on s.product_id = p.product_id
```

Q64. Write an SQL query that reports the average experience years of all the employees for each project, rounded to 2 digits.

```
create table if not exists employee2
(employee_id int not null,
name varchar(20),
experience_years int,
primary key (employee_id));
insert into employee2 values
(1, "Khaled", 3), (2, "Ali", 2), (3, "John", 1), (4, "Doe", 2)
create table if not exists project
(project_id int,
employee_id int,
primary key (project_id, employee_id),
```

```
foreign key (employee_id) references employee2 (employee_id));
insert into project values
(1, 1), (1, 2), (1, 3), (2, 1), (2, 4);
select p.project_id, round(avg(e.experience_years), 2) average_years
from employee2 e inner join project p
on e.employee_id = p.employee_id
group by project_id
```

Q65. Write an SQL query that reports the best seller by total sales price, If there is a tie, report them all.

```
create table if not exists product2
(product_id int not null,
product_name varchar(20),
unit_price int,
primary key (product_id));
insert into product2 values
(1, "S8", 1000), (2, "G4", 800), (3, "iPhone", 1400);
create table if not exists sales2
(seller_id int,
product_id int,
buyer_id int,
sale_date date,
quantity int,
price int,
foreign key (product_id) references product2(product_id));
insert into sales2 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);
select s.seller_id, sum(s.price) total_price from sales2 s inner join product2 p
on p.product_id = s.product_id
```

group by seller_id order by price desc

Q66. Write an SQL query that reports the buyers who have bought S8 but not iPhone. Note that S8 and iPhone are products present in the Product table.

```
select s.buyer_id from sales2 s inner join product2 p
on p.product_id = s.product_id
where product_name = "s8"
```

Q67. Write an SQL query to compute the moving average of how much the customer paid in a seven days window (i.e., current day + 6 days before). average_amount should be rounded to two decimal places. Return result table ordered by visited_on in ascending order

```
create table if not exists customer2
(customer_id int,
name varchar(20),
visited_on date,
amount int,
primary key (customer_id,visited_on));
insert into customer2 values
(1, "Jhon", "2019-01-01", 100),
(2, "Daniel", "2019-01-02", 110),
(3, "Jade", "2019-01-03", 120),
(4, "Khaled", "2019-01-04", 130),
(5, "Winston", "2019-01-05", 110),
(6, "Elvis", "2019-01-06", 140),
(7, "Anna", "2019-01-07", 150),
(8, "Maria", "2019-01-08", 80),
(9, "Jaze", "2019-01-09", 110),
(1, "Jhon", "2019-01-10", 130),
(3, "Jade", "2019-01-10", 150);
select customer_id, visited_on, amount,
sum(amount) over(order by visited_on range between interval "6" day preceding and current row)
as moving_avg
from customer2
```

Q68. Write an SQL query to find the total score for each gender on each day. Return the result table ordered by gender and day in ascending order

```
create table if not exists scores
(player_name varchar(20),
gender varchar(1),
day date,
score_points int,
primary key (gender,day));
insert into scores values
("Aron", "F", "2020-01-01", 17),
("Alice", "F", "2020-01-07", 23),
("Bajrang", "M", "2020-01-07", 7),
("Khali", "M", "2019-12-25", 11),
("Slaman", "M", "2019-12-30", 13),
("Joe", "M", "2019-12-31", 3),
("Jose", "M", "2019-12-18", 2),
("Priya", "F", "2019-12-31", 23),
("Priyanka", "F", "2019-12-30", 17);
select gender, day, sum(score_points)
over(partition by gender order by day) as total from scores
```

Q69. Write an SQL query to find the start and end number of continuous ranges in the table Logs.

```
create table if not exists students
(student_id int,
student_name varchar(20),
primary key(student_id));
create table if not exists subjects
(subject_name varchar(20),
primary key (subject_name));
create table if not exists examinations
```

```
(student_id int,
subject_name varchar(20),
foreign key (student_id) references students(student_id),
foreign key (subject_name) references subjects(subject_name));
insert into students values
(1, "Alice"), (2, "Bob"), (13, "John"), (6, "Alex");
insert into subjects values
("Math"), ("Physics"), ("Programming");
insert into examinations values
(1, "Math"), (1, "Physics"), (1, "Programming"), (2, "Programming"),
(1, "Physics"), (1, "Math"), (13, "Math"), (13, "Programming"),
(13, "Physics"), (2, "Math"), (1, "Math");
select st.student_id, st.student_name, sub.subject_name,
sum(st.student_id) over (partition by student_name,subject_name) tot
from Examinations e
inner join students st on st.student_id=e.student_id
inner join subjects sub on sub.subject_name=e.subject_name
```

create database set3
use set3

Q101. Write an SQL query to show the second most recent activity of each user. If the user only has one activity, return that one. A user cannot perform more than one activity at the same time.

```
create table useractivity

(username varchar(20),

activity varchar(25),

startDate Date,

endDate Date);

insert into useractivity values

("Alice", "Travel", "2020-02-12", "2020-02-20"),

("Alice", "Dancing", "2020-02-21", "2020-02-23"),

("Alice", "Travel", "2020-02-24", "2020-02-28"),

("Bob", "Travel", "2020-02-11", "2020-02-18");

select username, activity, startdate, enddate from (select u.*,

row_number() over (partition by username order by startdate desc) rnk,

count(activity) over(partition by username) num

from useractivity u) tmp

where (rnk=1 and num =1) or (num !=1 and rnk =2)

## Q102 same Q 102
```

Q103. Query the Name of any student in STUDENTS who scored higher than 75 Marks. Order your output by the last three characters of each name. If two or more students both have names ending in the same last three characters (i.e.: Bobby, Robby, etc.), secondary sort them by ascending ID.

```
create table if not exists students

(id int,

name varchar(20),

marks int);
```

```
insert into students values
(1, "Ashley", 81), (2,"Samantha",75), (4,"Julia",76),(3, "Belvet",84);
select name from students
where marks>75
order by name like "%___" desc
```

Q104. Write a query that prints a list of employee names (i.e.: the name attribute) from the Employee table in alphabetical order.

```
create table if not exists employee
(employee_id bigint,
name varchar(30),
months int,
salary int);
insert into employee values
(12228, "Rose", 15, 1968), (33645, "Angela", 1, 3443), (45692, "Frank",17,1608),
 (56118, "Patrick", 7, 1345), (59725, "Lisa", 11, 2330),
 (74197, "Kimberly", 16, 4372), (78454, "Bonnie", 8, 1771), (83565, "Michael", 6, 2017),
 (98607, "Todd", 5, 3396), (99989, "Joe", 9, 3573);
select name from employee
order by name
```

Q105. Write a query that prints a list of employee names (i.e.: the name attribute) for employees in Employee having a salary greater than $2000 per month who have been employees for less than 10 months. Sort your result by ascending employee_id.

```
select name from employee
where salary > 2000 and months < 10
order by name
## Q 106
## Q 108
select max(tot_earnings) from
(select e.*, (salary* months) as tot_earnings
from employee e) tmp
## Q111
```

```
create table if not exists bst
(n int, p int);
insert into bst values
(1,2),(3,2),(6,8),(9,8),(2,5),(8,5),(5,null)
select n,
case
when n not in (select p from bst where p is not null) then "Leaf"
when p is null then "Root"
else "Inner"
end as type_char
from bst order by n

## Q 117 sama as 103
## 118,119 sames as 104, 104
```