Question 51:

```
create table world
(name varchar(30) primary key, continent varchar(30),
area int,population int , gdp bigint);

insert into world 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);
```

```
--Approach I:
SELECT name, population, area
FROM world WHERE area >= 3000000 OR population >= 25000000;

--Approach II: Using WHERE clause and UNION
SELECT name, population, area FROM world WHERE area >= 3000000
UNION
SELECT name, population, area FROM world WHERE population >= 25000000;
```

Question 52:

```
create table customer
(id int primary key,name varchar(30),referee_id int);

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 IS NULL;
OR

SELECT name FROM customer  WHERE referee_id != 2 OR referee_id IS NULL;
```

Question 53:

```
create table customers (id int primary key,name varchar(30));

create table orders (id int primary key, customerId int,
Foreign key (customerId) references customers(id));

insert into customers values (1, 'Joe'),(2, 'Henry'),(3, 'Sam'),(4, 'Max');

insert into orders values (1,3),(2,1);
```

```
select customers.name as 'Customers'
from customers
where customers.id not in
(
    select customerid from orders
);
```

Question 54:

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

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

```
# Approach 1:
select
      e.employee_id,
      (select
      count(team_id)
      from employee
      where e.team_id = team_id
      ) as team_size
from employee e;

# Approach 2:
SELECT
      employee_id, COUNT(team_id) OVER (PARTITION BY team_id) team_size
FROM employee;
```

```sql
# Approach 3:
select
      e1.employee_id,
      count(*) as team_size
from
      employee e1
left join
      employee e2
on
      e1.team_id = e2.team_id
group by
      e1.employee_id;
```

Question 55:

```sql
create table person(
id int primary key,
name varchar(20),
phone_number varchar(20));

create table country(
name varchar(20),
country_code varchar(20) primary key);

create table calls(
caller_id int,
callee_id int,
duration int);

insert into person values
(3,'Jonathan','051-1234567'),
(12,'Elvis','051-7654321'),
(1,'Moncef','212-1234567'),
(2,'Maroua','212-652365'),
(7,'Meir','972-1234567'),
(9,'Rachel','972-0011100');


insert into country VALUES
('Peru',51),
('Israel',972),
('Morocco',212),
```

```
('Germany',49),
('Ethiopia',251);

insert into calls VALUES
(1,9,33),
(2,9,4),
(1,2,59),
(3,12,102),
(3,12,330),
(12,3,5),
(7,9,13),
(7,1,3),
(9,7,1),
(1,7,7);
```

Algorithm :
Step 1: People and Country
Step 2: Caller and Duration
Step 3: Join,avg, group by ,having

```
select c.name as country
from person p
inner join country c
on left (p.phone_number,3) = c.country_code
inner join (select caller_id as id, duration
            from calls
            union all
            select callee_id as id, duration
            from calls) phn
on p.id = phn.id
group by country
having avg(duration) > (select avg(duration) from calls);
```

Question 56:

```
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, '2016-03-02', 6),
(2, 3, '2017-06-25', 1),
(3, 1, '2016-03-01', 0),
(3, 4, '2016-07-03', 5);
```

```
select player_id,device_id
from
(
 select
      player_id,
      device_id,
      event_date,
      row_number() over(partition by player_id order by event_date asc)
      as rw
      from activity) a
where a.rw = 1;
```

Question 57:

```
create table orders
(order_number int primary key,
customer_number int);

insert into orders values (1,1),(2,2),(3,3),(4,3);
```

```
select
      customer_number
from
      (select customer_number, count(order_number) order_count
      from orders group by customer_number
 ) a
order by order_count desc limit 1;
```

Question 58:

```
create table cinema
(seat_id int primary key AUTO_INCREMENT, free bool);

insert into cinema values
```

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

To solve this problem, first, we need to do join the cinema table to itself using a self-join. And to find the consecutive seats we have to use abs(t1.seat_id - t2.seat_id) = 1 because the value of the t1.seat_id will be one more or less than the value of the t2.seat_id and we also have to make sure that the seat is available by using another condition for joining the table as t1.free = 1 AND t2.free = 1.

```
SELECT
    DISTINCT t1.seat_id
FROM cinema AS t1 JOIN cinema AS t2
ON abs(t1.seat_id - t2.seat_id) = 1
AND t1.free = 1 AND t2.free = 1
ORDER BY 1 ;
```

Question 59:

```
create table SalesPerson(
sales_id int,name varchar(20),salary int,commission_rate int,hire_date date);
insert into SalesPerson values
(1,'John', 100000,6,'2006-4-1'),
(2,'Amy',12000,5,'2010-5-1'),
(3,'Mark',65000,12,'2008-12-25'),
(4,'Pam',25000,25,'2005-1-1'),
(5,'Alex',5000,10,'2007-2-3');

create table Company
(com_id int,name  varchar(20),city varchar(20));

insert into Company values
(1,'RED','Boston'),
(2,'ORANGE','New York'),
(3,'YELLOW','Boston'),
(4,'GREEN','Austin');

create table Orders(
order_id int,order_date date,com_id int,sales_id int,amount int);

insert into Orders values
(1,'2014-1-1',3,4,10000),
(2,'2014-2-1',4,5,5000),
(3,'2014-3-1',1,1,50000),
(4,'2014-4-1',1,4,25000);
```

```
with cte AS
(select sales_id from  Orders o left join  Company c on o.com_id = c.com_id
where c.name ='RED')
select name from SalesPerson where sales_id not IN
(select * from cte);
```

Question 60:

```
create table triangle
(x int,y int,z int,
primary key (x,y,z));

insert into triangle values
(13,15,30),(10,20,15);
```

```
SELECT
    x,
    y,
    z,
    IF(x + y > z AND y + z > x AND z + x > y, 'Yes', 'No') triangle
FROM
    triangle;
```

Question 61:

```
create table point
(x int primary key);

insert into point values
(-1),(0),(2);
```

To solve the problem, we have to first join the points table to itself ON p1.x != p2.x because we don't want to subtract the number with itself as it will always give us 0. Next, to find the distance, we have to subtract x1 and x2 and take the ABS value of the difference as distance can not be negative. Now we can just use the MIN function to get the answer.

```

SELECT MIN(abs(p2.x - p1.x)) shortest
FROM point p1 JOIN point p2
ON p1.x != p2.x;
```

Question 62:

```
create table actorDirector
(actor_id int,
director_id int,
timestamp int primary key);
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
GROUP BY actor_id, director_id
HAVING COUNT(*) >= 3;
```

Question 63:

```
create table Sales
(sale_id int,
product_id int,
year int,
quantity int,
price int,
primary key (sale_id,year));

create table Product
(product_id int primary key,
product_name varchar(30));


insert into Sales values
(1,100,2008,10,5000),
(2,100,2009,12,5000),
(7,200,2011,15,9000);
```

```
insert into Product values
(100,'Nokia'),
(200,'Apple'),
(300,'Samsung');
```

```
select
    product_name, year, price
from Sales
join Product
on Sales.product_id = Product.product_id;
```

Question 64:

**** Same as 47 in set 1**

Question 65:

**** Same as 17 in Set 1**

Question 66:

**** Same as 17 in Set 1**

Question 67:

```
create table Customer
(customer_id int,
name varchar(30),
visited_on date,
amount int,
primary key (customer_id,visited_on));

insert into Customer 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);
```

Step 1:
we have to group the data by visited_on and calculate the total amount.
Step 2:

we have to use 3 window functions and change their default value to ROWS BETWEEN 6 PRECEDING AND CURRENT ROW for the seven days window.

Step 3:

We almost has the result. All we need to do is exclude the top six results using the ranking column and select only the columns that is required in the solution.

```
# Approach 1:

WITH result as (
SELECT
    visited_on,
    SUM(amount) as amount
FROM customer
GROUP BY visited_on
), result2 as (
SELECT
    visited_on,
    SUM(amount) OVER(ORDER BY visited_on ROWS BETWEEN 6 PRECEDING AND CURRENT
ROW) as amount,
    ROUND(AVG(amount) OVER(ORDER BY visited_on ROWS BETWEEN 6 PRECEDING AND
CURRENT ROW),2) as average_amount,
    DENSE_RANK() OVER(ORDER BY visited_on) as rnk
FROM result
)
SELECT
    visited_on, amount,
    average_amount
FROM result2
WHERE rnk > 6;

# Approach 2:

select c1.visited_on, sum(c2.amount) as amount,
    round(avg(c2.amount), 2) as average_amount
from (select visited_on, sum(amount) as amount
      from Customer group by visited_on) c1
join (select visited_on, sum(amount) as amount
      from Customer group by visited_on) c2
on datediff(c1.visited_on, c2.visited_on) between 0 and 6
group by c1.visited_on
having count(c2.amount) = 7
ORDER BY c1.visited_on;
```

Question 68:

```
create table scores
(player_name varchar(20),
gender varchar(20), 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);
```

To obtain the total score, calculate the sum of score_points for all entries where day is less than or equal to the current day. Use select sum(socre_points) and use gender and day as filters to achieve this. Then select gender, day and the total score from table Scores, group the entries by gender and day, and order the entries by gender and day.

```
select s.gender, s.day,
      (select sum(score_points)
      from scores
      where
      gender = s.gender and day <= s.day) as total
from scores s
group by gender, day;
```

Question 69:

```
create table logs (log_id int primary key);

insert into logs values (1),(2),(3),(7),(8),(10);
```

For each value log in table Logs, if log - 1 does not exist in Logs, then log is the start id of an interval. If log + 1 does not exist in Logs, then log is the end id of an interval. Join the start ids and the end ids to obtain the result.

```
select log_start.log_id as START_ID, min(log_end.log_id) as END_ID from
    (select log_id from logs where log_id - 1 not in (select * from logs))
log_start,
```

```
    (select log_id from logs where log_id + 1 not in (select * from logs))
log_end
    where log_start.log_id <= log_end.log_id
    group by log_start.log_id;
```

Question 70 :

```
create table students
(student_id int primary key,
student_name varchar(30));

create table subjects
(subject_name varchar(30) primary key);

create table examinations
(student_id int,
subject_name varchar(30));

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
      a.student_id, a.student_name, b.subject_name,
      count(c.subject_name) as attended_exams
from
      students as a
join
      subjects as b
left join
      examinations as c
on
      a.student_id = c.student_id
and
      b.subject_name = c.subject_name
```

```
group by a.student_id, b.subject_name
order by a.student_id, b.subject_name;
```

Question 71:

```
create table employees
(employee_id int primary key,
employee_name varchar(30),
manager_id int);

insert into employees values
(1, 'Boss', 1),
(3, 'Alice', 3),
(2, 'Bob', 1),
(4, 'Daniel', 2),
(7, 'Luis', 4),
(8, 'Jhon', 3),
(9, 'Angela', 8),
(77, 'Robert', 1);
```

```
-- Using JOIN

select a.employee_id as EMPLOYEE_ID
from
    employees as a
left join
    employees as b on a.manager_id = b.employee_id
left join
    employees as c on b.manager_id = c.employee_id
left join
    employees as d on c.manager_id = d.employee_id
where
    a.employee_id != 1
and
    d.employee_id = 1;

-- Using Subquery

select
      employee_id as EMPLOYEE_ID
from employees where manager_id in
```

```
      (
      select employee_id from employees WHERE manager_id
      in
         (select employee_id from employees where manager_id =1)
      )
and employee_id !=1;
```

Question 72:

```
create table transactions
(id int primary key,
country varchar(30),
state enum('approved','declined'),
amount int,
trans_date date);

insert into transactions values
(121, 'US', 'approved', 1000, '2018-12-18'),
(122, 'US', 'declined', 2000, '2018-12-19'),
(123, 'US', 'approved', 2000, '2019-01-01'),
(124, 'DE', 'approved', 2000, '2019-01-07');
```

Use date_format(trans_date, '%Y-%m') to obtain the month of each transaction.
For trans_count and trans_total_amount, simply use count and sum to obtain the values.
For approved_count and approved_total_amount, use if statement to obtain the values.
The query result is grouped by month and country.

```
select
      date_format(trans_date,"%Y-%m") as month,
      country,
      count(id) as trans_count,
      sum(case when state='approved' then 1  else 0 end) as approved_count,
      sum(amount) as trans_total_amount,
      sum(case when state='approved' then amount  else 0 end) as
      approved_total_amount
from
      transactions
group by month, country;
```

Question 73:

```
create table actions
( user_id int,
post_id int,
action_date date,
action enum ('view', 'like', 'reaction', 'comment', 'report', 'share'),
extra varchar(30));

create table removals
(post_id int primary key,
remove_date date);

insert into actions values
(1, 1, '2019-07-01', 'view', null),
(1, 1, '2019-07-01', 'like', null),
(1, 1, '2019-07-01', 'share', null),
(2, 2, '2019-07-04', 'view', null),
(2, 2, '2019-07-04', 'report', 'spam'),
(3, 4, '2019-07-04', 'view', null),
(3, 4, '2019-07-04', 'report', 'spam'),
(4, 3, '2019-07-02', 'view', null),
(4, 3, '2019-07-02', 'report', 'spam'),
(5, 2, '2019-07-03', 'view', null),
(5, 2, '2019-07-03', 'report', 'racism'),
(5, 5, '2019-07-03', 'view', null),
(5, 5, '2019-07-03', 'report', 'racism');


insert into removals values
(2,'2019-07-20'),(3,'2019-07-18');
```

```
select
      round(avg(daily_count), 2) as average_daily_percent
from
      (select count(distinct b.post_id)/count(distinct a.post_id)*100 as
       daily_count
from
      actions a
left join
```

```
      removals b
on
      a.post_id = b.post_id
where
      extra = 'spam'
group by
      action_date
) b;
```

Question 74:

****Same as 43 in Set 1**

Question 75:

****Same as 43 in Set 1**

Question 76:

```
create table Salaries(
company_id int, employee_id int,employee_name varchar(20), salary int);

insert into Salaries values
(1,1, 'Tony', 2000),
(1,2, 'Pronub', 21300),
(1,3, 'Tyrrox', 10800),
(2,1, 'Pam', 300),
(2,7, 'Bassem', 450),
(2,9, 'Hermione', 700),
(3,7, 'Bocaben', 100),
(3,2, 'Ognjen', 2200),
(3,13 ,'Nyan Cat', 3300),
(3,15 ,'Morning Cat', 7777);
```

```
with cte as
(select *,max(salary) over(partition by company_id) max_sal from Salaries)
select company_id , employee_id ,employee_name,
round(CASE
 when max_sal < 1000 then salary
 when max_sal >1000 and max_sal <10000 then
    salary - (salary *24/100)
 when max_sal > 10000 then salary -(salary *49/100)
 end) salary
 from cte;
```

Question 77:

****Same as 42 in Set 1**

Question 78:

```
create table Variables
(name varchar(2),value int);

insert into Variables VALUES
('x',66),
('y',77);
create table Expressions
(left_operand varchar(2),operator enum ('<', '>', '=')
,right_operand varchar(2));

insert into Expressions VALUES
('x', '>' ,'y'),
('x', '<' ,'y'),
('x', '=' ,'y'),
('y', '>' ,'x'),
('y', '<' ,'x'),
('x', '=' ,'x');
```

```
select left_operand,operator,right_operand,
case
        when e.operator = '<' then if(v1.value < v2.value,'true','false')
        when e.operator = '>' then if(v1.value > v2.value,'true','false')
        else if(v1.value = v2.value,'true','false')
    end as value
from Expressions e
left join Variables v1 on e.left_operand = v1.name
left join Variables v2 on e.right_operand = v2.name ;
```

Question 79:

****Same as 35 in Set 1**

Question 80:

****Same as 55 in Set 2**

Question 81:

```
create table students (id int, name varchar(15),marks int);

insert into students VALUES
```

```
(1,'Ashley',81),
(2,'Samantha',75),
(3,'Julia',76),
(4,'Belvet',84);
```

```
# Approach 1:
select name from students where marks > 75 order by substr(name, -3), id ;

# Approach 2:
select name from students where marks>75 order by
regexp_substr(name,'\w{3}$'),id;

# Approach 3:
SELECT name FROM students WHERE marks > 75 ORDER BY SUBSTR(name, -3), ID ASC;
```

Question 82:

```
create table employee(employee_id int,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;
```

Question 83:

```
SELECT name FROM employee WHERE salary >2000 AND months <10 ORDER BY employee_id;
```

Question 84:

```sql
create table triangles (A int , B int , C int);

insert into triangles values (20,20,23),(20,20,20),
(20,21,22),(13,14,30);
```

```sql
SELECT
  CASE
    WHEN A + B <= C or A + C <= B or B + C <= A THEN 'Not A Triangle'
    WHEN A = B and B = C THEN 'Equilateral'
    WHEN A = B or A = C or B = C THEN 'Isosceles'
    WHEN A <> B and B <> C THEN 'Scalene'
  END tuple
FROM triangles;
```

Question 85:

```sql
create table user_transactions
(transaction_id int,
product_id int,
spend float,
transaction_date datetime
);

insert into user_transactions values
(1341,123424,1500.60,'2019-12-31 12:00:00'),
(1423,123424,1000.20,'2019-12-31 12:00:00'),
(1623,123424,1246.44,'2019-12-31 12:00:00'),
(1322,123424,2145.32,'2019-12-31 12:00:00');
```

Step 1:

First, we need to obtain the year by using EXTRACT on the transaction date as written in the code below.

Step 2:

Next, we convert the query in step 1 into a CTE called yearly_spend.
With this CTE, we then calculate the prior year's spend for each product.
We can do so by applying LAG function onto each year and partitioning by product id to calculate the prior year's spend for the given product.

Step 3:

Finally, we wrap the query above in another CTE called yearly_variance.
Year-on-Year Growth Rate = (Current Year's Spend - Prior Year's Spend) / Prior Year's Spend x 100

In the final query, we apply the y-o-y growth rate formula and round the results to 2 nearest decimal places.

```
WITH yearly_spend AS (
  SELECT
    EXTRACT(YEAR FROM transaction_date) AS year,
    product_id,
    spend AS curr_year_spend
  FROM user_transactions
),
yearly_variance AS (
  SELECT
    *,
    LAG(curr_year_spend, 1) OVER (
      PARTITION BY product_id
      ORDER BY product_id, year) AS prev_year_spend
  FROM yearly_spend)

SELECT
  year,
  product_id,
  curr_year_spend,
  prev_year_spend,
  ROUND(100 * (curr_year_spend - prev_year_spend)/ prev_year_spend,2) AS yoy_rate
FROM yearly_variance;
```

Question 86:

```
create table inventory
(item_id int,
item_type varchar(20),
item_category varchar(20),
square_footage float);


insert into inventory values
(1374,'prime_eligible','mini refrigerator',68.00),
(4245,'not_prime','standing lamp',26.40),
(2452,'prime_eligible','television',85.00),
(3255,'not_prime','side table',22.60),
(1672,'prime_eligible','laptop',8.50);
```

Step 1:

First, we build a summary table of the necessary fields: item type ('prime_eligible', 'not_prime'), then sum the square footage and count of items.

Step 2:
After converting the above query into a CTE called summary, we find the number of times that we can stock the prime items in the warehouse space, since those have the highest priority.
We accomplish this by dividing 500,000 (the square-footage of the space) by the total square footage of prime items, and truncating the result to 0 decimal places using TRUNCATE.

We can stock 900 times of prime eligible items which can be mathematically expressed as:
900 times x 555.20 sq ft = 499,680 sq ft
Hence, the remaining warehouse space is 500,000 sq ft - 499,680 sq ft = 320 sq ft.

Step 3:
Next, we convert the previous query into a second CTE called prime_items, and write a similar query to find the number of times that we can stock the non-prime items to fill the remaining 320 sq ft.
Note that we use the sub-select SELECT prime_item_count * total_sqft FROM prime_items here to reference the square footage already taken up (~499,80 sq ft) by prime-eligible items.

Step 4:
Now that we have the number of prime and non-prime items in two separate tables, we can combine them using the UNION ALL operator. Note that UNION ALL only works when the two SELECT statements have the same number of result fields with similar data types.

```
WITH summary AS (
  SELECT
    item_type,
    SUM(square_footage) AS total_sqft,
    COUNT(*) AS item_count
  FROM inventory
  GROUP BY item_type
),
prime_items AS (
  SELECT
    DISTINCT item_type,
    total_sqft,
    TRUNCATE(500000/total_sqft,0) AS prime_item_combo,
    (TRUNCATE(500000/total_sqft,0) * item_count) AS prime_item_count
  FROM summary
  WHERE item_type = 'prime_eligible'
),
non_prime_items AS (
  SELECT
    DISTINCT item_type,
    total_sqft,
    TRUNCATE(
      (500000 - (SELECT prime_item_combo * total_sqft FROM prime_items))
      / total_sqft, 0) * item_count AS non_prime_item_count
```

```
  FROM summary
  WHERE item_type = 'not_prime')
SELECT
  item_type,
  prime_item_count AS item_count
FROM prime_items
UNION ALL
SELECT
  item_type,
  non_prime_item_count AS item_count
FROM non_prime_items;
```

Question 87:

```
create table user_actions (user_id int,event_id int,
event_type enum('sign-in','like','comment'),
event_date datetime);

insert into user_actions values
(445,7765,'sign-in','2022-05-31 12:00:00'),
(742,6458,'sign-in','2022-05-03 12:00:00'),
(445,3634,'like','2022-05-05 12:00:00'),
(742,1374,'comment','2022-05-05 12:00:00'),
(648,3124,'like','2022-06-18 12:00:00');
```

We can use a 2-step approach:
Find users who were active in the last month.
Find users who are active in the current month which exist in the last month.

Step 1
Let's start with finding users who were active in the last month.
In FROM, we alias the user_actions table as last_month to say that "Hey, this table contains last month's user information".
In WHERE clause, we match users in the last month's table to the current month's table based on user id.
EXTRACT (MONTH FROM last_month.event_date) = EXTRACT (MONTH FROM curr_month.event_date - 1) means a particular user from last month exists in the current month.
Hence, this user is an active user which is what we're finding for. "-1" means to subtracts one month from current month's date to give us last month's date.

Step 2
Using EXISTS operator, we find for users in the current month which also exists in the subquery, which represents active users in the last month (in step 1).

Note that the user_actions table is aliased as curr_month (to say "Hey, this is current month's user information!").
To bucket days into its month, we extract the month information by using EXTRACT.
Then, we use a COUNT DISTINCT over user id to obtain the monthly active user count for the month.

```
SELECT
  EXTRACT(MONTH FROM curr_month.event_date) AS mth,
  COUNT(DISTINCT curr_month.user_id) AS monthly_active_users
FROM user_actions AS curr_month
WHERE EXISTS (
  SELECT last_month.user_id
  FROM user_actions AS last_month
  WHERE last_month.user_id = curr_month.user_id
    AND EXTRACT(MONTH FROM last_month.event_date) =
    EXTRACT(MONTH FROM curr_month.event_date -  1)
)
  AND EXTRACT(MONTH FROM curr_month.event_date) = 7
  AND EXTRACT(YEAR FROM curr_month.event_date) = 2022
GROUP BY EXTRACT(MONTH FROM curr_month.event_date);
```

Question 88:

```
create table search_frequency
(searches int,
num_users int);

insert into search_frequency VALUES
(1,2),
(2,2),
(3,3),(4,1);
```

```
with recursive t1 as
  (select searches
          , 1 as search_seq
          , num_users
  from search_frequency
  UNION ALL
  SELECT searches
          , search_seq+1
          , num_users
  FROM t1
  where search_seq < num_users),
t2 as (SELECT *
        , row_number() over(order by searches) as search_order
```

```
        , count(searches) over() as total_obs
  from t1)
SELECT ROUND(AVG(searches),1) as median from t2
where search_order BETWEEN (total_obs*1.0/2) and (total_obs*1.0/2)+1  ;
```

Question 89:

```
create table advertiser (user_id varchar(20),status varchar(20));

insert into advertiser values ('bing','NEW'),('yahoo','NEW'),
('alibaba','EXISTING');

create table daily_pay (user_id varchar(20), paid float);

insert into daily_pay values ('yahoo',45.00),('alibaba',100.00),('target',13.00);
```

Step 1:
First, we merge the advertiser and daily_pay tables with LEFT JOIN.

Step 2:
To ensure that we have a complete table with all the existing and new advertisers, we have to merge both tables vertically.
After this, we LEFT JOIN daily_pay with advertiser. This will let us know which users are new.
At last, we join the results produced by these two left joins using UNION.
If paid is null, then that user have not paid at day t and if status is null then the user is a new user(who's information is not present in advertiser table but that user paid on day t, so this particular user is called as new user)

Step 3:
Our final step is to assign each advertiser to its new status using a conditional CASE statement based on the payment conditions set out:
New: users registered and made their first payment.
Existing: users who paid previously and recently made a current payment.
Churn: users who paid previously, but have yet to make any recent payment.
Resurrect: users who did not pay recently but may have made a previous payment and have made payment again recently.

```
WITH payment_status AS (
SELECT
  advertiser.user_id,
  advertiser.status,
  payment.paid
FROM advertiser
LEFT JOIN daily_pay AS payment
```

```
  ON advertiser.user_id = payment.user_id
UNION
SELECT
  payment.user_id,
  advertiser.status,
  payment.paid
FROM daily_pay AS payment
LEFT JOIN advertiser
  ON advertiser.user_id = payment.user_id
)
SELECT
  user_id,
  CASE WHEN paid IS NULL THEN 'CHURN'
    WHEN status != 'CHURN' AND paid IS NOT NULL THEN 'EXISTING'
    WHEN status = 'CHURN' AND paid IS NOT NULL THEN 'RESURRECT'
    WHEN status IS NULL THEN 'NEW'
  END AS new_status
FROM payment_status
ORDER BY user_id;
```

Question 90:

```
create table server_utilization (server_id int,
status_time timestamp,session_status varchar(20));

insert into server_utilization values
(1,'2022-08-02 10:00:00','start'),
(1,'2022-08-04 10:00:00','stop'),
(2,'2022-08-17 10:00:00','start'),
(2,'2022-08-24 10:00:00','stop');
```

```
with cte as(
select server_id,status_time start_time,lead(status_time,1)
over(partition by server_id ) end_time
from server_utilization)

select sum(datediff(end_time,start_time)+1) total_uptime_days from cte;
```

Question 91:

```
create table transactions
(transaction_id int,
merchant_id int,
credit_card_id int,
```

```
amount int,
transaction_timestamp datetime
);

insert into transactions values
(1,101,1,100,'2022-09-25 12:00:00'),
(2,101,1,100,'2022-09-25 12:08:00'),
(3,101,1,100,'2022-09-25 12:28:00'),
(4,102,2,300,'2022-09-25 12:00:00'),
(6,102,2,400,'2022-09-25 14:00:00');
```

Step 1:
For each transaction present in the transactions table, we will obtain the time of the most recent identical transaction.
With the LAG window function, it is feasible.
This function accesses the specified field's values from the previous rows.

Let's interpret the LAG function:

PARTITION BY clause separates the result's rows by the unique merchant ID, credit card and payment.
Each partition is applied with a separate application of the LAG function, and the computation is restarted for each partition (merchant ID, credit card and payment).
Rows in each partition are sorted based on transaction_timestamp in the ORDER BY clause.

Step 2:
Next, we should evaluate the difference in time between two consecutive identical transactions.

Step 3:
The last thing we need to do is to gather all the identical transactions which occurred within a 10-minute window.
To do that, we must first convert the query into a common table expression (CTE). Then, we will filter the records using the WHERE clause for transactions a 10-minute or lesser window.

```
WITH CTE AS(
SELECT *,
      (total_minutes - lag(total_minutes)
      OVER(PARTITION BY merchant_id,credit_card_id,amount
      ORDER BY transaction_timestamp)) AS time_less_10
FROM
    (SELECT *,
    (EXTRACT(hour FROM transaction_timestamp)*60 +
    EXTRACT(minute FROM transaction_timestamp) +
    EXTRACT(second FROM transaction_timestamp)) AS total_minutes
FROM transactions) AS temp_table1)
```

```
SELECT COUNT(*) AS payment_count FROM CTE WHERE time_less_10 <=10;
```

Question 92:

```
create table orders
(
order_id int,
customer_id int,
trip_id int,
status enum('completed successfully', 'completed incorrectly', 'never received'),
order_timestamp timestamp);

insert into orders values
(727424, 8472, 100463, 'completed successfully','2022-06-05 09:12:00'),
(242513, 2341, 100482, 'completed incorrectly','2022-06-05 14:40:00'),
(141367, 1314, 100362, 'completed incorrectly','2022-06-07 15:03:00'),
(582193, 5421, 100657, 'never received','2022-07-07 15:22:00'),
(253613, 1314, 100213, 'completed successfully','2022-06-12 13:43:00');

create table trips (dasher_id int,
trip_id int,
estimated_delivery_timestamp timestamp,
actual_delivery_timestamp timestamp);

insert into trips values
(101, 100463, '2022-06-05 09:42:00', '2022-06-05 09:38:00'),
(102, 100482, '2022-06-05 15:10:00', '2022-06-05 15:46:00'),
(101, 100362, '2022-06-07 15:33:00', '2022-06-07 16:45:00'),
(102, 100657, '2022-07-07 15:52:00', null),
(103, 100213, '2022-06-12 14:13:00', '2022-06-12 14:10:00');


create table customers
(customer_id int,
signup_timestamp timestamp);

insert into customers values
(8472, '2022-05-30 00:00:00'),
(2341, '2022-06-01 00:00:00'),
(1314, '2022-06-03 00:00:00'),
(1435, '2022-06-05 00:00:00'),
(5421, '2022-06-07 00:00:00');
```

```sql
WITH Order_1 AS (
SELECT
  orders.order_id,
  orders.trip_id,
  orders.status
FROM customers
INNER JOIN orders
  ON customers.customer_id = orders.customer_id
WHERE EXTRACT(MONTH FROM customers.signup_timestamp) = 6
  AND EXTRACT(YEAR FROM customers.signup_timestamp) = 2022
  AND customers.signup_timestamp +  14  > orders.order_timestamp)
SELECT ROUND(
  100.0 *
    COUNT(order_2.order_id)/
    (SELECT COUNT(order_id) FROM Order_1),2) AS bad_experience_pct
FROM Order_1 AS order_2
INNER JOIN trips
  ON order_2.trip_id = trips.trip_id
WHERE order_2.status IN ('completed incorrectly', 'never received')
  OR trips.actual_delivery_timestamp IS NULL
  AND trips.estimated_delivery_timestamp > trips.actual_delivery_timestamp;
```

Question 93:

****Same as 68 set 2**

Question 94:

**** Same as 55 set 2**

Question 95:

```sql
create table numbers
(num int primary key,
frequency int
);

insert into numbers values (0,7),(1,1),(2,3),(3,1);
```

```sql
select  ROUND((avg(n.num)),2) as median
from numbers n
where n.frequency >= abs((select sum(frequency) from numbers where num<=n.num) -
                         (select sum(frequency) from numbers where num>=n.num));
```

Question 96:

```
create table salary
(id int primary key,
employee_id int,
amount int,
pay_date date);

create table employee
(employee_id int primary key,
department_id int);

insert into salary VALUES
(1, 1, 9000, '2017/03/31'),(2, 2, 6000, '2017/03/31'),
(3, 3, 10000, '2017/03/31'),(4, 1, 7000, '2017/02/28'),
(5, 2, 6000, '2017/02/28'),(6, 3, 8000, '2017/02/28');


insert into employee values (1,1),(2,2),(3,2);
```

First, calculate the average salary of the company in each month.
Secondly, calculate the average salaries of the departments in each month.
Thirdly, join the two query results using pay_month, and use case-when statement to determine the results.

```
select department_salary.pay_month, department_id,
    case
        when department_avg>company_avg then 'higher'
        when department_avg<company_avg then 'lower'
        else 'same'
    end as comparison
from
    (
      select department_id, avg(amount) as department_avg, date_format(pay_date,
'%Y-%m') as pay_month
      from salary join employee on salary.employee_id = employee.employee_id
      group by department_id, pay_month
    ) as department_salary
join
    (
      select avg(amount) as company_avg,  date_format(pay_date, '%Y-%m') as
pay_month
```

```
        from salary
        group by date_format(pay_date, '%Y-%m')
    ) as company_salary
on department_salary.pay_month = company_salary.pay_month;
```

Question 97:

```
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, '2016-03-02', 6),
(2, 3, '2017-06-25', 1),(3, 1, '2016-03-01', 0),(3, 4, '2016-07-03', 5);
```

Step 1: For install_date, select the minimum date of each player.
A minimum date of a player is a date such that no date is less than the player's date.

Step 2:
For installs, count the number of players that logged in on the day.

Step 3:
For Day1_retention,
count the number of players that logged back in on the day right after the install date.
Use round to round the results to 2 decimal places.

```
select
      a1.event_date as install_dt,
      count(a1.player_id) as installs,
      round(count(a3.player_id) / count(a1.player_id), 2) as Day1_retention
from
      activity a1
left join
      activity a2
on
      a1.player_id = a2.player_id
and
      a1.event_date > a2.event_date
left join
      activity a3
on
      a1.player_id = a3.player_id
and
      datediff(a3.event_date, a1.event_date) = 1
```

```
where
      a2.event_date is null
group by
      a1.event_date;
```

Question 98:

```
create table players (player_id int primary key,group_id int);

create table matches (match_id int primary key,
first_player int, second_player int,
first_score int, second_score int);

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

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);
```

The Algorithm which we follow here is :
step-1: get player-score table
step-2: join to get group id, and order by max-score-first
step-3: group by group
step-4: Use CTE to get the desired result

```
WITH cte AS
 (
     SELECT   Group_ID ,Player_ID ,SUM(Score) AS Score
             ,ROW_NUmber() OVER (Partition BY group_id
             Order by suM(score) desc,player_id) AS RNM
     FROM (
         SELECT  P.group_id ,P.Player_ID ,COALESCE(M.first_score,0) AS Score
         FROM matches M RIGHT JOIN     players P
         ON   M.first_player = P.Player_id
         UNION ALL
         SELECT  P.group_id ,P.Player_ID,COALESCE(M.Second_score,0) AS Score
         FROM  matches M RIGHT JOIN     players P
         ON    M.Second_player = P.Player_id )A
     GROUP BY GROUP_ID, PLAYER_ID
 )
 SELECT GROUP_iD, pLAYER_id AS Winner_ID FROM cte WHERE rnm=1;
```

Question 99:

```
create table students(
student_id int primary key, student_name varchar(20));

create table exam (
exam_id int, student_id int,
score int, primary key (exam_id,student_id));

insert into students values
(1,'Daniel'),(2,'Jade'),(3,'Stella'),
(4,'Jonathan'),(5,'Will');

insert into exam VALUES
(10,1,70),(10,2,80),(10,3,90),
(20,1,80),(30,1,70),(30,3,80),
(30,4,90),(40,1,60),(40,2,70),
(40,4,80);
```

```
select distinct students.*
from students inner join exam
on students.student_id = exam.student_id
where students.student_id not in
    (select e1.student_id
    from exam as e1 inner join
        (select exam_id, min(score) as min_score, max(score) as max_score
        from exam
        group by exam_id) as e2
    on e1.exam_id = e2.exam_id
    where e1.score = e2.min_score or e1.score = e2.max_score)
```

Question 100:

****Same as 99 in set 2**