```
create database set3;

use set3;

101. Write an SQL query to show the second most recent activity of each user.

create table userActivity
(
    username varchar(30),
    activity varchar(30),
    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 *,
row_number() over (partition by username ORDER BY startDate DESC) as rnk,
count(username) over(partition by username ORDER BY username) as cnt
from userActivity) tb1
where (cnt<>1 and rnk=2) or (cnt=1 and rnk=1);



107. Write a query calculating the amount of error (i.e.: actual - miscalculated average monthly salaries),
and round it up to the next integer.

create DATABASE set3;
use set3;
create table employees
(
    id int,
    name varchar(30),
    salary int,
    check (1000<salary < 100000)
);
INSERT into employees VALUES
(1,'Kristeen',1420),
(2,'Ashley',2006),
(3,'Julia',2210),
(4,'Maria',3000);

SELECT round(avg(salary)-avg(REPLACE(salary,'0','')))
from employees;

'
with recursive cte as(
    SELECT 0 as n
    UNION ALL
    select power(10,n+1) as n from cte
    where n < (SELECT len(salary) as max_length from employees)
), next_cte as(
    select b.id, b.name, a.n
    from cte a
    JOIN
    employees b
    on a.
)'


108. Write a query to find the maximum total earnings for all employees as well as the total number of
employees who have maximum total earnings. Then print these values as 2 space-separated
integers

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 concat(max(salary*months),' ',count(name)) as output
from employee
where (salary * months) >= (select max(salary * months) from employee);
```

109. a) Query an alphabetically ordered list of all names in OCCUPATIONS, immediately followed by the first letter of each profession as a parenthetical (i.e.: enclosed in parentheses).

b) Query the *number* of occurrences of each occupation in OCCUPATIONS. Sort the occurrences in ascending order

```
CREATE table occupations
(
    name VARCHAR(30),
    occupation ENUM ('Doctor','Professor','Singer','Actor')
);

insert into occupations VALUES
('Samantha','Doctor'),
('Julia','Actor'),
('Maria','Actor'),
('Meera','Singer'),
('Ashely','Professor'),
('Ketty','Professor'),
('Christeen','Professor'),
('Jane','Actor'),
('Jenny','Doctor'),
('Priya','Singer');


a) select CONCAT(name,'(',substr(occupation,1,1),')') as output
from occupations
ORDER BY name;

b) select CONCAT('There are a total of ',tb1.cnt,' ',tb1.occupation,'s') as output
from (select count(*) cnt,occupation from occupations
GROUP BY occupation
ORDER BY cnt,occupation) tb1;
```

110. Pivot the Occupation column in OCCUPATIONS so that each Name is sorted alphabetically and displayed underneath its corresponding Occupation.

```
create DATABASE set3;
use set3;

CREATE table occupations
(
    name VARCHAR(30),
    occupation ENUM ('Doctor','Professor','Singer','Actor')
);

insert into occupations VALUES
('Samantha','Doctor'),
('Julia','Actor'),
('Maria','Actor'),
('Meera','Singer'),
('Ashely','Professor'),
('Ketty','Professor'),
('Christeen','Professor'),
('Jane','Actor'),
('Jenny','Doctor'),
('Priya','Singer');

select name,occupation,
dense_rank() over(ORDER BY occupation) rnk
from occupations;
```

111. Write a query to find the node type of Binary Tree ordered by the value of the node. Output one of the following for each node:
- Root: If node is root node.
- Leaf: If node is leaf node.
- Inner: If node is neither root nor leaf node.

```
create table 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 p is NULL then 'Root'
    when n in (select distinct p from bst) then 'Inner'
    else 'Leaf'
end as type
from bst;
```

113. Write a query to print all prime numbers less than or equal to 1000.

```
with recursive nums as
(
  select 2 as n
  union
  select n+1 from nums where n<1000
)
SELECT GROUP_CONCAT(n SEPARATOR '&') FROM nums
WHERE n NOT IN (
  SELECT n
  FROM nums
  JOIN ( SELECT n AS n2 FROM nums ) tb1
  WHERE n2 <  n
    AND n % n2 = 0
  ORDER BY n
);
```

114. The following pattern represents P(5):

```
*
* *
* * *
* * * *
* * * * *
```

Write a query to print the pattern P(20).

```
with recursive pattern as(
    select 1 as n
    union
    select n+1 from pattern where n<20
)
select repeat('*',n) from pattern;
```

115. P(R) represents a pattern drawn by Julia in R rows. The following pattern represents P(5):

```
* * * * *
* * * *
* * *
* *
*
```

Write a query to print the pattern P(20).

```
with recursive pattern as(
    select 20 as n
    union
    select n-1 from pattern where n>1
```

```
)
select repeat('*',n) from pattern;
```

150. Write a query to output all such symmetric pairs in ascending order by the value of X. List the rows such that X1 ≤ Y1.

```
create table students
(
    id INT,
    name VARCHAR(30)
);

create table friends
(
    id INT,
    friend_id INT
);

create table packages
(
    id INT,
    salary FLOAT
);

INSERT INTO students VALUES
(1,'Ashley'),
(2,'Samantha'),
(3,'Julia'),
(4,'Scarlet');

INSERT into friends VALUES
(1,2),
(2,3),
(3,4),
(4,1);

INSERT into packages VALUES
(1,15.2),
(2,10.06),
(3,11.55),
(4,12.12);

select name from
(SELECT s1.ID, s1.Name, p1.Salary, f.Friend_ID, s2.name as friend_name, p2.Salary as friend_salary
FROM students s1
JOIN packages p1 ON s1.ID = p1.ID
JOIN friends f ON s1.ID = f.ID
JOIN students s2 ON f.Friend_ID = s2.ID
JOIN packages p2 ON f.Friend_ID = p2.ID) tb1
where tb1.salary<tb1.friend_salary
ORDER BY tb1.friend_salary;
```

151. Write a query to print the respective hacker_id and name of hackers who achieved full scores for more than one challenge.

```
CREATE table hackers
(
    hacker_id INT,
    name VARCHAR(30)
);

create table difficulty
(
    difficulty_level INT,
    score int
);

create table challenges
(
    challenge_id int,
    hacker_id int,
    difficulty_level INT
);

create TABLE submission
(
```

```
    submission_id INT,
    hacker_id INT,
    challenge_id int,
    score INT
);

insert into hackers VALUES
(5580,'Rose'),
(8439,'Angela'),
(27205,'Frank'),
(52243,'Patrick'),
(52348,'Lisa'),
(57645,'Kimberly'),
(77726,'Bonnie'),
(83082,'Michael'),
(86870,'Todd'),
(90411,'Joe');

INSERT into difficulty VALUES
(1,20),
(2,30),
(3,40),
(4,60),
(5,80),
(6,100),
(7,120);

INSERT into challenges VALUES
(4810,77726,4),
(21089,27205,1),
(36566,5580,7),
(66730,52243,6),
(71055,52243,2);

INSERT into submission VALUES
(68628,77726,36566,30),
(65300,77726,21089,10),
(40326,52243,36566,77),
(8941,27205,4810,4),
(83554,77726,66730,30),
(43353,52243,66730,0),
(55385,52348,71055,20),
(39784,27205,71055,23),
(94613,86870,71055,30),
(45788,52348,36566,0),
(93058,86870,36566,30),
(7344,8439,66730,92),
(2721,8439,4810,36),
(523,5580,71055,4),
(49105,52348,66730,0),
(55877,57645,66730,80),
(38355,27205,66730,35),
(3924,8439,36566,80),
(97397,90411,66730,100),
(84162,83082,4810,40),
(97431,90411,71055,30);

select s.hacker_id, name
from submission s
JOIN
hackers h ON s.hacker_id = h.hacker_id
JOIN
challenges c ON s.challenge_id = c.challenge_id
JOIN
difficulty d ON c.difficulty_level = d.difficulty_level
where s.score = d.score
GROUP BY name, s.hacker_id
HAVING count(s.challenge_id) > 1
ORDER BY count(s.challenge_id) DESC, s.hacker_id;
```

152. Write a query to output the start and end dates of projects listed by the *number* of days it took to complete the project in ascending order. If there is more than one project that have the same *number* of completion days, then order by the start *date* of the project.

```
create table projects
(
    task_id INT,
    start_date DATE,
    end_date DATE
```

```
);

INSERT into projects VALUES
(1,'2015-10-01','2015-10-02'),
(2,'2015-10-02','2015-10-03'),
(3,'2015-10-03','2015-10-04'),
(4,'2015-10-13','2015-10-14'),
(5,'2015-10-14','2015-10-15'),
(6,'2015-10-28','2015-10-29'),
(7,'2015-10-30','2015-10-31');

-- INCOMPLETE DATA (there is supposed to be a table with task_id belonging to some project_id)
```

153. List the user IDs who have gone on at least 1 shopping spree in ascending order.

```
create table transactions
(
    user_id integer,
    amount float,
    transaction_date timestamp
);

insert into transactions VALUES
(1,9.99,'2022-08-01 10:00:00'),
(1,55,'2022-08-17 10:00:00'),
(2,149.5,'2022-08-05 10:00:00'),
(2,4.89,'2022-08-06 10:00:00'),
(2,34,'2022-08-07 10:00:00');

select distinct user_id from(select user_id,
count(*) over(partition by user_id) cnt
from transactions) tb1
where tb1.cnt>=3;
```

154. Write a query to find the number of two-way unique relationships in this data.

```
create table payments
(
    payer_id integer,
    recipient_id integer,
    amount integer
);

insert into payments VALUES
(101,201,30),
(201,101,10),
(101,301,20),
(301,101,80),
(201,301,70);

select distinct *
FROM(select count(payer_id) unique_relationships
from payments
GROUP BY (payer_id+recipient_id)) tb1
where unique_relationships>1;
```

156. Write a query to obtain the list of customers whose first transaction was valued at $50 or more.

```
create table user_transactions
(
    transaction_id integer,
    user_id integer,
    spend FLOAT,
    transaction_date timestamp
);

INSERT into user_transactions VALUES
(759274,111,49.50,'2022-02-03 00:00:00'),
(850371,111,51.00,'2022-03-15 00:00:00'),
(615348,145,36.30,'2022-03-22 00:00:00'),
(137424,156,151.00,'2022-04-04 00:00:00'),
(248475,156,87.00,'2022-04-16 00:00:00');
```

```
select count(user_id) users
from(SELECT user_id,spend,
rank() over(partition by user_id ORDER BY transaction_date) rnk
FROM user_transactions) tb1
where tb1.rnk=1 and tb1.spend>=50;
```

157. Write a query to obtain the sum of the odd-numbered and even-numbered measurements on a particular day, in two different columns.

```
create table measurements
(
    measurement_id integer,
    measurement_value FLOAT,
    measurement_time datetime
);

INSERT into measurements VALUES
(131233,1109.51,'2022-07-10 09:00:00'),
(135211,1662.74,'2022-07-10 11:00:00'),
(523542,1246.24,'2022-07-10 13:15:00'),
(143562,1124.50,'2022-07-11 15:00:00'),
(346462,1234.14,'2022-07-11 16:45:00');

with cte as(
    select measurement_value,measurement_time,
    rank() over(partition by day(measurement_time) ORDER BY measurement_time) rnk
    from measurements
)


select tb2.measurement_time,tb1.odd_sum,tb2.even_sum
FROM(SELECT day(measurement_time) measure_time,
round(sum(measurement_value),2) odd_sum
from cte
WHERE rnk%2 <> 0
GROUP BY measure_time) tb1
JOIN
(SELECT concat(YEAR(measurement_time),'-', MONTH(measurement_time),'-', DAY(measurement_time)) measurement_time,
round(sum(measurement_value),2) even_sum
from cte
WHERE rnk%2 = 0
GROUP BY measurement_time) tb2
on tb1.measure_time = day(tb2.measurement_time);
```

172.

```
create table personal_profile
(
    profile_id integer,
    name varchar(30),
    followers integer
);

insert into personal_profile VALUES
(1,'Nick Singh',92000),
(2,'Zach Wilson',199000),
(3,'Daliana Liu',171000),
(4,'Ravit Jain',107000),
(5,'Vin Vashishta',139000),
(6,'Susan Wojcicki',39000);

create table employee_company
(
    personal_profile_id integer,
    company_id integer
);

insert into employee_company VALUES
(1,4),
(1,9),
(2,2),
(3,1),
(4,3),
(5,6),
(6,5);

create table company_pages
```

```
(
    company_id integer,
    name varchar(30),
    followers integer
);


insert into company_pages VALUES
(1,'The Data Science Podcast',8000),
(2,'Airbnb',700000),
(3,'The Ravit Show',6000),
(4,'DataLemur',200),
(5,'YouTube',16000000),
(6,'DataScience.Vin',4500),
(9,'Ace The Data Science Interview',4479);


select distinct tb1.profile_id
from (select pp.profile_id,pp.followers p_follower,
sum(cp.followers) over (partition by pp.followers) tot_cp
from personal_profile pp
JOIN
employee_company ec
on pp.profile_id = ec.personal_profile_id
join
company_pages cp
on ec.company_id = cp.company_id) tb1
WHERE tb1.tot_cp<tb1.p_follower
ORDER BY tb1.profile_id;
```