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

Solution 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);


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.

Solution 2:

```
select NAME from city
where COUNTRYCODE = "USA" and POPULATION > 120000;
```

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

Solution 3:

```
select * from city;
```

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

Solution 4:

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

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

Solution 5:

```
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.

Solution 6:

```
select NAME from city

where COUNTRYCODE = "JPN";
```

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

Solution 7:

```
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.

Solution 8:

```
SELECT DISTINCT CITY ,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.

Solution 9:

```
SELECT (COUNT(CITY)-COUNT(DISTINCT CITY)) AS DIFFE

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.\

Solution 10:

```
select CITY as shortest_city,LENGTH(CITY) from STATION order by Length(CITY) asc, CITY limit 1;

select CITY as longest_city,LENGTH(CITY) from STATION order by Length(CITY) desc, CITY 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.

Solution 11:

```
SELECT DISTINCT city

FROM   station

WHERE   city LIKE 'A%'

     OR  city LIKE 'E%'
```

```
    OR  city LIKE 'I%'
    OR  city LIKE 'O%'
    OR  city LIKE 'U%';
```

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

Solution 12:

```
SELECT DISTINCT city
FROM   station
WHERE   city LIKE '%A'
    OR  city LIKE '%E'
    OR  city LIKE '%I'
    OR  city LIKE '%O'
    OR  city LIKE '%U';
```

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

Solution 13:

```
SELECT DISTINCT city
FROM   station
WHERE  NOT city LIKE 'A%'
    AND NOT city LIKE 'E%'
    AND NOT city LIKE 'I%'
    AND NOT city LIKE 'O%'
    AND NOT city LIKE 'U%';
```

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

Solution 14:

```
SELECT DISTINCT city
FROM   station
WHERE  NOT city LIKE '%A'
    AND NOT city LIKE '%E'
    AND NOT city LIKE '%I'
    AND NOT city LIKE '%O'
```

```
    AND NOT city LIKE '%U';
```

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.

Solution 15:

```
SELECT DISTINCT CITY

FROM STATION

WHERE (CITY NOT IN (SELECT DISTINCT CITY FROM STATION WHERE CITY LIKE '%a' OR CITY LIKE '%e' OR CITY LIKE '%i' OR CITY LIKE '%o' OR CITY LIKE '%u'))

OR

(CITY NOT IN (SELECT CITY FROM STATION WHERE CITY LIKE 'A%' OR CITY LIKE 'E%' OR CITY LIKE 'I%' OR CITY LIKE 'O%' OR CITY LIKE 'U%'));
```

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.

Solution 16:

```
SELECT DISTINCT city

FROM   station

WHERE  NOT city LIKE 'A%'

    AND NOT city LIKE 'E%'

    AND NOT city LIKE 'I%'

    AND NOT city LIKE 'O%'

    AND NOT city LIKE 'U%'

    AND NOT city LIKE '%A'

    AND NOT city LIKE '%E'

    AND NOT city LIKE '%I'

    AND NOT city LIKE '%O'

    AND NOT city LIKE '%U';
```

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.

Solution 17:

```
create table product(product_id int NOT NULL,

product_name varchar(10),

unit_price int,
```

```
primary key (product_id));

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);

select Product.product_id, product_name
from Product left join Sales
on Product.product_id = Sales.product_id
group by product_id
having min(sale_date) >= '2019-01-01' and max(sale_date) <= '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.

Solution 18:

```
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 distinct author_id
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.

Solution 19:

```
create table delivery
(delivery_id int NOT NULL,
customer_id int,
order_date date,
customer_pref_delivery_date date,
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 round(
    (select count(*) from Delivery where order_date = customer_pref_delivery_date)
    /(count(delivery_id)) * 100, 2)as immediate_percentage from Delivery;
```

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.

Solution 20.

```
create table ad
(
ad_id int NOT NULL,
user_id int NOT NULL,
action enum("Clicked", "Viewed", "Ignored"),
PRIMARY KEY (ad_id, user_id));

insert into ad 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 ad_id,
ifnull(
   round(
      avg(
```

```
        case
            when action = "Clicked" then 1
            when action = "Viewed" then 0
            else null
            end) * 100,2),0)
as ctr
from Ad
group by ad_id
order by ctr desc, ad_id asc;
```

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

Solution 21:

```
create table 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 e.*from
(select employee_id,count(team_id)over(partition by team_id order  by team_id desc)
 as team_size from employee)e
order by  e.employee_id asc;
```

Q22. Write an SQL query to find the type of weather in each country for November 2019.

Solution 22:

```
create table countries
```

```
(country_id int NOT NULL,
country_name varchar(30),
PRIMARY KEY (country_id));

create table weather
(country_id int NOT NULL,
weather_state int,
day date NOT NULL ,
PRIMARY KEY (country_id,day));

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

insert into weather value
(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 distinct 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 (select * from countries)c inner join
(select * from weather)w on
c.country_id=w.country_id
where c.country_name in('USA','Australia','Peru','Morocco','China')
group by c.country_name;
```

Q23. Write an SQL query to find the average selling price for each product. average_price should be rounded to 2 decimal places.

Solution 23:

```
create table prices
(product_id int NOT NULL,
start_date date NOT NULL,
end_date date NOT NULL,
price int,
PRIMARY KEY (product_id,start_date,end_date));

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

```
insert into prices value
(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 product_id,round(sum(prices_sum) / sum(units), 2) as average_price
   from (
      select p.product_id as product_id, units, price * units as prices_sum
         from Prices p left join UnitsSold u
         on p.product_id = u.product_id and purchase_date between start_date and end_date
   ) as temp_table
   group by product_id;
```

Q24. Write an SQL query to report the first login date for each player.

Solution 24:

```
create table activity
(player_id int NOT NULL,
device_id int,
event_date date NOT NULL,
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 a.player_id,a.event_date

from(select * , rank()over(partition by player_id order by event_date) as first_login from activity)a

where a.first_login=1;
```

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

Solution 25:

```
select a.player_id

,a.device_id

from(select *,row_number()over(partition by player_id order by device_id) from activity)a

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.

Solution 26 :

```
create table Products

(product_id int NOT NULL,

product_name varchar(30),

product_category varchar(30),

PRIMARY KEY (product_id));


create table Orders

(product_id int,

order_date date,

unit int,

FOREIGN KEY (product_id) REFERENCES product(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 a.product_name, sum(unit) as unit

from Products a

left join Orders b

on a.product_id = b.product_id

where b.order_date between '2020-02-01' and '2020-02-29'

group by a.product_id

having sum(unit) >= 100;

Q27. Write an SQL query to find the users who have valid email

Solution 27 :

create table Users

(user_id int NOT NULL,

```
name varchar(30),
mail varchar(30),
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 user_id,name,mail from  Users where mail not like '%#%' and mail not like '.%' and mail like '%@leetcode.com';
```

Q28. 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.

Solution 28:

```
create table Customers
(customer_id int NOT NULL,
name varchar(30),
country varchar(30),
PRIMARY KEY (customer_id));

create table Product
(product_id int NOT NULL,
name varchar(30),
price int,
PRIMARY KEY (product_id));

create table Orders
(order_id int NOT NULL,
customer_id int,
```

```
product_id int,
order_date date,
quantity int,
PRIMARY KEY (order_id));

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

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

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);

select c.customer_id, c.name
   from Customers c inner join Orders o on c.customer_id = o.customer_id
   inner join Product p on o.product_id = p.product_id
   group by c.customer_id, c.name
   having
```

```
sum(case when left(o.order_date, 7) = '2020-06' then p.price * o.quantity else 0 end) >= 100
and
sum(case when left(o.order_date, 7) = '2020-07' then p.price * o.quantity else 0 end) >= 100;
```

Q29. Write an SQL query to report the distinct titles of the kid-friendly movies streamed in June 2020.

Solution 29:

```
create table TVProgram
(program_date date NOT NULL,
content_id int NOT NULL,
channel varchar(30),
PRIMARY KEY(program_date, content_id));

create table Content
(content_id varchar(30) NOT NULL,
title varchar(30),
Kids_content enum('N','Y'),
content_type varchar(30),
PRIMARY KEY(content_id));

insert into TVProgram values
('2020-06-10',1,'LC-Channel'),
('2020-05-11',2,'LC-Channel'),
('2020-05-12',3,'LC-Channel'),
('2020-05-13',4,'Disney Ch'),
('2020-06-18',4,'Disney Ch'),
('2020-07-15',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 distinct title
from Content
join TVProgram using(content_id)
where kids_content = 'Y'
        and content_type = 'Movies'
        and (month(program_date), year(program_date)) = (6, 2020);
```

Q30. Write an SQL query to find the npv of each query of the Queries table

Solution 30:

```
create table NPV
(id int NOT NULL,
year int NOT NULL,
npv int,
PRIMARY KEY (id,year));

create table Queries
(id int NOT NULL,
year int NOT NULL,
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, ifnull(n.npv,0) as npv

from queries as q

left join npv as n

on (q.id, q.year) = (n.id, n.year);

Q31. Write an SQL query to find the npv of each query of the Queries table

Solution 31:

create table NPV

(id int NOT NULL,

year int NOT NULL,

npv int,

PRIMARY KEY (id,year));

create table Queries

(id int NOT NULL,

year int NOT NULL,

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, ifnull(n.npv,0) as npv

from queries as q

left join npv as n

on (q.id, q.year) = (n.id, n.year);

Q32. 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.

Solution 32:

create table Employees

(id int NOT NULL,

name varchar(30),

PRIMARY KEY (id));

create table EmployeeUNI

(id int NOT NULL ,

unique_id int NOT NULL,

```
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 emp.unique_id,e.name
from Employees e
left join EmployeeUNI emp
on e.id=emp.id;
```

Q33. Write an SQL query to report the distance travelled by each user.

Solution 33:

```
create table Users
(id int NOT NULL,
name varchar(30),
PRIMARY KEY (id));

create table Rides
(id int,
user_id int NOT NULL,
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 name, sum(ifnull((distance),0)) as travelled_distance
from rides r
right join users u
on r.user_id = u.id
group by name
order by travelled_distance desc,name asc;
```

Q34. Write an SQL query to get the names of products that have at least 100 units ordered in February 2020 and their amount.

Solution 34:

```
create table products
(product_id int NOT NULL,
```

```
product_name varchar(30),
product_category varchar(30),
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 a.product_name, sum(unit) as unit
from Products a
left join Orders b
on a.product_id = b.product_id
where b.order_date between '2020-02-01' and '2020-02-29'
group by a.product_id
having sum(unit) >= 100;
```

Q35. 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.

Solution 35:

```
create table Movies
(movie_id int NOT NULL,
title varchar(30),
PRIMARY KEY (movie_id));

create table Users
(user_id int NOT NULL,
name varchar(30),
PRIMARY KEY (user_id));

create table MovieRating
(movie_id int NOT NULL,
user_id int NOT NULL,
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 user_name AS results FROM
(
SELECT a.name AS user_name, COUNT(*) AS counts FROM MovieRating AS b
  INNER JOIN Users AS a
   on a.user_id = b.user_id
   GROUP BY b.user_id
   ORDER BY counts DESC, user_name ASC LIMIT 1
) first_query
UNION
SELECT movie_name AS results FROM
```

(

SELECT c.title AS movie_name, AVG(d.rating) AS rate FROM MovieRating AS d

INNER JOIN Movies AS c

on c.movie_id = d.movie_id

WHERE substr(d.created_at, 1, 7) = '2020-02'

GROUP BY d.movie_id

ORDER BY rate DESC, movie_name ASC LIMIT 1

) second_query;

Q36. Write an SQL query to report the distance travelled by each user.

Solution 36:

select name, sum(ifnull((distance),0)) as travelled_distance

from rides r

right join users u

on r.user_id = u.id

group by name

order by travelled_distance desc,name asc;

Q37. 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.

Solution 37:

select emp.unique_id,e.name

from Employees e

left join EmployeeUNI emp

on e.id=emp.id;

Q38. Write an SQL query to find the id and the name of all students who are enrolled in departments that no longer exist.

Solution: 38

create table Departments

(id int NOT NULL,

name varchar(30),

PRIMARY KEY (id));

create table Students

```
(id int NOT NULL,
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 s.id,s.name from
Departments d right join
Students s on
d.id=s.department_id
where s.department_id in(14, 33, 74,77);
```

Q39. 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.

Solution 39:

```
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 from_id as person1,to_id as person2,
   count(duration) as call_count, sum(duration) as total_duration
from (select *
     from Calls
     union all
     select to_id, from_id, duration
     from Calls) t1
where from_id < to_id
group by person1, person2;
```

Q40. Write an SQL query to find the average selling price for each product. average_price should be rounded to 2 decimal places

Solution 40:

```
create table prices
(product_id int NOT NULL,
start_date date NOT NULL,
end_date date NOT NULL,
price int,
PRIMARY KEY (product_id,start_date,end_date));
```

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



insert into prices value
(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 product_id,round(sum(prices_sum) / sum(units), 2) as average_price
   from (
      select p.product_id as product_id, units, price * units as prices_sum
         from Prices p left join UnitsSold u
         on p.product_id = u.product_id and purchase_date between start_date and end_date
   ) as temp_table
   group by product_id;
```

Q41. Write an SQL query to report the number of cubic feet of volume the inventory occupies in each warehouse

Solution 41:

```
create table Warehouse
(name varchar(30),
```

```
product_id int NOT NULL,
units int,
PRIMARY KEY(name, product_id));

 create table Products2
(product_id int NOT NULL,
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 Products2 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 name as warehouse_name, sum(units * vol) as volume
from Warehouse w
join (select product_id, Width*Length*Height as vol
    from Products2) p
on w.product_id = p.product_id
group by name;
```

Q42. Write an SQL query to report the difference between the number of apples and oranges sold each day. Return the result table ordered by sale_date

Solution 42:

```
create table Sales
(sale_date date NOT NULL,
fruit enum('apples','oranges') NOT NULL,
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 sale_date,
coalesce(sum(case when fruit='apples'then sold_num end),0)-
coalesce(sum(case when fruit='oranges' then sold_num end),0)diff
from sales
group by sale_date;
```

Q43. 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.

Solution 43:

```
create table Activity1
(player_id int NOT NULL,
device_id int,
event_date date NOT NULL,
games_played int,
```

```
PRIMARY KEY (player_id, event_date));

insert into Activity1 values
(1,2,'2016-03-01',5),
(1,2,'2016-03-02',6),
(2,3,'2017-06-25',1),
(3,1,'2016-03-02',0),
(3,4,'2018-07-03',5);

select round(count(distinct device_id)/sum(distinct player_id),2) as fraction from Activity1
where games_played=0;
```

Q44. Write an SQL query to report the managers with at least five direct reports.

Solution 44:

```
create table Employee
(id int NOT NULL,
name varchar(30),
department varchar(30),
managerId int default null,
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 name from Employee
having count(managerId)>=5;
```

Q45. 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). Return the result table ordered by student_number in descending order. In case of a tie, order them by dept_name alphabetically

Solution 45:

```
create table Department
(dept_id int NOT NULL,
dept_name varchar(30),
PRIMARY KEY (dept_id));

create table Student
(student_id int NOT NULL,
student_name varchar(30),
gender varchar(30),
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,coalesce(count(s.student_id),0) student_number from Department d
left join Student s on
(s.dept_id=d.dept_id)
group by d.dept_name
```

```
order by student_number desc, d.dept_name asc;
```

Q46. Write an SQL query to report the customer ids from the Customer table that bought all the products in the Product table.

Solution 46:

```
create table Product2
(product_key int NOT NULL,
PRIMARY KEY (product_key));

create table Customer
(customer_id int,
product_key int NOT NULL,
FOREIGN KEY (product_key) REFERENCES Product2(product_key));

insert into Product2 values
(5),
(6);

insert into Customer values
(1,5),
(2,6),
(3,5),
(3,6),
(1,6);
SELECT customer_id
FROM Customer
GROUP BY customer_id
HAVING COUNT( DISTINCT product_key) = (SELECT COUNT(*) FROM product2);
```

Q47. 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. Return the result table in any order.

Solution 47:

```
create table Employee
(employee_id int NOT NULL,
name varchar(30),
experience_years int,
PRIMARY KEY (employee_id));

create table Project
(project_id int NOT NULL,
employee_id int NOT NULL,
PRIMARY KEY (project_id, employee_id),
FOREIGN KEY (employee_id) REFERENCES Employee3(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
  ) x
WHERE rnk = 1;
```

Q48. 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.

Solution 48:

```
create table Books
(book_id int NOT NULL,
name varchar(30),
available_from date,
PRIMARY KEY (book_id));

create table Orders
(order_id int NOT NULL,
book_id int NOT NULL,
quantity int,
dispatch_date date,
PRIMARY KEY (order_id),
FOREIGN KEY (book_id) REFERENCES Books(dept_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');
select book_id, name
```

```
from books
where book_id not in (
   select book_id
   from orders
   where (dispatch_date between date_sub('2019-06-23',interval 1 year) and '2019-06-23')
   group by (book_id)
   having sum(quantity) >= 10)
and
   available_from < date_sub('2019-06-23', interval 1 month);
```

Q49. 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. Return the result table ordered by student_id in ascending order.

Solution 49:

```
create table Enrollments
(student_id int NOT NULL,
course_id int NOT NULL,
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 =
```

```
   (select max(grade) as max_grade
   from Enrollments e2
   where e1.student_id = e2.student_id)
group by e1.student_id
order by e1.student_id;
```

Q50. Write an SQL query to find the winner in each group.

Solution 50:

```
create table Teams
(team_id int NOT NULL,
group_id int,
PRIMARY KEY (team_id));

create table Matches
(match_id int NOT NULL,
host_team int,
guest_team int,
host_goals int,
guest_goals int,
PRIMARY KEY (match_id));

select group_id, player_id from (
          select p.group_id, ps.player_id, sum(ps.score) as score
          from Players p,
             (
         select first_player as player_id, first_score as score
         from Matches
         union all
         select second_player, second_score
         from Matches
             ) ps
          where p.player_id = ps.player_id
```

```
        group by ps.player_id
        order by group_id, score desc, player_id
) top_scores
group by group_id;
```

Q51. Write an SQL query to report the name, population, and area of the big countries.

Solution 51:

```
create table world
(name varchar(30) NOT NULL,
continent varchar(30),
area int,
population int,
gdp int,
PRIMARY KEY (name));

SELECT
   name, population, area
FROM
   world
WHERE
   area >= 3000000 OR population >= 25000000;
```

Q52. Write an SQL query to report the names of the customer that are not referred by the customer with id = 2.

Solution 52:

```
create table customer
(id int NOT NULL,
name varchar(30),
referee_id int,
PRIMARY KEY (id));

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

Q53. Write an SQL query to report all customers who never order anything

Solution 53:

```
create table customers
(id int NOT NULL,
name varchar(30),
PRIMARY KEY (id));

create table orders
(id int NOT NULL,
FOREIGN KEY (customerid) REFERENCES customers(id));

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

Q54. Write an SQL query to find the team size of each of the employees.

Solution 54:

```
create table Employee
(employee_id int NOT NULL,
team_id int,
PRIMARY KEY (employee_id));

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

Q55. Write an SQL query to find the countries where this company can invest.

Solution:

```
SELECT
 co.name AS country
```

```
FROM
 person p
 JOIN
    country co
    ON SUBSTRING(phone_number,1,3) = country_code
 JOIN
    calls c
    ON p.id IN (c.caller_id, c.callee_id)
GROUP BY
 co.name
HAVING
 AVG(duration) > (SELECT AVG(duration) FROM calls);
```

Q56.

Solution 56:

```
select
 a.player_id,a.device_id
 from (select player_id,device_id,row_number()over(partition by player_id order by device_id) as num from Activity)a
 where a.num=1
 group by a.player_id;
```

Q57.

Solution 57:

```
 create table Orders2
(order_number int,
customer_number int);

insert into Orders2 values
(1,1),
(2,2),
```

(3,3),

(4,3);

```
select  distinct customer_number from Orders2
group by customer_number
having count(order_number)=2;
```

Q58.

Solution 58:

```
create table Cinema
(seat_id int,
free bool);

insert into Cinema values
(1,1),
(2,0),
(3,1),
(4,1),
(5,1);

select
distinct a.seat_id from cinema a join cinema b on abs(a.seat_id - b.seat_id) = 1 and a.free = true and b.free = true
order by a.seat_id;
```

Q59.

Solution 59:

```
SELECT name
FROM salesperson
WHERE sales_id
```

```
NOT IN (
   SELECT s.sales_id FROM orders o
   INNER JOIN salesperson s ON o.sales_id = s.sales_id
   INNER JOIN company c ON o.com_id = c.com_id
   WHERE c.name = 'RED'
);
```

Q60.

Solution 60:

```
drop table if exists triangle;
Create table If Not Exists triangle (x int, y int, z int);
Truncate table triangle;

insert into triangle (x, y, z) values ('13', '15', '30');
insert into triangle (x, y, z) values ('10', '20', '15');

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

Q61.

Solution 61:

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

Q62.

Solution 62:

```
SELECT actor_id, director_id
FROM ActorDirector
GROUP BY actor_id, director_id
HAVING COUNT(*) >= 3;
```

Q63.

Solution 63:

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

Q64.

Solution 64:

```
select project_id , round(avg(experience_years), 2) as average_years
from project as p
left join employee as e
on p.employee_id = e.employee_id
group by project_id;
```

Q65.

Solution 65:

```
select distinct seller_id
from Sales
group by seller_id
having sum(price) = (
   select sum(price) as max_price
   from Sales
   group by seller_id
```

```
   order by max_price desc
   limit 1);
```

Q66.

Solution 66:

```
select distinct buyer_id from Sales s
join Product p
on p.product_id = s.product_id
where p.product_name = 'S8'
and buyer_id not in
(
select buyer_id from Sales s
   join Product p on p.product_id = s.product_id
   where p.product_name = 'iPhone'
);
```

Q67.

Solution 67:

```
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;
```

Q68.

Solution 68:

```
SELECT
```

```
    gender,
    day,
    SUM(score_points) OVER(PARTITION BY gender ORDER BY day) as total
FROM Scores;
```

Q69.

Solution 69:

```
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;
```

Q70.

Solution 70:

```
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;
```

Q71.

Solution 71:

```
select e3.employee_id from Employees e1, Employees e2, Employees e3
where e1.manager_id = 1 and e2.manager_id = e1.employee_id and e3.manager_id =
e2.employee_id and e3.employee_id != 1;
```

Q72.

Solution 72:

```
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;
```

Q73.

Solution 73:

```
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;
```

Q74.

Solution 74:

```
select round(
    ifnull(
        (
            select count(distinct a.player_id)
            from Activity as a join Activity as b
            on a.player_id = b.player_id and datediff(b.event_date, a.event_date) = 1
            where a.event_date = (
                select min(event_date) from Activity where player_id = a.player_id
            )
        )
        /
```

```
    (   select count(distinct player_id) from Activity   ),
  0),
2)
as fraction;
```

Q75.

Solution 75:

```
 SELECT
round((count(distinct c.player_id) / (select count(distinct player_id) from activity)),2)as fraction
FROM
CTE c
JOIN Activity a
on c.player_id = a.player_id
and datediff(c.event_start_date, a.event_date) = -1
```

Q76.

Solution 76:

```
SELECT
   t1.company_id,
   t1.employee_id,
   t1.employee_name,
   ROUND(CASE WHEN t2.max_sal >= 1000 AND t2.max_sal <= 10000 then salary * 0.76
      WHEN t2.max_sal > 10000 THEN salary * 0.51
      Else salary end, 0) as salary
FROM Salaries as t1 JOIN (SELECT company_id, MAX(salary) as max_sal FROM Salaries GROUP BY 1)
as t2
ON t1.company_id = t2.company_id;
```

Q77.

Solution 77:

```
select date(sale_date) as sale_date,
     sum(case when fruit = 'apples' then sold_num
           when fruit = 'oranges' then -sold_num end) as diff
```

```
from Sales
group by 1
order by 1
;
```

Q78.

Solution 78:

```
SELECT left_operand, operator, right_operand,
      CASE WHEN operator = '>' AND v1.value > v2.value THEN 'true'
          WHEN operator = '<' AND v1.value < v2.value THEN 'true'
          WHEN operator = '=' AND v1.value = v2.value THEN 'true'
          ELSE '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;
```

Q79.

Solution 79:

```
SELECT name FROM (
  SELECT u.name name, count(*) total
  FROM users u, movie_rating mr
  WHERE u.user_id=mr.user_id
  GROUP BY u.name
) ORDER BY total desc, name
LIMIT 1
UNION
SELECT title FROM (
 SELECT m.title title, AVG(mr.rating) average
 FROM movies m,  movie_rating mr
```

```
  WHERE m.movie_id=mr.movie_id
    AND created_at BETWEEN '2020-02' and '2020-03'
  GROUP BY title
) ORDER BY average desc, title
LIMIT  1;
```

Q80.

Solution 80:

```
SELECT
 co.name AS country
FROM
 person p
 JOIN
    country co
    ON SUBSTRING(phone_number,1,3) = country_code
 JOIN
    calls c
    ON p.id IN (c.caller_id, c.callee_id)
GROUP BY
 co.name
HAVING
 AVG(duration) > (SELECT AVG(duration) FROM calls)
```

Q81.

Solution 81:

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

Q82.

Solution 82:

```
SELECT name
FROM Employee
ORDER BY name;
```

Q83.

Solution 83:

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

Q84.

Solution 84:

```
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;
```

Q85.

Solution 85:

```
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;
```

Q86.

Solution 86:

```
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,
    TRUNC(500000/total_sqft,0) AS prime_item_combo,
    (TRUNC(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,
    TRUNC(
      (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;
```

Q87.

Solution 87:

```
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 - interval '1 month')
)
  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);
```

Q88.

Solution 88:

```
WITH searches_expanded AS (
  SELECT searches
  FROM search_frequency
  GROUP BY
    searches,
    GENERATE_SERIES(1, num_users))

SELECT
  ROUND(PERCENTILE_CONT(0.50) WITHIN GROUP (
    ORDER BY searches)::DECIMAL, 1) AS  median
FROM searches_expanded;
```

Q89.

Solution 89:

```
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;
```

Q91.

Solution 91:

```
 WITH payments AS (
  SELECT
    merchant_id,
    EXTRACT(EPOCH FROM transaction_timestamp -
      LAG(transaction_timestamp) OVER(
        PARTITION BY merchant_id, credit_card_id, amount
```

```
      ORDER BY transaction_timestamp)
   )/60 AS minute_difference
 FROM transactions)

SELECT COUNT(merchant_id) AS payment_count
FROM payments
WHERE minute_difference <= 10;
```

Q93.

Solution 93:

```
SELECT
   gender,
   day,
   SUM(score_points) OVER(PARTITION BY gender ORDER BY day) as total
FROM Scores;
```

Q94.

Solution 94:

```
SELECT
 co.name AS country
FROM
 person p
 JOIN
    country co
    ON SUBSTRING(phone_number,1,3) = country_code
 JOIN
    calls c
    ON p.id IN (c.caller_id, c.callee_id)
GROUP BY
 co.name
HAVING
```

```
AVG(duration) > (SELECT AVG(duration) FROM calls);
```

Q96.

Solution 96:

```
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;
```

Q97.

Solution 97:

```
  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;
```

Q98.

Solution 98:

```
    select group_id, player_id from (
            select p.group_id, ps.player_id, sum(ps.score) as score
            from Players p,
                (
           select first_player as player_id, first_score as score
           from Matches
           union all
           select second_player, second_score
           from Matches
                ) ps
            where p.player_id = ps.player_id
            group by ps.player_id
            order by group_id, score desc, player_id
            -- limit 1 -- by default, groupby will pick the first one i.e. max score player here
) top_scores
group by group_id;
```

Q99.

Solution 99:

```
WITH cte AS
(SELECT student_id,
           score,
           exam_id,
           (CASE WHEN score < MAX(score) OVER (PARTITION BY exam_id)
           AND score > MIN(score) OVER (PARTITION BY exam_id)
           THEN 'middle'
```

```
        ELSE 'highlow'
        END) AS category
FROM Exam
ORDER BY student_id),

cte1 AS (SELECT student_id
      FROM cte
      GROUP BY student_id
      HAVING SUM(CASE WHEN category = 'highlow'
            THEN 1 ELSE 0
            END) = 0
)

SELECT cte1.student_id, s.student_name
FROM cte1 JOIN Student s
ON cte1.student_id = s.student_id
ORDER BY cte1.student_id;
```

Q100.

Solution 100:

```
select s.*
from student s
inner join (
   select e.*,
      rank() over(partition by exam_id order by score) as rn_asc,
      rank() over(partition by exam_id order by score desc) as rn_desc
   from exam e
) e on e.student_id = s.student_id
group by s.student_id
having min(rn_asc) > 1 and min(rn_desc) > 1;
```

Q102.

Solution 102:

```
select distinct username, activity, startDate, endDate
from
   (select u.*,
        rank() over (partition by username order by startDate desc) as rnk,
        count(activity) over (partition by username) as num
   from UserActivity u) t
where (num <> 1 and rnk = 2) or (num = 1 and rnk = 1);
```

Q103.

Solution 103:

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

Q117.

Solution 117:

```
SELECT Name FROM STUDENTS WHERE Marks > 75 ORDER BY RIGHT(Name, 3), ID;
```

Q118.

Solution 118:

```
Select name from Employee
Order by name;
```

Q119.

Solution 119:

```
select name from Employee
where salary > 2000 and months < 10
order by name;
```